Learning Classifier Systems in Data Mining: An Introduction

Larry Bull[1], Ester Bernadó-Mansilla[2] & John Holmes[3]

[1]School of Computer Science
University of the West of England
Bristol BS16 1QY, U.K.
Larry.Bull@uwe.ac.uk

[2] Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
08022 Barcelona, Spain
esterb@salleURL.edu

[3]Centre for Clinical Epidemiology and Biostatistics
University of Pennsylvania
Philadelphia, PA 19104, U.S.A.
jholmes@cceb.med.upenn.edu

1. Introduction

There is now widespread recognition that it is possible to extract previously unknown knowledge from datasets using machine learning techniques. Learning Classifier Systems (LCS) [Holland, 1976] are a machine learning technique which combines evolutionary computing, reinforcement learning, supervised learning or unsupervised learning, and heuristics to produce adaptive systems. They are rule-based systems, where the rules are usually in the traditional production system form of "IF state THEN action". An evolutionary algorithm and heuristics are used to search the space of possible rules, whilst a credit assignment algorithm is used to assign utility to existing rules, thereby guiding the search for better rules. The LCS formalism was introduced by John Holland [1976] and based around his more well-known invention – the Genetic Algorithm (GA)[Holland, 1975]. A few years later, in collaboration with Judith Reitman, he presented the first implementation of an LCS [Holland & Reitman, 1978]. Holland then revised the framework to define what would become the standard system [Holland, 1980; 1986]. However, Holland's full system was somewhat complex and practical experience found it difficult to realize the envisaged behaviour/performance [e.g., Wilson & Goldberg, 1989] and interest waned. Some years later, Wilson presented the "zeroth-level" classifier system, ZCS [Wilson, 1994] which "keeps much of Holland's original framework but simplifies it to increase understandability and performance" [ibid.]. Wilson then introduced a form of LCS which altered the way in which rule fitness is calculated – XCS [Wilson, 1995]. The following decade has seen resurgence in the use of LCS as XCS in particular has been found able to solve a number of well-known problems optimally. Perhaps more importantly, XCS has also begun to be applied to a number of hard real-world

problems such as data mining, simulation modeling, robotics, and adaptive control (see [Bull, 2004] for an overview) and where excellent performance has often been achieved. Further, given their rule-based nature, users are often able to learn about their problem domain through inspection of the produced solutions, this being particularly useful in data mining. Formal understanding of how such systems work has also increased in recent years (see [Bull & Kovacs, 2005] for an overview). The purpose of this volume is to bring together current work on the use of LCS for data mining since this is the area in which they have experienced the most growth in recent years with excellent performance in comparison to other techniques [e.g., Bernadó et al., 2002].

The rest of this contribution is arranged as follows: Firstly, the main forms of LCS are described in some detail. A number of historical uses of LCS in data mining are then reviewed before an overview of the rest of the volume is presented.

2. Holland's LCS

Holland's Learning Classifier System [Holland, 1986] receives a binary encoded input from its environment, placed on an internal working memory space - the blackboard-like message list (Figure 1). The system determines an appropriate response based on this input and performs the indicated action, usually altering the state of the environment. Desired behaviour is rewarded by providing a scalar reinforcement. Internally the system cycles through a sequence of performance, reinforcement and discovery on each discrete time-step.

The rule-base consists of a population of *N* condition-action rules or "classifiers". The rule condition and action are strings of characters from the ternary alphabet {0,1,#}. The # acts as a wildcard allowing generalisation such that the rule condition 1#1 matches both the input 111 and the input 101. The symbol also allows feature pass-through in the action such that, in responding to the input 101, the rule IF 1#1 THEN 0#0 would produce the action 000. Both components are initialised randomly. Also associated with each classifier is a fitness scalar to indicate the "usefulness" of a rule in receiving external reward. This differs from Holland's original implementation [Holland & Reitman, 1978], where rule fitness was essentially based on the accuracy of its ability to predict external reward (after [Samuel, 1959]).

On receipt of an input message, the rule-base is scanned and any rule whose condition matches the external message, or any others on the message list, at each position becomes a member of the current "match set" [M]. A rule is selected from those rules comprising [M], through a bidding mechanism, to become the system's external action. The message list is cleared and the action string is posted to it ready for the next cycle. A number of other rules can then be selected through bidding to fill any remaining spaces on the internal message list. This selection is performed by a simple stochastic roulette wheel scheme. Rules' bids consist of two components, their fitness and their specificity, that is the proportion of non-# bits they contain. Further, a constant (here termed β) of "considerably" less than one is factored in, i.e., for a rule *C* in [M] at time *t*:

$$\text{Bid}(C,t) = \beta \cdot \text{specificity}(C) \cdot \text{fitness}(C,t)$$

Reinforcement consists of redistributing bids made between subsequently chosen rules. The bid of each winner at each time-step is placed in a "bucket". A record is kept of the winners on the previous time step and they each receive an equal share of the contents of the current bucket; fitness is shared amongst activated rules. If a reward is received from the environment then this is paid to the winning rule which produced the last output. Holland draws an economic analogy for his "bucket-brigade" algorithm (BBA), suggesting each rule is much like the middleman in a commercial chain; fitness is seen as capital. The reader is referred to [Sutton & Barto, 1998] for an introduction to reinforcement learning.

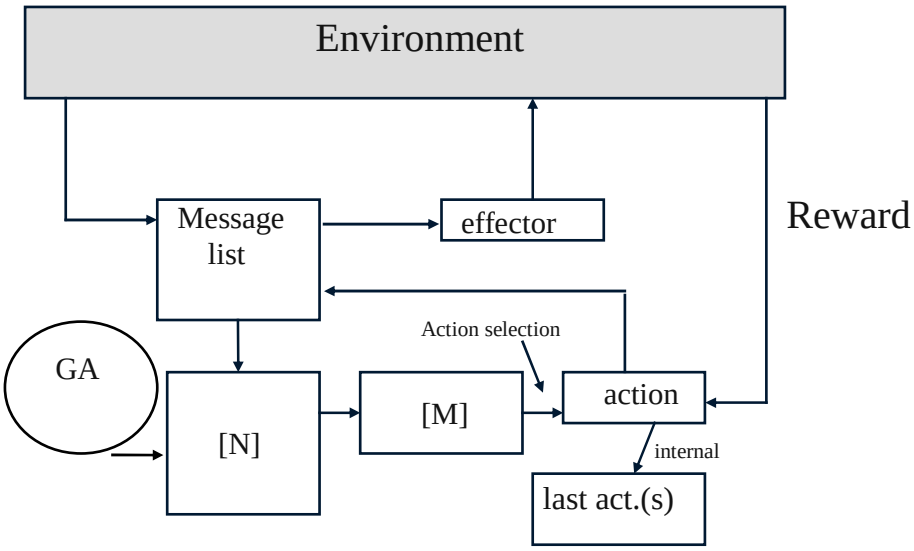

Fig. 1: Schematic of Holland's Learning Classifier System.

The LCS employs a steady-state Genetic Algorithm operating over the whole rule-set at each instance. After some number of time-steps the GA uses roulette wheel selection to determine two parent rules based on their fitness relative to the total fitness of the population:

$$\text{Probabilty\_Selection}(C,t) = \text{fitness}(C,t) / \Sigma\, \text{fitnesses}(t)$$

The effect of this scheme is to bias reproduction towards those rules which appear to lead to higher reward from the environment. Copies are made of the chosen rules which are then subjected to two genetic operators: mutation and crossover. Mutation is applied probabilistically at a per-locus rate (e.g., 1/100) along the length of the rule and upon satisfaction the value at that locus is altered – typically, a locus becomes one of the other two possible values with equal probability. For example, if the above mentioned rule 1#1:0#0 experiences a mutation event on its last locus it could become

1#1:0#1 or 1#1:0##. Crossover begins by randomly choosing a position within the rules and then swaps them from that point to their end. For example, the two rules 000:000 and 111:111 which experience crossover at position two would become 001:111 and 110:000 respectively. The purpose of the genetic operators is to introduce new rules into the population based on known good rules with the aim of discovering better rules. The new rules then replace two existing rules, often chosen using roulette wheel selection based on the reciprocal of fitness. The reader is referred to [Eiben & Smith, 2004] for a recent introduction to evolutionary computing.

It is important to note that the role of the GA in LCS is to create a cooperative set of rules which together solve the task. That is, unlike a traditional optimisation scenario, the search is not for a single fittest rule but a number of different types of rule which together give appropriate behaviour. The rule-base of an LCS has been described as an evolving ecology of rules - "each individual rule evolves in the context of the external environment and the other rules in the classifier system." [Forrest & Miller, 1991]. A number of other mechanisms were proposed by Holland but for the sake of clarity they are not described here (see [Holland et al., 1986] for an overview).

3. Wilson's ZCS

As noted above, Wilson introduced the simple ZCS to increase understandability and performance. In particular, Wilson removed the message list and rule bidding (Figure 2) and did not allow wildcards in actions. He introduced the use of action sets rather than individual rules, such that rules with the same action are treated together for both action selection and reinforcement. That is, once [M] has been formed a rule is picked as the output based purely on its fitness. All members of [M] that propose the same action as the selected rule then form an action set [A]. An "implicit" bucket brigade [Goldberg, 1989] then redistributes payoff in the subsequent action set.

A fixed fraction - equivalent to Holland's bid constant - of the fitness of each member of [A] at each time-step is placed in a bucket. A record is kept of the previous action set $[A]_{-1}$ and if this is not empty then the members of this action set each receive an equal share of the contents of the current bucket, once this has been reduced by a pre-determined discount factor γ (a mechanism used in temporal difference learning to encourage solution brevity [e.g., Sutton & Barto, 1998]). If a reward is received from the environment then a fixed fraction of this value is distributed evenly amongst the members of [A] divided by their number. Finally, a tax is imposed on the members of [M] that do not belong to [A] on each time-step in order to encourage exploitation of the fitter classifiers. That is, all matching rules not in [A] have their fitnesses reduced by factor τ thereby reducing their chance of being selected on future cycles. Wilson considered this technique provisional and suggested there were better approaches to controlling exploration. The effective update of action sets is thus:

$$\text{fitness}([A]) \leftarrow \text{fitness}([A]) + \beta\,[\,\text{Reward} + \gamma\,\text{fitness}([A]_{+1}) - \text{fitness}([A])\,]$$

where $0 \leq \beta \leq 1$ is a learning rate constant. Wilson noted that this is a change to Holland's formalism since specificity is not considered explicitly through bidding and pay-back is discounted by 1-γ on each step. ZCS employs two discovery mechanisms, a steady state GA and a covering operator. On each time-step there is a probability *p* of GA invocation. When called, the GA uses roulette wheel selection to determine two parent rules based on fitness. Two offspring are produced via mutation and crossover. The parents donate half their fitness to their offspring who replace existing members of the population. The deleted rules are chosen using roulette wheel selection based on the reciprocal of fitness. The cover heuristic is used to produce a new rule with an appropriate condition to the current state and a random action when a match-set appears to contain low quality rules, or when no rules match an input.

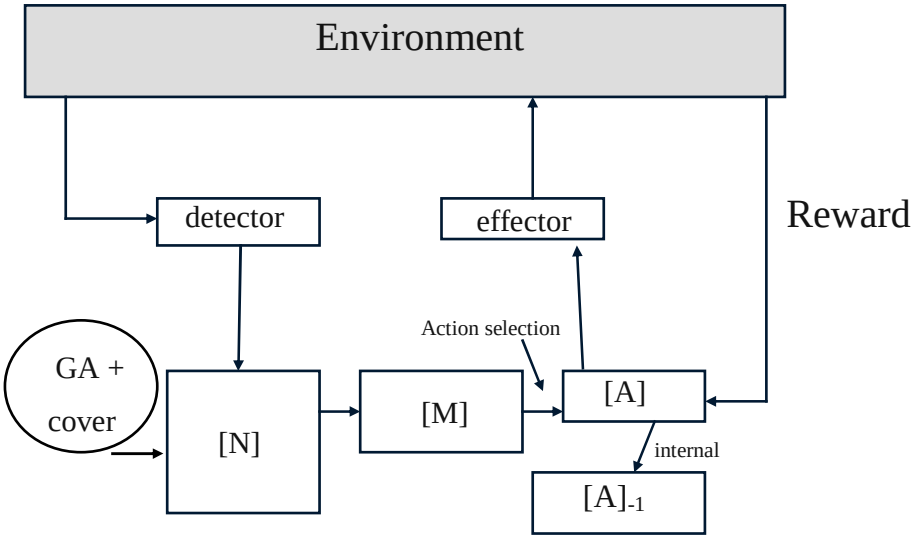

Fig. 2: Schematic of ZCS.

When ZCS was first presented, results from its use indicated it was capable of good, but not optimal, performance [Wilson, 1994][Cliff & Ross, 1995]. More recently, it has been shown that ZCS is capable of optimal performance, at least in a number of well-known test problems, but appears to be particularly sensitive to some of its parameters [Bull & Hurst, 2002].

4. Wilson's XCS

The most significant difference between XCS (Figure 3) and most other LCS (e.g., ZCS) is that rule fitness for the GA is not based on payoff received (*P*) by rules but on

the accuracy of predictions (*p*) of payoff. Hence, XCS has been termed an accuracy-based LCS, in contrast to earlier systems which were for the most part strength-based (also called payoff-based systems). The intention in XCS is to form a complete and accurate mapping of the problem space (rather than simply focusing on the higher payoff niches in the environment) through efficient generalizations. In RL terms, XCS learns a value function over the complete state/action space. In this way, XCS makes the connection between LCS and reinforcement learning clear and represents a way of using traditional RL on complex problems where the number of possible state-action combinations is very large (other approaches have been suggested, such a neural networks – see [Sutton & Barto, 1998] for an overview).

XCS shares many features with ZCS, and inherited its niche GA, deletion scheme and an interest in accuracy from Booker's GOFER-1 [Booker, 1982].

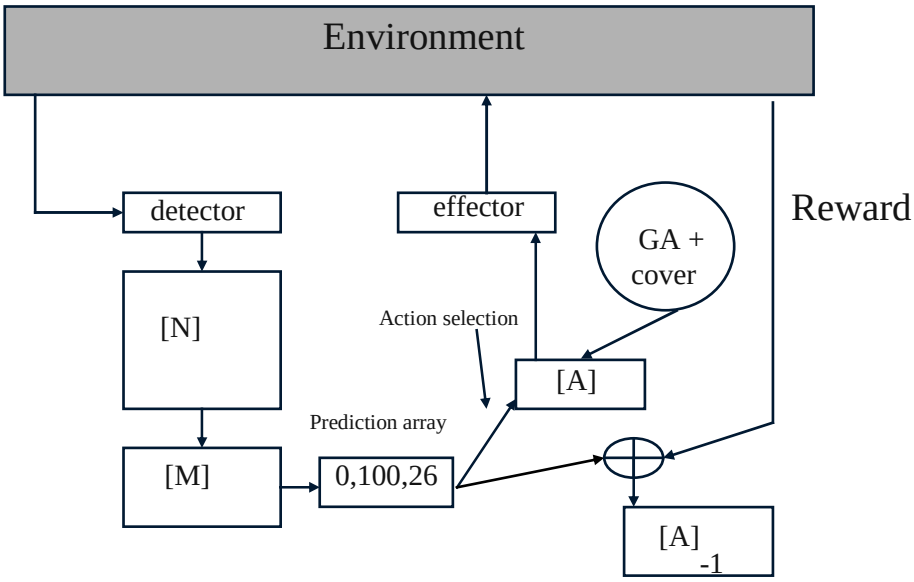

Fig. 3: Schematic of XCS.

On each time step a match set is created. A system prediction is then formed for each action in [M] according to a fitness-weighted average of the predictions of rules in each [A]. The system action is then selected either deterministically or randomly (usually 0.5 probability per trial). If [M] is empty covering is used.

Fitness reinforcement in XCS consists of updating three parameters, ε, p and F for each rule in the current [A]; the fitness is updated according to the relative accuracy of the rule within the set in five steps:

i) Each rule's error is updated: $\varepsilon_j = \varepsilon_j + \beta(| P - p_j | - \varepsilon_j)$ where as in ZCS $0 \leq \beta \leq 1$ is a learning rate constant.

ii) Rule predictions are then updated: $p_j = p_j + \beta(P-p_j)$

iii) Each rule's accuracy κ_j is determined:
$\kappa_j = \alpha(\varepsilon_0/\varepsilon)^{\nu}$ or $\kappa_j = 1$ where $\varepsilon < \varepsilon_0$
where ν, α and ε_0 are constants controlling the shape of the accuracy function.

iv) A relative accuracy κ_j' is determined for each rule by dividing its accuracy by the total of the accuracies in the action set.

v) The relative accuracy is then used to adjust the classifier's fitness F_j using the moyenne adaptive modifee (MAM) procedure: If the fitness has been adjusted $1/\beta$ times, $F_j = F_j + \beta(\kappa_j' - F_j)$. Otherwise F_j is set to the average of the values of κ_j' seen so far.

In short, in XCS fitness is an inverse function of the error in reward prediction, with errors below ε_0 not reducing fitness. The maximum $P(a_i)$ of the system's prediction array is discounted by a factor γ and used to update rules from the previous time step. Thus XCS exploits a form of Q-learning [Watkins, 1989] in its reinforcement procedure, whereas Holland's 1986 system and ZCS both use a form of TD(0) (as noted in [Sutton & Barto, 1998]).

The GA acts in action sets [A], i.e., niches. Two rules are selected based on fitness from within the chosen [A]. Rule replacement is global and based on the estimated size of each action set a rule participates in with the aim of balancing resources across niches. The GA is triggered within a given action set based on the average time since the members of the niche last participated in a GA (after [Booker, 1989]).

XCS is more complex than ZCS but results from its use in a number of areas have been impressive. Wilson originally demonstrated results on the Boolean multiplexer function and a maze problem [Wilson, 1995]. Early on Kovacs emphasised its ability to learn complete, accurate, and minimal representations of Boolean functions [Kovacs, 1997]. XCS has since been widely adopted in the LCS community; the majority of contributions to a recent volume on applications of LCS [Bull, 2004] used XCS. An algorithmic description of XCS can be found in [Butz & Wilson, 2001], while further details of XCS can be found in [Butz, 2005].

5. Pittsburgh-style LCS

The previously described forms of LCS operate under the principle of the GA population as whole forming a cooperative set of rules for the given task. Another approach is to create a more standard GA population in which each individual represents a *complete* set of rules. This approach is known as the Pittsburgh-style LCS after its development by Stephen Smith [1980] at the University of Pittsburgh, USA.

Typically, rules are again of the form "IF state THEN action" but there are no associated rule specific parameters. Rather utility is assigned to the complete set of rules once they have attempted the given task, this representing the fitness metric

under a standard GA. Recombination and mutation act over the set of concatenated rules, i.e., the rules in the set form one linear string of symbols.

Action selection is typically numerosity-based wherein, for a given input, an [M] is formed and the number of rules within each [A] used to determine the output, e.g., under roulette wheel selection.

A brief overview of selected works on LCS in data mining now follows. We concentrate on pre-ZCS and XCS systems in order to complement the remaining chapters of this text.

6. Previous Research on LCS in Data Mining

Data Mining is an overarching term under which various procedures to elicit information from data may be found, including:

- Data Extraction – the collation of data from one or more sources.
- Data Cleansing – the identification and treatment of erroneous or missing datum.
- Data Reduction – the removal of features which are insufficiently correlated to the given task.
- Data Modelling – the discovery of patterns in the data.
- Model Interpretation – identification of the discovered patterns.
- Model Application – use of the identified patterns, e.g., for future predictions.

Machine Learning techniques have shown themselves to be extremely useful in data reduction and modelling, with some utility in the other data mining procedures such as model interpretation (although this can be a somewhat subjective process as different users may find different patterns more interesting than others, e.g., outliers vs. majority).

Goldberg [1983] was the first to apply Holland's LCS to a real-world problem – gas pipeline control. Here the system received hourly readings from various sensors around a network, such as flow rates and pressures, and was required to deduce whether a leak was occurring and to set appropriate control of the pipeline inflow. Using a relatively small rule-base of 60 rules and a message list of size 5, Goldberg was able to train the LCS to become an effective controller after around 1000 days of simulated use. Other early applications of Holland's system include space vessel power management [Goodloe & Graves, 1988] and modelling economic markets [Marimon et al., 1990].

Following Wilson's [1987] early demonstration that a version of Holland's LCS – termed BOOLE - could learn Boolean functions effectively their use for classification has become the most common application of LCS in data mining. Bonelli and Parodi [1991] altered the reinforcement update of BOOLE in their system "Newboole" to penalize incorrect responses as well as reward correct ones. They showed roughly equal performance to the rule inducer CN2 [Clark & Niblett, 1989] and a traditional

neural network on three well-known data sets. They also considered the issue of rule compaction for knowledge discovery in LCS.

Riolo [1991], without the use of a GA, showed human-like behaviour on a two-class discrimination task using Holland's LCS. More recently, Hartley [1999] has reported similar behaviour for XCS, showing a closer agreement to humans in certain classes of problem than Newboole.

Sen [1993] showed how incorporating rule accuracy, in the form of percentage correct over total number of matches, into the action selection scheme of Newboole improves performance. Using versions of the well-known MONKS problem and another data set he showed better classification accuracy than a number of popular rule induction techniques of the time, such as ID3[Quinlan, 1986], CN2, and mFOIL [Lavarac & Dzeroski, 1994]. Saxon and Barry [2000] have used XCS on the MONKS problem.

Frey and Slate [1991] used the percentage correct metric noted above as the fitness measure in a version of Holland's LCS. That is, they used a specific form of accuracy for reproduction, in a similar vein to XCS, and reported good performance on a standard letter recognition task with the modification.

Greene and Smith (e.g., [1994]) developed a version of Holland's algorithm - COGIN - wherein the search is constrained to cover all training set data. The GA is modified to have random selection and competition for replacement based on the number of training examples correctly matched and the current occupation of the niches, i.e., training examples, other rules currently cover. This approach is shown to be competitive with C4. 5.

DeJong, Spears and Gordon (e.g., [DeJong, et al., 1993]) presented early demonstrations of the ability of the Pittsburgh-style systems for data mining with their system GABIL. Using a breast cancer data set they reported favourable performance against C4.5, another ID3 derivative, and two other techniques which form modified DNF hypotheses. In particular, they showed how biased mutation operators which alter the logical relationship between chosen features can greatly improve performance.

Giordana and Neri (e.g., [1995]) developed a hybrid Pittsburgh-Holland approach to evolve Horn clauses through use of a spatially distributed GA. Here each node represents a conjunction and the niching effect of the distribution enables the maintenance of multiple rule types which together solve a problem. Their system, named REGAL, was shown to be comparable to ID3 and C4.5 on a number of well-known test datasets.

Robertson and Riolo [1988] presented a version of Holland's system for a number of letter sequence prediction tasks. Using the traditional windowing approach they report optimality for some tasks and near optimality on alphabet prediction. Extensions to consider using the internal memory list mechanism only was also shown possible, although a number of extra heuristics were added.

Federman and Dorchak[1997] used a version of Goldberg's Simple Classifier System [Goldberg, 1989] which is much like Wilson's BOOLE, to predict the next note in a simple children's songs. They describe a correlation between prediction accuracy of the LCS and an information theory metric; perhaps unsurprisingly, the more information contained in a melody, the easier it is for the LCS to predict the sequence. This was shown to be true for the three rule representation schemes tried.

As discussed thus far, the rule representation of LCS means that the action is not a direct function of the input. Valenzuela-Rendon [1991] introduced a fuzzy set rule representation scheme for LCS which has been used both for classification tasks [e.g., Pena-Reyes & Sipper, 1999] and, in the Pittsburgh-style, for numerical prediction [e.g., Cordon et al., 1999]. In the latter work, Cordon et al. [ibid.] report superior performance to a number of non-linear regression methods and an artificial neural network approach. Ahluwalia and Bull [1999] presented a scheme wherein each action is represented by an evolving arithmetic LISP S-expression, i.e., of the form used in Genetic Programming (GP) [Koza, 1992]. Although they applied it to feature extraction in conjunction with the k-nearest-neighbours algorithm, it could equally be applied to regression problems. More recently, Wilson [2001] introduced a prediction estimation mechanism into XCS – termed XCSF. Weight vectors are added to each rule to enable piecewise-linear approximation based on the input. Bull and O'Hara [2002] show how a neural network representation scheme can be used within XCS and demonstrate its use on both discrete action and prediction tasks. More recently, they have included the use of established gradient descent techniques for learning connection weights in conjunction with the GA search to improve accuracy [O'Hara & Bull, 2005]. The general scheme for using local search heuristics in conjunction with the GA in XCS was introduced by Wyatt and Bull [2004] in their work on using XCS to classify continuous-valued problem spaces.

7. Learning Classifier Systems in Data Mining: An Overview

The rest of this book describes recent research on the use of LCS in the main areas of machine learning data mining: classification, clustering, time-series and numerical prediction, feature selection, ensembles, and knowledge discovery.

Jaume Bacardit et al. – Data Mining in Proteomics with Learning Classifier Systems. Protein structure prediction is a well-known and notoriously difficult problem whose solution offers very real benefits. This contribution describes the use of a Pittsburgh-style system to the problem and highlights the benefits from the human-readability of LCS solutions.

William Browne – Improving Evolutionary Computation based Data Mining for the Process Industry: The Importance of Abstraction. This contribution begins by describing the application of a Holland-style system to data gathered from a steel hot strip mill. The author then suggests that mechanisms for higher levels of abstraction are needed for the real benefits of such machine learning to be seen in many domains.

Hai Dam et al. – Distributed Learning Classifier Systems. For some time now it has been recognized that using multiple classifiers, so-called ensemble machines [e.g., Ho et al., 1994], can prove highly effective. No one technique will outperform all others on all problems, hence the principle of ensemble machines is to combine the output of several methods to find an overall solution that utilises the strength of the constituents and compensates for their individual weaknesses. This contribution presents the use

of LCS in ensembles in a truly distributed framework together with mechanisms to exploit their population-based search characteristics.

Faten Kharbat et al. – Knowledge Discovery from Medical Data: An Empirical Study with XCS. As noted above, XCS has proven to be a particularly effective data miner. This contribution describes the use of XCS to mine breast cancer data obtained from a UK health trust and its improved performance, both in terms of accuracy and the interestingness of its learned rules, compared to C4.5.

Albert Orriols-Puig & Ester Bernadó-Mansilla – Mining Imbalanced Data with Learning Classifier Systems. The class imbalance problem can be defined as a problem encountered by any inductive learning system in domains for which one class is under-represented and which assume a balanced class distribution in the training data. For a two-class problem, the class defined by the smaller set of examples is referred to as the minority class while the other class is referred to as the majority class. This contribution considers how XCS can be modified to consider this problem automatically, i.e., such that no extra forms of data manipulation, as typically used, can be avoided.

Robert Smith et al. – XCS for Fusing Multi-Spectral Data in Automatic Target Recognition. This contribution describes the use of XCS for both classification and as a pre-processor for classification. That is, XCS as a feature selection approach is demonstrated on an image processing task.

Christopher Stone & Larry Bull – Foreign Exchange Trading using a Learning Classifier System. The fact that LCS learn incrementally means they are particularly suited to on-line applications such as time-series prediction. This contribution describes a version of ZCS applied to an on-line trading task and is shown to be competitive with an approach which learns off-line in the traditional batch training mode.

Kreangsak Tamee et al. – Towards Clustering with Learning Classifier Systems. The LCS paradigm is also applicable to unsupervised learning tasks. This contribution describes modifications made to XCS to enable the identification of clusters within data sets without the prior definition of how many clusters are expected.

Albert Orriols-Puig et al. - Comparison of Several Genetic-Based Supervised Learning Systems. As noted above, it is possible to use a number of representation schemes within LCS. This contribution compares both Pittsburgh and Michigan systems using fuzzy logic representations to a number of well-known techniques.

8. Summary

Just over thirty years after Holland first presented the outline for Learning Classifier System paradigm, the ability of LCS to solve complex real-world problems is

becoming clear. In particular, their capability for rule induction in data mining has sparked renewed interest in LCS. This article has given a brief introduction to LCS and previous studies of their use for data mining. The rest of the book brings together work by a number of individuals who are demonstrating their good performance in a variety of domains.

Acknowledgements

Thanks to everyone involved in this edited collection: Professor Kacprzyk for agreeing to publish the book in his series and the authors without whose efforts there would be no book.

References

Ahluwalia, M. & Bull, L. (1999) Coevolving Functions in Genetic Programming: Classification using K-nearest-neighbour. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds) *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, pp947-952.

Bernadó, E., Llora, X., Garrell, J.M. (2002) XCS and GALE: A Comparative Study of Two Learning Classifier Systems on Data Mining. In Lanzi, Stolzmann & Wilson (eds) *Advances in Learning Classifier Systems*, pp. 115-132, LNAI 2321, Springer.

Bonelli, P. & Parodi (1991) An Efficient Classifier System and its Experimental Comparison with Two Representative Learning Methods on Three Medical Domains. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 288-295, Morgan Kauffman.

Booker, L. (1989) Triggered Rule Discovery in Classifier Systems. In Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 265-274. Morgan Kaufmann.

Bull, L. (2004)(ed.) *Applications of Learning Classifier Systems*. Springer.

Bull, L. & Hurst, J. (2002) ZCS Redux. *Evolutionary Computation* 10(2): 185-205.

Bull, L. & Kovacs, T. (2005)(eds.) *Foundations of Learning Classifier Systems*. Springer.

Butz, M. & Wilson, S.W. (2001) An Algorithmic Description of XCS. In *Advances in Learning Classifier Systems: Proceedings of the Third International Conference – IWLCS2000*. Springer, pp. 253-272.

Bull, L. & O'Hara, T. (2002) Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems. In W.B.Langdon, E.Cantu-Paz, K.Mathias, R. Roy, D.Davis, R. Poli, K.Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A.C. Schultz, J. F. Miller, E. Burke & N.Jonoska (eds) *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, pp905-911.

Butz, M. (2005) *Rule-Based Evolutionary Online Learning Systems*. Springer.

Clark, P. & Niblett, P. (1987) Induction in noisy Domains. In I. Bratko & N. Lavrac (eds) *Progress in Machine Learning*, pp11-30, Sigma Press.

Cliff, D. & Ross, S. (1995) Adding Temporary Memory to ZCS. *Adaptive Behavior* 3(2): 101-150.

Cordon, O., Herrera, F. & Sanchez, L. (1999) Solving Electrical Distribution Problems using Hybrid Evolutionary Data analysis Techniques. *Applied Intelligence* 10(1): 5-24.

DeJong, K., Spears, W. & Gordon, D. (1993) Using Genetic Algorithms for Concept Learning. *Machine Learning* 13: 161-188.

Eiben, A. & Smith, J. (2003) *Introduction to Evolutionary Computing*. Springer.
Federman, F. & Dorchak, S. F. (1997) Information Theory and NEXTPITCH, A Learning Classifier System. In T. Baeck (ed.) *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 442-448. Morgan Kaufmann.
Forrest, S. Miller, J.H. (1991) Emergent Behaviour in Classifier Systems. In S. Forrest (ed) *Emergent Computation*. MIT Press, pp213-227.
Frey, P. & Slate, D. (1991) Letter Recognition using Holland-style Adaptive Classifiers. *Machine Learning* 6: 161-182.
Giordana, A. & Neri, F. (1995) Search-Intensive Concept Induction. *Evolutionary Computation* 3: 375-416.
Goldberg, D.E. (1983) Computer-aided Gas Pipeline Operation using Genetic Algorithms and Rule-learning. Ph.D. Thesis, University of Michigan.
Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.
Goodloe, M. & Graves, S.J. (1988) Improving Performance of an Electric Power Expert System with Genetic Algorithms. In *Proceedings of the 1st International Conference on the Applications of Artificial Intelligence and Expert Systems*. IEA/AIE-88, pp298-305.
Greene, D.P. & Smith, S.F. (1994) Using Coverage as a Model Building Constraint in Learning Classifier Systems. *Evolutionary Computation* 2(1): 67-91.
Hartley, A. (1999) Accuracy-based fitness Allows Similar Performance to Humans in Static and Dynamic Classification. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds) *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, pp266-273.
Ho, T., Hull, J.J. & Srihari, S.N. (1994) Decision Combination in Multiple Classifier Systems. *IEEE Trans on PAMI* 16 (1): 66-75.
Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press.
Holland, J.H. (1976) Adaptation. In Rosen & Snell (eds) *Progress in Theoretical Biology*, 4. Plenum.
Holland, J.H. (1980) Adaptive Algorithms for Discovering and using General Patterns in Growing Knowledge Bases. *International Journal of Policy Analysis and Information Systems* 4(3): 245-268.
Holland, J.H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, Carbonell, & Mitchell (eds) *Machine learning, an artificial intelligence approach*. Morgan Kaufmann.
Holland, J.H. & Reitman, J.H. (1978) Cognitive Systems Based in Adaptive Algorithms. In Waterman & Hayes-Roth (eds*) Pattern-directed Inference Systems*. Academic Press.
Holland, J.H., Holyoak, K.J., Nisbett, R.E. & Thagard, P.R. (1986) *Induction: Processes of Inference, Learning and Discovery*. MIT Press.
Koza, J.R. (1994) *Genetic Programming.* MIT Press.
Kovacs, T. (1997) XCS Classifier System Reliably Evolves Accurate, Complete and Minimal Representations for Boolean Functions. In Roy, Chawdhry, & Pant (eds) *Soft Computing in Engineering Design and Manufacturing*, pp. 59-68. Springer-Verlag.
Lavarac, N. & Dzeroski, S. (1994) *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.
Marimon, R., McGrattan, E. & Sargent, .J. (1990) Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents. *Journal of Economic Dynamics and Contro*l 14: 329-373.
O'Hara, T. & Bull, L. (2005) A Memetic Accuracy-based Neural Learning Classifier System. In *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, pp2040-2045.

Riolo, R. (1991)_ Modeling Simple Human Category Learning with a Classifier System. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp324-333.

Robertson, G. & Riolo, R. (1988) A Tale of Two Classifier Systems. *Machine Learning* 3: 139-159.

Pena-Reyes, C. & Sipper, M. (1999) A Fuzzy-Genetic Approach to Breast Cancer Diagnosis. *Artificial Intelligence in Medicine* 17(2): 155.

Quinlan, J.R. (1986) Induction of Decision Trees. *Machine Learning* 1: 18-106.

Samuel, A.L. (1959) Some Studies in Machine Learning using the Game of Checkers. *IBM Journal of Research and Development* 3: 211-229.

Saxon, S. & Barry, A. (2000) XCS and the Monk's Problems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. pp 223-242. Springer.

Sen, S. (1993) Improving Classification Accuracy Through Performance History. In S. Forrest (ed) *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 652. Morgan Kaufmann.

Sutton, R. & Barto, A. (1998) *Reinforcement Learning*. MIT Press.

Valenzuela-Rendon, M. (1991) The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp346-353.

Watkins, C.J. (1989) Learning from Delayed Rewards. Ph.D. Thesis, Cambridge University.

Wilson, S.W. (1987) Classifier Systems and the Animat Problem. *Machine Learning* 2: 199-228.

Wilson, S.W. (1994) ZCS: A Zeroth-level Classifier System. *Evolutionary Computation* 2(1):1-18.

Wilson, S.W. (1995) Classifier Fitness Based on Accuracy. *Evolutionary Computation* 3(2): 149-76.

Wilson, S.W. (1998) Generalization in the XCS Classifier System. In Koza et al. (eds.) *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 322-334. Morgan Kaufmann.

Wilson, S.W. (2001) Function Approximation with a Classifier System. In W.B.Langdon, E.Cantu-Paz, K.Mathias, R. Roy, D.Davis, R. Poli, K.Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A.C. Schultz, J. F. Miller, E. Burke & N.Jonoska (eds) *GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, pp974-981.

Wilson, S.W. & Goldberg, D.E. (1989) A critical review of classifier systems. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 244-255, Morgan Kauffman.

Wyatt, D. & Bull, L. (2004) A Memetic Learning Classifier System for Describing Continuous-Valued Problem Spaces. In N. Krasnagor, W. Hart & J. Smith (eds) *Recent Advances in Memetic Algorithms*. Springer, pp355-396.