Learning classification rules from an ion chromatography database using a genetic based classifier system

A.H.C. van Kampen\textsuperscript{a}, Z. Ramadan\textsuperscript{b}, M. Mulholland\textsuperscript{b}, D.B. Hibbert\textsuperscript{b}, L.M.C. Buydens\textsuperscript{a,*}

\textsuperscript{a} Laboratory for Analytical Chemistry, Catholic University, Nijmegen, NL-6525 ED Nijmegen, Netherlands
\textsuperscript{b} Department of Analytical Chemistry, University of New South Wales, Sydney NSW 2052, Australia

Received 25 March 1996; accepted 19 September 1996

Abstract

A classifier system based on genetic algorithm methodology was developed for the automatic extraction of production rules from a database of about 6000 ion chromatography (IC) method examples. This machine learning strategy generated heuristics that can assist in the choice for a detection method for a specified set of IC method and solute properties. It was shown that the final set of rules proposed detectors that agreed with the database for 76% of the cases. Application to a separate test set showed a prediction ability of 82%. The database, because of the characteristics of the included cases, did not allow for a significant improvement of these results. However, the results are of significance for the further development of knowledge systems, which assist in the design of IC methods. Furthermore, this dataset comprised a considerable challenge to the applied machine learning method.

Keywords: Ion chromatography; Genetic algorithm; Classification

1. Introduction

Development and optimisation of ion chromatography (IC) methods, is an ongoing field of research, and often requires the experience of an expert in the field [1]. Since an expert is not always readily available, it would be profitable if part of this process were automated by means of an expert system [2,3] to provide the appropriate guidance in the development of an IC method [4,5]. A recognised bottleneck in the design of these systems is the knowledge acquisition process [6–8]. The first requirement is the availability of an expert who is willing to assist in the development of the expert system. Several projects were terminated prematurely because no expert was available or left during the project. Even when an expert is available, knowledge acquisition is still a troublesome process because many pitfalls are involved, e.g., the explicit formulation of the heuristics used by the expert.

This research presents a different approach to the knowledge acquisition process, in that an attempt was made to extract the required information from a large database containing IC method descriptions that implicitly contained heuristics about the design of IC methods. Towards this end, the application of machine learning techniques [9,10] seemed most appropriate.
Machine learning is a branch of artificial intelligence (AI) concerned with developing computation theories of learning and construction of learning systems. Many computer algorithms are deductive, i.e., they draw their conclusions (e.g., results of calculation) from knowledge (e.g., numerical algorithm) incorporated in them but cannot acquire or generate new knowledge. The knowledge of machine learning systems, however, improves gradually with experience and they can learn domain knowledge by experimentation. They have the ability to draw inductive inferences from information (examples) given to them. The machine learning algorithm examines these examples and is able to build up a knowledge representation such that future problems of the same type can be solved.

Chemistry was one of the first disciplines, aside from computer science, to actively engage in research on AI techniques [11,12]. The first chemistry AI project was the DENDRAL project, which began in 1964 and aimed at the automatic interpretation of mass spectral data [13]. Other examples of the applications of AI in chemistry involved chemical reaction synthesis [14,15], improving chemical instrumentation for infra red spectroscopy [16] and liquid chromatography [17,18], and the design of experiments [19].

The most commonly used techniques machine learning techniques comprise induction [20,21], neural networks [22,23], and genetic algorithms [24–26,10]. Neural networks and genetic algorithms have both been successfully applied to various classification problems in the field of chemistry [27–30]. The only previous application of induction, to a chemical domain, comprised of a commercial implementation of the ID3 algorithm, which was developed prior to 1978 [31–35].

Recently, the neural network and the induction method were applied to an IC database in order to transform the implicit heuristics into a representation that can be used in an expert system [36–38]. Although both methods differ with respect to the applied learning method, and the representation of the knowledge, both methods turned out to give satisfactory results.

This paper describes the application of a third alternative machine learning method to this database, i.e., the genetic based classification system [25,10]. These classifier systems are based on the genetic algorithm (GA) methodology, which comprise a family of evolutionary computer algorithms based on natural selection and the survival of the fittest. Basically, these systems maintain a set of candidate production rules, which represent information extracted from the environment (database). These rules are initialised at random and gradually improve (incremental learning) during an evolutionary process in which they are applied to a set of examples in order to establish their quality, and modified by recombination and mutation operators. Finally, the algorithm converges to a set of high-quality rules, which can be applied to future problems, or can become part of an expert system.

The classifier system developed in this research was designed such that the information contained in the database was transformed into a set of production rules that assisted in the selection of a specific detector, given a method and solute properties. This choice was made because the rest of an IC method was often designed around the detection method. Effectively, this meant that every method was classified according to the most probable detection method.

In this paper we show that the developed classifier system produced a set of rules which classified most cases in agreement with the database. These rules were applied to a separate test set of IC methods to demonstrate the applicability to examples that were not presented in the rules before.

2. Methods

2.1. Software

The GA application described in this paper was developed with use of GATES [39,40] (Genetic Algorithm Toolbox for Evolutionary Search, version 2.00). This toolbox was written in ANSI C, and has been extensively used for many other applications [41–46].

2.2. Problem description

Prior to this project, a database of published methods for IC was compiled, which consisted of 6166 cases covering most IC experiments reported in chemical literature in the period 1979 to 1989. For the experiments described in this paper, this database was
split at random into a test set of 1219 cases and a training set of 4947 cases. Each case in the database consisted of 19 attributes, and each attribute could adopt a number of discrete possible values, which is indicated by the number in the brackets:

1. **Ion class of solute [3]**: The ionic nature of the sample (cations, anions, organics).
2. **Acidity of solute [3]**: The acidic/basic nature of the solute (acidic, basic, neutral).
3. **Solubility [4]**: For example, positive, negative, weak positive, weak negative.
4. **Charge [9]**: Charge of the solute (e.g., +, ++, −, −−).
5. **Suppressor [2]**: Indicates whether conductivity detection was suppressed.
6. **Type of solute [6]**: e.g., organic, inorganic, anion, acidic.
7. **Mechanism [5]**: Indicates the chemistry of the separation (e.g., ion exchange, ion interaction).
8. **Post column [2]**: Indicates whether a post-column reaction was used.
9. **Application [14]**: The application domain was defined by this attribute (e.g., environmental, pharmaceutical, waste waters, mining, etc.).
10. **Number of solutes [3]**: The number of ions which needed to be separated and assayed by the method (1–5, 5–10, >10).
11. **Halides [4]**: This defines whether any halides were assayed and the nature of the halide, i.e., whether it is UV absorbing or not.
12. **Sulfates [2]**: This defines whether sulphate or sulphite ions were assayed.
13. **Nitrates [2]**: This defines whether nitrate or nitrite ions were assayed.
15. **UV absorbance [3]**: This defines whether the sample contains just UV absorbing ions, or a mixture of both absorbing and non absorbing ions.
16. **Mobile phase [48]**: This defines the type of mobile phase used by the method.
17. **Gradient [2]**: Indicates whether a gradient in the composition of the mobile phase was used.
18. **Column [7]**: e.g., neural silica, crown ether.
19. **Detector [18]**: This defines the detection method (Table 1).

During the time span covered by these cases IC has developed and some of the earlier experiments have later proved non-optimal. New detectors were devel-

### Table 1

<table>
<thead>
<tr>
<th>Detector</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
</tr>
<tr>
<td></td>
<td>abs</td>
</tr>
<tr>
<td>1. Conductivity</td>
<td>3022</td>
</tr>
<tr>
<td>2. Potentiometry direct</td>
<td>80</td>
</tr>
<tr>
<td>3. Refractive index direct</td>
<td>42</td>
</tr>
<tr>
<td>4. UV/visible direct</td>
<td>680</td>
</tr>
<tr>
<td>5. Atomic emission</td>
<td>46</td>
</tr>
<tr>
<td>6. Conductivity indirect</td>
<td>165</td>
</tr>
<tr>
<td>7. Atomic absorption spectroscopy</td>
<td>28</td>
</tr>
<tr>
<td>8. Coulometry</td>
<td>11</td>
</tr>
<tr>
<td>9. UV/visible indirect</td>
<td>551</td>
</tr>
<tr>
<td>10. Other electrochemical</td>
<td>11</td>
</tr>
<tr>
<td>11. Amperometry direct</td>
<td>123</td>
</tr>
<tr>
<td>12. Refractive index indirect</td>
<td>38</td>
</tr>
<tr>
<td>13. Amperometry indirect</td>
<td>23</td>
</tr>
<tr>
<td>14. Potentiometry indirect</td>
<td>19</td>
</tr>
<tr>
<td>15. Other spectroscopy</td>
<td>4</td>
</tr>
<tr>
<td>16. Fluorescence indirect</td>
<td>71</td>
</tr>
<tr>
<td>17. Fluorescence direct</td>
<td>29</td>
</tr>
<tr>
<td>18. Membrane reactor</td>
<td>4</td>
</tr>
</tbody>
</table>
oped, and an experiment previously performed with one detector would now be performed with a new and more efficient detector. Some researchers had personal choice of detectors and published enough of their experiments to skew some data. The database in other words included noise in terms of non-optimal but possible solutions. It nevertheless represented research and knowledge generated in the area.

When any statistical matching or modeling technique, such as the present classifier system, is applied to a set of data which was not compiled by designed statistical experiments, the task is defined as an attempt to model happenstance data [47]. The recognised hazards of dealing with such data are as follows:

- Inconsistency of data. It is rare for any long record of data to remain consistent, e.g., instruments can be updated, improved, etc.
- Confounding of effects. If the collection of data was not controlled by experimental designs, it is unlikely that confounding effects can be identified.
- Nonsense correlation. If all the relevant variables are, for some reason, not included, it is possible to draw a correlation between two or more variables which is really due to the effect of a third unmeasured or latent variable.

It is not always feasible to collect data from well controlled experiments, as was the case for this example, thus it is essential to be aware of these problems. It is also worth noting that the classification of detectors was a difficult problem because it represented some unusual circumstances under which to test the classifier system. To illustrate this consider some of the problems with the database that were observed during this work:

- The training database contained about 3000 cases which were incomplete, i.e., missing values for one of more attributes.
- Many method and solute properties were associated with each detector classification yet not all were relevant for the decision to use that detector. Therefore, the classifier system had to be capable of filtering the irrelevant attributes.
- The distribution of detectors present in the training database was not uniform. It ranged from over 60% to less than 1%.
- There was a certain level of multiplicity of classification in the database, i.e., one set of method and solute properties could define the use of more than one detector.
- The choice of a specific detector might have been dependent on the availability. This information was, however, rarely reported in literature and therefore could not be part of the database.

The aim of the classifier system was to generate a set of production rules (IF <conditions> THEN <action>) from examples provided by the training database. During the learning stage, relations between the condition and action (classification) part were established; irrelevant condition parts were filtered. Furthermore, parts of the rules were generalised in order to be applicable in different situations, i.e., method and solute properties. Specific rules were necessary to handle the exceptions.

The generation of production rules, which are able to assist in the selection of a detector type given a set of method and solute properties, is a profitable goal, and in addition, because of the problems described above, presented a considerable challenge to the classifier system.

2.3. The classifier system

In recent years, genetic algorithms (GAs) have become an expanding area of research in computational models for machine learning [10,48,49]. These models are loosely based on the principles of natural selection and the survival of the fittest, i.e., they maintain a population of candidate solutions (production rules), which are improved during an iterative cycle in which recombination and mutation operators are applied to an above-average selection of these solutions.

The classifier system basically consists of two parts. The first one is the performance system that receives examples from the environment (database), in order to evaluate the current set of rules for their ability to make classifications that agree with the input. This ability is reflected by the strength of the rule. The second part embodies the genetic operators, which provide mechanisms for generating new, hopefully improved, rules. These two parts are iterated, which allows for convergence to a solution, i.e., set of rules.
Basically, the incremental learning process is established by iterating the process of receiving information from the environment (database), evaluating the current rules by establishing their strengths, and finally, the selection of high strength rules and the application of the genetic operators in order to enhance their performance.

At the start of the GA run, every rule in the population is initialised at random within the allowed range of values, which include the don’t care symbol. Furthermore, the strength of each rule is initialised with a predefined value.

2.3.1. The production rules

The production rules have a pre-defined format according to their classification task. In the current application the rules were of the format:

IF <method and solute properties> THEN <detector>

where the method and solute properties were represented by the first 18 attributes discussed above. In the classifier system, the rules were encoded on so-called strings, i.e., a sequence of values which allowed for easy manipulation by (genetic) operators. To illustrate this, consider the following rule:

```
  2 1 1 2 2 5 5 2 1 1 2 1 3 3 15 1 2 17
```

which denotes that when ion class=2, pH of solute=1, solubility=1, etc., then this method requires detector 17, i.e., the method is assigned to the class with methods using detection method 17.

In order to allow for generalisation of the rules, all attributes which make up the condition part of the rule were allowed to assume the don’t care character #. Generalisation of the above rule might, for example, lead to:

```
  2 1 1  #  #  5 5 2  #  1 1 2 1 3 3 15 1  # 17
```

This rule matches cases in which the charge, suppressor, application and column attributes can have any value. Consequently, these attributes do not influence the classification of these cases. The example rule given here is applicable to $9 \times 2 \times 14 \times 7 = 1764$ different cases, which may or may not be part of the database. The specificity of a rule is defined by the number of specified condition attributes, which is 14 in this example.

The commonly used binary encoding was adopted to encode each attribute with a pre-specified number of bits. If $R$ denotes the number of possible values for a specific attribute, then the number of bits to encode this attribute is given by rounding $\ln R / \ln 2$ to the nearest integer value. For example, the attribute solubility may assume any of 5 different values (4 values according to Table 1, and the don’t care symbol), and consequently requires $\ln 5 / \ln 2 = 2.3 \rightarrow 3$ bits. Then the attribute values are encoded as: 000 $\rightarrow$ 1, 001 $\rightarrow$ 2, 010 $\rightarrow$ 3, 011 $\rightarrow$ 4, 100 $\rightarrow$ #. This binary encoding allowed the use of recombination and mutation operators that manipulate either the individual bits or treat each attribute as an integer.

2.3.2. The performance system

In analogy to regular optimisation problems that include an objective function which must be minimised, the performance system represents the objective function of the classifier system. This objective function comprises a complex iterative process during which the strengths of the current set of rules are established via application to a subset of database cases. In addition, under certain circumstances, new rules are inserted in the population. It is important to notice that the strength of a rule does not directly reflect the number of cases that is classified in agreement with the database. Furthermore, it is more important to have knowledge about the classification capabilities of the total set of rules. The procedure to obtain this information is explained in the results section.

```
  15 1  #  17
```

The performance system that was used for this research comprised only one example of a possible implementation. In the remainder of this section the implementation details of the performance systems are briefly discussed. For a comprehensive explanation of all details the reader is referred to [10,50].
1. Create match set. Each iteration of the performance cycle starts by a random selection of a case from the training database. Subsequently, all the rules in the population are checked for whether their condition parts match the selected case. The rules whose attribute values match the case gain a place in the match set and represent a set of candidate rules for classifying that particular case. Since only the condition part of the rules is matched with the selected case, the match set may contain rules with non-matching detector types. This is, however, only established in step 3. Since not all cases were fully specified, missing attribute values were neglected during the match process. Consequently, a range of rules with different values for the missing attribute(s) could match the selected case. Although this was not ideal it was the best solution because no additional information was available which allowed for another matching strategy.

2. Creation operator. In order to enhance the performance of the classifier system, a modified form of the creation operator was implemented to create matching rules [50]. This operator was applied when the match set was empty in order to make at least one classification at each iteration. The operator was also used to replace all rules with strengths below a pre-specified threshold. If neither of these two conditions applied, then the creation operator was utilised with probability $P_{\text{create}}$ to replace the rule with the lowest strength. This operator worked as follows:

(a) If the match set is empty or when the operator is invoked because of the $P_{\text{create}}$ criterion, then select the rule with the lowest strength from population, else select all rules with a strength below the threshold.

(b) Copy all attribute values from the current database case to the selected rule(s). This results, by definition, in a match set which contains at least one rule.

(c) Generalise the new rule by replacing each attribute value (except the detector type), with a pre-defined probability $P_{#}$, with the don’t care symbol.

(d) Re-assign the value of the prediction (detector) part of the rule, with a probability $P_{\text{action}}$, at random. This prevents a bias of the create operator towards detectors with a high frequency of occurrence in the database.

If new rules were created, a new match set was assembled before proceeding with the next step.

3. Selection of rule. In order to select a rule $i$ from the match set, each rule in this set takes part in an auction in which they make a bid $\text{Bid}_i(t)$ according to their strength $S_i(t)$ and a linear function of the specificity $sp$ [10]:

$$\text{Bid}_i(t) = C_{\text{bid}} \times f(sp) \times S_i(t)$$

where

$$f(sp) = \text{bid1} + \text{bid2} \times sp(t).$$

and $t$ denotes the iteration number of the performance cycle, and bid1, bid2 are explained below. The rule with the highest bid is assumed to be the most likely candidate for making a correct classification, i.e., the bid reflects the past performance of the rule.

Now it would be possible to select the rule with the highest bid deterministically but this could result in premature convergence of the system in a set of non-optimal rules. Instead the roulette wheel technique was used in which a rule is selected stochastically on basis of its relative strength, i.e., the probability for selecting rule $i$ is equal to $S_i(t)/\sum_i S_i(t)$. Although this selection method is not the best possible, and other methods are available, it proved to be adequate for the current application.

The calculation of the bids includes the specificity of the rule as a linear function $f(sp)$ to enhance the formation of so-called default hierarchies. In such a hierarchy, general rules (those with many #’s) cover the situations which are not covered by the specific rules, i.e., exceptions in the database are predicted by specific rules, whereas the rest is handled by more general rules. Accordingly, rules which cover an exception should have a higher chance of being selected, i.e., they should outbid the more general rule. The linear function $f(sp)$ allows to control the contribution of the specificity to the bid of the rule, and bid1 and bid2 should be taken such that $0 \leq f(sp) \leq 1$ because larger values might result in negative strengths, and negative function values do not allow convergence to a steady state [10]. The formation of default hierarchies contributes to a reduction of the number of rules that is required to make correct classifications.

4. Adjustment strengths. The final step in the performance system is the adjustment of the strengths
of the rules according to the new information (case) received from the database. Towards this end, a simple form of the so-called bucket brigade [10,25,48] was used in which the strengths were adjusted according to:

\[ S_i(t + 1) = S_i(t) - \text{Payoff}_i(t) - T_i(t) + R \]

where Payoff_i(t) = Bid_i(t) for the selected rule, and Payoff_i(t) = 0 for all other rules. If the selected rule classified the case correctly (a matching detector on rule and case), it received a reward R. In addition, the rules in the population were subjected to a tax T_i(t) defined by:

\[ T_i(t) = (\text{LifeTax} + \text{BidTax}) \times S_i(t) \]

The LifeTax applied to every rule in the population, and prevents that rules, which (almost) never matched a case, reside in the population by decreasing their strengths. The BidTax only applied to rules in the match set, which ensured that rules which often matched a method description but were never selected to classify the method, also finally disappeared from the population.

By iterating the above steps of the performance cycle, the rules were effectively applied to a subset of the cases of the training database, resulting in an estimate of the classification ability (strength) of the current set of rules. By increasing the size of the subset, the resulting values for the strengths become more reliable, i.e., they better reflect the quality of the rule. On the other hand, in order to minimise the computer-time involved, the subset should be kept as small as possible. However, a small subset requires more GA generations because the strength values become noisy, which makes an effective search difficult, and may even lead to premature convergence to a set of non-optimal rules.

2.3.3. Generation of new rules

Subsequently, to the performance system, the genetic operators are invoked to modify the rules such that improved rules result. Fig. 1 depicts the principle that is taken to generate new rules. A fraction of the population is selected and subjected to crossover, mutation and repair. The modified rules are re-inserted in the population by replacing rules with similar condition and action parts. This mechanism is called crowding, and prevents the population from converging to one 'super' rule. Instead, the system is encouraged to maintain a range of different rules, applicable to different cases.

Selection. In the current classifier system a rank-based threshold selection was used, which is based on the rank of the rules according to their strengths. A threshold is chosen defining the better fraction of the population and subsequently, only strings from this fraction are selected at random to take place in the sub-population. Importantly, by selecting above-average strings, the GA uses information that it has built up in the population during the past iterations, i.e., the GA exploits information from the past.

Crossover and mutation. In addition to the exploitation of previously gathered information, the GA also explores the search space by looking for new information (rules) in regions of the search space that were not visited before. In order to explore the search space, modifications are made to the previously selected strings. Two operators were used to that end. The first one was the crossover operator, which recombined two randomly selected strings, with a predefined probability. In the current implementation uniform crossover was applied to the integer attribute values. A predefined number of parameters (attributes) are selected at random and exchanged with the corresponding parameters on the paired string. From this operation two new strings resulted.

After recombination of the strings, the mutation operator was applied to each string. This operator randomly made changes to the bits on the string; it flipped each bit from 0 to 1 or vice versa with a predefined probability.

Repair. Since a binary encoding was used to represent each attribute on the rule, application of mutation could result in illegal attribute values. As discussed above, it required 3 bits to encode the solubility
attribute. However, this gives rise to \(2^3 = 8\) possible values instead of the required 5, and consequently, 3 values (101, 110, 111) are not allowed. Therefore, a repair function was implemented, which re-assigned an illegal attribute value at random.

**Insert.** Finally, the new rules were inserted in the original population by replacing those rules whose attribute values were most similar, and were contained in the fraction with the low strength rules. The similarity was calculated as the Euclidean distance between the rules on basis of the attribute values. However, because several attributes represented qualitative variables (e.g., type of solution), this distance merely indicated whether the attributes were different or not, and gave no indication about the degree of similarity. If recombination, mutation, and repairing resulted in a rule that is already present in this fraction, then insertion of the new rule was be without effect.

If application of re-combination and mutation resulted to an improvement of a rule, it was assigned an increased strength in the next generation (as a GA iteration is called) and accordingly could survive again in the selection process.

### 3. Experimental

For the experiments described in this paper a population of 1000 production rules was used. This large size was necessary to ensure that the population could accommodate a broad range of rules, and in the same time permitted the manipulation of rules without the good rules being destroyed. The threshold fraction for the rank-based selection method was set to 0.30, and consequently 300 rules were subjected to crossover and mutation in each generation. Uniform crossover was applied with a probability of 1.0, and was parameterised to exchange 4 attribute values between the paired rules. Point mutation was applied with a probability of 0.01. The new rules were inserted by replacing rules from the fraction defining the worst 30% of the population. These GA parameters represent commonly used values in many applications.

The next parameters, which are specific for the classifier system, were obtained by trial and error. It is unclear if these represent optimal values with respect to the performance of the system. Neither it is clear how these parameters influence the behaviour of the classifier system. The number of iterations in the performance system was set to 1500, i.e., a subset of about 1500 cases (because of random selection with replacement) was presented to the rules at each generation to establish the quality of the current population. The initial strength of the rules was set to 2.0, and the strength threshold was fixed at as 0.05. The bidding coefficients \(C_{\text{bid}}, \text{bid1}, \text{and bid2}\) were set to 0.1, 0.125 and 0.04 respectively, and the tax coefficients \(\text{BidTax}\) and \(\text{LifeTax}\) were set to 0.01 and 0.0001. A correct classification of a case resulted in a reward of \(R = 2.0\). The control parameters of the create operator were set to \(P_{\text{create}} = 0.2, P_{\#} = 0.4,\) and \(P_{\text{action}} = 0.2\).

### 4. Results and discussion

The classifier system discussed above was applied to the training database in order to generate a set of production rules. Thereafter, these rules were applied to a separate test set.

Since the strengths of the final set of rules didn’t reflect the number of cases that can correctly be classified, a re-evaluation of the rules was necessary. To this end, the rule set was applied to every example in the database. For each example a match set was assembled, and the rule with the highest strength was selected. Application of this rule revealed whether or not the database case was correctly classified. By counting the number of correctly classified cases the predictive power of the rule set was determined.

Fig. 2 depicts the percentage of correctly classified cases of the training database as a function of the generation. Note that a correct classification means an agreement with the training database. It does, however, not necessarily mean the choice for the best possible detection method (as would, for example, be given by an expert). From this figure it is observed that after the first 1500 iterations of the performance cycle (i.e., first generation), about 55% of the cases were correctly classified. This was mainly a result of the application of the creation operator, which replaced the random initialised rules with matching rules. In the next few generations this percentage increased exponentially, whereafter the progress became more equal. At generation 254, the system attains its highest performance and 3740 (76%) out of 4947 cases were correctly classified.
Fig. 2. Evolution of the percentage of correct classifications of cases in the training database.

Fig. 3. Frequency of use for each rule in \( \{N^*\} \) when applied to the training database. Each point represents a rule and is plotted according to its strength.

Fig. 4. Size of \( \{N\} \) as function of the generation. \( \{N\} \) contains a minimum subset of rules, which were necessary to make the maximum number of correct classifications for the training database.

From this curve it is observed that the number of correct classifications occasionally decreases, which may be due to the fact that rules which made a correct classification in the previous generation(s), were deleted from the population. Deletion could be caused by modification of a low strength rule by the creation, crossover, or mutation operator. Thereupon, re-inserting rules in the population by replacing similar low strength rules was an other potential cause of deletion of good rules. Note that a rule with a low strength was not necessarily a rule with no predictive power, i.e., the low strength could merely be a result of the fact that it classified a method that did not occur often in the training database, and therefore did not have the opportunity to reinforce its strength. This is confirmed by the data shown in Fig. 3, which depicts the strengths of the final set of rules \( \{N^*\} \).

In order to classify the cases in the training database not all 1000 rules in the population were required. To determine the subset of necessary rules \( \{N\} \), the match set for every case in the database was determined (as explained above), and then the rule with the highest strength of each match set was added to \( \{N\} \) if it proposed the correct detector, and if it was not already present in \( \{N\} \). Fig. 4 depicts the size of \( \{N\} \) as function of the generation, and from this it is observed that the size of this set dropped significantly during the first generations of the learning process. The size of \( \{N\} \) corresponding to 76% correct classifications of the database is 252. This set of optimal rules will be denoted as \( \{N^*\} \). However, 72% correct classifications, required less than 200 rules.

Fig. 4 also demonstrates that the rules are generalised during the learning process. In the first few generations, while the number of correct classifications exponentially increased, the size of \( \{N\} \) decreased exponentially. Consequently, the specificity of the rules decreased, which is clearly shown in
Fig. 5, which depicts the average specificity of the rules in \( \{N\} \) as function of the generation. The average specificity of \( \{N^*\} \) is 8.96, which indicates that, on average, the number of don’t care symbols is only increased by one for each rule. However, this already resulted in a reduction from over 400 rules to about 250 rules.

From Fig. 5 it is interesting to observe that the average specificity disclosed a low frequent oscillation, which synchronised with the oscillation observed in the size of \( \{N\} \) (Fig. 4). Apparently, two forces are acting on the rules, i.e., the first trying to produce more specific rules, and a second to generalise the rules. It is obvious that these forces converged to an equilibrium, although it is not clear how to influence this equilibrium in order to attempt a reduction in the size of \( \{N^*\} \).

Table 2 depicts the number of correctly classified cases in the training database specified for each detector by the rules in \( \{N^*\} \). This table also includes the number of rules that predict that specific detector. From this table it is seen that the cases with detector 1 or 4 had the highest percentage of correct classifications and used largest part of the rules in \( \{N^*\} \). The methods using detectors 12, 13, 14, or 15 are never classified correctly, i.e., no rules were learned for these cases. The percentages of correctly classified cases, with any of the other detector types were about 10 to 40%.

**Fig. 6** depicts the distribution of the detectors in the training database, and the rule set \( \{N^*\} \). Furthermore, it includes the number of correct classifications made by \( \{N^*\} \). The similarity of the distributions for the training database and \( \{N^*\} \) is remarkable and indicated that the number of rules necessary to predict a specific detector was proportional to the number of cases using that detector. If the distribution of detectors in \( \{N^*\} \) would be uniform, this would lead to the conclusion that cases using identical detection methods embody the same underlying principles, i.e., they can be generalised by one or few rules. This was clearly not the case in this example, indicating different method and solute properties using the same detection method.

An analysis of the examples in the database revealed that the number of correct classifications obtained in this experiment was not likely to increase much further, because the database contained many cases which were inconsistent, i.e., identical method and solute properties but different detector types. Since the use of different detectors for the same method is often possible, such examples could be found in the database. Accordingly, an inconsistent method did not necessarily mean a wrong method. Nevertheless, it presented a problem to the present classifier system. These inconsistencies may also have been caused due the appearance of missing attribute values. Consider the following example with three cases that are inconsistent (‘?’ denotes a missing attribute value):

![Image](image_url)
Fig. 6. For each detector the percentage of occurrence in the training database and \( \{N_i\} \) is depicted in the first and second bar, respectively. Furthermore, the percentage of correct classifications made by \( \{N_i\} \) is shown in each third bar.

| Case 1 | 1 | 2 | 4 | 8 | 2 | 5 | 1 | 1 | 13 | 2 | 2 | 2 | 1 |
| Case 2 | 1 | 2 | 4 | 8 | 2 | 5 | 1 | 1 | 13 | 2 | 2 | 2 | 1 |
| Case 3 | 1 | 2 | ? | 8 | 2 | 5 | 1 | 1 | 13 | 2 | 2 | 2 | 1 |

The first two cases have identical and specified method and solute properties but predict different detectors (17 and 15), whereas the third case has two missing attribute values (solubility and UV absorbance), which makes it possible to match the same and cases 1 and 3 implied an inconsistency in cases 2 and 3) because in several instances it might still have been possible to generate rules which distinguished between these cases. To illustrate this consider the following two rules.

| Rule1 | 1 | 2 | 4 | 8 | 2 | 5 | 1 | 1 | 13 | 2 | 2 | 2 | 1 |
| Rule2 | 1 | 2 | 3 | 8 | 2 | 5 | 1 | 1 | 13 | 2 | 2 | 2 | 1 |

The first rule matches all three cases and predicts detector 17. The second rule only matches with the third case because the attribute solubility has value 3, which matches the unknown value '?,' and consequently, predicts detector 15. This led to the conclusion that is was probably possible to predict more than 64% (100–36%) of the cases, which was indeed demonstrated by our experiments. However, a significant improvement of this result seemed unlikely.
In order to test whether the classifier system was capable of filtering out attributes which were irrelevant for the classification of the case, we determined the number of don’t care symbols for each attribute of the rules in \( \{ N^* \} \). The resulting distribution is shown in Fig. 7. It is clear that attribute 9 (application) contains a don’t care symbol in more than 200 rules, and consequently may considered to be irrelevant to the selection of a detection method. The use of a post column reaction (attribute 8) occurred to be the most important attribute because it was specified most often. For the other attributes it was more difficult to make any conclusions.

In order to get a more clear picture about the properties of the rules in \( \{ N^* \} \) several additional figures were made. The distribution of the specificities of the rules in \( \{ N^* \} \) is depicted in Fig. 8, from which it is observed that no rules have a full specification, i.e., no don’t care symbols. Neither are there rules which were completely general, i.e., the most general rule contained 15 don’t care symbols.

Fig. 9 depicts the distribution of the strengths of the rules in \( \{ N^* \} \). From this it is observed that only a few rules were able to reinforce their strengths to a high value. Many rules had a low strength, which indicated that the strength gave no information about its ability to make a classification that is in agreement with the database. All rules in \( \{ N^* \} \), with either low or high strengths, were able to make correct classifications.

However, rules with a high strength were, on average, more often applied (Fig. 3).

In order to test whether the rules in \( \{ N^* \} \) were applicable to cases which were not part of the training database they were applied to a separate test set. This test set of 1219 cases contained no inconsistencies like the type discussed above, but contained missing attribute values. The rules in \( \{ N^* \} \) classified 82% of the cases in this test set correctly, and only 116 out of 252 rules were used. In Table 3 the results are depicted, and reveal the same tendency as the predictions made for the training database.

The experiments presented in this paper clearly demonstrated that the classifier system was able to

<table>
<thead>
<tr>
<th>Detector</th>
<th>Prediction</th>
<th>Detector</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>abs</td>
<td>%</td>
<td>abs</td>
</tr>
<tr>
<td>1</td>
<td>832</td>
<td>97.4</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>12.0</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10.0</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>129</td>
<td>68.6</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>33.3</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>42.9</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>31.2</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0.0</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>28.9</td>
<td>18</td>
</tr>
</tbody>
</table>
learn rules by examination of the implicit heuristics contained in the cases of the training database. These rules were able to make classifications which agreed with 76% of the case in the training database, and with 82% of the cases of the separate test set. Due to the inconsistencies in the database (which were partly the result of missing attribute values), these results are not likely to improve much further.

The results obtained in this investigation make a comparison to the previously applied neural network and induction methods [36–38] possible, and will be the subject of a future paper. Furthermore, this research suggest that automated rule generation may indeed be an beneficial approach towards knowledge acquisition, in order to obtain heuristics to be used in knowledge systems.

Further research is necessary to validate the rules that were obtained with the classifier system. It is possible that for method examples, which were not part of the training database, the rules in \( \{ N^* \} \) proposed a wrong or non-optimal detector. However, if the expected detector is unknown, it is difficult to check whether or not this is true. Therefore, application of the rules to a state-of-the-art test set supplied by an expert in the field, would contribute to their validation, i.e., it would indicate how often, on average, a classification is correct. It is, however, more important to have an indication about the correctness of each individual classification. Accordingly, each rule should be associated with a certainty measure. At this moment, only strength, specificity, and frequency of use are associated with each rule. However, these do not allow for an easy (if possible) interpretation in terms of certainty. Consequently, more research is necessary in order to be able to associate a certainty factor with each rule.

A validation should also reveal whether the rules in \( \{ N^* \} \) are also applicable to IC methods that were developed after 1989, and consequently, were no part of the database. Further research is necessary to investigate how the current rule set is changed if new methods are added to the training database, and the learning is re-initiated.

An other part of the validation should focus on the information that finally emerged on the rules. An analysis must reveal whether or not the choice of a
detector is based on sensible relationships (with regard to IC) among the attributes. If this is true, it would contribute to the acceptance of these rules. However, it can be expected that not all rules are based on such relationships because of the problems that were inherent with the used data, and discussed above. This would make their interpretation much more difficult, but does not necessarily lead to a rejection of the rules. The interpretation of the information on rules is a difficult task, and therefore, subject for further research.

An important improvement of the current system would involve the prediction of multiple detectors (instead of a single one), which are compatible with the input method and solute properties. This would partly avoid the problem with the inconsistencies of the method. A certainty factor for each proposed detector would then be of great value in making the final choice (by the consultor of the rules).

5. Conclusion

This research clearly demonstrated that learning production rules from a large set of examples in order to obtain heuristics to be used in an expert system, may be a worthwhile part of the knowledge acquisition process. Although the database presented a difficult example to the classifier system, the final set of rules could classify large part of the training database, and separate test set. However, further research is necessary to validate these rules more extensively.

Acknowledgements

The Netherlands Organisation for Scientific Research (NWO) is acknowledged for their financial support. We thank the Laboratory for Artificial Intelligence of the University of New South Wales for allowing the use of their computer equipment.

References


