# The Effect of Numeric Features on the Scalability of Inductive Learning Programs

Georgios Paliouras* and David S. Brée

Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK
email: paliourg@cs.man.ac.uk, dbree@cs.man.ac.uk

**Abstract.** The behaviour of a learning program as the quantity of data increases affects to a large extent its applicability on real-world problems. This paper presents the results of a theoretical and experimental investigation of the scalability of four well-known empirical concept learning programs. In particular it examines the effect of using numeric features in the training set. The theoretical part of the work involved a detailed worst-case computational complexity analysis of the algorithms. The results of the analysis deviate substantially from previously reported estimates, which have mainly examined discrete and finite feature spaces. In order to test these results, a set of experiments was carried out, involving one artificial and two real data sets. The artificial data set introduces a near-worst-case situation for the examined algorithms, while the real data sets provide an indication of their average-case behaviour.

**Keywords:** empirical concept learning, scalability, decision trees, numeric features

## 1 Introduction

During the last two decades, a large number of empirical concept learning algorithms have been developed. Out of those, the ones we will examine here cover three major categories. The first (C4.5 [18]) induces decision trees and is the most recent version of the well-known ID3 algorithm [17]. The next two (AQ15 [10] and CN2 [2]) generate lists of decision rules and are based on the AQ algorithm [9]. The last (PLS1 [19]) is a conceptual clustering program which is based on the same principle as ID3.

The selected programs have several features which are important to the scalability analysis presented in the paper:

1. Despite differences in the representation of the learned concepts, all four algorithms perform *orthogonal clustering* of the feature space. In other words, the resulting concept can be graphically represented by a set of rectangles,

* The author was partially funded for this work by a research studentship from the Engineering and Physical Sciences Research Council.

aligned in parallel to the axes of the hyper-space, defined by the feature set[2]. This imposes a restriction to the concepts that can be efficiently learned by the algorithms.

2. The algorithms belong to two types, which have been shown (e.g. [21]) to have different computational requirements. The first type consists of algorithms like ID3 and PLS1, which are called *specialisation* algorithms, because they start from the most general concept description and specialise this until it discriminates perfectly between positive and negative examples of the concept. The second type corresponds to *generalisation* algorithms (e.g. AQ15 and CN2), which start from individual positive examples and generate all possible rules that discriminate between them and all negative ones.
3. All four programs can handle numeric features. With the exception of AQ15, the programs deal with numeric features in the following way: at each stage of the learning process they select a threshold value for the feature, which dichotomises the examined set of examples. AQ15 can only deal with integer features, with an upper bound on their value set (*bounded integer* features). It treats those like ordered discrete features, looking for characteristic value ranges.

There have been numerous analyses and comparisons of learning algorithms in the past (e.g. [15], [8], [2], [21], [22], [6], [12], [23], [11], etc.), most of which have concentrated on the classification accuracy of the algorithms. In general, the issue of scalability has been neglected, as only a few of these analyses have dealt with the computational performance of the examined algorithms. The ones which bear some relevance to the work presented here are those carried out by O'Rorke [15], Rendell et al. [21] and Clark and Niblett [2].

The common conclusion of the work on the computational performance of concept-learning algorithms is that *specialisation* algorithms are faster than *generalisation* ones, because they employ less expensive search methods. Additionally, the worst-case complexity of the algorithms has been estimated to be near-linear, in the size of the training set. For example, Rendell et al. [21] estimate the complexity of specialisation algorithms to be of order $O(kae)$, where $e$ is the number of examples, $a$ the number of features and $k$ the number of nodes in the final decision tree or the number of hyper-rectangles generated by the clusterer. $k$ is assumed to be independent of $a$ and $e$, being determined by the complexity of the problem. Clark and Niblett [2] examine also the use of numeric features, deriving the worst-case estimates $O(ae \log e)$ and $O(as(e + \log(as))$ (where $s$ is the maximum star size parameter) for the core components of ASSISTANT (a variant of ID3) and the AQ-based algorithms respectively. These estimates do not deal with the size of the final concept, implicitly making the same assumption as in Rendell et al. [21].

The work presented in the following sections includes a theoretical analysis and an experimental comparison of the examined algorithms. Section 2 describes worst-case scenarios for the two types of algorithms and estimates their computational complexity. Worst-case estimates of the size of the learned concept are

[2] A more extensive account of the orthogonality problem appears in [18], chapter 10.

included in the scenarios. Section 3 presents the results of three experiments, comparing the near-worst-case and average performance of the programs. Finally, Sect. 4 and 5 summarise and compare the results of the theoretical and the experimental analyses.

## 2 Theoretical Analysis

### 2.1 The focus points

For the purpose of calculating the computational complexity of the algorithms, the scale of a learning problem can be defined in terms of two parameters: the *size of the training set* and the *size of the search space*. The former is determined by the number of instances in the training set, while the latter depends on the number of features and their value-sets, which define the set of all possible target concepts that can be learned. This paper concentrates on the size of the training set, examining its effect on the computational requirements of the algorithms. However, the type and value-sets of the features are also used in the definition of the worst-case scenarios.

In most of the previous studies, dealing with the complexity of ML algorithms, the assumption is made that only finite value-sets are used for the features. This assumption was valid for early versions of the algorithms, which could only deal with nominal features. More recent versions, however, can deal with numeric features, whose value-sets need not be finite. The effect of allowing infinite value-sets is that the search space becomes infinite and needs to be limited, typically using the feature values encountered in the training set. As a result, the order of complexity of the algorithms, with respect to the size of the training set, increases.

Most of the above-mentioned studies have also assumed that the size of the final concept description is determined only by the nature of the problem, not the feature-types nor the number of examples. In a worst-case situation, however, it will be shown that the size of the concept description depends on the size of the training set. There are some empirical results [1] which suggest that this is also true in some real-world learning problems.

### 2.2 Specialisation Algorithms

The two specialisation algorithms examined here, i.e., C4.5 and PLS1, behave in a very similar way to the basic ID3 algorithm, which is described in Fig. 1. Thus, ID3 can be used to derive an estimate of the complexity of the algorithms.

The main computational cost of ID3 arises at the stage where all the features are evaluated, in order for the best discriminant to be selected. At this stage, C4.5 discretises numeric features by evaluating all binary splits, based on the feature-values that are observed in the examined subset of the training set. This involves a sorting of the values and the calculation of the entropy for each binary split. In the worst case, each example will assign a different value to the numeric

**Input:** A set of examples $E$, a set of features $A$, a set of class values $C$.
**Output:** A decision tree $T$.

Initially $S = E$ and $N = T =$ root node.

**Dichotomise**$(S, N)$**:**

1. **If** all the members of $S$ belong to the same class make N a *leaf node* and stop dichotomisation.
   **Else** select feature $a_i$ that best discriminates between positive and negative examples in $S$.
2. **For each** value of $a_i$: $v_{ij}$ **do:**
   **Create** new node $N_j$ under $N$.
   **Dichotomise**$(S_{ij}, N_j)$, where $S_{ij}$ is the subset of examples corresponding to $v_{ij}$.
3. **Return** $T$.

**Fig. 1.** The basic ID3 algorithm

feature, resulting in $e' - 1$ possible splits, where $e'$ is the size of the examined subset of examples. Combining this with the linear complexity of the entropy calculation, would result in a quadratic estimate for this stage alone. However, as the feature-values are sorted, the entropy calculations can be optimised, by maintaining frequency counts (see [18]). As a result, the most expensive process is the sorting, which is of order $O(e' \log e')$. This process has to be repeated for all features, resulting in a total cost of $O(ae' \log e')$ for the calculations per node, where $a$ is the number of features.

In the above worst-case situation, it is possible that one node is generated for each value of the numeric feature in the training set. This would happen in the case where perfect discrimination was sought and no better features were provided. Thus, the maximum size of the generated decision tree, measured by the number of non-leaf nodes, is $e - 1$, where $e$ is the size of the training set. An additional worst-case assumption is that the resulting tree be highly skewed, which is the case if a single example is discriminated at each node. In that situation the average size of the subset of the training set examined at each node would be $e' = e/2$. Based on those results, the total cost of the algorithm is $O(ae^2 \log e)$.

Therefore, by estimating the size of the learned concept description, in the worst case, it is shown that the complexity of the ID3 algorithm is over-quadratic in the size of the training set. The assumptions, underlying this result are:

1. Numeric features are used.
2. Each example in the training set assigns a different value to the numeric features.
3. A complete and highly skewed tree results. This means that there are no informative features to support generalisation and no pre-pruning takes place.

These assumptions are strong and are not expected to hold in a typical learning problem. They illustrate, however, how the use of numeric features can affect the computational requirements of the algorithm. Section 3.2 presents a simple artificial problem, which satisfies most of the above assumptions.

### 2.3 Generalisation Algorithms

Out of the two generalisation algorithms examined here, i.e., AQ15 and CN2, only the latter can handle unbounded numeric features, i.e., real numbers and integers of an unlimited range. AQ15 can only deal with integer features, the range of which has to be specified. Therefore the behaviour of CN2, will be examined here. Figure 2 reproduces the description of the algorithm, presented in [2].

As with ID3, the most expensive process, during the search for the best decision rule (*complex*) in CN2, is the evaluation of all possible complexes. The evaluation in this case is done using the likelihood ratio statistic and the complexity of this calculation for each complex is $O(ce)$, where $c$ is the number of classes and $e$ the number of examples in the training set. This process has to be repeated for all the generated complexes, the number of which is determined by the size of the *SELECTORS* set. The maximum size of this set, in the worst-case scenario described above for ID3, is $ae/2$, where $a$ is the number of features, which is the upper limit for the length of the complex. This process is repeated a maximum of $a$ times, since each complex gets specialised by the addition of a selector (conjunctively added condition), which uses a feature that has not been used in the complex so far. Thus the complexity of the search process is $O(a^2e^2)$. Again this estimate can be improved, by optimising the calculation of the likelihood ratio. As in ID3, this involves sorting the numeric feature values and updating the frequency counts, instead of recalculating them. The revised worst-case estimate is $O(a^2e\log e)$.

The number of times this search has to be repeated is $k$, the size of the final concept description (set of decision rules). In the worst case, a complex is generated for each example in the training set and $k = e$. The computational complexity of the process is then $O(a^2ce^2\log e)$. One interesting observation is that the *maximum star size* parameter does not affect the worst-case complexity of the algorithm. The reason for this is that the maximum number of distinct complexes in the worst case is bounded by the size of the *SELECTOR* set, $O(ae/2)$.

Due to the fact that AQ15 can only handle bounded numeric features, the worst-case scenario for the algorithm differs slightly from CN2. In brief, this scenario involves:

1. The generation of as many complexes, as the size of the training set.
2. The examination of all negative examples, each time a new complex is produced, i.e., each complex is maximally specific.
3. The use of an evaluation function that needs to examine the whole training set each time.

Let $E$ be a set of classified examples.
Let *SELECTORS* be the set of all possible selectors.

**Procedure** *CN2(E)*
  Let *RULE-LIST* be the empty list.
  Repeat until *BEST-CPX* is nil or $E$ is empty:
    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 $E'$ from $E$.
      Let $C$ be the most common class of examples in $E'$.
      Add the rule 'If *BEST-CPX* then the class is $C$'
        to the end of the *RULE-LIST*.
  Return *RULE-LIST*.

**Procedure** *Find-Best-Complex(E)*
  Let *STAR* be the set containing the empty complex.
  Let *BEST-CPX* be nil.
  While *STAR* is not empty,
    Specialise 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 unspecialised ones) or null.
      For every complex $C_i$ in *NEWSTAR*:
        If $C_i$ is statistically significant and better than
        *BEST-CPX* by user-defined criteria when tested on $E$,
        Then replace the current value of *BEST-CPX* by $C_i$.
      Repeat until size of *NEWSTAR* $\leq$ user-defined maximum:
        Remove the worst complex from *NEWSTAR*.
      Let *STAR* be *NEWSTAR*.
  Return *BEST-CPX*.

**Fig. 2.** The CN2 algorithm

The second element of this list is the main difference between the two algorithms and increases the complexity of the algorithm, with respect to the size of the training set, by an order of magnitude[3]. The complexity estimate for AQ15 is $O(sa^2ve^3)$, where $v$ is the largest set of feature-values.

[3] The details of the calculation of the complexity estimate for AQ15 are not of particular interest to the paper, and can be found in [16].

### 2.4 Summary of Results

Table 1 presents the results of the computational complexity analysis for each of the four algorithms and for each type of feature they can handle[4].

**Table 1.** Summary of complexity estimates.

| *Algorithms* | *Attribute Types* | | | |
|---|---|---|---|---|
| | nominal/ bounded integer | value-sets[a] | *numeric* unbounded integer | continuous |
| C4.5 | $O(ca^2ve)$ | $O(c^2av^4e^2)$ | $O(ae^2 \log e)$ | $O(ae^2 \log e)$ |
| PLS1 | — | — | $O(cae^2 \log e)$ | — |
| CN2 | $O(ca^3v^{a+1}e)$ | — | $O(ca^2e^2 \log e)$ | $O(ca^2e^2 \log e)$ |
| AQ15 | $O(sa^2ve^3)$ | — | — | — |

[a] Value-sets for nominal features.

Notes:

*$a$ = number of features, $c$ = number of classes, $e$ = the size of the training set, $s$ = maximum star size, $v$ = maximum number of values per feature.*

The following conclusions can be drawn from the presented results:

1. Most of the algorithms can handle nominal features quite efficiently, with respect to the size of the training set. This is because the search space is bounded by the domain-definition of the features.
2. AQ15 is more expensive than the other algorithms.
3. The value-grouping facility provided by C4.5, is expensive in terms of computations.
4. The complexity of the algorithms which can handle integer and real features is the same for both these types. Moreover, this complexity is higher than quadratic for all algorithms.

Nevertheless, one has to be very careful with the interpretation of the results of a worst-case analysis. The situations which were assumed in order to obtain those results are extreme and very atypical of the problems that concept-learning systems are usually required to solve.

## 3 Experimental Investigation

### 3.1 The Set-up

The analysis presented in this section examines the scaling behaviour of the four algorithms, using one artificial and two large real data sets. The desired

[4] These are the results of an extensive analysis, presented in [16].

outcome of the scalability analysis is a relationship between the performance of each algorithm and the size of the training set, which can be compared to the corresponding computational complexity estimate. This relationship can be derived, by measuring the rate at which the CPU-time consumption[5] changes, as the size of the training set increases. For that purpose, the CPU-time consumption of the learning process is measured at different *size-steps*, i.e., training sets of different sizes. The size-steps are determined on a logarithmic scale, starting from a small power of 2, usually $2^6 = 64$ and multiplying by 2 each time. For the real data sets three randomly sampled training sets are used at each size-step and the results of the three individual tests are averaged. The results of the analysis are plotted on a logarithmic scale for both axes and the slope at each point is examined.

In order to reduce the possibility of implementation inefficiencies, the examined algorithms are written in similar, procedural programming languages (C and Pascal) and are all original versions, provided by their developers. Especially CN2 was provided only in the form of executables. Finally, all the experiments were carried out on a 'Sun-SPARCsystem-400' machine (a server similar to a SPARC2), which contains 32 MBytes of fixed memory and 100 MBytes of swap memory. The system runs the 'SUNOS 4.1.2' operating system.

### 3.2 Learning Even Numbers

This is a simple artificial problem, whose purpose is to verify the results of the theoretical analysis. The task is the discrimination between even and odd natural numbers, provided no more information but the numbers themselves. Training sets range from 64 examples to the maximum number that each algorithm can handle and each set contains the first $n$ numbers, where $n$ is the size of the set. With the exception of AQ15, the examined algorithms can deal with this problem, because they can handle unbounded numeric features. However, AQ15 can also be included in the experiment, by varying its upper limit for numeric feature-values, according to the size of the training set. In this way, $v = e$ (Table 1), raising the worst-case complexity estimate to quartic: $O(sa^2e^4)$.

The problem examined here has a number of properties which generate a near-worst-case situation for most of the algorithms:

1. The key element is that there are no similarities between the instances on which induction can be based. In a typical learning problem informative features would be defined, usually by a domain expert, and the data would be represented using these features[6]. For example, in this problem the feature **divisible-by-2**, which would examine the divisibility of each number by 2 is very informative. Since, however, the raw data are used, such additional information is not available.

[5] The built-in C and Pascal functions **getrusage** and **clock** are used for the measurement of time consumption.

[6] Rendell [20] estimates that in the fifteen puzzle problem the feature extraction process provides about 80% of the acquired knowledge.

2. The pattern followed by the data is one that cannot be detected by orthogonal clustering algorithms. As mentioned above, the examined algorithms look for ranges of numeric feature-values that correspond to objects of the same class. In this problem, the largest range of that kind contains a single example.
3. The outcome of the learning process is complete and highly skewed decision trees and complete decision-rule lists. In other words, no generalisation is achieved.
4. The problem is easy to reproduce and uses a single discrete numeric feature and a binary class, minimising the effect of parameters other than the size of the training set.

The only algorithm for which this problem does not approximate the worst-case scenario is AQ15. The reason for this is that the concept-description generated by AQ15 consists of single-selector complexes, decreasing the complexity of the process by two orders of magnitude:

1. Not all negative examples need to be examined during the search for each complex.
2. The evaluation of each complex does not depend on the size of the value-set of the feature, which in this case is equal to the size of the training set.

The results of this experiment are shown in Fig. 3 and 4. Figure 3 presents, on a logarithmic scale, the CPU-time in seconds over the number of examples, so straight lines indicate a polynomial relationship, with the slope indicating the power. In Fig. 4 the slope of the curves at each point is plotted.

The main conclusion to be drawn from these results is that, although polynomial, the performance of the algorithms is mostly worse than quadratic, as predicted by the theoretical analysis. However, the algorithms perform significantly different from each other. More specifically, the generalisation algorithms (AQ15 and CN2) have a consistent, near-quadratic behaviour throughout the experiment. As expected, the performance of AQ15 is better than its worst-case estimate. PLS1 is somewhat worse, starting with an over-quadratic time consumption, which quickly approaches the cubic threshold. The results are less clear for C4.5, which starts below quadratic, but deteriorates to reach the cubic threshold for large sets, which is worse than predicted. A possible explanation for this is that the optimisation assumed in the analysis of the ID3 algorithm (Sect. 2.2) is not implemented.

### 3.3 Letter Recognition

While worst-case analysis serves as a warning, most applications will not encounter such extreme conditions. For this reason, we have carried out experiments using two real data sets, in order to get an indication of the average-case performance. The first set deals with the problem of classifying typed upper case letters of the Latin alphabet, based on a number of statistical properties of their pixel images. The data set was acquired from the UCI Repository [13] and its

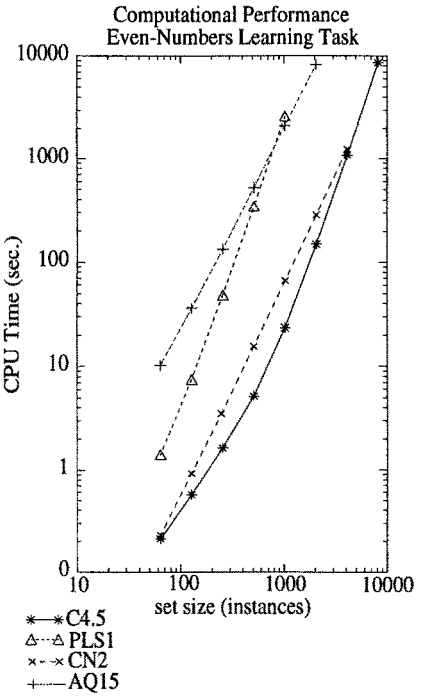


**Fig. 3.** Scalability Results, using the *Even-Numbers* learning task.

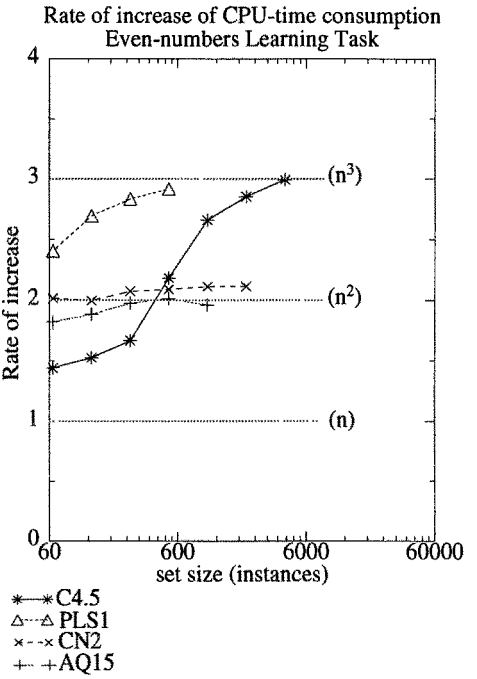


**Fig. 4.** *Even-Numbers Learning Task:* The rate of increase of the CPU-time consumed at each size-step.

original donor was D.J. Slate. Its author has used it as an application domain for Holland-style genetic classifier systems [7]. More recently the data set has also been used in the StatLog project [11]. The data set contains 20,000 instances, of which roughly 16,000 have been used for learning in this experiment. Each instance corresponds to an upper case letter, described in terms of 16 integer features, which take values in the range of 0–15. All of the algorithms examined here can handle bounded integer features and can thus participate in this experiment.

Figures 5 and 6 present the results of the experiment, in the same manner as previously. In addition, the deviations between the three measurements at each size-step are indicated. These are sufficiently small not to affect the results of the comparison.

The performance of the algorithms, in this problem, is polynomial and near-linear. However, despite their similar performance, the actual CPU-time consumption of the algorithms varies substantially. In general, the two generalisation algorithms (CN2 and AQ15) seem to have a higher computational *unit cost* than the specialisation ones (C4.5 and PLS1). This agrees with previously re-

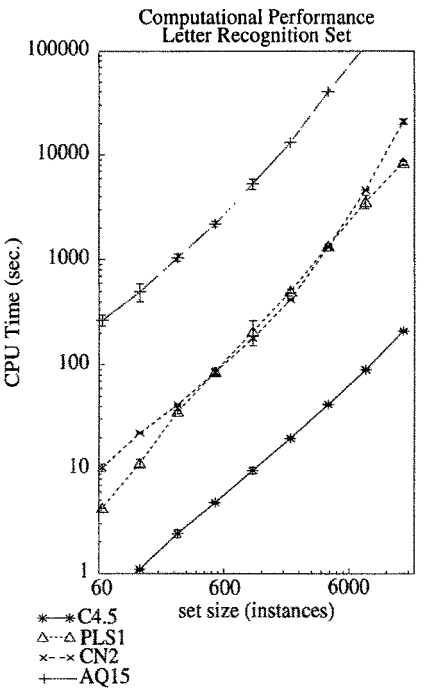


**Fig. 5.** Scalability Results, using the *Letter Recognition* data set.

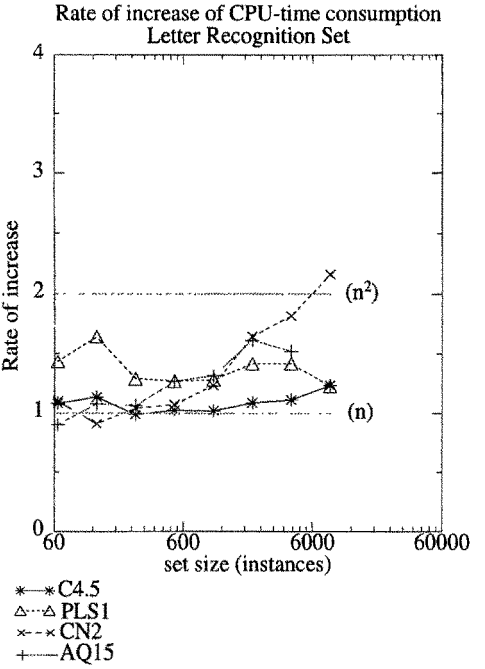


**Fig. 6.** *Letter Recognition Set:* The rate of increase of the CPU-time consumed at each size-step.

ported comparisons (Sect. 1). Another interesting observation, drawn from Fig. 6 is that most of the algorithms start with a very close to linear performance, which worsens as the size of the training set increases. This may be explained by the effect of fixed start-up costs.

### 3.4 Chromosome Classification

The second real data set describes a chromosome analysis task. It is the Copenhagen data set, also used in [4], where an artificial Neural Network system was used for the classification of chromosomes. The data set was provided by the Department of Medical Biophysics, University of Manchester and contains 8,106 examples, of which roughly 6,000 were used for learning in this experiment. Each example corresponds to an instance of a set of 24 chromosomes and is described in terms of 15 real-valued features, which correspond to the grey-level profile of the chromosome. Due to the use of continuous numeric features, only two of the algorithms could be used in this experiment, i.e., C4.5 and CN2.

Figures 7 and 8 present the results, which are similar to the letter recognition experiment, with the exception of an overall increase in the actual CPU-time

consumption values, as a result of the increased difficulty of the problem. The performance of the algorithms remains close to linear.

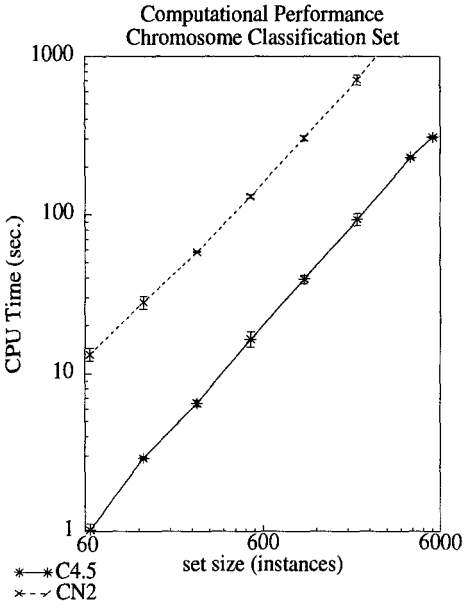


**Fig. 7.** Scalability Results, using the *Chromosome Classification* data set.

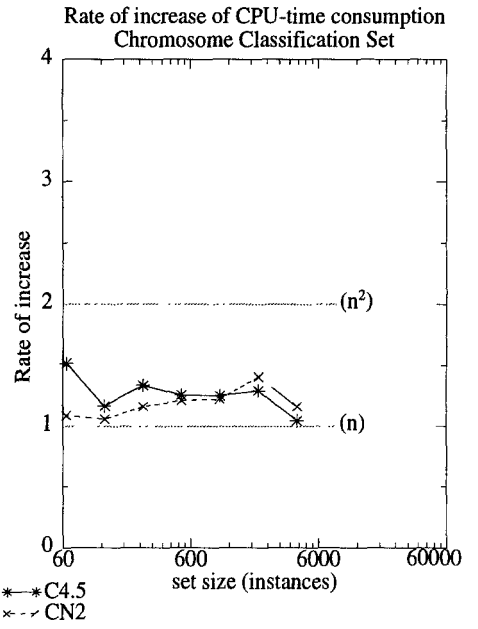


**Fig. 8.** *Chromosome Classification Set:* The rate of increase of the CPU-time consumed at each size-step.

## 4 Discussion of Results

The results of the experiment on the artificial data illustrate how the use of numeric features can increase the computational requirements of the examined algorithms. This is in accordance to the theoretical over-quadratic complexity estimates, presented in Sect. 2, which give an explanation of why this happens. Despite the extreme assumptions of the worst-case scenarios, this analysis supports attempts to reconsider the methods of handling numeric features in concept-learning problems (e.g. [3], [5], [14]).

Another observation, drawn from the results of the first experiment, is that although the generalisation algorithms are in general more expensive, their behaviour is very stable and their CPU-time consumption is comparable to that of the specialisation ones for large data sets. This points to a potential advantage of the AQ-based programs, which however is not observed in the two experiments that use real data.

The average-case performance of the algorithms is significantly better than the worst-case. Their behaviour in both the experiments using real-world data sets was near-linear, despite the fact that non-nominal features were used. Additionally, during these experiments a large difference between the real unit cost of different algorithms has been observed. Although their order of complexity is very similar, some of the algorithms, in particular the generalisation ones, became prohibitively slow for large data sets.

## 5 Conclusion

This paper has looked at the behaviour of four empirical concept learning algorithms on data sets of variable size. Using a computational complexity analysis, it has been shown that, in the worst case, the behaviour of the algorithms is not linear, as previously reported, but higher than quadratic. This result was achieved by the analysis of a parameter, i.e., the size of the concept description, which was assumed to be independent of the size of the training set, an assumption that does not hold in a worst-case situation. The results of this analysis were empirically confirmed, using an artificial problem, which generates a near-worst-case situation. Additionally, two large real data sets were used, in order to gain an indication of the average-case performance of the algorithms. These experiments suggest that the average-case behaviour of the algorithms is near-linear and some of the algorithms (i.e., those having a smaller unit cost) can deal efficiently with large data sets.

**Acknowledgements**
We are grateful to the following people for supplying programs, documentation, data sets and other valuable information:

G. Blix, E. Bloedorn, R. Boswell, P. Errington, J. Graham, R. Michalski, R. Nakhaeizadeh, T. Niblett, P. O'Rorke, R. Quinlan, L. Rendell, M. Rissakis, D. Slate, D. Sleeman

This work was greatly facilitated by the exchange of materials available within the *Concerted Action of Automated Cytogenics Groups*, supported by the European Community, Project No. II.1.1/13, and the use of material from the *UCI Repository of machine learning databases* in Irvine, CA: University of California, Department of Information and Computer Science.

## References

1. J. Catlett. Megainduction: a test flight. In *Proceedings of the Eighth International Workshop in Machine Learning*, pages 596–599, 1991.
2. P. Clark and T. Niblett. The CN2 Algorithm. *Machine Learning*, 3(4):261–283, 1989.
3. T. V. de Merckt. Decision Trees in Numerical Attribute Spaces. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 1016–1021, 1993.

4. P. Errington and J. Graham. Application of Artificial Neural Networks to Chromosome Classification. *Cytometry*, 14:627–639, 1993.
5. U. Fayyad and K. Irani. Mutli-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 1022–1027, 1993.
6. D. Fisher and K. McKusick. An Empirical Comparison of ID3 and Backpropagation. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 788–793, 1991.
7. P. Frey and D. Slate. Letter Recognition Using Holland-Style Adaptive Classifiers. *Machine Learning*, 6:161–182, 1991.
8. M. Gams and N. Lavrač. Review of Five Empirical Learning Systems Within a Proposed Schemata. In *Proceedings of 2$^{nd}$ European Workshop on Machine Learning*, pages 46–66, 1987.
9. R. Michalski. A Theory and Methodology of Inductive Learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–138. Kaufmann, 1983.
10. R. Michalski, I. Mozetic, J. Hong, and N. Lavrač. The Multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *AAAI Proceedings*, pages 1041–1045, 1986.
11. D. Michie, D. Spiegelhalter, and C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Harwood, 1994.
12. R. Mooney, J. Shavlik, G. Towell, and A. Gove. An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 775–780, 1991.
13. P. Murphy and D. Aha. UCI Repository of machine learning databases. Machine Readable data repository, 1994.
14. S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: Randomised Induction of Oblique Decision Trees. In *AAAI Proceedings*, pages 322–327, 1993.
15. P. O'Rorke. A Comparative Study of Inductive Learning Systems AQ11P and ID3 Using a Chess-End Game Problem. Technical Report ISG 82-2, Computer Science Department, University of Illinois at Urbana-Champaign, 1982.
16. G. Paliouras. The Scalability of Machine Learning Algorithms. Master's thesis, Department of Computer Science, University of Manchester, 1993.
17. J. Quinlan. Learning Efficient Classification Procedures and Their Application to Chess End Games. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 463–482. Kaufmann, 1983.
18. J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.
19. L. Rendell. A New Basis for State-Space Learning Systems and a Successful Implementation. *Artificial Intelligence*, 20(4):369–392, 1983.
20. L. Rendell. Conceptual Knowledge Acquisition in Search. In L. Bolc, editor, *Computational Models of Learning*, pages 89–159. Springer Verlag, 1987.
21. L. Rendell, H. Cho, and R. Seshu. Improving the Design of Similarity-Based Rule-Learning Systems. *International Journal of Expert Systems*, 2:97–133, 1989.
22. P. Utgoff. Incremental Induction of Decision Trees. *Machine Learning*, 4(2):161–186, 1989.
23. S. Weiss and I. Kapouleas. An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. In *Proceedings of the Int. Joint Conf. on Artificial Intelligence*, pages 781–787, 1991.