Sequential Cover Rule Induction with PA3

Pedro de Almeida\(^1, 2\) and Carlos Bento\(^1\)

\(^1\) CISUC – Centro de Informática e Sistemas da Universidade de Coimbra, Polo II, 3030 Coimbra, Portugal
nop00997@mail.telepac.pt, bento@dei.uc.pt
\(^2\) Universidade da Beira Interior, Rua Marquês de Ávila e Bolama, 6200 Covilhã, Portugal

Abstract. Algorithms for induction of concept descriptions from examples are important tools in the fields of machine learning and knowledge discovery in databases. This paper presents an induction algorithm, named PA3, that learns a set of ordered rules from examples. Functionally inspired in AQ and CN2, this algorithm attempts to generalize the learned rules during the induction process and introduces features that allow tuning of several important parameters. This paper describes PA3 and presents a comparison between this algorithm and CN2 on six data sets (three in the domain of stock market and three in an artificial domain).

1 Introduction

The development of PA3 was inspired by data mining work in stock market time series prediction – a high-noise problem. In this kind of domains, learned rule sets present clear advantages over other methods of knowledge representation, because they allow effective human analysis and selection of the represented (learned) knowledge, and also explicit addition of expert knowledge. However, this is a domain in which high levels of noise and very incomplete knowledge content in the learning data must be expected, a situation in which most existing rule-based learning methods tend to present a degraded performance. Those reasons convinced us to try to develop a rule induction algorithm that could prove better suited to our needs. In this paper we describe that algorithm, named PA3, and present the results of comparative tests with CN2 [2], a classic rule induction algorithm.

2 PA3 Description

Sequential cover algorithms learn rules directly and are based on the strategy of learning one rule, removing the examples it covers, and then repeating this process. Among sequential cover algorithms, some develop rules in a specific-to-general search, starting with very specific rules, usually covering only one example, and then generalizing the rules to (successfully) cover as many examples as possible. Other sequential cover algorithms search for rules starting with the most general rule and then narrowing its coverage to accurately fit some of the available learning examples. This strategy is followed by CN2 and FOIL [9] and is advantageous in high noise
contexts because it does not depend on a particular (perhaps atypical) example to seed rule development.

PA3 is a sequential cover algorithm but doesn’t completely fit into the general-to-specific or specific-to-general classification because (like AQ [7] and R-Mini [5]) it uses both techniques to generate each rule.

The PA3 algorithm is summarized in Table 1. The PA3 base procedure is a classic sequential cover rule learner. In this procedure, the rule learning cycle will only stop when every learning example is covered. Key features of this procedure include a rule expansion step after the basic rule induction and the elimination of only the correctly covered examples (after learning each rule).

In traditional sequential cover algorithms all the examples covered by a newly discovered rule are removed from the list of examples that will be used for the induction of the following rules. In PA3 we opted for the elimination of only the examples the rule covers accurately. This way, the examples wrongly covered by the rule remain in the set of examples and will influence (and improve) the discovery of the next rules. In fact, our empirical tests revealed a consistent improvement in the overall predictive accuracy of the resulting rule list when only the correctly covered examples were eliminated, although this choice also results in a slight increase in the algorithm run-time and in the number of resulting rules.

The last step in the base procedure is a rule selection step that will create the final rule list. Since the rules learned by this algorithm form an ordered list, this rule selection step has to retain a set of the first contiguous rules (also maintaining the order of those rules). Our strategy for rule selection is based on evaluating every discovered rule according to a very simple metric and then calculating the average of all the rule values. If this average value is above a user-defined threshold, the first rule in the list is transferred to the list of selected rules. Then the average of the rules remaining in the original list is calculated. If this average is still above the threshold, the first rule in that reduced list is also accepted and transferred to the new list of selected rules. This process continues until one average fails to be above the threshold, and the rest of the rules in the original list is rejected, or until every rule in the original list is transferred to the list of selected rules.

The rule evaluation metric used in this selection procedure is not the same used in the rule learning process. This metric evaluates a rule with a value of 1 or 0, according to the following criteria: If the rule only covers one example or covers an equal number of positive and negative examples in the set of learning examples filtered by the preceding rules, it is evaluated with 0. If the rule covers wrongly or more examples then it covers correctly in the complete (unfiltered) set of learning examples, it is also evaluated with 0. Otherwise the rule is evaluated with 1.

This method for rule selection works because the first rules in the original ordered list are the strongest and so the average value of the rule evaluations tends to go down as rules go on being selected and moved to the final rule list. However, it should be noted that the reduction of the average value is not monotonic, and the selection process stops the first time the average goes below the user-defined parameter. This parameter must be set between 0 (to accept all the discovered rules) and close to 1 (to accept only the first, stronger, rules).

Globally, this rule filtering method allows the user to choose the tradeoff level between a more complete case-space coverage and a reduced coverage using only the stronger rules (and therefore with greater accuracy). This seems to be a potentially useful feature in some noisy domains.
### Table 1. The PA3 algorithm

**Procedure** PA3: **returns** Rule_list

```
Learned_Rules := empty list
Examples_Set := set of learning examples
While Examples_Set is not empty
   New_Rule := Learn_Rule(Examples_Set)
   Expanded_Rule := Expand_Rule(Examples_Set, New_Rule)
   Add Expanded_Rule to the end of Learned_Rules
Remove from Examples_Set the examples Expanded_Rule correctly covers
Rule_list := Selection of Learned_Rules based on user set parameter.
```

**Procedure** Learn_Rule(Examples): **returns** New_Rule

```
Base_Rules := All possible rules with one pre-condition
Evaluate over Examples every rule in Base_Rules and make
   Star := Best rules in Base_Rules
While rules in Star can be made more specific
   Candidate_Rules := empty list
   For each Rule in Star
      Develop Rule adding to it each still allowable pre-condition
      Evaluate over Examples every development of Rule and make
         Candidate_Rules := Candidate_Rules + Best developed rules
   Evaluate over Examples every rule in (Candidate_Rules + Star) and make
      Star := Best rules in (Candidate_Rules + Star)
   Evaluate over Examples every rule in Star and make
      New_Rule := Best rule in Star.
```

**Procedure** Expand_Rule(Examples, Rule): **returns** Expanded_Rule

```
While the expansion procedure improves the rule
   For each pre-condition in Rule
      Value := pre-condition value
      If Value is different from minimum value for that attribute
         New_Rule := Rule with pre-condition value expanded downwards
         Evaluate over Examples Rule and New_Rule and make
            If New_Rule is better then Rule
               Rule := New_Rule
      If Value is different from maximum value for that attribute
         New_Rule := Rule with pre-condition value expanded upwards
         Evaluate over Examples Rule and New_Rule and make
            If New_Rule is better then Rule
               Rule := New_Rule
      Expanded_Rule := Rule.
```

For PA3’s general-to-specific part of rule development, we first considered adopting the simple beam search used in the propositional version of FOIL proposed by Cohen [3]. However, after some initial testing, we opted for a modified version of the similar but star-expanded\(^1\) method used in CN2, since, in most situations, the better quality of the rules resulting from the more complete hypothesis space coverage seems to justify the extra complexity of the algorithm. The most significant difference PA3 introduces in this search algorithm is the metric used for rule evaluation. In CN2, rules are evaluated by the entropy of the covered examples and every selected rule

---

\(^1\) Instead of developing only the best rule already found, every rule in a size-limited set of the best rules found so far is developed.
must pass a significance test. For PA3, however, we developed a new rule evaluation function to allow easier domain dependent parameterization and to include a rule simplicity measure. PA3 evaluates rules using a measurement function that combines explicit evaluations of the rule accuracy, coverage and simplicity. This function is

$$v = a^\beta \times c^{1-\beta} + \chi s,$$

where $v$ is the rule value, $a$ is the rule accuracy over the learning examples, $c$ is the rule coverage, $s$ is the rule simplicity and $\beta$ and $\chi$ are constants.

The rule accuracy is defined as $a = (n_c - n_w)/n$, and the rule coverage as $c = (n_c - n_w)/n$, $n_c$ being the number of examples the rule covers correctly, $n_w$ the number of examples the rule covers incorrectly, $n_t = n_c + n_w$ the total number of examples covered by the rule and $n$ the total number of available learning examples.

The rule simplicity, $s$, is defined as $s = (r_m - r_c)/r_m$, where $r_m$ is the maximum number of selectors that can be present in a rule (corresponding to the number of decision attributes in the examples) and $r_c$ is the actual number of selectors in the rule.

Notice that $a$, $c$ and $s$ will always be between 0 and 1 and $a$ is always greater or equal to $c$. In most situations, rules selected by this evaluation function will tend to have $a$ with values close to 1 and $c$ with considerably lower values.

The constants $\beta$ and $\chi$ must be chosen in the interval between 0 and 1. $\beta$ should normally be chosen with a value greater than 0.5, (0.6 to 0.8 are typical values) since the accuracy of the rule is usually the most important criterion for rule selection. Anyway, to reinforce the accuracy bias, rule coverage is defined as $(n_c - n_w)/n$ and not just as $n_t/n$ or even $n_c/n$. To justify this choice with an example, consider that, with $n_t/n$ or $n_c/n$, a rule that covers correctly 10 examples and covers wrongly 30 could be preferred (would be, in terms of rule coverage) over another that covers correctly 9 examples and none wrongly. The constant $\chi$ must have a small value. In normal conditions the rule simplicity criterion should not dominate the accuracy and coverage criteria. Although rule simplicity is important (to maintain the rules understandable and to avoid overfitting) it should be noticed that the coverage criterion already introduces a bias to prefer more general (usually simpler) rules. Also, PA3’s general-to-specific beam search only replaces a previously discovered rule by a more specific one (with more selectors) if the later results in a better evaluation value: If the evaluation value is the same, the first (simpler) rule is preferred. Thus, even with $\chi$ set to 0, there is already a clear bias to prefer simple rules. Anyway, we found empirical evidence that a low (non-zero) $\chi$ value (typical values could be 0.001 to 0.01), tends to improve the accuracy results. Notice that, with this kind of values for $\chi$, the simplicity criterion will only influence the choice of rules if they are evaluated almost equally in terms of accuracy and coverage. On the other hand, our tests suggest that increasing the $\chi$ value above a low critical threshold tends to quickly degrade the predictive performance of the learned rule list.

PA3 rule evaluation function allows the choice of parameters that efficiently substitute the marginal role of significance testing in Clark and Boswell’s [1] alternative rule measure for CN2. As an interesting result, unlike CN2, PA3 always continues to generate rules until all the available learning examples are covered. This seems desirable, since a more powerful rule selection (i.e. the filtering of weak rules from the list) can be made after the complete rule list is generated.
Rule “creation”, at the start of the Learn_Rule procedure, involves generating all the possible attribute-value pairs and associating each with a new rule. These initial rules, with only one selector, need to be completed with a postcondition, chosen to be equal to the value of the most common result (target attribute) in the examples covered by the rule. The rule development step (also in the Learn_Rule procedure), in which a new attribute-value pair is added to an already existing rule to create a new one, implies resetting the rule postcondition.

In PA3 each rule selected in the initial general-to-specific rule development step is immediately subjected to a rule generalization step that tries to expand the rule coverage. The rule expansion procedure uses a simple greedy search that tries to expand the admissible attribute values downwards and upwards. Notice that the PA3 expansion procedure was designed for value-ordered attributes and will not be very effective when the attributes have nominal, unordered, values. In this later case, the expansion procedure could be adapted to try all the expansion possibilities for the attribute values, and not just the “adjacent” values.

This generalization step is a key feature of PA3 and results in a rule list formed by less numerous, stronger rules, while also reducing potential overfitting problems and the overall run-time of the algorithm. It should be noted that, theoretically, the expanded rules that result from this process must be at least as good as the original unexpanded rules since they are evaluated by the same evaluation function and any expansion is adopted only if its evaluation proves better.

2.1 Data and Rule Format

Like most rule induction algorithms, PA3 works over completely discretized data. For PA3, we opted to isolate the transformation of continuous attributes into discrete values from the rule induction algorithm itself. When needed, this transformation was performed in a previous data preparation step, using Hong’s discretization method [6].

This way, PA3 only has to deal with fully discretized learning or testing examples. The examples include an arbitrary (constant) number of discrete decision attributes and a single, binary, result attribute (or class value). If the domain implies multiple class values, prediction with PA3 needs to be iterated one class at the time, as in Cohen’s IREP version [3]. We also use the method of Cohen’s IREP version to deal with unknown decision attribute values: All tests involving the attribute A are defined to fail on instances (examples) for which the value of A is missing.

PA3 induces an ordered list of “if...then...” rules. As in CN2 or IREP [4], each rule has the form “if <complex> then predict <class>”, where <complex> is a conjunct of attribute tests, the “selectors”.

In PA3, each selector implies testing an attribute to see if its value is included in a specified range of values. So, each selector indicates the attribute to be tested and the (inclusive) upper and lower limits of the range of values it has to be tested against.

The postcondition of a PA3 rule is a single Boolean value that specifies the class that rule predicts for the cases that comply with all the selectors.

It should be noted that, while a single PA3 rule includes a simple conjunction of tests, the final rule set is equivalent to a DNF formula. DNF (or CNF) are the most common formats for rule knowledge representation and are powerful enough to represent most real-life concepts. However, parity based concepts are difficult to represent in standard DNF [6]. This difficulty was obvious in one of the problems we
used to test PA3, the second of the “Monk’s Problems” [10].

2.2 Time Complexity of PA3

To study the PA3 complexity, we assume only two classes of examples and will use the following variables:

- $e$: Number of examples in the learning set.
- $a$: Number of attributes in each example.
- $v$: Maximum number of attribute values.
- $s$: Star size.
- $r$: Number of rules in the unfiltered learned rule set.

The dominant operation in PA3 is the procedure “Learn_Rule”, which has to be called $r$ times. This procedure’s main cycle is the rule specialization, which as to be run, at most, $a$ times and is bounded by $o(a^s v(e+\log(a s v))).$

This way, the time complexity of PA3 is bounded by $o(r a^s v(e+\log(a s v)))$, exactly the same order of time complexity of CN2. In practice, however, PA3 is considerably faster than CN2 (mostly because it produces a smaller set of rules).

3 Comparative Tests

Since PA3’s basic architecture was inspired by CN2, this algorithm is an obvious benchmark to evaluate the performance of PA3. As an added bonus, the basic functional similarity between our algorithm and CN2 allowed the design of a simple set of tests to evaluate separately some of the PA3’s key features. Those tests compare PA3 with three alternative algorithms: CN2, CN2mod, and PA3mod.

The CN2 version used in our tests was programmed to faithfully represent the original version, described in [2]. CN2mod is the CN2 algorithm modified with the Laplace expected error estimate [8] proposed by Clark and Boswell [1] as an alternative to entropy in the evaluation of rules. PA3mod is the PA3 algorithm with the rule evaluation metric replaced by the Laplace expected error estimate used in CN2mod. PA3 is the algorithm described in this paper.

In order to compare the four algorithms in equal circumstances, we chose to guarantee prediction over every possible test case. So, in our tests, there was no final rule selection in PA3 and PA3mod, and no significance test in CN2mod. In addition, a default rule (a rule with no preconditions that predicts the most frequent class in the learning examples) was added to all the generated rule lists. The star size is the same (10) for all the algorithms. Since, in some domains, the default rule by itself is able to produce reasonably accurate predictions and, in any case, it is the minimum standard any concept learning algorithm must beat, we also compare the results of the four described algorithms with the results of the default rule.

We use six data sets in our tests. Three of them correspond to the artificial test problems used in the competition performed at the 2nd European Summer School on Machine Learning (“The Monk’s Problems” [10]). These data sets were chosen because they were developed by a relevant group of researchers precisely to allow an efficient comparative test of machine learning algorithms, and because the results over them of several classic algorithms are readily available. These problems are defined over a domain in which each example has six discrete-valued attributes. Each
problem involves learning a binary function defined over this domain, from a sample of learning examples of that function. Following the methodology used in the original competition, all the domain examples (including those sampled to be learning examples) are used to test the accuracy of the learned rule sets, and a single run is used for each algorithm over each data set.

The other three data sets were created by preprocessing daily quotes (from 3 Nov 1997 to 29 Oct 1999) of the three most actively traded stocks listed in the Portuguese BVL exchange (BCP, EDP and PT). Each stock’s original data were transformed into 478 time-sequenced examples with ten discrete decision attributes (each with five possible values), and a binary result attribute. The decision attributes in each example correspond to a summary of the information contained in the original data up to (and including) a specific day. The binary result attribute represents the behavior of the stock in the following day (rise or fall). A bootstrap test was carried for the algorithms over each of the three stock market data sets. Each of those tests included 20 runs using 100 randomly extracted examples as a test set and using the remaining 378 examples as the learning set. The results of the 20 runs were used to produce averaged accuracy, duration and complexity results.

The predictive accuracy of the learned rule sets, measured over the test examples, is shown in Table 2. Table 3 shows in each entry the number of rules and the total number of selectors (summed over all the rules) in each final rule set. The selectors are attribute-value pairs for CN2 and CN2mod and attribute-range (of values) pairs in PA3mod and PA3. Table 4 shows the time the algorithms took to learn each rule set. All the algorithms were run in a Pentium II (300Mz) computer with 64 MB of memory. To achieve a fair time comparison, all the base procedures shared by the algorithms (e.g. rule ordering procedures, etc.) were coded the same way.

The function involved in the “Monk’s 1” problem is easily expressed in DNF, and the learning set of examples is noise-free. All the rule induction algorithms we tested were able to learn an exact match for this function.

Table 2. Predictive accuracy of the learned rule sets (%)

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>CN2</th>
<th>CN2mod</th>
<th>PA3mod</th>
<th>PA3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monk’s 1</td>
<td>50.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Monk’s 2</td>
<td>67.13</td>
<td>73.84</td>
<td>76.16</td>
<td>75.93</td>
<td>75.93</td>
</tr>
<tr>
<td>Monk’s 3</td>
<td>47.22</td>
<td>92.36</td>
<td>91.20</td>
<td>91.20</td>
<td>96.30</td>
</tr>
<tr>
<td>BCP</td>
<td>52.51</td>
<td>53.25</td>
<td>54.45</td>
<td>55.90</td>
<td>60.85</td>
</tr>
<tr>
<td>EDP</td>
<td>53.14</td>
<td>51.55</td>
<td>52.40</td>
<td>53.30</td>
<td>54.40</td>
</tr>
<tr>
<td>PT</td>
<td>51.67</td>
<td>57.85</td>
<td>60.10</td>
<td>62.95</td>
<td>63.10</td>
</tr>
</tbody>
</table>

Table 3. Number of rules and total number of selectors in each rule set

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>CN2</th>
<th>CN2mod</th>
<th>PA3mod</th>
<th>PA3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monk’s 1</td>
<td>1 0</td>
<td>10 12</td>
<td>10 15</td>
<td>8 11</td>
<td>8  9</td>
</tr>
<tr>
<td>Monk’s 2</td>
<td>1 0</td>
<td>55 145</td>
<td>43 105</td>
<td>38 88</td>
<td>38 82</td>
</tr>
<tr>
<td>Monk’s 3</td>
<td>1 0</td>
<td>24 39</td>
<td>20 38</td>
<td>16 27</td>
<td>7  5</td>
</tr>
<tr>
<td>BCP</td>
<td>1 0</td>
<td>132 256</td>
<td>87 188</td>
<td>77 165</td>
<td>47 88</td>
</tr>
<tr>
<td>EDP</td>
<td>1 0</td>
<td>142 280</td>
<td>94 201</td>
<td>81 179</td>
<td>58 120</td>
</tr>
<tr>
<td>PT</td>
<td>1 0</td>
<td>131 254</td>
<td>83 178</td>
<td>76 160</td>
<td>43 80</td>
</tr>
</tbody>
</table>
The “Monk’s 2” problem is also noise-free, but the underlying function is very difficult to learn by algorithms based on DNF-equivalent knowledge representation. The difficulty of learning and representing this function in DNF is clear from the high number of rules and selectors all the algorithms developed and from the relatively low prediction accuracy they achieved in this relatively simple and well-structured artificial problem.

The “Monk’s 3” problem is again easy to express as a DNF function, but random noise was inserted in the (relatively small) learning set of examples. PA3 proved to be more efficient in dealing with the noise involved in this problem. The other three algorithms produced similar, worse, results.

The three stock market problems use real-life data. However, in this domain, the use of historical data, alone, is known to be insufficient to achieve a very high prediction capability.

In the “BCP” problem all the four algorithms tested performed better that the default rule, with PA3 clearly achieving the best result.

In the “EDP” problem the algorithms seemed to fail to find a clear prediction pattern and only PA3 and PA3mod were able to achieve (marginally) better results than the default rule.

In the “PT” problem, all the algorithms obtained reasonably good results. The consistency and closeness of the results of the four algorithms seems to indicate that, over the tested time frame, the (near) future behavior of that stock was indeed partially induced from the learning data.

In the Monk’s problems, the default rule accuracy mainly shows if the test set of examples is biased towards one of the classes, and if that bias is in the same sense as in the learning set. In the three stock market problems the default rule accuracy indicated is the proportion of the most common class in each full set of examples.

CN2 is the main comparison base for PA3. That way, the results it achieves in our six tests are difficult to comment by themselves, although they can be considered good in the “Monk’s 1” problem (where 100 % prediction success was achieved) and weak in the “EDP” problem (where the default rule achieved a slightly better result). However, it must be stressed that in the three Monk’s problems our implementation of CN2 achieved clearly better results than those obtained by Kreuzinger, Hamann and Wenzel [10] in the original Monk’s competition. The accuracy results achieved by CN2 in the Monk’s competition were respectively 100%, 69.0% and 89.1% for the “Monk’s 1”, “Monk’s 2” and “Monk’s 3” problems. Our version of CN2 achieved

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>CN2</th>
<th>CN2mod</th>
<th>PA3mod</th>
<th>PA3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monk’s 1</td>
<td>00:00</td>
<td>00:46</td>
<td>00:44</td>
<td>00:35</td>
<td>00:36</td>
</tr>
<tr>
<td>Monk’s 2</td>
<td>00:00</td>
<td>05:10</td>
<td>03:22</td>
<td>03:01</td>
<td>03:03</td>
</tr>
<tr>
<td>Monk’s 3</td>
<td>00:00</td>
<td>01:44</td>
<td>01:19</td>
<td>01:00</td>
<td>00:28</td>
</tr>
<tr>
<td>BCP</td>
<td>00:00</td>
<td>20:47</td>
<td>11:52</td>
<td>10:23</td>
<td>05:18</td>
</tr>
<tr>
<td>EDP</td>
<td>00:00</td>
<td>22:16</td>
<td>12:39</td>
<td>11:34</td>
<td>07:19</td>
</tr>
<tr>
<td>PT</td>
<td>00:00</td>
<td>21:04</td>
<td>11:14</td>
<td>09:55</td>
<td>04:53</td>
</tr>
</tbody>
</table>
100%, 73.8% and 92.2%. The rule sets obtained were of very similar complexity: 10, 58 and 24 rules in the original competition (we obtained 10, 55 and 24) and 13, 145 and 38 selectors in the original competition (we obtained 12, 145 and 39). The better results of our version of CN2 are probably due to different choices of star size and of significance threshold.

In the six problems used in our tests, CN2mod obtained slightly better accuracy results then CN2, but the advantage did not prove significant. (The comparison between CN2 and CN2mod, and the theoretical justification for the change in the rule evaluation technique are presented in [1].)

PA3mod global accuracy results also did improve over CN2mod, but the improvement also does not prove significant. It should be noted that the main difference between PA3mod and CN2mod is the use of the rule generalization step. The results obtained seem to suggest that the reduced effectiveness of the rule generalization method used in PA3 and PA3mod over the unordered values of the arguments of the three “Monk’s Problems” is the main reason for the small global accuracy improvement of PA3mod over CN2mod.

PA3 presented better global accuracy results than the other algorithms and achieved better results than the base CN2 in all the data sets except in the “Monk’s 1”, where both attained 100% accuracy. It should be noted that PA3 only differs from PA3mod by using the new rule evaluation function described in this paper, instead of the Laplace rule measure. The fact that PA3 was able to achieve the same result as PA3mod in the first two problems and a better result in the last four (precisely the problems that include noise) seems a strong indication that the PA3 rule evaluation function is efficient in the presence of noise. Considering only the limited set of tests performed on the three stock market problems, a paired, one-sided, t-test indicates a 95% significance level for the mean 5.23% improvement of PA3 over CN2 and 90% significance for the mean 3.80% improvement of PA3 over CN2mod.

For the parameters of the PA3 rule evaluation function, we opted to use “standard” values that were set from the start (before the algorithm was run over the learning sets of examples), in order to prevent a “feedback” adjustment that could affect the validity of the results.

The values used in the three Monk’s problems were 0.8 for $\beta$ and 0.01 for $\chi$.

In the stock market problems we used 0.6 for $\beta$ and 0.01 for $\chi$. This lower $\beta$ value was chosen because the low information and high noise in the learning data are better dealt by more general rules then by rules that fit the learning examples more exactly.

In terms of rule list complexity (both in number of rules and in total number of selectors), our tests show a clear improvement of CN2mod over CN2, and also of PA3mod over CN2mod and of PA3 over PA3mod. When comparing directly PA3 with CN2, it is clear that the greatest improvements are associated with the four problems that include noise (“Monk’s 3”, “BCP”, “EDP” and “PT”). The improvements are smaller, but are still significant, in the two noise-free problems.

Notice that in the “Monk’s 3” problem the rule set that PA3 learns has 7 rules and only 5 selectors. This is possible because PA3 is allowed to learn a rule with no selectors (covering all the remaining examples) if that rule evaluates better than any other rule the algorithm discovers. In the ”Monk’s 3” problem, PA3 first learned 5 rules with only one selector and then terminated the rule set creation process with a final rule with no selectors. To maintain an identical situation for all the algorithms, the default rule was still added to the end of the learned rule set.
The time to learn the complete rule sets also improves significantly from CN2 to CN2mod, from CN2mod to PA3mod, and from PA3mod to PA3. That improvement can be related to the size of the developed rule sets, since the time to learn the complete rule sets is directly proportional to the number of learned rules.

Comparing PA3 with PA3mod, it can be seen that the time taken to learn the rule sets is improved only in the problems that include noise. In fact, in the first two (noiseless) problems, PA3 and PA3mod behave similarly in terms of accuracy, number of rules, and total time needed. On the other hand, in the three problems that include noise, PA3 improves significantly over PA3mod in all those points.

The final rule list filtering stage of PA3 has not been employed in the previous comparative tests. To test this feature, we ran again PA3 over the same six problems, but this time varying the value of the rule filtering parameter. For the stock market problems, the original (random) splits of the examples in test and learning sets were reused, in similar 20-run bootstrap tests.

Figure 1 shows the accuracy of the filtered rule sets developed by PA3 for values of the rule filtering parameter ranging between 0.0 (no filtering) and 1.0. No default rule is added to any of the rule sets. The first rule of each set is never filtered out.

![Figure 1](image)

**Fig. 1.** Accuracy of the filtered rule sets developed by PA3 as a function of the rule filtering parameter value. Accuracy (in %) appears in the vertical axis, the parameter value appears in the horizontal axis.

Figure 1 shows that there is a global improvement of accuracy with increased filtering, but clear improvements only occur with strong filtering of the rule set: There is only a slight improvement for less severe pruning. That behavior seems to suggest that the first few rules developed are clearly stronger then the rest and that after those first rules the rule quality tends to stabilize at a lower level. Since, by design, at least one rule is retained in the final rule list, no additional filtering takes place after that situation is reached, even if the filtering parameter is still increased.

Table 5 presents more detailed results of PA3 with the rule filtering parameter set to 0.6. The table entries show the accuracy achieved over each test set, the number of rules and selectors in each final rule set, the time needed to develop the rule sets, and the degree of coverage achieved by the final rule lists over the test sets of examples.
Comparing Table 5 with the PA3 results in Table 2, it can be seen that the accuracy of the resulting rule sets over the test examples was always improved by the rule filtering procedure (except in the problem that already presented 100% accuracy).

Rule filtering after the development of a complete rule list is certain to reduce significantly the complexity of the final rule sets. PA3 with the rule filtering parameter set to 0.6 did indeed strongly reduce the number of rules and the total number of selectors in the final rule lists. The simplification of the final rule lists is in itself a considerable advantage of the introduction of the rule filtering step.

The time PA3 needed to produce the final filtered rule sets was similar to the time needed earlier to produce the unfiltered rule sets. This happens because the specific implementation we used calculates the final rule evaluation needed for rule filtering even when the value of the rule filtering parameter is zero (as used for unfiltered rule sets). This way, the time PA3 needs for the rule filtering step is already almost completely included in the values indicated in Table 4. Anyway, this step is computationally very fast, and does not affect the global time complexity of the algorithm.

The coverage of the final rule sets, measured over the test examples, was obviously reduced by rule filtering. However, as can be judged by the values shown in Table 5, even a relatively strong pruning of the rule sets allows the final rule lists to cover a considerable part of the test examples. This happens because the first rules in the original rule list (those that are retained) tend to cover a proportionally very large percentage of the cases. This is clear in the “Monk’s 3” problem, where the only two rules that were not rejected still cover 75% of the test examples.

It should be noted that the default rule can be added even to actively filtered rule lists, to achieve full domain prediction coverage. This is a domain dependent decision that needs careful evaluation, since filtering the rule list and then adding the default rule usually tends to reduce the accuracy in relation to an unfiltered (complete) rule list with the same default rule added.

5 Conclusions

This paper presented PA3, a new rule induction algorithm basically inspired in CN2 and AQ. The key features of this algorithm include a rule generalization step after the basic general-to-specific induction of each rule, a rule evaluation function that measures explicitly the rule accuracy, coverage and simplicity, and a final user controllable rule selection step.
The tests performed over three artificial and three real-life data sets show that PA3 without final rule filtering improves the global accuracy, rule list simplicity and run time of the classical CN2 algorithm. These improvements are more noteworthy over noisy data sets. The tests performed also show that the final rule filtering step of PA3 is an effective way of allowing a choice between better prediction accuracy and greater coverage.

Present work in the development of PA3 is concentrated on automatic selection of the rule measurement function parameters, based on an analysis of the learning data.

References