รายงานวิจัยฉบับสมบูรณ์

โครงการเงื่อนไขแบบวงรีในการจัดกลุ่มข้อมูลด้วยระบบการเรียนรู้ของตัวจำแนกประเภท

ดร.เกรียงศักดิ์ เตมีย์

เดือน เมษายน ปี 2557 ที่เสร็จโครงการ

สัญญาเลขที่ R2556C062

รายงานวิจัยฉบับสมบูรณ์

เงื่อนไขแบบวงรีในการจัดกลุ่มข้อมูลด้วยระบบการเรียนรู้ของตัวจำแนกประเภท

ดร.เกรียงศักดิ์ เตมีย์
ภาควิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ
คณะวิทยาศาสตร์ มหาวิทยาลัยนเรศวร

สนับสนุนโดยกองทุนอุดหนุนวิจัยมหาวิทยาลัยนเรศวร

กิตติกรรมประกาศ

โครงการวิจัยนี้ได้รับทุนอุดหนุนจากงบประมาณรายได้ ทุนอุดหนุนการวิจัย มหาวิทยาลัยนเรศวร ประจำปีงบประมาณ พ.ศ. 2556 วงเงินงบประมาณ 180,000 บาท

ขอขอบคุณภาควิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ คณะวิทยาศาสตร์ มหาวิทยาลัยนเรศวร ที่ให้การสนับสนุนวัสดุอุปกรณ์ รวมถึงสถานที่ในการทำวิจัย

ประโยชน์อันพึงมีจากรายงานวิจัยฉบับนี้ ผู้จัดทำขอมอบให้กับผู้มีพระคุณทุกท่าน หากรายงานฉบับนี้มีข้อผิดพลาดประการใดผู้จัดทำขอน้อมรับไว้เพียงผู้เดียว

เกรียงศักดิ์ เตมีย์

บทคัดย่อ

ในรายงานฉบับนี้แสดงให้เห็นถึงวิธีการจัดกลุ่มข้อมูลแบบใหม่คือเงื่อนไขแบบวงรีในการจัดกลุ่มข้อมูลด้วยระบบการเรียนรู้ของตัวจำแนกประเภท โดยใช้หลักการทั้งขั้นตอนวิธีเชิงวิวัฒนาการ การเรียนรู้แบบเสริมกำลัง และวิธีการต่างๆ ตามรูปแบบของการเรียนรู้แบบ XCS จุดประสงค์ของงานวิจันนี้คือการสร้างกฎที่สามารถเรียนรู้ได้ พร้อมทั้งนำกฎเหล่านั้นไปใช้ในการจัดกลุ่มข้อมูลได้อย่างถูกต้องโดยปราศจากการบ่งบอกจำนวนกลุ่มของข้อมูล จากผลการทดลองทั้งข้อมูลที่สังเคราะห์ขึ้น และข้อมูลจริงๆ แสดงให้เห็นได้อย่างชัดเจนว่าวิธีการจัดกลุ่มที่ได้นำเสนอมีความถูกต้องทั้งในแง่ของการจัดกลุ่มข้อมูล และจำนวนกลุ่ม

Abstract

This report presents a novel approach to clustering on the ellipsoidal condition structures using an accuracy-based Learning Classifier System. Our approach achieves this by exploiting evolutionary computing, reinforcement learning and the generalization mechanisms inherent to XCS. The purpose of our work is to develop learning rules which accurately describe clusters without prior assumptions as to their number within a given dataset. Experiments on synthetic and real datasets confirm that improvements in both accuracy and finding number of clusters are achieved.

Contents

CHAPTER 1

Introduction

In previous works, we have showed initial results from a rule-based approach to clustering through the development of an accuracy-based Learning Classifier System (LCS) in two versions. First, a version of the simple accuracy-based YCS [Bull, 2005], called YCSc [Tamee et al., 2006]. Second, a version of the accuracy-based XCS [Wilson, 1995], called XCSc [Tamee et al., 2007]. Then, review YCSc and XCSc together [Tamee et al., 2007]. All of them, the condition part specifiers a hyperrectangle as in the original XCSR algorithm [Wilson, 2000]. Wilson has introduced the condition as hyperrectangle by Center-Spread representation. In the Center-Spread Representation, an interval predicate takes the form (ci, si) where ci is the center of the interval and si is the width of the interval from the center. The later, Wilson [2001] modified the Center-Spread representation to the Lower-Upper representation, an interval predicate is represented as (li, ui) where li and ui are the minimum and maximum bounds of the interval. Stone and Bull [2003] analyzed two interval-based representations which are commonly used in XCS. They proved with sufficient evidences that there exists a bias in representational methods and used operators. They proposed a new interval-based representation called the Unordered-Bound Representation. They also proposed a new test problem which can be used to determine wheatear there exist bias or not. Recently, Butz [2005] have proposed XCSR applies as a function approximation system that partitions the input space. That paper changes the classifier conditions from hyberrectangle to hyperspheres and hyperellipsoidal and investigates the consequent performance impact. That paper conclude that hyperellipsoidal further facilitate the approximation of unequally structured dimensions.

In this paper is interested in the utility of such systems to perform unsupervised learning tasks. The paper is structured as follows: first we are talking about the alterations to XCS and then present initial results. Then, we describe the rule compaction for clustering with LCS. Next, the conditions of rules are changed to hyperellipsoidal and then present the results. Next, we give performance on increasingly difficult synthetic datasets are used to test the algorithm and an application to real datasets. Finally, we present our conclusions.

CHAPTER 2

Learning Classifier Systems

2.1 Introduction

Learning Classifier Systems (LCS) are a machine learning technique which combines evolutionary computing, reinforcement learning, supervised learning or unsupervised learning, and heuristics to produce adaptive systems. They are rulebased 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 and based around his more wellknown invention – the Genetic Algorithm (GA). A few years later, in collaboration with Judith Reitman, he presented the first implementation of an LCS. Holland then revised the framework to define what would become the standard system. However, Holland's full system was somewhat complex and practical experience found it difficult to realize the envisaged behaviour/performance and interest waned. Some years later, Wilson presented the "zeroth-level" classifier system, ZCS which "keeps much of Holland's original framework but simplifies it to increase understandability and performance". Wilson then introduced a form of LCS which altered the way in which rule fitness is calculated – XCS.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 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 areas such as data mining or safety-critical control for example. However their combination of two machine learning techniques and potentially many heuristics means that formal understanding of LCS is non-trivial. That is, current formal understanding of, for example, Genetic Algorithms and Reinforcement Learning is significant but understanding of how the two interact within Learning Classifier Systems is severely lacking. The purpose of this volume is to bring together current work aimed at understanding LCS in the hope that it will serve as a catalyst to a concerted effort to produce such understanding. The rest of this contribution is arranged as follows: Firstly, the main forms of LCS are described in some detail. A number of historical studies are then reviewed before an overview of the rest of the volume is presented.

2.2 Holland's LCS

Holland's Learning Classifier System receives a binary encoded input from its environment, placed on an internal working memory space - the black board like message list (Figure 2.1). 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, where rule fitness was essentially based on the accuracy of its ability to predict external reward.

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*:

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

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 "bucketbrigade" algorithm (BBA), suggesting each rule is much like the middleman in a commercial chain; fitness is seen as capital.

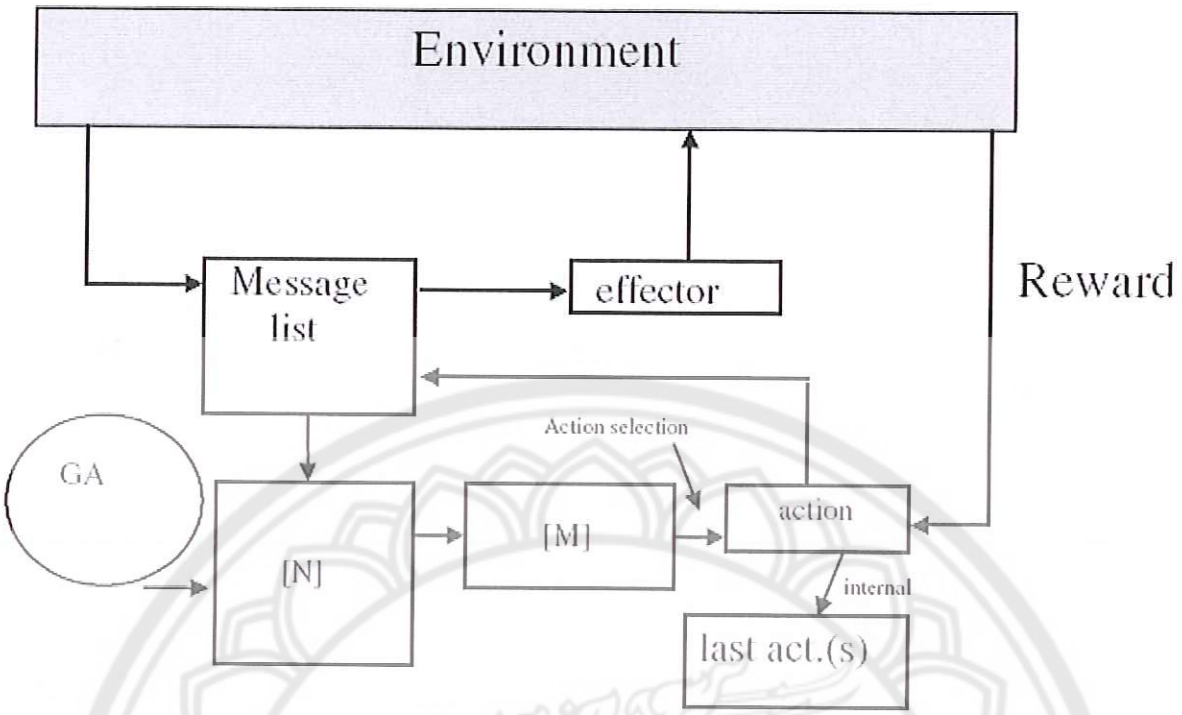


Figure 2.1 Schematic of Holland's Learning Classifier System.

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:

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

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 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 optimization 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." A number of other mechanisms were proposed by Holland but for the sake of clarity they are not described here for an overview.

2.3 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 2.2) 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 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 γ factor (a mechanism used in temporal difference learning to encourage solution brevity). 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 T 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 $1-\gamma$ 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.

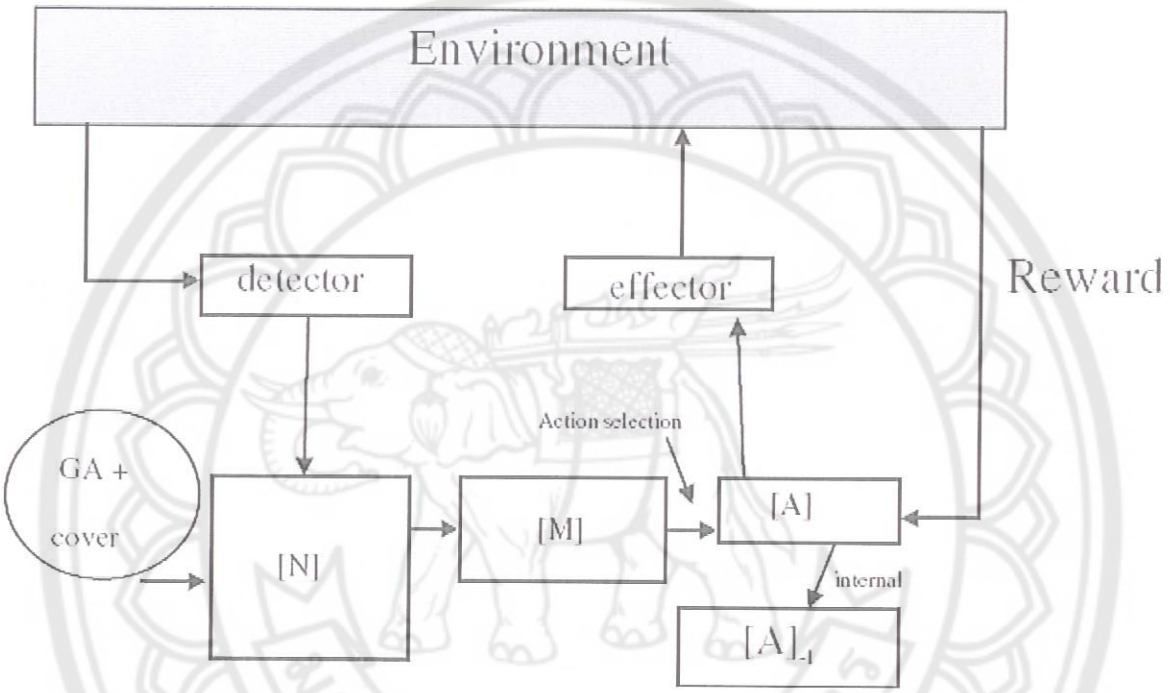


Figure 2.2 Schematic of ZCS.

When ZCS was first presented, results from its use indicated it was capable of good, but not optimal, performance. 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 Environment detector effector Reward parameters. It should be noted that ZCS has two closely related forerunners, namely BOOLE and NEWBOOLE.

2.4 Wilson's XCS

Figure. 2.3 gives an overall picture of the system, which is shown in interaction with an environment via detectors for sensory input and effectors for motor actions. In addition, the environment at times provides a scalar reinforcement, here termed reward. Many aspects of XCS are copied from ZCS, a "zeroth level" classifier system intended to simplify Holland's canonical framework while retaining the essence of the classifier system idea.

Some descriptive material is omitted here because it can be found in the ZCS paper. The differences between XCS and ZCS lie in the definition of classifier fitness, the GA mechanism, and the more sophisticated action selection that accuracy-based fitness makes possible.

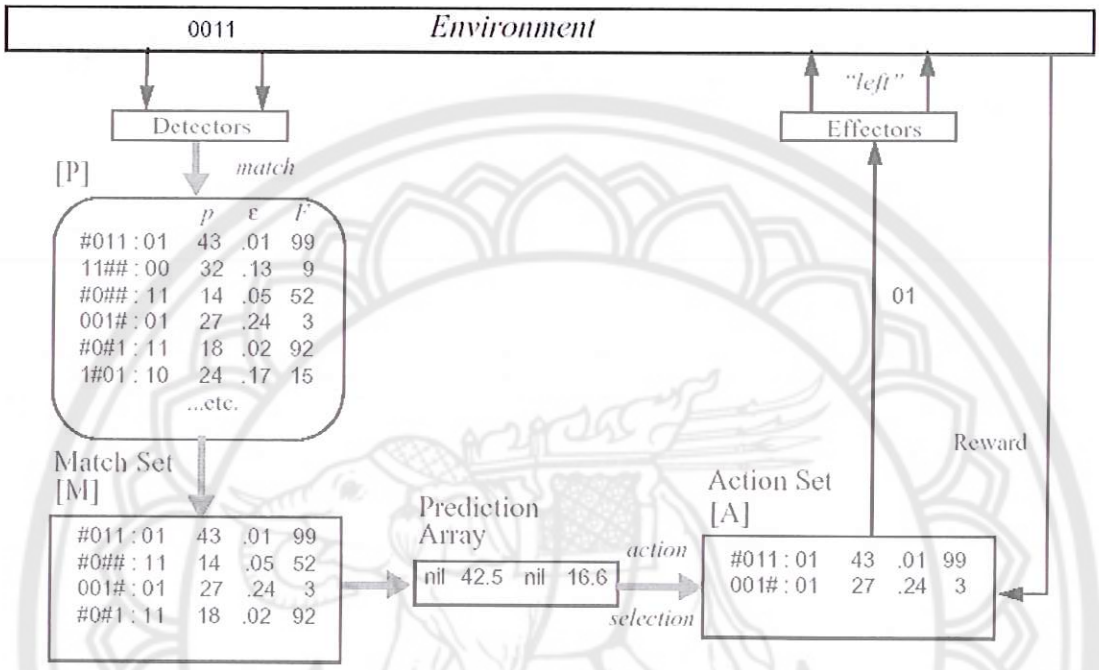


Figure 2.3 Schematic of XCS.

The box labeled [P] contains the classifier population, and shows some example classifiers. The left side of each classifier consists of a single condition; the right side codes an environmental action. Associated with each classifier are prediction, prediction error, and fitness parameters, symbolized by *p*, e, and *F*, respectively. The population has a fixed maximum size *N* and may be initialized in a variety of ways: with *N* randomly generated classifiers; with potentially useful "seed" classifiers; with no classifiers; or with one general (condition consisting of #'s) classifier for each action; etc. The initial values of *p*, e, and *F* can be set more or less arbitrarily; there is little effect on performance.

The most significant difference between XCS (Figure. 2.3) 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 accuracybased 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 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.

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 Environment detector effectors Reward 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 appropriate rule; 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:

$\varepsilon_j = \varepsilon_j + \beta(|P - pj| - \varepsilon_j)$ where as in ZCS $0 \leq \beta \leq 1$ is a learning rate constant.

ii) Rule predictions are then updated:

$pj = pj + \beta(P-pj)$

iii) Each rule's accuracy K_j is determined:

$K_j = \alpha(\varepsilon_0/\varepsilon)^{\nu}$ or $K=1$where $\varepsilon < \varepsilon_0$ where ν, α and ε_0 are constants controlling the shape of the accuracy function.

iv) A relative accuracy K_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 Fj using the moyenne adaptive modifee (MAM) procedure: If the fitness has been adjusted $1/\beta$ times, $Fj = Fj + \beta(K_j' - Fj)$. Otherwise Fj is set to the average of the values of K' 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 $P(ai)$ 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 in its reinforcement procedure, whereas Holland's 1986 system and ZCS both use a form of TD(0).

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. 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 . XCS has since shown good performance on data mining tasks and has been widely adopted in the LCS community. The majority of contributions to a recent volume on applications of LCS used XCS.

CHAPTER 3

Clustering with Learning Classifier Systems

3.1 Introduction

This chapter presents initial results from a rule-based approach to clustering through the development of an accuracy-based Learning Classifier System (LCS). A number of studies have indicated good performance for LCS in classification tasks. We are interested in the utility of such systems to perform unsupervised learning tasks.

Clustering is an important unsupervised classification technique where a set of data are grouped into clusters in such a way that data in the same cluster are similar in some sense and data in different clusters are dissimilar in the same sense. For this it is necessary to first define a measure of similarity which will establish a rule for assigning data to the domain of a particular cluster centre. One such measure of similarity may be the Euclidean distance D between two data x and y defined by $D=||x-y||$. Typically in data clustering there is no one perfect clustering solution of a dataset, but algorithms that seek to minimize the cluster spread, i.e., the family of centre-based clustering algorithms, are the most widely used (e.g., [Xu & Winch, 2005]). They each have their own mathematical objective function which defines how well a given clustering solution fits a given dataset. In this paper our system is compared to the most well-known of such approaches, the *k*-means algorithm. We use as a measure of the quality of each clustering solution the total of the *k*-means objective function:

$$o(X,C) = \sum_{i=1}^{n} \min_{j \in \{1 \ldots k\}} \| x_i - c_j \|^2$$

Define a *d*-dimensional set of n data points $X = \{x_1, \ldots, x_n\}$ as the data to be clustered and k centers $C = \{c_1, \ldots, c_k\}$ as the clustering solution. However most clustering algorithms require the user to provide the number of clusters (k), and the user in general has no idea about the number of clusters. Hence this typically results in the need to make several clustering trials with different values for k where $k = 2$ to k_{max} = square-root of n (data points) and select the best clustering among the partitioning with different number of clusters. The commonly applied Davies-Bouldin validity index is used as a guideline to the underlying number of clusters here.

Previously, evolutionary algorithms have been used for clustering in two principle ways. The first uses them to search for appropriate centers of clusters with established clustering algorithms such as the *k*-means algorithm, e.g., the GA-clustering algorithm.

However this approach typically requires the user to provide the number of clusters. Tseng and Yang [2001] proposed the CLUSTERING algorithm which has two stages. In the first stage a nearest-neighbor algorithm is used to reduce the size of data set and in the second the GA-clustering algorithm approach is used. Sarafis [2003] has recently proposed a further stage which uses a density-based merging operator to combine adjacent rules to identify the underlying clusters in the data. We suggest that modern accuracy-based LCS are well-suited to the clustering problem due to their generalization capabilities.

The chapter is structured as follows: first we describe the general scheme for using accuracy-based LCS for clustering and then present initial results. The adoption of a more sophisticated fitness function is found to be beneficial. A form of rule compaction for clustering with LCS, as opposed to classification, is then presented. A form of local search is then introduced before a number of increasingly difficult synthetic datasets are used to test the algorithm.

3.2 A Simple LCS for Clustering

In this chapter we begin by presenting a version of the simple accuracy-based YCS [Bull, 2005] which is derived from XCS [Wilson, 1995], here termed YCSc. YCSc is a Learning Classifier System without internal memory, where the rulebase consists of a number (*N*) of rules. Associated with each rule is a scalar which indicates the average error (e) in the rule's matching process and an estimate of the average size of the niches (match sets - see below) in which that rule participates (σ). The initial random population of rules have their parameters set to 10.

On receipt of an input data, the rulebase is scanned, and any rule whose condition matches the message at each position is tagged as a member of the current match set [M]. The rule representation here is the Centre-Spread encoding (see [Stone & Bull, 2003] for discussions). A condition consists of interval predicates of the form $\{\{c_1, s_1\}, \ldots\ldots \{c_d, s_d\}\}$, where c is the interval's range centre from [0.0,1.0] and s is the "spread" from that centre from the range $(0.0, s_0]$ and d is a number of dimensions. Each interval predicates' upper and lower bounds are calculated as follows: $[c_i - s_i, c_i + s_i]$. If an interval predicate goes outside the problem space bounds, it is truncated. A rule matches an input x with attributes x_i if and only if $c_i - s_i \leq x_i < c_i + s_i$ for all x_i.

Reinforcement in YCSc consists of updating the matching error e which is derived from the Euclidean distance with respect to the input x and c in the condition of each member of the current [M] using the Widrow-Hoff delta rule with learning rate β:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta(\ ((\sum_{l=1}^{d}(x_l - c_{lj})^2))^{1/2} - \ \varepsilon_j)$$

Next, the niche size estimate is updated:

$$\sigma_j \leftarrow \sigma_j + \beta(|[M]| - \sigma_j)$$

YCSc employs two discovery mechanisms, a niche genetic algorithm (GA) [Holland, 1975] and a covering operator. The general niche GA technique was introduced by Booker [1989], who based the trigger on a number of factors including the payoff prediction "consistency" of the rules in a given [M], to improve the performance of LCS. XCS uses a time-based mechanism under which each rule maintains a time-stamp of the last system cycle upon which it was consider by the GA. The GA is applied within the current niche when the average number of system cycles since the last GA in the set is over a threshold θ_{GA}. If this condition is met, the GA time-stamp of each rule in the niche is set to the current system time, two parents are chosen according to their fitness using standard roulette-wheel selection, and their offspring are potentially crossed and mutated, before being inserted into the rulebase. This mechanism is used here within match sets, as in the original XCS algorithm [Wilson, 1995], which was subsequently changed to work in action sets to aid generalization per action [Butz & Wilson, 2001].

The GA uses roulette wheel selection to determine two parent rules based on the inverse of their error:

$$f_i = \frac{1}{\varepsilon_i^{\upsilon} + 1}$$

Offspring are produced via mutation (probability μ) where, after [Wilson, 2000], we mutate an allele by adding an amount + or - $rand(m_0)$, where m_0 is a fixed real, rand picks a real number uniform randomly from $(0.0,m_0]$, and the sign is chosen uniform randomly. Crossover (probability χ, two-point) can occur between any two alleles, i.e., within an interval predicate as well as between predicates, inheriting the parents' parameter values or their average if crossover is invoked. Replacement of existing members of the rulebase uses roulette wheel selection based on estimated niche size. If no rules match on a given time step, then a covering operator is used which creates a rule with its condition centre on the input value and the spread with a range of $rand(s_0)$, which then replaces an existing member of the rulebase in the same way as the GA.

Recently, Butz et al. [2004] have proposed a number of interacting "pressures" within XCS. Their "set pressure" considers the more frequent reproduction opportunities of more general rules. Opposing the set pressure is the pressure due to fitness since it represses the reproduction of inaccurate overgeneral rules. Thus to produce an effective, i.e., general but

appropriately accurate, solution an accuracy-based LCS using a niche GA with global replacement should have these two pressures balanced through the setting of the associated parameters. In this chapter we show how the same mechanisms can be used within YCSc to identify clusters within a given dataset; the set pressure encourages the evolution of rules which cover many data points and the fitness pressure acts as a limit upon the separation of such data points, i.e., the error.

3.3 Initial Performance

In this section we apply XCSc as described above on two datasets for the first experiment to test the performance of the system. The first dataset is well-separated as shown in Fig 3.1(a). We use a randomly generated synthetic dataset. This dataset has k = 25 true clusters arranged in a 5x5 grid in d = 2 dimension. Each cluster is generated from 400 data points using a Gaussian distribution with a standard deviation of 0.02, for a total of n = 10,000 datum. The second dataset is not well-separated as shown in Fig 3.1(b). We generated it in the same way as the first dataset except the clusters are not centred on that of their given cell in the grid.

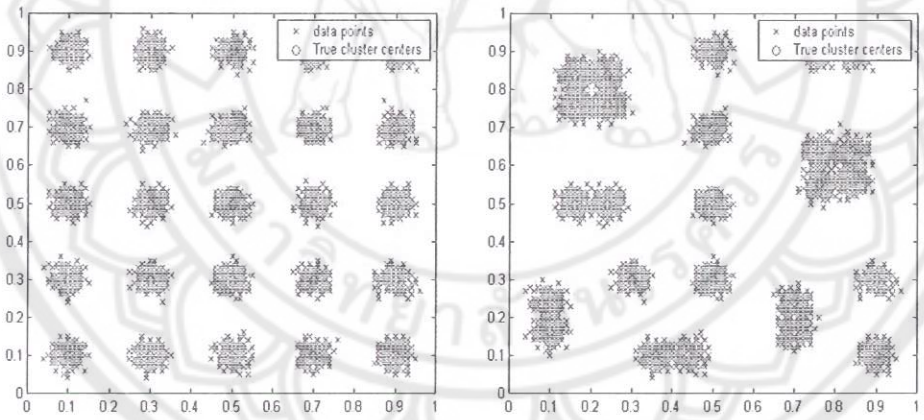


Figure 3.1: The well-separated (a) and less-separated (b) data sets used.

The parameters used were: N=800, β=0.2, ν=5, χ=0.8, μ =0.04, θ_{GA} =12, s_0 =0.03, m_0 =0.006. All results presented are the average of ten runs. Learning trials consisted of 200,000 presentations of a randomly sampled data point.

Figure 3.2 shows a typical example solutions produced by YCSc on both data sets. That is, the region of the 2D input space covered by each rule in the final rule-base is plotted along with the data. As can be seen, in the well-separated case the system roughly identifies all 25 clusters whereas in the less-separated case contiguous clusters are covered by the same rules.

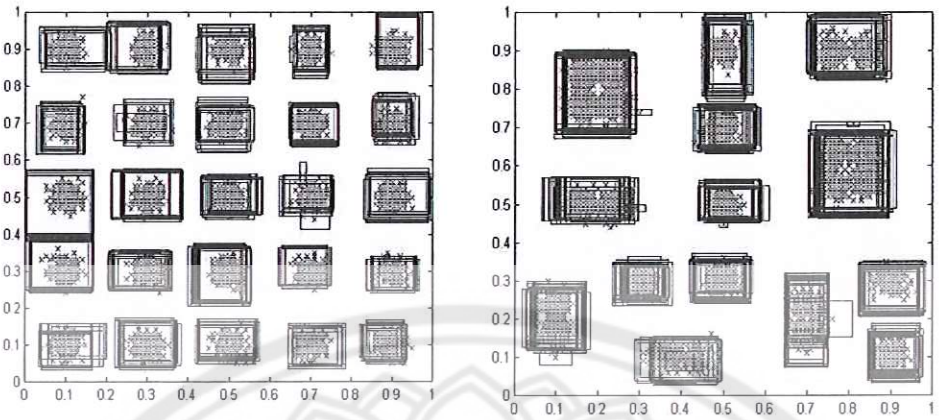


Figure 3.2 Typical solutions for the well-separated (a) and less-separated (b) data sets.

As expected, solutions contain many overlapping rules around each cluster. The next section presents a rule compaction algorithm which enables identification of the underlying clusters.

3.4 Rule Compaction

Wilson [2002] introduced a rule compaction algorithm for XCS to aid knowledge discovery during classification problems (see also [Fu & Davis, 2002][Dixon et al., 2003][Wyatt et al., 2004]). We have developed a compaction algorithm for clustering:

Step 1 Delete the useless rules: The useless rules are identified and then deleted from the ruleset in the population based on their coverage. Low coverage means that a rule matches a small fraction (20%) of the average coverage.

Step 2: Find the required rules from numerosity: The population $[P]_{N[deleted]}$ is sorted according to the numerosity of the rules and delete the rules that have lower numerosity, less than 2. Then $[P]_M$ ($M < N$) is formed by selecting the minimum sequential set of rules that covers all data (ignoring data not covered by a rule).

Step 3: Find the required rules from average error : The population $[P]_M$ is sorted according to the average error of the rules. Then $[P]_P$ ($P < M$) is formed by selecting the minimum sequential set of rules that covers all data (ignoring data not covered by a rule).

Step 4: Remove redundant rules: This step is an iterative process. On each cycle it selects the rule in $[P]_P$ that maximum number of match set. This rule is removed into the final ruleset $[P]_F$ and the data that it covers deleted from the dataset. The process continues until the dataset is empty (ignoring data not covered by a rule).

Figure 3.3 shows the final set $[P]_F$ for both the full solutions shown in Figure 2. YCSc's identification of the clusters is now clear. Under the (simplistic) assumption of non-overlapping regions as described by rules in $[P]_F$ it is easy to identify the clusters after compaction. In the case where no rules subsequently match new data we could of course identify a cluster by using the distance between it and the centre of each rule.

We have examined the average quality of the clustering solutions produced during the ten runs by measuring the total objective function described in equation (1) and checking the number of clusters defined. The average quality on the well-separated dataset is 8.12 +/- 0.54 and the number of clusters is 25 +/- 0. That is, it correctly identifies the number of clusters every time. The average quality on the not well-separated dataset is 24.50 +/- 0.56 and the number of clusters is 14 +/- 0. Hence it is not correct every time due to the lack of clear separation in the data.

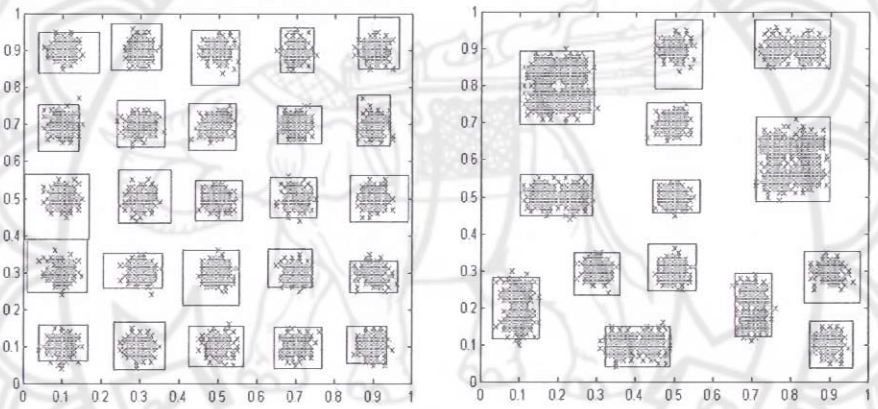


Figure 3.3 Showing the effects of the rule compaction on the typical solutions shown in Figure 3.2 for the well-separated (a) and less-separated (b) data sets.

For comparison, the *k*-means algorithm was applied to the datasets. The *k*-means algorithm (assigned with the known *k*=25 clusters) averaged over 10 runs gives a quality of 32.42 +/- 9.49 and 21.07 +/- 5.25 on the well-separated and less-separated datasets respectively. The low quality of solutions in the well-separated case is due to the choice of the initial centres; *k*-means is well-known for becoming less reliable as the number of underlying clusters increases. For estimating the number of clusters we ran, for 10 times each, different *k* (2 to 30) with different random initializations. To select the best clustering with different numbers of clusters, the Davies-Bouldin validity index is shown in Figure 4. The result on well-separated dataset has a lower negative peak at 23 clusters and the less-separated dataset has a lower negative peak at 14 clusters. That is, it is not correct on both

datasets, for the same reason as noted above regarding quality. Thus YCSc performs as well or better than *k*-means whilst also identifying the number of clusters during learning.

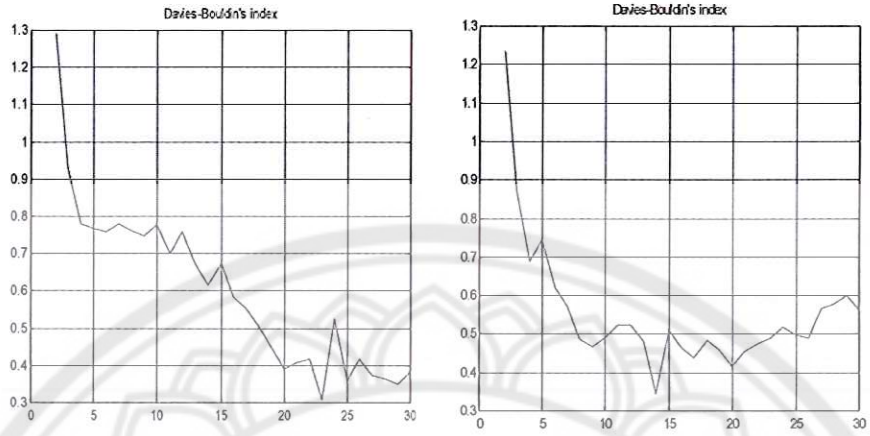


Figure 3.4: *K*-means algorithm performance using the Davies-Bouldin index for the well-separated (a) and less-separated (b) data sets.

3.5 Modifying XCS for Clustering

As noted above, YCS is a simplified version of XCS, presently primarily to aid understanding of how such accuracy-based LCS learn [Bull, 2005]. The principle difference is that fitness *F* is slightly more complex. First, the accuracy κ_j and the relative accuracy κ'_j are computed as

$$\kappa_j = \begin{cases} 1, \ldots\ldots\ldots\ldots\ldots\ldots if \varepsilon_j > \varepsilon_0 \\ \alpha\left(\dfrac{\varepsilon_j}{\varepsilon_0}\right)^{-\upsilon}, \ldots\ldots\ldots otherwise \end{cases}$$

$$\kappa'_j = \frac{\kappa_j}{\sum_{j=[M]} \kappa_j}$$

The parameter $\varepsilon_0 (\varepsilon_0 > 0)$ controls the tolerance for rule error (ε); the parameter $\alpha (0 < \alpha < 1)$ and the parameter $\upsilon (\upsilon > 0)$ are constants controlling the rate of decline in accuracy κ when ε_0 is exceeded. Finally, fitness *F* is updated toward the current relative accuracy as follows:

$$F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$$

The reader is referred to [Butz & Wilson, 2001] for a full algorithmic description of XCS.

Using the same parameters as above, with ε_0 = 0.03 and □=0.1, we have examined the average quality of the clustering solutions produced during the ten runs by measuring the total objective function described in equation (1) and checking the number of clusters defined. The average of quality on the well-separated dataset is 6.65 +/- 0.12 and the number of clusters is 25.0 +/- 0. The average quality on the not well-separated dataset is 6.71 +/- 0.14 and the number of clusters is 25.0 +/- 0. That is, it correctly identifies the number of clusters every time. Thus XCSc performs better than both YCSc and *k*-means whilst also identifying the number of clusters during learning. That is, YCSc struggled with the less-separated data and analysis of solutions indicates that the difference in error between more appropriate descriptions of the underlying clusters and those typically promoted is very small, which are not sufficiently amplified under the fitness scaling of equation 4. The function of XCS therefore seems more appropriate for such problems (note no difference was seen for a number of classification tasks [Bull, 2005]).

3.6 Local Search

Previously, Wyatt and Bull [2004] have introduced the use of local search within XCS for continuous-valued problem spaces. Within the classification domain, they used the Widrow-Hoff delta rule to adjust rule condition interval boundaries towards those of the fittest rule within each niche on each matching cycle, reporting significant improvements in performance. Here good rules serve as a basin of attraction under gradient descent search thereby complimenting the GA search. The same concept has also been applied to a neural rule representation scheme in XCS [O'Hara & Bull, 2005]. We have examined the performance of local search for clustering using Wyatt and Bull's scheme: once a focal rule (the highest fitness rule) has been identified from the current match set all rules in [M] use the Widrow-Hoff update procedure to adjust each of the two interval descriptor pairs towards those of the focal rule, e.g., $c_{ij} < -c_{ij} + \beta_l[F_j - c_{ij}], \forall i, j,$ where c_{ij} represents gene j of rule i in the match set, F_j represent gene j of the focal rule, and □$_l$ is a learning set to 0.1. The spread parameters are adjusted in the same way and the mechanism is applied on every match cycle before the GA trigger is tested. Initial results using Wyatt and Bull's scheme gave a reduction in performance, typically more specific rules, i.e., too many clusters, were identified (not shown).

We here introduce a scheme which uses the current data as the target for the local learning to adjust only the centres of the rules:

$$c_{ij} < -c_{ij} + \beta_l (x_j - c_{ij})$$

Where c_{ij} represents the centre of gene j of rule i in the current match set, x_j represents the value in dimension j of the current input data, and $\square_l$ is the learning rate, here set to 0.1. This is applied on every match cycle before the GA trigger is tested, as before. In the well-separated case, the quality of solutions was 6.50 +/- 0.09. In the less-separated case, the quality of solutions was 6.48 +/- 0.07. The same number of clusters was identified as before, i.e., 25 and 25 respectively. Thus results indicate that our data-driven local search improves the quality of the clustering over the non-local search approach and is used hereafter. The same was found for YCSc but it does not improve the cluster identification [Tamee et al., 2006].

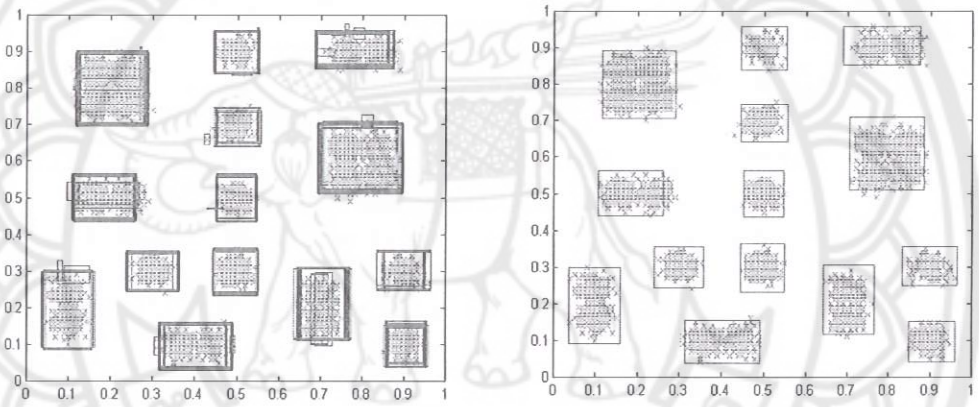


Figure 3.5 Typical solutions using ε_0 =0.1 before (a) and after (b) rule compaction, for the less-separated dataset.

3.7 Adaptive Threshold Parameter

The ε_0 parameter controls the error threshold of rules and we have investigated the sensitivity of XCSc to its value by varying it. Experiments show that, if ε_0 is set high, e.g., 0.1, in the less-separated case the contiguous clusters are covered by the same rules (Figure 3.5). We therefore developed an adaptive threshold parameter scheme which uses the average error of the current [M]:

$$\varepsilon_0 = 1.2 * (\sum \varepsilon_j / N_{[M]})$$

Where ε_j is the average error of each rule in the current match set and $N_{[M]}$ is the number of rules in the current match set. This is applied before the fitness function calculations.

Figure 3.6 shows how in the well-separated case, the average quality and number of clusters from 10 runs is as before, being 6.39 +/- 0.04 and 25.0 +/- 0 respectively. In the less-separated case the average quality is again almost unchanged at 6.40 +/- 0.09 and the number of clusters is 25.0 +/- 0. There are no significant differences in average quality but with the adaptive technique there is a reduction in the number of parameters that require careful, possibly problem specific, setting by the user.

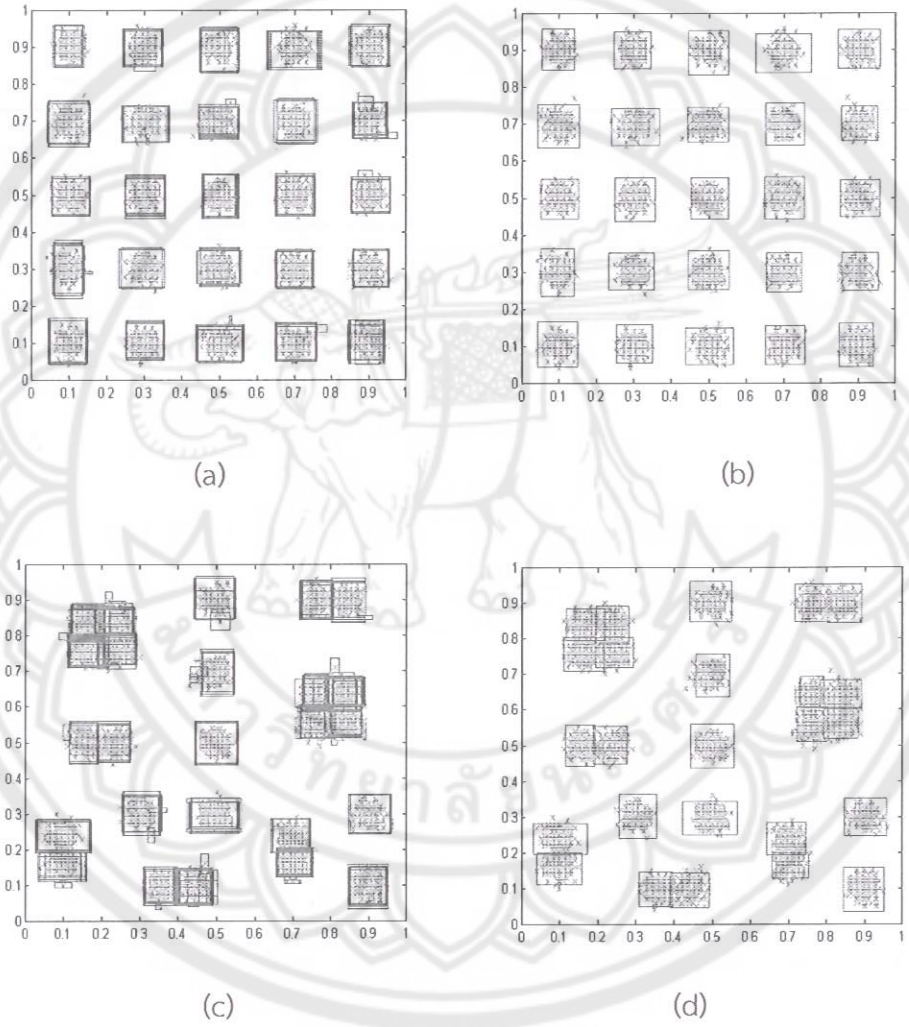


Figure 3.6 Typical solutions using adaptive ε_0 approach before and after rule compaction, for well-separated (a-b) and less-separated (c-d) dataset

CHAPTER 4

Ellipsoidal Condition in Clustering with XCS

4.1 Clustering with XCS

In this section we present a version of the accuracy-based XCS, here termed XCSc. XCSc is a Learning Classifier System without internal memory, where the rulebase consists of a number (N) of rules. Associated with each rule is a scalar which indicates the average error (ε) in the rule's matching process and the fitness (F) estimates the accuracy of the average error and an estimate of the average size of the niches (match sets - see below) in which that rule participates (σ).

On receipt of an input data, the rulebase is scanned, and any rule whose condition matches the message at each position is tagged as a member of the current match set [M]. The rule representation here is the Centre-Spread encoding (see [Stone & Bull, 2003] for discussions). A condition consists of interval predicates of the form $\{\{c_1, s_1\}, \ldots\ldots \{c_d, s_d\}\}$, where c is the interval's range centre from [0.0,1.0] and s is the "spread" from that centre from the range $(0.0, s_0]$ (Note that, we start with a fairly general population due to the high value s_0 = 0.5 throughout the experiment) and d is a number of dimensions. Each interval predicates' upper and lower bounds are calculated as follows: $[c_i - s_i, c_i + s_i]$. If an interval predicate goes outside the problem space bounds, it is truncated. A rule matches an input x with attributes x_i if and only if $c_i - s_i <= x_i < c_i + s_i$ for all x_i.

Reinforcement in XCSc consists of updating the matching error ε which is derived from the Euclidean distance with respect to the input x and c in the condition of each member of the current [M] using the Widrow-Hoff delta rule with learning rate β:

$$\varepsilon_j <- \varepsilon_j + \beta(((\sum_{l=1}^{d}(x_l - c_{lj})^2))^{1/2} - \varepsilon_j) \qquad (1)$$

Next, the niche size estimate is update:

$$\sigma_j <- \sigma_j + \beta(|[M]| - \sigma_j) \qquad (2)$$

The rest of the fitness update follows that of standard XCS with parameter ε_0, α and υ (The reader is referred to [Butz & Wilson, 2001] for a full algorithmic description of XCS.)

XCSc employs two discovery mechanisms, a niche genetic algorithm (GA) and a covering operator. The general niche GA technique was introduced by Booker [1989], who based the trigger on a number of factors including the payoff prediction "consistency" of the rules in a given [M], to improve the performance of LCS. XCS uses a time-based mechanism under which each rule maintains a time-stamp of the last system cycle upon which it was considered by the GA. The GA is applied within the current niche when the average number of system cycles since the last GA in the set is over a threshold ϑ_{GA}. If this condition is met, the GA time-stamp of each rule in the niche is set to the current system time, two parents are chosen according to their fitness using standard roulette-wheel selection, and their offspring are potentially crossed and mutated, before being inserted into the rulebase. This mechanism is used here within match sets, as in the original XCS algorithm [Wilson, 1995], which was subsequently changed to work in action sets to aid generalization per action [Butz & Wilson, 2001].

Offspring are produced via mutation (probability μ) where, after [Butz, 2005], we use a relative real-valued mutation instead of the fixed interval mutation. If an attribute of the center is mutated, we mutate it by adding an amount + or - $rand(mc_0)$, where mc_0 is a relative real-valued, rand pick a real number uniform randomly from zero to its current spread. If an attribute of the spread is mutated, we mutate it by adding an amount + or - $rand(ms_0)$, where ms_0 is a relative real-valued, rand pick a real number uniform randomly from zero to fifty percent of the current spread. The sign is chosen uniform randomly. Crossover (probability χ, two-point) can occur between any two alleles, i.e., within an interval predicate as well as between predicates, inheriting the parents' parameter values or their average if crossover is invoked. Replacement of existing members of the rulebase uses roulette wheel selection based on estimated niche size (if its fitness F is significantly lower than the average fitness of rules in [P], its deletion probability is further increased as in XCS). If no rules match on a given time step, then a covering operator is used which creates a rule with its condition centre on the input value and the spread with a range of $rand(s_0)$, which then replaces an existing member of the rulebase in the usual way (see [Butz & Wilson, 2001]).

We apply XCSc as described above on datasets for the first experiment to test the performance of the system as shown in Figure 4.1(a). The parameters used were: N = 400, β = 0.2, v = 5, α = 0.1, χ = 0.8, μ = 0.04, ϑ_{GA} = 50, ε_0 = 0.03. Figure 4.1(b) shows typical solutions produced by XCSc using local search mechanism and adaptive ε_0 scheme (see

[Tamee et al., 2007] for details) on example dataset. That is, the region of the *2D* input space covered by each rule (represented by hyperrectangle) in the final rule-base is plotted along with the data. As can be seen, the system roughly identifies all 25 clusters.

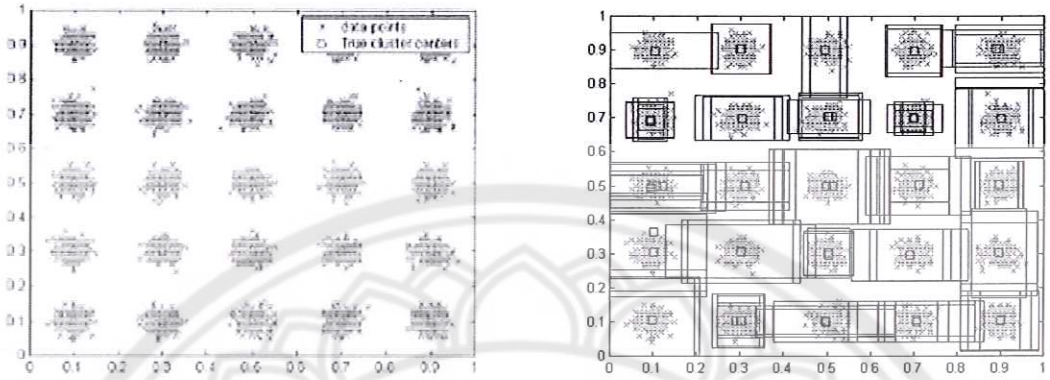


Figure 4.1 The dataset used (a) and Typical solutions for the dataset(b).

As expected, solutions contain many overlapping rules around each cluster. The rule compaction algorithm which enables identification of the underlying clusters (see [Tamee et al., 2007] for details). Figure 4.2 shows the final set $[P]_F$ for the full solutions shown in Figure 1. XCSc's identification of the clusters is now clear.

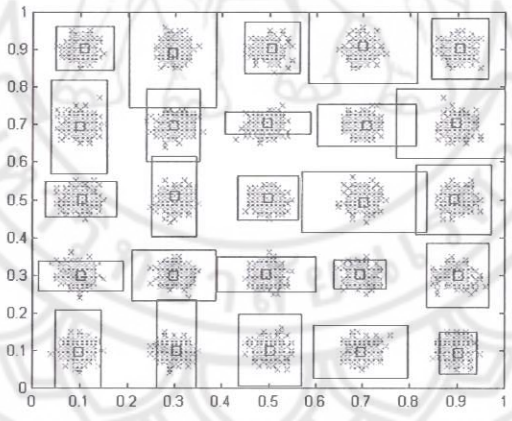


Figure 4.2 Effects of compaction on the typical solutions in Figure 2 for dataset.

4.2 Ellipsoidal Condition in Clustering with XCS

Here in this section, the conditions are changed to hyperellipsoidal. A condition consists of a centre point and a deviation, that is, $C = (c_1, c_2, \ldots, c_d, r_1, r_2, \ldots, r_d)$, where c is the interval's range centre from [0.0,1.0] and r is the radii or deviation from that centre from the range $(0.0, r_0]$ (Note that we start with a fairly general population due to the high value $r_0 = 0.5$ throughout the experiment) and d is a number of dimention. Build match set [*M*] from [*P*] whose conditions are cover the current input. A rule is match, if its activity *r.ac* more than

threshold ϑ_{ac}. We use the Gaussian kernel function to determine the activity of a rule given the current input x is defined as:

$$r.ac = \exp(-\sum \frac{(x_i - c_i)^2}{2r_i^2})$$

effectively dividing in each dimension the squared distance from the center by twice the variance in that dimension. In a sense, ϑ_{ac} is the pendant to r. Since r is evolved, ϑ_{ac} can be fixed. It is set to 0.7 throughout the experiment.

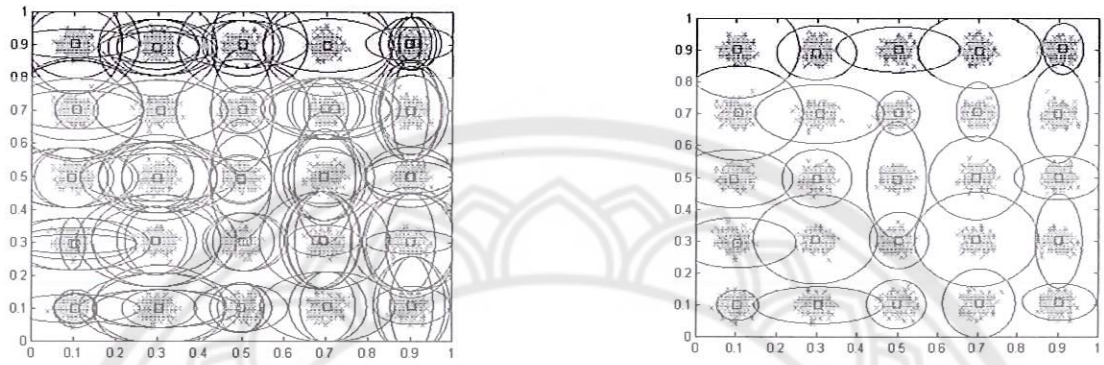
Hereby, we use the usual reinforcement update rule parameters in the match set and discovery mechanisms as XCSc with hyperrectangle. Covering and mutation need to be adjusted. When the covering occurs, creates a rule with its condition centre on the input value and the deviation with a range of $rand(r_0)$. For mutation, we use a relative real-valued mutation. If an attribute of the center is mutated, we mutate it by adding an amount + or - $rand(mc_0)$, where mc_0 is a relative real-valued, rand pick a real number uniform randomly from zero to its current spread. If an attribute of the deviation is mutated, we mutate it by adding an amount + or - $rand(mr_0)$, where mr_0 is a relative real-valued, rand pick a real number uniform randomly from zero to fifty percent of the current deviation. The sign is chosen uniform randomly. If any r_i goes outside the deviation necessary to contain the whole problem demension, it is set to that value, that is:

$$\forall i : r_i \leq \frac{u_i^* - l_i^*}{\sqrt{-2\log\theta_{ac}}}$$

Where u_i^* and l_i^* denote the maximum upper and lower values of each dimension. As crossover operator can occur between any two alleles.

The parameters used were as before: N = 400, β = 0.2, ν = 5, α = 0.1, χ = 0.8, μ = 0.04, ϑ_{GA} = 50. Figure 3 shows typical solutions produced by XCSc with hyperellipsoids using local search mechanism and adaptive ε_0 scheme as before on example dataset. That is, the 2D input space coveraed by each rule (represented by hyperellipsoids) in the final rule-base before rule copaction as show in Fig 4.3(a) and after rule compaction as show in Fig 4.3(b). As can be seen, in the example dataset case the XCSc with hyperellipsoids identify all 25 clusters as same as XCSc with hyperrectangle. The average quality and number of clusters from 10 runs as before, being 0.9928 +/- 0.01 and 25.0 +/- 0 respectively. The same number of clusters and there are no significant differences in average quality were identified as XCSc

with hyperrectangle. However, the XCSc with hyperellipsoids have more general codition structures than XCSc with hyperrectangle. So, we expect that rules which more general codition structures can evole further facilitate, consequent rules in population well fit to each cluster in dataset.



(a) (b)

Figure 4.3 Typical solution XCSc with hyperellipsoidal before (a) and after (b) rule compaction for the dataset

4.3 Increased Complexity

Here we examine the performance of XCSc on both condition structures compared to *k*-means for nine synthetic datasets over randomly generated datasets in two-dimensions with varying data properties such as overlap between clusters, unequally density clusters and elongated cluster shapes. These data sets generators are available from http://www.dbk.ch.umist.ac.uk/handl/generators/.

The parameters used were as before: N = 400, β = 0.2, ν = 5, α = 0.1, χ = 0.8, μ = 0.04, ϑ_{GA} = 50. Fig. 4.4 and Fig. 4.5 show the results of a typical run of XCSc with hyperrectangle and XCSc with hyperellipsoids in the varying datasets respectively. Tables 1 show the average quality and standard deviation of clustering and the average number of clusters from XCSc with hyperrectangle, XCSc with hyperellipsoids and *k*-means from 10 runs as before. The statistical significance at a significance level of 0.05 are denoted by ● if significance improvement of XCS with hyperellipsoids with respect to another method, and by O if significance degradation of XCS with hyperellipsoids with respect to another method.

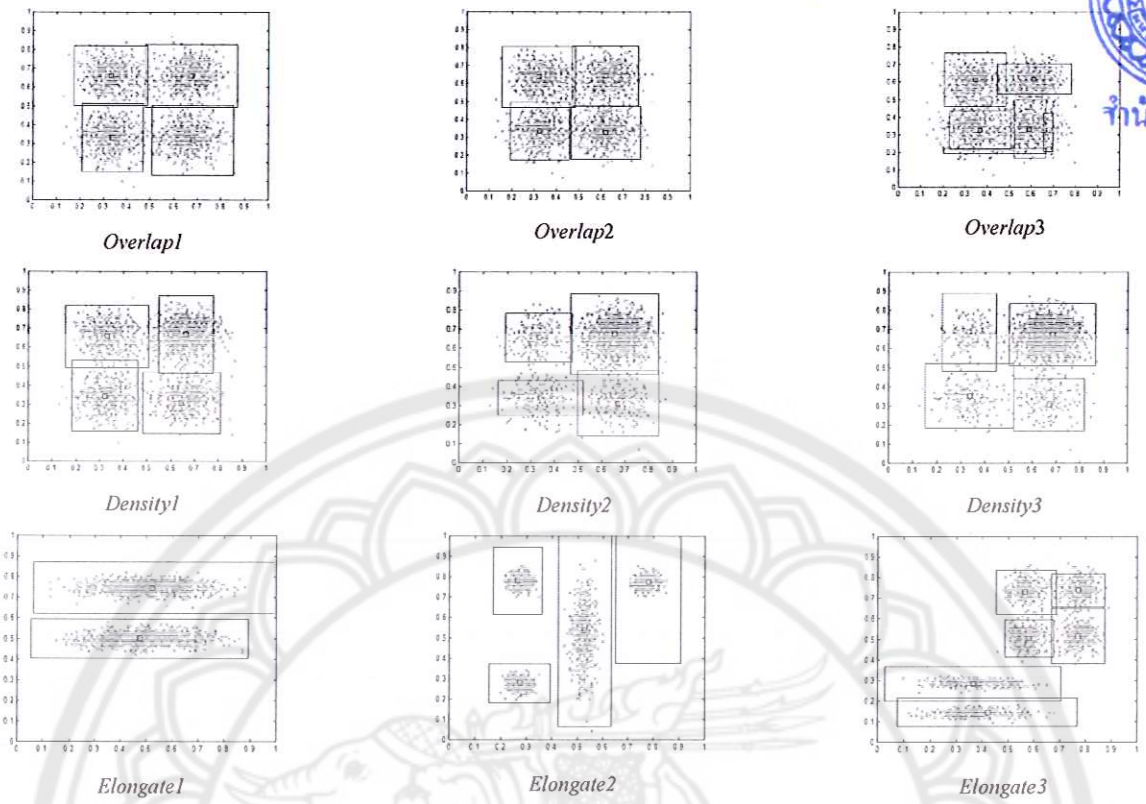


Figure 4.4 Typical solution XCSc with hyperrectangle after rule compaction for nine synthetic datasets

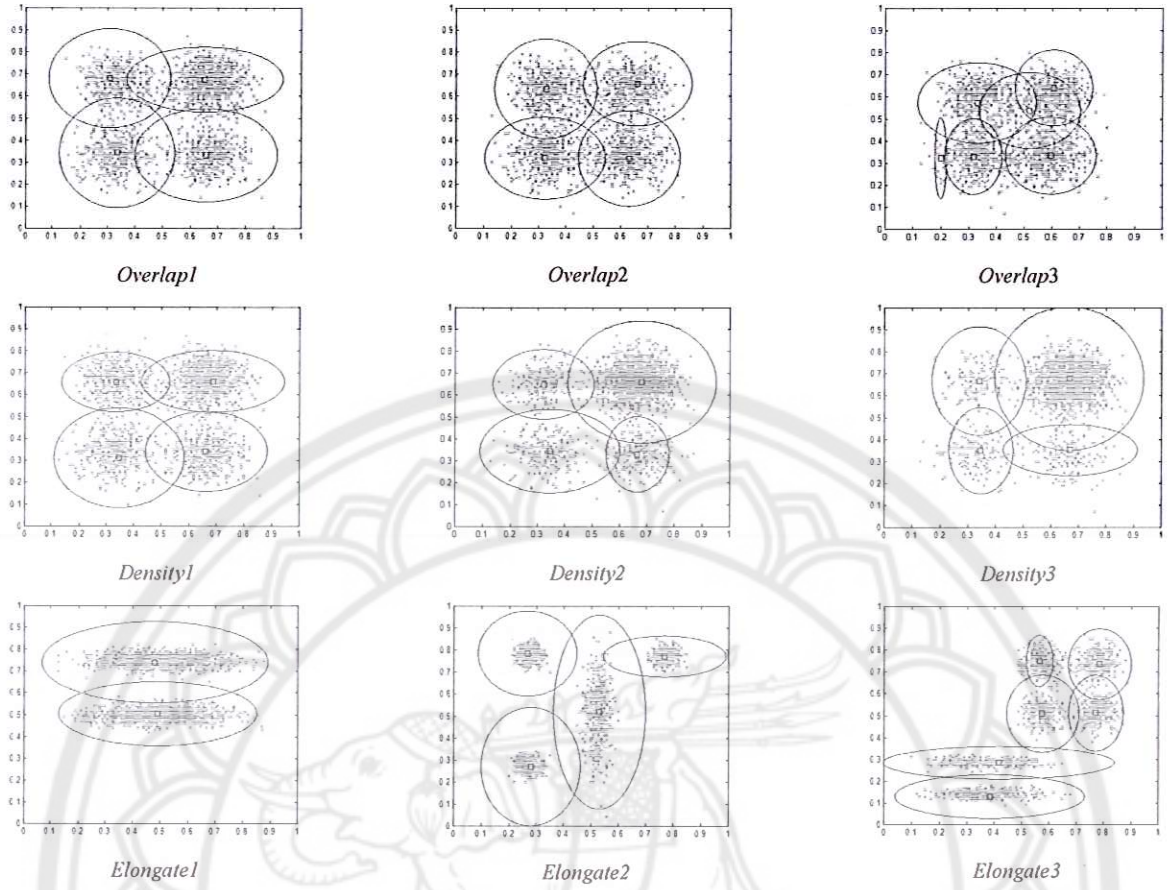


Figure 4.5 Typical solution XCSc with hyperellipsoidal after rule compaction for nine synthetic datasets

In particular, the XCSc with hyperellipsoids show always gives superior quality and gives an equivalent or closer estimate of the number of clusters compared to XCSc with hyperrectangle. Nevertheless, XCSc with hyperrectangle and XCSc with hyperellipsoids show good performance not better than k-means on all nine synthetic datasets. Espcially, the less-separated dataset (see Overlap3 in Tables 4.1), we hypothesize that adaptive ε_0 technique can not calculated the correct value for clusters that not well-separable from the other clusters. The parameter ε_0 calculated from the average error of the current [M] at each time before the fitness function is calculated. For the linearly separable cluster from the other clusters, under a genetic algorithm, the specific rules tend to cover data around the cluster and the over-general rules tend to be removed from the population. After a period of time, the rules in [M] should not difference too much. So, the adaptive ε_0 technique can calculated the correct value for this cluster. Under the assumption of genetic algorithm for non-linearly separable cluster from the other clusters tend to be as above. But, after a period of time, the rules in [M] change all the time because some rules in [M] and another [M] that closely are sameness. So, the adaptive ε_0 technique can not calculated the correct value for clusters that non-linearly separable.

The most striking result, however, is that XCSc with hyperellipsoids performs the most robustly across the range of different datasets. On those datasets containing spherically shaped cluster, it does not perform quite as well as the k-means, but its performance is still high and always gives superior to the data set containing elongated cluster shapes . Its performance breaks down only for the less-separated cases.

Table 4.1: XCSc with hyperrectangle and XCSc with hyperellipsoids and k-means on varying datasets.

dataset	*k*-means		XCSc with hyperrectangle		XCSc with hyperellipsoids	
	k	ARI	k	ARI	k	ARI
Overlap1	4.00	0.9614+/-0.00○	4.00	0.9271+/-0.08●	4.00	0.9495+/-0.01
Overlap2	4.00	0.9345+/-0.00○	4.00	0.9147+/-0.01	4.00	0.9150+/-0.02
Overlap3	4.00	0.8782+/-0.00○	4.10	0.5949+/-0.29	7.00	0.6040+/-0.23
Density1	4.00	0.9637+/-0.00	4.00	0.9557+/-0.01	4.00	0.9565+/-0.01
Density2	3.80	0.9565+/-0.03	4.00	0.9415+/-0.05●	4.00	0.9571+/-0.02
Density3	3.50	0.9499+/-0.01●	4.00	0.9617+/-0.01	4.00	0.9658+/-0.01
Elongate1	3.00	0.7074+/-0.04●	2.00	0.9960+/-0.01	2.00	0.9984+/-0.00
Elongate2	4.20	0.9512+/-0.01●	4.00	0.9818+/-0.04●	4.00	0.9961+/-0.00
Elongate3	7.50	0.8529+/-0.06●	5.70	0.9309+/-0.08	5.80	0.9394+/-0.06

CHAPTER 5

Conclusions

Our experiments clearly show how a new clustering technique based on the accuracy-based learning classifier system can be effective at finding clusters of high quality whilst automatically finding the number of clusters. That is, XCSc, with its more sophisticated fitness function, when adapted slightly, appears able to reliably evolve an optimal population of rules through the use of reinforcement learning to update rule parameters and a genetic algorithm to evolve generalizations over the space of possible clusters in dataset. The compaction algorithm presented reduces the number of rules in the total population to identify the rules that provide the clustering. Lastly, we are changed the condition of rules to hyperellipsoidal which more general condition structures than hyperrectangle; results show that the XCSc with hyperellipsoidal performs the most robustly across the range of different datasets.

References

[1] Bandyopadhyay, S. & Maulik, U. (2002) Genetic clustering for automatic evolution of clusters and application to image classification, Pattern Recognition 35 1197–1208.

[2] Booker, L.B. (1989) Triggered Rule Discovery in Classifier Systems. In J.D. Schaffer (ed) Proceeding of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, pp265-274

[3] Bull, L. (2004)(ed.) Applications of Learning Classifier Systems. Springer.

[4] Bull, L. (2005) Two Simple Learning Classifier Systems. In L. Bull & T. Kovacs (eds) Foundations of Learning Classifier Systems. Springer.

[5] Butz, M. and Wilson, S. (2001) An algorithmic description of XCS. In Lanzi, P. L., Stolzmann, W., and S. W. Wilson (eds.), Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000). Springer, pp253-272.

[6] Butz, M., Kovacs, T., Lanzi, P-L & Wilson, S.W. (2004) Toward a Theory of Generalization and Learning in XCS. IEEE Transactions on Evolutionary Computation 8(1): 28-46

[7] Butz, M.V., (2005) Kernal-based, Ellipsoidal Conditions in the Real-Valued XCS Classifier System. Proceedings of GECCO 2005. Washington, DC, USA.

[8] Davies, D. L. & Bouldin, D. W. (1979) A Cluster Separation Measure. IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. PAMI-1 (2): 224-227.

[9] Dixon, P., Corne, D., Oates, M. (2003) A Ruleset Reduction Algorithm for the XCS Learning Classifier System. In Lanzi, Stolzmann & Wilson (eds.), Proceedings of the 5th International Workshop on Learning Classifier Systems. Springer, pp.20-29.

[10]Frigui, H. & Krishnapuram, R. (1997) Clustering by competitive agglomeration. Pattern Recognition, vol. 30, no. 7, pp. 1109-1 119.

[11]Fu, C. & Davis, L. (2002). A Modified Classifier System Compaction Algorithm. In Banzhaff et al. (eds.) Proceedings of GECCO 2002. Morgan Kaufmann, pp 920-925.

[12]Ghosh, J., Strehl A., and Merugu, S. (2002) A consensus framework for integrating distributed clustering under limited knowledge sharing. in Proc. NSFWorkshop Next Generation Data Mining, pp. 99–108.

[13]Holland, J.H. (1975) Adaptation in Natural and Artificial Systems. Univ. of Michigan Press.

[14]Holland, J.H. (1976) Adaptation. In Rosen & Snell (eds) Progress in Theoretical Biology, 4. Plenum

[15]Jolion, J.M., Meer, P. & Bataouche, S. (1991) Robust clustering with applications in computer vision," IEEE Transactions on Panem Analysis and Machine Intelligence, vol. 13, no. 8, pp. 791-802.

[16]Krishna, K. & Murty, M.N. (1999) Genetic K-Means Algorithm, IEEE Trans. Syst. Man Cybern. Part B 29 433–439.

[17]Krishnapuram , R. & Freg, C. P. (1992) Fitting an unknown number of lines and planes to image data through compatible cluster merging. Pattern Recognition, vol. 25, no. 4.

[18]Maulik, U. & Bandyopadhyay, S. (2000) Genetic algorithm-based clustering technique. Pattern Recognition 33 1455-1465.

[19]Murthy, C.A. & Chowdhury, N. (1996) In search of optimal clusters using genetic algorithms, Pattern Recognition Lett. 7 825–832.

[20]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.

[21]Rand,W. (1971) Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc., 66, 846–850.

[22]Stone, C. and Bull, L. (2003) For real! XCS with continuous-valued inputs. Evolutionary Computation, 11(3):299–336.

[23]Sutton, R. & Barto, R. (1998) Reinforcement Learning. MIT Press.

[24]Tamee, K., Bull, L. & Ouen, P. (2008) Towards Clustering with Learning Classifier Systems. In L. Bull, E. Bernado-Mansilla & J. Holmes (eds) Learning Classifier Systems in Data Mining. Springer

[25]Tamee, K., Bull, L. & Ouen, P. (2007) Towards Clustering with XCS. In D. Thierens et al. (eds) GECOO-2007: Proceedings of the

[26] Genetic and Evolutionary Computation Conference. ACM Press, pp1854-1860.

[27]Tamee, K., Bull, L. & Ouen, P. (2006) A Learning Classifier System Approach to Clustering. In Proceedings of the 6th International Conference on Intelligent Systems Design and Applications. IEEE, pp621-626. .

[28]Tseng, L. Y. and Yang, S. B. (2001) A genetic approach to the automatic clustering problem. Pattern Recognition 34, pages 415-424.

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

[30]Wilson, S. W. (2000) Get real! XCS with continuous-valued inputs. In P. L. Lanzi, W. Stolzmann and S. W. Wilson (eds.) Learning Classifier Systems. From Foundations to Applications. Springer, pages 209–219.

[31]Wilson, S.W. (2001) Mining oblique data with XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, Proceedings of the Third International Workshop (IWLCS-2000), Lecture Notes in Artificial Intelligence, pages 158-174.

[32]Wilson, S. (2002). Compact Rulesets from XCSI. In Lanzi, Stolzmann & Wilson (eds.), Proceedings of the 4th International Workshop on Learning Classifier Systems. Springer, pp. 197-210.

[33]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.

[34]Wyatt, D., Bull, L. & Parmee, I. (2004) Building Compact Rulesets for Describing Continuous-Valued Problem Spaces Using a Learning Classifier System. In I. Parmee (ed) Adaptive Computing in Design and Manufacture VI. Springer, pp235-248.

[35]Xu, R & Winch, D. (2005) Survey of Clustering Algorithms. IEEE Transactions on neural networks 16 (3).Holmes, J. (2000). Learning Classifier Systems Applied to Knowledge Discovery in Clinical Research Databases. In: P. Lanzi, W. Stolzmann, S. W. Wilson (eds). Learning Classifier Systems: From Foundations to pplications. Springer-Verlag Heidelberg, pp 243-261.

[36]Holmes, J. (1997). Discovering risk of disease with a learning classifier system. In: T. Baeck (ed.). Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97). Morgan Kaufmann, San Francisco, CA.

[37]Stone, C. and Bull, L. (2003). For real! XCS with continuous valued inputs. Evolutionary Computation 11(3): pp. 299–336.

[38]Butz, M. and Wilson, S.W. (2001). An algorithmic description of XCS. In Lanzi, P. L., Stolzmann, W., and S. W. Wilson (eds.), Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000). Springer, pp. 253-272.

[39]Butz, M. V., Kovacs, T., Lanzi, P-L, and Wilson, S.W. (2004). Toward a Theory of Generalization and Learning in XCS. IEEE Transactions on Evolutionary Computation 8(1): pp. 28-46.

[40]Holland, J.H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: R. Michalski, J. Carbonell, T. Mitchell (eds). Machine Learning: An Artificial Intelligence Approach. Morgan Kaufmann, San Francisco, CA.

[41]Booker, L.B. (1989). Triggered Rule Discovery in Classifier Systems. In J.D. Schaffer (ed) Proceeding of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, pp. 265-274.

[42]Butz, M. and Wilson, S.W. (2001). An algorithmic description of XCS. In Lanzi, P. L., Stolzmann, W., and S. W. Wilson (eds.), Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000). Springer, pp. 253-272.

[43]Wilson, S.W. (2000) Get real! XCS with continuous-valued inputs. In P. L. Lanzi, W. Stolzmann and S. W. Wilson (eds.) Learning Classifier Systems. From Foundations to Applications. Springer, pp209–219.

[44]Dixon, P. W., Corne, D., and Oates, M. J. (2001). A Preliminary Investigation of Modified XCS as a Generic Data Mining Tool, Advances in Learning Classifier Systems: 4th International Workshop, IWLCS, Springer-Verlag Berlin Heidelberg, pp. 133-150.

[45]Wilson, S.W. (2001). Compact Rulesets from XCS. Fourth International Workshop on Learning Classifier Systems (IWLCS-2001). San Francisco, CA, pp. 197-210.