In: Machine Learning Journal, 3 (4), pp261-283, Netherlands: Kluwer (1989)
http://www.cs.utexas.edu/users/pclark/papers

The CN2 Induction Algorithm

Peter Clark (pete@turing.ac.uk)
Tim Niblett (tim@turing.ac.uk)
The Turing Institute,
36 N. Hanover St.,
Glasgow, G1 2AD, U.K.

October 1988

Abstract

Systems for inducing concept descriptions from examples are valuable tools for assisting in the task of knowledge acquisition for expert systems. This paper presents a description and empirical evaluation of a new induction system, CN2, designed for the efficient induction of simple, comprehensible production rules in domains where problems of poor description language and/or noise may be present. Implementations of the CN2, ID3 and AQ algorithms are compared on three medical classification tasks.
Keywords: concept learning, rule induction, noise, comprehensibility, cn2.

1 Introduction

In the task of constructing expert systems, systems for inducing concept descriptions from examples have proved useful in easing the bottleneck of knowledge acquisition [1]. Two families of systems, based on the ID3 [2] and AQ [3] algorithms, have been especially successful. These basic algorithms assume no noise in the domain, searching for a concept description that classifies training data perfectly. However for the application of systems based on these algorithms to real-world domains, methods for handling noisy data are required. In particular, mechanisms for avoiding the *overfitting* of the induced concept description to the data are needed, requiring relaxation of the constraint that the induced description must be classify the training data perfectly.

Fortunately the ID3 algorithm lends itself to easy modification allowing this constraint to be relaxed, by the nature of its general-to-specific search. Tree pruning techniques (e.g. [4, 5]), as used for example in the systems C4 [6] and ASSISTANT [7], have proved to be effective methods of avoiding overfitting. The AQ algorithm, however, is less easy to modify due to its dependence on specific training examples during its search. Existing implementations (e.g. AQ11 [8] and AQ15 [9]) deal with noisy data by using pre- and post-processing techniques while leaving the basic AQ algorithm intact. Our objective in designing CN2 is to modify the AQ algorithm itself in such a way that this dependence on specific examples is removed and the space of rules searched is increased. As a result statistical techniques analogous to those used for tree pruning can then be applied in the generation of if-then rules, and a simpler algorithm is achieved.

We can identify several requirements that learning systems should meet if they are to prove useful in a variety of real-world situations:

Accurate classification. The induced rules should be able to classify new examples accurately, even in the presence of noise.

Simple rules. For the sake of comprehensibility, the induced rules should be as short as possible. However, when noise is present, rules that are overfitted tend to be long. Thus, to induce short rules, one must usually relax the requirement that the induced rules be consistent with all the training data. The choice of how much to relax this requirement involves a trade-off between accuracy and simplicity [10].

Efficient rule generation. If one expects to use large example sets, it is important that the algorithm scales up to complex situations. In practice, it is desirable that the time taken for rule generation be linear in the size of the example set.

With these requirements in mind, this paper presents a description and empirical evaluation of CN2, a new induction algorithm. It combines the efficiency and ability to cope with noisy data of ID3 with the if-then rule form and flexible search strategy of the AQ family. The representation for rules output by CN2 is an ordered set of if-then rules, also known as a 'decision list' [11]. CN2 uses a heuristic function to terminate search during rule construction, based on an estimate of the noise present in the data. This results in rules that do not necessarily classify all the training examples correctly, but that perform well on new data.

In the following section we describe CN2 and three other systems used for our comparative study. These include: AQR, the authors' reconstruction of Michalski et al's AQ algorithm; Kononenko, Bratko and Roskar's (1984) ASSISTANT, a variant of ID3; and a simple Bayesian classifier which is used to provide a reference for the performance of the other algorithms. In each case we consider the time complexity of the various algorithms. In section 4, we compare the performance of the algorithms on three medical tasks; we also compare the performance of ASSISTANT and CN2 on two synthetic tasks. In section 5 we discuss the significance of these results, and we follow this with some suggestions for future work in section 6.

2 CN2 and Algorithms for Comparative Study

CN2 and the other algorithms used in experiments are now presented. Because CN2 has been developed from study of both the ID3 and AQ algorithms, we first present the ID3-based system ASSISTANT and the AQ-based system AQR before presenting CN2 and discussing its relationship to these algorithms.

We characterize the systems along three dimensions. These are:

- The representation language for the induced knowledge;
- The performance engine for executing the rules; and
- The learning algorithm and its associated search heuristics.

In all of our experiments, the example description language consisted of attributes, attribute values and user-specified classes. This language was the same for each algorithm.

2.1 Assistant

The ASSISTANT algorithm [7] is a descendant of Quinlan's ID3 (1983), and incorporates a tree pruning mechanism for handling noisy data.

```
Let: $E$ be a set of examples
     $A$ be a set of attributes for describing examples
     $TE(E)$ be a termination criterion
     $IDM(a_i, E)$ be an evaluation function where $a_i \in A$

Procedure ASSISTANT($E$) returning $TREE$:

    If: $E$ satisfies the termination criterion $TE(E)$ then return a leaf node
        for $TREE$, labelled with the most common class of examples in $E$.

    Else: determine the attribute $a_{best} \in A$ with the largest value of the
        function $IDM(a_{best}, E)$. Then, for each value $v_j$ of attribute $a_{best}$,
        generate subtrees using ASSISTANT($E_j$) where $E_j$ are those examples
        in $E$ with value $v_j$ for attribute $a_{best}$. Return a node labelled as a
        test on attribute $a_{best}$ with these subtrees attached.
```

Table 1: The core of the ASSISTANT algorithm

2.1.1 Concept Description Language and Interpretation

ASSISTANT represents acquired knowledge in the form of decision trees. An internal node of a tree specifies a test of an attribute, with each outgoing branch corresponding to a possible result of this test. Leaf nodes represent the classification to be assigned to an example.

To classify a new example, a path from the root of the decision tree to a leaf node is traced. At each internal node reached, the branch corresponding to the value of the attribute tested at that node is followed. The class at the leaf node represents the class prediction for that example.

2.1.2 Learning Algorithm

ASSISTANT induces a decision tree by repeatedly specializing leaf nodes of an initially single-noded tree. The specialization operation involves replacing a leaf node with an attribute test, and adding new leaves to that node corresponding to the possible results of that test. Heuristics determine which attribute to test on and when to stop specialization. Table 1 summarizes this algorithm.

2.1.3 Heuristic Functions

ASSISTANT uses an entropy measure to guide the growth of the decision tree, as described by Quinlan (1983) . This corresponds to the function *IDM* in Table 1. In addition, the algorithm can apply a tree cutoff method based on an estimate of maximal classification precision. This technique estimates whether additional branching would reduce classificational accuracy and if so, terminates search (there are no user-changeable parameters in this calculation). This cutoff criterion corresponds to the function *TE* in the Table 1. If ASSISTANT is to generate an 'unpruned' tree, the termination criterion *TE(E)* is satisfied if all the examples E have the same class value.

2.2 AQR

AQR is an induction system that uses the basic AQ algorithm [3] to generate a set of classification rules. Many systems use this algorithm in a more sophisticated manner than AQR to improve predictive accuracy and rule simplicity (e.g., AQ11 [8] uses a more complex method of rule interpretation that involves degrees of confirmation). AQR is a reconstruction of a straightforward AQ-based system.

2.2.1 Concept Description Language and Interpretation

AQR induces a set of decision rules, one for each class. Each rule is of the form 'if <cover> then predict <class>', where <cover> is a boolean combination of attribute tests as we now describe.

The basic test on an attribute is called a *selector*. The following are examples of selectors:

$$\langle \text{Cloudy} = \text{yes} \rangle$$
$$\langle \text{Weather} = \text{wet} \wedge \text{stormy} \rangle$$
$$\langle \text{Temp} > 60 \rangle$$

AQR allows tests in the set $\{=, \leq, >, \neq\}$. A conjunct of selectors is called a *complex*, and a disjunct of complexes is called a *cover*. We say that an expression (a selector, complex, or cover) *covers* an example if the expression is true of the example. Thus, the empty complex (conjunct of zero attribute tests) covers all examples and the empty cover (disjunct of zero complexes) covers no examples. A cover is stored along with an associated class value, representing the most common class of those training examples which it covers.

In AQR, a new example is classified by finding which of the induced rules have their conditions satisfied by the example. If the example satisfies only one rule, then the class predicted by that rule is assigned to the example. If the example satisfies more than one rule, the most common class of training examples that were covered by those rules is predicted. If the example is not covered by any rule, then it is assigned by default to the class that occurred most frequently in the training examples.

2.2.2 The Learning Algorithm

The AQ rule-generation algorithm has been described elsewhere (e.g. [8, 12, 13]), and the AQR system is an instance of this general algorithm. AQR generates a decision rule for each class in turn. Having chosen a class on which to focus, it forms a disjunct of complexes (the cover) to serve as the condition of the rule for that class. This process occurs in stages; each stage generates a single complex, and then removes the examples it covers from the training set. This step is repeated until enough complexes are found to cover all the examples of the chosen class. This whole process is repeated for each class in turn. Table 2 summarizes the AQR algorithm.

2.2.3 Heuristic Functions

The particular heuristic functions used by the AQ algorithm are implementation dependent. The heuristic used by AQR to choose the best complex is "maximize the number of positive examples covered". The heuristic used to trim the partial star during generation of a complex is "maximize the sum of positive examples covered and negative examples excluded". In the case of a tie for either heuristic, the system prefers complexes with fewer selectors. Seeds are chosen at random and negative examples are chosen according to their distance from the seed (nearest ones are picked first, where distance is the number of attributes with different values in the seed and negative example). In the case of contradictions (i.e. if the seed and negative example have identical attribute values) the negative example is ignored and a different one is chosen, since the complex cannot be specialized to exclude it but still include the seed.

2.3 A Bayesian Classifier

To establish a reference point, we also implemented a simple Bayesian classifier and compared its behavior to that of the other algorithms.

Table 2: The AQR covering algorithm: Generating a cover for class C

```
Procedure AQR(POS, NEG) returning COVER:

let COVER be the empty cover;
while COVER does not cover all positive examples in POS
    select a SEED, i.e. a positive example not covered by COVER;
    call procedure STAR(SEED, NEG) to generate the STAR (a set) of
        complexes that cover SEED but no examples in NEG;
    select the best complex BEST from the star according to user-defined criteria;
    add BEST as an extra disjunct to COVER;
return COVER.


Procedure STAR(SEED, NEG) returning STAR:

let STAR be the set containing the empty complex;
while one or more complexes in STAR covers some negative examples in NEG,
    select a negative example $E_{neg}$ covered by a complex in STAR;
    Specialize complexes in STAR to exclude $E_{neg}$ by:
        let EXTENSION be all selectors that cover SEED but not $E_{neg}$;
        let STAR be the set $\{x \wedge y | x \in STAR, y \in EXTENSION\}$;
        remove all complexes in STAR subsumed by other complexes in STAR;
    Remove the worst complexes from STAR
        until size of STAR is less than or equal to user-defined maximum (maxstar).
return STAR.
```

2.3.1 Concept Description Language and Interpretation

This classifier represents its 'decision rule' as a matrix of probabilities $p(v_j|C_k)$ specifying the probability of occurrence of each attribute value given each class. To classify a new example, one applies Bayes' theorem

$$p(C_i|\bigwedge v_j) = \frac{p(\bigwedge v_j|C_i)p(C_i)}{\sum_k p(\bigwedge v_j|C_k)p(C_k)}$$

where the summation is over the n classes and $p(C_i|\bigwedge v_j)$ denotes the probability that the example is of class C_i given v_j ($\bigwedge$ is the symbol for conjunction, $\bigwedge v_j$ denoting a conjunct of attribute values all occurring in an example). One calculates this probability for every class, and then selects the class with the highest probability. The term $p(C_k)$ is estimated from the distribution of the training examples among classes. If one assumes independence of attributes, $p(\bigwedge v_j|C_k)$ can be calculated using

$$p(\bigwedge v_j|C_k) = \prod_j p(v_j|C_k)$$

and the values $p(v_j|C_k)$ from the probability matrix. Note that, unlike the other algorithms we have discussed, our implementation of the Bayesian classifier requires one to examine the values of all attributes when making a prediction.

We should note that there also exist more sophisticated applications of the Bayes rule in which the attribute tests are ordered [14]. Such a sequential technique adds the contribution of each test to a total; when this score exceeds a threshold, the algorithm exits with a class prediction. Such an interpretation may be more comprehensible to a user than the approach we have used, as well as limiting the tests required for classification.

2.3.2 The Learning Algorithm

The Bayesian learning method constructs the matrix $p(v_j|C_k)$ from the training examples by examining the frequency of values in each class. One can compute this matrix either incrementally, incorporating one instance at a time, or non incrementally, using all data at the outset.

2.3.3 Heuristics

Sometimes a value of zero is calculated from the training data for some elements of the $p(v_j|C_k)$ matrix. Like all elements of the matrix, this number is subject to error due to the finite training data available. However as as classification of new examples involves multiplying elements together, a zero element can have drastic effect, nullifying the effect of all other probabilities in the multiplication. To avoid this, we assume that zero elements in the matrix would, given more data, converge on a small, non-zero value and hence replace the zeros with some appropriate estimate. In our implementation a value of $p(C_i) \times (1/N)$ was used, where N is the number of training examples. The factor $1/N$ represents the increasing certainty that this element must have an almost-zero value with increasing size of training data.

2.4 The Default Rule

Finally, an 'algorithm' which simply assigns the most commonly occurring class to all new examples, with no reference to their attributes at all, was used for comparison with the other algorithms. Interestingly, this simple procedure was of comparable performance to the other algorithms in one domain tested (described later), and thus proved a useful extra algorithm to consider.

2.5 The CN2 Algorithm

Having presented ASSISTANT, AQR, and the other algorithms used in the experiments, we now present CN2, first describing how the general algorithm arises naturally from consideration of the ID3 and AQ algorithms and then describing its details.

ID3 is easily adaptable to handle noisy data by virtue of its top-down tree generation approach. During induction, *all* possible attribute tests are considered when 'growing' a leaf node in the tree, and entropy is used to select the best one to place at that node. Overfitting of decision trees can thus be avoided by halting tree growth when no more significant information can be gained by further growth. We wish to apply a similar method to the production of if-then rules rather than decision trees.

The AQ algorithm, when generating a complex, also performs a general-to-specific search for the best complex. However, only specializations which exclude some particular covered negative example from the complex while ensuring some particular 'seed' positive example remains covered are considered, iterating until all negative examples are excluded. As a result, AQ searches only the space of complexes completely consistent with the training data. The AQ algorithm employs a beam search, which can be viewed as several hill-climbing searches in parallel.

For the CN2 algorithm, we have retained the beam search method of the AQ algorithm but firstly removed its dependence on specific examples during search and secondly extended its search space to include rules which do not perform perfectly on the training data. This is simply achieved by broadening the specialization process to examine all specializations of a complex, similar to the way ID3 considers all attribute tests when growing a node in the tree. Indeed, with a beam width of one the CN2 algorithm behaves equivalently to ID3 growing a single tree branch. Thus by performing a top-down search for complexes, it is now possible to apply a cutoff method similar to decision tree pruning to halt specialization when no further statistically significant specializations are possible.

Finally, we note that the rules CN2 produces are an ordered list of if-then rules, rather than an unordered set of if-then rules as produced by AQ-based systems. Both representations have their respective advantages and disadvantages for comprehensibility – order independent rules require some additional mechanism to be provided to resolve any rule conflicts which may occur (thus detracting from a strict logical interpretation of the rules), while ordered rules also sacrifice a degree in comprehensibility as the interpretation of a single rule is dependent on which other rules preceded it in the list. It is possible to make CN2 produce unordered if-then rules by appropriately changing the evaluation function[1].

2.5.1 Concept Description Language and Interpretation

Rules in the ordered list which CN2 induces are each of the form 'if <complex> then predict <class>', where <complex> has the same definition as for AQR, namely a conjunct of attribute tests. This ordered rule representation is a version of what Rivest (1987) has termed decision lists. The last rule in CN2's list is a 'default rule' which simply predicts the most commonly occurring class in the training data for all new examples.

To use the induced rules to classify new examples, CN2 applies an interpretation in which each rule is tried in order until one is found whose conditions are satisfied by the example being classified. The resulting class prediction of this rule is then assigned as the class of that example. Thus, the ordering of the rules is important. If no induced rules are satisfied, the final default rule assigns the most common class to the new example.

2.5.2 Learning Algorithm

Table 3 presents a summary of the CN2 algorithm. This works in an iterative fashion, each iteration searching for a complex covering a large number of examples of a single class C and few of other classes. The complex must be both predictive and reliable, as

[1]E.g. replace entropy with E=(+ve exs. covered minus -ve exs. covered), then generate rules for each class in turn.

determined by CN2's evaluation functions. Having found a good complex, those examples it covers are removed from the training set and the rule 'if <complex> then predict C' is added to the end of the rule list. This process iterates until no more satisfactory complexes can be found.

The system searches for complexes by carrying out a pruned general-to-specific search. At each stage in the search, CN2 retains a size-limited set or *star* S of 'best complexes found so far'. The system examines only specializations of this set, carrying out a beam search of the space of complexes. A complex is specialized by either adding a new conjunctive term or removing a disjunctive element in one of its selector. Each complex can be specialized in several ways, and CN2 generates and evaluates all such specializations. The star is trimmed after completion of this step by removing its lowest ranking elements as measured by an evaluation function that we will describe shortly.

Our implementation of the specialization step is to repeatedly *intersect*[2] the set of all possible selectors with the current star, eliminating all the null and unchanged elements in the resulting set of complexes. (A null complex is one that contains a pair of incompatible selectors, e.g. $\texttt{big} = \texttt{y} \wedge \texttt{big} = \texttt{n}$). CN2 deals with continuous attributes in a manner similar to ASSISTANT - by dividing the range of values of each attribute into discrete sub-ranges. Tests on such attributes examine whether a value is greater or less (or equal) than the values at sub-range boundaries. The complete range of values and size of each sub-range is provided by the user.

For dealing with unknown attribute values, CN2 use the simple method of replacing unknown values with the most commonly occurring value (or midvalue of the most commonly occurring sub-range, in the case of numeric attributes) for that attribute in the training data.

2.5.3 Heuristics

The CN2 algorithm must make two heuristic decisions during the learning process, and it employs two evaluation functions to aid in these decisions. First it must assess the quality of complexes, determining if a new complex should replace the 'best complex' found so far and also which complexes in the star S to discard if the maximum size is exceeded. Computing this involves first finding the set E' of examples which a complex *covers* (i.e., which satisfy all of its selectors) and the probability distribution $P = (p_1, \ldots p_n)$ of

[2]The intersection of set $\texttt{A}$ with set $\texttt{B}$ is the set $\{\texttt{x} \wedge \texttt{y} | \texttt{x} \in \texttt{A}, \texttt{y} \in \texttt{B}\}$. For example, using '.' to abbreviate '$\wedge$', $\{a.b, a.c, b.d\}$ intersected with $\{a, b, c, d\}$ is $\{a.b, a.b.c, a.b.d, a.c, a.c.d, b.d, b.c.d\}$. If we now remove unchanged elements in this set we obtain $\{a.b.c, a.b.d, a.c.d, b.c.d\}$.

Table 3: The CN2 Algorithm

```
Let: E be a set of training examples;

Procedure CN2(E) returning RULE_LIST:

let RULE_LIST be the empty list;
repeat
    let BEST_CPX be Find_Best_Complex(E);
    if BEST_CPX is not nil then
        Let E' be the examples covered by BEST_CPX;
        Remove from E the examples E' covered by BEST_CPX;
        Let C be the most common class of examples in E';
        Add the rule 'if BEST_CPX then class=C' to the end of RULE_LIST,
until BEST_CPX is nil or E is empty.
return RULE_LIST.

Procedure Find_Best_Complex(E) returning BEST_CPX:
let the set STAR contain only the empty complex;
let BEST_CPX be nil;
let SELECTORS be the set of all possible selectors;
while STAR is not empty,
    specialize all complexes in STAR as follows:
    let NEWSTAR be the set $\{x \wedge y | x \in STAR, y \in SELECTORS\}$;
    Remove all complexes in NEWSTAR that are either in STAR (i.e., the
        unspecialized ones) or are null (e.g. $big = y \wedge big = n$)
    for every complex $C_i$ in NEWSTAR:
        if $C_i$ is statistically significant when tested on E and better than
            BEST_CPX according to user-defined criteria when tested on E,
        then  replace the current value of BEST_CPX by $C_i$;
    repeat remove worst complexes from NEWSTAR
        until size of NEWSTAR is $\leq$ user-defined maximum;
    let STAR be NEWSTAR;
return BEST_CPX.
```

examples in E' among classes (where n = number of classes represented in the training data). CN2 then uses the information-theoretic entropy measure

$$Entropy = -\sum_i p_i \log_2(p_i)$$

to evaluate complex quality (the lower the entropy the better the complex). This function thus prefers complexes covering a large number of examples of a single class and few examples of other classes, and hence such complexes score well on the training data when used to predict the majority class covered. It should be noted that this function was used in preference to a simple 'percentage correct' measure (e.g. by taking max(P)), most importantly because entropy will distinguish probability distributions such as $P = (0.7, 0.1, 0.1, 0.1)$ and $P = (0.7, 0.3, 0, 0)$ in favor of the latter where as max(P) will not. This is desirable, since there exist more ways of specializing the latter to a complex identifying only one class; if the examples of the majority class are excluded by specialization, the distributions become $P = (0, 0.33, 0.33, 0.33)$ and $P = (0, 1, 0, 0)$ respectively. In addition, the entropy measure tends to direct the search in the direction of more significant rules; empirically, rules of high entropy also tend to have high significance.

The second evaluation function tests whether a complex is *significant*. By this we refer to a complex that locates a regularity unlikely to have occurred by chance, and thus reflects a genuine correlation between attribute values and classes. To assess significance, CN2 compares the *observed* distribution among classes of examples satisfying the complex with the *expected* distribution that would result if the complex selected examples randomly. Some differences in these distributions will result from random variation. The issue is whether the observed differences are too great to be accounted for purely by chance. If so, CN2 assumes that the complex reflects a genuine correlation between attributes and classes.

To test significance, the system uses the likelihood ratio statistic [15]. This is given by

$$2\sum_{i=1}^{n} f_i \log(f_i/e_i),$$

where the distribution $F = (f_1, \ldots, f_n)$ is the observed frequency distribution of examples among classes satisfying a given complex and $E = (e_1, ..., e_n)$ is the expected frequency distribution of the same number of examples under the assumption that the complex selects examples randomly. This is taken as the $N = \sum f_i$ covered examples distributed among classes with the same probability as that of examples in the entire training set.

This statistic provides an information-theoretic measure of the (non-commutative) distance between the two distributions[3]. Under suitable assumptions, one can show that this statistic is distributed approximately as χ^2 with $n-1$ degrees of freedom. This provides a measure of indicates significance — the lower the score, the more likely that the apparent regularity is due to chance.

Thus these two functions - entropy and significance - serve to determine whether complexes found during search are both 'good' (i.e. high accuracy when predicting the majority class covered) and 'reliable' (high accuracy on training data is not just due to chance) respectively. CN2 uses these functions to repeatedly search for the 'best' complex which also passes some minimum threshold of reliability until no more reliable complexes can be found.

3 Time Complexity of the Algorithms

The ASSISTANT, AQR and CN2 algorithms are all searching a very large space of concept descriptions, and all use heuristics to guide this search. Furthermore, the CN2, ASSISTANT and AQR algorithms attempt to produce structures that are both consistent with the training examples and as compact as possible. In the design of such algorithms, there is a tradeoff between execution speed and the size of the induced structures. In each case, the exhaustive search for a *smallest* set of structures, although desirable, is computationally infeasible.

A major application of these algorithms is to extract useful information from very large databases, perhaps with millions of examples. With this in mind, it is worth examining the complexity of each algorithm. To be practical for very large problems, their behavior should be linear, or near-linear, in the number of examples and attributes.

Since the overall complexity of each algorithm is domain-dependent, we instead provide upper bounds for the critical components of the algorithms. We do not consider the complexity of the cutoff procedure used by ASSISTANT.

In our treatment we use the following variables to denote parameters:

e: The size of the example set

a: The number of attributes

s: The maximum star size (for CN2 and AQR)

[3]We assume that F is continuous with respect to E, i.e. that the f_i are zero when the e_i are zero.

We also assume that each attribute is binary valued and that there are two classes[4].

3.1 Time Complexity of Assistant

The critical component in ASSISTANT is the process of selecting a test attribute on which to branch. Each such choice involves the following operations:

1. For each attribute, example counts are put in an array, indexed by class and attribute. This takes **Time:** $o(e \cdot a)$;
2. The entropy function is calculated for each attribute taking **Time:** $o(a)$;
3. Once the best attribute is found, the examples are divided into two sets; this takes time **Time:** $o(e)$[5].

Therefore, the overall time for a single attribute choice is $o(a \cdot e)$. The time taken to construct the complete tree depends very much on the structure of the tree. It seems reasonable to use the first figure only for comparative purposes, as argued above. Thus the amount of time taken by ASSISTANT for the basic attribute selection operation is a linear function of the number of examples, when the number of classes and attributes are held constant.

We should note that extensions to this algorithm that use real-valued attributes (such as ACLS [16]) must sort the examples by attribute value at the first stage. This increases the overall time bound to $o(a \cdot e \log e)$.

3.2 Time Complexity of CN2

The basic operation in CN2 is the specialization of the complexes in the current star. The number of single selector complexes without disjuncts is $2a$. The number of intermediate complexes generated is at most $a \cdot s$, and the time taken to evaluate an example against a complex is bounded by $o(a)$. Three steps are required for this specialization operation:

- Multiplying each complex in the star by the set of single selector rules; this takes **Time:** $o(a \cdot s)$;

[4] One might also consider the complexity as a function of the number of distinct attribute values and classes. We have not done this in our analysis.

[5] With appropriate data structures, it may be possible to much of this work in the first stage, but this does not affect the complexity class. Similarly, one can include any termination test that is linear in the number of examples.

- Evaluation of each complex, taking **Time:** $o(s{\cdot}e{\cdot}a)$;
- Sorting the complexes by value and then trim the star, which takes **Time:** $o(a{\cdot}s\log(a{\cdot}s))$.

Therefore, the overall time for a single specialization step is bounded by $o(a{\cdot}s(e + \log(a{\cdot}s)))$. As with ASSISTANT, the time required is a linear function of the number of examples. If we restrict the the size of the star to one, the time required has the same order as for ASSISTANT. In general, experience indicates that the time constants involved are somewhat less for ASSISTANT and other variants on ID3 than for CN2.

3.3 Time Complexity of AQR

In AQR, the basic operation is the specialization of complexes in a star. This operation is similar to that of CN2, except that only specializations causing a negative example to be uncovered by complexes in AQR's star are generated. We show the complexity of this operation is the same as that of CN2.

For each negative example, the following steps are performed:

- A negative example is found by iterating through the negative set. We assume that the number of negative examples is not less than some fixed fraction of the entire example set. This takes **Time** $o(e{\cdot}s)$;
- The set of selectors that distinguish the negative example from the seed are found; This takes **Time** $o(a)$;
- Each complex in the star is specialized by intersection with this set of selectors, taking **Time** $o(a{\cdot}s)$;
- The resulting complexes are evaluated, which takes **Time** $o(a{\cdot}s{\cdot}e)$;
- The complexes are sorted and the star trimmed, taking **Time** $o(a{\cdot}s\log(a{\cdot}s))$.

Thus, for each negative example the time is bounded by $o(a{\cdot}s(e + \log(a{\cdot}s)))$. This is the same figure as obtained for CN2. Observe that the number of iterations of this process (making the star disjoint from a negative example) is bounded by the number of attributes, not by the number of examples.

In practice, although the order of time taken by the algorithms for this particular operation of producing a new star is the same, CN2 is faster overall than AQR. This is because the number of iterations of this operation is lower in CN2 than in AQR, since CN2 may halt specialization of a complex before it performs perfectly on the training examples. Also, CN2 may halt the entire search for rules before all the training examples are covered if no further statistically significant rules can be found.

3.4 Time Complexity of the Bayesian Classifier

The time complexity of the Bayes' classifier for generating a probability matrix is $o(a{\cdot}e)$, where a is the number of attributes and e the number of examples. This learning algorithm was substantially faster than the other algorithms because the run time is independent of the decision 'rule' generated. In addition this basic operation is performed only once, unlike the above algorithms where the basic operation is repeatedly applied.

3.5 Summary and Actual Run Times

We have shown that the time complexity of the basic learning step for all the algorithms tested is linear in the number of examples ($o(a.e)$ for ASSISTANT, $o(a.e.s)$ for AQR and CN2). This is an essential requirement for any algorithm that must work with very large data sets.

We briefly consider the time complexity of the entire induction process, requiring iteration of the basic learning steps analysed above. With ideal noise-tolerant algorithms, given a certain minimum number of examples, concept descriptions representing only the genuine regularities in the data should be induced; additional examples should not cause the concept description to grow further and become overfitted, hence in this ideal case the above figures also represent the time complexity of the overall learning task. When this ideal is not met (e.g. when a concept description classifying the training data perfectly is sought for), CN2 would at worst induce e rules of length a giving an overall time complexity of $o(a^2.e^2.s)$ [17]. ASSISTANT sorts a total of e examples among a attributes for each level of the tree, giving an overall time complexity of $o(a^2.e)$ as the tree depth is bounded by a. A similar worst-case time complexity to CN2 holds for AQR.

We provide the actual run times for interest, while noting that it is difficult to make fair comparisons of speed due to differences in implementation language and method. All run times are for inducing a classification procedure in the lymphography domain (section 4.2.1) using a four-megabyte Sun 3/75. ASSISTANT, implemented in about 5000 lines of Pascal, took one minute run-time. CN2 and AQR, each implemented in about 400 lines of Prolog and with a value of fifteen for *maxstar*, took 15 and 170 minutes run-time respectively. The Bayesian classifier, implemented in 150 lines of Prolog, took a few seconds to calculate its probability matrix. While it is difficult to draw conclusions from the absolute run times, it is our opinion that the ordering of these run-times (Bayes fastest, followed by ASSISTANT, CN2 and AQR) is a fair reflection of the relative computation involved in using the algorithms. More detailed empirical comparisons of time and memory requirements of ID3 and the AQ-based system AQ11P have been conducted by O'Rorke (1982) and Jackson (1985) in the domain of chess end-games.

4 Experiments with the Algorithms

4.1 Dependent Measures

In addition to computational complexity, we evaluate the behavior of these systems by two other criteria, classificational accuracy and syntactic complexity of the acquired structure. This twofold evaluation is motivated by considering these systems as tools for knowledge acquisition for expert systems. A useful system should induce rules that are accurate - so that they perform well - as well as comprehensible - so that they can be validated by an expert and used for explanation.

We measure each algorithm's classification accuracy by splitting the data into a training set and a test set, presenting the algorithm with the training set to induce a concept description and then measuring the percentage of correct predictions made by that concept description on the test set. Quinlan (1983,1987) and others have taken a similar approach to measuring accuracy.

Cross-algorithm comparisons of the complexity of concept descriptions are difficult due to the differences in representation and the degree of subjectivity involved in judging complexity. Thus, we will only compare the gross features of the knowledge structures induced by the different algorithms. For ASSISTANT's decision trees, we measure complexity by the number of nodes (including leaves) in the tree. For CN2 and AQR, we measure complexity by the number of selectors in the final rule list and rule set respectively. These measures reveal the gross features of the induced decision rules. More detailed measures of rule complexity have been made by O'Rorke (1982) but are not used here. We assign a complexity of one to the default rule based on its equivalence to a decision tree with a single node.

Assessing the complexity of a Bayesian rule is more difficult. One could count the number of elements in the $p(V_j|C_k)$ matrix. Thus, for a domain with n classes and a attributes, each with an average of v possible values, the complexity would be $a \times v \times n$. However, such a measure is independent of the training examples, and it ignores features of the matrix that may make it more comprehensible (e.g., a few elements may be very large and the rest small). Still, lacking any better measure, we provide the size of the matrix as a rough guide.

4.2 Experiments on Natural Domains

The above algorithms were tested on three sets of medical data, which we will describe shortly. These data were obtained from the Institute of Oncology at the University Medical Center in Ljubljana, Yugoslavia [7]. In each test, 70% of the training examples were selected at random from the entire data set, and the remaining 30% of the data were used for testing. The algorithms were all run on the same training data and their induced knowledge structures tested using the same test data. Five such tests were performed for each of the three domains, and the results were averaged. These data are thus identical to those used to test AQ15 in [9], though the particular random 70% and 30% samples are different.

Both CN2 and AQR were given a value of 15 for *maxstar* in all runs.

4.2.1 Three Medical Domains

Table 4 summarizes the characteristics of the three medical domains used in the experiments. The first of these involved lymphography. For patients with suspected cancer, it is important for physicians to distinguish between diagnoses such as metastases, malignant lymphoma or simply normal findings; patient data relating to this task were collected from Ljubljana's Oncology Institute. These data were consistent, i.e. examples of any two classes were always different. All the tested algorithms produced fairly simple and accurate rules. Unlike the other two domains, this data set was not submitted to a detailed checking after its original compilation by the Medical Center, and thus may contain errors in attribute values.

The second domain involved predicting whether patients who have undergone breast cancer operations will experience recurrence of the illness within five years of the operation. The recurrence rate is about 30%, and hence such prognosis is important for determining post-operational treatment. These data were verified after collection, and thus are likely to be relatively free of errors.

The final medical domain focussed on predicting the location of primary tumor. Physicians distinguish between 22 possible locations, predicted from data such as age, hystologic type of carcinoma, and possible locations of detected metastases, and is again im-

Table 4: Description of the three medical domains

Domain Property	Domain		
	Lymphography	Breast Cancer	Primary Tumor
No. of attributes	18	9	17
Min-max no. of vals/att	2-8	2-5	2-3
Average values per attribute	3.3	2.8	2.2
Number of classes	4	2	22
Total no. of examples	148	286	339
Distribution of examples among classes	2, 81, 61, 4	85, 201	84, 20, 9, 14, 39, 1, 14, 6, 0, 2, 28, 16, 7, 24, 2, 1, 10, 29, 6, 2, 1, 24

portant in determining treatment of patients. These data were inconsistent, i.e. examples of different classes existed with identical attribute values. These data was verified after collecting, and thus are likely to be relatively error-free. The set of attributes is relatively incomplete, i.e. not sufficient to induce high quality rules.

4.2.2 Results with Natural Domains

Table 5 presents the results for each algorithm on each domain, averaged over five runs. In each case, we present the average accuracy on the test data and the average complexity of the resulting knowledge structures. CN2 was tested using three values of significance threshold and ASSISTANT was run with and without pruning. Only one version of the other systems were run.

The table contains some interesting regularities. Most important is that the algorithms designed to reduce problems caused by noisy data achieve a lower complexity without damaging their predictive accuracy. For example in the lymphography domain, the version of CN2 with the highest threshold achieved the same classification accuracy as the other algorithms by inducing (on average) only eight rules each containing 1.6 selectors. The tree pruning version of ASSISTANT produced similar results.

Both systems apply a similar technique to reducing complexity, namely sometimes halting specialization of concept descriptions before they classify the training examples perfectly. As a result, ASSISTANT and CN2 avoid overfitting their decision trees and rules

Table 5: Accuracy and complexity of knowledge structures acquired by the algorithms in three natural domains. (Complexity for the Bayes classifier is the size of the probability matrix).

Algorithm	Lymphography Domain		Breast Cancer Domain		Primary Tumor Domain	
	Accuracy	Complexity	Accuracy	Complexity	Accuracy	Complexity
Default rule	56%	1	71%	1	26%	1
ASSISTANT :						
(no pruning)	79%	41	62%	112	40%	178
(pruning)	78%	36	68%	44	42%	52
Bayes	83%	(240)†	65%	(540)†	39%	(465)†
AQR	76%	76	72%	208	35%	562
CN2 :						
(90% threshold)	78%	24	70%	28	37%	33
(95% threshold)	81%	22	70%	20	36%	42
(99% threshold)	82%	12	71%	4	36%	19

†See discussion in section 4.1 about difficulties in measuring the complexity of Bayesian classifier

Table 6: Accuracy of the different algorithms on training and test data. The reported version of ASSISTANT incorporated pruning and the version of CN2 used a 99% threshold.

Algorithm	Accuracy of decision procedures on training and test data:					
	Lymphography		Breast Cancer		Primary Tumor	
	Train	Test	Train	Test	Train	Test
Default rule	54%	56%	70%	71%	23%	26%
ASSISTANT	98%	78%	85%	68%	53%	42%
Bayes	89%	83%	70%	65%	48%	39%
AQR	100%	76%	100%	72%	75%	35%
CN2	91%	82%	72%	71%	37%	36%

to the training data. This contrasts with the AQR algorithm, which specializes its rule set until it achieves as nearly complete consistency with the training data as possible, resulting in an overfitted rule set. Table 6 illustrates this effect by comparing accuracy on the training and test data.

The results also show that the Bayesian classifier does well, performing comparably to the more sophisticated algorithms in all three domains and giving the highest accuracy in the lymphography domain. Table 6 shows that this method regularly overfits the training data, but that its performance in the test set is still good. Even more surprising is the behavior of the frequency-based default rule, which outperforms ASSISTANT and the Bayes method on the breast cancer domain. This suggests that in the breast cancer domain there are virtually no significant correlations between attributes and classes in the data. This is reflected by CN2's inability to find significant rules in this domain at 99% threshold, suggesting that, in this domain at least, the significance test has been effective in filtering out rules representing chance regularities.

In general, the differences in performance seem to be due less to the learning algorithms than to the nature of the domains; for example the best classification accuracy for lymphography was barely half as high as that for primary tumor. This suggests the need for additional studies to examine the role of domain regularity on learning.

4.3 Experiments on Artificial Domains

To better understand the effects of overfitting, we experimented with CN2 and ASSISTANT on two artificial domains that let us control the amount of noise in the data. Both domains contained twelve attributes and 200 examples which were evenly distributed between two classes. They differed only in the number of values each attribute could take (two in the first domain and eight in the second).

In both cases, the target concept for one class could be stated as a simple conjunctive rule of the form '**if** $(a = v_1) \wedge \cdots \wedge (d = v_1)$ **then** class X'. Both algorithms can represent such a regularity compactly. The second class was simply the negation of the first. 50% of the data was used for training, 50% for testing, and results averaged over five trials.

For each domain, we varied the amount of noise in the training data and measured the effect on complexity and accuracy on the test data. Table 7 reports the results for the first artificial domain with two values per attribute, and Table 8 for the second with eight values per attribute.

The percentage of noise added indicates the proportion of attribute and class values

Table 7: Results in artificial domain A1 (12 atts, 2 vals/att)

% of Noise Added	CN2 RULE (99% threshold)			ASSISTANT RULE			
		Non-default†		unpruned		pruned	
	Accuracy	Accuracy	Size	Accuracy	Size	Accuracy	Size
0%	95%	100%	3	99%	8	99%	8
2%	88%	99%	5	96%	16	98%	11
5%	88%	95%	10	91%	32	95%	16
10%	82%	95%	15	86%	45	91%	24
20%	73%	86%	20	76%	60	84%	27
40%	67%	76%	25	65%	74	76%	23
60%	56%	64%	26	62%	75	67%	23
100%	45%	49%	28	46%	85	43%	12

†this refers to the accuracy of those CN2 rules found by search, i.e. *excluding* the extra default rule ('everything is class X') at the end of the rule list. See discussion in Section 4.3.1.

in the training examples that have been randomized, where attributes and classes chosen for randomization have equal chance of taking any of the possible values for that attribute or class. Note that no noise was introduced to the test data.

4.3.1 Results with Artificial Domains

In experimenting with artificial domains, we are able to examine several features of the algorithms relating to their ability to handle noise. Firstly we are interested in the degradation of accuracy and simplicity of concept descriptions as noise levels are increased. Secondly, for CN2, it is also interesting to examine how the accuracy and simplicity of individual rules (as well as that of the rule set as a whole) is affected by noise in the data.

The results reveal some surprising features about both CN2 and ASSISTANT. Comparing classificational accuracy alone, ASSISTANT performed better than CN2 in these particular domains. However, comparing complexity of concept description, CN2 produced simpler concept descriptions than ASSISTANT except at high levels of noise in the first domain.

Ideally, as the level of noise approaches 100%, both algorithms should fail to find any significant regularities in the data and thus converge on a concept description of complexity one (for CN2's default rule alone or a single node decision tree). However for

Table 8: Results in artificial domain A2 (12 atts, 8 vals/att)

% of	CN2 RULE (99% threshold)			ASSISTANT RULE			
Noise		Non-default†		unpruned		pruned	
Added	Accuracy	Accuracy	Size	Accuracy	Size	Accuracy	Size
0%	93%	98%	8	99%	6	99%	6
2%	83%	99%	10	97%	12	97%	12
5%	86%	94%	13	96%	15	96%	15
10%	80%	98%	10	93%	22	93%	22
20%	73%	88%	15	85%	27	85%	27
40%	68%	82%	5	75%	33	75%	33
60%	63%	90%	4	66%	40	66%	40
100%	50%	58%	1	55%	43	55%	43

†this refers to the accuracy of those CN2 rules found by search, i.e. *excluding* the extra default rule ('everything is class X') at the end of the rule list. See discussion in Section 4.3.1.

CN2, this occurred only in the second of the two domains tested and did not occur in either domain for ASSISTANT. Indeed, in the second domain ASSISTANT's tree pruning mechanism did not prune the tree at all. CN2's finding of rules in the first domain, even at 100% noise level, can be understood as due to a combination of the large number of rules searched (e.g. there are $12 \times 11 \times 10 = 1320$ rules of length three in the space) and the high coverage of these rules (each length three rule covers on average $100/2^3 = 12$ examples). Enough rules are searched so that, even with 99% significance test, some chance coverage of the 12 (average) examples will appear significant. This did not occur in the second domain as the coverage of rules was considerably less; each length three rule covers on average $100/8^3 \approx 0.5$ examples, too few for the significance test to succeed.

These comparative results and behavior as noise level approaches 100% suggests that the thresholding methods used in both CN2 and ASSISTANT need to be more sensitive to the properties of the application domain. Research on improvements to CN2's significance test [17] and ASSISTANT's pruning mechanism [19] is currently being conducted.

Finally, we consider the accuracy of CN2's individual rules as well as that of the whole rule set. The figures presented in Tables 7 and 8 for 'non-default accuracy', referring to the accuracy of CN2's rules excluding cases where the default rule fires, indicate that the rule list consists of high accuracy rules plus a low accuracy (50% in this domain) default rule at the end. This is a desirable property of the rule list if it is to be used for helping an expert articulate his or her knowledge, as each individual rule found (apart from the default rule) represents a strong regularity in the training data. The decision tree equivalent would be to examine the individual branches generated, and their use in assisting an expert. Quinlan (1987) has recently conducted work in this area.

5 Discussion

The results on the natural domains indicate that different methods of halting the rule specialization process, besides having the effect of reducing rule complexity, do not greatly affect predictive accuracy. This effect has been reported in a number of papers ([7, 9, 21]) Indeed it perhaps may be the case that any technique will have this effect, providing a certain maximum level of pruning is not exceeded. If this is the case then an algorithm should be preferred if it most closely estimates this maximum level.

The results in Table 6 suggest that the 99% threshold for the CN2 algorithm is appropriate for the three natural domains; the accuracy on training data is close to that on test data indicating that, in these domains at least, the algorithm is not overfitting the data. Additionally, high accuracy is maintained, indicating that the concept description is not underfit either.

The results of the tests on the artificial domains, in particular the tests with 100% noise, indicate that the current measure of significance used by CN2 could be improved. As the noise level reaches 100%, the algorithm should ideally almost always find no rules. The fact that this only occurred in one of the two artificial domains suggests that the significance measure should be more sensitive to properties of the domain in question.

In many ways the comparisons with the AQR system are unfair, as the AQ algorithm was never intended to deal alone with noisy data. It was included in these experiments to examine the basic AQ algorithm's sensitivity to noise. In practice it is rarely used on its own, and instead enhanced by a number of pre- and post-rule-generation techniques. Experiments with the AQ15 system [9] show that with post-pruning of the rules and a probability-based or 'flexible matching' rule application method, one can achieve results similar to those of CN2 and ASSISTANT in terms of accuracy and complexity.

The principal advantage of CN2 over AQR is that the algorithm supports a cutoff mechanism — it does not restrict its search to only those rules that are consistent with the training data. CN2 demonstrates that one can successfully control the search through the larger space of inconsistent rules with the use of judiciously chosen search heuristics.

Secondly, by including a mechanism for handling noise in the algorithm itself, we have achieved a simple method for generating noise tolerant if-then rules, allowing easy reproducibility and analysis of the algorithm. In addition, we feel that the requirements of interactive induction, where the user interacts with system during and after rule generation, in particular the requirement for good explanation facilities, indicate that the logical rule interpretation used by CN2 will have practical advantages over the more complex probabilistic rule interpretation necessary for applying order-independent rules such as generated by AQR where rule conflicts may occur.

Another result of interest is the high performance of the Bayesian classifier. Although the independence assumption of the classifier may be unjustified in the domains tested, it did not perform significantly worse in terms of accuracy than other algorithms and it remains an open question as to how sensitive Bayesian methods are to violated independence assumptions. Although the probability matrices produced by the tested classifier are difficult to comprehend, the experiments suggest that variants of the Bayes classifier producing more comprehensible decision procedures would be worthy of further investigation.

6 Conclusion

In this paper we have demonstrated an induction algorithm that combines the best features of the ID3 and AQ algorithms, allowing the application of statistical methods similar to tree pruning in the generation of if-then rules. It is similar to ASSISTANT in its efficiency and ability to handle noisy data, whereas it partially shares the representation language and flexible search strategy of AQR. By including a mechanism for handling noise into the algorithm itself, a simple method for generating noise tolerant if-then rules has been achieved allowing easy reproducibility and analysis of the system.

The experiments we have conducted show that the algorithm has, in noisy domains, comparable performance to ASSISTANT. By inducing concept descriptions based on if-then rules, it provides a tool for assisting in the construction of knowledge-based systems where classification procedures based on rules rather than decision trees are desired. The most obvious improvement to the algorithm, suggested by the results on artificial domains, is an improvement to the significance measure used.

Acknowledgements

We thank Donald Michie and Claude Sammut for their careful reading and valuable comments on earlier drafts of this paper. We also thank Pat Langley and the reviewers for their detailed comments and suggestions about the presentation. We are grateful to G. Klanjšček, M. Soklič and M. Zwitter of the University Medical Center, Ljubljana for the use of the medical data and I. Kononenko for its conversion to a form suitable the induction algorithms. This work was supported by the Office of Naval Research under contract N00014-85-G-0243 as part of the Cognitive Science Research Program.

References

[1] Peter Mowforth. Some applications with inductive expert system shells. TIOP 86-002, Turing Institute, Glasgow, UK, 1986.

[2] J. Ross Quinlan. Learning efficient classification procedures and their application to chess endgames. In J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, editors, *Machine Learning, vol. 1.* Tioga, Palo Alto, Ca, 1983.

[3] R. S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the 5th international symposium on Information Processing (FCIP 69), Vol. A3 (Switching circuits), Bled, Yugoslavia*, pages 125–128, 1969.

[4] J. R. Quinlan. Simplifying decision trees. *Int. Journal of Man-Machine Studies*, 27(3):221–234, September 1987.

[5] Tim Niblett. Constructing decision trees in noisy domains. In I. Bratko and N. Lavrač, editors, *Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning)*, pages 67–78. Sigma, Wilmslow, UK, 1987.

[6] J. R. Quinlan, P. J. Compton, K. A. Horn, and L. Lazarus. Inductive knowledge acquisition: a case study. In *Applications of Expert Systems*, pages 157–173. Addison-Wesley, Wokingham, UK, 1987.

[7] Igor Kononenko, Ivan Bratko, and Egidija Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, Faculty of Electircal Engineering, E. Kardelj University, Ljubljana, 1984.

[8] R. S. Michalski and J. Larson. Incremental generation of vl_1 hypotheses: the underlying methodology and the description of program aq11. ISG 83-5, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, 1983.

[9] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system : an overview and experiments. In *Proceedings of IMAL 1986*, Orsay, France, 1986. Université de Paris-Sud.

[10] Wayne Iba, James Wogulis, and Pat Langley. Trading off simplicity and coverage in incremental concept learning. In John Laird, editor, *Proc. 5th Int. Conf. on Machine Learning*, pages 73–79. Kaufmann, Ca, 1988.

[11] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[12] R. S. Michalski and R. Chilausky. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean diagnosis. *Policy Analysis and Information Systems*, 4(2):125–160, 1980.

[13] P. O'Rorke. A comparative study of inductive learning systems AQ11P and ID3 using a chess end-game test problem. ISG 82-2, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, 1982.

[14] A. Wald. *Sequential Analysis*. Wiley, New York, 1947.

[15] J. Kalbfleish. *Probability and Statistical Inference*, volume 2. Springer-Verlag, NY, 1979.

[16] A. Paterson and T. Niblett. *ACLS Manual. Version 1.* Glasgow, 1982.

[17] Philip K. Chan. A critical review of cn2: A polythetic classifier system. Technical Report CS-88-09, Dept. of Computer Science, Vanderbild Univ., Tennessee, 1988.

[18] J.A. Jackson. Economics of automatic generation of rules from examples in a Chess end-game. UIUCDCS-F 85-932, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, 1985.

[19] Bojan Cestnik, Igor Kononenko, and Ivan Bratko. Assistant 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrač, editors, *Progress in Machine Learning (proceedings of the 2nd European Working Session on Learning)*, pages 31–45. Sigma, Wilmslow, UK, 1987.

[20] J. Ross Quinlan. Generating production rules from decision trees. In J. McDermott, editor, *IJCAI-87*, pages 304–307, Ca, 1987. Kaufmann.

[21] T. Niblett and I. Bratko. Learning decision rules in noisy domains. In M. A. Bramer, editor, *Research and Development in Expert Systems III*, pages 25–34. Cambridge University Press, 1987.