Learning Classifier Systems for Data Mining: A Comparison of XCS with Other Classifiers for the Forest Cover Data Set

A. J. Bagnall
School of Information Systems
University of East Anglia
Norwich, NR4 7TJ, England
e-mail : ajb@sys.uea.ac.uk

G. C. Cawley
School of Information Systems
University of East Anglia
Norwich, NR4 7TJ, England
e-mail : gcc@sys.uea.ac.uk

Abstract— **This paper compares the performance, in terms of prediction accuracy, of a learning classifier system based on Wilson's XCS with commonly used classifiers from the fields of decision trees, neural networks and support vector machines. The experiments are performed on the Forest Cover Type database, a large data set available at the UCI KDD Archive [13]. The first objective of this paper is to highlight the potential of XCS as a data mining tool. The second objective is to provide extensive benchmarking results for experiments performed under randomised conditions for several modelling techniques. We find that C5 Decision trees perform significantly better than other techniques, and that the learning classifier system performs better or as well as three of the eight classifiers used. We discuss why C5 outperforms the other classifiers and identify ways in which XCS could be adapted to make it more suitable for data mining.**

I. Introduction

Learning Classifier Systems (LCSs) are rule based classifiers, often called Genetics Based Machine Learning tools, consisting of a set of rules and procedures for performing classifications and discovering rules using genetic and non-genetic operators. A comprehensive description of LCS can be found in the literature, for example [15], [19]. Recent LCS research is described in [20], [21], [22]. Traditionally, the most common applications of LCSs have been from the domain of reinforcement learning (e.g. Markov decision problems [29]). However, the potential for LCS in supervised learning for data mining has been known for some time [16]. The rationale for believing in this potential is based in part on the following observations concerning the following characteristics of LCS:

- LCS have been shown to be capable of learning complex, non-linear classification functions that can be used to accurately predict new cases, on a variety of problem domains.
- LCS generalise over the attribute space and under ideal conditions can discover a maximally general, accurate rule set to perform classifications.
- The fact that LCS are rule based means they offer the potential for explanatory data analysis in addition to predictive modelling. In real world data mining exercises being able to explain how a technique forms classifications is often as important as accuracy, and techniques where this is difficult (such as neural networks) are often treated with suspicion in industry.
- Unlike most rule induction algorithms LCS do not discover and evaluate rules in isolation. Instead, they search the space of possible rule sets defined for a particular problem.
- In addition to being able to form complete classifications LCS can also be used for nugget discovery (the discovery of classifications for some subset of the attribute space). The degree of coverage of the attribute space required can be controlled by careful parameterisation.
- The way LCS evaluate rules and rule sets (described in Section III) make them ideal for modelling problems where the model may be changing over time, and for maintaining and updating a classification function without the requirement of retraining on all the data.

LCS are similar to Neural Networks and Support Vector Machines in the fact that the generality of models that can potentially be constructed can make them hard to parameterise properly (the similarities between LCS and NN are discussed in [27]). In order to understand how well procedures developed for generated problems such as the multiplexer and the Monk's problem work on real world data sets, a thorough examination and evaluation needs to be conducted. Recently there have been several investigations into applying LCS to machine learning and data mining classification problems [34], [2], [17], [26]. This paper continues this investigation by applying an adaptation of a recently developed LCS, Wilson's XCS [32], to a large multi-class benchmark data set available at the The UCI KDD Archive [13], the Forest Cover Type data set. The Forest Cover data set has been used in some classifier comparisons [4], [23], [10] but the work presented here is, to the best of our knowledge, the first comparative study using Neural Networks (NNs), Support Vector Machines (SVMs) and Decision Trees (DTs). The rest of this paper is structured as follows: Section II describes the data set and the experimental procedure adopted. Section III briefly describes XCS in a data mining context. Section IV outlines the NN, SVM and DT structures used in experimentation. Section V presents the results and Section VI discusses in more detail these results and suggests how XCS could be extended to improve performance.

II. Forest Cover Data

Experiments were performed on the Forest Cover Type data set, available from the UCI KDD Archive [13]. The classification task is to predict the forest cover type (seven classes) for 30m × 30m cells, given only cartographic data [4]. A case consists of observation of ten continuous variables and two nominal categorical variables, wilderness area designation (four types) and soil type (forty types) plus the forest type designation. The nominal categorical variables are represented by 44 binary dummy variables. The database consists of 581012 cases. This data is a representative instance of a domain of problems common in data mining and thus provides a useful benchmark for comparing classifiers. The characteristics that make it of particular interest are:

1) a large number of cases;
2) a relatively large number of attributes and classes for the response;
3) an unequal class distribution for the response classes;
4) both continuous and categorical variables.

Another feature of the data worth noting is that there are no missing values. This is often not the case in data mining problems, but this characteristic allows us to focus on the key performance issues of the classifiers compared. Unless otherwise indicated, the continuous attributes were linearly scaled to $[0, 1]$ and the nominal categorical data was represented with dummy variables.

A. Experimental Design

Previously published results for this data set have used the first 11,340 records for training data, the next 3,780 records for validation data, and last 565,892 records used for testing data. The data was rearranged so that the class frequencies are equal in the training and validation data (thus significantly reducing the number of rare classes in the testing data). Using this design, published accuracy results on the testing data include 70% using back propagation [4], 58% using Linear Discriminant Analysis [4], 71% using Support Vector Machines [23]and 73.41% using SVM modified to allow for the unrepresentative class distribution in the training data [10].

As discussed in [10], balancing the data so that the classes are equally represented in the training data by sampling the complete data set can make the final accuracy measure unrepresentative of the true model. Our goal is to provide an assessment of the accuracy performance of several classification techniques on this data, thus providing results for future classifier comparisons. Hence, the experimental procedure followed is based on that recommended in [25]. The data set was randomly split into 10 sets (thus removing the class frequency bias introduced by balancing). Each set was further split into approximately 38000 training cases and 20000 testing cases. If parameter tuning occurred it was done so on the basis of performance on the first training data set. The size of the data set means there is no need for cross validation, hence when comparing performance it seems reasonable to use a simple t-test with a Bonferroni adjustment for the number of comparisons rather than the alternative tests described in [12].

III. Learning Classifier Systems (LCS) and Wilson's Error Based Classifier (XCS)

LCSs construct a rule set, and hence a classifier, through the iterated exposure to test cases. The LCS receives a single case, attempts to classify this case, then receives a reward quantifying whether the classification was correct or not. A full description of the fundamentals of "Michigan" LCS is given in [15].

LCS rules are usually of the form *if* **condition** *then* **classification**, where the condition is commonly a conjunction of logical expressions on some subset of the attributes. A rule has a set of associated parameters to estimate the quality of the rule and its suitability for use as a basis for creating new rules. The three primary components of an LCS are:

1) a production system that specifies how to construct a mapping, from the rule set, from the attribute space to either a single classification or a prediction of the suitability of some or all possible classifications;
2) a reinforcement algorithm that controls how the rule parameters are adjusted based on the reward feedback concerning the accuracy of predictions;
3) a rule discovery component that alters the rules in the rule set through (possibly genetic) operators.

The training of an LCS proceeds as follows:

- LCS is passed a single training case;
- rules whose condition match the case are formed into a match set;
- the match set is used to form a prediction array, an estimate of the reward (a value attributed to correct classification) for each possible class the case may take;
- based on the prediction array, the LCS selects a classification;
- the reinforcement algorithm receives the reward (often 1000 for a correct classification and 0 for incorrect one) and uses it to adjust the parameters of the rules in the match set advocating the selected action;
- the rule creation algorithm may, depending on some triggering procedure, create new rules by applying genetic operators of crossover and mutation to rules selected from the rule set.

It is worth noting that LCSs are usually applied to reinforcement learning problems, and as such are involved in unsupervised learning (unsupervised in the sense that it is not informed what the best classification could have been or what reward would have been received for alternative classifications).

There are many alternative implementations of LCSs (see [19] for an overview). Wilson's error based classifier, XCS, is a recently developed LCS which is designed to discover accurate, maximally general rules that form a complete classification of the attribute space. XCS was introduced in [32], and has the subject of much recent research [20], [21], [22]. A full description of the basic XCS algorithm is given in [9].

Briefly, XCS works as follows: Each rule in XCS has a prediction parameter, an estimate of the expected reward of the rule, an error parameter, an estimate of the absolute deviation of the reward around the expected value, and a fitness value, an estimate of the accuracy of the rule (an inverse function of its error) relative to other rules in the rule set matching similar attribute values and advocating the same classification.

These parameters are used to form predictions and are updated based on reward feedback. Some of the distinguishing features of XCS are that:

- XCS uses the Widrow-Hoff reinforcement learning algorithm [32] for updating rule parameters.
- XCS assesses a rules fitness (the basis for selection for the genetic algorithm) using an estimate of the classification error of a rule, rather than the expected reward. Thus in terms of creating new rules XCS treats a rule that is always wrong similarly to a rule that is always correct.
- XCS niches the genetic algorithm in a way that aims to maintain complete coverage of the attribute space for all classes.

IV. Classifier Implementations

To provide extensive benchmark results for this data set we assessed the performance of the following classifiers:

A. Classification/Decision Trees

Experiments were performed with three classification tree techniques: C5 [24], CHAID [18] and Classification and Regression Trees (CART) [5] using the Clementine package [28] for C5 and CART and KnowledgeSeeker software [1] for CHAID. The continuous attributes were not scaled for the decision trees since it would make no difference to the trees constructed.

B. Neural Networks

We used two Neural Networks to construct classifiers. The first was the NN provided by Clementine, which we denote ClemNN, with the default parameter settings. The multi-layer perceptron networks [3] have a single hidden layer containing 20 nodes.

The second NN also has a single hidden layer, initially containing 32 or 64 neurons. The output layer utilised the Softmax activation function [6], [7] with a cross-entropy error metric [14] and a standard 1-of-c encoding system. A Bayesian regularisation scheme was used to avoid overfitting, adopting a Laplace prior [31], where the usual regularisation parameters were integrated out analytically in the style of [8]. An important advantage of a Laplace prior over the more common Gaussian prior is that it sets redundant weights to exactly zero, allowing them to be pruned from the network. We call these NNs BayesNN$_{32}$ and BayesNN$_{64}$.

C. Support Vector Machines

SVMs [30] were assessed using the Java implementation of LIBSVM, an integrated software for support vector classification [11]. LibSVM implements the basic SVM algorithm of Platt [23]. The SVM we used was a standard C-Support Vector Classification using the one-against-one approach for multi-class classification in which $k(k-1)/2$ classifiers are constructed and each one trains data from two different classes. Classification is achieved by using a voting strategy for the $k(k-1)/2$ classifiers. We experimented with two kernel types: a linear kernel (LinSVM) and a radial basis function kernel (RadSVM). An adjustment was made to allow for the unbalanced class distributions.

D. XCS

A rule condition for the forest cover data combines the real valued implementation described in [33] and a standard bitstring implementation using the ternary alphabet $\{0, 1, \#\}$. A $\#$ means the rule will match this attribute whether the attribute is value 0 or 1. Wilson's real valued representation stores an interval predicate for each attribute. A rule stores two values for attribute i, $int_i = (c_i, s_i)$ and matches an input value x if and only if $c_i - s_i \leq x \leq c_i + s_i$. XCS was implemented using a maximum rule set size $N = 15,000$, a run size of 500,000 and the standard operators and parameter settings described in [33], [9].

V. Results

The accuracy results for the 9 classifiers used are shown in Table I.

The first point to note is that the results fall into broad categories: less than 70% (LinSVM, CART and XCS), 70%-75% (RadSVM, CHAID and clemNN) and greater than 80% (C5 and BayesNN). Although there are significant differences within these categories, it is probably possible to experiment with parameters to improve each technique's performance within the categories. This implies there are levels of complexity in the underlying relationship between the attributes and the response, and the level of accuracy attained is determined in part by the level of complexity of model considered. This point is reinforced by the fact that the more complex model BayesNN$_{64}$ outperforms BayesNN$_{32}$.

Table I shows that C5 is the best technique for classifying this data. Using a t-test assuming unequal variance, the mean testing accuracy of C5 tests significantly higher than that of all other techniques at the 1% level (even after making a Bonferroni adjustment to allow for multiple tests). An examination of the rule sets derived from the C5 tree indicates that C5 generates approximately 2500 rules, with the majority covering less than 10 cases. Couple this with the fact that C5 is clearly overfitting the data, then the implication is that C5 is modelling almost on a case by case basis.

Restricting C5 to a minimum leaf node size of 20 cases (the default parameter in Clementine is 2) reduces the training/testing accuracy to 84% and 79% and the number of rules to 531 for data set 1. Both CHAID and CART implement stricter stopping criteria than C5, and this is probably why they perform less well. Altering CHAIDs automatic stop size from the default of 5% of the data base to 10 cases improves training/testing accuracy for data set 1 to 87.12% and 79.8%.

TABLE I

ACCURACY RESULTS FOR 10 DISJOINT DATA SETS OF THE FOREST COVER DATA

Class	Model	Train Mean	Test Mean	Test Min	Test Max	Test SD
Trees	C5	95.81%	83.71%	83.44%	83.96%	0.178
	CHAID	74.55%	72.67%	71.71%	73.57%	0.550
	CART	69.16%	68.87%	68.09%	69.3%	0.414
NN	ClemNN	75.54%	74.83%	74.1%	75.55%	0.428
	BayesNN_{32}	82.00%	80.32%	79.71%	80.82%	0.408
	BayesNN_{64}	82.97%	81.08%	80.73%	81.57%	0.278
SVM	LinSVM	69.63%	68.32%	67.35%	69.29%	0.510
	RadSVM	71.55%	70.66%	69.71%	71.3%	0.465
XCS	XCS	67.14%	66.90%	64.13%	70.88%	2.39

The Neural network results illustrate the benefits of the Bayesian regularisation, and reinforce the observation from the DT results that increasing the complexity of the model improves testing accuracy. The NN results are better than those reported in [4]. Balancing the data by duplicating training cases was found to have a detrimental performance on overall accuracy with all techniques, and this may explain the difference.

The SVMs performed worse than C5, CHAID and the NNs, but the results are broadly in line with those reported in [23], [10]. Further experimentation with the regularisation and kernel parameters would probably improve performance. For reasons discussed below, Standard XCS performed poorly.

A. Alterations to XCS

The standard implementation of XCS performed relatively poorly on the training data (see Table I). Consideration of the problem lead us to conclude that this was caused in part by the following factors:

1) The use of dummy variables means that any case will always have exactly 2 one values and 42 zero values. Using standard crossover means that offspring will often have more than 2 one values and will hence never match a case.
2) XCS deletes rules with the goal of maintaining an equal action set size for all niches. However, given the unbalanced class frequencies, this means that XCS is allocating a large proportion of its available rule space to modelling very rare classes.
3) XCS is unsupervised, hence when it performs a classification it is informed (via the passing of a reward) whether a classification was correct or not, rather than being told the correct classification. It samples possible classifications using an exploit/explore strategy (in exploit mode it chooses the classification with the highest estimate prediction, in explore mode it chooses a random classification). This means XCS is spending a large proportion of its run time attempting to gather information which is provided to other techniques in training.
4) XCS attempts to form a complete mapping from the attribute space to the class space, hence it assigns an equal proportion of its rule set to finding incorrect classifications as it does to finding correct classifications.
5) The parameter settings may be suboptimal. Problems 2, 3 and 4 outlined above could all, in theory, be overcome with a large enough rule set and a long enough run. In addition, the variability of the accuracy results relative to the other techniques may indicate that the genetic algorithm is activating too frequently or that the rule set is not large enough.

Time constraints make it unfeasible to properly assess the effect of parameter values on performance (informal experimentation suggested that increasing the rule set size to 30,000 and the run size to 1,000,000 did not significantly improve performance). Instead, we concentrated on alterations specific to data mining in general and the type of problem characterised by the Forest Cover data set. We implemented two alterations to XCS to address the first two points raised above.

1) XCS for Forest Cover Data: XCS_2: To stop the production of infeasible rules, crossover was altered so that it was forced to create a valid offspring. We maintain a bit string representation rather than use a categorical coding, but restrict crossover so that any child will always inherit all the binary variables for either soil type or wilderness area. Retaining a bitstring representation allows for conditions of the form (if soil type != 1 and soil type != 2) with the coding (00**). This type of condition is not easily encoded using a categorical coding.

Mutation is still allowed to change any bit, but now if it changes a bit to a one in either the soil type or wilderness area section and another bit was already set to one, it will adjust the bitstring so that the condition remains feasible.

2) XCS weighted for prior class distributions : XCS_3: In order to force XCS to concentrate on finding rules for the most frequently occurring classes, the deletion probabilities were weighted to increase the probability of deleting rules advocating the classification of the rarer classes. In addition, the GA triggering test was weighted to make the GA occur more frequently in action sets advocating the more frequent classes. XCS_3 uses the GA of XCS_2 and the class weightings.

TABLE II

RESULTS FOR XCS WITH PROBLEM SPECIFIC MODIFICATIONS

Model	Train Mean	Test Mean	Test Min	Test Max	Test SD
XCS_1	67.14%	66.90%	64.13%	70.88%	2.39
XCS_2	69.62%	69.46%	66.91%	70.53%	1.14
XCS_3	71.37%	71.15%	70.06%	72.24%	0.9379

The results for XCS_2 and XCS_3 are given in Table II. XCS_2 provides a significant improvement to XCS_1, and XCS_3 significantly outperforms XCS_2. XCS_3 is significantly better than CART and LinSVM and not significantly worse than RadSVM. XCS_3 also has lower standard deviation than the other XCS implementations, although the fact it is still higher than the standard deviation of the other classifiers is an indication that further improvement could be attained through better parameterisation.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

The first objective of this paper was to provide extensive benchmarking results for the Forest Cover data set using DTs, NNs, SVMs and LCSs. It may well be possible to improve results for these techniques through experimentation with alternative parameters. For example, CHAID may perform better if dummy variables were not used, since it implements special procedures for handling nominal categorical variables (doing this gives training/testing accuracy to 76% and 74% for data set 1). However, it is clear that high accuracy requires a very complex model. The explanatory benefits of these complex models is questionable, and in many data mining problems this complexity is undesirable since it leads to considerable overfitting. C5 achieves the improvement from 75% to 85% by very low coverage leaf nodes which almost adopt a case by case classification. Techniques such as CHAID which are designed to avoid this situation are bound to underperform. When using this data to assess a classifier we would recommend three grades of performance: less than 70% is poor, 70%-75% is adequate and greater than 75% is good. Since it was not the goal of this paper to gain an understanding into the underlying relations in the data, we have not presented any confusion matrices, discussed performance on individual classes or examined rules to look for explanatory relationships.

The second objective of this paper was to assess XCS on the data set. XCS, with problem specific alterations, performs adequately. Although LCS offer real benefits to data miners in terms of ease of understanding and wide scope of applicability, further work is required to properly understand how the numerous parameters and alternative operators employed affect performance on complex, real world data sets. It is thought that more alterations could improve the accuracy on the Forest Cover data set, and may also improve XCS performance on other data mining problems. For example, the real value representation described in [33] and used here involves an interval split on an attribute, and it may be worthwhile experimenting with a binary split representation. In addition, in training XCS does not utilise all the information it could. Rather than only reinforce the selected classification, XCS could be adapted to reinforce all the rules in the match set. Initial experimentation with this supervised learning approach indicates that a re-evaluation of the parameterisation of XCS may be required. XCS attempts to construct a rule set that can evaluate all classifications, even incorrect classifications. This is often desirable but requires a large amount of resources in terms of maximum number of rules. Alterations to direct XCS to focus more on mapping correct classifications may improve performance. Finally, XCS is normally applied to problems where it is assumed a completely correct classification exists, and some of the parameters relating to assessing a rule's fitness reflect this in punishing any misclassification harshly. Further experimentation with a more controlled test problem is required to understand the effects of inherent classification error on performance.

REFERENCES

[1] Angoss. KnowledgeSeeker data mining tool. See `www.angoss.com/ProdServ/AnalyticalTools`.

[2] E. Bernado, X. Llora, and J. M. Garrell. XCS and GALE: A comparative study of two learning classifier systems. In Lanzi et al. [22], pages 115–132.

[3] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press, 1995.

[4] J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 2000.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[6] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogleman Soulie and J. Herault, editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Berlin: Springer-Verlag, 1990.

[7] J. S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 211–217. San Mateo: Morgan Kaufmann, 1990.

[8] W. L. Buntine and A. S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

[9] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. In Lanzi et al. [21], pages 253–272.

[10] G. C. Cawley and N. L. C. Talbot. Manipulation of prior probabilities in support vector classification. In *Proceedings of the International Conference on Neural Networks (IJCNN-2001)*, pages 2433–2438. Washington D.C., 2001.

[11] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[12] T. G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[13] S. Hettich and S. D. Bay. The UCI KDD archive. Irvine, CA: University of California, Department of Information and Computer Science, 1999. `http://kdd.ics.uci.edu`.

[14] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[15] J. H. Holland. *Adaption in Natural and Artificial Systems*. the University of Michigan Press, 1975.

[16] J. H. Holland. Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach*, pages 593–623. Morgan Kaufmann, San Mateo, California, 1986.

[17] John H. Holmes, Dennis R. Durbin, and Flaura K. Winston. The learning classifier system: an evolutionary computation approach to knowledge discovery in epidemiologic surveillance. *Artificial Intelligence In Medicine*, 19(1):53–74, 2000.

[18] G. V. Kaas. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2), 1980.

[19] P. L. Lanzi and R. L. Riolo. A roadmap to the last decade of learning classifier research (from 1989 to 1999). In Lanzi et al. [20], pages 33–61.

[20] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *LNAI*, Berlin, 2000. Springer-Verlag.

[21] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*, Berlin, 2001. Springer-Verlag.

[22] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems*, volume 2321 of *LNAI*, Berlin, 2002. Springer-Verlag.

[23] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In S. A. Solla, T. K. Leen, and K.-R. Muller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 547–553. MIT Press, 2000.

[24] J. R. Quinlan. *C4.5. Programs for Machine Learing*. Morgan Kaufmann, 1993.

[25] S. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

[26] S. Saxon and A. Barry. XCS and the Monk's problem. In Lanzi et al. [20], pages 223–242.

[27] R. E. Smith and H. Brown Cribbs III. Is a learning classifier system a type of neural network? *Evolutionary Computation*, 2(1), 1994.

[28] SPSS. Clementine data mining tool. See `http://www.spssscience.com/clementine/index.cfm`.

[29] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[30] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[31] P. Williams. Bayesian regularization and pruning using a laplace prior. *Neural Computation*, 7:117–143, 1995.

[32] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), 1995.

[33] Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In Lanzi et al. [20], pages 209–222.

[34] Stewart W. Wilson. Mining oblique data with XCS. In Lanzi et al. [20], pages 158–176.