A Scalable Evolutionary Learning Classifier System for Knowledge Discovery in Stream Data Mining

Hai Huong Dam

M.Sci. University of Western Australia, Australia

B.Sci. (Hons) Curtin University of Technology, Australia

A thesis submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy at the

School of Information Technology & Electrical Engineering

University of New South Wales

Australian Defence Force Academy

Abstract

Data mining (DM) is the process of finding patterns and relationships in databases. The breakthrough in computer technologies triggered a massive growth in data collected and maintained by organisations. In many applications, these data arrive continuously in large volumes as a sequence of instances known as a data stream. Mining these data is known as stream data mining. Due to the large amount of data arriving in a data stream, each record is normally expected to be processed only once. Moreover, this process can be carried out on different sites in the organisation simultaneously making the problem distributed in nature. Distributed stream data mining poses many challenges to the data mining community including scalability and coping with changes in the underlying concept over time.

In this thesis, the author hypothesizes that learning classifier systems (LCSs) - a class of classification algorithms - have the potential to work efficiently in distributed stream data mining. LCSs are an incremental learner, and being evolutionary based they are inherently adaptive. However, they suffer from two main drawbacks that hinder their use as fast data mining algorithms. First, they require a large population size, which slows down the processing of arriving instances. Second, they require a large number of parameter settings, some of them are very sensitive to the nature of the learning problem. As a result, it becomes difficult to choose a right setup for totally unknown problems.

The aim of this thesis is to attack these two problems in LCS, with a specific focus on UCS - a supervised evolutionary learning classifier system. UCS is chosen as it has been tested extensively on classification tasks and it is the supervised version of XCS, a state of the art LCS.

In this thesis, the architectural design for a distributed stream data mining system will be first introduced. The problems that UCS should face in a distributed data stream task are confirmed through a large number of experiments with UCS and the proposed architectural design.

To overcome the problem of large population sizes, the idea of using a Neural Network to represent the action in UCS is proposed. This new system - called NLCS – was validated experimentally using a small fixed population size and has shown a large reduction in the population size needed to learn the underlying concept in the data.

An adaptive version of NLCS called ANCS is then introduced. The adaptive version dynamically controls the population size of NLCS. A comprehensive analysis of the behaviour of ANCS revealed interesting patterns in the behaviour of the parameters, which motivated an ensemble version of the algorithm with 9 nodes, each using a different parameter setting. In total they cover all patterns of behaviour noticed in the system. A voting gate is used for the ensemble. The resultant ensemble does not require any parameter setting, and showed better performance on all datasets tested.

The thesis concludes with testing the ANCS system in the architectural design for distributed environments proposed earlier.

The contributions of the thesis are: (1) reducing the UCS population size by an order of magnitude using a neural representation; (2) introducing a mechanism for adapting the population size; (3) proposing an ensemble method that does not require parameter setting; and primarily (4) showing that the proposed LCS can work efficiently for distributed stream data mining tasks.

Keywords

Action map, classification, data mining, data stream, distributed data mining, dynamic environment, ensemble learning, evolutionary computation, genetic algorithm, knowledge discovery, learning classifier system, negative correlation learning, neural network, noisy data, non-stationary environment, reinforcement learning, rule-based system, static environment, stream data mining, supervised learning.

Acknowledgement

This research could not have been completed without the help and support of many people, whom I would like to acknowledge here.

First and foremost I would like to thank Hussein Abbass, my principal supervisor, for his guidance and support. His advice and opinions helped me to overcome many problems during the last few years.

I would like to extend a thank you to Chris Lokan, my co-supervisor, for his help and support in the last three years.

Thanks go to everyone in the Artificial Life and Adaptive Robotics (ALAR) laboratory, especially Lam Thu Bui, Minh Ha Nguyen, Sameer Alam, Kamran Shafi, and Pornthep Rojanavasu for their friendship and support.

Special thanks go to my husband, Khoa Dang Ly, without his encouragement and sacrifice, I would not have been able to concentrate on the research. I would also want to thank my parents (Dam Quang Mau and Do Thi Luong), my aunties (Bui Khoat and Dam Le Duc), my family and my family in law for their support and encouragement.

Without these people, I could not have been able to pursue this research.

Hai Huong Dam

Certificate of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by colleagues, with whom I have worked at UNSW or elsewhere, during my candidature, is fully acknowledged.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Hai Huong Dam

Contents

List of Figures

List of Tables

List of Acronyms

ANCS	Adaptive Neural-Based Learning Classifier System
DDM	Distributed Data Mining
DLCS	Distributed Learning Classifier System
DM	Data Mining
EC	Evolutionary Computation
GA	Genetic Algorithms
KDD	Knowledge Discovery in Databases
LCS	Learning Classifier System
MDL	Minimum Description Length
MLP	Multi-layer Perceptron
MoC	Magnitude of Change
NLCS	Neural-Based Learning Classifier System
NCL	Negative Correlation Learning
NN	Neural Network
UCS	Supervised Classifier System
XCS	Extended Classifier System

List of Publications

Journals

1. H.H. Dam, H.A. Abbass, C. Lokan, X. Yao (2008) Neural-Based Learning Classifier Systems, *IEEE Transactions on Data and Knowledge Engineering*, 20(1):26-39.

2. P. Rojanavasu, H.H. Dam, H.A. Abbass, C. Lokan and O. Pinngern A Self-Organized, Distributed, and Adaptive Rule-Based Induction System, IEEE Transactions on Neural Networks. (Accepted to appear in January 2009)

Book Chapters

1. H.H. Dam, P. Rojanavasu, H.A. Abbass and C. Lokan (2008) Distributed learning classifier systems. In Learning Classifier Systems in Data Mining, Larry Bull, Ester Bernadó-Mansilla and John Holmes (eds), Springer, In Press.

2. H.H. Dam, C. Lokan and H.A. Abbass (2007) Evolutionary Online Data Mining: An Investigation in a Dynamic Environment. In Evolutionary Computation in Dynamic and Uncertain Environments, Shengxiang Yang, Yew-Soon Ong and Yaochu Jin (eds), Studies in Computational Intelligence Series, Springer, Volume 51/2007, pages 153-178, ISBN 978-3-540-49772-1.

Refereed Conferences

1. C.Y. Chen, H.H. Dam, P. Lindsay and H.A. Abbass (2007) Biasing XCS with Domain Knowledge for Planning Flight Trajectories in a Moving Sector Free Flight Environment, IEEE Symposium on Artificial Life (IEEE-ALife), Honolulu, 1-4 April.

2. H.H. Dam, K. Shafi and H.A. Abbass (2005). Can evolutionary computation handle large datasets? A study into network intrusion detection. The 18th Australian Joint Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, LNAI 3809, Springe-Verlag, pp. 1092-1095.

3. H.H. Dam, H.A. Abbass, and C. Lokan (2005) The Performance of the DXCS System on Continuous-Valued Inputs in Stationary and Dynamic Environments. The IEEE Congress on Evolutionary Computation (CEC'2005), Edinburgh, UK, 2005, Vol.1, 618- 625.

4. H.H. Dam, H.A. Abbass, and C. Lokan (2005) DXCS: an XCS system for distributed data mining. Genetic and Evolutionary Computation Conference (GECCO'2005), Washington D.C., ACM Press, 2005.

5. H.H. Dam, H.A. Abbass, and C. Lokan (2005) Be Real! XCS with Continuous-Valued Inputs. The Eighth International Workshop on Learning Classifier Systems (IWLCS-2005), Washington D.C., ACM Press, 2005.

Chapter 1

Introduction

1.1 Overview

The breakthrough of computer technology has facilitated the ability to generate, transfer, collect, and store data in the world. At the same time, data are produced in greater amounts with a greater frequency than at any time in the past.

Data mining, the process of discovering novel and potentially useful patterns in databases (Fayyad *et al.*, 1996), has become a key discipline to assist companies and organizations to discover the tacit knowledge hidden in the overwhelming amount of data. Within many learning tasks offered by data mining, classification is one of the most popular ones.

In many applications, the data arrive continuously in large volumes as a sequence of instances known as a data stream. Mining these data is known as stream data mining. There is a range of tasks in stream data mining and this thesis only focusses to classification tasks where the data is labeled and a classifier is required.

Due to the large amount of data arriving in a data stream, each record is normally expected to be processed only once. The data can be re-routed to different sites in the organization, or get sent to different sites by the customers. Processing data at different sites adds another dimension to the problem that it becomes distributed in nature. Distributed stream data mining poses many challenges to

the data mining community including scalability and coping with changes in the underlying concept/class in the data, over time.

The author hypothesizes that the genetics based machine learning systems (Holland, 1975) (Goldberg, 1989), particularly its Michigan-style learning classifier system (LCS), has the potential to be useful and efficient in this domain due to three main reasons.

- LCSs have been successfully applied to various data-mining problems, and can achieve competitive accuracy in comparison with other machine learning algorithms including the decision tree learner C4.5, Naive Bayes classifier, rule extraction methods such as PART, and support vector machines (Bernadó-Mansilla *et al.*, 2002),(Butz, 2004),(Dixon *et al.*, 2001).
- The mining knowledge is captured by a set of rules that can be easily interpreted by a human. This can be important: in many data mining problems, the ability to understand the learnt knowledge is sometimes as important as obtaining an accurate model.
- LCSs are incremental learners, and being evolutionary based they are inherently adaptive.

However, LCSs suffer from two main drawbacks that hinder their use as fast data mining algorithms. First, they require a large population size, which slows down the processing of arriving instances. Second, they require a large number of parameter settings, some of them are very sensitive to the nature of the learning problem. Consequently, it becomes difficult to choose a right setup for totally unknown problems. The aim of this thesis is to attack these two problems in LCSs.

Many models of LCSs have been introduced in the last two decades. Recently, the accuracy-based models have captured most attention from researchers in the field because of their better performance on both supervised and reinforcement learning problems. XCS (Wilson, 1995) (Wilson, 1998) is the first system of this type, which works for both supervised learning and reinforcement learning problems. UCS (Bernadó-Mansilla and Garrell-Guiu, 2003) is a derivation of XCS that

specializes on classification tasks. Since we are working on classification (supervised learning) problems, UCS was chosen as our baseline LCS learner.

Three main issues of stream data mining considered in this thesis are (1) how does the system adapt in dynamic environments, (2) how fast the system is to process instances as they arrive continuously, and (3) how does the system work in distributed environments. An empirical study of UCS, carried out in Chapter 3, shows that UCS is a potential online system for stream data mining because it is robust to noise and is able to adapt quickly to changes in dynamic environments.

A distributed framework is also proposed using the clients-server architecture to make UCS more efficient in distributed environments. This framework allows each local site (or client) to employ a completely independent UCS. Local models (or UCS's populations) are transmitted occasionally to the server for combination. The server then represents a complete knowledge base of the overall classification problem. Experiments show that this framework helps to reduce traffic load in the system, to protect raw data, and importantly maintain the predictive accuracy in comparison to the centralized framework.

However, the study in this chapter also shows that UCS suffers from two main drawbacks in stream data mining: (1) they require a large population size which increases communication overhead and the processing time for each instance, (2) some of the parameters are sensitive to the initial setup values such as the maximum population size.

This thesis is about a distributed neural-based learning classifier system. To overcome the problems of large population sizes, the idea of using a Neural Network to represent the action in UCS is proposed. This new system - called NLCS – is validated experimentally using a small fixed population size and has shown a large reduction in the population size needed to learn the underlying concept in the data.

An adaptive version of NLCS called ANCS is then introduced. A comprehensive analysis of the behaviour of ANCS revealed interesting patterns in the behaviour of the parameters which motivated an ensemble version of the algorithm with 9 nodes, each using a different parameter setting. In total they cover all pat-

terns of behaviour noticed in the system. A voting gate was used for the ensemble. The adaptive version in the ensemble framework dynamically controls the population size of the system. The resultant ensemble does not require any parameter setting, and showed better performance on all data sets tested.

The thesis concludes with testing the ANCS system in the architectural design proposed earlier for UCS for both logical and physical distributed environments.

1.2 Research Questions and Hypothesis

The scope of this thesis is the study of evolutionary learning classifier systems on classification problems. The study in this thesis aims at answering the following research question:

Can an evolutionary-based classifier system - such as UCS - meet the challenges imposed by distributed stream data mining?

This research question can be broken down into several sub-questions for investigation as follows:

- **Is traditional UCS suitable for distributed stream data mining?** Several studies have tested UCS on data mining problems and claimed that the algorithm is competitive in terms of the predictive accuracy in comparison to other non-evolutionary machine learning algorithms. Testing environments were mainly static, which are not always the case in stream data mining. Depending on the environment, the underlying concept of the data may change; the noise level in the data may fluctuate; the number of learning concepts may vary; to name a few. Therefore understanding the adaptive ability of the system in dynamic and noisy environments would give new insights into whether UCS is appropriate for stream data mining. The experiments in Chapter 3 are designed to answer these concerns.

  Moreover, experiments on UCSs traditionally assume that the training data is hosted at a centralized data repository. This is not always the case in the

real world, where many organizations might have distributed data sources. Despite the fact that the centralized UCS can still work well in the distributed environment by transmitting the data to a central location, the distributed framework is better because traffic load can be reduced and security and privacy issues in raw data can be protected. The experiments in Chapter 3 are also devoted to investigate a distributed framework of UCS called DUCS, which is compared against the centralized framework in terms of generalization and communication load.

- **Is a neural network's representation beneficial for evolutionary learning classifier systems?**

  One of the main drawbacks of UCS is that the number of classifiers generated is normally quite large. This is a bottleneck in distributed data streams since the computation time required to process an instance increases, and the transmission time for the models between sites in a distributed environment also increases.

  The author hypothesizes that a neural network's representation will help to compact the population of UCS, while maintaining the predictive accuracy. Chapter 4 proposes the neural representation in UCS and compares the predictive accuracy of NLCS (Neural-based Learning Classifier System) against the traditional UCS. Negative correlation learning is also added to NLCS in order to localize those networks in the same region.

- **How to reduce the bias caused by the initial choice of parameters' values?** Many parameters of UCS need to be tuned in order to perform well on a new data set. Some of them are very sensitive such as the maximum population size. This parameter influences the overall performance of the system and its best value is problem-dependent. As a result, UCS is restrained when it gets exposed to a new domain, especially those ones where the analyst has limited knowledge.

  Chapter 5 proposes an adaptive framework called ANCS which does not rely on a predefined maximum population size as in UCS. ANCS allows the pop-

ulation to grow freely over time without the upper limit as in UCS.

Moreover, an ensemble framework of several ANCSs with non-overlapping setups that cover the parameter space is investigated in this chapter as a potential solution to overcome the bias in parameters' setting.

- **Is the proposed system appropriate for distributed stream data mining?** Chapter 6 employs ANCSs in two distributed frameworks for distributed stream data mining. The first framework is simulated using a client-server architecture in Chapter 3, aiming at reducing the traffic load in a physically distributed environment. The second framework utilizes the ensemble technique to logically route the data in order to boost the overall accuracy and also to allow more instances to be processed at the same time.

  The chapter is mainly devoted to comparing these two systems with DUCS, considering several issues of distributed data streams such as accuracy, concept drift and noise levels.

1.3 Original Contributions of the Thesis

This section summarizes a list of scientific contributions from this research work. The main contribution of this thesis is to bear out a claim that UCS, the evolutionary-based classifier, can meet the challenges of distributed stream data mining. Testing UCS on distributed stream data mining reveals two key drawbacks that may limit its use in real life. These are a large population size and sensitivity to initial parameters' setup. The author proposes a neural representation and ensemble framework to overcome these problems.

In general, the contributions of the thesis can be gathered as follows:

- **Providing an empirical study of UCS in distributed stream data mining.** Although testing UCSs in data mining has been done elsewhere in the literature, the novelty of this work is to take several issues of stream data mining into consideration, including concept drifts, noise levels, processing

time, and distributed environments. The study reveals that UCSs are potentially good for stream data mining as they are robust and also can recover quickly after the concept changes. A client-server framework is proposed to employ UCSs in physically distributed environments. This framework reduces the data transmission within the system, and provides a high level protection on the raw data.

- **A neural-based representation that reduces the population size of UCS by orders of magnitude.** The thesis introduces a neural representation in UCS, that modifies the traditional representation by replacing a scalar action by a simple neural network. This neural representation helps to compact the population of UCS, while maintaining the predictive accuracy.

  Moreover, this is the first attempt to apply negative correlation learning in UCS in order to build the correlation between classifiers that match to the same environmental state. The negative correlation learning is added to the error function for training neural networks of those classifiers, aiming to push these networks far away from each other.

- **Proposing an adaptive mechanism to dynamically eliminate the initial setup of the sensitive parameters (e.g. the maximum population size).** An adaptive framework is employed in NLCS to internally change the parameters' values, based on the training performance in order to control the population size.

  To visualize the knowledge obtained in the end, a multi-dimensional scaling function is employed in LCS for the first time. The visualization reveals that the system is able to decompose closely to the hidden underlying structure of the testing data in some data sets.

  Moreover, an ensemble method is proposed to dynamically control the parameters setup in the system. The study shows that this framework does not only reduce the bias due to parameters' values, but also increases the predictive accuracy.

- **The enhanced UCS is able to face the distributed stream data mining challenges.** Two frameworks are proposed to help the enhanced system handle logically and physically distributed environments. The first case allows more data instances to be handled at once by distributing the work load over multiple classifier systems. The latter case reduces the traffic load required in the system, while maintaining the predictive accuracy.

1.4 Road Map

The thesis is organized as follows:

Chapter 1 presents an introduction to the thesis. It first provides a glance at the research field, followed by the motivation and research questions. A list of scientific contributions from this research work is discussed in detail. Finally, a brief outline of the thesis is given.

Chapter 2 provides a short review of data mining in general, concentrating on the classification task. It is followed by a short discussion about challenges of stream and distributed classification in the literature. Two techniques for classification are reviewed: evolutionary learning classifier systems and feed-forward artificial neural networks.

Chapter 3 starts with an initial investigation of UCS for stream data mining. A number of aspects of data streams is considered such as noise levels, concept drifts, and processing time. This chapter also proposes a distributed framework of UCS to handle distributed data sources. Several aspects of a distributed system are explored in this chapter such as knowledge combination methods at the central location, traffic load, and knowledge exchange.

A novel neural representation is proposed in chapter 4. The enhanced system, a neural-based learning classifier system (NLCS), differs from UCS in the representation of the action. NLCS is then evaluated on real-world data mining problems.

NLCS is extended with an adaptive framework in chapter 5. An ensemble system is proposed in order to reduce the effect of initial parameter settings.

The proposed system is applied to the distributed framework and tested on distributed stream data mining problems in chapter 6.

Chapter 7 concludes the thesis and opens several research directions that emerged from the current work.

Chapter 2

A Survey of Data Mining and Evolutionary Learning Classifier Systems

2.1 Overview

The interest in data mining (DM) has been growing rapidly in both the research and industry communities, coupled with the increasing volume of data. The breakthrough of computer technology has facilitated the ability to generate, transfer, collect, and store data at a greater frequency than at any time in the past. Many DM techniques have been introduced in the past and many of them have been claimed to effectively extract "knowledge" from massive amounts of data.

Recently, data streams, continuously and rapidly arriving data, have posed a number of challenges to traditional data mining methods (Aggarwal, 2007). Some of them include the ability to process the data on the fly instead of using multiple passes; and the ability of the system to handle concept drifts, to name a few.

Michigan-style evolutionary learning classifier systems (Holland, 1975) were chosen in this study due to several reasons: their rule-based learning knowledge is the easiest representation for a human to interpret and understand; their design to

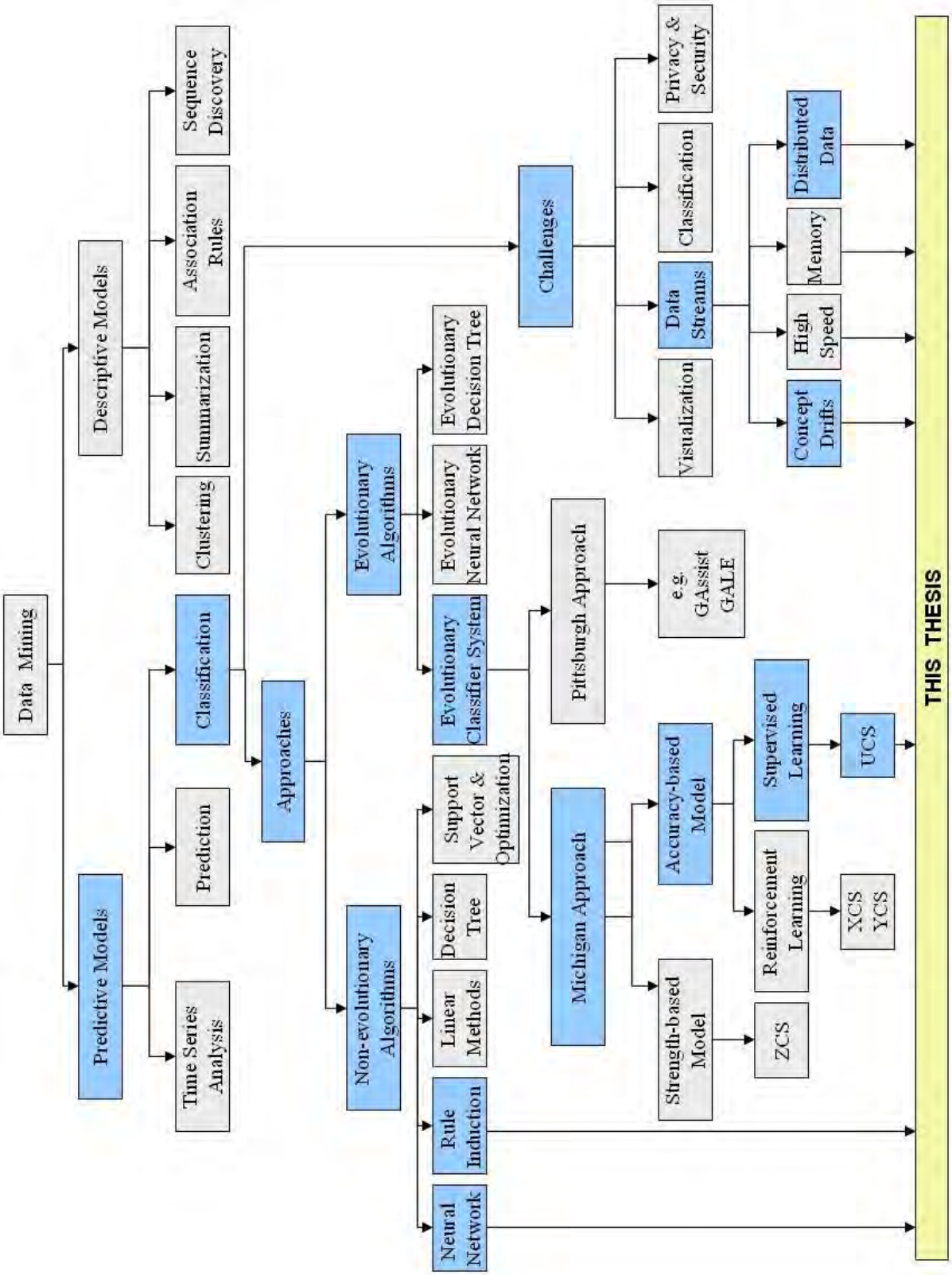

Figure 2.1: The road-map to research on stream data mining.

Table 2.1: A summary of literature survey on learning classifier systems that is closely relevant to the main issues of this thesis

Research Areas	Year	Authors
Representational Issues		
Condition	2000	Wilson
	2001	Wilson
	2003	Stone and Bull
	2005	Butz
	2005	Dam, Abbass, and Lokan
	2006	Butz
	2006	Lanzi
Action	2002	Wilson
	2007	Lanzi and Loiacono
	2007	Loiacono, Marelli, and Lanzi
Others	1999	Lanzi and Perrucci
	2002	Bull
	2002	Bull and O'Hara
	2004	Hurst and Bull
	2005	Mellor
	2005	O'Hara and Bull
	2007	Bull, Lanzi, and O'Hara
Population Compactness	2002	Wilson
	2002	Fu and Davis
	2003	Dixon, Corne and Oates
	2004	Wyatt, Bull, and Parmee
Scalability	2002	Llorà
	2006	Llorà and Sastry
	2006	Llorà, Marelli and Lanzi
Adaptive Parameters	2000	Bull and Hurst
	2001	Hurst and Bull
	2002	Hurst and Bull
Ensemble Learning	2005	Bull, Studley, Bagnall and Whittley
	2005	Gao, Huang, Rong and Gu
	2007	Bull, Studley, Bagnall and Whittley
Data Mining Classification	2001	Bernadó-Mansilla, Llorà, and Garrell-Guiu
	2003	Bernadó-Mansilla, Garrell-Guiu
	2007	Bacardit and Butz

process data instances on the fly may potentially work for data streams; and their abilities to achieve an equivalent accuracy as other traditional techniques on several classification problems (Butz, 2004).

This chapter provides a brief review of distributed stream data mining. Figure 2.1 demonstrates a road-map of the literature of the thesis. Table 2.1 reviews

the main focus of this thesis in evolutionary learning classifier systems.

The chapter is structured as follows. Section 2.2 presents a brief overview of data mining. The classification problem in data mining is studied in Section 2.3. Challenges of classification in stream data mining are discussed in Section 2.4. The background of evolutionary learning is presented in Section 2.5 followed by a review of Michigan-style learning classifier systems. Section 2.7 explains feed-forward artificial neural networks. Finally, several emergent questions are discussed in Section 2.8 and conclusions are drawn in the last section.

2.2 Data Mining

Data mining (DM) has been attracting considerable attention from research, industry, and media in the last two decades because of its powerful tools and techniques for connecting the dots from raw data. Either an inside view of the database or predictive outcomes would promise some potential competitive advantages to organizations, especially once combined with opinions from experts in the field.

2.2.1 The Scope of Data Mining

The term "Data Mining" has been widely used in the last two decades by statisticians and database researchers, but only recently recognized within the business community. There are some debates about the exact definition of this term. Many researchers consider DM as a single step in the multi-step process of knowledge discovery in databases (KDD). Alternatively, others view it as a synonym for KDD. However according to Cantu-Paz and Kamath (2002) most people have agreed that DM combines ideas from multiple fields including machine learning and artificial intelligence, statistic, signal and image processing, mathematical optimization, and pattern recognition. Several definitions of DM are presented as follows.

> Data Mining or KDD as it is also known, is the nontrivial extraction of implicit, previously unknown, and potentially useful information from

> data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency networks, analyzing changes, and detecting anomalies (Frawley *et al.*, 1992).
>
> Data Mining is a step in the KDD process consisting of particular data mining algorithms that, under some acceptable computational efficiency limitations, produces a particular enumeration of patterns (Fayyad *et al.*, 1996).
>
> Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner (Hand *et al.*, 2001)

In this thesis DM is referred to as a single step of KDD, which automatically searches for potentially useful patterns of interest from large volumes of data and presents those patterns in a particular form that can be potentially understood by a human. Patterns or models, outcomes from a data mining task, often draw the relationships of the underlying data or describe the data in particular ways.

In general, the KDD process starts with raw data (e.g. records of financial transactions, information of coming network packets, etc.) and in the end produces some valid, novel, and potentially useful patterns, which promise unfolding the hidden information.

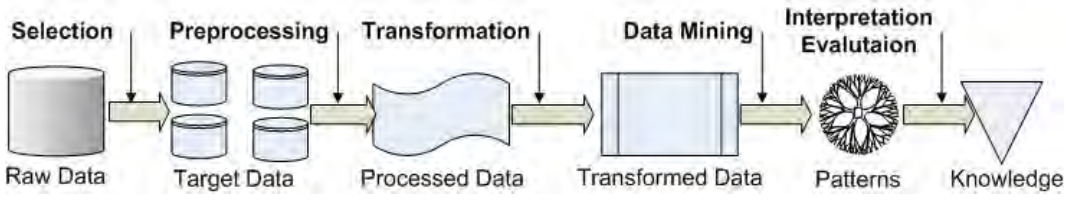

Figure 2.2: Knowledge discovery in databases (Reproduced from (Cantu-Paz and Kamath, 2002))

The following steps depicted in Figure 2.2 are required to fulfil the KDD process:

- Data selection: this step requires an understanding of the domain and the goals of the end-user. As a result, potentially useful and relevant features are selected and extracted from the raw data. This step aims at reducing the amount of time needed for processing; thanks to human experts.

- Data cleaning and preprocessing: this step is responsible for cleaning the data in such a way that it is useful for the program. It includes the removal of noise, outliers and irrelevant data; fixing incorrect formats of data; deciding on methods to handle missing values, etc.

- Data transformation: converts the data to another form using dimensionality reduction or transformation, if needed, for better understanding of the relationship among the features or filtering irrelevant data.

- Data Mining: searches for patterns of interest by employing the data mining algorithm(s) on the data.

- Data interpretation and Evaluation: reorganizes the learning patterns in such a way that a human can easily understand. This also involves in resolving potential conflicts with previously believed knowledge.

While these above steps are equally important, the DM is a key step. This thesis mainly focuses on this step. The author assumes that the data is clean, consistent, and ready to be used by the mining algorithm.

2.2.2 Reasons for Using Data Mining

To explain why DM has become so important recently, Han (2005) presented two key reasons: (i) the wide availability of massive amounts of data; (ii) the imminent need for transforming such data into useful knowledge and information.

In the last few decades, a huge amount of data can be collected daily in many organizations. For example, phone companies have the huge transactional data of customers' calls; internet companies collect data of network load, faults, etc. Previously we are familiar with Terabyte and Gigabyte sizes of databases. However,

many companies and organizations have started to use databases of up to the size of Petabyte and Exabyte. Figure 2.3 from the Red Brick company illustrates the exponential growth of data during the past few years.

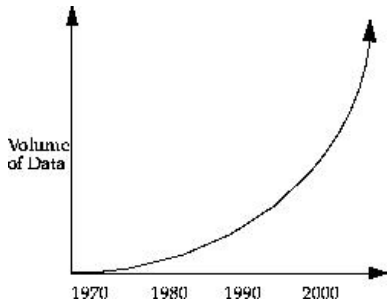

Figure 2.3: The growth of electronic data (Reproduced from http://www.redbrick.com)

Hidden knowledge from these huge databases is likely to benefit many organizations in marketing, decision making, management, future prediction, to name a few. Although knowledge discovery is not a new idea, it became popular recently due to the tremendous amounts of data available. Naisbitt (1988) stated that *we are drowning in information, but starving for knowledge.* It is because the scale of data seems to grow beyond human capacities to analyze and interpret manually.

In fact, traditional analysis methods have become impractical after the exponential growth of data volumes. For instance, human analysts used to take an important role in understanding and obtaining knowledge from the data with the assistance of software such as Microsoft Excel, SPSS, etc. Three main problems associated with this approach are the human brain, which is normally unable to search for complex multi-dependencies in data; the analysis packages/software that can not handle large data sets; and the lack of objectivity in such an analysis.

A further attractiveness of using DM is that it requires lower cost and produces faster results than a team of professional statisticians. In 1989, the first KDD Workshop was organized in response to the need of an automated method for knowledge discovery in large databases. Later, this workshop grew into the KDD conference as interests in the field continued to increase.

2.2.3 Data Mining Tasks

The goals of DM can be broken down into two main high-level categories of data analysis: description and prediction (Han, 2005). The former aims at identifying general properties of the data, reorganizing it, and transforming it into some forms of human-interpretable patterns. An example of patterns is a customer similarity that can be used to form a group of customers with similar behaviours. The latter performs some kinds of model fitting based on current and past data in order to make predictions. The importance of these goals might vary considerably depending on domains and applications. For example, machine learning is concerned more about prediction, while KDD tends to favour description.

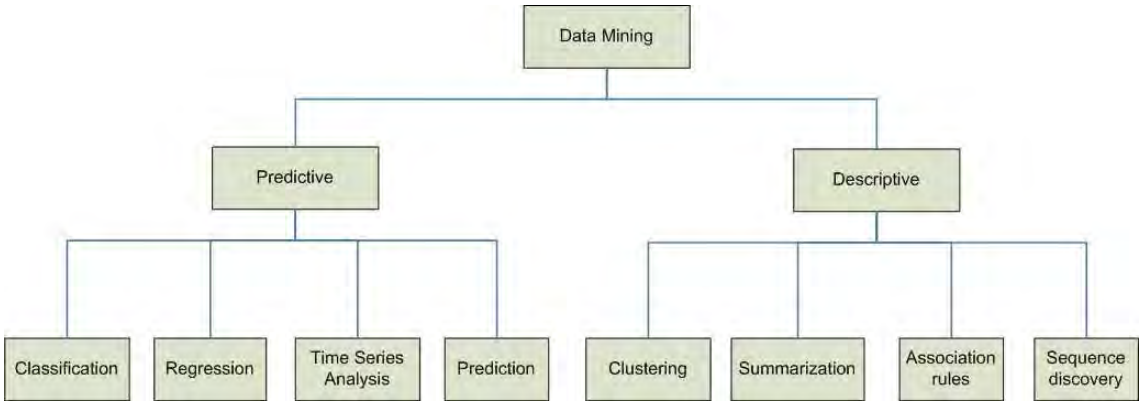

Figure 2.4: Data mining models and tasks

Figure 2.4 shows several primary learning tasks in data mining applications to accommodate these goals. The popular learning tasks of DM are:

- Classification: aims at developing a model that can discriminate accurately unseen samples among a finite set of categories. The model is designed to identify commonalities in a data set through training instances.

- Regression: tries to fit a model to identify the underlying trend in the data.

- Clustering: attempts to group a set of examples that are close to each other based on some sorts of a similarity measure to form a descriptive model to describe the data.

- Association learning: intends to classify input instances into appropriate classes as well as to build the relationship among those features.

Among these, classification is one of the most popular tasks. This thesis focuses on the classification problem in data mining.

2.2.4 Framework of Data Mining

After the learning task is identified, one might consider what to do next. Normally the answer to this question involves two main phases: building the model and using it. In the first phase, mining algorithms need to be constructed in order to fit a model to the characteristics of the data. Fayyad *et al.* (1996) outlines three primary components in any data mining algorithm: model representation; model evaluation; and the optimization and search method. The simple algorithm of DM is described in Algorithm 1.

Learning task: Identify the learning task based on the goals of the job;
Model representation: Determine an appropriate representation of the learning knowledge;
Model Evaluation: Decide how to evaluate the learning model (e.g. choosing a score function);
Optimization and Search Method: Choose a learning algorithm to optimize the score function;
repeat
  Search for particular patterns (either parameters and model);
  Add them to the knowledge base and revise the learning model accordingly;
  Evaluate how good the learning model is by the chosen evaluation model;
until *the termination conditions are met* ;

Algorithm 1: A DM algorithm (Fayyad *et al.*, 1996)

The model representation component forms the format in which proposed patterns and models of the underlying data will be described. The model evaluation component is used to measure how well the proposed patterns or models match the data. Finally, the search component is responsible for determining the appropriate models or patterns.

Once the model is built, it can be used for solving several tasks stated in the

previous section. The first phase is very important in forming models, which are potentially useful later.

2.3 Classification

Classification is one of the most popular tasks of data mining. Han (2005) presented five criteria for comparison between different learning algorithms: predictive accuracy, learning speed, robustness, scalability, and interpretability. This section will provide a brief review of classification.

2.3.1 The Problem Definition

Classification is the task of predicting the category/concept associated with some data instances based on a model derived from previously observed data. This process is made up from two key phases, which are normally called the training and testing phases. The first one aims at building a descriptive model or extracting potentially useful patterns from a set of previously observed data instances. The second one is carried out by applying the model on new instances of the data.

Normally data is divided into two subsets for training and testing. The training set consists of an input vector and a label vector that are used together to train the system. The testing set, on the other hand, consists only of an input vector and is used to measure the generalization of the learning model. That is, how good the model is when confronted with unseen instances.

The training set contains a set of data instances or examples as

$$D = (x_1, t_1), .., (x_n, t_n), ..., (x_N, t_N)$$

A data instance is a tuple (x_i, y_i), where $x_i \in X$ is the input vector to the system, and $t_i \in T$ is the associated target. The output space T is discrete, where $T = c_1, c_2, c_3, ...$ and c_i is a possible 'class' or target output. Consider an unknown function $\phi : X \longrightarrow T$, which maps an input space X to an output space T. The

classification task is to learn a target function $f(x;\xi)$ as our estimated model of ϕ. When we do not wish to refer to any specific data or parameter, f is simply used.

Both training and testing sets are assumed to represent samples of the underlying problem. In some cases, the test and training data are distinct in nature. If they are not distinct to each other, the performance of testing and training data are likely similar, and therefore the performance of testing data is not a good indicator of the performance on future data. All updates and revisions of the model happen only during the training phase. The testing instances are only used to measure the performance of the system, and are not used to update the model.

The first step is to observe the behaviour of the random process within a period of time in order to build the model. This step is done by employing one or more learning algorithms to accomplish the task.

Once the learning model has accumulated enough information, it can be used for prediction. Figure 2.5 depicts the use of a classification model in the testing phase. The model can be considered as a black box to map a testing input x into its outcome y. This prediction can be used to draw a future trend with a reference

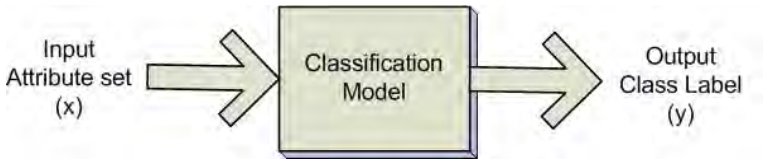

Figure 2.5: Classification model for mapping an input x and an output y

to the history. Traditionally, the data is assumed to be static, that once the model has learned it, the model can be used all the time. However, this is not always the case. More details will be provided in Section 2.4.

2.3.2 Classification Criteria

In order to verify the learning algorithm, several criteria are normally used for judging a model. These include accuracy, compactness, robustness, efficiency, and interpretability. The best algorithm would be the one which satisfies all these criteria in all datasets. However, according to the "No Free Lunch Theorem",

no single model will always outperform all other models. This was theoretically investigated by Wolpert and Macready (1997).

Mitchell (1997) argues that the choice of representation involves a crucial trade-off between expressive, accurate, and compact models. The predictive accuracy measures the generalization of the learning model when exposed to unseen instances. Compactness refers to the length or size of the learned model, while expressiveness relates to the understandability of the knowledge represented by the learned model. These three criteria will be used throughout the thesis for designing an algorithm. This subsection will give a brief summary for some of these criteria.

2.3.2.1 Predictive Accuracy

Classification tasks are solved by one or more machine learning algorithms. The main target is to capture as much underlying information as possible from the training data. A good algorithm is the one which can generalize wider than its original training data so that it is still able to predict correctly in new and unknown situations.

After receiving a training/testing instance, the classifier will provide its prediction based on its current knowledge. If the predicted label is identical to the real target of the instance, it will be counted as a success; otherwise it is a misclassification. The predictive accuracy of training/testing is the percentage of successful predictions with regards to the whole training/testing sets. The function of the predictive accuracy is described as follows:

$$a(f(;\xi)) = \frac{1}{N}\sum_{n=1}^{N} L(f(x_n;\xi), y_n) \tag{2.1}$$

$$L(a,b) = \begin{cases} 1 \text{ if a=b} \\ 0 \text{ if a} \neq \text{b} \end{cases} \tag{2.2}$$

High accuracy on training sets reveals good learning ability of the system but it does not indicate good generalization. A model that generalizes well is the one

which achieves as high accuracy as possible on the testing set.

Many learning methods can suffer from either under-fitting or over-fitting. The sign of over-fitting is that the model fits very well to the training data, but produces poor results on unseen instances. In this case the learning knowledge cannot generalize to other cases. In contrast, an under-fitting model is the one which under-fits the data because the underlying problem is too complicated for it to learn.

2.3.2.2 Compactness and learning speed

This refers to the computational cost (time and space complexity) of the system to form a learned model and to classify an instance. The faster model provides a better solution when dealing with data streams as it is able to handle more data instances.

2.3.2.3 Robustness

This refers to the ability of the system to deal with noisy environments. Noise is an unavoidable factor in the real world. Noisy data would mislead learning and therefore might lead to inaccurate models. A good model is the one which can cope well with noise in order to produce correct predictions.

2.3.2.4 Efficiency and scalability

Data streams always involve large amounts of data. A good model on data streams is the one that can perform efficiently on not only small data sets but also large ones.

2.3.2.5 Expressiveness

This refers to the ease of understanding and gaining an insight from the learned model. A good model not only produces a good predictive accuracy, but also represents the knowledge in an easy format that a human can understand. Many researchers value the extracted knowledge in data mining. Many believe that the

explicit knowledge structures acquired from learning are at least as important as the ability to perform well on new examples in many data mining applications (Witten and Frank, 2005) (Freitas, 2003).

2.3.3 Representation

In data mining, the learning knowledge needs to be represented in an explicit form so that an algorithm can recognize and manipulate a given task. In order for the DM program to formulate the target function and therefore to discover interesting patterns, the representation of the mined knowledge needs to be chosen in advance. This is the way that the mining patterns are structurally expressed and described. There are many ways for representing knowledge in machine learning such as: decision tables, decision trees, classification rules, neural networks, etc. Each representation has its own advantages and disadvantages depending on the mining techniques being used.

- Decision tables: are the simplest approach but their big problem is to define attributes to leave out without affecting the overall performance.
- Decision trees: are a "divide-and-conquer" approach to the problem of learning, where the knowledge is represented naturally in a tree format.
- Decision rules: are a popular alternative to decision trees because a set of classification rules can be formed easily from a decision tree. Each leaf in the tree can be expressed by a rule, which includes conditions of all nodes on the path from the root to the leaf, and the class at the leaf becomes the class of the rule.
- Neural networks: are presented as a black box as they are very complicated for a human to understand. This representation mimics the human brain with many units interconnected in layers that form a directed acyclic graph.

Mitchell (1997) recommended that a ruleset is one of the most expressive and human readable representations. The major benefit of this representation is the

accessibility of the subsequent knowledge (a set of rules) obtained by the system. The rule is normally represented in the following form:

$$IF < some\_conditions\_are\_satified > THEN < predict\_some\_value >$$

An example of a set of rules derived from a learning system for forecasting the weather is as follows:

$$IF < todayishumidandcloudy > THEN < itwillraintomorrow >$$

$$IF < todayishotanddry > THEN < itwillbesunnytomorrow >$$

Learning decision rules can be carried out by traditional algorithms such as C4.5 (Quinlan, 1993). Alternatively, genetic algorithms and learning classifier systems (Goldberg, 1989) have been confirmed to work well in this domain. This thesis focuses on learning classifier systems.

2.3.4 Training Modes

There are two principal training modes in the literature, mainly depending on the presentation of the training examples: online and offline learning.

- Online Learning (also known as sequential training or stochastic training mode): in this mode, the learning algorithm processes one example at a time, then updates and revises the learning model accordingly.

- Offline Learning (also known as batch learning): in this conventional method, changes to the learning model are accumulated over an entire presentation of the training data (an epoch). The update is only applied at the end of an epoch.

Some people believe that the batch learning mode would produce a better performance due to its use of the true gradient direction for updates. Others argue

that the gradient direction might be trapped easily in a local optimum while the online mode can overcome this problem due to its stochastic nature (Bishop, 1995).

The main drawback of offline learning in data mining is that it is not adequate for large data sets as it requires more time and computational cost in comparison to the online learning (Bottou *et al.*, 1990). Moreover, if a training environment changes, the learning algorithm using the online mode can adapt more quickly to new information. Therefore, training in the batch mode would be inappropriate for many domains due to the size of the data and the rate at which the training data is generated.

Online learning becomes more important than ever for dealing with these problems. The main problem is to deal with an infinite stream of one-pass instances, which requires the learning algorithm to continuously refine and revise the current knowledge.

2.3.5 Classifier Ensembles

The ensemble technique has been widely used to improve the accuracy of the classifiers. In this approach, predictions from multiple classifiers are combined to produce a single classifier, which is generally more accurate than any of the individual classifiers in the ensemble (Krogh and Vedelsby, 1995). Many researchers have shown that a committee of experts can produce better performance than an individual expert (Nguyen *et al.*, 2004),(Liu *et al.*, 2000).

Polikar (2006) stated five reasons why an ensemble system is preferred: (1) statistical reasons - the averaging of ensemble reduces the overall risk of making a poor selection; (2) large volumes of data - training different classifiers with subsets of data is often more efficient than training a single classifier; (3) too little data - re-sampling the data to create several overlapping subsets for training different classifiers has been proven to work effectively ; (4) divide and conquer – the complex problem can be divided into several simpler ones for different classifiers to learn; (5) data fusion – if data comes from different sources with heterogeneous features or different noise levels, a single classifier cannot learn the whole data.

In order to build an ensemble classifier, two major components need to be considered when designing an algorithm: pre-gate and post-gate (Abbass, 2003). The first one is how to route data to several individual classifiers. The second one is how to combine their responses at the gate level.

Polikar (2006) suggested that no single ensemble generation algorithm or combination method is universally better than others. A good ensemble is one where individual classifiers learn different parts of the input space and therefore make different errors (Hansen and Salamon, 1990) (Krogh and Vedelsby, 1995)(Nguyen *et al.*, 2004). In other words, diverse individuals need to be maintained so that there is always disagreement among them. Studies in (Opitz and Shavlik, 1996a) (Opitz and Shavlik, 1996b) empirically verified that such ensembles generalize well.

One of the most popular methods for promoting the diversity within an ensemble is to re-sample the training set. Two typical methods of this type are bagging (Breiman, 1999) and boosting (Schapire, 1990). The B=bagging approach was initially used in (Breiman, 1996) as a "bootstrap" to increase the accuracy of a learning algorithm by averaging the outputs from different models generated with different data subsets (not necessarily distinct). It will randomly generate a new training set with a uniform distribution for each network member, from the original data set. The boosting approach, on the other hand, re-samples the data set with a non-uniform distribution for each ensemble member. The whole idea of boosting and bagging is to improve the performance by creating some weak and biased classifiers. When aggregating these classifiers, using an average or other mechanisms, the bias of the ensemble is hoped to be less than the bias of an individual classifier. In short, boosting and bagging are used to diversify the ensemble in order to perform better in comparison to a non-diverse ensemble (Skurichina *et al.*, 2002).

Beside the re-sampling technique, other methods try to create different bias within classifiers in order to maintain diversity. Alpaydin (1993) appointed different learning parameter sets to each classifier so that they will converge to different solutions. Other work by Maclin and Opitz (1997) generated various initial neural network weight settings. Hashem (1997) applied different classifier architectures.

Additionally, other researchers exploit the evolutionary computation to generate diversity (Opitz and Shavlik, 1996b) (Opitz and Shavlik, 1996a). Moreover, Abbass (2003) proposed the multi-objective optimization approach to evolve a group of networks with different objectives. Other group of work promotes diversity based on individual's correlation (Krogh and Vedelsby, 1995) (Rosen., 1996) (Liu and Yao, 1997) (McKay and Abbass, 2001).

Predictions from individuals need to be combined for the final prediction. There are three simple but quite popular methods for decision making at the post-gate level in ensemble learning:

- Majority voting : The output of the ensemble is the class which receives the vote of the majority of networks in the ensemble.
- Simple averaging : The output of the ensemble is the arithmetic mean of the individual outputs of the network.

$$\hat{y}_{ens} = \sum_{i=1}^{M} \frac{\hat{y}_i}{M} \tag{2.3}$$

- Winner take all: The output of the ensemble is the output of the network whose output is maximally different from the classification threshold.

$$\hat{y}_{ens} = arg_{max,i}(|\hat{y}_i - threshold|) i \in [1, M] \tag{2.4}$$

Research by Lincoln and Skrzypek (1990) and Mani (1991) showed that a simple averaging is a good composite method. Recently, other studies suggested that the voting scheme would provide better performance (Maclin, 1998) (Hashem, 1997) (Wolpert, 1992).

Chan and Stolfo (1993) introduced another approach for combining models called *meta-learning*. In this approach, each local site may employ a different inductive learning algorithm for learning its local model. A meta-classifier is trained using data generated by the local models. This process is applied recursively to produce an arbiter tree, which is a hierarchy of meta-classifiers. Meta-learning has

been used for fraud detection in the banking domain.

An alternative to meta-learning is the *knowledge probing* method developed by Guo *et al.* (1997). This method is similar to meta-learning but it does not build an arbiter tree. Instead, one descriptive model is generated from the predictions of local models from a distributed environment.

Both meta-learning and knowledge probing methods were applied to traditional inductive learning algorithms such as decision tree learning: ID3, CART, C4.5; memory-based learning: WPEBLS; Bayesian learner based on conditional probabilities: Bayes, etc (Guo and Sutiwaraphun, 2000) (Prodromidis *et al.*, 2000). These approaches are efficient and effective. However, they are normally used as off-line learning mechanisms, where the training of the model needs to be completed before one can use the model for mining. Moreover, the model structure can be sensitive to small modifications in the data.

According to Sharkey (1996), other methods for fusing the knowledge include averaging and weighted averaging, non-linear combination methods with rank-based, probability distribution for experts' knowledge, stacked generalization, decision template, and the Dempster-Shafer based classifier fusion.

2.4 Data Stream Classification

Many algorithms designed for classification work in the off-line mode, which requires the whole data set to remain in the memory for multiple passes through the algorithm. In some cases, the data set is static and small so that the algorithm can be updated at the end of each pass.

However, in many applications such as sensor networks, satellite communication, internet, email logs, network, survey, and credit card transactions, the amount of data can be so large that it may not be possible to be stored in the memory. Moreover, the data is generated continuously and there is a need to analyze data in near real time, for example, to detect network intrusions, credit card fraud, anomalous activity, etc. Such kind of data, or so-called *data streams*, have posed many

challenges to traditional data mining techniques.

Aggarwal (2007) expressed that it is impossible to process data streams with multiple passes as in many conventional methods. Moreover, the data may evolve over time in many applications. Therefore, a straightforward adaptation of one-pass learning algorithms need to be considered with a clear focus on the evolution of the underlying data.

2.4.1 Data Streams

A data stream is a sequence of instances that continuously arrive in real time. Several unique characteristics of data streams have been identified by Babcock *et al.* (2002) as follows:

- The data instances arrive continuously and rapidly in real time. Each data record can be processed once or a small number of times due to limited computation and storage capabilities of the system.
- The system has no control over the order of which data instances arrive to be processed.
- Data streams are potentially unbounded in size.
- Once an element from a data stream has been processed or archived, it cannot be retrieved easily unless it is explicitly stored in memory, which is typically small relative to the size of the data streams.

In order to handle data streams, these characteristics need to be taken into account. Gaber *et al.* (2007) summarized several research issues of stream data mining as follows:

- High speed nature of data streams: The rate of building a classification model needs to be higher than the data arrival rate. Moreover the one-pass constraint is enforced during training.

- Unbounded memory requirements: the volume of data streams is always large. Many researchers have concentrated on this issue using load shedding, sampling to create subset of data, etc.

- Concept drifting: data normally change over time, which requires the classifier to update accordingly. The use of an outdated model could result in inaccurate predictions and low accuracy models.

- Tradeoff between accuracy and efficiency: this is the main tradeoff in stream data mining algorithms.

- Challenges in distributed applications: many applications are structured in a distributed nature. The challenge in these applications is to reduce the bandwidth limits in transferring data.

- Visualization of stream data mining results: the need for visualization in data mining is always essential. Stream data mining algorithms need to be able to visualize on the fly the discovered patterns for the user to make quick decisions.

- Modeling change of mining results over time: modeling the change in data streams over time could be important in some cases.

- Interactive mining environment to satisfy user results: the fast nature of data streams could make it difficult to allow user interaction.

- The integration of data stream management systems and stream data mining approaches: many applications might require to add storage, querying, and reasoning into the mining algorithm in order to obtain a complete system for stream processing.

- Hardware and other technological issues: burden of data streams can be shared with hardware solutions.

- Real time accuracy evaluation and formalization.

The methods proposed in this thesis consider speed, and the trade off between efficiency and accuracy, and some contribution is made in visualization. However, the main focus is on two issues: concept drift, and distributed environment.

2.4.2 Concept Drifts

An essential requirement of an algorithm in data streams is to be able to detect and recover quickly from hidden changes, while reusing previous knowledge. In the literature of data mining, a dynamic environment is mainly concerned with *concept change* (or *concept drift*), in which a target learning concept changes over time (Widmer and Kubat, 1996). In the real world, concept change occurs so frequently that any online data mining application needs to take it into account seriously. For example, a company's policy may need to change frequently to reflect the consumer's market.

2.4.2.1 Characteristics

The target learning concept can change in many ways under many aspects. Abbass *et al.* (2004) determined six common kinds of changes:

- Change in the model. A learned model may become incorrect after a period of time. For example, customers' preferences for shopping change due to changes in seasons, fashion, etc. Hence a model of customers' preferences in summer is quite different to the one in winter.
- Change in the number of underlying concepts. The number of concepts may not stay constant forever in real-life applications. Some new concepts may be introduced and some old ones may become obsolete. For example, a school may introduce a new class on network security, due to high demand from students, and may stop teaching programming in Pascal because not many students take the unit. A model to classify a unit based on students' preferences needs to be modified to capture new concepts and eliminate obsolete ones.

- Change in the number of features. The number of available features may vary over time.
- Change in the level of noise. Noise is an unwanted factor but it occurs very often in real world data. The noise level may change reflecting the condition of an environment. For instance, voice data may have a high level of noise at a supermarket during daytime, but may have a low level of noise during nighttime.
- Change in the class distribution. The class distribution of training data can be changed over time.
- Change in the sample bias. Sample bias can be changed reflecting different conditions under which data is collected.

They grouped these changes into two main areas: model boundary changes and sample changes. The first three changes belong to the first category and the last three changes belong to the latter category. This thesis focuses mainly on changes in the model boundary.

2.4.2.2 Techniques

An effective learning algorithm for tracking concept change is one that can identify changes in the target concept without being explicitly informed about them; can recover quickly from changes by adjusting its knowledge; and can use previous knowledge in the case that old concepts reappear (Widmer and Kubat, 1996).

Much work has been focusing on utilizing traditional classification techniques for data streams. Gaber *et al.* (2007) grouped these into two main categories: data-based and task-based techniques. The idea of the first technique is to obtain a smaller set of data by re-sampling the data set (e.g. sampling, load shedding, sketching, etc.) or transforming the data set (e.g. synopsis structure, aggregation, etc.). The latter approach, in contrast, employs some modification techniques (such as approximation algorithms, sliding window, algorithm output granularity, etc.) in order to cope with time and memory requirements of data streams.

The first system designed for concept changes is STAGGER (Schlimmer and Fisher, 1986), using probabilistic concept description. The system responds to concept change by adjusting weights in the model, and discarding any concepts that fall below a threshold accuracy.

Since then, researchers have proposed several rule-based algorithms to deal with concept changes. A family of FLORA algorithms (Widmer and Kubat, 1996) is one of them, which learns rules from an explicit window of training instances. A window means that oldest examples are replaced by newly arrived ones so that they fit into the window. The learner trusts only the latest examples. FLORA2 allows dynamic size of the window in response to the system performance. FLORA3 stores concepts for future use and reassesses their utilities when context changes are perceived. FLORA4 is designed to deal with noise in the input data.

Klinkenberg and Joachims (2000) developed an alternative method to detect concept change using support vector machines. They also use a window of recent instances to detect changes. Even though the system might be sensitive to changes in the distribution of instances, the window technique is almost an acceptable approach.

Classifier ensembles are an alternative approach in data mining for dealing with concept change. A set of experts is maintained. The prediction is made by combining the experts' knowledge, using voting and/or weighted voting.

Street and Kim (2001) suggested that building separate classifiers on sequential chunks of training data is also effective to handle concept change. These classifiers are combined into a fixed size ensemble using a heuristic replacement strategy. Kolter and Maloof (2003) presented an approach based on the weighted majority algorithm to create and remove base learners in response to changes in performance.

An important challenge of the ensemble approach is that the system needs to have a strategy to create new experts and eliminate old ones to adapt to the environment.

Beside those conventional methods, many researchers are working on evolutionary techniques for classification as in (Goldberg, 1989), (Wilson, 1995), (Bernadó-

Mansilla and Garrell-Guiu, 2003).

This thesis focuses on an evolutionary learning classifier system. The system can be trained on the fly, potentially good for data streams. Moreover, the approach employs the genetic algorithm as a search technique, which has been claimed in (Goldberg, 1989), (Goldberg, 2002) to be suitable for problems with complex fitness landscapes, noise, change over time, and many local optima.

2.4.3 Distributed Environments

Most databases in large companies/organizations are often inherently distributed in multiple locations. A centralized mining approach is a straightforward process as shown in the left hand side of Figure 2.6. It involves two main steps in which the first one is mainly concerned with transferring all local data to a central location. A data mining algorithm is employed on the integrated data at the central site. This approach would be inefficient, especially in the domain of data streams, as fast communication channels with large bandwidths are required. In many cases, security is a big concern if the raw data needs to be sent over the network.

The primary purpose of distributed data mining (DDM) is to allow parallel processing of databases at multiple sites, to discover and combine useful knowledge from multiple models (Prodromidis *et al.*, 2000).

DDM is a relatively new area but has been receiving much attention, especially in distributed environments where trust between sites is not always complete or mutual (Jones *et al.*, 2000). In many applications, data are privacy-sensitive, so that centralizing the data is usually not acceptable (Giannella *et al.*, 2004). Therefore, a DDM technology needs to be adopted in these applications to minimize the transmission of raw data and thus protect raw data.

Data in DDM can be divided into two categories: homogeneous and heterogeneous. In homogeneous DDM, the databases located at different sites have the same attributes in the same format, while in heterogeneous DDM, the attributes at each site are different or in different format. The focus of this thesis is on homogeneous

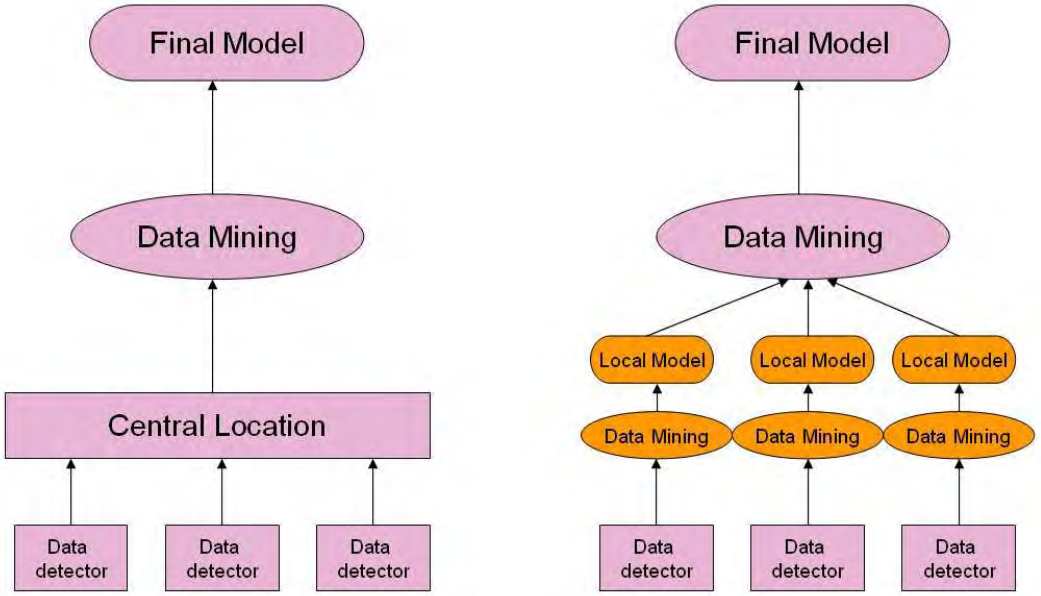

Figure 2.6: Distributed environments - centralized data mining model (left hand side) and distributed data mining model (right hand side)

DDM.

DDM overlaps with parallel data mining in many ways. Kargupta and Chan (2000) divided parallel data mining into two main approaches: coarse-grained and fine-grained. The former means that data is distributed physically, and a collection of loosely coupled algorithms is employed at each location for mining local knowledge. This approach is quite similar to distributed data mining. The latter is for tightly coupled systems.

Scaling up the algorithm is desirable in stream data mining due to the large size of the training set. Provost (2000) stated that the motivation of parallel data mining starts from the need to scale up to massive data. He explained that the run-time complexity of data mining is linear or worse in the total number of instances, and large data sets can be expensive to mine unless attention is paid to scaling up.

A large fraction of DDM approaches focuses on centralized ensemble-based methods (Kuncheva and Whitaker, 2003), which first form local models at the local sites and then combine these models at a central site. For example, a company may have different branches. Each local model represents an independent data mining for each branch, then the central office of the company combines the models sent

by each branch to gain an overall view of the company as a whole. Giannella *et al.* (2004) state two main advantages of DDM using ensembles. The first advantage can be obviously seen when the local model is much smaller than the local data: sending only the model thus reduces the load on the network and the network bandwidth requirement. The second one is that sharing only the model, instead of the data, gains reasonable security for some organizations since it overcomes issues of privacy.

2.5 Evolutionary Learning

The previous three sections have summarized the problem areas: what data mining is, what classification is, and issues with data streams and distributed data mining. The next three sections summarize some techniques that are investigated in this thesis as potential solutions to these problems, beginning with evolutionary learning.

2.5.1 Genetic Algorithms

Inspired by biological evolution, Genetic algorithms (GAs) are a stochastic population-based technique for search and optimization. A solution is encoded in GAs by one or many chromosomes in the population. The *survival of the fittest* (Darwin, 1859) of Darwin's theories of evolution is a key concept used in the system, aiming at preserving best individuals through the evolution. In order to apply this concept computationally, one needs to distinguish inferior solutions from good solutions, encapsulating the distinction in an appropriate fitness function. The fitness function plays an important role to guide the selection process so that good solutions are chosen over inferior ones for recombination.

GAs differ from other conventional search techniques in four ways (Goldberg, 1989): (i) they work with a coding of the parameter set instead of the parameters themselves; (ii) they search for a population of solutions, not a single solution; (iii) they evolve with the help of an objective function; (iv) they use probabilistic rules instead of deterministic ones.

A typical evolutionary algorithm is summarized in Algorithm 2.

```
Initialize the population;
Evaluate individuals in the population;
repeat
    Select parents for recombination based on their fitness;
    Apply genetic operators to generate offspring;
    Form a new population by offspring with partial/whole current
    population;
    Evaluate individuals in the population;
until the termination conditions are met ;
```

Algorithm 2: An evolutionary algorithm

As pointed out by Goldberg (1989), one of the main strengths of GAs is that they can perform well for problems with complex fitness landscapes, which are noisy; change over time; and have many local optima. His recent study (Goldberg, 2002) showed that selection and mutation contribute to the improvement of the population, while selection and recombination are sources for innovation in the population. The first two operators together work as a hillclimbing technique, in which the mutation searches through the neighborhood of the current solution and the selection will accept only those good ones, thus moving towards to a better set of solutions. The last two operators work together to invent new solutions in the population.

Another strength of GAs is their ability to perform global search with little prior knowledge. However, GAs also have several disadvantages including: (i) a genetic search is comparatively slow due to its requirements to evolve a population of solutions before they can be used; (ii) stochastic search techniques using genetic operators (e.g. selection, mutation, crossing–over) normally require several runs with different seeds.

2.5.2 Evolutionary Learning Classifier Systems

The use of GA to evolve a set of rules was first introduced by John Holland and his colleagues in 1970s (Holland, 1975), (Holland and Reitman, 1977) as a cognitive model called the learning classifier system (LCS). In this framework, the knowledge

(experience) of a particular person (cognitive entity) is viewed as a set of rules that are evolved over time by interaction with the environment. Beside using natural selections and genetics as other GA applications to evolve a population, the knowledge about the problem which is represented by a set of rules is accumulated in real time reflecting their competence with others for survival under certain conditions of the environment. In this first version of LCS, GA was described as a primary learning system and the credit assignment system is a second one. Each rule is associated with a *strength*. The credit assignment adjusts rules' strength through backwards averaging. The first real-life application of LCS was implemented by Goldberg (1983) to simulate the gas pine-lines problem.

Since each environmental state might match with more than one rule in the population, how payoff can be distributed between those rules has become a serious problem. Holland (1986) developed a "bucket brigade" algorithm for solving this problem by distributing the payoff to those rules actively involved in prediction. Over time, those stronger rules are more likely to fire in the decision making problem.

Goldberg (1989) mentioned in his book a wide range of areas where LCS is investigated such as biology and medicine; business; computer science; engineering and operations research; machine learning; parallel implementations; and social sciences. Some impressive examples of LCS applied in real-life applications are control vision systems (Wilson, 1985), classification of letters (Frey and Slate, 1991), Wisconsin breast cancer diagnosis (Wilson, 2000), free flight traffic environments (Chen *et al.*, 2007), intrusion detection (Shafi *et al.*, 2007), to name a few.

Recently, many researchers have started to compare LCSs with other conventional techniques. An investigation on a benchmark synthetic test-bed, the Monk's problem, by Saxon and Barry (2000) showed that LCS's performance is at least as good as traditional machine learning techniques.

Moreover, studies on real-world problems also confirmed that LCSs are competitive for data-mining problems in comparison to other non-evolutionary learning algorithms in terms of predictive accuracy. Bernadó-Mansilla *et al.* (2002) com-

pared LCSs with 0-R, IB1, IBk, NBa, C4.5, PART, and SMO on 15 data sets. Dixon *et al.* (2001) applied LCS to twelve data sets and also found that LCS has the potential to be a powerful data mining tool. Researches in (Butz, 2004) and (Butz, 2006) compared LCS with C4.5, the Naive Bayes classifier, PART, the instance based learning algorithm with one and three nearest neighbor settings, and the support vector machine on 42 data sets. The statistical test on accuracy in these studies showed that LCSs outperform the other learners in some data sets, while being themselves outperformed in the other data sets.

2.5.3 Knowledge Discovery

According to Bull and Kovacs (2005) the ability of LCS to solve complex real-world problems is clear. However, the accuracy is not the only concern in data mining, but the discovery of knowledge is also important.

An early work by Holmes (1996) in an epidemiologic surveillance domain demonstrated that LCSs can be effective for both predicting and describing a rapidly evolving phenomenon, such as the occurrence of a certain epidemic disease. Furthermore, LCSs have successfully been employed in (Arthur *et al.*, 1996), (Schulenburg and Ross, 2001) to model artificial stock markets, which requires rapid adaptation to changing in the market situations.

An alternative study by Ferrandi *et al.* (2003) showed that LCS is able to mine interesting patterns, which can be used for the analysis of data from Hardware-Software Codesign applications. The obtained population of classifiers represents high level and accurate knowledge about the cost function (the interrelationships among hardware-software components). The authors suggested that the information can be used to guide the search for the best partitioning or to provide knowledge for a human designer.

Recently, Kharbat *et al.* (2007a) confirmed that LCSs can be more effective than C4.5 in terms of knowledge discovery. A complete knowledge discovery process using LCS is discussed in this paper including data preparation, knowledge exploration, and knowledge compactness. In this study, a rule compactness ap-

proach is added to LCS to gain better understanding of the generated rules and their underlying knowledge. The analysis showed that LCS has obtained a set of rules which is more informative and qualitatively useful than the one obtained from C4.5.

2.5.4 Michigan vs. Pittsburgh Systems

Work on LCS can be divided into one of three categories: the Pittsburgh (Smith, 1980) (De Jong *et al.*, 1993), the Michigan (Holland, 1975) (Goldberg, 1983), and the hybrid approaches. The major difference between the Michigan and Pittsburgh approaches is in the encoding of genetic chromosomes. In the Pittsburgh approach, each individual is a set of rules representing a complete solution to the learning problem. In contrast, an individual of the Michigan approach is a single rule that represents a partial solution to the overall learning task. Thus, the Michigan and the Pittsburgh systems are quite different approaches to learning.

The Pittsburgh systems that can be efficiently used for machine learning and data mining techniques include GALE (Llorà, 2002) and GAssist (Bacardit and Garrell, 2007). Examples of the Michigan system that have been confirmed to work for data mining include XCS (Butz, 2004) and UCS (Bernadó-Mansilla and Garrell-Guiu, 2003).

Several studies have conducted a comparison of the two approaches on several classification problems in order to determine circumstances that one approach would perform better than the other. Bernadó-Mansilla *et al.* (2002) compared the learning performance between XCS (a Michigan system) and GALE (a Pittsburgh system). The study found no significant difference in the predictive accuracy of both systems on sixteen data sets even though two different rule sets were produced. However GALE required a longer training time than XCS.

Another study of Bacardit and Butz (2007), conducted on XCS and GAssist, also suggested a comparative performance in terms of predictive accuracy. The population size of the best individual of GAssist is much smaller than the one of XCS. However, since GAssist maintains a population of individuals, the overall

number of rules in this system is actually similar to the number of rules of XCS. However, GAssist is observed to run faster than XCS due to its parallel learning, and also because XCS learns each instance one by one.

Both approaches are suitable under certain conditions, and the question of which approach is the best depends largely on the form of problem. The popular view in general is that the Michigan approach is more useful in an online, real time environment, whereas the Pittsburgh approach is usually applied for off-line learning problems (Butz, 2004).

2.6 Michigan Classifier Systems

2.6.1 Overview

In Michigan-style LCSs, a set of syntactically simple string rules (called the population of classifiers) is evolved cooperatively that together guide the system to achieve some tasks in an arbitrary environment. A classifier contains two main components: a rule and its associated parameters. One of the most prominent parameters is the fitness, which rates how good a classifier is with respect to others in the population.

A rule is represented in the stimulus-response (*i.e.* condition-action) formula. Those rules are potentially symbolic, which can be easily interpreted by a human. Traditionally, the condition is encoded in ternary (0,1,#) and the action is encoded in binary. The # symbol is employed in the condition to enforce the generalization in the population. This symbol is also called *don't care*, which indicates that the particular position of the condition can match with any corresponding states of the environment. The structure of a rule is

$$< condition >:< action >$$

For example: a rule *#011:1* matches both an input *0011* and an input *1011*.

LCSs employ two biological metaphors to accomplish the task as shown in

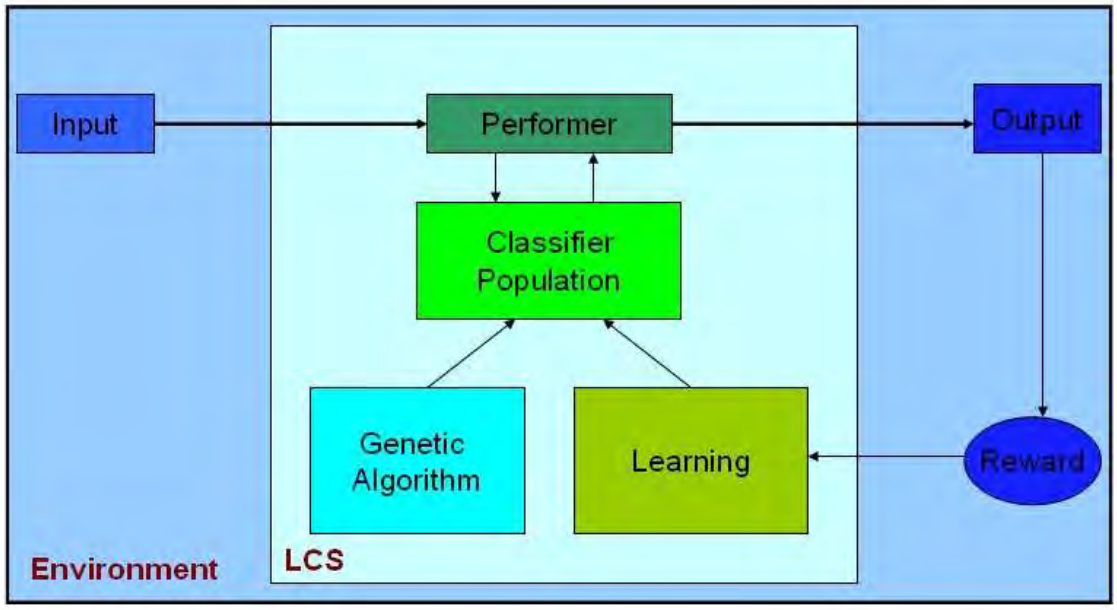

Figure 2.7: A simple structure of a learning classifier system

Figure 2.7. The evolutionary component plays a key role in discovering and introducing novel and potentially useful rules into the population. This function works as a search technique for the system. Unlike conventional evolutionary algorithms, two parents might be chosen at each time step from the population/niche. The recombination and mutation are carried out on these parents and two new offspring are created and inserted in the population. Offspring inherit the fitness from their parents.

The learning component, on the other hand, is responsible for updating classifiers' parameters by assigning credits to classifiers based on their contributions to execute some tasks and the feedback from the environment. Classifiers in the population are re-evaluated frequently in order for their parameters to reflect accurately their real qualities. Based on these parameters, the system will nominate a classifier for the GA or will decide which one to delete.

Thereby, the learning component will guide the search by the evolutionary component to move towards a better set of rules. The two components interact with each other and together contribute to the success of LCS.

2.6.2 Fitness Update

Many LCSs traditionally rely on the *strength*, the magnitude of the reward it received. The strength-based fitness serves twofold: a predictor of the future payoff and also a guider of the evolutionary component. There are many ways to distribute the reward for individual classifiers, the *bucket brigade* algorithm introduced in (Holland, 1985) is one of the most common early methods.

In this approach, the reward is paid to a number of sequentially acting classifiers through their local interactions as an information trading. Each classifier maintains a net worth called the *strength* and makes a bid proportional to its *strength* once it is matched. Classifiers in the population have the right to buy and sell the information and a chain of classifiers is formed from information manufacturer (the environment) to information consumer (the final outcome).

However, Wilson and Goldberg (1989) suggested two primary weaknesses of this technique: (i) it is difficult to maintain long bucket brigade chains, and (ii) those chains are difficult to generate in the first place.

ZCS (Wilson, 1994)(Bull and Hurst, 2002) simplifies the original framework for better performance and understanding. However, Cliff and Ross showed that ZCS can suffer by two main problems: over-general classifiers and greedy classifier creation (Cliff and Ross, 1994). The first problem refers to over-general classifiers which match in multiple states which might be correct or incorrect. Those classifiers, which match correctly in many states, might end up with a high fitness. It is not reliable to use for prediction in the system in areas where they overgeneralized. Second, classifiers with high payoff levels tend to multiply faster in the population. This might create gaps in the covering map, which results in poor performance. Third, it challenges the system to evolve a population of general but accurate classifiers as an overgeneralization problem.

Some researchers proposed the use of the fitness sharing technique to prevent these problems (Horn *et al.*, 1994). However, the sharing makes the meaning of the strength become unclear to GAs in order to distinguish accurate classifiers from inaccurate ones.

The accuracy-based fitness was then introduced in (Wilson, 1995) and has captured most attention from researchers in the field. Its important improvement is that the fitness is based on the accuracy of the predictive reward with regards to the actual one. By using the accuracy of the predictive reward instead of the reward itself, this model can overcome the previous problems. It was later shown that the accuracy-based fitness is able to achieve better generalization, smaller population sizes, and a complete mapping from inputs and actions to reward predictions (Kovacs, 1997) (Wilson, 1998) (Kovacs, 2000).

Mutual information based fitness is another study that utilizes mutual information for fitness feedback (Smith and Jiang, 2007b), (Smith and Jiang, 2007a). The mutual information is calculated from the probability bivariate distribution over the matched/unmatched condition of each classifier. The preliminary results of the proposed system, MILCS, show a slower convergence in comparison to XCS. However, MILCS is able to produce a smaller and more explanatory rule set.

2.6.3 Learning Approaches

Learning can take many forms: reinforcement learning, supervised learning, etc. The eXtended Classifier System (XCS) (Wilson, 1995) (Wilson, 1998) is the first accuracy-based classifier system. It was designed for reinforcement learning. XCS works for both single- and multiple-step tasks. Since this thesis concentrates on classification, the discussion is limited to single-step environments.

UCS (supervised classifier system) (Bernadó-Mansilla and Garrell-Guiu, 2003) is a derivation of XCS that specializes on supervised environments. The accuracy is computed differently as the proportion of correct classifications with respect to the number of matches.

Since the author is working on classification, UCS was chosen as the baseline LCS learner. However, UCS preserves the principal features of XCS such as a fitness based on the accuracy, a niche GA, generalization mechanism, etc. Moreover most works are tested on XCS. Therefore, it is worthwhile to discuss both UCS and XCS in order to have in depth understanding of the underlying system.

2.6.3.1 Reinforcement learning - XCS

The full description of XCS is discussed in (Butz and Wilson, 2001). The feedback from the environment is given to XCS after each time step in the form of a reward to indicate how good was the prediction.

A single strength parameter is replaced by three new ones: prediction P, prediction error ε, and fitness F. The prediction parameter estimates an average payoff received by the classifier when it is fired by the system. The prediction error parameter measures an average error in the prediction parameter. The fitness parameter is an inverse function of the prediction error, which is the accuracy of the classifier to predict the receiving payoff.

A classifier in XCS is a macro–classifier, which contains a distinct combination of the *Condition* and *Action* parts in the population. Whenever a new classifier is introduced, the population is scanned to see if the new classifier already exists. If so, the new classifier is not added to the population, but a *numerosity* parameter of its copy in the population is incremented by one; otherwise, the new classifier is added to the population and its *numerosity* parameter is set to 1.

The *experience* parameter indicates how often the classifier has appeared in the match set; over time, this reflects its generality.

Given an input, the match set $[M]$ is formed as a collection of classifiers in the population whose conditions match the input. The system then forms a system prediction set $[PA]$ for each action that appears in $[M]$ using a fitness–weighted average of the predictions. The action with the largest prediction is selected and exported to the environment. The action set $[A]$ is then formed of the classifiers in $[M]$ that have the selected action.

The fitness is updated during the exploration phase. The update is based on the predictive accuracy of the received reward instead of the reward itself from the environment. After receiving the reward R from the environment, XCS updates the values of prediction P, prediction error ε, and fitness F parameters of each classifier

appeared in the current action set $[A]$:

$$P \longleftarrow P + \beta(R - P) \tag{2.5}$$

where $\beta(0 < \beta \leq 1)$ denotes the learning rate. The prediction error ε is updated as:

$$\varepsilon \longleftarrow \varepsilon + \beta(|R - P| - \varepsilon) \tag{2.6}$$

The fitness F is updated according to the current relative accuracy k' of the classifier as follows:

$$k = \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha(\varepsilon_0/\varepsilon)^v & \text{otherwise} \end{cases} \tag{2.7}$$

$$k' = \frac{k \times num}{\sum_{cl \in [A]} k \times num}$$

$$F \longleftarrow F + \beta(k' - F) \tag{2.8}$$

where k is the accuracy, k' is the relative accuracy, num is the *numerosity* field of the classifier; α and v are constants controlling the rate of decline in the accuracy k when ϵ_0 is exceeded.

XCS executes GAs in niches defined by the match set. Later, Wilson makes it to the action set (Wilson, 1998). This modification helps GA to choose parents not too different from each other, and therefore it has a higher chance to obtain better offspring. During the selection process, two parents from the action set $[A]$ are selected with probability proportional to their fitness. Two offspring are generated by reproducing, crossing–over, and mutating the parents. Parents continue to stay in the population competing with their offspring. If the population size is less than a certain number, offspring are inserted into the population; otherwise, two of the most inaccurate classifiers are deleted from the population before the offspring can

be inserted.

The principle of XCS's evolution was first outlined by Wilson's *generalization hypothesis*, which stated that the interaction of accuracy based fitness and the use of a niche GA could result in evolutionary pressure toward classifiers that are not only accurate but also maximally general (Wilson, 1995). An example of the accurate, maximally general classifier corresponding to the inputs of 000000, 000001, 000010, 000011, 000100, 000101, 000110, 000111 is 000###. He also showed experimental results that support his hypothesis. Later, Kovacs investigated this hypothesis in more details and suggested that XCS is able to evolve accurate, complete, and minimal representations in boolean function problems (Kovacs, 1997).

Butz *et al.* (2003b) took a further step to investigate Wilson's hypothesis and also lay out a fundamental theory of XCS. In this reference, several evolutionary pressures are present including the set pressure, the mutation pressure, the deletion pressure, and the subsumption pressure. An equation of the set pressure, to demonstrate an interaction between the deletion pressure and evolutionary pressure, along with a number of experiments confirm Wilson's hypothesis.

2.6.3.2 Supervised learning - UCS

Since UCS is a supervised learner, a desired class/outcome accompanies the input. A *correct set [C]* is formed, containing those classifiers in *[M]* that have the same *action* as the input. If *[C]* is empty, *covering* is applied, where a classifier that matches the input is created and assigned the same outcome as the input. UCS is an incremental learner, where knowledge is updated as more data becomes available. All parameters of the classifiers in *[M]* are revised for each training instance, reflecting the system's response to new knowledge.

UCS removed the fitness-relative adjustment of XCS. The fitness of classifiers is based on their accuracy, which is computed as the ratio of correct classifications by the classifier to the number of times the classifier has been in the match set:

$$acc = \frac{N_{correct}}{N_{matches}} \tag{2.9}$$

The fitness is computed as a function of accuracy:

$$F = (acc)^v \tag{2.10}$$

where v is a predefined constant. The selection purely based on error (such as the one of UCS) is able to solve similar boolean function problems as normal XCS, but requires less parameters and depends less on the learning rate.

Unlike XCS with the relative fitness sharing, the fitness of UCS initially is calculated based on the real accuracy of the classifier without the fitness sharing. Later Orriols-Puig and Bernadó-Mansilla (2006) proposed fitness sharing in UCS. Results showed that fitness sharing is beneficial in UCS, especially with class imbalances. However, a study by Butz *et al.* (2007) showed that fitness sharing is actually not necessary in the LCS/XCS framework. The study in this thesis will be based on the basic UCS without fitness sharing.

Similar to XCS, the niche GA in UCS also create the generalization pressure to push the population towards a set of maximal general and accurate classifiers. GA is invoked in *[C]* if the average time since the last GA activation of classifiers in *[C]* surpasses a user-defined threshold. Two parents are selected from *[C]* with a probability that is proportional to their fitness. Two offspring are generated by reproducing, crossing–over, and mutating the parents with certain probabilities. Offspring are inserted in *[P]* if they are not subsumed by the parents. If the population size hits a predefined limit, some classifiers are removed by voting within the population.

UCS inherits the generalization technique from XCS, which mainly applies the GAs on niches and deletion happens in the whole population. UCS replaces the action set by the correct set in order to take full advantage of supervised learning. The correct set *[C]* contains all classifiers in the match set *[M]*, which predict correctly the outcome of the input. The selection operator occurs in the correct set, which selects two best classifiers for crossing-over and mutation. The deletion is to choose a classifier from the whole population in a similar way as XCS.

Bernadó-Mansilla and Garrell-Guiu (2003) showed that UCS is able to achieve

comparable accuracy as XCS on real world domains. Moreover, experiments on two artificial data sets show that UCS converges faster and also obtains smaller population size in comparison to XCS.

2.6.3.3 The underlying difference between XCS and UCS

XCS aims at developing a complete and accurate mapping of the problem space through efficient generalization (Bull and Kovacs, 2005). Since the fitness of XCS is based on the accuracy of the prediction, the GA is responsible for searching for classifiers that can predict accurately. Those classifiers can belong to two opposite categories: correct and incorrect classifiers. The correct classifiers are those leading to the maximum payoff in return if they are fired. The incorrect classifiers, on the other hand, would result in the lowest payoff. Both classifiers are preserved in the population of XCS as long as they predict consistently and correctly the payoff that they will receive.

UCS, on the other hand, tends to form the best action map, which consists of only high-rewarded classifiers (Bernadó-Mansilla and Garrell-Guiu, 2003). The fitness of UCS is based on the actual accuracy of the classifier, therefore only correct classifiers can survive in the population. Incorrect classifiers will receive low fitness and therefore would not be part of the GA process and also have more chance of deletion than correct ones.

The classification problem is normally a single step problem with two payoff levels: a maximum payoff for correct predictions, and a minimum payoff for incorrect ones. In this case, a set of correct rules in the best action map would be more favoured than a set of both correct and incorrect rules in the complete action map as incorrect rules have no use for classification. Bernadó-Mansilla and Garrell-Guiu (2003) have laid out five different aspects when comparing between two maps as follows:

- Population size: In data mining, a problem with high numbers of attributes and classes is quite common. If a problem has n classes, the complete action map would be as much as n times larger than the best action map. The more

classes and attributes the problem has, the bigger the complete action map becomes. It is obvious that the complete action map is normally much larger than the best action map. Therefore, evolving a complete action map requires higher population sizes and more computational resources than a best action map.

- Exploration: Since the complete action map is much bigger than the best action map, a longer time would be needed to search completely.

  Others believe that the existence of incorrect rules would avoid exploring them repetitively and also would be an advantage in the exploitation phase to guide the system not doing things (Kovacs, 2002).

- Complexity: The larger the population is, the greater the number of cycles required for learning. Therefore, the evolution of complete action maps may require longer time than best action maps.

- Generalization: the complete action map might generalize better than the best action map.

- Changing environment: Some researchers argue that the complete action maps would help the system to recover faster than the best action map when the environment changes abruptly (Hartley, 1999). It is because some incorrect rules might become correct after the change. Therefore the system does not need to discover the rule but only has to adjust the parameters. If the magnitude of change is small or moderate, the system can recover quickly, otherwise it would take much longer time to recover since XCS needs to delete inaccurate classifiers before adding a new one (Dam *et al.*, 2007).

2.6.4 Adaptive Parameters

Traditional XCS with a ternary representation requires around 20 parameters to control the learning process (Butz and Wilson, 2001). The number of parameters is reduced to 12 in UCS. Some of them are very sensitive to the overall performance

of the system. In XCS, the most sensitive parameters relate to the learning rate, mutation rate, population size, and probability of covering. UCS, on the other hand, does not require the learning rate for updating the knowledge.

Several researchers have taken efforts to create self-adaptive parameters for XCS/UCS in order to avoid the effect of initial parameter setting. An early study was carried out by Hurst and Bull (2001), that made the discount factor and the tax rate of ZCS adaptive. Each classifier has its own values for those adaptive parameters. The *enforced co-operation* method is used to adapt these parameters in order to prevent classifiers selfishly adapting their parameters. However, these parameters did not respond to changes in the environment.

Another study by Bull and Hurst (2000) made the mutation rate vary as in evolutionary programming (Meta-MP) in (Fogel, 1992). In this approach, each classifier has its own mutation rate, which is transferrable from parents to offspring. After getting the value from the parent, the mutation rate of the offspring is re-calculated using a Gaussian distribution before being used. Their study indicated that an adaptive mutation rate in ZCS responded well to environmental change. A similar study on XCS also confirmed the benefit of an adaptive mutation rate (Hurst and Bull, 2002).

2.6.5 Ensembles of Classifiers

Several researchers are working on applying evolutionary algorithms for ensemble learning (Guerra-Salcedo and Whitley, 1999), (Jong *et al.*, 2004). However, in traditional ensemble learning, the members of the ensemble are usually trained on the same datasets. Moreover, the training is usually done in batch off-line mode.

In the early days, Dorigo (1993) developed ALECSYS, which is a single LCS parallelized on transputer network through parallelizing the CFprocess (i.e. the process for creating the match set). Each CFprocess responded for equal number of classifiers over N CFprocesses. The higher N is, the more intensive the concurrency is. ALECSYS was then extended to include a number of LCS in a hierarchical structure in (Colombetti *et al.*, 1996). Level 0 receives an input from the environment

and the other levels receive inputs from the immediate predecessor level.

A more specific implementation of classifier systems for distributed environments was examined in (Bull *et al.*, 2004), where a distributed control system for traffic signal was developed. The system simulates a distributed environment where each traffic light is operated by a classifier system and the time needed for an object to travel is investigated.

An alternative study by Gao *et al.* (2005) also found that an ensemble of LCSs is able to improve the generalization. They employed the bootstrap technique for creating different bias in individuals.

Bull *et al.* (2005) presented the idea of rule sharing in ensembles of LCS. Their system consisted of 10 LCSs and was tested on the 20-bit multiplexer problem. The prediction of the system is computed by majority voting among the outputs from the 10 LCSs. The study found that the ensemble framework improves the performance, particularly as task complexity increases. A rule migration technique helps to improve the performance of the ensemble further. Their later study Bull *et al.* (2007b) also confirmed these findings. Moreover, they showed a better than linear speed-up for the 70-bit multiplexer problem through niche triggered sharing where each of the 10 nodes learns individually on effectively randomly sampled data. This sharing scheme is shown to be better than a global rule-base scheme.

The first attempt to use LCS in heterogeneous distributed environments was investigated by Gershoff and Schulenburg (2007). In their framework, called CB-HXCS, the environment is partitioned vertically into smaller subspaces. Each subspace is handled by a team of XCS through voting. The study found that CB-HXCS is able to get a similar degree of accuracy as one XCS. Moreover, less data is required to solve classification problems.

2.6.6 Knowledge Representation

The rule-based population of LCS is one of the most common representations as the condition-action format is straightforward for a human to understand. Tradi-

tionally, the condition is denoted by ternary alphabets (0,1,#), whereas the action is represented in binary. This ternary representation is only applicable in binary problems. Other representations have been introduced in LCS for solving more complex problems in the real-world. One direction is to develop new representations for the condition in order to improve the decomposition of the problem. The other direction is to modify the action.

2.6.6.1 Classifier Conditions

According to Stone and Bull (2003) and Llorà *et al.* (2005a), rule encoding may introduce bias to LCS that would affect the scalability of learning maximally general and accurate rules.

One of the earliest works of alternative encoding is the interval-based representation introduced by Wilson (2000) for handling continuous-valued instances. The proposed representation, called the center-spread, takes the form (c_i, s_i) where $c_i, s_i \in [p_{min}, q_{max})$, p_{min} and q_{max} are the lower and upper bound of an interval. c_i is the center of the interval and s_i is the width of the interval from the center.

Wilson (2001c) later proposed the min-max representation (MMR) for integer values. An interval predicate in the min-max representation is represented as (l_i, u_i) where l_i and u_i are the minimum and maximum bounds of the interval. This representation is well known in traditional machine learning but was new for LCS.

Stone and Bull (2003) argued that the MMR can also be applied for real-valued inputs and they extended it to what they called the Unordered-Bound Representation (UBR). Stone and Bull proposed this representation to fix the problem of the MMR by allowing the interval to be represented as (p_i, q_i) where either p_i or q_i can be the maximum or minimum bound. UBR has solved the problem of generating infeasible intervals but raises another challenge: the unordered bound representation. We pointed out in (Dam *et al.*, 2005a) that this representation may violate Holland's schema theorem (Holland, 1975) and the building block hypothesis. We explained that LCS depends heavily on the evolutionary learning process to drive classifiers of low accuracy to those of high accuracy (Butz *et al.*, 2003a). However,

the UBR changes the semantics of the chromosome by alternating between the min and max genes; that is, in one generation the genes representing the lower bound of an interval can become the genes representing the upper bound of an interval in the following generation. This discrepancy will generate a challenge to building blocks. We introduced the Min-Percentage representation which maintains the semantics of the genes over the whole evolutionary run.

The min-percentage representation (MPR) (Dam *et al.*, 2005a) was introduced to overcome this problem by maintaining the semantics of the genotype. Similar to other approaches, each attribute in the *Condition* of the classifier is represented as an interval predicate in the form of (m_i, p_i) where m_i is the minimum bound of an interval in phenotype and p_i is the proportion of the distance between the minimum and maximum bound of an interval and the distance between the upper bound and minimum bound of an interval of the phenotype. The transformation from genotype (m_i, p_i) to phenotype (l_i, u_i) is taken as follows:

$$l_i = m_i \tag{2.11}$$

$$s_i = p_i * (p_{max} - l_i) \tag{2.12}$$

$$u_i = m_i + s_i \tag{2.13}$$

where s_i is the distance between the lower and upper bound.

A study by Butz (2005) found that LCS with the hyper-rectangular condition suffered from problems with nonlinear decision boundaries. The experiments showed that learning takes longer and the consequent population size is much higher on oblique data sets. Butz (2005) conducted an experiment in which the hyper-rectangular conditions were replaced by hyper-spheres or hyper-ellipsoids. The study also confirmed that LCS is a general and flexible learning system since the condition structures can be easily changed to other forms without affecting the overall performance. LCS with the kernel-based condition is able to obtain better rules over complex problems. Their later study (Butz *et al.*, 2006) analyzed the evolving ellipsoidal structures, which showed that XCS is able to stretch and rotate

the evolving ellipsoids according to the shape of the underlying function.

Furthermore, Lanzi and Wilson (2006) represented the conditions by sets of points in the problem space. The convex hull, the smallest convex region enclosing those points, defines the boundary of classifiers. A number of points needed in each classifier can be either fixed or variable. Their experiments showed that the proposed representation helps to converge faster in comparison to the interval representation. This representation is more flexible since the conditions can approximate any shapes depending on the problem. However, extra computation cost is added to build convex hulls during the matching process.

Alternatively, other representations include the messy (Lanzi and Perrucci, 1999a), S-expressions (Lanzi and Perrucci, 1999b), fuzzy representation (Bonarini, 2000), and first order logic (Mellor, 2005).

This thesis focuses on the interval representation since this is the most widely used in LCS due to its simple but very effective representation.

2.6.6.2 Classifier Actions

A pioneer work is made by Ahluwalia and Bull (2009) to introduce a general form of a traditional LCS with a computed action in genetics programming.

Wilson later applied the computed action in XCS for the function approximation problems (Wilson, 2001b), (Wilson, 2002). Later Wilson (2004) applied XCSF to a single step problem with continuous payoff functions with respect to the learning space. The system, called XCS-LP, obtains high accuracy with smaller population compared to XCS. A more recent study of Lanzi *et al.* (2007) showed that XCSF is sensitive to the range of classifier inputs because of its weight's update mechanism.

Another research direction is to build an anticipatory classifier system by replacing an action by a function. The neural anticipation is added to each classifier in the population of the neural-based XCS (X-NCS) (O'Hara and Bull, 2005). The additional anticipatory neural networks are trained in a supervised mode based on

the current input and the next input that is encountered after the classifier action has been performed. Related study by Bull et. al. proposed XCSAM using the anticipation mapping for XCS based on an array of perceptron array or neural networks (Bull *et al.*, 2007a). XCSAM can provide as accurate anticipatory predictions as X-NCS, while requiring a smaller population.

A similar but separate series of studies by Lanzi and Loiacono proposed XCS with Computed Action (XCSCA) to tackle a problem with a large number of discrete actions (Lanzi and Loiacono, 2007). XCSCA is restricted to supervised learning problems and borrows ideas from UCS, XCS, and XCSF. Each classifier in this system consists of a condition and a set of parameters. The action is not included as it is computed on the fly for each instance based on the *action functions*. The authors suggest four different action functions for computing the action: (1) a constant action function which represents the action by one parameter w_0. This parameter is changed every time the classifier enters the match set during training; (2) a perceptron function which feeds the input through a simple perceptron, the weights associated with the perception are updated accordingly; (3) the sigmoid function which extends the perceptron with a sigmoid function; (4) a complete neural network. An initial investigation found that XCSCA with neural networks requires less classifiers in comparison to other methods. Moreover XCSCA is slightly faster than a single neural network in terms of number of learning iterations, but slower in terms of computation time as it has multiple networks.

Loiacono *et al.* (2007) extended XCSCA to the Support Vector Machines (SVM) for computing the classifier action. The proposed system, named XCSCAsvm, replaces the action function by the SVM with a set of support vectors. Each time the classifier is in the match set, its SVM is trained from scratch using the actual set of support vectors and the latest available sample. To overcome the problem of an indefinitely large set of support vectors, XCSCAsvm removes the oldest sample from the memory. The study shows that XCSCAsvm reaches the optimal performance faster than XCSCA. However, the computational complexity of XCSCAsvm is more expensive than XCSCA due to the need to retrain SVM from scratch at each time step.

2.6.6.3 Other Representations

Bull and colleagues proposed to use neural networks (NNs) in LCSs (Bull, 2002) (Bull and O'Hara, 2002). In their framework, called NCS, the *condition-action* of classifiers is replaced by a fully connected multi-layer perceptron (MLP). The classifier, in this framework, consists of an MLP and a set of parameters.

Neurons of each MLP at the output layer are responsible for each possible outcome (classification problems). An extra neuron is added to the output layer for signifying its membership of a match set *[M]*.

For each input from the environment, all classifiers (neural networks) in the population are required to process independently. The input values are fed forward through hidden layers by sigmoid transfer functions in each network. If the extra neuron at the output layer has the highest activation value, the classifier will not form part of the match set *[M]*. Otherwise, it will belong to the match set, proposing the action corresponding to the output neuron with the highest activation value. Action selection in NCS is performed in the match set using a simple roulette wheel selection policy based on fitness.

NNs in NCS are built around the neurobiological theory of neural constructivism (Quartz and Sejnowski, 1997). NCS starts with a small network. An appropriate structure is then added through the learning process, particularly through growing/pruning dendritic connectivity, until some satisfactory level of utility is reached. The use of self-adaptive constructivism helps the realization of appropriate autonomous behaviour. Experiments showed that NCS is able to solve several maze problems (Bull and O'Hara, 2002) and to learn appropriate structure in simple robotics applications (Hurst and Bull, 2004).

In a later study, back-propagation is used in conjunction with the genetic algorithm to evolve the population (O'Hara and Bull, 2007). In the exploration phase, the weights of MLPs in the match set are updated using this function. Other operators such as the activation of GA and the parameters' update are carried out in a similar way as XCS.

However, by replacing a rule completely by a neural network, NCS lacks the main advantage of LCSs; that is, being a rule-based system that is potentially easy to understand. In many data mining problems, the ability to understand learnt knowledge can be as important as obtaining an accurate model. For instance, a company might want to profile customers' expenditures in terms of their consumption, services, location, income, season, etc. LCS will provide a set of rules drawing the relationship between those features with regards to customers' spending. Understanding their purchase behaviour might help managers to identify the best segments that influence their spending so that they can be used for future prediction as well as investment decisions.

2.6.6.4 Population Compactness

Even though LCS has been successfully applied in many problems, its main issue is the scalability of the learning knowledge. The rule-based representations always require a large population in order to cover the whole input space. When applying LCS for data mining rules in the steel industry, Browne (1999) found that many rules had similar patterns. As a result, the population becomes unnecessarily hard to interpret and slow to learn.

A study by Lanzi (2001) found that XCSL (a version of XCS using Lisp-like s-expressions) performs better than C4.5 from the point of view of a predictive data mining approach. However as a descriptive data mining, the knowledge in the data is complex and the solutions might be difficult to analyze and present. It is because the population of XCSL is quite big, which requires many pages of text. In the end, Lanzi raised a question *"are symbolic representations or general variable length representations interesting?"*

Some early work of Butz and Wilson (2001) showed that XCS scales up polynomially in problem length and exponentially in order of problem difficulty.

Llorà and Sastry (2006) recently pointed out that the population size can have a great impact on the computation time in LCS. They showed that the process of finding a set of matching classifiers (or forming a match set) in LCS is the one that

takes most of the execution time. Their experiments revealed that the matching process of small data sets with tens of attributes and few thousand of records takes more than 85% of the overall execution time, and more than 98% has been observed on bigger data sets with few hundreds of attributes and few hundred thousands of records.

There are several ways for solving this problem. The first one is to exploit the hardware for faster rule matching as in (Llorà and Sastry, 2006). Moreover, the matching process can be carried out in parallel since each matching is independent of each other. This study engaged vector instructions to perform parallel logical, integer, and floating point operations in order to speed up the matching process.

An alternative is to build a coarse-grain parallelism model for distributing the evaluation load (Llorà *et al.*, 2006). This approach utilizes several processors, each runs the same LCS algorithm. Each processor only accesses a portion of the population (a chunk), the fitness of classifiers in its chunk is shared with the rest of the processors. The population is the collection of chunks of all processors.

Another approach is to replace traditional genetic operators (crossing-over and selection) by the estimation distribution algorithm (EDA). First of all, a probabilistic model of promising rule sets is built. After that, a new set of rules is sampled from the obtained model. A study by Llorà *et al.* (2005b) demonstrated that using a compact genetic algorithm, the simplest version of EDA, in a Pittsburgh-style classifier system would help to evolve a compact population with maximally general and accurate rules. Later, Butz and Pelikan (2006) showed that XCS with EDA is able to solve challenging hierarchical problems more efficiently.

Furthermore, other researchers proposed an abstraction algorithm in LCS to produce higher order (abstracted) rules from the rule-based population (Browne and Scott, 2005). The authors suggested that higher level rules are needed in real-world data mining in order to produce more useful patterns. The process of abstraction is similar to the information processing theory in conventional machine learning. The study showed that the performance of LCS is improved. The compactness of the abstracted population is not discussed, however. Later work in this direction

(Browne and Ioannides, 2007) showed that this method is able to provide compact results and has potential for scaling up well in complex domains.

One of the most common approaches is to reduce the population size by improving the generalization. The advantage of this approach is to provide compact and accurate knowledge. This is, in fact, very prominent in data mining, where the knowledge discovery is as important as the predictive accuracy. A rule can be easily interpreted by a human when considered by itself. However, each rule in LCS represents a partial solution to the problem. In order to understand a complete solution, the whole population needs to be interpreted simultaneously. In this case, a population of a few thousand classifiers surpasses the ability of a human to understand.

Wilson suggested that the population of XCS will compact over time due to the increase in generalization (Wilson, 2001a). His study on the Wisconsin Breast Cancer database showed that XCS reaches 100% accuracy after exploring approximately 50,000 instances. Continuing the training after this point does not make a difference to the accuracy, but helps to decrease the population size. From approximately 4,000 classifiers after 50,000 instances, XCS's population size decreases gradually to a stable point of around 1,000 classifiers. Wilson suggested that it is the smallest population obtained by XCS in this problem. A question arises here is whether all of those classifiers are really necessary or they are simply an artifact of XCS?

One series of studies to compact the population started from the work of Wilson (2001a). He proposed the *Compact Ruleset Algorithm* to filter out unnecessary classifiers. This algorithm first arranges all classifiers in the population in order of their performance and generality. The next step is to find a smallest subset B of top classifiers (or best classifiers), that is able to maintain similar predictive accuracy as the whole population on the training set. The final step is to find the smallest set of classifiers in B, which is able to match to all training instances.

In the end, this algorithm is able to obtain a population of 25 classifiers from 1155 classifiers (a reduction of 97.8%) on the Wisconsin Breast Cancer problem,

whilst the predictive accuracy remains highly competitive. Several works continued in this direction (Dixon *et al.*, 2003), (Fu and Davis, 2002), (Wyatt *et al.*, 2004), (Gao *et al.*, 2006). One big challenge of those works is that selecting classifiers based on a data set, which is normally quite small, would bias towards the sample's size and data distribution. Especially, noisy data sets would mislead the compaction process and therefore end up with inaccurate subsets of the population. To overcome this problem, a study conducted by Kharbat *et al.* (2007b) proposed a clustering method to measure the similarity of attributes and features instead.

The key disadvantage of those approaches is the explicit assumption that the level of predictive accuracy on the training set needs to reach a certain level. If the system has not accumulated enough knowledge, the compaction process might eliminate potential classifiers, resulting in longer training time. Clearly, this process needs to be delayed until the population is experienced and knowledgable.

Perceptrons have been used in LCS to compact the population. Study in (Lanzi and Loiacono, 2007) has shown that a perceptron was able to reduce the population size. Chapter 4 will look further into this idea.

2.7 Artificial Neural Networks

An alternative popular method for classification tasks in the literature is artificial neural networks. This method is quite different to LCS in the sense that its knowledge is a black box to a human. Its model is very compact, however.

The idea of artificial neural networks, inspired by the human brain, was first introduced in early 1950s. Artificial neural networks, commonly referred to as neural networks (NNs), started to attract attention for further research in late 1970s and early 1980s. Haykin (1999) presented several powerful properties and characteristics of neural networks including an ability to learn nonlinearity that is suitable to model complex problems; a capability of built-in adaptivity that can respond to changes of surrounding environments, a capability of robust computation, and a compact learned model. In a survey of neural networks for classification, Zhang (2000)

noticed that NNs have emerged as a promising alternative to various traditional methods for classification as the vast research activities have been established in this field. According to Zhang (2000), a variety of real world classification problems has been solved successfully by neural networks such as bankruptcy prediction, handwriting recognition, speech recognition, product inspection, fault detection, medical diagnosis, etc.

2.7.1 Multi-layer Perceptrons

A NN is built out of a group of artificially interconnected units (neurons). One type of units is the *perceptron*, where each input to the perceptron associates with a real-valued constant, or weight, that determines the contribution of that input to the output of the perceptron. Each unit is responsible for taking a number of real-valued inputs (or outputs from other units) and produces a single real-valued output using an activation function. The activation function is used to introduce nonlinearity into the network. One of the most popular activation functions is the sigmoidal function

$$\varphi(x) = \frac{1}{1 + e^{-ax}} \tag{2.14}$$

where the coefficient a determines the shape of the sigmoid curve.

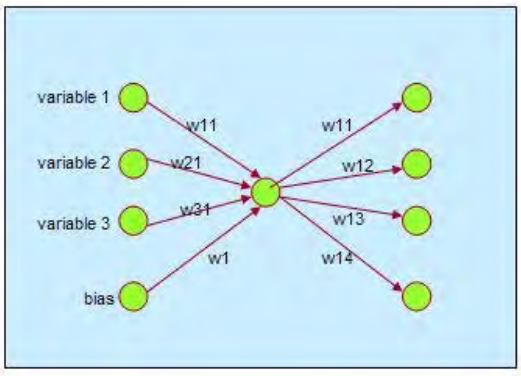

Figure 2.8: A neural network for classification – three input variables and four possible outcomes.

A NN might consist of one to many perceptrons arranged in one or many hidden layers. The number of neurons in the input layer is the same as the number

of input features. The outputs are encoded using m output neurons corresponding to m classes.

Figure 2.8 illustrates a single layer perceptron. In this example, each input is a vector of three variables and an outcome needs to be drawn from four distinct groups (classes). The decision is made based on the network's knowledge (a set of weights) as the input is fed-forward from the input layer through the network to the output layer. Once the output layer is reached, the output neuron with the highest activation value is chosen as the prediction.

Training an NN is mainly concerned with tuning a set of weights that minimizes the output error of the network. The back-propagation algorithm (Rumelhart *et al.*, 1986), which is quite similar to the gradient search, has proven successful in many applications such as handwritten characters recognition (LeCun *et al.*, 1990).

2.7.2 Negative Correlation Learning

In order to improve further the performance of neural networks within ensembles, one way is to train individuals interactively so that each member specializes on a part of the task. This method is called anti-correlation learning, which adds an additional penalty term into the error function in order to maximize the distance between all individuals in an ensemble to achieve a nice spread in the ensemble space. The negative correlation term should not have a larger magnitude than the original error function and is dimensionally consistent with the error function. Negative correlation learning (NCL) (Liu, 1999) is an approach of this type. McKay and Abbass (2001) reveal that negative correlation learning acts to push the members of the ensemble away from their mean, but not necessarily away from each other.

NCL (Liu, 1999) (Liu *et al.*, 2000) was introduced in NN ensembles to improve the learning performance in classification, regression and time-series problems. The idea of NCL is to introduce a correlation penalty term into the error function of each individual network, so that all networks can be trained simultaneously and interactively (Liu and Yao, 1997). The success of NCL was explained by Brown

(2004): training individual networks with error functions plus their contributions to overall ensemble error helps to diversify individuals in the ensemble, and therefore improves the learning performance.

Negative correlation works as follows. Given a training data set

$D = (x_1, y_1), \ldots, (x_n, y_n), \ldots, (x_N, y_N)$

we want to estimate the output $\hat{y}$ and update the knowledge of the population for each instance.

Consider the n^{th} training instance , and an ensemble of M neural networks. We determine $\hat{y}_{en}$ (the output of the ensemble on the n^{th} training instance) by averaging the outputs from the networks in the ensemble:

$$\hat{y}_{en}(n) = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i(n) \tag{2.15}$$

where $\hat{y}_i(n)$ is the output of the i^{th} network in the ensemble on the n^{th} training instance.

Negative correlation is added to the error function E_i of each network i in the ensemble before back-propagation is executed. If the mean-squared error is used, the error function of each individual network i is defined by

$$E_i = \frac{1}{N}\sum_{n=1}^{N}E_i(n) \tag{2.16}$$

$$= \frac{1}{N}\sum_{n=1}^{N}[\frac{1}{2}(\hat{y}_i(n) - y(n))^2] \tag{2.17}$$

where N is a number of training instances, $E_i(n)$ is the value of the error function of network i on the n^{th} training instance, and $y(n)$ is the desired output of the n^{th} training instance. A negative correlation term is added to the error function:

$$E_i = \frac{1}{N}\sum_{n=1}^{N}[\frac{1}{2}(\hat{y}_i(n) - y(n))^2 + \lambda p_i(n)] \tag{2.18}$$

where the parameter λ is used to adjust the strength of the penalty and p_i is a

penalty term. The purpose of the back-propagation is to minimize the error in the future by adjusting the weights accordingly. Minimizing p_i is to negatively correlate each individual's error with the rest of the ensemble. The penalty function p_i is defined by

$$p_i(n) = (\hat{y}_i(n) - \hat{y}_{en}(n)) \sum_{j \neq i} (\hat{y}_j(n) - \hat{y}_{en}(n)) \tag{2.19}$$

The partial derivative of E_i with respect to the output of individual i on the n^{th} training instance is

$$\begin{aligned} \frac{\partial E_i(n)}{\partial \hat{y}_i(n)} &= \hat{y}_i(n) - y(n) + \lambda \frac{\partial p_i(n)}{\partial \hat{y}_i(n)} && (2.20) \\ &= \hat{y}_i(n) - y(n) + \lambda \sum_{j \neq i} (\hat{y}_j(n) - \hat{y}_{en}(n)) && (2.21) \\ &= \hat{y}_i(n) - y(n) - \lambda (\hat{y}_i(n) - \hat{y}_{en}(n)) && (2.22) \end{aligned}$$

under the assumption that the output of the ensemble $\hat{y}$ has constant value with respect to $y_i(n)$. λ is a parameter used to adjust the negative term in each individual. $\lambda = 0$ means that negative correlation is not enforced in the system. The higher λ is, the more strongly the errors of the rest of the ensemble are correlated in each individual's error. This parameter is important because it controls the weight of the negative correlation term when adding it to the error function.

A single weight connected to the output layer in the network is updated by the formula

$$\Delta w_i(n) = \beta [(\hat{y}_i(n) - y(n)) - \lambda (\hat{y}_i(n) - \hat{y}_{en}(n))] \frac{\partial \hat{y}_i(n)}{\partial w_i(n)} \tag{2.23}$$

where β is a learning rate to decide on the update step.

The value of λ was initially forced to lie inside the range $[0, 1]$. Brown (2004) later showed that this constraint is not necessary. He also introduced a strength parameter γ, a function of λ and the number of networks in the ensemble:

$$\gamma = \lambda [\frac{M}{2(M-1)}] \tag{2.24}$$

In this thesis the author shall refer to the γ parameter instead of λ.

2.7.3 Generating Rules from a Neural Network

The main challenge of using neural networks in data mining is its black-box knowledge obtained from the algorithm. In order to understand the knowledge of NN, many studies have tried to trigger classification rules out of the learning model. In fact, it is possible to obtain a set of rules from the trained neural network but the process requires a special rule extraction algorithm as in (Towell and Shavlik, 1993)(Andrews *et al.*, 1995). This post-processing generally needs to run off-line and therefore cannot provide an explicit set of rules on the fly as required in many stream data mining applications.

2.8 The Emergent Questions

As we have seen, the research into LCS/UCS for data mining has mainly focused on static training data sets. Surprisingly, few works in LCS have considered concept drift (Wyatt *et al.*, 2004). This problem will be investigated in Chapter 3.

LCS/UCS has been reported as a good technique for data mining because it can get high accuracy and provide a comprehensive rule-set. A rule in UCS is represented as stimulus-response (*i.e.* condition-action). The major benefit of this representation is that the rules can be easily interpreted by a human. This can be important: in many data mining problems, the ability to understand the learnt knowledge is sometimes as important as obtaining an accurate model. However, the rule-based populations of LCS/UCS are normally so large that they grow beyond the ability for a human to understand (Lanzi, 2001) (Butz, 2005). This makes the system less efficient in terms of computational time, memory usage and comprehensibility. This is the motivation of Chapter 4, where a new representation, aiming for a population with fewer classifiers while still maintaining the predictive accuracy, is proposed.

NCS benefits from using neural networks to solve simple robotics applications. However, by replacing a whole rule completely by a neural network, NCS destroys the expressiveness of LCS. The issue of how to use neural networks in LCS without

losing too much expressiveness, is tackled in Chapter 4.

2.9 Chapter Summary

This chapter has provided a brief review of data mining classification tasks for data streams. It has been widely accepted that data mining is an effective method for triggering novel, potentially useful patterns hidden within massive amounts of data. This thesis focuses on the classification task, which attempts to develop a model from previous observed data for future prediction of values in a finite set.

Finally this chapter gives an overview of parallel learning. The review looks at how scalability can be enhanced through parallel or distributed learning. This thesis will show how these mechanisms can be used to improve the performance of LCS system.

Chapter 3

Challenges Facing UCS in Stream Data Mining: An Empirical Study

The following papers are based on this chapter:

1. H.H. Dam, P. Rojanavasu, H.A. Abbass and C. Lokan (2008) Distributed learning classifier systems. In Learning Classifier Systems in Data Mining, L. Bull, E. Bernadó-Masilla and J. Holmes Editors, Springer-Verlag, In Press.

2. H.H. Dam, C. Lokan and H.A. Abbass (2007) Evolutionary Online Data Mining: An Investigation in a Dynamic Environment. In Evolutionary Computation in Dynamic and Uncertain Environments, Shengxiang Yang, Yew-Soon Ong and Yaochu Jin Editors, Studies in Computational Intelligence Series, Springer-Verlag, Volume 51/2007, pages 153-178, ISBN 978-3-540-49772-1.

3. H.H. Dam, H.A. Abbass, and C. Lokan (2005) DXCS: an XCS system for distributed data mining. Genetic and Evolutionary Computation Conference (GECCO'2005), Washington D.C., ACM Press, 2005.

3.1 Overview

UCS is the main focus in this thesis because it is specially designed for classification tasks. Most work of UCS has assumed that the environment is static and centralized. However, the environment of data streams is likely to be dynamic and distributed in real life. The objective of this chapter is to investigate the behaviour of UCS, when confronted with three issues of stream data mining: (1) fast processing, (2) concept change (or concept drift), (3) distributed environments.

The chapter will start with an investigation on the time required for UCS to process a training instance. The faster speed UCS can perform, the better it is when dealing with data streams. The population size is an important factor that affects the processing time of UCS. It motivates the author to propose the neural representation in the next chapter.

The concept drift is then examined in UCS. The noise factor is also taken into account. It is important to notice that the noise of data streams is not fixed as it is likely to change over time. Experiments in this chapter generate the noise following the Gaussian distribution.

Finally, UCS is investigated in a distributed environment. The architectural design of a set of UCSs, called DUCS, for distributed data streams is proposed in this chapter. This framework can work in either centralized or distributed environments. In the first case, it helps to reduce the processing time in order to keep pace with high speed arriving data. In the second case, the data transmission in the system can be reduced and also a high level protection is given to the raw data.

The chapter is structured as follows. The next section provides the experimental setup used in this chapter for stream data mining. Section 3.3 discusses the processing time of UCS. Section 3.4 investigates UCS in dynamic and noisy environments. Section 3.5 proposes DUCS, the framework of UCS in distributed environments, followed by an investigation of DUCS in Section 3.6 and Section 3.7. Finally, the summary section concludes the chapter.

3.2 Methodology

3.2.1 Synthetic Data Streams

Some systems designed to handle concept change have been tested on real-world data such as spam filtering data (Delany *et al.*, 2005), US Census Bureau data (Street and Kim, 2001), and credit card fraud data (Wang *et al.*, 2003). A repository of such data sets (Blake and Merz, 1999) is maintained by the University

of California at Irvine (UCI).

The main disadvantage of those data is that they do not contain significant concept change. Thus concept changes are added artificially, and therefore it becomes an artificial problem.

Artificial problems tend to be favoured by researchers, however. They have the advantage that the researcher can control the type and rate of concept change, context recurrence, presence of noise, irrelevant features, etc.

The most popular artificial testing benchmark for concept change in data mining is the STAGGER concept (Harries *et al.*, 1998), (Kolter and Maloof, 2003), (Schlimmer and Fisher, 1986), (Widmer and Kubat, 1996). The problem contains 3 simple Boolean concepts of 3 features with a total of 120 instances and 3 cycles of change. However, this problem cannot be extended to a large scale of data. Scalability is very important as concept change mostly occurs in a stream of data.

In this chapter, we experiment on the multiplexer problem. A binary string of length $k+2^k$ is used as an input. The first k bits determine the position of an output bit in the last 2^k bits. The multiplexer problem is one of the most popular testing benchmarks in learning classifier system research in general and XCS in particular (Butz *et al.*, 2003b), (De Jong *et al.*, 1993), (Wilson, 1995).

Wilson (2000) extended the multiplexer problem to a continuous domain by introducing a real-threshold to convert a real number to a binary number. For example, assume a real number is in the range $[0, 1)$ and the real-threshold is 0.5. The real input number r will be converted to 0 in binary if $r < 0.5$, otherwise it is set to 1 in binary. The real multiplexer problem then becomes a traditional binary multiplexer problem.

The real multiplexer problem is also considered a challenging artificial problem. It can extend to large scale data. The magnitude and rate of changes, as well as noise in the system, can be controlled easily by the researcher. Unless stated differently, experiments in this chapter are carried out with the 6-real multiplexer problem.

3.2.1.1 Noisy Data

In this chapter, the noise distribution is normal. This is one of the most common distributions for noise in the literature. A parameter called *noise level* refers to how noisy the data set is.

Noise is incorporated only in the training set by flipping the class with a certain probability. For example, a noise level of 0.05 means that the flipping probability is 5% (or approximately 5 noisy instances occur in each 100 training inputs).

3.2.1.2 Concept Change

The target concepts may change at different rates depending on the nature of the application. Some concepts may change slowly, resulting in ambiguity and uncertainty in between periods of stability. Other concepts might change suddenly; new instances become no longer consistent with the current concepts of the learned model. This thesis investigates the second case; leaving the first one for future work.

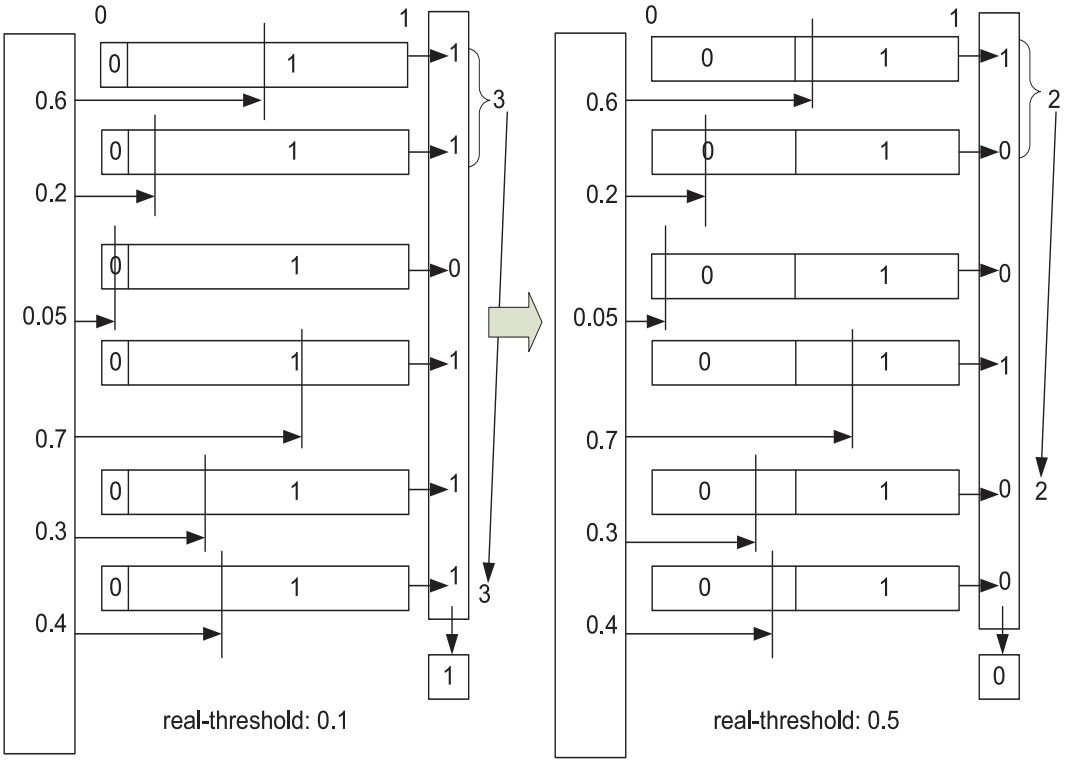

Figure 3.1: A concept change in the 6-real-multiplexer problem

The concept changes in the real multiplexer problem are simulated by changing the real-threshold used for mapping a real number to a binary number. Figure 3.1 illustrates the dynamic environments of the 6-real multiplexer problem. In this example, the concept of data is altered as the real-threshold is changed. This influences the mapping from a real number to a binary number. The concept at the continuous level is changed, but the concept at the binary level remains unchanged.

The magnitude of change (MoC) in this problem is the absolute difference of the real-threshold before and after the change. For example, if the real-threshold is changed from 0.4 to 0.5, the MoC is 0.1.

3.2.2 Experimental Setup

In this chapter, the author conducted several experiments to study UCS in stream data mining. The first experiment aims to learn the processing time of UCS in a centralized static environment.

The second experiment investigates UCS in dynamic environments by learning the impact of different MoCs on the system's predictive performance in both noisy and noise-free environments. In this experiment, a single run consists of two cycles. The real-threshold of the first cycle is set to 0.1. After 4000 iterations (each iteration includes 50 training instances and 50 testing instances) the threshold is changed. The author chooses 4000 iterations because it gives any learning algorithm enough time to accumulate full knowledge in this problem. Hence, we can evaluate accurately its adaptive ability after the concept change. The real-threshold is altered with different levels of MoC, such as $MoC = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. For example, if $MoC = 0.4$, the real-threshold of the second cycle will change from 0.1 to 0.5.

The last experiment aims to understand UCS in a distributed environment. Several issues related to the distributed environment are investigated such as the effect of different numbers of clients, the effect of different methods to combine local knowledge at the server, the traffic load, the effect of knowledge sharing, etc.

In (Dam *et al.*, 2005c), the author found out that imbalance of class distribution has an effect on the system performance. Experiments in this chapter avoid this problem, by generating data with equal distribution of each class.

All experiments presented in this paper are averaged over 30 independent runs, each has different random seed consistent in all experiments.

3.2.3 System Setup

If not stated differently, UCS is set up with the same parameter values used by Wilson (2000), Stone and Bull (2003), and Dam *et al.* (2005b) as follows: $N = 1000, \alpha = 0.1, v = 10, \theta_{GA} = 12, \chi = 1, \mu = 0.04, \theta_{del} = 20, \delta = 0.1, \theta sub = 20, p_I = 10, \varepsilon_I = 0, f_I = 0.01$.

3.3 Processing Time of UCS

In many stream data mining applications, a training instance can only be passed through a learning system once due to the high speed and the large volume of data. The first requirement for any algorithm in such an application is to learn on the fly. Moreover, the processing time of an instance is another important factor to judge the algorithm. A smaller processing time means more instances a system can handle and therefore more up-to-date knowledge it can capture in real time. A large processing time, on the other hand, indicates that some arriving instances might be discarded as the learning speed is slower than the speed of data arrival.

This section is designed to study the processing time of UCS for each data instance. The investigation starts with a theoretical analysis, followed by some experiments of UCS with a number of different population sizes and a number of different features.

3.3.1 A Theoretical Study

Let P be the number of classifiers in the population, f the number of learning features, Φ the number of time steps required before GA can be activated. Let T_p be the real time needed to update all these parameters in one classifier. Let T_c be the real time needed for one comparison.

For each training instance, UCS performs three major steps as follows.

- Evaluating the instance with respect to the current knowledge. Let T_{eval} be the time required to complete this component.
- Updating the population in light of the new information. Let T_{update} be the time required to complete this step.
- Evolving the population. Let T_{evolve} be the time required to complete this step.

3.3.1.1 Evaluation Time

Evaluation of a training instance starts by generating a match set for the instance from a population of classifiers. It involves comparing the instance against each classifier's condition in the population. Assume the interval predicate representation and particularly the min-max representation is used to represent a condition in UCS. In this case, each gene consists of two values representing the lower and upper bounds. The upper and lower bounds on the total number of comparisons (n_c) required for each classifier are $2 * f$ and 2 respectively. Let T_1 be the time required for comparing an instance with one classifier. T_1 can be estimated as follows

$$T_1 = n_c \times T_c \tag{3.1}$$

$$2 \times T_c \leq T_1 \leq 2 \times f \times T_c \tag{3.2}$$

In order to form a match set *[M]*, the instance needs to be compared with all

P classifiers in the population. Hence the cost of forming the match set is

$$T_{[M]} = P \times T_1 \tag{3.3}$$

$$2 \times P \times T_c \leq T_{[M]} \leq 2 \times f \times P \times T_c \tag{3.4}$$

If the match set is empty, the covering process is performed to create a matching classifier and insert it to the population. It is normally assumed that the covering only happens at the very beginning of the learning cycle thanks to the evolutionary component. It means the match set always has at least one classifier and the cost of covering can be ignored at this stage. The match set size $n_{[M]}$ can be

$$1 \leq n_{[M]} \leq P \tag{3.5}$$

UCS divides classifiers in the match set into a correct set *[C]* and an incorrect set *[!C]*. A classifier in *[M]* belongs to *[C]* if its action is the same as the input's class. The correct set size $n_{[C]}$ has the minimal size of 1 classifier and maximum size as the match set.

$$n_{[C]} \approx n_{[M]} \tag{3.6}$$

$$1 \leq n_{[C]} \leq P \tag{3.7}$$

The evaluation time of each instance is the time $T_{[M]}$ needed to form a match set.

$$2 \times P \times T_c \leq T_{eval} \leq 2 \times f \times P \times T_c \tag{3.8}$$

3.3.1.2 Updating Time

UCS is an incremental learner as all classifiers' parameters in the match set are updated for each training instance. All classifiers in the match set are revised after the evaluation. The parameters in each classifier required to be updated are the accuracy, fitness, experience, correct set size, etc. The update time T_{update} for

all classifiers in the match set is

$$T_{update} = T_p \times n_{[M]} \tag{3.9}$$

$$T_p \leq T_{update} \leq P \times T_p \tag{3.10}$$

3.3.1.3 Evolving Time

Evolving the population in UCS occurs if the average time since the last GA activation hits a threshold Φ, which is equivalent of the frequency of around $1/\Phi$.

The GA process starts by selecting two classifiers from the correct set with probability proportional to their fitness. However, if the tournament selection is used, the time needed for the reproduction would be similar for all UCS. Therefore we can ignore this time.

If children are not subsumed by parents, they will be added to the population. It requires a maximum of $(P-2)$ classifiers to be compared with (except its parents) in order to ensure a distinct set of classifiers. The time required to complete this job is:

$$T_{comp} = T_1 \times (P-2) \tag{3.11}$$

$$2 \times (P-2) \times T_c \leq T_{comp} \leq 2 \times f \times (P-2) \times T_c \tag{3.12}$$

The time needed to complete this phase is

$$T_{evolve} = \frac{1}{\Phi} T_{comp} \tag{3.13}$$

$$\frac{2 \times (P-2) \times T_c}{\Phi} \leq T_{evolve} \leq \frac{2 \times f \times (P-2) \times T_c}{\Phi} \tag{3.14}$$

3.3.1.4 Total Time

Adding up the computation time derived from 3.8, 3.10, and 3.14, we have

$$T_{Total} = T_{eval} + T_{update} + T_{evolve} \tag{3.15}$$

$$T_p + 2 \times (P + \frac{(P-2)}{\Phi}) \times T_c \leq T_{Total} \leq P \times T_p + 2 \times f \times (P + \frac{(P-2)}{\Phi}) \times T_c \tag{3.16}$$

It can be easily seen that Equation 3.16 is dominated by the term $2 \times f \times P \times T_c$. It is in fact the matching time of the system. In general, the matching time requires a large portion of processing time in UCS, which is affected by two main factors: the population size and the number of feature. The next two sub-sections will investigate the effect of these factors in UCS through several experiments.

3.3.2 The Effect of the Population Size

Experiments in this section study the effect of the population size on the performance of UCS in terms of the predictive accuracy and the processing time on the 6-bit multiplexer problem. In order to understand the effect of the macro population size, the author decided to merge the macro population and the micro population. In other words, the numerosity of each classifier is always 1 and duplicate classifiers are allowed in the population. In this version, UCS evolves to the maximum population size from an empty population before deletion takes place. The maximum population size defined by the user will become the population size of the system.

Figure 3.2 shows the learning curves over time of UCS with different population sizes. The result reveals that the maximum population size is indeed very important for the learning performance. Extremely small population sizes would make the system suffer from insertion-deletion cycles. The system will not be able to learn in this case as the deletion is too active and destroys the accumulated knowledge. In this experiment, the population sizes of 100 and 300 can be considered as too

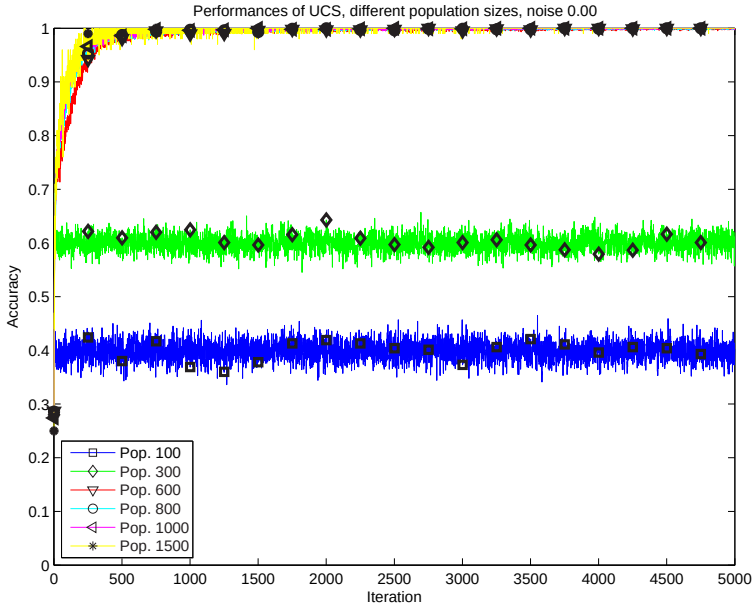

Figure 3.2: The learning curves of UCS with different population sizes

small as the learning curves do not present a sign of improvement. The population size of 300 is able to give better accuracy than the population size of 100 because it has lower deletion pressure and also allows more classifiers in the population. The population sizes of 600 or over enable the system to reach 100% accuracy after 1000 iterations. Increasing the population size from 600 to 1500 does not make a difference to the accuracy in the end but in fact affects the learning rate. The population size of 1500 achieves 100% accuracy after approximately 400 iterations, while it requires almost 500 iterations for the population size of 600 to get similar accuracy.

Figure 3.3 shows the processing time (T_{Total}) and the matching time (T_{eval}) needed for UCS to process 5000 iterations with different population sizes. Experiments in this section were run on the supercomputer: 304 processors Dell Linux Pentium 4 Beowulf cluster and each node consists of two 3GHz CPUs with 2GB memory. The time reported in this section is the smallest time recorded within 30 runs. The minimal time is used instead of the averaged time because the waiting time is also included. The smallest time indicates the smallest waiting time, therefore it is closer to the real time.

As we can see easily, the higher the population size is the longer time needed

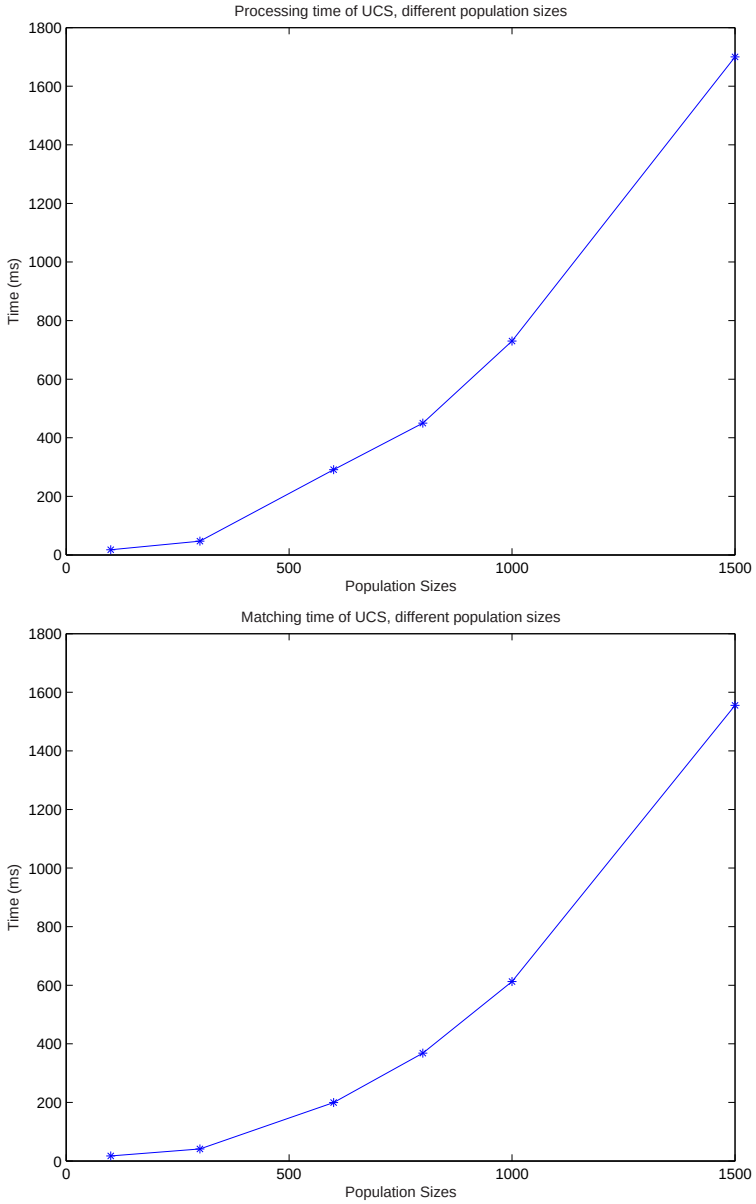

Figure 3.3: The processing time (above) and the matching time (below) of UCS with different population sizes

for UCS to complete the job or to do the matching. The curves again confirmed that the matching time takes most of the processing time of LCS. Increasing the population size from 600 to 1500 requires more than double comparisons for each data instances and therefore double the time would be expected to run. In fact, 200(ms) is required for the population size of 600 in comparison to 1700(ms) for the population size of 1500. The equation 3.16 has showed that the learning time is also related to the frequency of GA activation. That is the overhead time needed

to put on top of the comparison times.

3.3.3 The Effect of the Number of Features

This section studies the effect of the number of features on matching time of UCS, since the matching time is the dominant time during the training and testing. UCS used in this section is similar to the one in previous experiments. UCS will be tested for 2000 iterations on four multiplexer data sets, each of which has different number of features: the 6 real multiplexer, the 11 real multiplexer, the 20 real multiplexer, and the 37 real multiplexer.

Figure 3.4 presents the matching time of UCS for problems with different number of features using two maximum population sizes: 2000 (above) and 6400 (below).

Increasing the number of features requires longer time for matching as the condition becomes bigger. As a result, the matching time required increases.

3.3.4 Summary

In conclusion, this section shows that the population size is very important in UCS in terms of the processing time. In general, increasing the population size would result in better accuracy and faster learning rate. It also increases the processing time, however. There is always the trade-off between the learning rate and the learning time.

The study confirms the trend in UCS: a large population is required to maintain accuracy. However, it highlights a problem in stream data mining related to the slow processing due to a large population, which required longer time in the matching process. This critical problem motivates the author to propose the neural representation discussed in the next chapter.

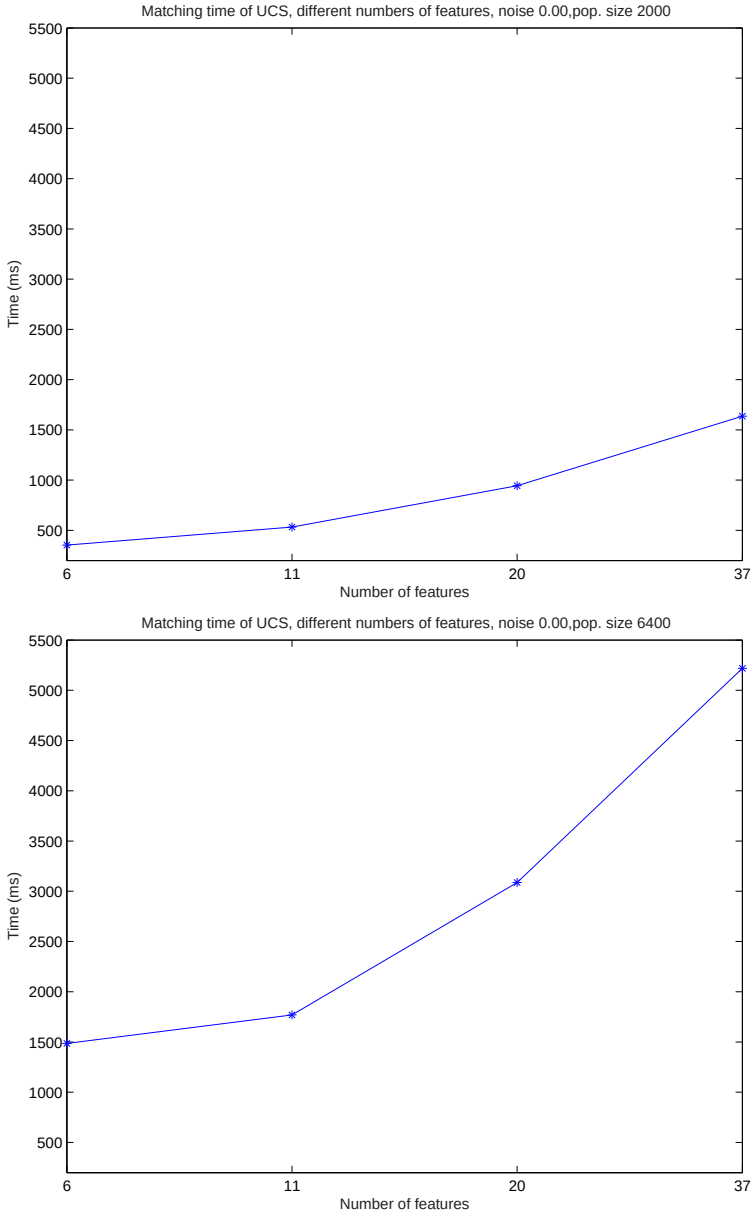

Figure 3.4: The matching time of UCS with different numbers of features using the population size of 2000 (above) and 6400 (below)

3.4 A Study of UCS on Dynamic and Noisy Environments

This section will investigate UCS in a dynamic environment, which is very common in stream data mining. The dynamic environment reflects the popular phenomenon in the real world that things keep changing over time. A good learning algorithm needs to revise its knowledge over time to capture new situations.

Table 3.1: The accuracy of UCS after the change and the recovering time upto 98% with different MoC.

MoC	Accuracy after the change	Recovering time
0.10	0.660 ± 0.074	≈ 5000
0.20	0.600 ± 0.071	≈ 5000
0.30	0.548 ± 0.064	≈ 5000
0.40	0.531 ± 0.063	≈ 5000
0.50	0.530 ± 0.057	≈ 5000
0.60	0.524 ± 0.062	≈ 5000
0.70	0.517 ± 0.054	≈ 5000
0.80	0.509 ± 0.057	≈ 5000

3.4.1 The Influence of MoC

Figure 3.5 shows the predictive accuracy curve and the population size curve of UCS over 8000 iterations, with the change occurring at the 4000^{th} iteration. The magnitudes of change are $MoC = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ respectively. Observing the graphs, a sharp decrease in the predictive accuracy is obvious at the 4000^{th} time step, where the concept change takes place.

Table 3.1 presents UCS's accuracy right after a concept is changed, and the number of time steps required for UCS to recover to 98% accuracy, at each level of MoC.

A similar study on XCS has been investigated in (Dam *et al.*, 2007). The study reveals that XCS can recover quickly from a small MoC. If MoC is large (more than 0.2), XCS requires a much longer time to learn and adapt to the new concepts. If MoC is extremely large (more than 0.4), it is quicker to start learning from scratch rather than continuing with the current knowledge since XCS is not able to recover.

Surprisingly UCS behaves quite differently. Increasing MoC does not seem to require much longer time to recover even though it has an influence on the accuracy. The accuracy after the change drops down to about 65% with $MoC = 0.1$, approximately 57% with $MoC = 0.2$, and almost 50% with other MoC. It is not unusual because UCS aims at developing a partial action map with only accurate classifiers, whereas XCS maintains a complete action map.

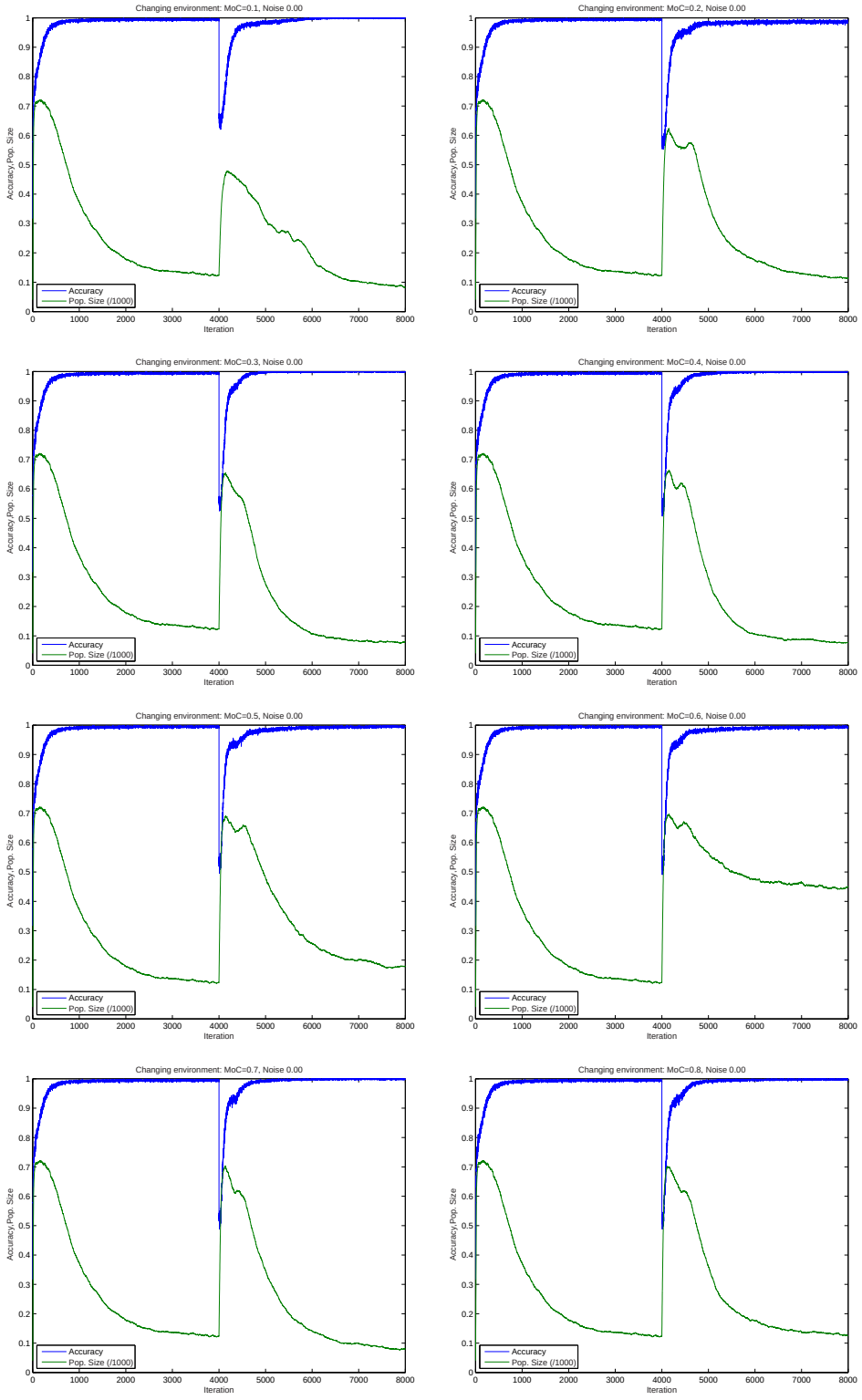

Figure 3.5: The learning curves and the population sizes of UCS in dynamic environment with different MoC values in noise-free environments; MoC = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8.

After small changes, some classifiers in the population remain accurate, resulting in better predictions than random guesses (normally would be 50% for a two classes problem). Moreover, when the threshold is changed from 0.1 to 0.2, any real numbers in the range $(0.1, 0.2)$ in the previous time step that were considered as a binary 1, but now become a binary 0. Since data is generated with a uniform distribution, about 10% of the real numbers are converted to a different binary number in the new environment. There are $50 \times 6 = 300$ real numbers required for 50 training instances (a training window). 10% of them which is about 30 real numbers, would be converted differently. Therefore, at least 5 instances (6 real numbers form an instance) and at most 30 instances contain the real numbers that are modified. At least 20 instances (40%) in the training window are unchanged in the new condition. This explains why the learning curve does not drop as low as the starting point at time step 0 and also the population size does not rise as high as 0.7 as the first time step.

If MoC is higher (more than 0.1), UCS seems to offer random guesses as most classifiers become inaccurate after the change and also most of the data instances would get different classes after the change. As a result, the predictive accuracy right after the change drops lower than the first cycle to around 50%.

Three different behavioural patterns can be drawn with further observing the population sizes when MoC is more than 0.1

The first pattern happens when $MoC = 0.2, 0.3, 0.4$. The population size rises quickly after the change, but not as high as the peak in the first cycle. At the end of the second cycle, the population size stabilizes at around 0.1 as the end of the first cycle. A lower peak of the population size in the second cycle indicates that some classifiers in the first cycle are still useful after the change, therefore the system does not have to learn from scratch.

The second pattern happens when $MoC = 0.5, 0.6$. The population size rises quickly after the change, but not as high as the peak in the first cycle. At the end of the second cycle, the stable point of the population size is much higher than the one in the first cycle. Similar to the previous region, the peak of the

population size in the second cycle is lower than the one of the first cycle, indicating that the system keeps some classifiers in the previous cycle as they are acceptable. The higher population size in the end indicates that UCS fails to generalize in this situation. One reason to explain this behaviour is that some classifiers that survived from the first cycle cannot be subsumed by others in the population as they cover different areas (a mixture of the area of previous concepts and new concepts after the change). They in fact cannot subsume other classifiers because they are not general and accurate in the new situation.

The last pattern happens when $MoC = 0.7, 0.8$. The population size also rises quickly after the change with a similar peak as in the first cycle. The population size manages to get stable at around 0.1 similarly to the end of the first cycle. The results suggest that most classifiers become incorrect after the change. As a result, UCS managed to delete all and starts learning from scratch.

In essence, the behaviour of UCS with different MoCs might be different due to several suggestions above. The learning accuracy in all cases does not change much as 100% accuracy is obtained in most cases. This indicates that UCS is able to adapt in dynamic environments under different levels of change.

3.4.2 The Effect of Noise

Figures 3.6, 3.7, 3.8 show the learning curves and the population size curves with different levels of noise after different MoCs of 0.1, 0.5, and 0.8 respectively.

Observing the graphs, we notice three regions of noise that affects the recovery time of UCS from concept changes. These are: low noise, which results in a slow recovery; medium noise, which helps the system to recover almost as fast as the original learning; and high noise, which increases the recovery time dramatically. The levels of noise in each region are $[0.00, 0.03)$,$[0.03, 0.15)$, and $[0.15, 1.00)$ for low, medium, and high respectively.

In the first cycle, UCS starts without prior knowledge. The population curve increases dramatically after a few steps at the beginning for all three regions. At

this stage, GA is working hard to introduce new classifiers into the system. Those classifiers might be either good or bad. The learning component is responsible for updating their parameters accurately to reflect their suitability.

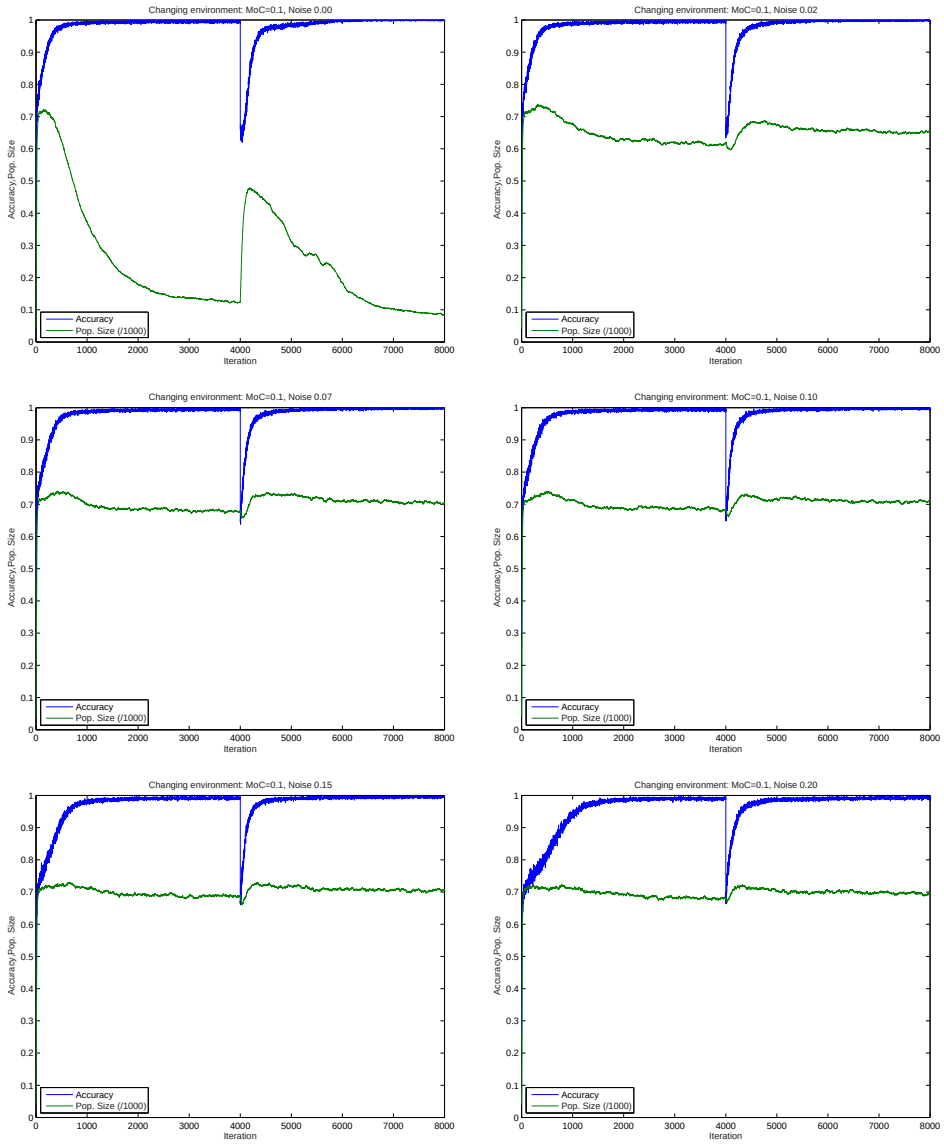

Figure 3.6: The learning curves and the population sizes of UCS in smooth dynamic environments (MoC=0.1) and noisy environments (Noise = 0.00, 0.02, 0.07, 0.10, 0.15, 0.20)

When noise is small, UCS is able to differentiate effectively good classifiers from the bad ones. As a result, the population curve starts to decrease gradually after the peak. By the time the concept change happens, the system has achieved a compact population, which contains only accurate and maximally general classifiers

(Kovacs, 1997).

When the concept change is small ($MoC = 0.1$), some classifiers in the population becomes incorrect. However, the number of these classifiers may be small, resulting in a lower peak of the population size after the change than the one in the first cycle.

When the concept change is medium ($MoC = 0.5$) or severe ($MoC = 0.8$), most of these classifiers become inaccurate after the change. GA starts working hard as a large number of classifiers are inserted in the population. In order to recover completely, the incorrect classifiers need to be re-evaluated and removed from the population. In order to remove those classifiers, it requires a sufficient time to update their parameters from good to bad so that they can be eliminated. This explains why the accuracy in the second cycle improves more slowly than in the first cycle.

Medium noise, on the other hand, affects the learning process as Lanzi and Colombetti (1999) has suggested. UCS is not able to compact the population as much as it could in low noise cases. The population curve decreases insignificantly in the first cycle. Before a concept change, the population contains both good and bad classifiers. As a result, the performance can never achieve 100% accuracy. Observing the decrease in the population suggests that UCS has achieved a certain level of generalization. After the concept change, only a small number of classifiers are inserted in the population. Some inaccurate classifiers in the previous cycle might become accurate under new conditions. UCS does not have to remove and discover these classifiers. Hence the recovering time is reduced significantly.

When the noise level is high, UCS is unable to eliminate bad classifiers completely. The population always evolves both bad and good classifiers. The last row of Figure 3.8 shows that the population curves don't decrease at all after the initial peak. A large population implies that the system was not able to generalize. It seems that the population contains many bad and specific classifiers. Unlike the previous cases, some inaccurate classifiers appear to be still inaccurate after the concept change because they are over-specific. Hence, the recovery time starts

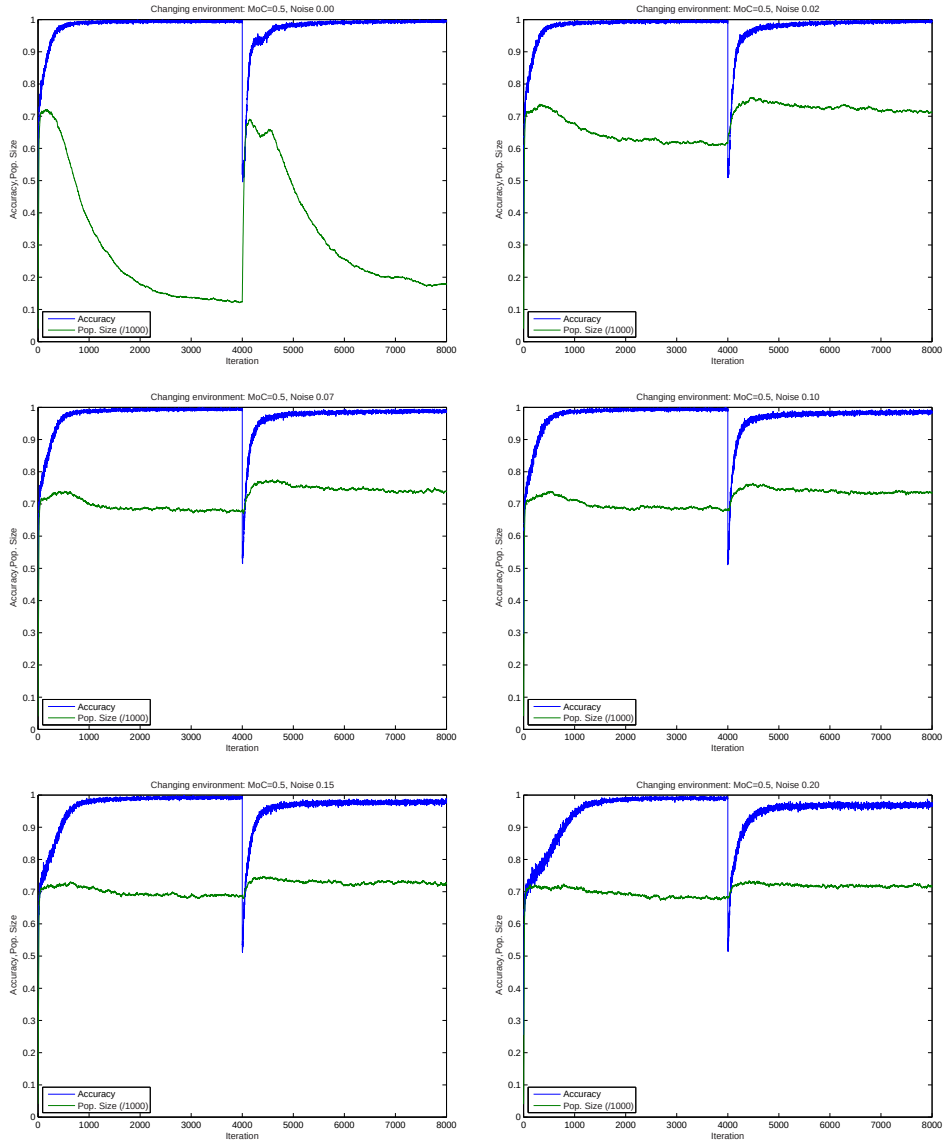

Figure 3.7: The learning curves and the population sizes of UCS in medium dynamic environments (MoC=0.5) and noisy environments (Noise = 0.00, 0.02, 0.07, 0.10, 0.15, 0.20)

increasing as the noise level increases in this region.

3.4.3 Summary

The study in this section indicates that UCS is able to work well in dynamic environments under different MoCs. Noise indeed affects the overall learning in UCS as higher population size is required in noisy environments. However, in general, the predictive accuracy is not affected by the noise. In general, UCS is robust and

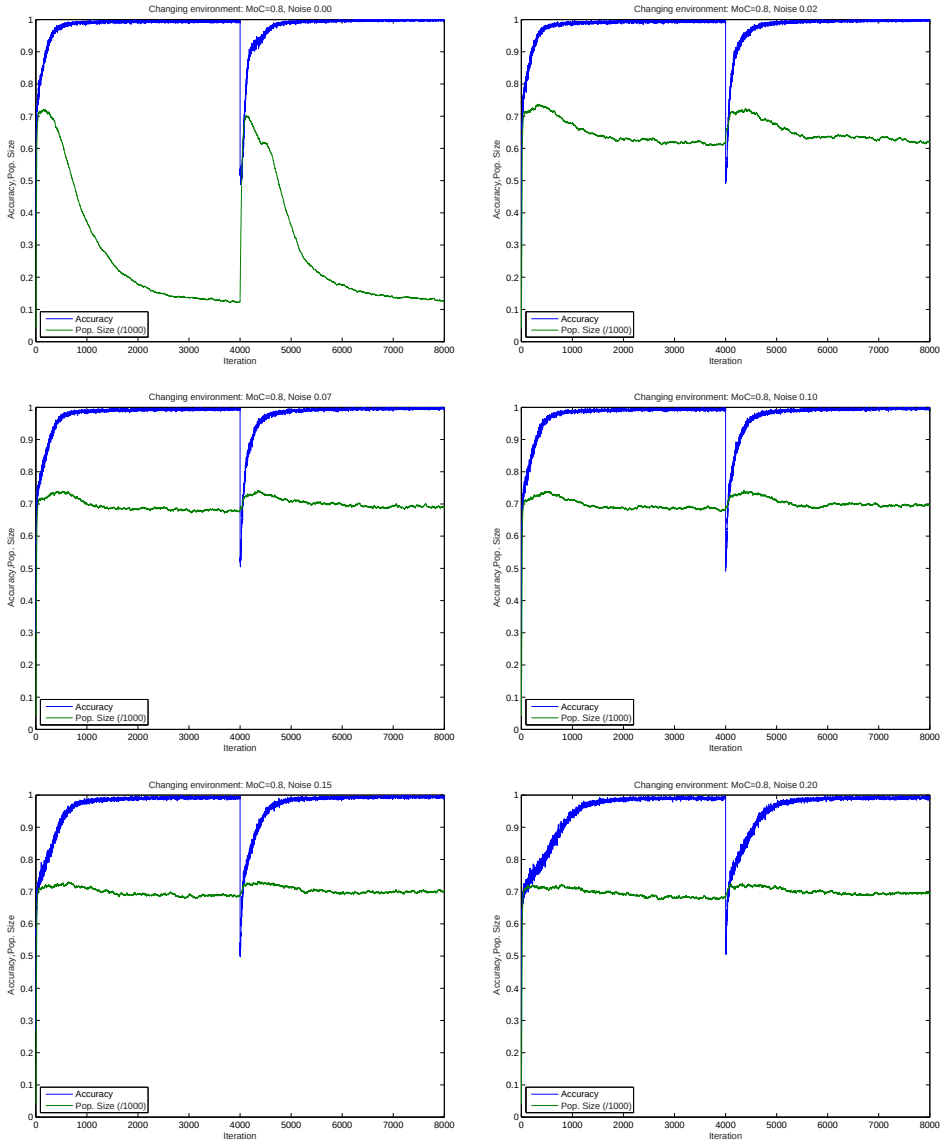

Figure 3.8: The learning curves and the population sizes of UCS in severe dynamic environments (MoC=0.8) and noisy environments (Noise = 0.00, 0.02, 0.07, 0.10, 0.15, 0.20)

suitable for dynamic environments.

3.5 Distributed Environments

This section describes a distributed framework of UCSs, called DUCS, for a distributed environment. The system is built up from several distributed sites and a central site. These sites are described in terms of the client and server architecture.

Figure 3.9 depicts the structural architecture of DUCS.

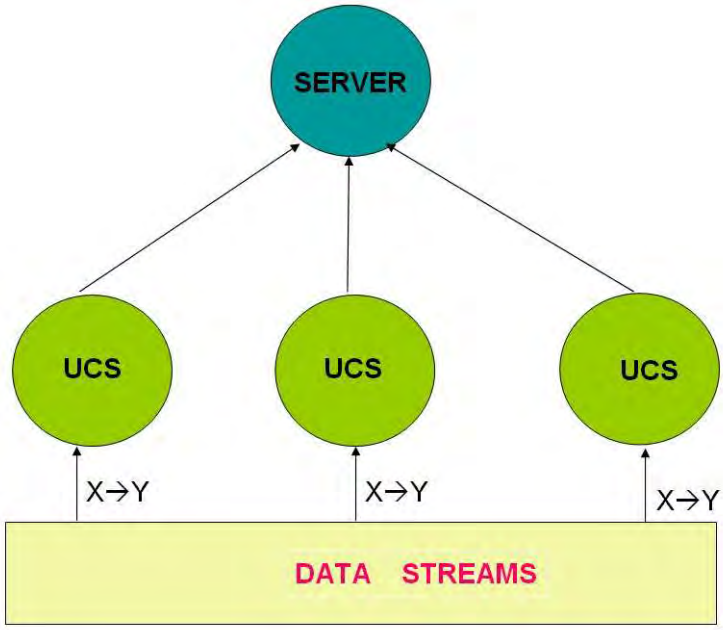

Figure 3.9: The framework of DUCS with three clients and one server

A client is placed at each distributed site and is responsible for extracting the knowledge (learning model) from the local database. The local learning model needs to be transferred occasionally to the server. The server combines the information from clients to form a descriptive model for the global environment.

3.5.1 Distributed Sites

Each client employs a complete UCS, which is trained independently on the local data stream. Local UCSs normally start with an empty population, assuming no prior knowledge is available in advance. They are responsible for evolving the population during the training process, aiming at capturing the local knowledge.

The data transmitted to the server consists of the local model along with a set of data instances. The local model is a population of classifiers learned by the local UCS. The set of data instances is mainly used for training at the server if required.

3.5.2 Server Site

The server stores local models in the memory and avoids any modification to their contents. Local models at the server are synchronized with the client's knowledge through the exchanging data coming from local sites. It allows the server to update its knowledge with the most up-to-date information. Any change at the clients will be realized by the server through the exchanged model. The frequency of transmission normally affects the knowledge of the server. In many cases, if the local models are updated more frequently, the server is more up-to-date.

After receiving the updated models and training data from all clients, the server will combine the local models into a single coherent knowledge base. The clients' models are aggregated at the server using a knowledge fusing technique.

Each input instance at the server is processed by all local models, which might result in different predictions due to different bias learned by clients. The final output, chosen between those predictions, is decided by a fusing method employed at the server.

3.6 An Investigation of DUCS

This section is designed to investigate DUCS in several issues: (1) how to combine the local knowledge at the server; (2) how does DUCS handle data transmission; (3) what is the effect of having different number of clients?

3.6.1 Knowledge Combination at the Server

This thesis chose two approaches of knowledge combination for investigation: knowledge probing and majority voting.

3.6.1.1 Knowledge Probing

A simple UCS is employed at the server to learn the outputs from the local models. This technique requires a training process at the server. Clients are required to send some data along with the local model for training the server. The first step is to generate training instances for the global UCS based on local UCSs. Figure 3.10 illustrates the process to create a new training instance at the server. In this example, the server maintains four local models. 11110001 is a server training instance to input into each local model. 1 is a target class associated with that training sample. Outputs from local models are 1, 0, 0, 1 respectively. Thus 1001 : 1 is an ensemble training instance.

After receiving the updated models from all clients, the server applies the *knowledge probing* approach (Guo *et al.*, 1997) to combine the local models. The key idea of this approach is to derive a descriptive model using the output of the local models. All training data received by the server are used as inputs for all copies of local models available at the server, then the server trains a UCS to learn the mapping between the output of these local models and the target class. In other words, the training instances for the server's UCS are created online at the server by all local models. Each server's training instance is inserted to each local UCS and the class exported out of the model becomes an attribute of the ensemble training instance.

The training phase at the server using this knowledge probing technique is described as follows:

- Form New Inputs:
  - A set T of n training instances obtained from clients
    $T = \{(t_1, c_1); (t_2, c_2); ..; (t_n, c_n)\}$ where t is a training instance and c is an associated class.
  - A set L of m local UCS models $L = \{l_1, l_2, .., l_m\}$
  - A learning algorithm UCS on top of local models, which provides a final descriptive output of the server. That model represents the global view

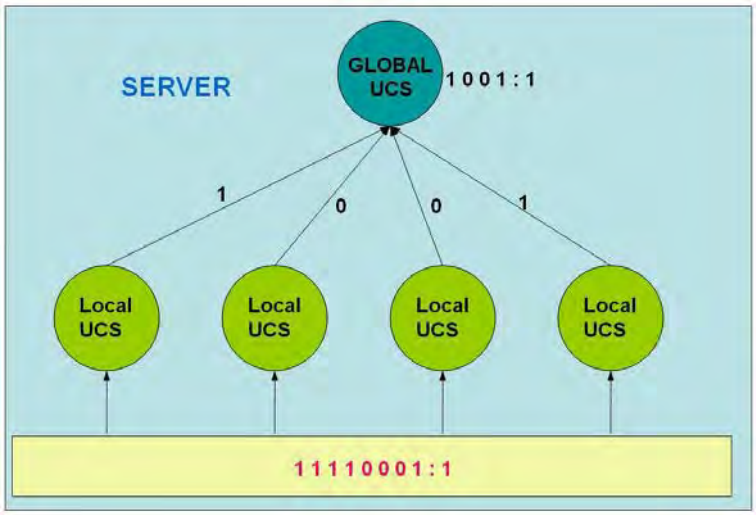

Figure 3.10: An example of the knowledge probing approach at the server in DUCS

of the server.

- Prediction Phase: Obtain outputs from each model in L for each data item in T and form an ensemble of training instances. The outputs of the n^{th} instance t_n from the set of models in L are a set O_n: $O_n = \{o_{n1}, o_{n2}, .., o_{nm}\}$. The set O consists of the ensemble of training instances for UCS $O = \{O_1, O_2, ..O_n\}$. The new server training set becomes $S = \{(O_1, c_1); (O_2, c_2); ...; (O_n, c_n)\}$
- Learning Phase: Learning from data entries in the server training set S, $L* = UCS\{(O_1, c_1); (O_2, c_2); ...; (O_n, c_n)\}$
- Output: descriptive model L^* obtained from the learning phase.

Once the descriptive model is defined, it will be used to combine the outputs from clients and propose the final output at the server.

3.6.1.2 Majority Voting

An outcome of the server is contributed by all local models in DUCS. The server will choose the final outcome by voting between local models.

Assume DUCS has a set L of m local UCS models $L = \{l_1, l_2, .., l_m\}$. For each testing instance $T_i \in T(T = \{(t_1, c_1); (t_2, c_2); ..; (t_n, c_n)\}$, where t is a training instance and c is an associated class) we have:

- Obtain predictive outputs from local models in L for each data item T_i.

- Voting is carried out between local predictive outputs in order to choose the most preferred outcome.

- The winning outcome is the one whose number of votes is greater than half of the numbers of individuals.

3.6.1.3 An Investigation of the Knowledge Probing Approach

Before comparing voting against knowledge probing, the author first investigates one of the most sensitive parameters of the knowledge probing approach: the training set of UCS at the server. The training data at the server is combined from all local data received from clients. More training instances at the server would result in faster learning at the server.

Figure 3.11 presents the learning curves at the server with different training sizes (5, 10, 50, 100 instances) in both noise-free and noisy environments. The training size refers to the amount of training instances that each client sends to the server along with its model. The data is chosen randomly from incoming instances at clients.

Clearly, learning is slowest with the training size of 5 instances, fastest with the training size of 50 instances. The difference between the size of 50 instances and 10 instances is mainly observed during the first 400 iterations. Because UCS starts without prior knowledge, it requires a period of time to build up the knowledge base. Sending more training data to the server would provide more information and therefore speed up the initial learning of UCS. A small training size would not provide enough data for training the server, resulting in unstable performance.

The difference between learning with 100 instances and 50 instances is fairly small. However, the training size of 100 instances requires the transmission of double the amount of training data. The author decided to choose the training size of 50 instances for investigation from this point in the chapter.

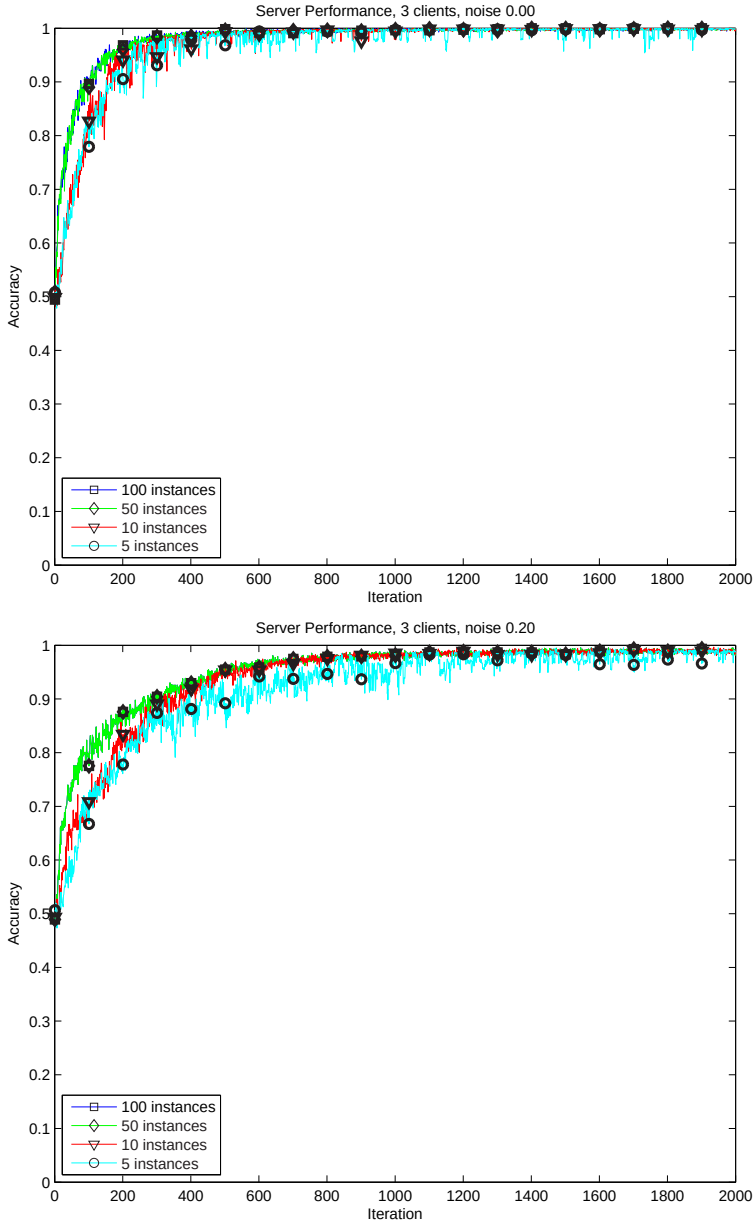

Figure 3.11: The learning curves at the server using the knowledge probing approach with different training sizes – In the noise-free environment (upper) and the noisy environment (lower).

3.6.1.4 Comparison of Knowledge Probing and Voting

Figure 3.12 shows the learning curves at the server of DUCS using the voting and knowledge probing approaches in noise-free environments. The learning at the server with the knowledge probing approach is slower at the beginning (the first 200 iterations) in comparison to the majority voting method. This is because the

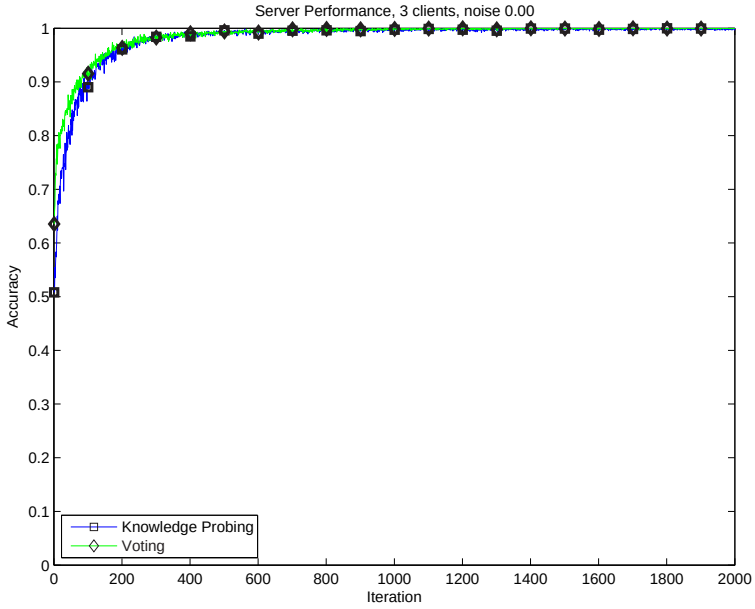

Figure 3.12: The learning curves at the server of knowledge probing and voting in noise-free environments

knowledge probing approach has another UCS on top of the local models. The server first forms its new training instances from the predictions of local models and then uses them for training the extra UCS. In order for UCS to obtain enough knowledge, initial training time is required. The majority vote, on the other hand, does not require learning time because the decision is based directly on voting between clients' predictions. However, once UCS accumulates enough knowledge in the knowledge probing approach, it starts to speed up its learning and converges faster than the voting approach. Both approaches are able to achieve 100% accuracy after about 200 iterations.

Figure 3.13 shows the learning curves at the server, using the knowledge probing and majority voting methods, with the noise level of 0.2. A similar pattern applies for noise level of 0.1.

It is easily observed that majority voting is more robust to noise than knowledge probing, as it learns faster and achieves higher accuracy than the knowledge probing approach in noisy environments.

In the knowledge probing approach, the training data of UCS at the server is constructed from predictions of local models. If predictions of local models are in-

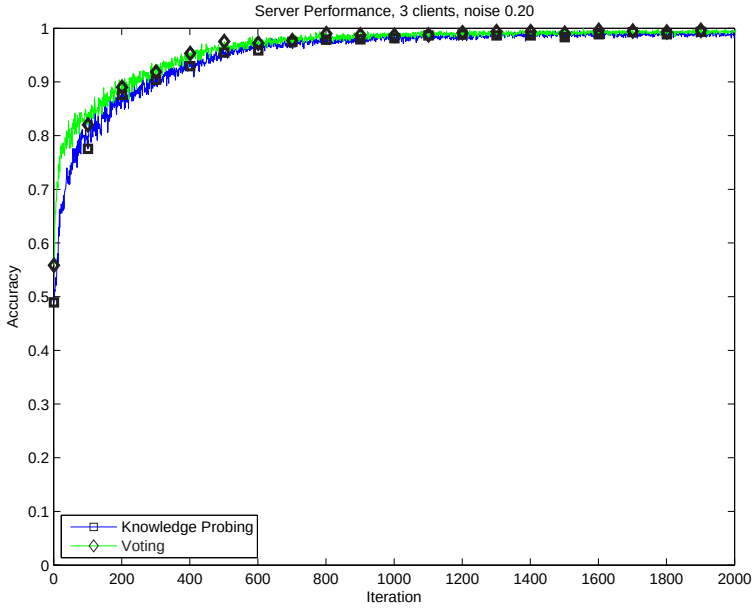

Figure 3.13: The learning curves at the server of knowledge probing and voting in noisy environments (noise level of 0.2)

correct, the training data becomes noisy for training. At an early stage of learning, some predictions of local models are incorrect due to their inexperience, since UCS normally requires a substantial time for the evolutionary and learning components to explore the search space and evolve the population. Once UCS receives enough training data, its knowledge (population of classifiers) becomes more reliable. There is also a high chance of misclassification by local models as the learning algorithm normally requires more time to eliminate the effects of noise. The knowledge accumulated in the system is unreliable due to noisy data encountered in the past and more training is required to reduce their impact.

When the training size is too small (50 instances), knowledge probing is not able to converge as seen with a larger training size (500 instances). Clearly increasing the training size will help the system to overcome noise more quickly.

The voting method is more advantageous than the knowledge probing approach in noisy environments because it can reduce the bias in the system by their votes.

In conclusion, both the knowledge probing approach and the voting approach can attain 100% accuracy. The knowledge probing approach learns slower than the voting approach in noise-free environments. When noise occurs in the data,

majority voting is more robust. Thus from this point, the author will only consider the majority voting method for combining predictions at the server.

3.6.2 Effects of the Number of Clients

In this sub-section, DUCS is simulated with three, five, seven, and nine clients in noise-free environments. Each client is trained by a similar amount of data. In the other words, the training data of a system with nine clients is three times larger than the one with three clients.

The purpose of this experiment is to reassure us with DUCS performance in a real environment, since it means that people could use a number of clients that suits their business arrangements, or as needed to cope with the rate of arriving instances, rather than a particular number of clients that makes a difference to accuracy.

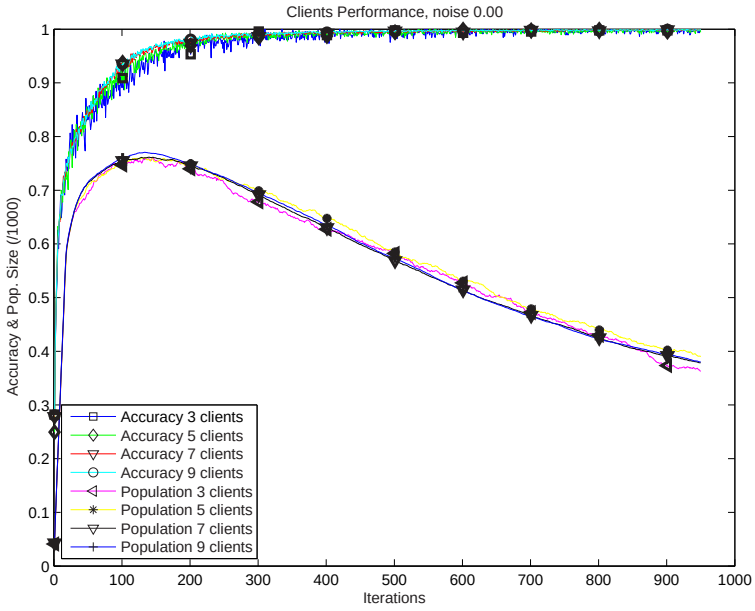

Figure 3.14: The learning curves and population sizes at clients with different numbers of clients in noise-free environments

Figure 3.14 shows the averaged learning curves and the averaged population size curves at the client with different numbers of clients in noise-free environments.

Increasing the number of clients slightly improves the learning rate in the be-

ginning, but the population size does not seem to be affected. In this experiment, each client might learn a different bias as it is exposed to different portions of the data. As a result, the averaged accuracy is slightly better for the system with more clients. The population size curves, on the other hand, are not different because they are averaged over the number of clients; and also each client would go through a similar generalization process after training with a similar amount of training data. This indicates that increasing the number of clients will result in bigger data communication in DUCS because all clients need to update their models at the server and each client will end up with a model with similar size.

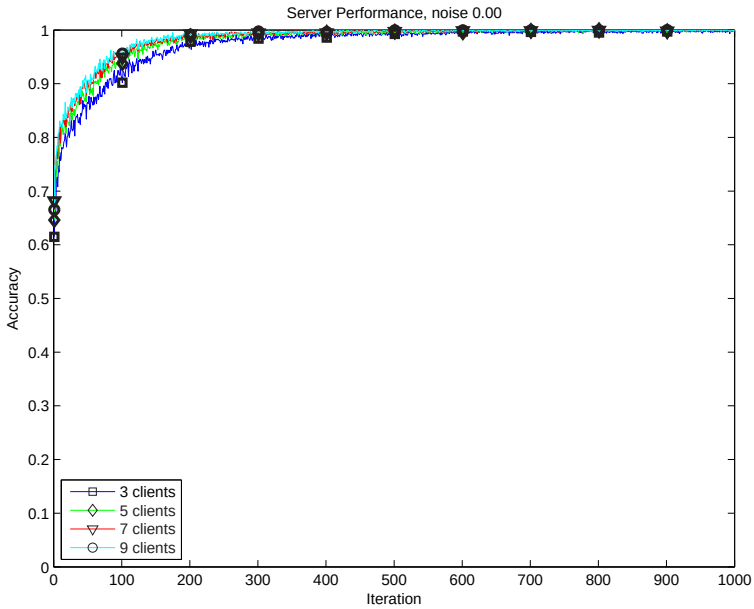

Figure 3.15: The learning curves at the server with different numbers of clients in noise-free environments

Figure 3.15 shows the learning curves at the server with different numbers of clients in noise-free environments. The number of clients affects the predictive performance of the server only at the beginning. The server with nine clients learns faster than with three and five clients. Increasing the number of clients from five to nine results in a faster learning rate. At the beginning, the clients might not have yet discovered all hidden information due to bias in data distributions. Therefore having more clients gives more information to the server, and results in faster learning by the server. Once the clients have learned completely, the performance at the server is not affected by the number of clients. More clients also means a larger data

transmission, without any benefit in improved accuracy once the server has learned enough.

In conclusion, the number of clients seems to affect the learning rate of the server. More clients tends to result in faster learning, but only up to a fairly small number of clients. Moreover, more clients mean larger data transmission is required in the system.

3.7 Knowledge Passing in DUCS

This section is designed to understand the effect of knowledge passing in DUCS. The system builds up its knowledge over time by learning at clients. The knowledge of clients is represented by populations of classifiers. Knowledge passing refers to the migration of classifiers within the system. Two main channels of transmission are explored in this section: between the clients and the server, and between clients.

3.7.1 Data Transmission between Client and Server

In (Dam *et al.*, 2005b) the author proposed the data transmission between clients and server for binary domain based on the Minimum Description Length (MDL) principle following the path of Bacardit and Garrell (2007). MDL is normally used to evaluate the complexity and accuracy of the model in terms of data compression. The main difference between this work and the one of Bacardit and Garrell (2007) is in the way that the MDL principle is used. In (Bacardit and Garrell, 2007), the MDL principle is used in the fitness function to combine the accuracy and the complexity of the individual, whereas the MDL principle is used in this work to measure the real traffic between clients and server.

The data needed for transmission between clients and the server includes the model and the training instances. Therefore, the cost of transmission is equivalent to the number of bits needed to encode the model (theory bits) plus training instances

(exception bits).

$$MDL = TheoryBits(TL) + ExceptionBits(EL) \tag{3.17}$$

The length of the theory bits (TL) is the number of bits required to encode a set of classifiers that will be transferred in the network. In the previous papers, the author ignored all parameters associated with each classifier. This section presents a formula that takes into account the length of all parameters associated with all classifiers.

The classifiers have a common structure: $Condition \longrightarrow Action : parameters$. Their lengths are defined as follows:

$$TL = \sum_{i=1}^{nr}(TL_i) + \sum_{i=1}^{nr}(CL_i) + \sum_{i=1}^{nr}(PL_i) \tag{3.18}$$

Where nr is the number of classifiers needed for transmission; TL_i, CL_i, PL_i are the length of a condition, an action and a set of parameters in one classifier respectively. Assume that the interval predicate is used to encode a condition, the action is presented in integer, and that each classifier will transfer three parameters: fitness (a real value), numerosity (an integer), and experience (an integer). The length of each component of a classifier can be estimated as follows:

$$TL_i = 2 \times nc \times nreal \tag{3.19}$$

$$CL_i = nint \tag{3.20}$$

$$PL_i = 2 \times nint + nreal \tag{3.21}$$

where nc is the number of features; $nreal$ and $nint$ are the number of bits required to encode a real value and an integer respectively. Therefore, the theory bits are estimated as:

$$TL = ((2 \times nc + 1) \times nreal + 3 \times nint) \times nr \tag{3.22}$$

Similarly, the exception part (EL) is estimated as follows:

$$EL = nu \times (nc \times nreal + nint) \tag{3.23}$$

where nu is the number of training instances sent to the server.

Thus, the length of data sent from a client to the server is:

$$MDL = (2 \times nr + nu) \times (nc) \times (nreal) + (nr) \times (nreal) + (3 \times nr + nu) \times (nint) \tag{3.24}$$

This equation is used to measure the communication load between a client and a server in DUCS. This data consists of the learning model and the training data at the server. Reducing the traffic requires two conditions: (i) small model and (ii) less data instances. If voting is used at the server, the second condition can be ignored. Therefore, obtaining smaller local model is a focus of this thesis.

3.7.2 Investigations of Data Transmission Between Clients and Server

The knowledge passed between clients and the server includes the clients' learnt models and some training data. The training data will be used for training at the server, if required in order to resolve any conflicts between local models. A collection of local models at the server represents the knowledge of the whole distributed databases. The training data is normally small in comparison to the local model. In particular, the simple voting approach does not require any training data at the server. Hence, knowledge passing between clients and server in this section is mainly concerned with the transmission of local models without training data.

In the previous DXCS framework (Dam *et al.*, 2005b), populations of classifiers at clients are regularly transmitted to the server. The transmission data is reduced over time as the learnt models become more accurate and therefore compact. The advantage of this approach is that it is quite simple and also reserves the local knowledge at the server. However sending the whole population when the system

is in a stable condition uses more network bandwidth than is actually needed.

To reduce the amount of data transmitted between clients and the server, in DUCS the author introduces an approach to transfer partial knowledge. In this approach, each classifier is associated with a distinct identification number (ID) that differentiates it from others in the population. The main purpose of this parameter is to synchronize the local models and their copies at the server without sending the whole population. Figure 3.16 illustrates the data transmission from a client to a server using the partial knowledge passing approach.

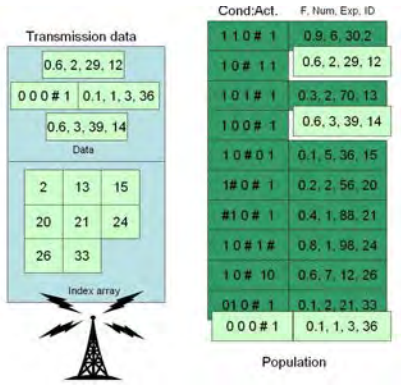

Figure 3.16: The partial knowledge passing approach - An example of the data contained in a transmission packet from a client to the server

Data transmitted to the server consists of three main types of data: newly created classifiers, parameters of those classifiers that have been updated since the last transmission, and the unchanged index. The unchanged index contains the ID of those classifiers that have not been modified from the previous transmission. This index is used to synchronize the local model at the server.

When the data arrives at the server, it will have to wait for the server to update its corresponding local model. The server will first insert newly created classifiers into the population. The parameters of the updated classifiers are revised to record their new values. The server will discard classifiers that were not updated and whose ID was not listed in the unchanged index.

When we tested the approach using partial model transmission, the learning rate and accuracy were exactly the same as with full model transmission. This is as expected, since the difference between the approaches is just how the models are

transmitted; the models still contain the same information.

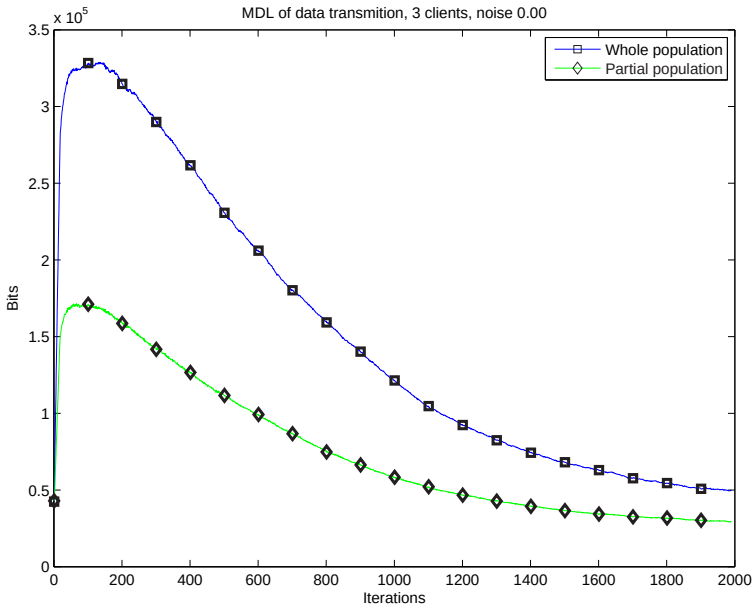

Figure 3.17: The data transmission between clients and a server with whole/partial population

Figure 3.17 shows the amount of data transmission in DUCS from clients to the server, measured by the MDL equation, with full model and partial model transmission. It can be seen that the amount of data transmitted using the partial knowledge passing approach is about half that with full knowledge passing. Thus the partial knowledge passing helps to reduce substantially the network traffic while still maintaining equivalent accuracy and learning speed.

3.7.3 Investigations of Knowledge Transferring Between Clients

In this section, the learning speed at clients is improved by sharing knowledge between clients. In (Bull *et al.*, 2005) and (Bull *et al.*, 2007b), the rule sharing takes place when the average time since the last rule sharing exceeds a threshold defined by users. A single rule is chosen according to fitness using the standard roulette-wheel selection. This rule is sent to another population in the ensemble. The study found that sharing rules chosen from an action set is superior than from the whole population.

In this chapter, a similar idea of rule sharing between clients is applied. The rule sharing is carried out at the client level, where clients exchange their available knowledge to each other. The main difference with the work of Bull and Kovacs (2005) and Bull *et al.* (2007b) is that clients in this chapter are trained with independent data sets and the migration classifiers are chosen in the whole population. The sharing decision is chosen after each fixed time window. Each client maintains its own temporary pools for keeping migration classifiers from other sites. After a time window, in this experiment set to 5 instances, each client chooses a classifier from its population with probability proportional to its fitness. The client sends the chosen classifier to a random client in the system. The classifier is placed in the pool of that client waiting for integration into its population.

Figures 3.18 and 3.19 show the learning curves at the clients and server with and without knowledge sharing. We can see that clients learn a little bit faster with the knowledge sharing. The population with knowledge sharing is able to cut down faster. The migration of classifiers in the system increases the diversity in the population, giving faster convergence.

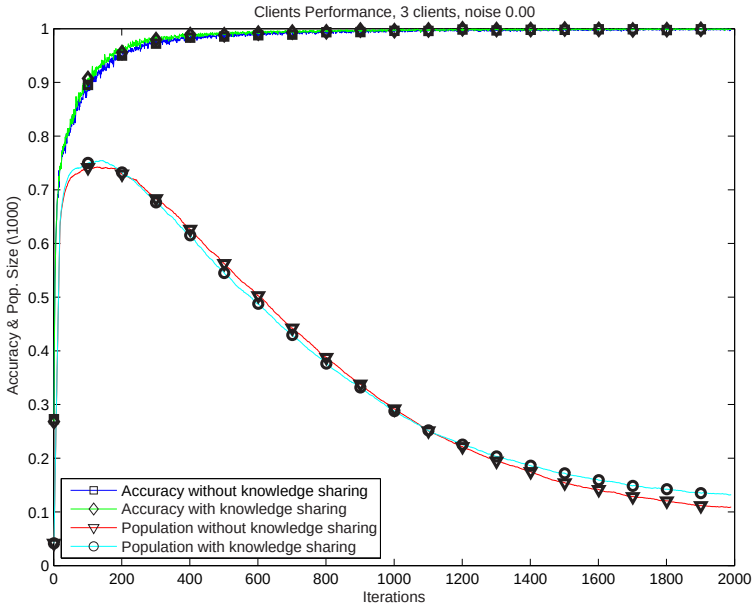

Figure 3.18: The learning curves and population sizes at clients with/without knowledge sharing

However, its population is not able to achieve the same compactness as the

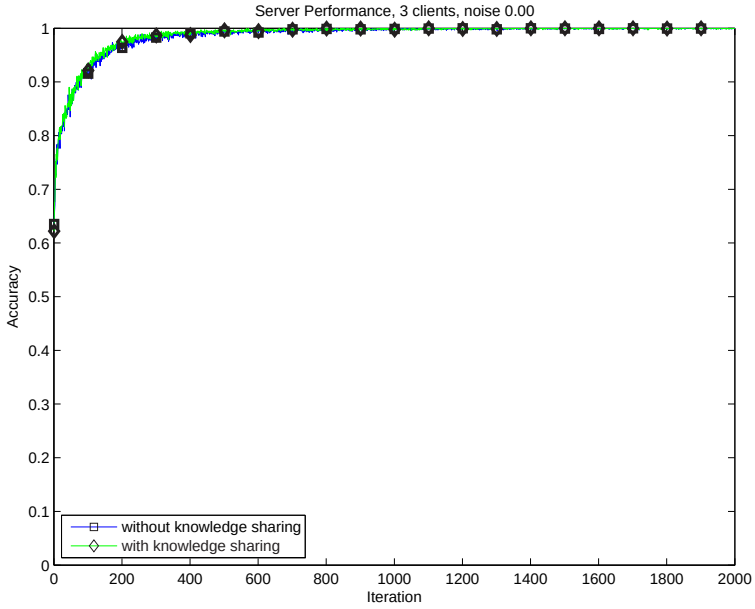

Figure 3.19: The learning curves at the server with/without knowledge sharing

population without knowledge sharing. Continuing the knowledge sharing when the system is stable prevents UCS from compacting its knowledge because diversified classifiers are introduced to the population.

Results so far have been for the 6-real-multiplexer problem. The knowledge sharing also has a significant impact in the binary domain. Figures 3.20 and 3.21 show the learning curves at client and server, respectively, of the binary 20-bits multiplexer problem. Similar pattern can be observed in the binary domain. Clients learn much faster with knowledge sharing. UCS is able to achieve 100% accuracy after 400 iterations in comparison to 900 iterations without knowledge sharing. The population with the knowledge sharing is able to reduce in size faster as it becomes stable after 400 iterations, which is around half time required without the knowledge sharing. The migration of classifiers in the system increases the diversity in the population, which is why faster convergence is observed.

The improvement of learning speed at the clients has a direct impact at the server. The server is able to achieve 100% accuracy after 200 iterations in comparison to more than 300 iterations without knowledge sharing.

Thus knowledge sharing between clients helps to speed up the learning at clients

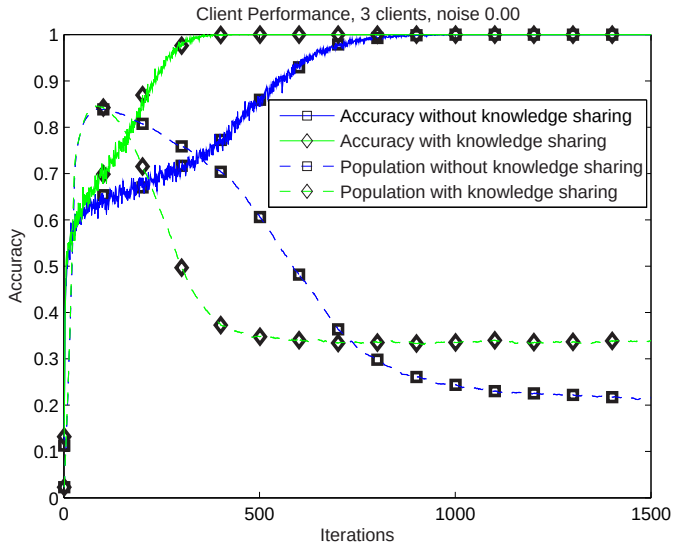

Figure 3.20: The learning curves and population size curves at clients with/without knowledge sharing on the 20-bits multiplexer

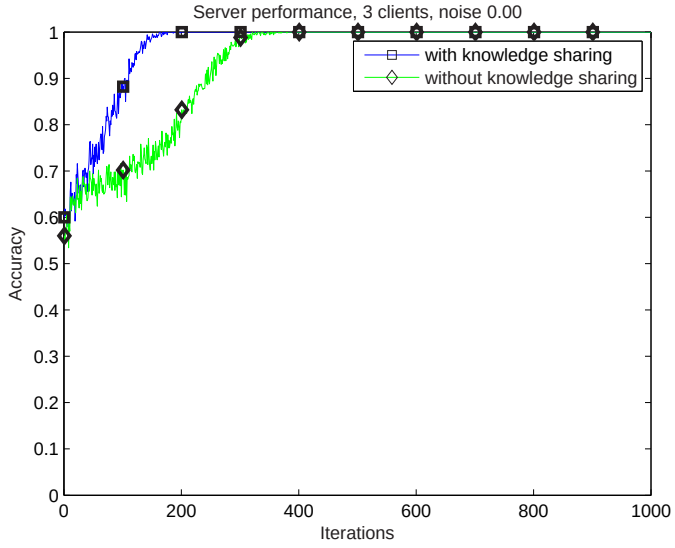

Figure 3.21: The learning curves at the server with/without knowledge sharing on the 20-bits multiplexer problem

and therefore at the server. However, the population size is bigger with knowledge sharing than without. This is the trade-off between faster learning speed and the compactness of the population.

3.7.4 Summary

This section studies DUCS, the proposed distributed framework for UCS. The first experiment investigates DUCS under two approaches for combining the knowledge at the server. The study found that both approaches are comparable in terms of the predictive accuracy. The knowledge probing approach is able to learn faster in noise-free environments, whereas, the voting approach is more robust to noise. Since noise always exists in real life, the author decided to choose the voting approach for further investigation.

The second experiment is carried out to test DUCS with different numbers of clients. This experiment tries to simulate the real life where an organization might have different branches depending on their need. A study found that more clients results in faster learning at the server, but only up to a fairly small number of clients.

The last experiment investigates knowledge passing in DUCS. The MDL function is proposed to measure the traffic between a client and a server. The author believes that the traffic load within the system can be reduced by cutting down local models. A method to transfer partial models in DUCS is then proposed. The result shows that DUCS requires much less data transmission using this method. Finally, the knowledge sharing between clients is investigated indicating that faster learning can be obtained using this method.

3.8 The Comparison between DUCS and Centralized UCS

Two algorithms are investigated in a simulated distributed environment with three clients and one server. The first algorithm is our proposed DUCS and the second one is a centralized UCS. The centralized UCS system is equivalent to having one UCS in the server and the clients keep sending all local data to the server.

In this experiment, the transmission between the client and the server occurs

using a fixed window. We call the training of each local UCS in DUCS using each window an epoch. After each epoch, the clients send their local information to the server. A similar process is followed for the centralized UCS system, where all data (since there is no local learning) get transmitted from the clients to the server after an epoch.

3.8.1 The Predictive Accuracy

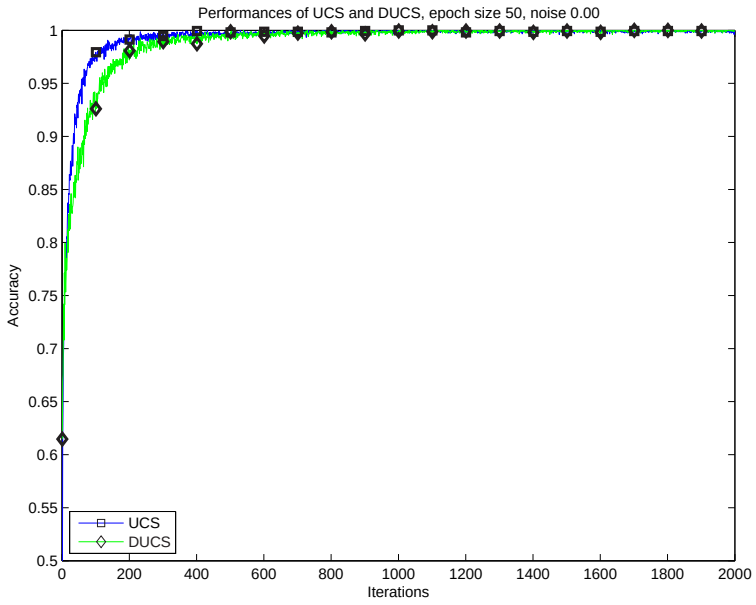

Figure 3.22: The learning curves at the server of DUCS and the centralized UCS with the epoch size of 50 in noise-free environments

Figure 3.22 shows the testing curves at the server of DUCS and the centralized UCS in terms of the accuracy. The testing accuracy of DUCS is almost as good as the testing accuracy of the centralized UCS. In fact, DUCS seems to generalize slower than the centralized UCS at the start of the time interval. This behaviour is expected since DUCS needs more time in the beginning to combine correctly the models from the clients. However, with time the testing performance of DUCS achieves the same level as the centralized UCS.

Figure 3.23 plots the MDL values over time to measure the data transmission between the clients and the server after each epoch. Because the system starts with an empty population, the MDL value is high at the start. The covering

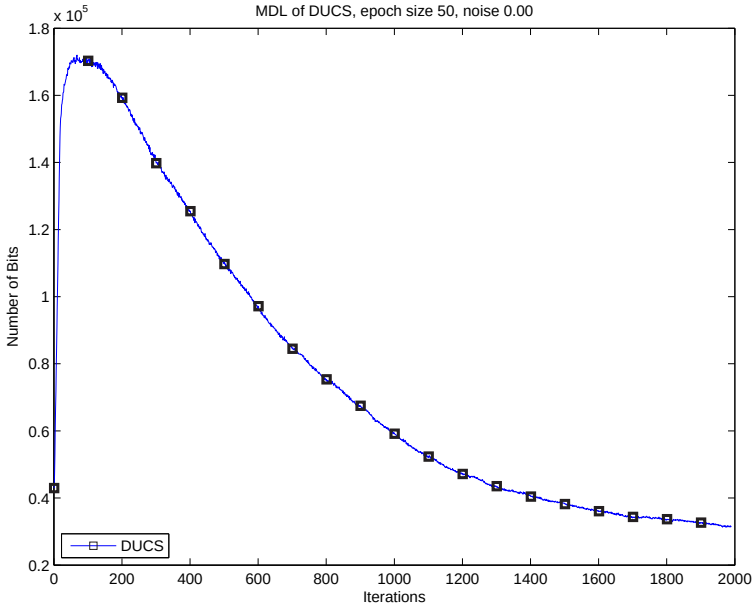

Figure 3.23: The data transmission of DUCS with the epoch size of 50 in noise-free environments

technique is invoked often, creating new classifiers to match each of the previously unseen data. During this stage, the population is full of inaccurate macro-classifiers and many misclassified instances are sent to the server. As training proceeds, the fitness pressure of UCS pushes its population towards accurate and general classifiers. There are fewer misclassified instances, and the model becomes smaller as the system discards inaccurate and specific classifiers. After about 1500 generations, MDL seems more stable.

Overall, the results show that DUCS is very competitive when compared with the centralized UCS. Data transmission in DUCS is high to start with, but stabilizes rapidly.

3.8.2 The Effect of Epoch Size

We have shown that data transmission in DUCS decreases over time. However, if the epoch size is small, we will need to send the model more frequently. In addition, the size of the data used to train the model in each epoch may actually be smaller when compared to the size of the model itself. Therefore, it may sound as though there is not any real saving in terms of network load.

In this sub-section, the author decided to increase the epoch size; thus, more data is used to train each client before the model is sent to the server. In this way, the model size would be smaller than the size of the data used, and we can also estimate the effect of the epoch size on the performance of the server.

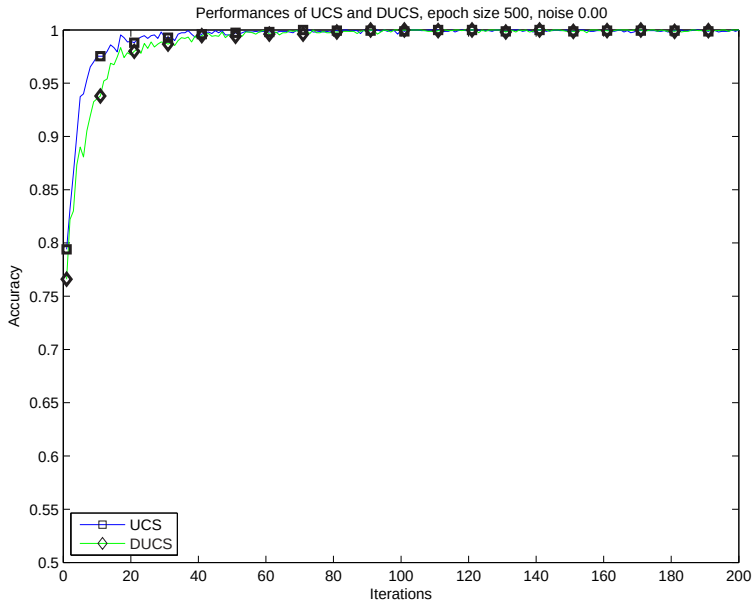

Figure 3.24: The learning curves at the server of DUCS and the centralized UCS with the epoch size of 500 in noise-free environments

Figure 3.24 depicts the testing performance of DUCS and centralized UCS when the epoch size is increased from 50 to 500. The training accuracy of DUCS is almost as good as the centralized UCS. The two curves may look smoother than the corresponding ones with epoch size of 50 because we now have less number of epochs and we average over more data. The real differences are not much when we inspected the data. In terms of generalization, however, it seems that both the decentralized and centralized versions improved their generalization when the epoch size increases. This is logical since each model is exposed to more data before it is sent to the server, allowing it to learn better and improve its generalization.

Figure 3.25 shows the MDL of both DUCS and centralized UCS with the new epoch size. Since centralized UCS sends a constant amount of data to the server (as the author assumes data arrives at the client and is passed onto the server at a constant rate), its curve is a horizontal line. Data transmission in DUCS quickly

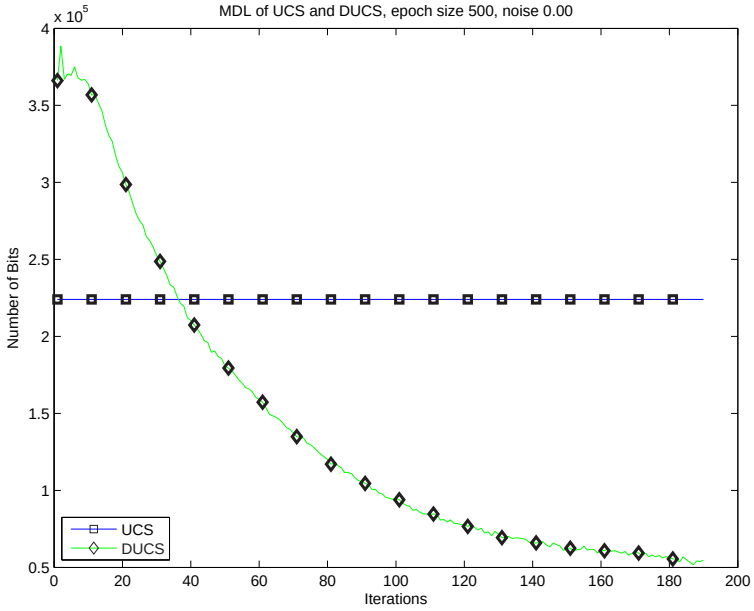

Figure 3.25: The data transmission of DUCS and the centralized UCS with the epoch size of 500 in noise-free environments

drops below that line, and settles at less than half the data transmission for the centralized UCS. The results show that an increase in the epoch size improves generalization and reduces data transmission. However, the author cannot generalize these findings to other problems.

Increasing the epoch size means the server is updated less frequently, delaying its response to changes in the environment. The trade-off between accuracy, data transmission, and up-to-date server will vary depending on the nature of the problem.

3.9 Chapter Summary

This section has investigated UCS with three issues of stream data mining: processing speed, concept changes, and distributed environments.

The first study in this chapter has shown that the population size is a crucial issue in UCS, which affects the learning performance in terms of accuracy and processing time. Theoretical equations and experimental results show that the execution of UCS is dominated by the matching time. In many cases, larger

population size means faster learning, better accuracy, and higher matching time (in other words, higher processing time). However, in stream data mining, smaller processing time is more favourable as less training instances are discarded to keep pace with the high speed data arrived.

The second study tests UCS in dynamic and noisy environments. UCS showed to be robust to noise and adaptive to changes in the underlying data.

The last study is carried out in a distributed environment. The distributed framework of UCS, called DUCS, is proposed and investigated. Two knowledge fusion methods at the server were tested. The voting approach is favoured as it learns faster and is more robust to noise. Other issues of DUCS such as different number of clients, knowledge sharing and different epochs size are also studied. Moreover, the data transmission in DUCS is studied by theory and experiment. In general, this study found that DUCS is able to offer equivalent accuracy, smaller traffic, more accurate models in comparison to the centralized UCS.

In conclusion, the study in this chapter confirms the first research sub-question, stated in Chapter 1, that UCS is suitable for stream data mining as it is robust to noise, adaptive in dynamic environments, and able to handle distributed environments. However, the population size is a significant issue affecting: (i) learning in centralized environments; (ii) learning at clients and server and data transmission in distributed environments; (iii) and more importantly the processing time. The big question related to the population size in UCS is how to reduce it while still maintaining the accuracy and expressiveness of the rule-based population. The next chapter is designed to tackle this problem.

Chapter 4

Neural-based Representation

The following paper is based on this chapter:

1. H.H. Dam, H.A. Abbass, C. Lokan, X. Yao (2008) Neural-Based Learning Classifier Systems, *IEEE Transactions on Data and Knowledge Engineering*, 20(1):26-39.

4.1 Overview

The representation of a rule in UCS as a univariate classification rule is straightforward for a human to understand. However, the system may require a large number of rules to cover the input space. This makes the system less efficient in terms of computational time, memory usage, and comprehensibility. This motivated the author to propose a new representation, aiming for a population with fewer rules while still maintaining the predictive accuracy.

Artificial neural networks, commonly referred to as neural networks (NNs), are inspired by the human brain. Haykin (1999) presented several useful properties and capabilities of neural networks, such as nonlinearity, built-in adaptivity to changes in the surrounding environment, capability of robust computation, and a compact model. These properties motivated the author to investigate the use of neural networks in UCS. Their disadvantage is in the expressiveness: they are normally treated as black boxes, due to their complicated representation.

The use of neural networks in LCS was first proposed by Bull and colleagues

in (Bull, 2002) and (Bull and O'Hara, 2002). In their approach (called NCS) the *condition-action* parts of classifiers were replaced by a fully connected multi-layer perceptron. Experiments showed that NCS is able to learn appropriate structure in simple robotics applications (Hurst and Bull, 2004), and can solve several maze problems (Bull and O'Hara, 2002). However, by replacing a rule completely by a neural network, NCS lacks the main advantage of LCS; that is, being a rule-based system that is potentially easy to understand.

In order to use neural networks without losing too much expressiveness, the author decided to only modify the action part of classifiers. In the proposed representation, the conditions are unchanged but each action is replaced by a simple neural network. The conditions are used to decompose a complex problem into a number of relatively simple tasks. Each task is then learnt by a complete but simple neural network.

A smaller number of more general classifiers can be the result of this representation. Those classifiers might not be accurate, and can cover an area containing some boundaries. It is then the responsibility of the neural network to capture the decision boundaries inside the local area and provide an appropriate outcome when required by the system. This in theory should result in a smaller population size than the traditional univariate rule representation used by UCS.

The author also employs negative correlation learning (NCL) (Liu, 1999) in the proposed neural-based LCS (NLCS) for maintaining diversity in the population in order to improve the system's performance.

This chapter addresses two primary questions in order to answer the second research sub-question, stated in the first chapter:

- Is the proposed neural-based representation beneficial to UCS? The author hypothesizes that the neural-based representation will help to compact the population while still maintaining an equivalent predictive accuracy.

- Can negative correlation learning help NLCS to achieve better accuracy? The author hypothesizes that NCL is able to diversify individuals in the neural

network ensemble in order to improve the overall predictive accuracy. This is the first attempt to apply NCL in LCS; hence it is important to study the effect of NCL on the system.

The chapter is structured as follows. Section 4.2 describes the proposed representation and discusses the framework of NLCS. The experimental setup is explained in Section 4.3. Sections 4.4 and 4.5 investigate each of the above research questions. The final section provides the conclusion of the chapter.

4.2 Neural-based Learning Classifier Systems (NLCS)

In the proposed representation, a rule in NLCS consists of two main components as in traditional LCS: the *condition* and the *action*. The classifier condition can be encoded by any existing representation of LCS such as interval predicates (Wilson, 2001c), kernel-based ellipsoid (Butz, 2005), messy coding (Lanzi and Perrucci, 1999a), S-expression (Lanzi and Perrucci, 1999b), to name a few. In this chapter, the interval representation is used for better expressiveness. The major modification is with respect to the *action* of NLCS, where the traditional class value is replaced by a complete, but simple, neural network (NN).

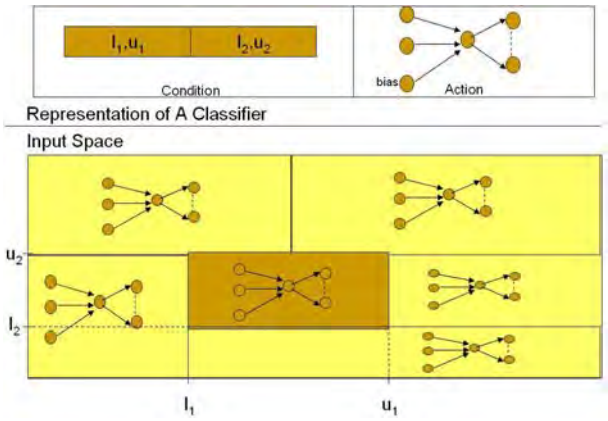

Figure 4.1: An example of a classifier in NLCS. The input space is divided into six local spaces by the upper-bound and lower-bound of classifiers' intervals. A neural network of each classifier is responsible for learning its local area and deciding on the possible outcome of the classifier depending on the input values.

Figure 4.1 illustrates a population of neural-based classifiers for a binary clas-

sification. In this example, the population has six rules. Each rule is responsible for a small input region matching its condition. The neural network of a rule is only fed by inputs belonging to the local region.

The idea of the proposed representation is that the whole input space is divided into relatively small areas by the classifier conditions. Each local region is then handled by one or more NNs.

The following example demonstrates the compactness of the population using neural network for recognizing a fruit. A complete set of rule is as follows:

$$IF(shape = round)and(color = red)and(weight < .50)THEN(itisagrape)$$

$$IF(shape = round)and(color = red)and(weight > .50)THEN(itisanapple)$$

By using the neural network in the action, the above two rules can be merged into the following rule:

$$IF(shape = round)and(color = red)THEN(IFweight > .50(itisanapple)ELSE(itisagrape))$$

The NN is then responsible to distinguish several common objects once the conditions are satisfied. It can easily be seen that expressiveness is lost in the second example.

The type of NN used in this paper is the MLP, which has been shown to be a universal approximator (Hornik *et al.*, 1989). This is a directed graph with several layers, including an input layer, hidden layers, and an output layer. A bias unit is connected to each hidden unit. Static and simple MLPs with only one hidden layer and one hidden node are used in this framework. The number of neurons in the input layer is the same as the number of input features. The outputs are encoded using m output neurons corresponding to m classes. The output node with the highest activation designates the class.

Figure 4.2 shows the simple neural network in each classifier. In this example,

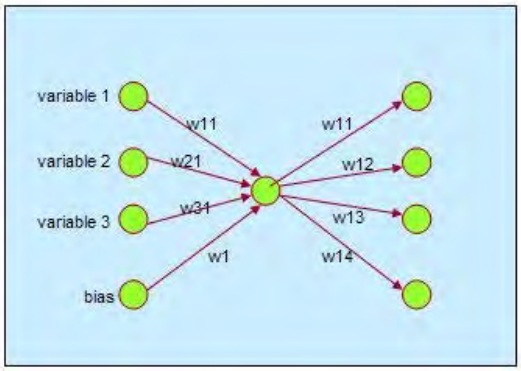

Figure 4.2: A neural network for classification, with three input variables and four possible outcomes.

each input is a vector of three variables and an outcome needs to be drawn from four distinct groups (classes). The decision is made based on the network's knowledge (a set of weights) as the input is fed-forward from the input layer through the network to the output layer. Once the output layer is reached, the output neuron with the highest activation value is chosen as the prediction. Usually, each output neuron is associated with a group (or class) defined by users.

In order to preserve the expressiveness of a rule-based system, simple neural networks with one hidden node are used. The more complex a neural network is the bigger search space it can handle, and therefore the rule can be more general. As a result, the less expressive the system is because the neural network is normally more difficult for a human to understand.

4.2.1 The NLCS Algorithm

The algorithm of NLCS is presented in Algorithm 3. NLCS inherits most parameters from UCS, except the deletion and subsumption parameters. In short, the system is evolved and guided by the discovery and update components. These are described in the next two subsections respectively.

During the training process, the error is computed by Equation 2.17 which compares the system prediction against the desired class. The error is then back-propagated and the network weights are altered as in Equation 2.23. The key goal

```
Initialize NLCS parameters;
Initialize [P] to empty;
repeat
    Given the training data set D = (x_1, y_1),...,(x_N, y_N) from the
    environment;
    for each training instance (x_i : y_i) do
        Form a match set [M] of those classifiers in [P] that match the input
        x_i;
        if [M] is empty then
            if the population size is less than N then
                Create a general classifier C_cover;
                Insert C_cover to [P] and [M];
            else
                Find the closest classifier in the search space and extend its
                condition to cover x_i;
                Insert this classifier to [M];
            end
        end
        if the population size is less than N and at the end of the
        covering_window then
            Create a classifier that matches all misclassified instances during
            the last covering_window;
            Insert the classifier to [P];
        end
        Update the fitness of classifiers in [M];
        Adjust networks' weights of classifiers in [M];
        if the average time since last GA activation of classifiers in [M] is
        higher than θ_GA and the population size is less than N then
            Select and reproduce (mutation and crossover) two classifiers in
            [M];
            Insert offspring in [P];
        end
    end
until the termination conditions are met ;
```

Algorithm 3: Algorithm of NLCS

of back-propagation is to determine a set of weights that minimizes the error in the future.

4.2.2 The Discovery Component

As in UCS, the rule discovery of NLCS is conducted by the covering and evolutionary components.

The covering component is activated if one of the following two conditions is true: (i) the match set *[M]* is empty; or (ii) some instances have been misclassified, and the current population size has not exceeded the predefined maximum population size. The first condition occurs when the system starts with an empty population. The covering function will create the most general rule, which matches all instances or extends a rule to cover this instance. With the latter condition, the system will generate a rule to cover all instances misclassified by the system during a previous covering_window. The size of this window affects the level of generality of newly-created classifiers. Section 4.4.4 investigates in detail the effect of this window size.

The evolutionary component is applied to the correct set *[C]* by selecting two parents, with probability proportional to fitness, for crossover and mutation.

Children are produced by crossing–over and then mutating parents' genes. The offspring are inserted into the population if their genes are distinct from the parents. This process occurs when the mean *experience* of rules in *[C]* is greater than a predefined threshold and the current population size is less than a predefined maximum.

4.2.3 The Update Component

Rules are updated incrementally whenever they appear in the match set *[M]*. Their parameters such as fitness and weights of MLPs are updated appropriately.

As in UCS, the fitness of classifiers is based on their accuracy, which is computed as the ratio of correct classifications by the classifier to the number of times the classifier has been in the match set:

$$acc = \frac{N_{correct}}{N_{matches}} \tag{4.1}$$

The fitness is computed as a function of accuracy:

$$F = (acc)^v \tag{4.2}$$

where v is a predefined constant.

The MLPs learn to perform various tasks by adaptation of their weights using the back-propagation algorithm.

The error of the output at neuron i when presented with the n^{th} training instance is defined by:

$$e_i(n) = y_i(n) - \hat{y}_i(n) \tag{4.3}$$

where $y_i(n)$ refers to the desired outcome and $\hat{y}_i(n)$ is the actual outcome. The instantaneous value of the error of neuron i is $\frac{1}{2}e_i(n)^2$. The instantaneous value of the total error of the network with M classes (M neurons in the output layer) for instance n is:

$$E(n) = \frac{1}{2}\sum_{i=1}^{M} e_i(n)^2 \tag{4.4}$$

The back-propagation algorithm updates the weight w_i connecting the input of neuron i by $\Delta w_i(n)$, which is proportional to the partial derivative $\frac{\partial E(n)}{\partial w_i(n)}$. It can be expressed as:

$$\frac{\partial E(n)}{\partial w_i(n)} = \frac{\partial E(n)}{\partial e_i(n)}\frac{\partial e_i(n)}{\partial \hat{y}_i(n)}\frac{\partial \hat{y}_i(n)}{\partial w_i(n)} \tag{4.5}$$

Differentiating both sides of Equations 4.3 and 4.4, we get

$$\frac{\partial e_i(n)}{\partial \hat{y}_i(n)} = -1 \tag{4.6}$$

$$\frac{\partial E(n)}{\partial e_i(n)} = e_i(n) \tag{4.7}$$

The Δw_i is applied to $w_i(n)$ by the delta rule as following:

$$\Delta w_i(n) = -\beta\frac{\partial E(n)}{\partial w_i(n)} \tag{4.8}$$

where β is the learning rate parameter. Thus, the correction Δw used for a single update of each weight in the network is

$$\Delta w_i(n) = \beta e_i(n)\frac{\partial \hat{y}_i(n)}{\partial w_i(n)} \tag{4.9}$$

Equation 4.9 indicates that the calculation of Δw depends on the error function at the output neuron i. There are two ways to calculate $e_i(n)$ depending on its layer. Equation 4.3 can be used to compute the error of neurons at the output layer. If a neuron is a hidden node, the error needs to be computed recursively in terms of the errors of all neurons to which that hidden node is directly connected.

4.3 Experimental Setup

UCS and NLCS are developed in C++. If not stated differently, UCS is setup as follows: $v = 5, \theta_{GA} = 50, \chi = 1, \mu = 0.04, \theta_{del} = 50, \theta_{sub} = 50, m_0 = 0.1, s_0 = 0.6, N = 6400$. Two points crossover and roulette wheel selection are used. For NLCS, each MLP has one hidden layer and one hidden node. The learning rate of MLPs is $\beta = 0.1$. Other parameters are: $v = 5, \theta_{GA} = 50, \chi = 1, \mu = 0.04, m_0 = 0.1, s_0 = 0.1, covering\_window = 50, \gamma = 0.5$.

Each learner performs three stratified 10-fold cross validations in each data set (that is 30 runs). Each run uses different random seeds which are consistent in all experiments. The results reported in this paper are averaged over those 30 runs. The term *iteration* is used to refer to a single pass through the training set. The statistical data is collected after 500 iterations in each experiment.

4.4 An Investigation of the Neural-based Representation

This section is designed to answer the first research question about NLCS. The author starts by investigating the effect of the neural representation on UCS, followed by a comparison of UCS and NLCS in terms of accuracy and compactness. Several factors such as the population size, the size of the covering window, and the conflict resolution methods among NNs in the match set are also considered to provide a better understanding of NLCS.

4.4.1 Effects of the neural representation

To understand the effect of the neural representation on UCS, we need to compare the prediction accuracy of UCS with the traditional representation and the proposed neural representation.

For the first experiment, the genetic algorithm is switched off and a simple covering technique is used. If covering alone is sufficient to give high accuracy, the whole UCS mechanism within NLCS is not useful.

The top half of Table 4.1 presents the mean and the standard deviation (over 30 runs) of the predictive accuracy of the traditional and neural representations without GA. The statistical test of significance used is the t–test with a significance level of 0.05.

There is a statistically significant improvement in accuracy with the neural representation, in nine of the fourteen data sets (balance-scale, bupa, credit-a, glass, heart-statlog, iris, segment, tao, and vowel). The opposite trend is not observed in any data set. Moreover, ten data sets (balance-scale, bupa, credit-a, diabetes, ionosphere, iris, segment, sonar, tao, and vowel) have a smaller standard deviation with the neural representation. This experiment suggests that the neural representation has an advantage over the traditional representation, with improved and more consistent prediction accuracy.

The performance on several data sets is bad (e.g. heart-statlog (31%), ionosphere (29%), sonar (4%)). This can be attributed to disabling the GA component, which means the input space is explored only by the covering technique. If the covering process is not able to generalize between instances, the system will not be able to handle test cases in areas of the input space that are not represented in the training set.

The bottom half of Table 4.1 presents the mean and the standard deviation of the predictive accuracy of the traditional and neural representations with GA enabled. It can be easily observed that the accuracy of both UCS and NLCS with GA are better than accuracy without GA. This confirms that covering alone is not

Table 4.1: The mean and standard deviation of accuracy of NLCS and UCS with/without GA. ■(□) symbols indicate that the neural representation is better(worse) than the traditional representation at a significance level of 0.05.

	No GA	
Problem	UCS	NLCS
balance-scale	0.788 ± 0.040	0.825 ± 0.034 ■
breast-w	0.759 ± 0.033	0.761 ± 0.040
bupa	0.549 ± 0.077	0.612 ± 0.066 ■
credit-a	0.716 ± 0.064	0.796 ± 0.048 ■
diabetes	0.676 ± 0.043	0.693 ± 0.043
glass	0.491 ± 0.079	0.555 ± 0.094 ■
heart-statlog	0.275 ± 0.091	0.319 ± 0.104 ■
ionosphere	0.288 ± 0.064	0.298 ± 0.064
iris	0.809 ± 0.128	0.929 ± 0.052 ■
lymph	0.566 ± 0.114	0.563 ± 0.126
segment	0.810 ± 0.036	0.829 ± 0.033 ■
sonar	0.045 ± 0.050	0.042 ± 0.047
tao	0.768 ± 0.087	0.847 ± 0.029 ■
vowel	0.487 ± 0.050	0.575 ± 0.050 ■

	With GA	
Problem	UCS	NLCS
balance-scale	0.815 ± 0.038	0.886 ± 0.022 ■
breast-w	0.969 ± 0.012	0.970 ± 0.015
bupa	0.683 ± 0.062	0.723 ± 0.055 ■
credit-a	0.835 ± 0.028	0.860 ± 0.032 ■
diabetes	0.748 ± 0.044	0.765 ± 0.042
glass	0.825 ± 0.078	0.830 ± 0.079
heart-statlog	0.648 ± 0.116	0.612 ± 0.071
ionosphere	0.729 ± 0.051	0.874 ± 0.065 ■
iris	0.949 ± 0.042	0.949 ± 0.067
lymph	0.759 ± 0.112	0.846 ± 0.094 ■
segment	0.968 ± 0.008	0.809 ± 0.055 □
sonar	0.736 ± 0.073	0.799 ± 0.073 ■
tao	0.887 ± 0.015	0.867 ± 0.023 □
vowel	0.910 ± 0.031	0.560 ± 0.072 □

sufficient for accurate predictions, and that GA indeed plays a positive role in both UCS and NLCS.

NLCS performs significantly better than UCS on six data sets: the balance scale, bupa, Australian credit card, ionosphere, lymph, and sonar problems. Both systems have achieved similar accuracy on five data sets: breast cancer, diabetes, heart statlog, glass, and iris. In three of these five the accuracy is marginally better with NLCS, but the difference is not statistically significant.

If we take a closer look at the result, the accuracy gain typically lies in the range [0.02, 0.08], as in balance-scale, bupa, credit-a, lymph, and sonar. Ionosphere shows the biggest gain of around 0.14.

UCS, on the other hand, does significantly better than NLCS on three data sets: segment, vowel, and tao. The segment and vowel problems have many classes (7 classes for the segment and 11 classes for the vowel) and high number of features (19 features for the segment and 13 features for the vowel). The tao problem challenges NLCS due to its non-linear boundaries in the data.

The author hypothesizes that the reason for NLCS's low performance on these three data sets is the use of a small population size, and low training time for the neural network.

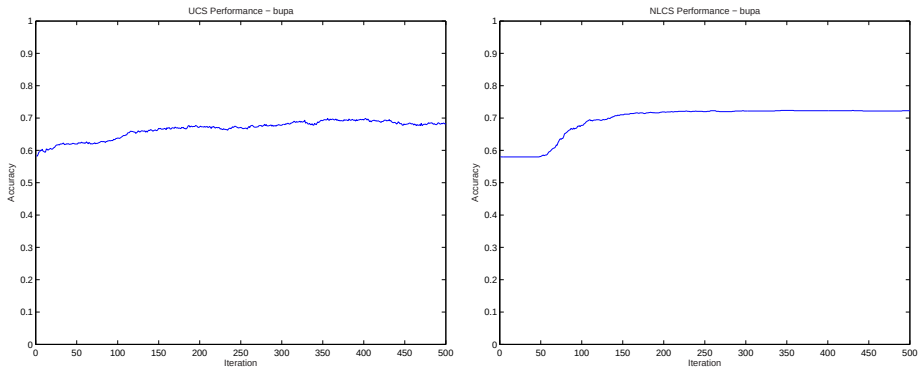

Figure 4.3: The learning curves of UCS (left) and NLCS (right) over time. An iteration refers to a single pass of the training data set through the system.

NLCS learns more slowly than UCS in many data sets such as bupa, glass, iris, lymph, and vowel. Figure 4.3 shows the learning curves of both UCS and NLCS over time, for one illustrative data set. Each iteration is a complete feed of a training

set (or exploration phase) following by a testing set (or exploitation phase).

NLCS normally requires extra time at the beginning for adjusting the weights of neural networks; hence the slow climb in accuracy at the start of the training for NLCS. UCS, on the other hand, has only one action and therefore does not need this time. Training neural networks is slowest in problems with many classes, as a high number of weights needs to be adjusted. This is seen on the segment and vowel problems, for which accuracy grows gradually at the beginning. Longer training times are needed for problems with many features and classes. (This is investigated further in Section 4.5.3.)

Overall, the results show that NLCS has obtained an equivalent or better predictive accuracy than UCS on most of the tested data sets. Hence, a neural-based representation of the classifiers' action can be considered to be applicable, with good accuracy, to be used as a LCS for classification tasks.

4.4.2 The Effect of Population Size

UCS requires a few thousand classifiers to capture the whole input space on each problem. The results confirm the finding by Butz (2005) that lower and upper boundaries in the classifiers require a large population size to approximate non-linear classification boundaries such as those in real-world data sets.

The results for NLCS in Table 4.1 are all obtained using a population of only 25 macro-classifiers, which even so yields similar or better predictions than UCS in many cases. It is clear that the neural representation can help to compact the population.

A classifier generated by the covering technique can be either general or specific. Since the covering spread used in the experiments is 0.6, it is likely to create more general classifiers than specific ones. The traditional representation allows each classifier to have one action, while the proposed method allows all possible actions to be associated with a classifier. If a classifier is too general, more than one class can be observed in the area. The traditional representation can only predict one

action correctly, while the neural representation can predict multiple actions as long as enough training is provided. That might explain why the neural representation helps to compact the population, by allowing general classifiers in the population instead of only accurate classifiers as in UCS.

To understand the effect of the population size on the performance of NLCS, the author experimented with population sizes of 1, 10, 25, 50, 75, and 100. Table 4.2 presents the performance of NLCS with different population sizes.

With many data sets there is a slight trend in favour of larger population sizes. In most cases, though, the difference is small and not statistically significant.

A larger population size is clearly helpful with the vowel and segment problems. This supports the hypothesis that the low accuracy of NLCS on segment, tao, and vowel problems is due to the population size.

In essence, the population size of 25 classifiers yields an acceptable performance on most data sets. From this point on in this chapter, all experiments use a population of 25 rules unless stated differently.

The population size has an impact on execution time. LCS starts by first comparing the condition of all classifiers in the population against each input instance. A population of m classifiers would require m times of comparison against the whole input instance. If there are n features in the input, we need up to mn comparisons for each input instance. Increasing m raises dramatically the computational cost of LCS. Thus, it clearly indicates that NLCS is better than UCS in terms of computation time. However, the cost of NLCS to back-propagate the error in order to adapt the weights of its MLPs is also added. Since there are only 25 classifiers in the population and even less in the match set, the author expects this time to be much smaller than the comparison time. In addition, with a simple neural network architecture, back-propagation can be optimized to work faster than the traditional genetic algorithm. The population size of 25 classifiers would be much faster in terms of execution time in comparison to thousands of classifiers.

Table 4.2: The mean and standard deviation of accuracy of NLCS with different population sizes. ■ symbol means that accuracy with that population size is better than with a population size of 25 at a significance level of 0.05.

Population	1	10	25
balance-scale	0.865 ± 0.023	0.884 ± 0.021	0.886 ± 0.022
breast-w	0.966 ± 0.015	0.969 ± 0.015	0.970 ± 0.015
bupa	0.714 ± 0.050	0.726 ± 0.057	0.723 ± 0.055
credit-a	0.858 ± 0.040	0.857 ± 0.034	0.860 ± 0.032
diabetes	0.767 ± 0.047	0.766 ± 0.047	0.765 ± 0.042
heart-statlog	0.816 ± 0.078	0.832 ± 0.085	0.830 ± 0.079
glass	0.597 ± 0.073	0.607 ± 0.083	0.612 ± 0.071
ionosphere	0.863 ± 0.061	0.865 ± 0.069	0.874 ± 0.065
iris	0.967 ± 0.034	0.944 ± 0.066	0.949 ± 0.067
lymph	0.842 ± 0.101	0.844 ± 0.099	0.846 ± 0.094
segment	0.581 ± 0.037	0.780 ± 0.069	0.809 ± 0.055
sonar	0.785 ± 0.075	0.797 ± 0.082	0.799 ± 0.073
tao	0.842 ± 0.014	0.863 ± 0.023	0.867 ± 0.023
vowel	0.415 ± 0.046	0.523 ± 0.065	0.569 ± 0.072

Population	50	75	100
balance-scale	0.889 ± 0.021	0.890 ± 0.022	0.891 ± 0.022
breast-w	0.968 ± 0.016	0.967 ± 0.015	0.968 ± 0.016
bupa	0.721 ± 0.055	0.721 ± 0.053	0.723 ± 0.054
credit-a	0.861 ± 0.032	0.859 ± 0.032	0.857 ± 0.033
diabetes	0.764 ± 0.041	0.766 ± 0.043	0.766 ± 0.043
heart-statlog	0.826 ± 0.080	0.825 ± 0.078	0.824 ± 0.076
glass	0.607 ± 0.078	0.607 ± 0.064	0.613 ± 0.059
ionosphere	0.874 ± 0.068	0.870 ± 0.068	0.875 ± 0.066
iris	0.953 ± 0.056	0.962 ± 0.038	0.964 ± 0.038
lymph	0.848 ± 0.094	0.855 ± 0.092	0.855 ± 0.092
segment	0.831 ± 0.044 ■	0.858 ± 0.041 ■	0.847 ± 0.043 ■
sonar	0.810 ± 0.075	0.812 ± 0.077	0.815 ± 0.077
tao	0.871 ± 0.020	0.872 ± 0.020	0.874 ± 0.022
vowel	0.614 ± 0.069 ■	0.632 ± 0.061 ■	0.632 ± 0.060 ■

4.4.3 The Match set [M]

Figure 4.4 plots the average match set size in proportion to the population size. The trend is that as the population size increases, the size of the match set decreases as a proportion of the total population size. This is associated with a reduction in the variance, thus the networks are almost distributed uniformly.

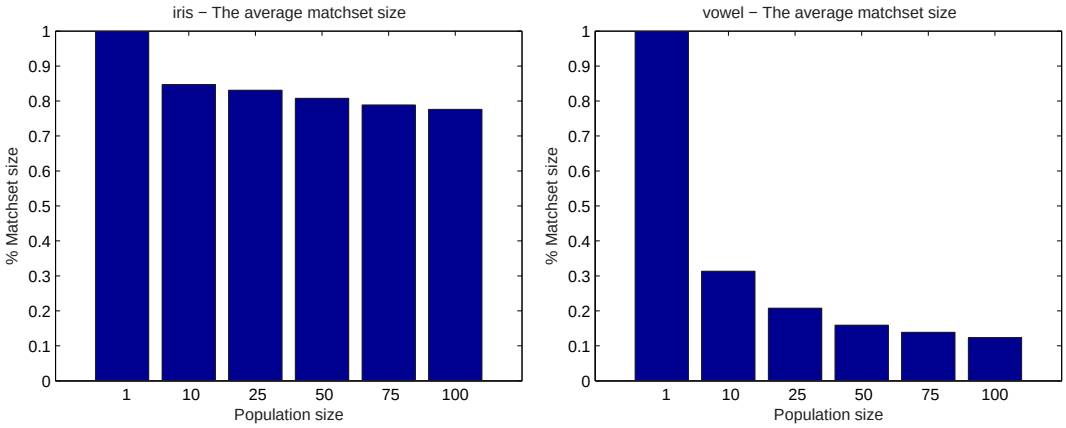

Figure 4.4: The average match set size as a proportion of the population size in NLCS on the training data set

4.4.4 The Effect of the Covering Window

The covering approach is used in NLCS in two cases: when the match set *[M]* is empty, and when it fails to classify a training instance.

In the first case, a new classifier is created to cover an instance that the system currently has no knowledge about. The covering classifier is added straight to the population and the match set. This process happens mainly at the beginning of the learning cycle. If the population is full, a closest existing classifier is extended to cover the new instance.

The second case happens when the system produces incorrect predictions. The covering approach in this case aims to put extra knowledge of the area into the system, hoping that instances around that area will be predicted correctly in the future. Since the system already holds some knowledge about the instance (because *[M]* is not empty), it is not as urgent to insert a new classifier into the population.

Table 4.3: The mean accuracy and standard deviation of NLCS with different covering window sizes.

Window Size	10	50	75	200
balance-scale	0.90 ± 0.02	0.89 ± 0.02	0.89 ± 0.02	0.89 ± 0.02
breast-w	0.97 ± 0.01	0.97 ± 0.02	0.97 ± 0.02	0.96 ± 0.02
bupa	0.72 ± 0.06	0.72 ± 0.06	0.73 ± 0.06	0.72 ± 0.05
credit-a	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.03	0.86 ± 0.04
diabetes	0.76 ± 0.04	0.77 ± 0.04	0.76 ± 0.04	0.77 ± 0.05
glass	0.60 ± 0.08	0.61 ± 0.07	0.61 ± 0.08	0.60 ± 0.07
heart-statlog	0.83 ± 0.07	0.83 ± 0.08	0.83 ± 0.08	0.83 ± 0.09
ionosphere	0.88 ± 0.06	0.87 ± 0.06	0.87 ± 0.06	0.87 ± 0.07
iris	0.96 ± 0.04	0.95 ± 0.07	0.96 ± 0.05	0.97 ± 0.04
lymph	0.83 ± 0.08	0.85 ± 0.09	0.85 ± 0.10	0.84 ± 0.10
segment	0.89 ± 0.03	0.81 ± 0.05	0.79 ± 0.04	0.76 ± 0.05
sonar	0.79 ± 0.07	0.80 ± 0.07	0.81 ± 0.08	0.80 ± 0.07
tao	0.89 ± 0.02	0.87 ± 0.02	0.86 ± 0.02	0.85 ± 0.02
vowel	0.65 ± 0.04	0.57 ± 0.07	0.51 ± 0.08	0.46 ± 0.07

Instead the author uses a covering window, which represents a particular number of training instances: at the end of each window, a new classifier is added to cover all instances that were classified incorrectly during that window. The size of the covering window affects the generality of the classifier: the larger the window, the more general the new classifier becomes as it must cover more instances.

Table 4.3 shows the accuracy achieved on each data set with four different sizes of covering window: 10, 50, 75 and 200 instances. On most data sets, the covering window does not seem to have much effect on the predictive accuracy of the system.

Once again, tao, segment and vowel behave differently. A small covering window, which means specific classifiers are more likely to be added than more general classifiers, is clearly better for these problems.

For the tao problem, this is explained by the problem containing non-linear boundaries. Specific classifiers are more accurate than general classifiers in this situation, because they are responsible for a smaller area. For vowel and segment, the explanation could be that it is hard for general classifiers to be accurate in problems with many output classes, so there is a preference for specific classifiers.

The covering window of size 10 or 50 tends to obtain good performance in all

data sets.

4.4.5 Neural Network Ensembles

This section addresses the question of how to combine knowledge from different rules in the match set in the exploitation phase. The author investigates three simple methods, which are quite popular for decision making at the gate level in ensemble learning: majority voting, simple averaging, and winner take all.

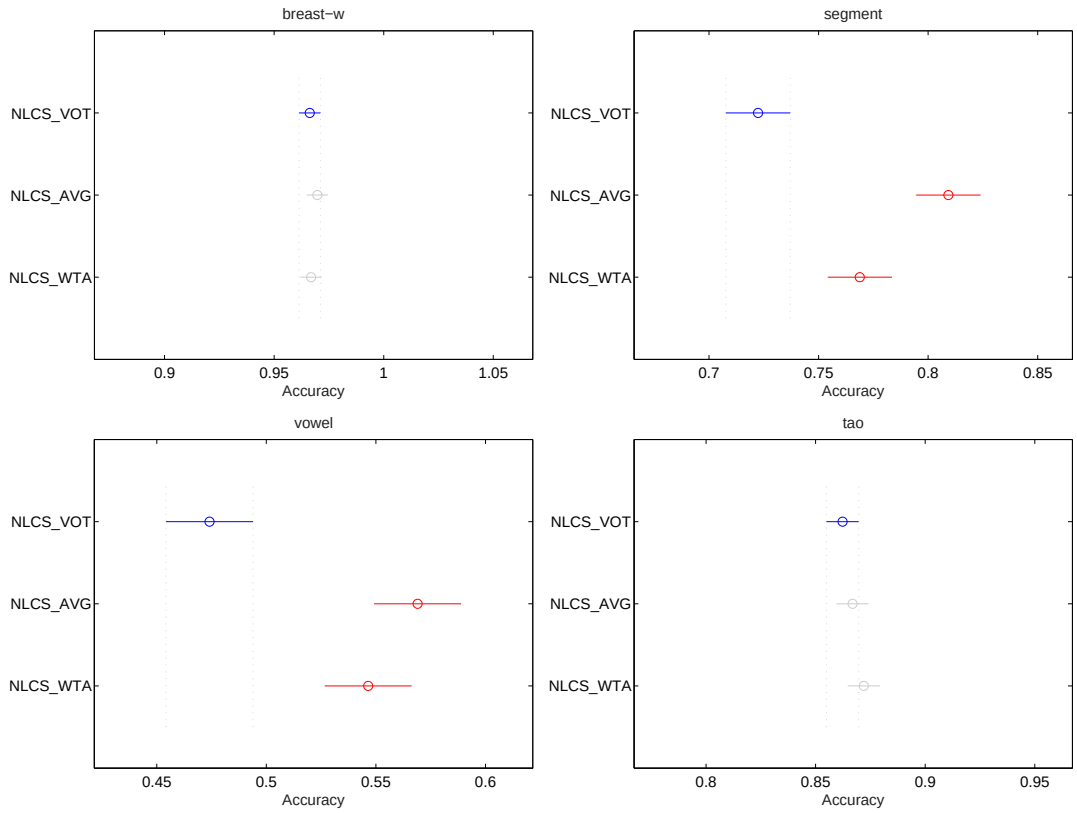

Figure 4.5: Predictive accuracy of NLCS using different gates: majority voting (VOT), simple averaging (AVG), winner-takes-all (WTA)

Each of these three methods was tested on the fourteen data sets. ANOVA tests were used to determine whether there were statistically significant differences in prediction accuracy using the different gates. There were statistically significant differences in three data sets: segment, vowel, and tao. Figure 4.5 shows predictive accuracy with each gate on these four data sets.

The segment and vowel problems show the best accuracy when using the simple

averaging approach. The winner-takes-all method is the second best, significantly outperforming the majority voting method.

The tao problem favours the winner-take-all approach, because this favours the more specific classifiers that suit a problem with non-linear boundaries. The difference is small though; it is statistically significant because it is consistent rather than because it is large.

There is no significant difference in the accuracy on the other eleven data sets. Six (breast-w, credit-a, diabetes, glass, iris, and lymph) do best with the simple averaging method; three (balance-scale, ionosphere, and sonar) do worst with simple averaging. The differences in mean accuracy with the different gates are small.

In summary, in the only data sets where the gate makes an important difference to mean accuracy, the simple averaging approach performs best. Hence, from this point in this paper, the author only report the results of NLCS using the simple averaging approach.

4.4.6 Comparison to Other Classifier Systems

Table 4.4 shows the predictive performance of 14 datasets on UCS, NLCS and three other popular classifiers including Majority, C4.5, and Naive Bayes. The accuracy of the last three classifiers was published in (Butz, 2004) using the WEKA system Garner (1995).

Clearly, NLCS is significantly better than Majority algorithm in all problems. The performance of NLCS is better than C4.5 and Naive Bayes in nine problems, while is lower in other six problems. In general, NLCS is comparable with other existing learning classifiers in terms of the predictive accuracy.

4.4.7 Summary

This section has investigated the neural representation in learning classifier systems. The author tested NLCS on several data sets under several conditions.

Table 4.4: Comparing NLCS's performance with other typical machine learning algorithms. The mean accuracy and standard deviation of Majority, C4.5, Naive Bayes and NLCS.

Problem	Majority	C4.5	Naive Bayes	NLCS
balance-scale	0.461 ± 0.002	0.779 ± 0.040	0.905 ± 0.018	0.886 ± 0.022
breast-w	0.655 ± 0.003	0.945 ± 0.026	0.960 ± 0.021	0.970 ± 0.015
bupa	0.580 ± 0.008	0.650 ± 0.086	0.548 ± 0.083	0.723 ± 0.055
credit-a	0.555 ± 0.004	0.854 ± 0.042	0.778 ± 0.042	0.860 ± 0.032
diabetes	0.651 ± 0.003	0.742 ± 0.046	0.756 ± 0.049	0.765 ± 0.042
glass	0.356 ± 0.014	0.674 ± 0.088	0.481 ± 0.082	0.830 ± 0.079
heart-statlog	0.556 ± 0.000	0.778 ± 0.080	0.839 ± 0.064	0.612 ± 0.071
ionosphere	0.641 ± 0.006	0.900 ± 0.053	0.825 ± 0.071	0.874 ± 0.065
iris	0.330 ± 0.000	0.948 ± 0.059	0.955 ± 0.051	0.949 ± 0.067
lymph	0.550 ± 0.029	0.777 ± 0.111	0.830 ± 0.086	0.846 ± 0.094
segment	0.143 ± 0.000	0.968 ± 0.011	0.801 ± 0.018	0.809 ± 0.055
sonar	0.534 ± 0.012	0.738 ± 0.085	0.678 ± 0.096	0.799 ± 0.073
tao	0.500 ± 0.000	0.956 ± 0.013	0.809 ± 0.029	0.867 ± 0.023
vowel	0.091 ± 0.000	0.801 ± 0.036	0.632 ± 0.049	0.560 ± 0.072

The experiments show that:

- The neural representation does help to improve predictive accuracy, compared to the traditional representation;
- A search method like genetic algorithms is essential to cover the input space;
- NLCS is able to obtain equal or even better accuracy than UCS, while requiring a much smaller population;
- Increasing the population size generally improves the accuracy in NLCS, but a population of only about 25 classifiers seems to be sufficient for acceptable accuracy;
- The size of the covering window does not have a major effect on the overall predictive performance of NLCS.
- To resolve conflicts between neural networks in the match set, the simple averaging approach is the best choice.

4.5 Negative Correlation Neural Learning based Classifier Systems

4.5.1 Diversity and Accuracy in Ensembles

The notion of combining networks to form more reliable ensembles was first raised by Nilsson (1965). Ensembles normally perform better than single networks by minimizing loss of information due to bias. Sharkey (1996) defined the notion of combining NNs for improving the performance in that we try exploiting rather than losing the information contained by imperfect networks.

In order for an ensemble to generalize well, two vital factors need to be considered: diversity and accuracy. Brown *et al.* (2005) suggests that two neural networks are diverse if they make different errors on the same data points/inputs. Accuracy refers to how good the learning model is in comparison to random guessing on a new input (Brown *et al.*, 2005).

In LCS, the training data is inherently re-sampled by classifier conditions. Each NN is trained by partial data which belongs to its local region. Therefore, LCSs implicitly maintain diversity in the population without using Bagging or Boosting techniques.

Each time a rule appears in the match set *[M]*, its weights and fitness are updated. This raises another question in this research: how will the system perform if those individual networks in the match set are trained interactively like (Islam *et al.*, 2003), (Liu, 1999), etc? It is similar to anti-correlation learning, where members in the ensemble are trained interactively so that each member specializes on a part of the task. Anti-correlation learning adds an additional penalty term into the error function in order to maximize the distance between all individuals in an ensemble to achieve a nice spread in the ensemble space. The negative correlation term should not have a larger magnitude than the original error function and is dimensionally consistent with the error function. Negative correlation learning (NCL) (Liu, 1999) is an approach of this type. McKay and Abbass (2001) reveal

that negative correlation learning acts to push the members of the ensemble away from their mean, but not necessarily away from each other. The author decided to blend NCL with NLCS by adding a negative term into the error function before back-propagating in those classifiers in the match set *[M]*.

4.5.2 The Effect of Negative Correlation Learning

The second research question in this chapter is to find out if NCL helps NLCS to achieve better accuracy. This section focusses on answering this research question by learning the effects of NCL on NLCS. In other words, the strength parameter γ of NCL is investigated in several experiments, each with different values of γ in the range $[0.0, 0.7]$. $\gamma = 0.0$ implies that NCL is not used in the system, which was the case in all the experiments reported in Section 4.4.

Table 4.5 shows accuracy achieved on each data set with each value of γ from 0.0 to 0.7. The statistical t-test was used to compare the means of predictive accuracy between NLCS with NCL ($\gamma > 0$) and NLCS without NCL ($\gamma = 0$).

In five out of fourteen data sets, NCL produces a statistically significant improvement in predictive accuracy. Within those five data sets, there is little significant difference in the performance when γ is small ($\gamma = 0.1$). As γ increases, we can observe significant improvement. However, there is also a significant decrease in the performance in most of the data sets when γ is high ($\gamma > 0.6$).

In general, five out of the fourteen data sets present an identifiable trend: increasing γ results in improvement of the predictive performance, up to a particular point, beyond which the performance starts decreasing. This result is consistent with the finding by Brown (2004) as he believes that the higher γ will push individuals be far away from each other and therefore destroys the correlation of classifiers in the match set.

Some problems such as balance-scale, bupa, diabetes, glass, segment, and vowel suffer a huge decrease in accuracy when γ surpasses 0.6. With other data sets the decrease in accuracy is more gradual when γ is bigger than 0.6.

Table 4.5: The mean and standard deviation of accuracy of NLCS with different values of γ. ■(□) indicate that NLCS with NCL is better(worse) than the one without NCL ($\gamma = 0$) at a significance level of 0.05

γ	0.0	0.1	0.2	0.3
balance-scale	0.89±0.02	0.89±0.02	0.90±0.02■	0.90±0.02■
breast-w	0.97±0.02	0.97±0.01	0.97±0.01	0.97±0.01
bupa	0.72±0.06	0.73±0.05	0.72±0.06	0.73±0.05
credit-a	0.86±0.03	0.86±0.03	0.86±0.03	0.86±0.03
diabetes	0.77±0.04	0.76±0.04	0.77±0.05	0.77±0.04
glass	0.61±0.07	0.61±0.07	0.62±0.07	0.62±0.05
heart-statlog	0.83±0.08	0.83±0.08	0.83±0.08	0.84±0.08
ionosphere	0.87±0.06	0.87±0.06	0.88±0.07	0.88±0.06
iris	0.95±0.07	0.95±0.06	0.95±0.06	0.96±0.05
lymph	0.85±0.09	0.85±0.10	0.85±0.09	0.85±0.09
segment	0.81±0.05	0.85±0.04■	0.88±0.03■	0.90±0.02■
sonar	0.80±0.07	0.80±0.08	0.82±0.07	0.82±0.07
tao	0.87±0.02	0.87±0.02	0.88±0.02■	0.89±0.02■
vowel	0.57±0.07	0.61±0.05■	0.62±0.06■	0.63±0.06■
γ	0.4	0.5	0.6	0.7
balance-scale	0.90±0.02■	0.90±0.01■	0.90±0.01■	0.79±0.07□
breast-w	0.97±0.01	0.97±0.01	0.97±0.02	0.96±0.02□
bupa	0.73±0.06	0.73±0.06	0.58±0.04□	0.58±0.04□
credit-a	0.86±0.03	0.86±0.04	0.86±0.03	0.85±0.03
diabetes	0.77±0.04	0.77±0.04	0.74±0.05□	0.66±0.02□
glass	0.61±0.06	0.60±0.07	0.45±0.06□	0.36±0.07□
heart-statlog	0.83±0.08	0.84±0.08	0.83±0.06	0.81±0.07
ionosphere	0.87±0.07	0.88±0.06	0.87±0.06	0.83±0.08□
iris	0.96±0.05	0.96±0.05	0.94±0.07	0.87±0.12□
lymph	0.84±0.08	0.84±0.08	0.84±0.08	0.85±0.08
segment	0.92±0.02■	0.92±0.02■	0.76±0.07□	0.44±0.07□
sonar	0.83±0.07■	0.83±0.06■	0.81±0.07	0.75±0.08□
tao	0.89±0.02■	0.89±0.02■	0.88±0.02■	0.81±0.05□
vowel	0.64±0.05■	0.66±0.05■	0.67±0.06■	0.27±0.07□

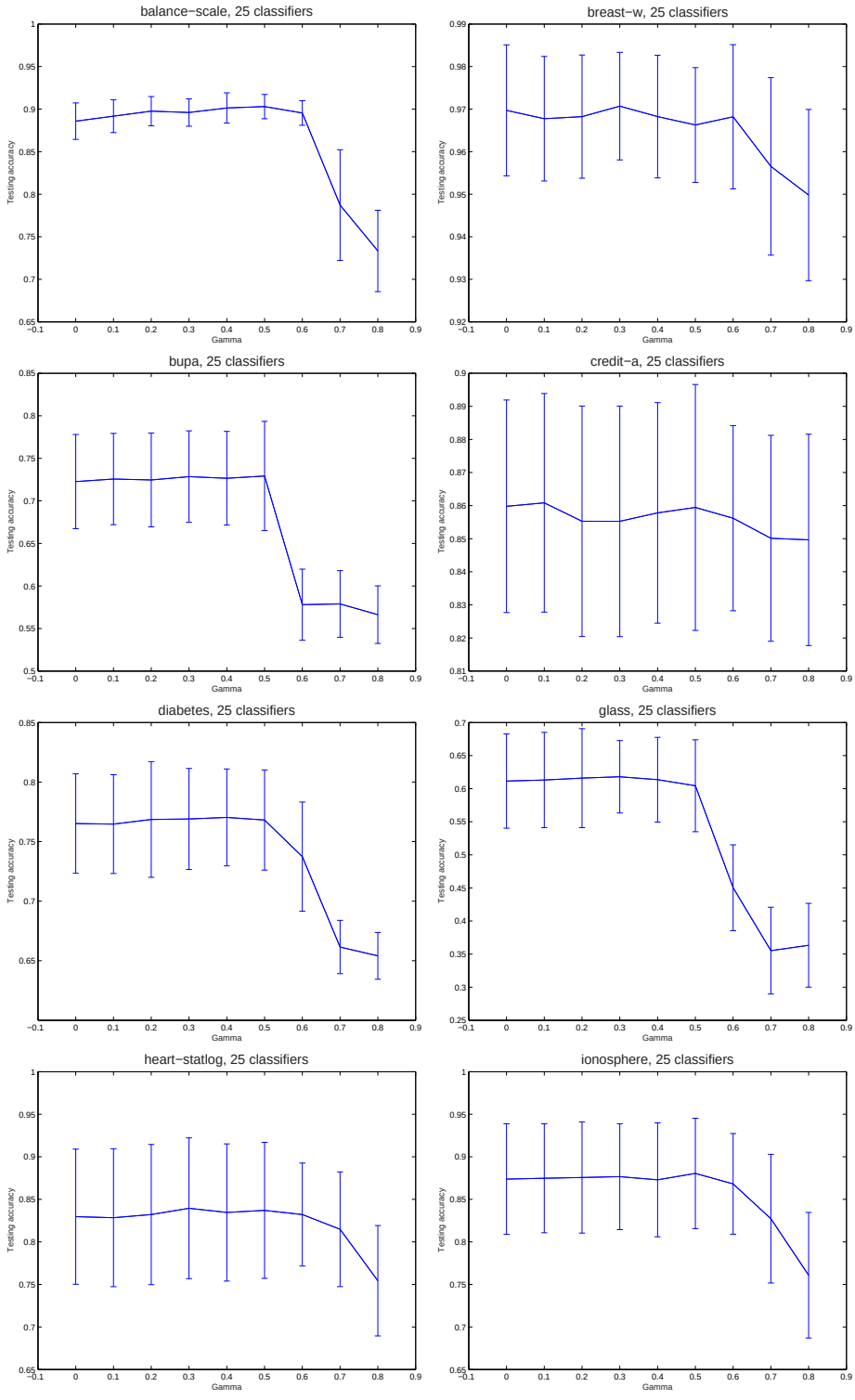

Figure 4.6: The mean and error bar of the predictive accuracy of NLCS with regard to different values of the parameter γ

In conclusion, NCL does help to improve the accuracy of NLCS for medium γ values. If γ is greater than 0.6, it yields a dramatic decrease in the performance

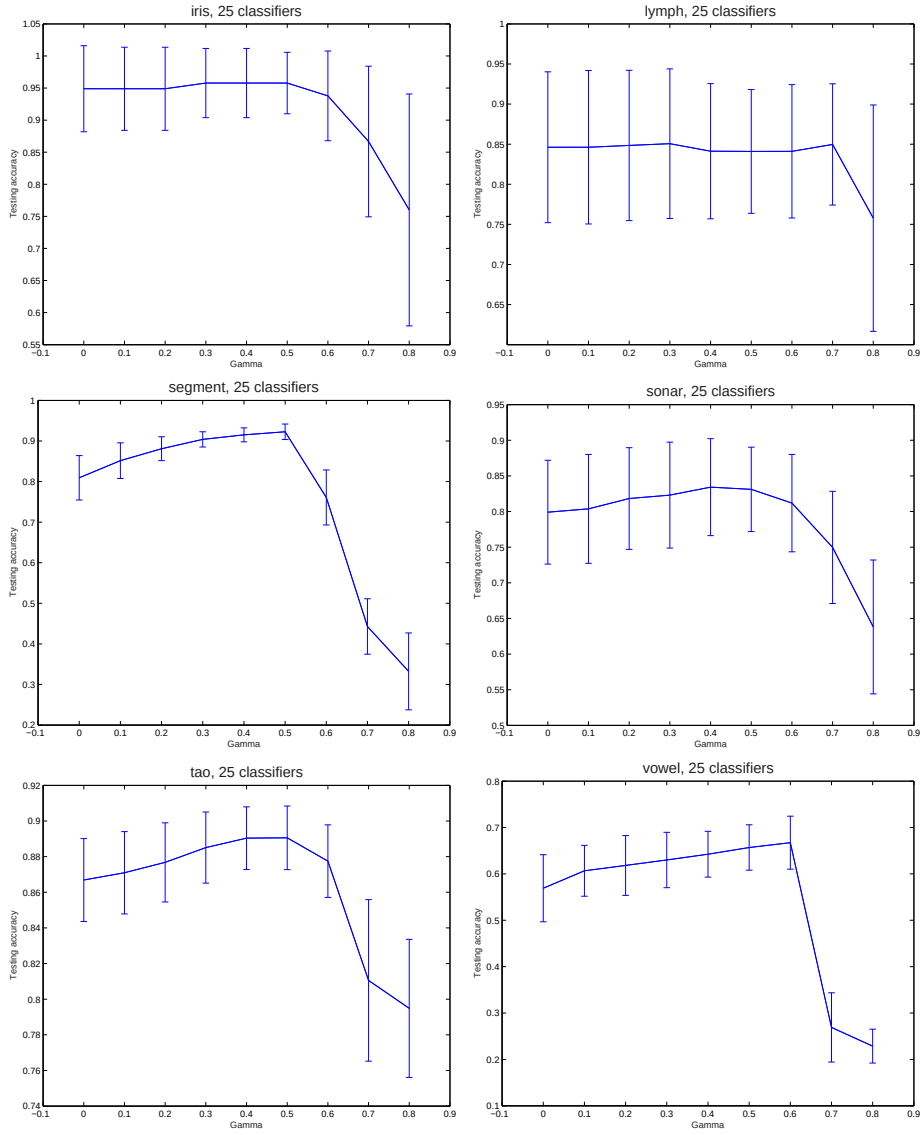

Figure 4.7: The mean and error bar of the predictive accuracy of NLCS with regard to different values of the parameter γ

of the system. Thus, the key requirement for achieving better performance using NCL in NLCS is to tune the γ parameter carefully. As suggested by Brown (2004) and also from the experimental results, $\gamma = 0.5$ seems to give the best accuracy.

4.5.3 Population Size and Training Epochs

This section investigates three special data sets — segment, tao, and vowel — because of the low performance of NLCS compared to the traditional UCS.

Table 4.6: The mean and standard deviation of accuracy of NLCS on segment, tao and vowel problems with different values of γ and the population size of 100. ■(□) symbols indicate that NLCS with NCL is better(worse) than the one without NCL at a significance level of 0.05

γ	0.0	0.1	0.2	0.3
segment	0.84±0.04	0.88±0.03■	0.89±0.03■	0.92±0.02■
tao	0.87±0.02	0.88±0.02	0.89±0.02■	0.90±0.01■
vowel	0.62±0.05	0.64±0.05	0.64±0.06	0.65±0.06■
γ	0.4	0.5	0.6	0.7
segment	0.93±0.02■	0.93±0.02■	0.75±0.06□	0.58±0.10□
tao	0.90±0.01■	0.90±0.02■	0.88±0.02■	0.83±0.02□
vowel	0.68±0.04■	0.74±0.05■	0.28±0.04□	0.26±0.04□

NCL helps to improve significantly the predictive accuracy of these data sets especially when $\gamma = 0.5$ as shown in Table 4.5. The accuracy of the segment problem raises from 81% without NCL to 92% with NCL. The accuracy of the tao problem increases from 87% to 89%. The accuracy of the vowel problem increases from 57% to 66%. The tao problem seems to be able to achieve the same level of accuracy as UCS (average accuracy of UCS is 88% and average accuracy of NLCS is 89%). However the accuracy of the segment problem is still worse than UCS at 92% compared to 96%. Also, the accuracy of the vowel problem is much worse than UCS at 66% compared to 91%.

The author hypothesizes that NLCS performs significantly worse than UCS on these data sets due to two main reasons: small population size and small number of training epochs. We noted in section 4.4.1 that NLCS learns more slowly than UCS in several data sets.

The vowel and segment problems are similar in that they have many possible outcomes (seven for segment and eleven for vowel), and many learning features (nineteen for segment and thirteen for vowel). Therefore the number of weights of each neural network increases dramatically in both problems. It requires more time for each neural network to adjust these weights correctly before they can be used. The tao problem, in contrast, is a binary classification problem with a small number of features. Its challenge is the non-linear boundaries that favour specific classifiers.

Table 4.6 presents the mean and standard deviation of accuracy on these three data sets using a larger population size (of 100). Accuracy improves on all three data sets (as also seen earlier in table 4.2), and peaks at $\gamma = 0.5$. The tao problem reaches the same level of accuracy as UCS. However, the performance of NLCS on the segment and vowel problems is still not as good as UCS.

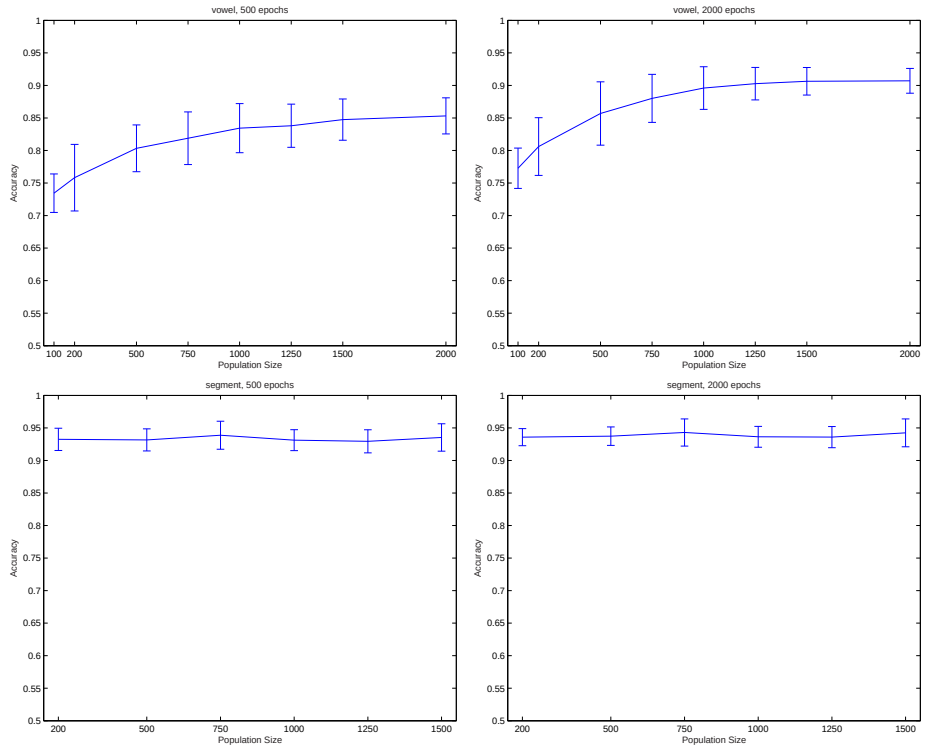

Figure 4.8: Predictive accuracy of NLCS with different population sizes and training time

Figure 4.8 shows the mean and standard deviation of accuracy on the vowel problem, with different population sizes and different numbers of training epochs. Increasing the population size clearly leads to an improvement in the predictive accuracy.

Moreover, the number of epochs (or the number of times a whole training set is fed through the system) has a strong influence on the accuracy of the vowel problem. As mentioned before, neural networks in the vowel problem are more complex than in other problems since there are more weights to tune. In order for each neural network to gain enough information, a longer training time is needed. As we can see, feeding training data through the system 2000 times instead of 500

times improves the accuracy, to a level that now matches UCS.

A similar pattern applies for segment, though it is not as strong and still does not quite achieve the accuracy of UCS.

Another factor that would degrade the performance of the system is over-fitting. In fact, NLCS employs single hidden networks, which are very simple and would not be over-fitted. Also, the generalization pressure in non-enhanced UCS/LCS systems prevents it from over-fitting as presented in (Shafi *et al.*, 2006). Therefore over-fitting in NLCS is not a concern.

Hence, the results in this section support the hypothesis that the population size and the training time have some impact on the predictive accuracy.

4.6 Chapter Summary

In this chapter, the author argued that the predictive accuracy is not the only factor to judge a classifier system. The compactness and expressiveness of the system are other important issues in traditional machine learning. The author proposed a neural representation in LCS to balance these factors.

The author proposes a novel way to incorporate neural networks into UCS. The approach offers a good compromise between compactness, expressiveness, and accuracy. By using a simple artificial neural network as the classifier's action, the system can obtain a more compact population size, better generalization, and the same or better accuracy, while maintaining a reasonable level of expressiveness.

The first research question of this chapter is whether the neural representation is beneficial. The author considered this by testing UCS and NLCS on fourteen data sets. The t-test results show that NLCS performs equivalently to UCS on five data sets, significantly better on six data sets, but significantly worse on three data sets.

The population size also plays an important role in comparing the performance of UCS and also NLCS. In all tested data sets, UCS requires at least a few thousand

rules to capture the whole search space. NLCS, in general, requires only 25 rules to perform as well as UCS. The population size of 25 classifiers would be much faster in terms of execution time in comparison to thousands of classifiers. Thus, the neural-based representation is indeed beneficial to LCS.

The author also investigated three common approaches for decision making in the match set: majority voting, simple averaging, and winner-takes-all. Simple averaging is the best choice in most data sets.

The second research question was whether negative correlation learning (NCL) will improve the performance of NLCS. The author tested NLCS with different values for the strength parameter, γ. The results show that NCL improves the predictive performance of NLCS on several data sets when $\gamma < 0.6$. The author concluded that $\gamma = 0.5$ is the best value to use in NLCS in order to achieve good performance in most problems.

Three data sets (tao, vowel, and segment) were investigated further. The author found that the population size and the training time have an impact on these three data sets, especially on the vowel data set.

Overall, the neural-based representation is able to achieve equivalent or even better performance than the traditional representation. Also, the new representation in general requires a significantly smaller population size than the traditional one. The incorporation of negative correlation learning in the training improves the predictive accuracy of the system. The author also found a consistent region of the strength parameter, with which NLCS performs well.

Though answering two research questions of this chapter, the second research sub-question in the first chapter is also solved. It is clearly shown that the neural representation is indeed beneficial for evolutionary learning classifier systems in terms of population size and predictive accuracy.

Chapter 5

Adaptive Neural-based Learning Classifier Systems

5.1 Overview

The previous chapter proposed a neural representation, to compact the population of UCS. The proposed system, called NLCS, replaces the action of the rule-based classifier by a simple neural network. Experiments reveal that a much smaller population is required to reach an equivalent accuracy in comparison to UCS.

The main disadvantage of NLCS is that the population size remains fixed and unchanged after reaching the maximal threshold. The neural networks play an important role in learning after this point. This chapter will show that the condition part of NLCS in some cases can not decompose the search space. Therefore, one of the main benefits of LCSs in data mining, the comprehensive rule-based representation, may not be realized.

In this chapter, the author will propose an enhanced version of NLCS, called Adaptive NCS (ANCS), which allows the population to vary. Several visualization tools are proposed in this chapter in order to have a better understanding of the underlying system.

In order to answer the third research sub-question, stated in the first chapter, this chapter needs to solve two smaller questions. The first one is:

- Is the proposed adaptive framework beneficial in NLCS? The author hypothesizes that ANCS is able to achieve the equivalent predictive accuracy as NLCS and UCS. Moreover, by allowing the population to grow and shrink, ANCS is able to form better patterns than NLCS.

The investigation of ANCS reveals some sensitivity to the setting of some parameters. This motivates the author to propose an ensemble framework, each of which has a different parameter setup. The author hypothesizes that the ensemble framework can overcome the problem of sensitivity to parameters of LCS in general when dealing with new problems.

The second question is:

- Does the ensemble framework help to reduce the number of parameters?

The chapter is structured as follows. Section 5.2 describes the proposed framework of ANCS. The experimental setup is explained in Section 5.3. Section 5.4 describes visualization methods that help us to obtain a better insight into the knowledge learned by ANCS. Sections 5.5 and 5.6 investigate each of the research questions. Section 5.7 concludes the chapter.

5.2 Adaptive Neural-based Learning Classifier Systems (ANCS)

ANCS extends NLCS, by removing a fixed population size. ANCS allows the population to vary through the cooperation of five key operators: covering, GA, subsumption, deletion, and merge. The first three functions operate similarly to those of UCS. We will use both subsumption in GA and subsumption in the correct set. The deletion and merge operators are explained below.

The subsumption and deletion functions are responsible for shrinking the population. The covering, GA, and merge functions, on the other hand, introduce more classifiers into the population. A balance between these functions is necessary to keep the population stable. In UCS, the maximal population size is defined to control the balance. In this chapter, the author will not use a fixed upper bound of the population size; instead a technique is proposed to control the frequency of activation of those functions.

5.2.1 Deletion Function

The first improvement of ANCS over NLCS is the deletion technique, which eliminates inaccurate classifiers from the population.

Normally, inaccurate classifiers arise due to their hyper-rectangle conditions, which form a larger local area than a simple neural network can learn. In this case, the neural network will be under-fitted as it does not have enough power to handle the area. These classifiers can be either removed, or could employ more complicated neural networks. The first option seems better, since a full neural network would dramatically reduce the expressiveness of the model, which is quite important in data mining.

The deletion method in ANCS operates similarly to deletion in UCS/XCS as described in (Kovacs, 1999). Deletion will occur under two conditions: the population size (P_{lb}) exceeds a threshold, and a place is needed for offspring in GA. The deletion decisions are carried out within the whole population. All classifiers in the population compete against each other for survival. Those classifiers with high fitness and belonging to small niches are more likely to remain in the population. Classifiers that have not yet been in the population for 5000 training instances are not considered for deletion, to give them time for the neural networks to learn.

As well, ANCS employs a self–deletion technique based on the accuracy of local neural networks. Each classifier constantly estimates the difference between its lowest and highest accuracy of any two classes. A parameter called θ_{Sdel} is used as a threshold. If the difference in accuracy exceeds the threshold, and the classifier

has been in the population for at least 5000 training instances, the classifier is deleted.

This technique will eliminate classifiers that are accurate in one class but inaccurate in the others. This might be due to an unbalanced data set, which makes some classifiers favour the majority class. Although the overall accuracy of these classifiers might be high, they are unable to handle minority classes within the area. Deleting them will make room in the population for more specific classifiers that can handle minority classes.

5.2.2 Merge Function

The merge function is introduced in ANCS to improve the generalization of the population. The subsumption and merge functions work together to compact the population. Both functions take place in the correct set.

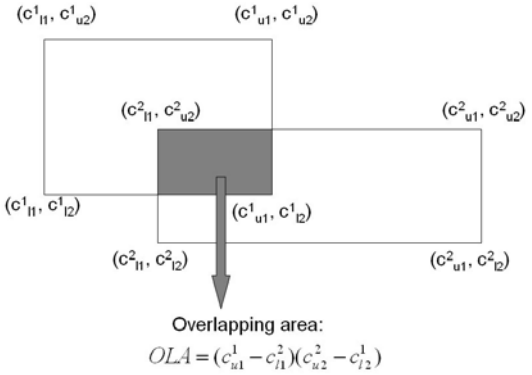

Figure 5.1: Overlapping area of two classifiers for merging

The first task of the merge function is to find a set of experienced classifiers in the correct set *[C]*. A classifier is considered "experienced" if it has survived in the population for longer than some threshold. In this chapter, the author uses 5000 learning steps as the threshold.

The amount of overlap is then calculated between each pair of experienced classifiers in the set. Note that there must be some overlap between any two classifiers in the correct set, because they all match at least this input instance. The overlapping area represents the commonality of two classifiers. We hypothesize that

the bigger area they have in common, the closer their decisions would be when confronted with the same problem.

Figure 5.1 illustrates the way to calculate the overlapping area. A pair with the largest overlap is chosen for merging. A new classifier is created by combining the condition of these classifiers. Its action will be inherited from the classifier with higher accuracy.

The classifier is then inserted in the population to compete with the parents. It will survive in the population if it is accurate. If so, subsequently it will subsume its parents. Otherwise it will end up being deleted.

5.2.3 Fixed Parameters

As described above, P_{lb} is used to activate the deletion function when GA needs a place for its offspring. It is important to clarify that the population can grow beyond this limit, through covering and merging. P_{lb} determines how big the population can get before GA starts to trigger deletion. The value of P_{lb} in theory should not affect the performance of the system, but might delay or speed up the convergence.

An important parameter in ANCS is the deletion threshold θ_{Sdel}. It is used in the self-deletion function, to decide when to delete a classifier based on its network's accuracy.

The third important parameter is the covering range. It is used in covering, to decide how general or specific a new classifier should be. In theory this parameter should not affect the performance of the system, since the covering function mainly happens at the beginning. However, it affects the convergence of the system, depending on the nature of the training problem. Some problems might suit general classifiers, so it would be better to generate general classifiers from the start rather than having to generalize from specific classifiers. Others might favour specific classifiers, so starting with a population of specific classifiers would save a lot of training time.

The effect of each of these parameters on the accuracy of the system is investigated in Section 5.5.

5.2.4 Adaptive Parameters

The two most important components of ANCS in the exploration phase are the GA and merging.

GA has a role as a search function that introduces new classifiers into the population. GA needs to be fully active when the system lacks knowledge (in the learning stage), to give the capacity to explore the full search space.

However, when the system is stable, an active GA might damage the balance in the system due to the deletion/GA cycles. In this case we want to reduce the activity of GA.

The merge function, on the other hand, needs to wait while the system is still learning. It becomes involved once the system has reached a stable condition. Its main objective is to improve the generalization based on the current population. The stable condition indicates that the knowledge in the system is reliable, so it is time to start thinking of improving the generalization.

This analysis motivated the introduction of two parameters to control the frequency of GA and merging: the probability of GA, called P_{GA}, and the probability of merge, called P_M. P_{GA} and P_M are not fixed. Their values are changed over time, adapting as the performance of the system indicates the trend of learning.

In order to identify the trend of learning, the performance of ANCS (in terms of the predictive accuracy on the training data) is measured over time. The trend in performance indicates whether the system is learning or stable.

The training data is divided into windows, and the predictive accuracy within each window is calculated. The values of the current window are compared against the ones of the previous window. The statistical t-test is used to test the significance of changes in the predictive accuracy of the system. There are three possibilities:

- Improved performance: the accuracy of the current window significantly im-

proves over the previous window. This implies that the system is still in the learning phase. GA needs to continue its job to further improve the accuracy.

- Decreased performance: the predictive accuracy is reduced in comparison to the previous window. This might be caused by several things: a new area in the search space is seen; there are concept drifts; or good classifiers were mistakenly deleted. In any case, GA needs to continue working in order to explore the whole search space.

- Stable performance: the predictive accuracy is unchanged in comparison to the previous window. This indicates that the system is stable, and the predictive accuracy might not be changed by GA. In this case, GA needs to reduce working and merge needs to be activated to compress the population.

The relationship between the learning performance and these functions is described in Table 5.1.

Table 5.1: The relationship between the system performance and GA/Merge functions

Learning Performance	GA	Merge
IMPROVED	ACTIVE	INACTIVE
DECREASED	ACTIVE	INACTIVE
STABLE	INACTIVE	ACTIVE

repeat
  if *the end of the current window* **then**
    Compare the accuracy of the current window with the previous window.
    if *significant change* **then**
      P_{GA} = p_max_value;
      P_M = p_min_value;
    end
    if *stable* **then**
      P_{GA} = p_min_value
      P_M = p_max_value
    end
  end
until *stop condition is met* ;

Algorithm 4: Adaptive parameters in ANCS

P_{GA}, P_M are changed according to the performance of the system, as shown in Algorithm 4. We allow two levels of probabilities. The minimum value is greater than zero, allowing both GA and merge functions to work all the time with low probabilities. This means that the GA function always has a chance to help the system escape from local optima.

5.2.5 Description of ANCS

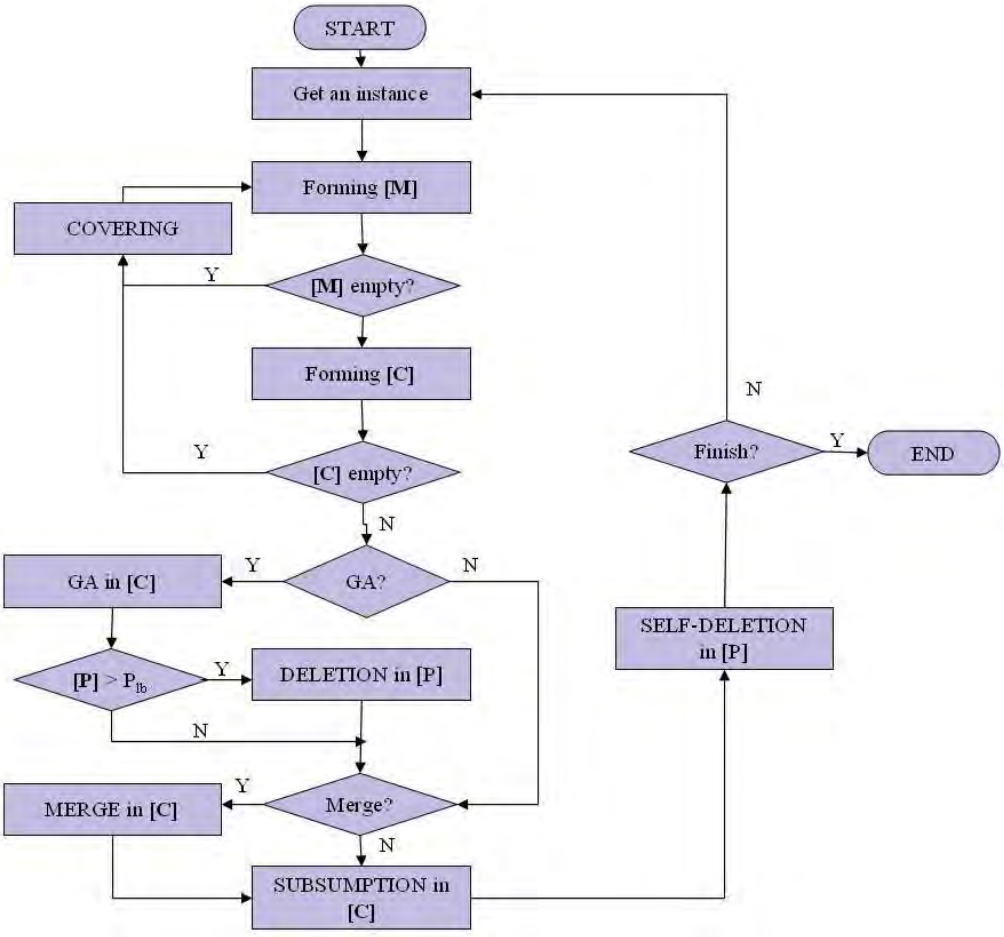

Figure 5.2: The flowchart of ANCS

Figure 5.2 shows the flowchart of the training phase in ANCS. Assuming no available prior knowledge at the beginning, ANCS will start from an empty population. For each training instance, the system will search through its population in order to form a match set *[M]* as in NLCS and UCS. If the match set is empty, normally happening early in learning, the covering function is activated to introduce

some knowledge into the system.

ANCS works as a supervised learner, in which each training/testing instance is associated with a desired outcome. The correct set *[C]* is immediately formed from those classifiers in *[M]* making a correct prediction. As in UCS, if the correct set is empty the covering function is used to hasten convergence.

GA is carried out in the correct set under two conditions: the average time since the last activated GA of all classifiers in the set exceeds a threshold, and a random number is less than the probability of GA P_{GA}.

The decision to delete a classifier is made immediately after GA inserts a new classifier to the population. If the current number of macro classifiers exceeds P_{lb}, deletion is enforced in order to put pressure towards a more accurate and compact population.

The merge is carried out under similar conditions of GA: if the average time since the last merge is higher than a threshold, and a random number is less than P_M. Subsumption will then remove redundant classifiers from the population.

The self-deletion function then checks each classifier in the population in terms of its internal accuracy and experience. Unappropriate classifiers are eliminated in this process to make rooms for others.

Finally, the learning performance is recalculated at the end of each training window defined by the user. This function is responsible for updating P_{GA}, P_M with regards to the current learning trend in the system.

5.3 Experimental Setup

Unless stated differently, UCS is setup with the same parameter values used by Bernadó-Mansilla and Garrell-Guiu (2003) as follows: $v = 5, \theta_{GA} = 50, \chi = 1, \mu = 0.04, \theta_{del} = 50, \theta_{sub} = 50, m_0 = 0.1, s_0 = 0.6, N = 6400$. Two point crossover and roulette wheel selection are used.

For NLCS, each MLP has one hidden layer and one hidden node. The learning

rate of MLPs is $\beta = 0.1$. Other parameters are: $v = 5, \theta_{GA} = 50, \chi = 1, \mu = 0.04, m_0 = 0.1, s_0 = 0.1$, covering_window = 50, $\gamma = 0.5$, population size $N = 25$.

ANCS is setup using similar parameters as NLCS, except the following parameters: v=10, $P_{lb} = 100$, $\theta_{Sdel} = 0.3$, covering range c=1, p_max_value=1, p_min_value=0.01. Two points crossover and roulette wheel selection are used.

Each learner performs three stratified 10-fold cross validation runs in each data set (that is 30 runs). Each run uses different random seeds which are consistent in all experiments. The results reported in this chapter are averaged over those 30 runs. The term *iteration* is used to refer to a single pass through the training set. The statistical data is collected after 500 iterations in each experiment.

5.4 The Learning Knowledge of ANCS

This section aims to give more insight into the knowledge learned by ANCS, and therefore to help us obtain a better understanding of the learning patterns.

A classifier consists of a rule and a set of parameters that qualify the rule. The knowledge learned by a classifier system is the final set of rules. This section will visualize the final population of ANCS, looking at both the condition and action parts of the rule. The first part reveals the ability to decompose the search space. The second one, on the other hand, reveals the learning ability of the system within its local search area.

5.4.1 Decomposition of the search space

The first visualization is taken on the tao problem, which has two features and can easily be pictured in two dimensions. The search space of this problem can be divided into three regions: left, middle, and right. The left and right areas contain only one class. The middle area contains several non-linear boundaries, that make this problem harder to learn by any machine learning.

Figure 5.3 presents the visualization of the final population of ANCS and NLCS

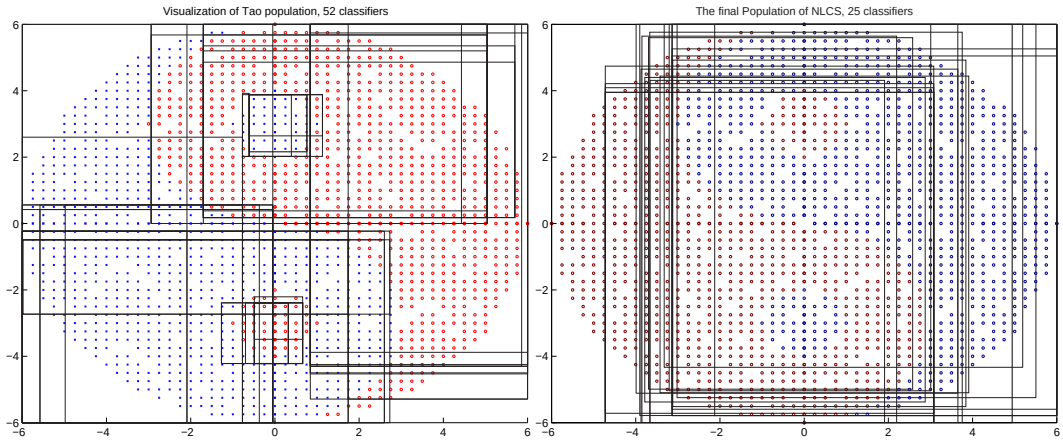

Figure 5.3: Visualization of the final population of ANCS (left) and NLCS (right) on the tao problem. Both systems are able to achieve similar accuracy of around 89% using their best parameter setting.

on the tao problem. The best parameter settings were used for NLCS, which achieved accuracy of around 89%. A small covering range was used for ANCS, and other parameters were set so that it achieved the same accuracy.

Most classifiers obtained by NLCS are general. The conditions do not seem to help the system to decompose the search space well in the middle area. The high performance of NLCS would be mainly due to the local neural networks. As there is no deletion in NLCS, the networks can accumulate a lot of knowledge by the end. Also, the negative correlation learning in NLCS becomes very important to push those networks in the middle area far from each other so that each can specialize on different areas.

Observing ANCS, we can see that the middle area seems to gather more classifiers than the other areas. The left and right hand sides of the search space contain a single class, and therefore do not need many classifiers. Many specific classifiers are created in the middle area by the covering and GA functions. This visualization illustrates that ANCS with its adaptive framework is better than NLCS in terms of mining patterns.

Many datasets in the real world have more than two features and therefore cannot be visualized as obviously as the tao problem. In order to visualize the population of other data sets the author decided to employ multi-dimensional scaling

methods, which are widely used in the literature to map data from high dimensions into lower dimensions. Some researchers have used this approach to visualize rule-based knowledge (Berthold and Holve, 2000), (Berthold and Hall, 2003), (Tsumoto and Hirano, 2003). To the best of the author's knowledge, it has not been used previously to visualize a population of a LCS.

The first step is to filter the population, keeping a portion of classifiers with high accuracy and experience. Each classifier will be sampled with a number of important points. Those points will be transformed into two dimensions, and later used to reconstruct the classifier. In this thesis, the author chooses the corner and center points of a classifier for transformation.

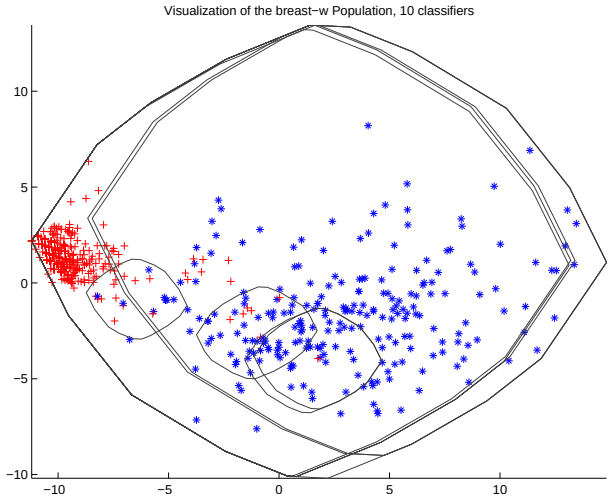

Figure 5.4: Visualization of the final population of ANCS on the breast-w problem

The final population of the breast-w problem can be viewed in Figure 5.4. As we can see, this is an easy problem as the negative and positive data points are quite distinguishable from each other. Therefore a simple neural network can easily solve the problem. As a result, ANCS tends to maintain several very general classifiers. Several smaller classifiers also exist in the intersection area of the two classes, so that more accurate predictions can be made within that area.

Figure 5.5 presents the final population of the iris problem. This problem has three classes, located in three different areas. One class is totally separated from the other two classes. It can easily be seen that ANCS divides the population into three areas, so that each area can be handled by one classifier.

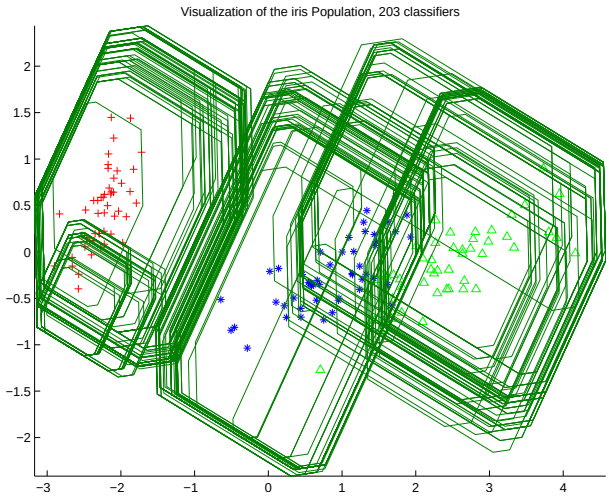

Figure 5.5: Visualization of the final population of ANCS on the iris problem

These two visualizations reflect that ANCS is able to decompose the problem nicely using the condition part. This technique also provides more insight into the learning problem. Thus it is a useful technique to understand the problem as well as the learned knowledge.

5.4.2 Neural Network boundaries

In order to understand how the neural networks work within a classifier, the predictions of neural networks are visualized and discussed in this section.

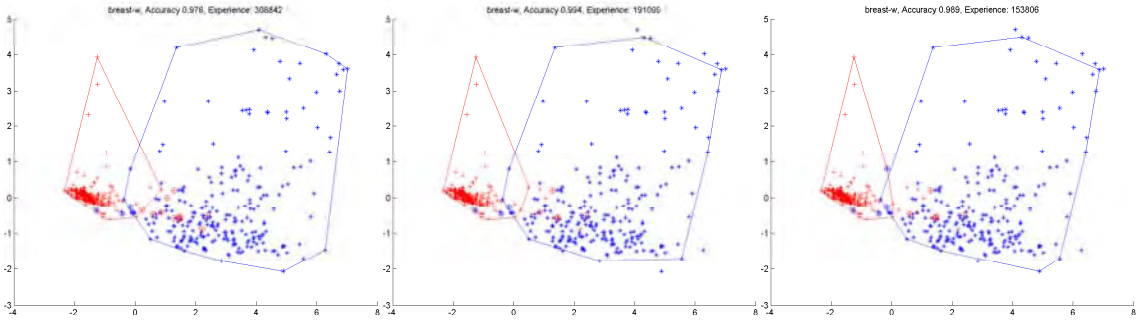

Figure 5.6: Visualization of three most common patterns in the last population on the breast cancer problem

In this experiment, the predictions of each classifier in the final population are determined, and used to judge its ability to learn. This process consists of two steps. The first step is to determine which data instances in the training set

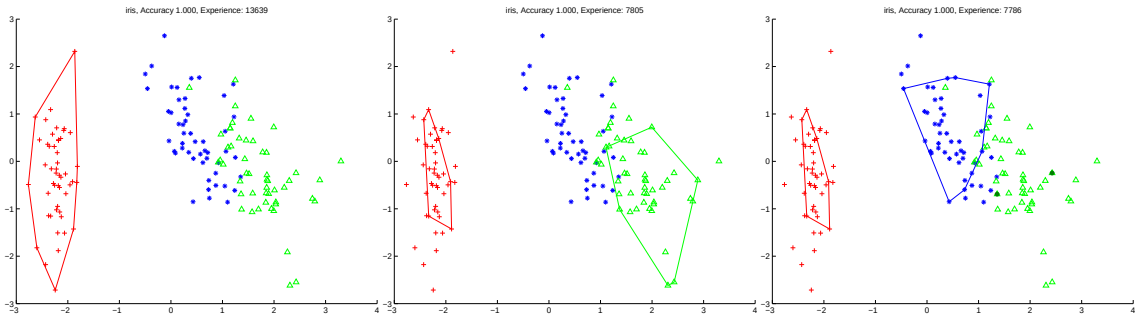

Figure 5.7: Visualization of three most common patterns in the last population on the iris problem

belong to which classifiers. This is done by feeding each data instance through each classifier so that the classifier's condition can decide whether or not it can deal with the instance. The second step requires each classifier to make a prediction on each data instance that it does match. The decision will be made by its neural network based on its accumulated knowledge.

As in the previous section, the data sets of each classifier are re-scaled into two dimensions using the multi-dimensional scaling function. The decision of each classifier on the same class is bounded using the convex hull algorithm as described in (Nguyen, 2006).

Figures 5.6 and 5.7 show the decision boundaries of the neural networks of the three most experienced and accurate classifiers in the final population, on the breast cancer and iris problems respectively.

As we can see the boundaries of the breast cancer problem in the three classifiers are quite similar. A small difference between neural networks is in the intersection area of two classes. Due to different bias caused by different training instances, each network will propose different predictions in this overlapping area.

On the iris problem, neural networks can distinguish the separate class well. However, the overlapping area of the other two classes is not separable and therefore is very difficult for a simple neural network to handle. As in the breast cancer problem, different bias in the training data will derive networks with slightly different decisions in this area.

In general, we can see that a complex learning boundary would challenge simple

neural networks, as expected. Neural networks are able to provide better predictions within a simpler boundary. To get full advantage of the neural networks it helps to decompose the search space well with the condition parts of the classifiers.

This simple visualization technique is quite effective for helping us to understand the learning of neural networks, adding more insight into the analysis besides the results reported in tables or figures. The results also showed that ANCS is able to mine better pattern than NLCS.

5.5 The Performance of ANCS

This section aims to provide a better understanding of ANCS.

First, the effect of each of the three fixed parameters is studied. Since it is very difficult to estimate their effects all together, experiments in this section are presented based on an assumption that these parameters can be treated independently. In each preliminary experiments, the value of one parameter is varied in order to find reasonable initial values. The following experiments then identify the best value for each parameter. In each experiment, one parameter is varied while keeping the others unchanged.

The experiments are carried out on a subset of seven of the fourteen problems. These problems have a small number of features, but otherwise have all the characteristics of the full set of problems: different representations (real, integer, and category), different numbers of classes (2 and 3 classes), popular data sets, non-linear boundary, etc.

The overall performance of ANCS (with the fixed parameters set to what seem good values) is then compared with NLCS on all of the test problems.

5.5.1 The effect of the population threshold P_{lb}

ANCS allows the population to grow freely over time. P_{lb} is a threshold which affects whether deletion takes place after GA inserts a classifier into the population.

Table 5.2: The mean and standard deviation of accuracy of ANCS with different values of P_{lb}. The highest accuracy in the row is bolded.

Problem	25	100	200	500
balance-s	0.884 ± 0.03	**0.895 ± 0.02**	0.891 ± 0.03	**0.895 ± 0.02**
breast-w	0.968 ± 0.02	**0.970 ± 0.02**	**0.970 ± 0.01**	0.969 ± 0.01
bupa	0.663 ± 0.05	0.663 ± 0.07	**0.700 ± 0.06**	0.683 ± 0.06
credit-a	0.856 ± 0.03	0.859 ± 0.03	**0.861 ± 0.04**	0.857 ± 0.04
diabetes	0.754 ± 0.04	0.746 ± 0.04	**0.757 ± 0.05**	0.754 ± 0.04
iris	0.951 ± 0.06	0.951 ± 0.05	**0.967 ± 0.05**	**0.967 ± 0.05**
tao	0.854 ± 0.02	0.866 ± 0.02	0.879 ± 0.02	**0.890 ± 0.02**

It plays an important role to create pressure within the system towards a population of accurate and maximally general classifiers. An interaction between deletion and GA in LCS was first identified by Wilson in his generalization hypothesis (Wilson, 1995). Butz and his colleagues later provided equations to support it theoretically in (Butz *et al.*, 2003b). Wilson's generalization hypothesis suggested that the set pressure is obtained by taking reproduction in niches and deletion in the population. General classifiers normally have higher chances to appear in the correct set. As a result, they participate more often in the reproduction of GA. This favours generality in the population. Deletion in the population, on the other hand, could happen to any classifier within the population with higher probability toward the ones with low fitness. The deletion allows the population to keep its most accurate ones.

Table 5.2 presents the mean and the standard deviation of the predictive accuracy of ANCS with four different values for P_{lb}. The best value for each problem is highlighted. Table 5.3 displays the mean and standard deviation of the final population size with each value of P_{lb}.

In terms of accuracy, $P_{lb} = 200$ seems to be the optimal threshold as it produces the highest accuracy on five data sets, followed by $P_{lb} = 500$ and $P_{lb} = 100$ with the highest accuracy on three and one data sets respectively. The differences are mainly small. This confirms the author's expectation that P_{lb} should not affect accuracy in the long run. However, we can see some different behaviours on different problems.

In principle, increasing the population size allows the system to accumulate

Table 5.3: Final population sizes of ANCS with different values of P_{lb}. The smallest population size in the row is bolded.

Problem	25	100	200	500
balance-scale	**27 ± 9**	65 ± 9	169 ± 7	428 ± 12
breast-w	**4 ± 4**	37 ± 8	73 ± 13	216 ± 25
bupa	**284 ± 86**	327 ± 124	744 ± 146	1165 ± 237
credit-a	365 ± 81	**152 ± 43**	427 ± 114	609 ± 101
diabetes	**94 ± 35**	146 ± 71	279 ± 152	568 ± 111
iris	122 ± 53	**59 ± 22**	196 ± 41	384 ± 42
tao	80 ± 23	**53 ± 12**	159 ± 11	291 ± 24

more knowledge. Therefore the system is able to obtain better accuracy. However, this is normally not the case in LCS. Too large a population reduces the deletion pressure required to lead to a population of accurate and maximally general classifiers. That is clearly shown with $P_{lb} = 500$: final population sizes are always larger, but accuracy decreases compared to $P_{lb} = 200$ on several problems. The biggest drop can be observed in the bupa problem, where accuracy drops from 70% to 68.3% while the population size increases greatly. It seems that the deletion threshold of $P_{lb} = 500$ is not strong enough to eliminate inaccurate classifiers from the population. Inaccurate classifiers staying in the population can lead to inaccurate productions in GA. It delays the convergence of the system.

P_{lb} only triggers deletion after GA, not after covering or merging. Merging is the main function that increases the population, while subsumption decreases the population. In Table 5.3 we can see several problems where the final population size is much larger than P_{lb}. This shows that merging is more frequent than subsumption, indicating the problem favours more specific classifiers. In contrast, some problems favour more general classifiers, and the final population size can be smaller than P_{lb}.

It can be best not to set P_{lb} too small. The smallest final population sizes were obtained with $P_{lb} = 25$ on four problems, but with $P_{lb} = 100$ on three problems. For these three problems, it is likely that when P_{lb} is small, too many useful classifiers are being deleted after GA; the merge function cannot find classifiers with enough overlap, and generalization is hard.

Table 5.4: The mean and standard deviation of accuracy of ANCS with different covering ranges. The highest accuracy in the row is bolded.

Problem	0.10	0.20	0.30	0.40	0.50
balance-scale	0.888 ± 0.03	0.885 ± 0.02	0.894 ± 0.02	0.894 ± 0.02	0.892 ± 0.02
breast-w	0.968 ± 0.01	0.971 ± 0.01	0.969 ± 0.02	**0.974 ± 0.01**	0.968 ± 0.02
bupa	0.624 ± 0.08	0.635 ± 0.06	0.627 ± 0.06	0.617 ± 0.07	0.640 ± 0.06
credit-a	0.857 ± 0.04	0.863 ± 0.03	0.860 ± 0.03	0.859 ± 0.03	0.862 ± 0.02
diabetes	0.744 ± 0.04	0.742 ± 0.04	0.741 ± 0.03	0.743 ± 0.03	0.741 ± 0.04
iris	0.964 ± 0.03	0.953 ± 0.05	0.951 ± 0.04	0.953 ± 0.04	0.951 ± 0.05
tao	0.880 ± 0.03	0.868 ± 0.03	**0.880 ± 0.02**	0.867 ± 0.02	0.873 ± 0.02

Problem	0.60	0.70	0.80	0.90	1.00
balance-scale	0.894 ± 0.02	0.894 ± 0.02	0.893 ± 0.02	**0.899 ± 0.02**	0.895 ± 0.02
breast-w	0.970 ± 0.01	0.972 ± 0.01	**0.974 ± 0.01**	0.969 ± 0.01	0.970 ± 0.01
bupa	0.640 ± 0.07	0.648 ± 0.07	0.648 ± 0.07	0.636 ± 0.06	**0.663 ± 0.07**
credit-a	0.860 ± 0.03	0.865 ± 0.03	0.861 ± 0.03	0.865 ± 0.03	**0.867 ± 0.03**
diabetes	0.745 ± 0.03	0.749 ± 0.03	**0.751 ± 0.03**	0.743 ± 0.04	0.746 ± 0.04
iris	0.947 ± 0.06	0.960 ± 0.04	0.960 ± 0.04	**0.969 ± 0.04**	0.951 ± 0.05
tao	0.870 ± 0.03	0.869 ± 0.02	0.872 ± 0.02	0.868 ± 0.02	0.866 ± 0.02

It can be observed that $P_{lb} = 100$ gives slightly better accuracy than $P_{lb} = 25$ on most problems, and achieves the best accuracy overall on two problems. $P_{lb} = 100$ seems a better choice than $P_{lb} = 25$.

It appears that $P_{lb} = 200$ gives better accuracy while $P_{lb} = 100$ produces smaller populations. The trade-off between accuracy and compactness is quite common in machine learning. It would be difficult to say definitely which P_{lb} is the best in this experiment. Except for the bupa problem, all other problems do not show much difference in accuracy between $P_{lb} = 100$ and higher thresholds. However, the final population size is usually less than half. Thus from this point, $P_{lb} = 100$ is used.

5.5.2 The Effect of the Covering Range

The covering function happens in ANCS when the correct set is empty. When a new classifier is created by this function, the covering range affects the size of the boundaries of the classifier around the instance. Random values within the covering range are chosen for the lower and upper bounds of the condition. If the covering range is large, more general classifiers are created. If the range is small, more specific classifiers are created. Either could be advantageous depending on the application domain.

Table 5.4 presents the mean and the standard deviation of the predictive accuracy of ANCS on the seven problems, with different values for the covering range. The best value for each problem is highlighted. The variation in performance between different covering ranges is generally small. This confirms the author's expectation that the covering range is likely to affect the rate of convergence rather than accuracy.

Three different patterns can be seen in Table 5.4. In the breast cancer, credit-a, diabetes, and iris problems, the covering range has almost no effect on accuracy. In the balance scale and bupa problems, accuracy is better with large covering ranges of 0.9 or 1. In the tao problem, accuracy is best with small covering ranges of 0.1 or 0.3.

Each classifier in ANCS tries to capture a local area which represents a simple part of the target problem. A neural network is responsible for providing further discrimination within that area. Neural networks can solve complex problems, but they normally require lots of hidden layers and hidden units. We used very simple neural networks, to preserve expressiveness, which limits their effectiveness to simple problems.

The non-linear boundaries in tao are clearly not a simple problem. If ANCS creates general classifiers, simple networks cannot provide correct predictions within most of them so they are inaccurate. They will just have to be deleted anyway, so it is better not to create general classifiers in the first place. Thus small covering ranges are best for the tao problem.

Problems in which the covering range makes little difference do not suffer from either specific or general sites. The evolutionary pressure together with deletion pressure in ANCS cooperate nicely to eliminate the effect of the covering range.

The responsibility to bring generality in LCSs normally belongs to GA. In ANCS, merging also takes part of this responsibility. If the population starts from general classifiers this duty can be reduced, but a heavier load is placed on deletion to filter out inaccurate classifiers from the population.

Bupa is the hardest problem. Even though its final population size is larger

than for any other problem, its best accuracy is only 66%. If the system starts with specific rules, a significant time is required until the whole search area is covered. As a result, a low accuracy is obtained until then.

Overall, a high covering range seems to be better in many problems. From this point in the chapter the author will choose a value of 1 unless stated differently.

5.5.3 The Effect of the Deletion Threshold

The last parameter to be investigated is the deletion threshold θ_{Sdel}. This is used in the self-deletion function, which deletes classifiers that have an imbalance in the accuracy of their predictions for different classes.

A small threshold increases the pressure toward accurate classifiers. For example, suppose a problem has two classes M and N. A classifier with 100% accuracy on class M and 89% accuracy on class N will be deleted if the threshold is 0.1, even though its overall accuracy is more than 90%. A small threshold will only accept highly accurate classifiers, which might not be good in many cases.

A high threshold, on the other hand, make this form of deletion rare. Classifiers with high differences in their predictive accuracy are likely to be deleted beforehand by the normal deletion function due to low accuracy in general.

Table 5.5 presents the mean and the standard deviation of ANCS with different deletion thresholds ranging from 0.1 to 0.7. The best accuracy in each problem is bolded. From the result, we can see three regions of the threshold: small ([0.1,0.3)), moderate ([0.3,0.6)), and large ([0.6,0.7]).

Observing each region, we can see that the best accuracy is generally obtained within moderate thresholds. Six out of seven problems achieve their highest accuracy when the deletion threshold is in the range [0.3,0.6). Bupa is the only problem for which accuracy is best outside that range, but the difference between 71.1% with a threshold of 0.5 and 71.4% with a threshold of 0.6 is very small.

From this experiment it seems that this self-deletion function is indeed useful, since it does affect accuracy. Moderate thresholds seem to support better accuracy

Table 5.5: The mean and the standard deviation of accuracy of ANCS with different deletion thresholds. The highest accuracy in the row is bolded.

Problem	0.10	0.20	0.30	0.40
balance-scale	0.883 ± 0.03	0.883 ± 0.03	**0.892 ± 0.02**	0.883 ± 0.03
breast-w	**0.971 ± 0.02**	0.970 ± 0.01	0.968 ± 0.02	**0.971 ± 0.01**
bupa	0.707 ± 0.06	0.693 ± 0.06	0.640 ± 0.06	0.707 ± 0.06
credit-a	0.856 ± 0.03	0.857 ± 0.04	**0.862 ± 0.02**	0.858 ± 0.04
diabetes	0.756 ± 0.04	0.753 ± 0.04	0.741 ± 0.04	0.755 ± 0.04
iris	0.960 ± 0.05	**0.964 ± 0.05**	0.951 ± 0.05	**0.964 ± 0.05**
tao	0.866 ± 0.02	0.864 ± 0.02	**0.873 ± 0.02**	0.865 ± 0.02

Problem	0.50	0.60	0.70
balance-scale	0.883 ± 0.03	0.883 ± 0.03	0.883 ± 0.03
breast-w	0.971 ± 0.01	0.969 ± 0.01	0.969 ± 0.01
bupa	0.711 ± 0.06	**0.714 ± 0.05**	0.705 ± 0.05
credit-a	0.859 ± 0.04	0.858 ± 0.03	0.856 ± 0.04
diabetes	**0.761 ± 0.05**	0.761 ± 0.04	0.759 ± 0.04
iris	0.960 ± 0.05	0.960 ± 0.05	0.958 ± 0.05
tao	0.867 ± 0.02	0.866 ± 0.02	0.865 ± 0.02

than high thresholds. When the threshold is 0.3, three problems have obtained the best accuracy. Fewer problems achieve the best accuracy with higher thresholds (two when the threshold is 0.4, and only one when the threshold is 0.5), so 0.3 seems the best value to use. From this point, the author will test the system with a threshold of 0.3.

5.5.4 Comparison with NLCS

Table 5.6 shows the mean accuracy and standard deviation of ANCS on each of the fourteen problems. Based on the results from the previous sections, the parameter values used were $P_{lb} = 100$, covering range c=1, $\theta_{Sdel} = 0.3$.

Table 5.6 also presents the results for NLCS (with a population size N=25), and compares the two systems. The statistical t-test is used to check for statistically significant differences in accuracy, at a significance level of 0.05. The results show statistically significant differences in accuracy in six problems. ANCS is better on two problems, NLCS is better on four problems, and there is no significant difference on the other eight problems.

Table 5.6: The mean and the standard deviation of accuracy of NLCS and ANCS. ■(□) symbols indicate that ANCS is better(worse) than NLCS at a significance level of 0.05.

Problem	NLCS	ANCS
balance-scale	0.903 ± 0.014	0.895 ± 0.024
breast-w	0.966 ± 0.013	0.970 ± 0.015
bupa	0.729 ± 0.064	0.663 ± 0.074 □
credit-a	0.859 ± 0.037	0.859 ± 0.032
diabetes	0.768 ± 0.042	0.746 ± 0.037 □
glass	0.604 ± 0.069	0.627 ± 0.100
heart-statlog	0.837 ± 0.080	0.832 ± 0.063
ionosphere	0.880 ± 0.065	0.900 ± 0.055
iris	0.958 ± 0.048	0.951 ± 0.049
lymph	0.841 ± 0.077	0.822 ± 0.081
segment	0.923 ± 0.019	0.943 ± 0.014 ■
sonar	0.831 ± 0.059	0.789 ± 0.071 □
tao	0.891 ± 0.018	0.866 ± 0.024 □
vowel	0.657 ± 0.049	0.912 ± 0.035 ■

The key difference between ANCS and NLCS is that ANCS employs the adaptive architecture to vary the population while NLCS has a fixed population size.

ANCS does better on the segment and vowel problems. These have many classes, and are hard for NLCS to learn with a small population size. The adaptive population size is important for these problems.

Sensitivity to parameters may explain why ANCS performs worse than NLCS on some problems. As noted above, a covering range of 1 does not suit the tao problem, and a relatively low value of $P_{lb} = 100$ is not good for the bupa problem. Similar reasons may explain why ANCS performs worse than NLCS on the diabetes and sonar problems.

Considering accuracy alone, it is unclear how to answer the first research question in this chapter: ANCS is not clearly better than NLCS. In terms of knowledge discovery, though, there is a difference. ANCS and NLCS differ in how their learning is shared between their conditions and their actions (neural networks). Once its maximum population size is reached, NLCS will not allow any insertion. From this point on, covering still plays some role but most of the learning is put on the

Table 5.7: Comparing ANCS's performance with other typical machine learning algorithms. The mean accuracy and standard deviation of Majority, C4.5, Naive Bayes and ANCS.

Problem	Majority	C4.5	Naive Bayes	ANCS
balance-scale	0.461 ± 0.002	0.779 ± 0.040	0.905 ± 0.018	0.895 ± 0.024
breast-w	0.655 ± 0.003	0.945 ± 0.026	0.960 ± 0.021	0.970 ± 0.015
bupa	0.580 ± 0.008	0.650 ± 0.086	0.548 ± 0.083	0.663 ± 0.074
credit-a	0.555 ± 0.004	0.854 ± 0.042	0.778 ± 0.042	0.859 ± 0.032
diabetes	0.651 ± 0.003	0.742 ± 0.046	0.756 ± 0.049	0.746 ± 0.037
glass	0.356 ± 0.014	0.674 ± 0.088	0.481 ± 0.082	0.627 ± 0.100
heart-statlog	0.556 ± 0.000	0.778 ± 0.080	0.839 ± 0.064	0.832 ± 0.063
ionosphere	0.641 ± 0.006	0.900 ± 0.053	0.825 ± 0.071	0.900 ± 0.055
iris	0.330 ± 0.000	0.948 ± 0.059	0.955 ± 0.051	0.951 ± 0.049
lymph	0.550 ± 0.029	0.777 ± 0.111	0.830 ± 0.086	0.822 ± 0.081
segment	0.143 ± 0.000	0.968 ± 0.011	0.801 ± 0.018	0.943 ± 0.014
sonar	0.534 ± 0.012	0.738 ± 0.085	0.678 ± 0.096	0.789 ± 0.071
tao	0.500 ± 0.000	0.956 ± 0.013	0.809 ± 0.029	0.866 ± 0.024
vowel	0.091 ± 0.000	0.801 ± 0.036	0.632 ± 0.049	0.912 ± 0.035

shoulders of the neural networks. The decomposition part of NLCS is relatively unimportant, and the neural networks within classifiers have the most important role in the learning of the system. ANCS, on the other hand, employs deletion and subsumption to ensure that the condition is also an important component. As seen in Section 5.4, the result is that ANCS shares its learning better between the conditions and the neural networks. This means that ANCS is better than NLCS in terms of knowledge discovery.

5.5.5 Comparison with Other Classifiers

Table 5.7 shows the performance of three typical classifiers in the literature including Majority, C4.5, and Naive Bayes on the same datasets. The results obtained from the WEKA system by Butz (2004).

Similar to NLCS, ANCS outperforms the majority algorithm in all datasets. ANCS is better than C4.5 in ten problems and the naive bayes in eight problems. Even though the significant test were not performed, the results indicate that ANCS is able to achieve similar accuracy as other classifiers in the literature.

5.6 An Ensemble Framework

The previous section presented the performance of ANCS when the fixed parameters were always set to the same fixed values that seemed best overall: $P_{lb} = 100$, covering range c=1, $\theta_{Sdel} = 0.3$.

We noted above that the accuracy of ANCS can be sensitive to the values of the fixed parameters. Table 5.4 showed three different patterns, suggesting that covering range can be divided into three different areas: small ([0.1–0.4)), moderate ([0.4–0.8)), and large ([0.8–1]). Similarly, we saw three ranges for the deletion threshold: small ([0.1,0.3)), moderate ([0.3,0.6)), and large ([0.6,0.7]). Each of these settings might suit different types of problem. In practice the characteristics of a data set might not be known in advance, so there is no way to know the best parameter values to use.

This motivates the author to propose an ensemble version of ANCS, called EANCS. Figure 5.8 depicts the framework of EANCS. Each individual in the ensemble will be set up with one of three different values for the covering range and one of three different values for the deletion threshold. We choose values close to the midpoint of each range: 0.2, 0.55, 0.9 for the covering range, and 0.15, 0.4, 0.65 for the deletion threshold. The proposed framework contains nine nodes, representing all nine possible combinations of values for the covering range and deletion threshold.

In the training phase, all individuals in EANCS are trained independently with the same training data. Different parameter settings might lead to different knowledge in the system, resulting in different prediction for testing instances.

In the testing phase, each data instance is evaluated by each individual in the ensemble. The final outcome is decided by a voting scheme at the gate level.

5.6.1 Individuals in EANCS

Table 5.8 shows the predictive accuracy of each individual in the ensemble, and the ensemble as a whole. The results for the individuals show variation in the

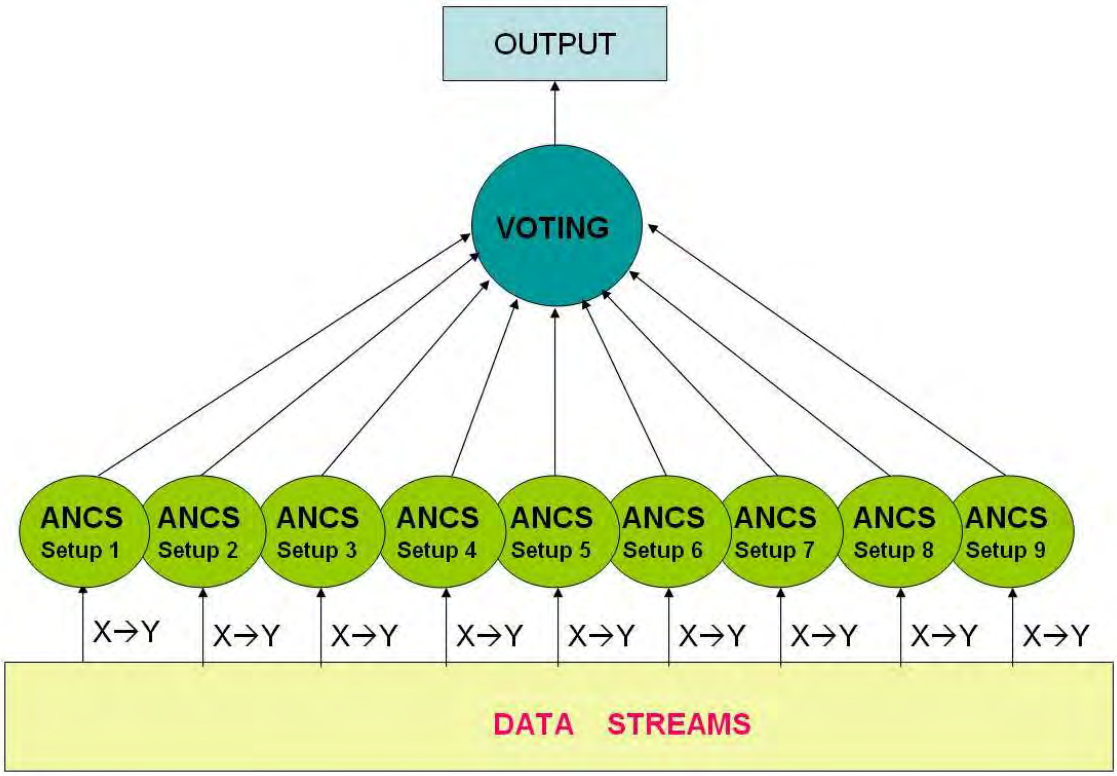

Figure 5.8: The framework of EANCS

predictive accuracy in all problems except the breast cancer. This again confirms that these parameters have an impact on the predictive accuracy of ANCS.

The accuracy of the ensemble is higher than that of almost every individual. This confirms that the ensemble framework is able to overcome the bias of different setups.

The covering range and deletion threshold parameters are handled internally within the ensemble. In effect the only fixed parameter the author has introduced is P_{lb}. The ensemble framework does reduce the number of parameters. This answers the second research question.

5.6.2 Comparison with Other Systems

Table 5.9 shows the mean accuracy and standard deviation of EANCS, NLCS, and UCS. The statistical t-test is used to compare NLCS and UCS against EANCS at a significant level of 0.05. The results for a single ANCS are also presented and

Table 5.8: The mean accuracy of each individual and the ensemble of EANCS.

Covering range	0.20	0.55	0.90	0.20	0.55	0.90	0.20	0.55	0.90	
Deletion threshold	0.15	0.15	0.15	0.40	0.40	0.40	0.65	0.65	0.65	Ensemble
balance-scale	0.886	0.895	0.898	0.890	0.898	0.902	0.893	0.899	0.894	0.899 ± 0.018
breast-w	0.963	0.963	0.964	0.963	0.964	0.964	0.963	0.963	0.964	0.965 ± 0.014
bupa	0.594	0.654	0.664	0.577	0.665	0.678	0.594	0.664	0.668	0.712 ± 0.059
credit-a	0.841	0.860	0.860	0.860	0.860	0.860	0.836	0.850	0.860	0.865 ± 0.008
diabetes	0.752	0.749	0.749	0.755	0.758	0.754	0.755	0.753	0.761	0.769 ± 0.041
glass	0.619	0.642	0.659	0.613	0.655	0.665	0.622	0.621	0.639	0.701 ± 0.099
heart-statlog	0.822	0.820	0.823	0.804	0.826	0.821	0.823	0.823	0.821	0.833 ± 0.069
ionosphere	0.826	0.836	0.858	0.816	0.845	0.864	0.814	0.833	0.871	0.866 ± 0.081
iris	0.944	0.953	0.947	0.956	0.947	0.956	0.958	0.949	0.947	0.967 ± 0.034
lymph	0.830	0.793	0.785	0.769	0.800	0.790	0.798	0.807	0.794	0.827 ± 0.097
segment	0.938	0.938	0.935	0.936	0.938	0.942	0.936	0.939	0.879	0.942 ± 0.012
sonar	0.805	0.759	0.793	0.789	0.776	0.789	0.786	0.803	0.733	0.810 ± 0.069
tao	0.904	0.879	0.879	0.906	0.901	0.891	0.910	0.895	0.891	0.899 ± 0.013
vowel	0.867	0.862	0.868	0.871	0.853	0.872	0.858	0.849	0.672	0.887 ± 0.032

Table 5.9: The mean and standard deviation of accuracy of EANCS, ANCS, NLCS, and UCS. ■/□ symbols indicate that ANCS, NLCS, or UCS is better/worse than EANCS at a significance level of 0.05.

Problem	Ensemble	ANCS	NLCS	UCS
balance-scale	0.899 ± 0.018	0.895 ± 0.024	0.903 ± 0.014	0.815 ± 0.038 □
breast-w	0.965 ± 0.014	0.970 ± 0.015	0.966 ± 0.013	0.969 ± 0.012
bupa	0.712 ± 0.059	0.663 ± 0.074 □	0.729 ± 0.064	0.683 ± 0.062 □
credit-a	0.865 ± 0.008	0.859 ± 0.032	0.859 ± 0.037	0.835 ± 0.028 □
diabetes	0.769 ± 0.041	0.746 ± 0.037 □	0.768 ± 0.042	0.748 ± 0.044 □
glass	0.701 ± 0.099	0.627 ± 0.100 □	0.604 ± 0.069 □	0.648 ± 0.116 □
heart-statlog	0.833 ± 0.069	0.832 ± 0.063	0.837 ± 0.080	0.825 ± 0.078
ionosphere	0.866 ± 0.081	0.900 ± 0.055 ■	0.880 ± 0.065	0.729 ± 0.051 □
iris	0.967 ± 0.034	0.951 ± 0.049	0.958 ± 0.048	0.949 ± 0.042 □
lymph	0.827 ± 0.097	0.822 ± 0.081	0.841 ± 0.077	0.759 ± 0.112 □
segment	0.942 ± 0.012	0.943 ± 0.014	0.923 ± 0.019 □	0.968 ± 0.008 ■
sonar	0.810 ± 0.069	0.789 ± 0.071	0.831 ± 0.059	0.735 ± 0.073 □
tao	0.899 ± 0.013	0.866 ± 0.024 □	0.891 ± 0.018 □	0.887 ± 0.015 □
vowel	0.887 ± 0.032	0.912 ± 0.035 ■	0.657 ± 0.049 □	0.910 ± 0.031 ■

compared, to show what accuracy can be obtained if a single ANCS is preferred because an ensemble is too expensive.

The results show that EANCS gets better accuracy than ANCS on four problems (bupa, diabetes, glass, and tao), and worse accuracy on two problems (ionosphere, vowel). The accuracy of both ionosphere and vowel problems seems to vary a lot with different setups of parameters as in Table 5.8, [0.814–0.871] for the ionosphere and [0.672–0.872] for the vowel. The parameter values for a single ANCS are not the same as any individual in the ensemble, which is why a single ANCS still sometimes does better.

EANCS achieves significantly better accuracy than NLCS on four problems

(glass, segment, tao, and vowel), and equivalent accuracy on the other ten problems. The ensemble version of ANCS is clearly better than NLCS. This gives a positive answer to the first research question.

Similarly, EANCS achieves significantly better accuracy than UCS on ten problems, equivalent accuracy on two problems, and significantly worse accuracy on only two problems (segment and vowel).

It was noted earlier that the segment and vowel problems are a challenge for the neural-based representation, due to the large number of possible outcomes – seven classes for the segment problem and eleven for the vowel problem. Simple neural networks have difficulty learning a large set of weights. The adaptive framework in EANCS has helped in these problems, compared to the fixed population size of NLCS, but UCS still does better unless much more training time is allowed to train the neural networks correctly.

Overall, EANCS performs at least competitively with other systems, and in most cases better, except on problems that are a challenge for any system using a neural-based representation.

5.6.3 Population Size

Figure 5.9 presents the population sizes in the final population of EANCS (the sum of all nine individuals), ANCS and UCS. As we can see, ANCS obtains much smaller population sizes in thirteen out of fourteen testing data sets, because the neural networks compact the population.

EANCS always requires a larger population than ANCS due to the use of nine ANCS systems. It requires larger population sizes than UCS on six problems, and smaller population sizes on eight problems. This means there is a trade-off between accuracy and population size.

This section investigated EANCS, having nine ANCSs with each system set up differently. The study found that the ensemble system is able to reduce bias caused by different setups of parameters, while maintaining or even improving accuracy.

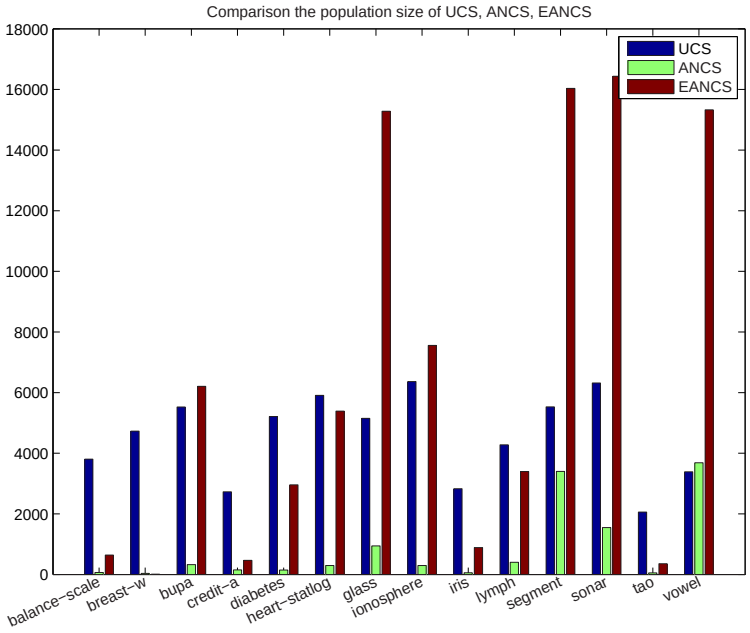

Figure 5.9: The population sizes of UCS, ANCS and EANCS

EANCS requires larger population sizes than a stand-alone system, however.

5.7 Chapter Summary

In this chapter, the author argued that the fixed population size of NLCS is a disadvantage because the decomposition of the search space by the condition part might be limited and that NLCS has to rely mainly on the neural networks in learning.

ANCS is an adaptive version of NLCS, varying the population size as needed.

The visualization tools reveal that in ANCS the condition part is able to decompose the problem correctly, and a simple network in each classifier is working well within its local area.

An investigation of ANCS reveals a set of parameters that affects the predictive accuracy of the system. This motivates the author to introduce the ensemble system, aiming at no parameter setting and better performance. The result shows that the ensemble framework can overcome the bias caused by the parameters, effectively removing them. The ensemble framework achieves equal or better predictive

accuracy than either single ANCS, NLCS, or UCS, except in problems that are a challenge for any system using a neural-based representation.

The results confirm that (1) the proposed adaptive framework is beneficial in NLCS; (2) the ensemble framework helps to reduce the number of parameters. In conclusion, this chapter was designed to answer the third research sub-question "How to reduce the bias caused by the initial choice of parameters' values?" The results showed that ANCS/EANCS is a potential solution to overcome the bias caused by the initial setup.

Chapter 6

Distributed Stream Data Mining Systems

The following paper is partially based on this chapter:

1. P. Rojanavasu, H.H. Dam, H.A. Abbass, C. Lokan and O. Pinngern (2007) A Self-Organized, Distributed, and Adaptive Rule-Based Induction System, IEEE Transactions on Neural Networks. (conditionally accepted, revised version submitted on 29-Jan-2008)

6.1 Overview

Chapter 3 proposed the framework of UCS in distributed computing environments, called DUCS. In this framework, the distributed and central sites are described in terms of a clients and a server architecture. To validate the system, it was compared against the centralized UCS, which basically transfers all data from remote sites to a central location for normal data mining. The results revealed that DUCS is competitive as a distributed data mining system due to two reasons. First, DUCS has similar accuracy to the centralized system. Second, the amount of data that needs to be transferred to the central location is reduced enormously in DUCS compared to the centralized UCS system.

This chapter focuses on how to build a distributed system based on ANCS. A comprehensive analysis of the behaviour of ANCS revealed interesting patterns in the behaviour of the parameters which motivated an ensemble version of the

algorithm with 9 nodes, each using a different parameter setting. In total they cover all patterns of behaviour noticed in the system. A voting gate was used for the ensemble. EANCS does not require any parameter setting, and showed better performance on all datasets tested. However, the computation of EANCS is normally higher than ANCS because EANCS evolves multiple ANCSs, each of which has a different parameter setting.

This chapter investigates ANCS in distributed environments, in which the training data can be logically or physically distributed. This first case is handled by EANCS. The distributed framework in Chapter 3 is revisited in this chapter with the use of ANCS to handle the latter case. The chapter focuses on the following issues:

- Stream data mining architecture: how to build a framework to handle distributed environments of stream data mining using LCS?

- Dynamic and noisy environments: how quickly ANCS/EANCS can recover after the changes and how robust the system is?

- Processing time: how fast ANCS/EANCS is to process one training instance?

- Traffic load: the amount of data being transferred in the system.

The purpose of this chapter is to verify that ANCS/EANCS, whose development in Chapter 5 was studied in a centralized environment, can be effective in a distributed environment.

This chapter is structured as follows. Section 6.2 describes the design of physical and logical distributed environments. The methodology is then given in Section 6.3, followed by an investigation of DANCS in Section 6.4. The comparison between DANCS, EANCS, and DUCS is carried out on a synthetic data set in Section 6.5 and on a large real-world data set in Section 6.6. Finally, the last section summarizes the findings in this chapter.

6.2 Distributed Architectures

This section describes the integration of ANCS in the distributed environment for data streams. In general, the environment can be distributed either physically or logically. This chapter proposes a distributed adaptive neural-based classifier system (DANCS) for the physically distributed environment, and an ensemble adaptive neural-based classifier system (EANCS) for the second environment.

Both frameworks are responsible for building a complete knowledge as a whole from local mining systems.

6.2.1 Description of DANCS

DANCS is built based on the client-server framework discussed in Chapter 3. Figure 6.1 displays the framework of DANCS. The distributed environment consists of multiple data sources, which are called clients in this thesis. A server is responsible for combining the information from local clients. This framework is a many-to-one relationship, in which all clients are required to communicate with the server occasionally in order to update their information at the server.

6.2.1.1 The Client

Clients might or might not communicate with the each other for exchanging their experience. Experiments in Chapter 3 showed that the communication between clients might help to speed up the learning at each client, especially in binary domains. The difference is not significantly observed in real-valued problems, however. Since this chapter focuses on real-valued problems, this communication will not be examined here.

Each client is an independent component, which is able to operate in its own right. A complete ANCS is employed at each client in order to obtain local patterns coming from local data sources. Data arrives at each client in a stream fashion, so its learning model updates the knowledge on the fly.

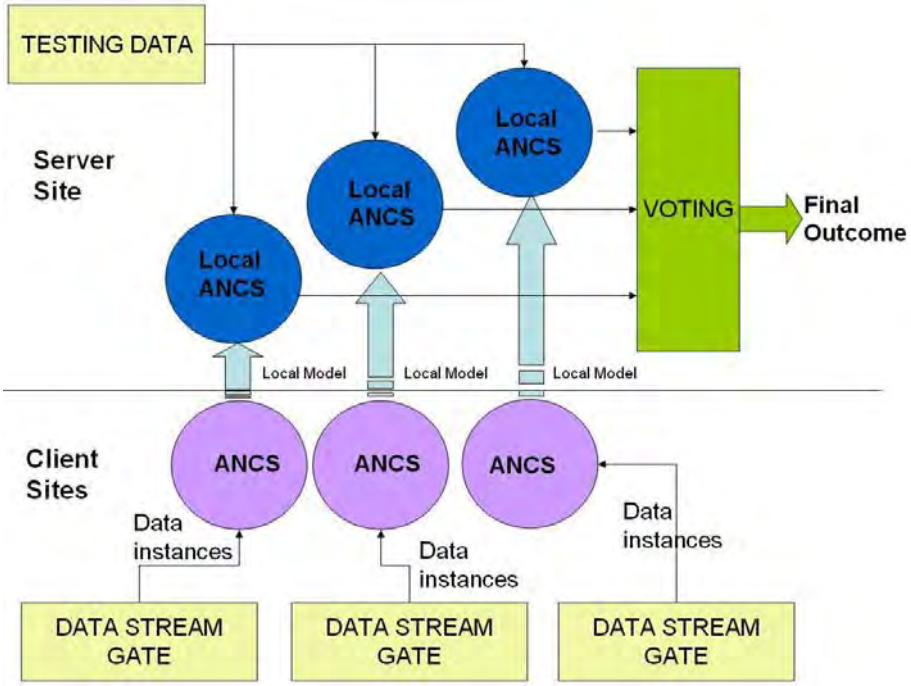

Figure 6.1: The framework of DANCS in a distributed environment

Each client is responsible for updating its knowledge at the server. The communication frequency is decided by the user. Experiments in Chapter 3 showed the trade-off between accuracy, data transmission, and up-to-date server may vary depending on the nature of the problem. In general, more frequent communication provides faster update at the server, but more traffic load is required. This chapter will not focus on this issue as it was discussed in Chapter 3.

6.2.1.2 The Server

The server contains all separate local models in its memory. Clients are responsible for updating their models regularly. The server does not make any changes to the content of local models as they are only used in the testing mode. Figure 6.1 shows three local models, being maintained at the server. These models represent the local knowledge of clients. From the management's point of view, a company might want to build a single knowledge source rather than several independent sources of information. Commonly, the local knowledge has many things in conflict due to different learning bias under different local situations. The responsibility of

the server is to integrate these local models into a single coherent knowledge-based system.

Experiments in Chapter 3 showed that the majority voting is a simple but quite efficient approach for combining the knowledge from multiple models since no training is required and acceptable performance was observed. Therefore, DANCS also employs the voting approach to combine local knowledge. Any conflict in the outcomes of local models is resolved through voting. The majority outcome will be chosen as the final decision at the server.

Each testing instance at the server is fed directly into all local models. Based on the current knowledge, each model proposes an outcome after processing the instance. At the end, the server chooses the best one within these outcomes by voting.

6.2.1.3 The Traffic Load in DANCS

Chapter 3 introduced the MDL formula for estimating the traffic flow within DUCS in terms of the number of transmission bits. The main difference between DANCS and DUCS is that the action of each classifier in DUCS is replaced by a neural network. This section provides the modified MDL formula, considering the cost of transferring neural networks.

MDL is the theory bits or the length of the model. That is the number of bits required to encode a set of classifiers for transferring in the network.

The classifiers have a common structure: $Condition \longrightarrow Action : parameters$. Their lengths are defined as follows:

$$MDL = \sum_{i=1}^{nr}(CL_i) + \sum_{i=1}^{nr}(AL_i) + \sum_{i=1}^{nr}(PL_i) \tag{6.1}$$

Where nr is the number of classifiers needed for transmission; CL_i, AL_i, PL_i are the length of a condition, an action and a set of parameters in one classifier respectively. Assuming that the interval predicate is used to encode a condition, the action is presented by a neural network, and each classifier will transfer three parameters:

fitness (a real value), numerosity (an integer), and experience (an integer). The length of each component of a classifier can be estimated as follows:

$$CL_i = 2 \times nc \times nreal \tag{6.2}$$

$$AL_i = (nc + 2 + na) \times nreal \tag{6.3}$$

$$PL_i = 2 \times nint + nreal \tag{6.4}$$

where nc is the number of features; na is the number of actions; each neural network has nc input nodes, 1 hidden node, 1 bias node, and na output nodes; $nreal$ and $nint$ are the number of bits required to encode a real value and an integer respectively. Thus, the length of data sent from a client to the server is estimated as:

$$MDL = (CL_i + AL_i + PL_i) \times nr \tag{6.5}$$

$$MDL = nr \times ((3nc + na + 3) \times (nreal) + 2 \times nint) \tag{6.6}$$

where nr is the total number of classifiers being sent to the server.

Comparing this MDL with the one in Chapter 3 ignoring the exception part, we can see that the amount of data needed to transfer using this equation is higher due to the extra data used to encode the neural network. However, since the population size is smaller using the neural-based representation, we can expect the data transmission to be smaller.

6.2.2 Description of EANCS

EANCS was presented in the previous chapter as an ensemble system for reducing the effect of the parameters setting. This chapter provides further details of using EANCS for stream data mining. In contrast to DANCS, EANCS is designed for a centralized environment, which has only one data source. Figure 6.2 demonstrates EANCS in centralized environments.

6.2.2.1 The Ensemble Framework

EANCS consists of several ANCSs, each of which is trained independently. To maintain the diversity in EANCS, each ANCS uses different parameters setups. A setup with nine ANCSs such as the one in the previous chapter is a good example of how to initialize the system.

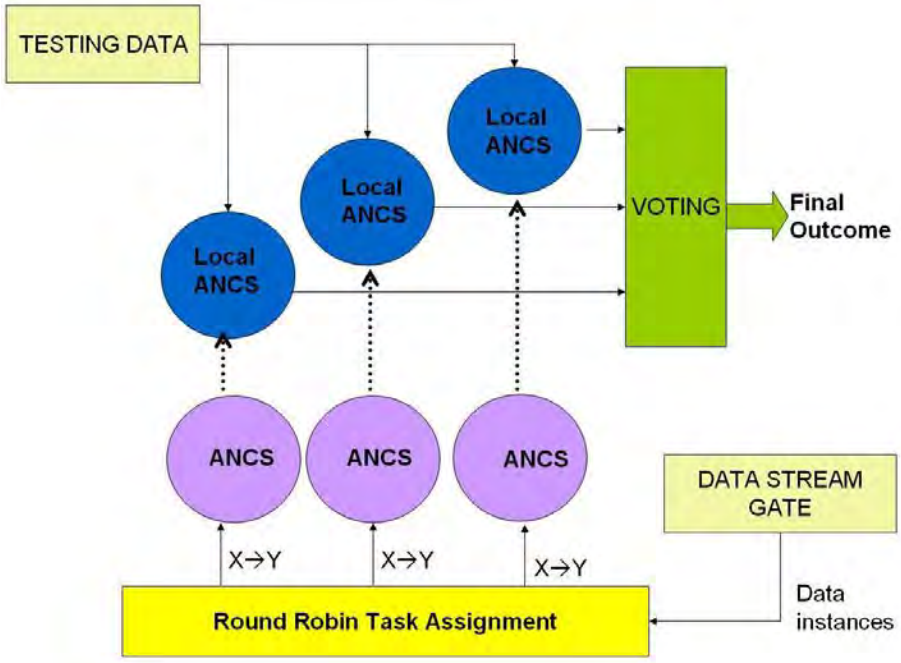

Figure 6.2: The framework of EANCS in a centralized environment

All training instances are required to go through a router that is responsible for splitting a single stream of data into several logically distributed streams of data. Each logically distributed stream of data is assigned to a complete ANCS, which operates independently. The number of ANCS can be decided based on several factors, such as the data generated speed, the computational power of the system, etc. This chapter inherits the study in the previous chapter by using similar setups with nine local models and nine sets of parameters' values.

In order to distribute the training data between multiple ANCSs, a simple task assignment function is employed at the router. The author uses round robin task assignment because this is the simplest technique that can do the job well and also requires a limited computation time.

In the training phase, the task assignment component assigns the arriving instances equally to individual ANCSs. Each ANCS learns only a part of the data set. The communication between ANCSs is not investigated in this chapter. Each ANCS is responsible for building its own knowledge independently.

If EANCS is operated in one machine, the server does not really exist. The communication between clients and server is no concern. For each testing instance, the data is fed directly to all ANCSs without going to the task assignment function. The final decision is made by the voting scheme among outcomes of all ANCSs.

It is important to make it clear that EANCS in this chapter is different to the one in the previous chapter. The previous EANCS is designed to test on small data sets, which are required to pass the whole data set through each ANCS several times. As a result, each training instance is seen by all members in the ensemble. EANCS in this chapter, on the other hand, is designed for data streams. It would be impossible to process data streams in a similar manner because they are normally large in volume and fast in arriving speed. Each local ANCS in this chapter is given a separate portion of the data, split by the router. Each portion of data might not contain complete information. But it is not really a concern because the data stream is normally very large.

6.2.2.2 EANCS in Logically Distributed Environments

It is well known in distributed environments (Cantu-Paz and Kamath, 2002) however, that we cannot simply scale the population size linearly with the number of processors. In other words, a population size of 1000 on a single processor does not map to a population size of 250 in a 4-processor environment.

One may wonder why we need an ensemble if the individual nodes require a worst case population size equal to the population size of the single UCS. A simple queuing analysis below can be used to explain.

Assume that the traditional UCS is able to process T_N instances per second, where N is the population size. Let R instances per second denote the rate of data arrival. Assume a single pass-learner (for stream data mining) as being presented

in (Dam *et al.*, 2005c).

We now need to compare between the single population approach and the ensemble environment. In this analysis, we will assume that the inter-arrival time is uniform for simplicity. In the former, if $R > T_N$, queuing theory would tell us that we need an infinite queue to accumulate the data waiting for processing. In a multi-server environment, the condition for handling this data set without the need for any queues is $R < min(U_M, M \times T_N)$, where M is the minimum number of UCS nodes in EANCS and U_M is the number of instances a router can route per second (in EANCS, the router is a simple round robin function) to distribute the incoming data to one of the M UCS populations. We assume here that all UCS nodes have the same population size and therefore, the processing time of an instance across the nodes is homogenous.

Let us take a hypothetical example to illustrate this. Assume that the data arrival rate is 10,000 instances per second. Assume that the population size for a single UCS is 1,000 rules and the time needed to scan this population size is 0.001 second; thus, in a second, the single UCS can process 1000 data instances. This implies that we can't use a single population as the queue will grow exponentially. Here we are assuming for illustrative purposes that all data require processing and the data arrives in a constant rate indefinitely. If we use a EANCS environment, we may need 11 UCS (notice that we need exactly 10 UCSs if the time to route the data is 0, but because of the time needed by the simple round robin function to route an instance is assumed in this example to be much less than the time needed to process an instance using UCS, we will need 11 UCSs). This simple example illustrates our future use of EANCS and one of the main advantages of such an architecture.

6.3 Methodology

6.3.1 Experiment Setup

This chapter conducts three experiments to test EANCS, DANCS, and DUCS in data stream environments. Since both EANCS and DUCS have been studied in previous chapters, the first experiment will examine DANCS alone. DANCS is studied on several issues such as the effect of the population threshold, the effect of learning at the server with different numbers of clients, and the learning performance in comparison to the one of DUCS. Unless stated differently, the system in this experiment uses a basic setup with three clients as in Chapter 3.

The second experiment compares EANCS, DANCS, and DUCS on a synthetic data set in noisy and dynamic environments. This experiment aims to learn the behaviour of three systems on a simulated data stream problem. Similar tests as Chapter 3 on noisy and dynamic environments are carried out for this experiment. Since EANCS has nine clients, both DANCS and DUCS are simulated with nine clients.

The last experiment investigates EANCS, DANCS, and DUCS on a large data set to simulate a data stream, using the forest data set from UCI repository (Blake and Merz, 1999). Again each system is setup with nine clients as in the previous experiment. Originally, Blackard (1998) used this data set to compare neural networks and discriminant analysis. Blackard divided the data set into three subsets: training set (1,620 instances), validation set (540 instances), and testing set (565,892 instances). The chosen training set is able to represent roughly 60% of the data set.

Since this thesis focuses on stream data mining, the whole data set is used to simulate data streams. The data is divided into ten stratified subsets; one set is used for testing, and nine sets are used for training different clients.

In this experiment, all systems: EANCS, DANCS and DUCS are set up with nine clients, where each one is trained one time by one training set. The training at clients is recorded after each 50 instances in order to monitor the learning ability.

The testing data is used at the server, after finishing all training at clients, to validate the learning of the whole system.

6.3.2 System Setup

Unless stated differently, UCS in DUCS is setup with the same parameter values used by Bernadó-Mansilla and Garrell-Guiu (2003) as follows: $v = 5, \theta_{GA} = 50, \chi = 1, \mu = 0.04, \theta_{del} = 50, \theta_{sub} = 50, m_0 = 0.1, s_0 = 0.6, N = 6400$. Two points crossover and roulette wheel selection are used.

ANCS is setup using similar parameters found in the previous chapter. Each MLP has one hidden layer and one hidden node. The learning rate of MLPs is $\beta = 0.1$. Other parameters are: $v = 10, \theta_{GA} = 50, \chi = 1, \mu = 0.04, m_0 = 0.1, s_0 = 0.1, \gamma = 0.5, P_{lb} = 1000, \theta_{Sdel} = 0.3$, covering range $c = 1$, p_max_value=1, p_min_value=0.01. Two points crossover and roulette wheel selection are used.

For the forest data set, each learner performs a stratified 10-fold cross validation runs in each data set (that is 10 runs). Each run uses different random seeds which are consistent in all experiments. The results reported in this chapter are averaged over those 10 runs.

For the multiplexer data set, each learner performs 30 runs, in which each run uses different random seeds. The results reported in this chapter are averaged over those 30 runs.

6.4 Preliminary Investigation of DANCS

This section investigates the performance of DANCS over time on synthetic data sets, in terms of the learning accuracy and the traffic load. The learning curves and traffic load of DUCS are used for comparison.

Three experiments are conducted in this section. The first one studies the effect of the lower bound population size in DANCS. The second experiment learns the behaviour of the learning at the server with different numbers of clients. This

experiment tests DANCS using three, five, seven, and nine clients. The last experiment aims at studying the predictive accuracy and traffic load over time of DANCS in comparison to DUCS. This experiment is setup with three clients and a server in both DANCS and DUCS.

6.4.1 Population Threshold P_{lb}

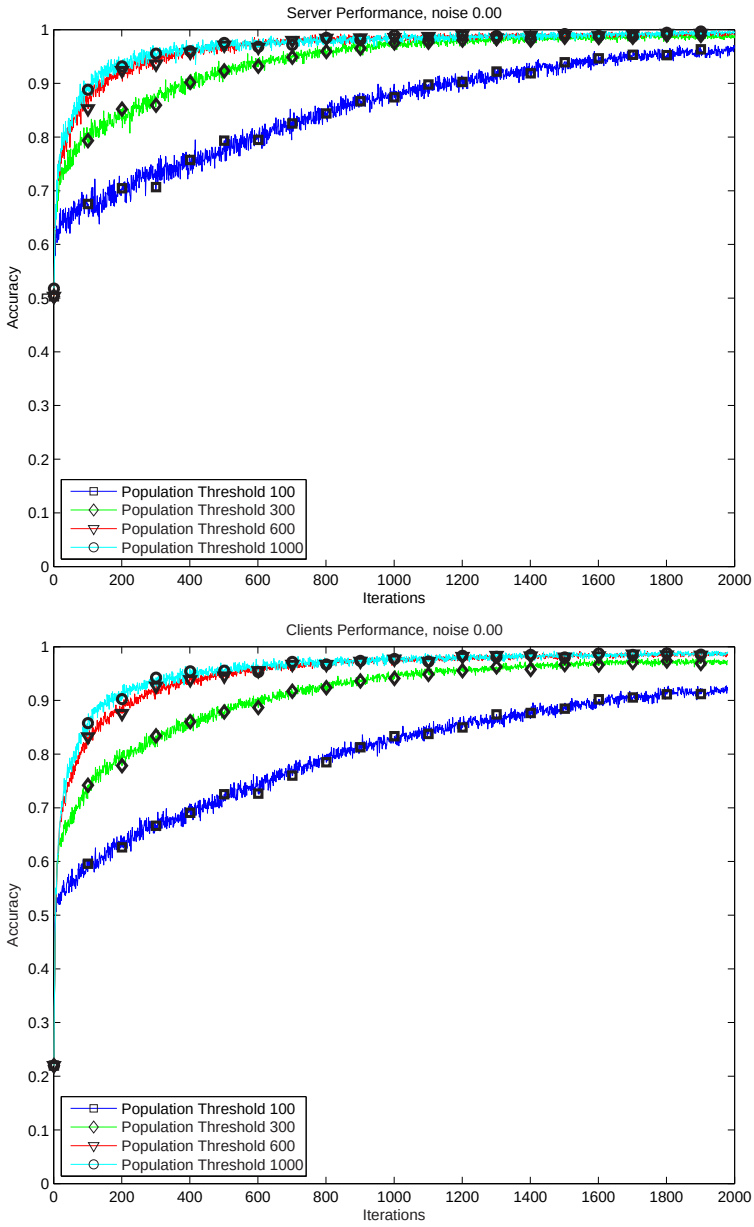

Figure 6.3: The learning curves of DANCS at the server (above) and the client (below) with different population thresholds

Figure 6.3 shows the learning curves at the client and server of DANCS with different population thresholds varied from 100 to 1000. It can be observed from the learning curves that increasing the population threshold P_{lb} speeds up the learning in both client and server.

A small population threshold (e.g. 100) increases the pressure for deletion in the system, resulting in many deletions which might also delete good classifiers before they are fully developed. Therefore the system converges slower in comparison to the other with a higher threshold.

Increasing the population threshold from 100 to 300 results in the faster learning, where the learning curve at clients rises very steeply at the beginning. Moving the population threshold from 300 and 600 also appears to have a similar behaviour. However, the difference is not clearly observed between the population thresholds 600 and 1000. It is worthy to repeat a finding from the last chapter that the population threshold affects the learning speed at the client. A population threshold of 1000 is used from here on.

Increasing the learning at clients directly affects the learning at the server. The accuracy at the server seems to be higher than at the clients, because the server can overcome the bias in one model through the combination of several models. It is quite consistent with many findings in the literature that the ensemble framework increases the accuracy in comparison to a single model.

6.4.2 Learning with Different Numbers of Clients

To understand the learning at the server, DANCS is tested with different numbers of clients. Figure 6.4 shows the learning curves of DANCS at the server with three, five, seven, and nine clients in noise-free environments.

Clearly, increasing the number of clients results in a slightly faster learning and the higher accuracy obtained at the end. The difference in the accuracy is not really significantly observed, but the increase in the learning speed can be noticed. More clients means that more knowledge becomes available at the server. Also testing

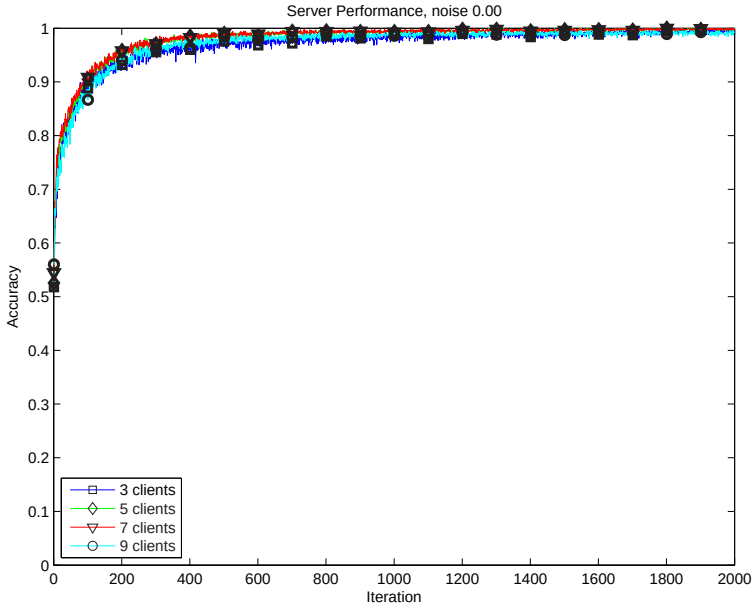

Figure 6.4: The learning curves at the server of DANCS with different numbers of clients in noise-free environments.

at the server involves the contributions from all local models, therefore overcomes more bias in the system.

A similar trend can be observed in noisy environments. Figures 6.5 and 6.6 shows the learning curves of DANCS at the server with three, five, seven, and nine clients in noisy environments: noise levels 0.1 and 0.2 respectively. Noise affects both the training speed and the training accuracy at the server.

Adding noise to the training data at the clients would result in some instability in the system. It is not easily observed in the noise level 0.1, but the learning curves with noise level 0.2 show that the learning speed is reduced in comparison to noise-free environment. A small decrease in accuracy reveals the robustness of the system in noisy environments. Similar to DUCS discussed in chapter 3, DANCS seems to be quite robust in noisy environments.

6.4.3 Comparing DANCS and DUCS

This subsection studies the learning ability of DANCS and DUCS over time. A system of three clients and the voting scheme at the server is used in this experiment.

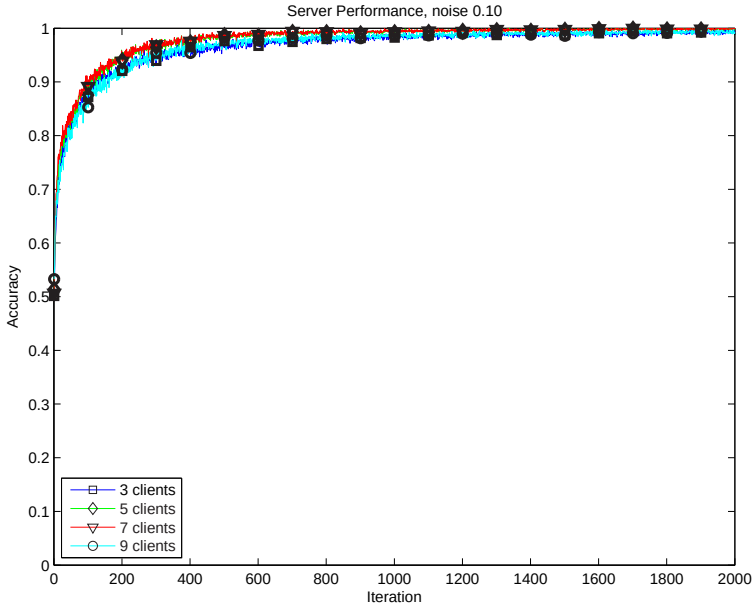

Figure 6.5: The learning curves at the server of DANCS with different numbers of clients in noisy environments (noise level: 0.1).

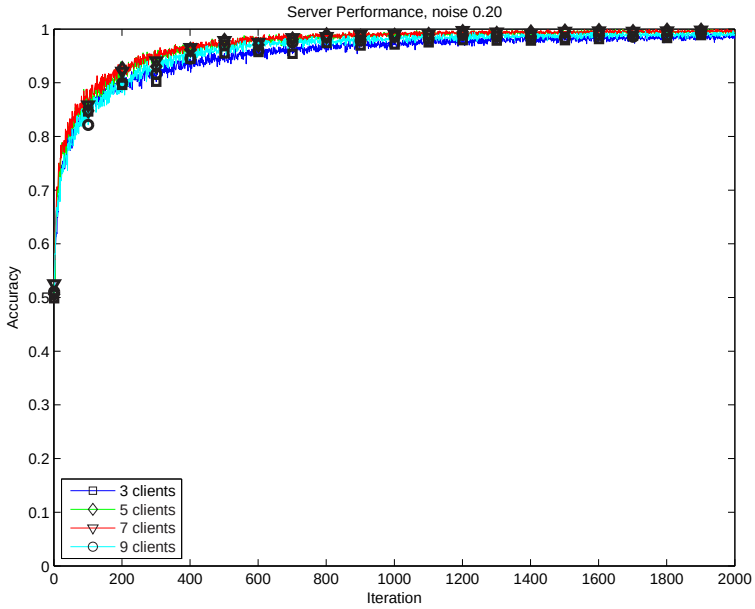

Figure 6.6: The learning curves at the server of DANCS with different numbers of clients in noisy environments (noise level: 0.2).

Figure 6.7 shows the learning curves over time at the client and server of DANCS and DUCS. The learning curves at clients indicate that DUCS learns faster than DANCS. DUCS is able to achieve up to 98% accuracy after 800 iterations, while DANCS requires as much as 1200 iterations to get similar accuracy. This is not a

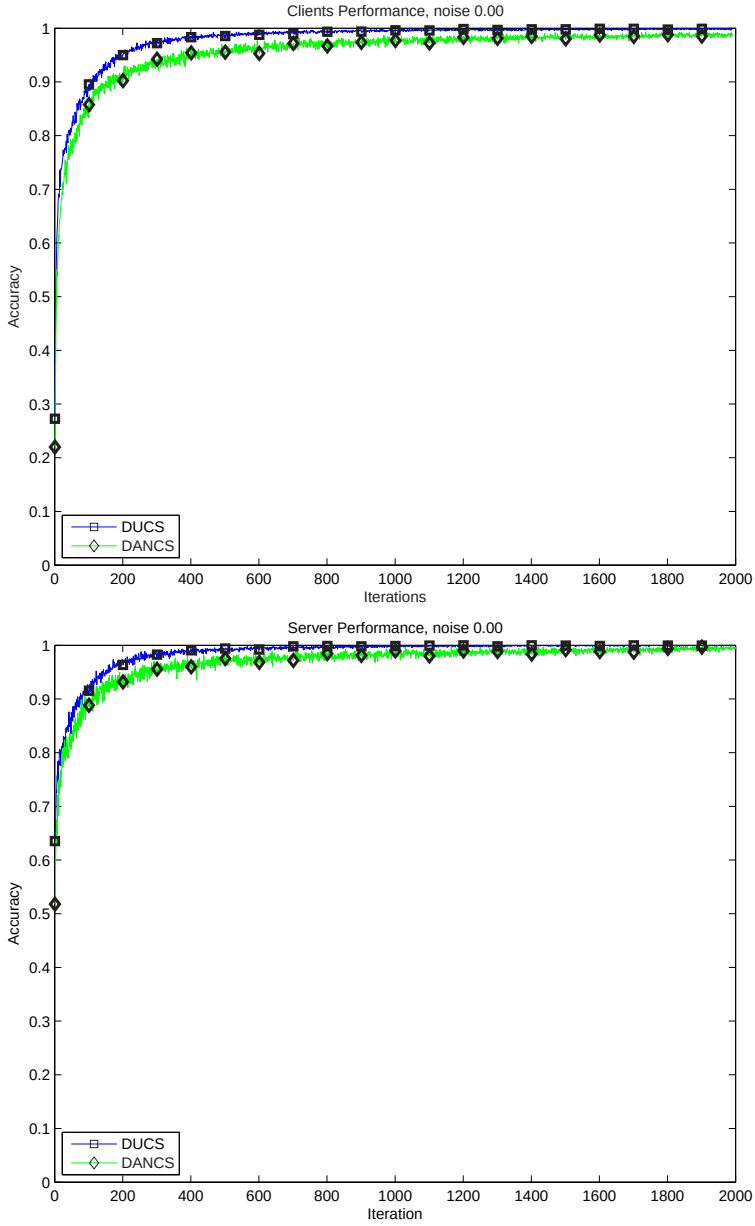

Figure 6.7: The learning curves at the server (below) and the clients (above) of DANCS and DUCS in noise-free environments

surprise. It confirms the findings in previous chapters: the neural representation requires longer to converge than the traditional representation due to the need of extra training to adjust neural networks' weights.

In noisy environments, the difference between the accuracy in both systems is not significant, as displayed in Figure 6.8. Increasing noise level from 0.1 to 0.2 degrades the performance of the server in both DANCS and DUCS. Both DANCS and DUCS are very robust since the performance does not decrease a lot even though

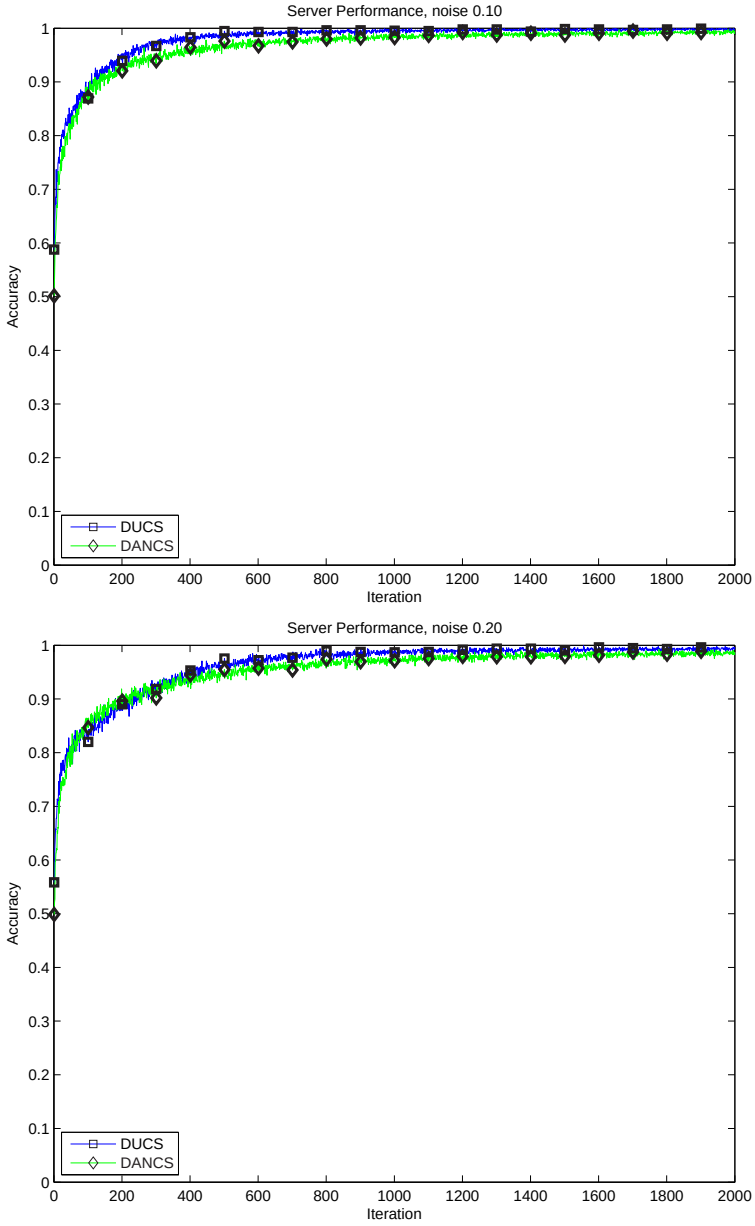

Figure 6.8: The learning curves of the server of DANCS and DUCS in noisy environments: noise level of 0.1 (upper) and noise level of 0.2 (lower)

noise level at the client is quite high (e.g 0.2 which means 20 noisy instances within 100 instances).

6.5 A Case Study of EANCS, DANCS, and DUCS in Dynamic Environments

This section investigates the three systems EANCS, DANCS, and DUCS on a synthetic data set, simulating a dynamic environment. A concept change is simulated in a similar way as in Chapter 3. Noisy data is also investigated.

6.5.1 Small Changes

6.5.1.1 Noise-free Environments

Figure 6.9 shows the system performance of the client and server on the 6-real-multiplexer in the dynamic environment with small changes at the 2000^{th} iteration. Each point in the graph represents the classification accuracy of 50 time steps. In this experiment, the threshold is changed one time from 0.1 to 0.2 (or MoC=0.1). The graphs of both the clients and server display two cycles of performance curves, where each cycle corresponds to one threshold value.

The performance of the server and the clients are synchronized with each other because the server gains its knowledge from the local models obtained at the clients. When the environment changes, the client models of the previous time step become unsuitable. It results in many misclassified instances at that time. It requires a period of time for the clients to recover and adapt to the new situation. Since the clients keep sending incorrect learning models to the server, the bad performance at the server can be understood at the beginning of the environment changes. Again, the server seems to converge faster and better than the clients because knowledge from the clients may be obtained differently depending on the training data. One client might learn something which another client at that time has not yet learned.

At the 2000^{th} iteration, the threshold is changed from 0.1 to 0.2. Any real numbers in the range $[0.1, 0.2)$ in the previous time step were considered as a binary 1, but now become a binary 0. Since data is generated with a uniform distribution, about 10% of the real numbers are converted to the binary number differently in

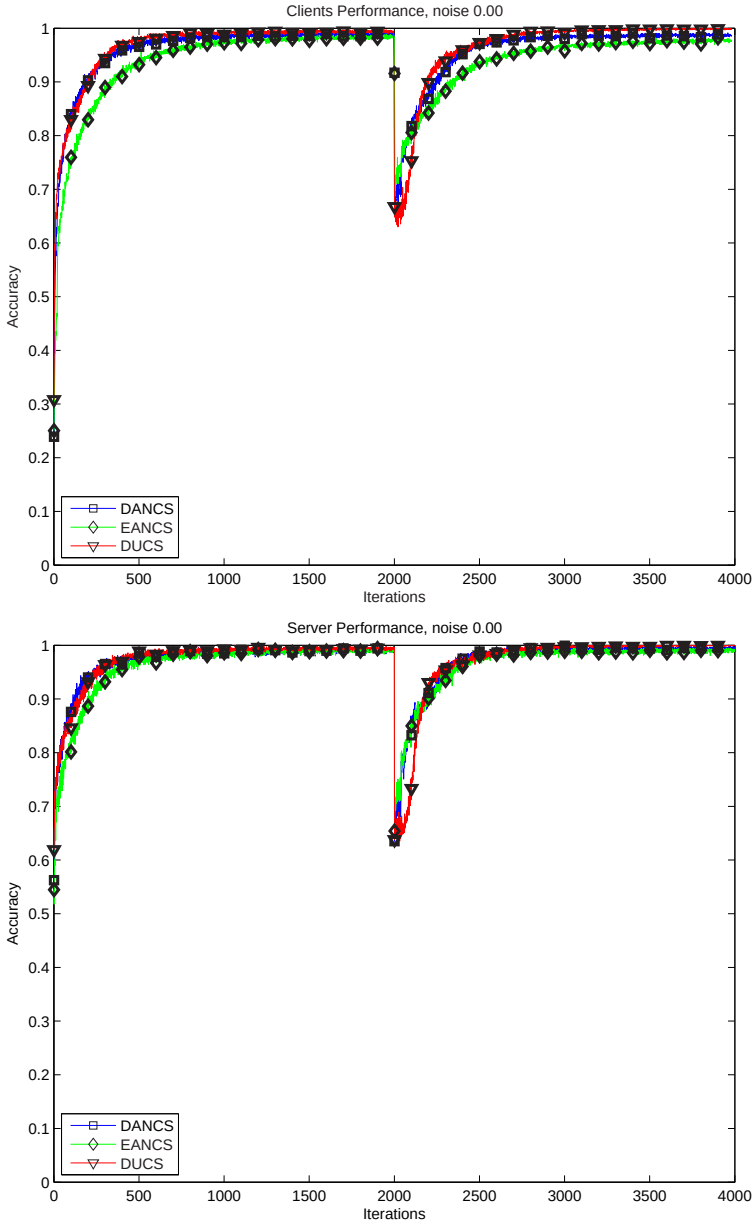

Figure 6.9: The learning curves at the client (above) and the server (below) of DANCS, EANCS, and DUCS in a noise-free environment with a small concept change (MoC=0.1)

the new environment. There are $50 \times 6 = 300$ real numbers required for 50 training instances (a training window). 10% of them, which is about 30 real numbers, would be converted differently. Therefore, at least 5 instances (6 real numbers form an instance) and at most 30 instances contain real numbers which are converted differently. At least 20 instances (40%) in the training window are unchanged in the new condition. This explains why the learning curve does not drop as low as

the starting point at time step 0, but drops down to about 80% accuracy.

All three systems evolve a *best action map*, which contains only correct rules (correct classification) in the population. Since the change is small, only part of the population becomes incorrect. The rest needs to update the quality of their parameters in order to adapt to changes in the environment. As a result, the system would adapt quickly to changes. Thus, the learning curve manages to recover faster than the first cycle.

The accuracy at the server of DANCS and EANCS right after the change is higher than DUCS because of their neural networks that can provide different outcomes based on the conditions. However, DUCS is able to adapt more quickly to a change than both DANCS and EANCS, as DUCS converged faster at the client. It is because DANCS and EANCS need additional time for training neural networks on top of the time needed to eliminate incorrect classifiers; to discover new and potentially useful classifiers; and to update the population.

After the change, the learning at the client of both DANCS and EANCS does not have much difference, but the learning at the server does. DANCS is able to achieve better accuracy at the server than EANCS. It is because EANCS employs different parameters' setups at each client. DANCS, on the other hand, employs the best parameters' setup suggested in the previous chapter. Diversity is important for complicated problems, such as those ones of UCI datasets, but the multiplexer problem is a straight forward problem, where the hyper-rectangular condition plays an important role to decide on the performance. The best parameters' setup seems to provide better accuracy in comparison to a set of parameter setups.

Figure 6.10 shows the traffic load from one client to the server in DANCS and DUCS (EANCS is not considered since it simulates a logically distributed but physically centralized environment). The improvement in the client performance results in less misclassified instances and a smaller learning model at the end of each cycle. The more training one system receives, the more compact its model becomes. All systems keep filtering out irrelevant and incorrect classifiers in order to maintain a smallest set of correct and general classifiers.

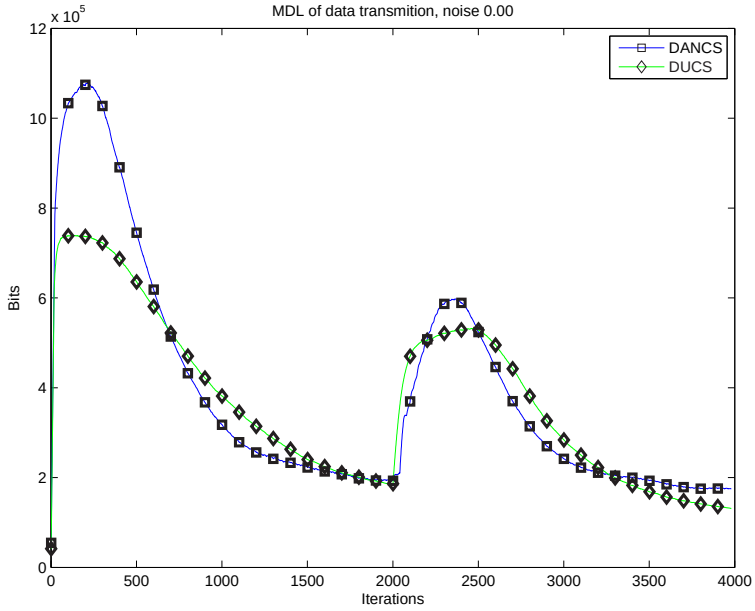

Figure 6.10: Data transmission of DANCS and DUCS in noise-free environments with a small concept change (MoC=0.1)

When the environment changes, the models become partially incorrect. Some correct classifiers in the previous cycle certainly become misclassified ones that increase the size of the learning model. The genetic algorithm needs to introduce more classifiers into the system in order to expand the search space for adapting to the new situation. That is why the traffic load in the three systems grows after the change.

The traffic load of DUCS is normally less than DANCS due to two reasons. The first one is that the neural representation of the action in two systems require extra bits in comparison to the one of DUCS. Secondly, the real-multiplexer is more suitable to DUCS because the condition of DUCS is able to decompose the search space correctly using the hyper-rectangle condition. For the real 6 multiplexer problem, eight classifiers are required to represent the whole search space. Therefore, the neural representation is not really an advantage in comparison to the traditional representation in this problem. However, all three systems seem to converge to a similar amount of bits required in the end.

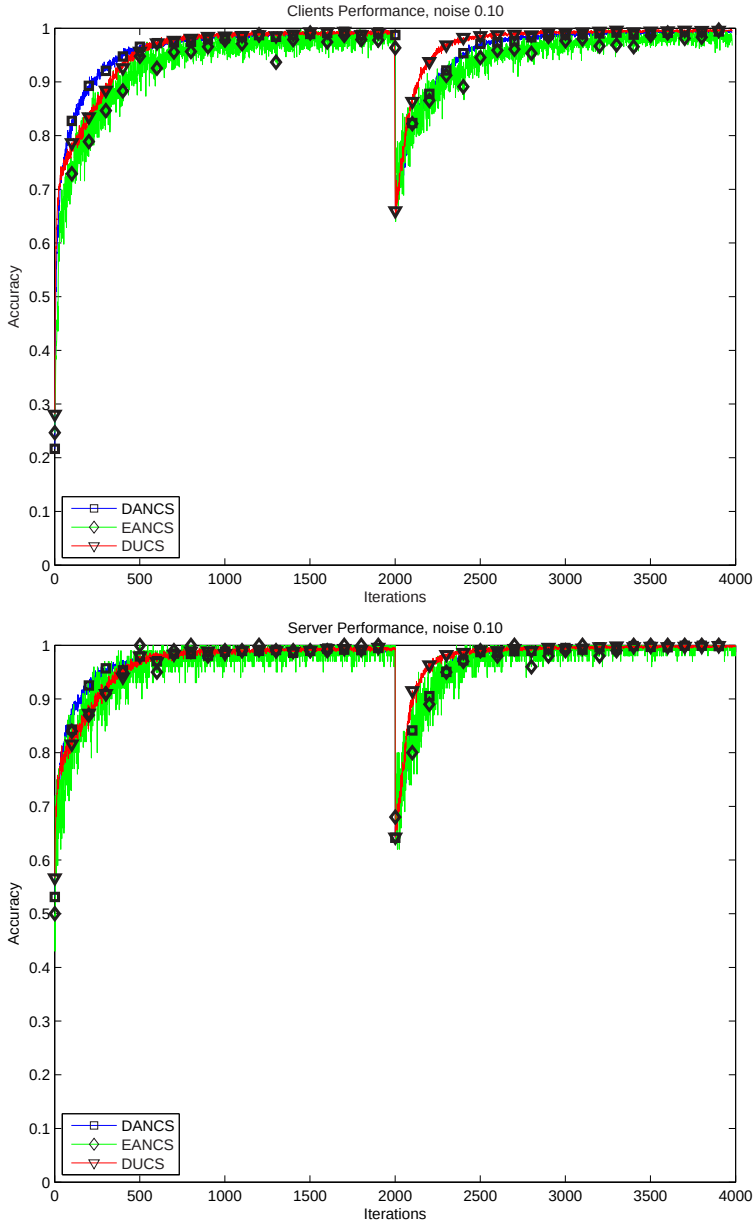

Figure 6.11: The learning curves of DANCS, EANCS, and DUCS at the server (below) and the clients (above) in a noisy environment with a small concept change (MoC=0.1, noise=0.1)

6.5.1.2 Noisy Environments

Figure 6.11 shows the system performance of the client and server on the 6-real-multiplexer in the dynamic environment with small changes from 0.1 to 0.2 (or MoC=0.1) in noisy environments. The graphs of both the clients and server display two cycles of performance curves, in which each cycle corresponds to one threshold

value.

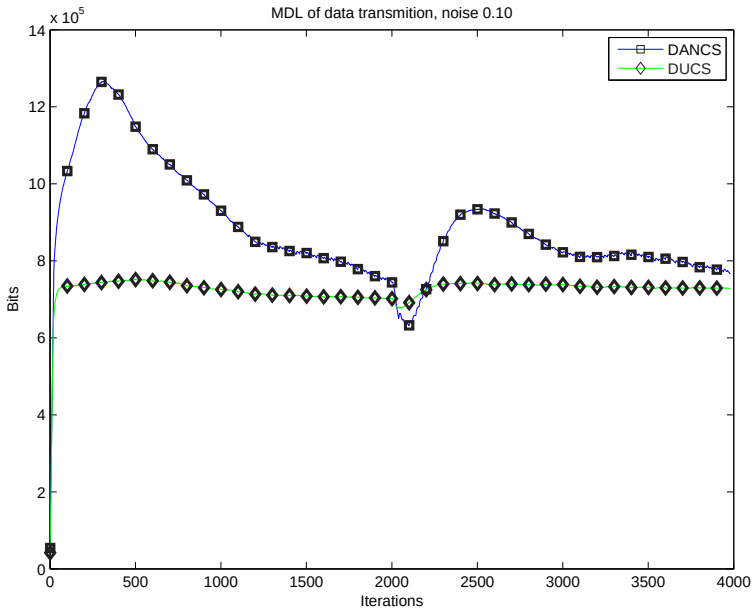

Figure 6.12: Data transmission of DANCS and DUCS in a noisy environment with a small concept change (MoC=0.1, noise=0.1)

Adding noise to the training instances in all three systems does not affect the accuracy at both client and server, but the traffic load is affected as showed in Figure 6.12. DUCS is unable to compact its population, and the traffic load becomes stable after the increase at the beginning. DANCS, on the other hand, are able to compact the population as their transmission data drops less than DUCS at the end of cycle 1. It indicates that the neural representation is more robust to noise in comparison to the traditional representation.

In conclusion, experiments in this section show that both DANCS and EANCS are able to recover in dynamic environments as well as DUCS. In noisy environments, DANCS and EANCS are able to generalize better than DUCS.

6.5.2 Severe Changes

This subsection investigate both DUCS and DANCS in dynamic environments with severe change. In this experiment, the threshold is changed in the order of MoC=0.8 (from 0.1 to 0.9), after 2000 iterations.

6.5.2.1 Noise-free Environments

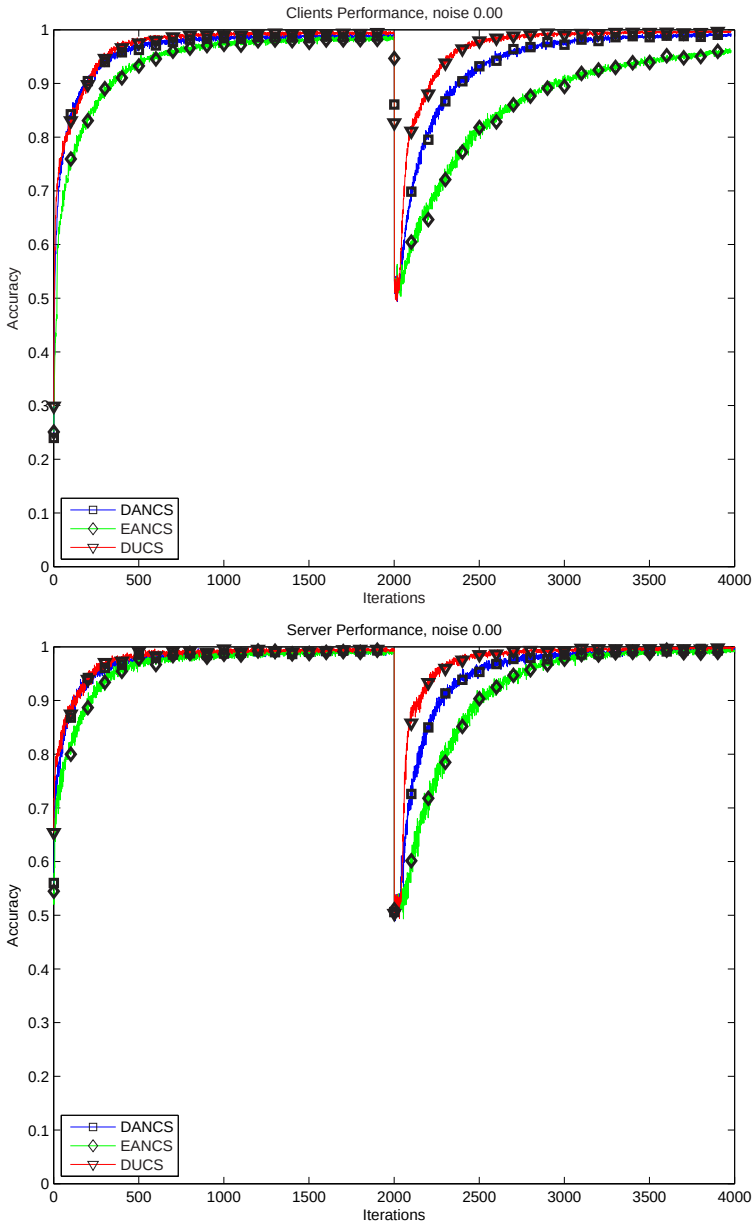

Figure 6.13: The learning curves at the client (above) and the server (below) of DANCS, EANCS, and DUCS in noise-free environments with a severe concept change (MoC=0.8)

Figure 6.13 shows the learning curves of the server and the client on 6-real-multiplexer problem with severe changes to the underlying data in noise-free environments.

Clearly, DUCS is able to adapt more quickly than both DANCS and EANCS.

It can easily be observed that the accuracy at the server of DUCS gets back to approximately 100% at the 2500^{th} iteration, whereas both DANCS and EANCS seem to reach to the similar accuracy at the 3000^{th} iteration. After that all three systems achieve the same level of accuracy. The slower convergence in DANCS and EANCS can be blamed on the neural representation, where the neural networks require extra time to revise their knowledge.

Taking a close look at the learning performance of DANCS and EANCS, we can see that EANCS converges slower than DANCS, especially at the client. One set of parameters in DANCS seems to be better than several sets of parameters in EANCS. It can be explained that some sets of parameters in EANCS might not be suitable for this problem, whereas parameters' values of DANCS seem to be better. However the similar level of accuracy at the server again confirms that the ensemble approach is able to overcome the bias of different parameters' setups at clients.

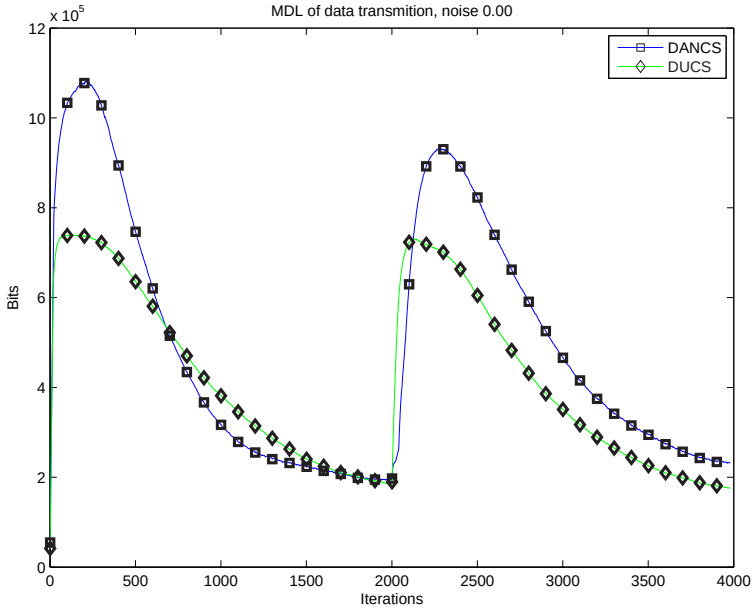

Figure 6.14: Data transmission of DANCS and DUCS in noise-free environments with a severe concept change(MoC=0.8, noise=0.0)

Figure 6.14 shows the data transmission using the MDL function of DANCS and DUCS in the noise-free environment with a severe dynamic change. Similar to the result of previous section, the data transmission in DUCS is less than that of DANCS. Again it confirms that the multiplexer is indeed more suitable for DUCS

than DANCS.

6.5.2.2 Noisy Environments

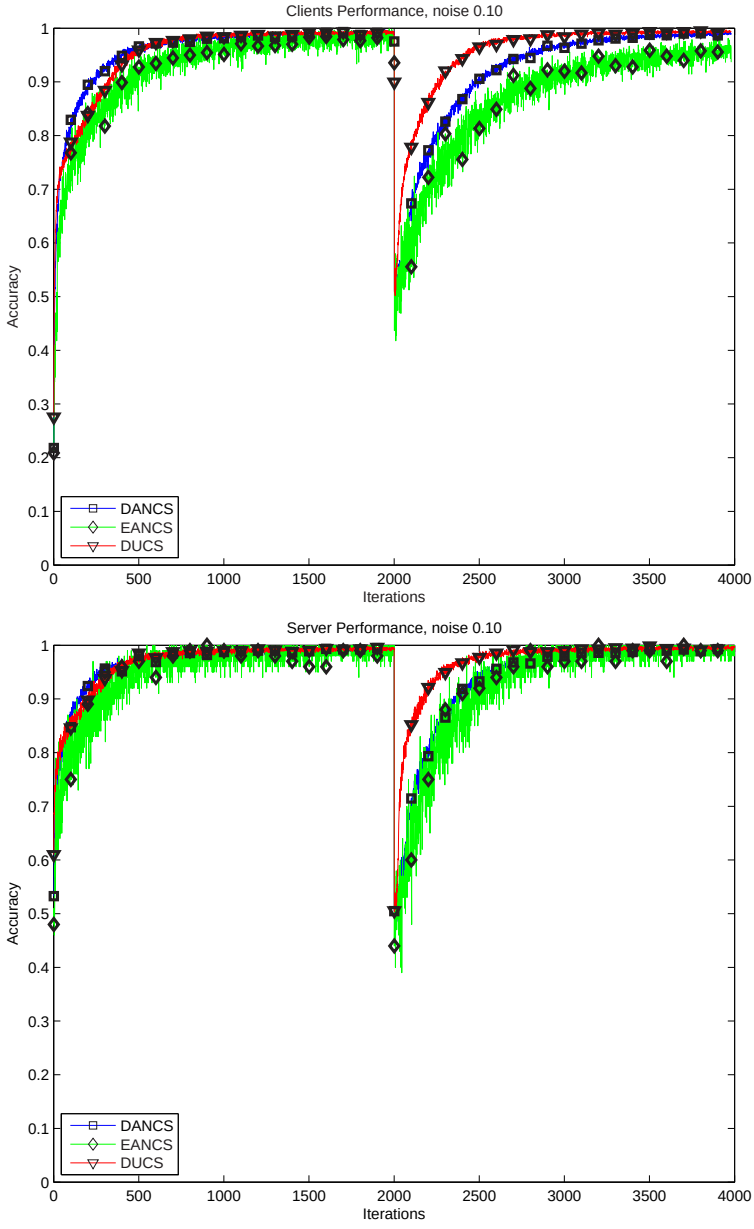

Figure 6.15: The learning curves at the client (above) and the server (below) in noisy environments with a medium concept change (MoC=0.5, noise=0.10)

Figure 6.15 shows the learning curves of the server and the client on 6-real-multiplexer problem with severe changes to the underlying data in the noisy environment. All three systems are able to achieve quite similar accuracy as in the

noise-free environment. Similar behaviour in EANCS, DANCS and DUCS in the noise-free environment can be observed here. This indicate that all three systems are robust to noise as the learning accuracy does not drop in the noisy environment.

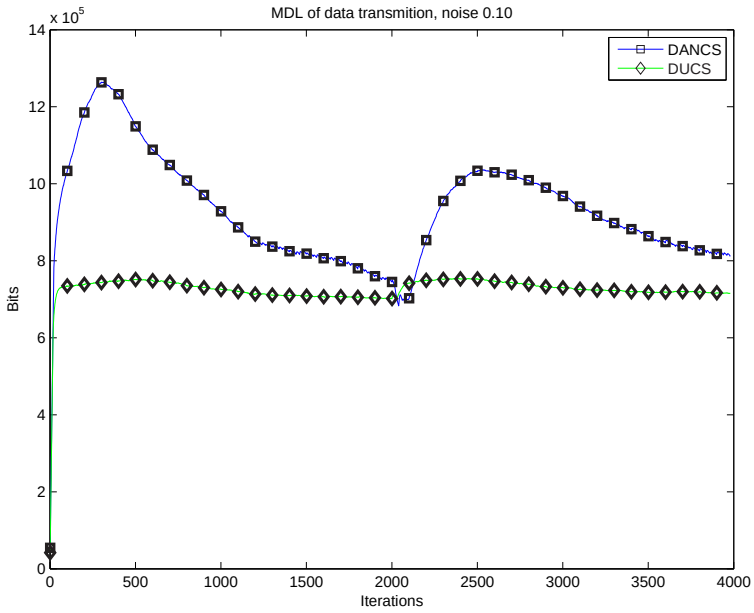

Figure 6.16: Data transmission of DANCS and DUCS in noisy environments with a severe concept change (MoC=0.8, noise=0.10)

Figure 6.16 shows the data transmission of DANCS and DUCS in the severe dynamic and noisy environment. Again, DUCS does not seem to generalize as the amount of data transmission in the system stay stable after the peak. DANCS, on the other hand, seems to generalize very well as the population increases at the beginning, but then decreases dramatically. At the end of the first cycle, it seems to achieve a smaller model than DUCS.

In conclusion, experiments in this section show that the multiplexer problem is more suitable for DUCS than both DANCS and EANCS. Due to the neural-representation, both DANCS and EANCS seem to converge slower than DUCS. However, both systems are able to recover after the change in both noise-free and noisy environments. Moreover, all three systems seem to be robust to noise as the learning accuracy does not decrease in comparison to the noise-free environment.

6.6 A Case Study of DANCS, EANCS, and DUCS on a Large Data Set

This section presents an experiment with DANCS, EANCS and DUCS on a large and real data set.

6.6.1 The Forest Data Set

The forest cover type data set of the Roosevelt national forest in northern Colorado (Blake and Merz, 1999) is chosen for testing in this section. According to Blackard (1998), the collected data covers an area of 70 miles northwest of Denver in Colorado, which has seven major forest cover types (or seven classes). The data was obtained from the U.S. Geological Survey.

The data set has 581,012 observations, 54 attributes, and no missing values. This data is used to model a stream input by continuously feeding instances into the system. The processing time will be reported in order to measure if the system is able to handle a stream of data.

6.6.2 Learning at Clients

This subsection will investigate the learning at the client in all three systems using two experiments: the noise-free environment and the noisy environment.

6.6.2.1 Noise-Free Environments

Figure 6.17 shows the learning curves over time of EANCS, DANCS and DUCS during the training at clients. We can see that all three systems are able to achieve similar accuracy and similar learning speed over time. The accuracy of DUCS seems slightly better than both DANCS and EANCS, even though the difference is not statistically significant. The 70% accuracy that these three systems achieve is equivalent to the accuracy obtained in (Blackard, 1998). DANCS and EANCS

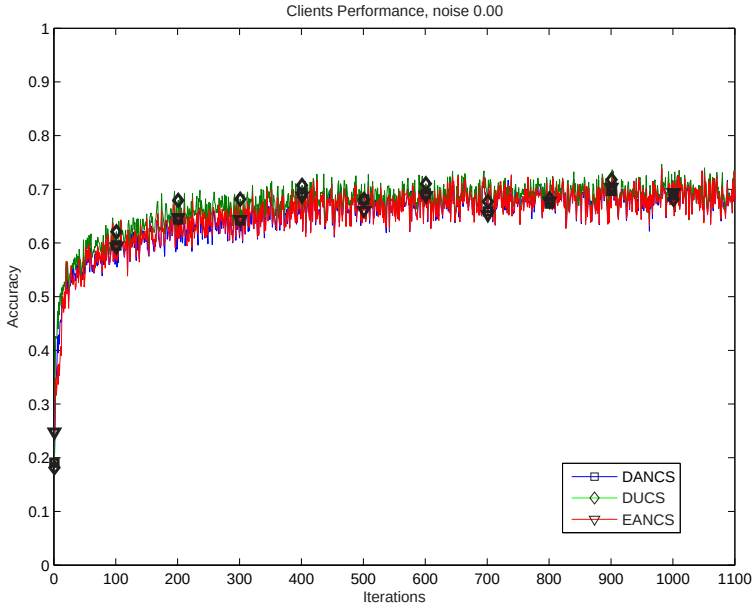

Figure 6.17: The training accuracy of clients on the forest problem

then catch up with DUCS and the learning curves of three systems seem to merge in the end. Again, it confirms that the neural representation in many situations is slower than the traditional representation, since the internal learning of each classifier requires extra time for training neural networks' weights.

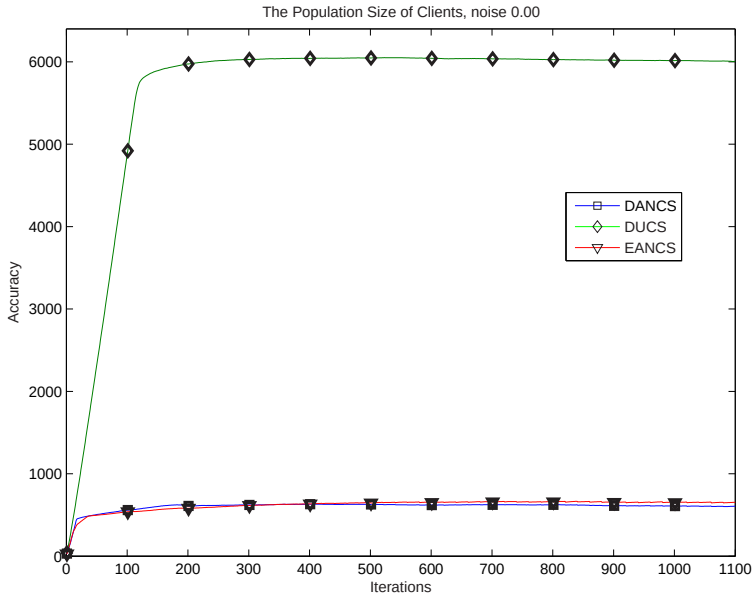

Figure 6.18: The population size of clients on the forest problem

Figure 6.18 shows the population size curves over time of EANCS, DANCS and

DUCS during the training process at clients. The population sizes of both EANCS and DANCS are quite similar. The population size of clients in DUCS is much larger than both other systems. Even though DUCS seems to generalize in the end as the population size gradually decreases over time, the final population sizes of EANCS and DANCS are much smaller than DUCS.

This study on the large data set shows that both EANCS, DANCS and DUCS are able to perform as well as each other on a large data set. EANCS and DANCS are able to offer a more compact population size than DUCS in the end.

6.6.2.2 Noisy Environments

Figure 6.19 shows the learning curves of EANCS, DANCS, and DUCS in noisy environments. Adding a noise level in training instances directly affect the testing performance at the client in all systems. In the noise-free environment, the final accuracy of three systems is approximately about 70%. The accuracy is decreased to about 60% and to around 55% with noise levels 0.1 and 0.2, respectively.

Figure 6.20 shows the population size curves of EANCS, DANCS, and DUCS in noisy environments. Surprisingly, the population sizes of all systems do not increase as in experiments with the multiplexer in the previous section. It is because the large number of attributes in this data set normally leads to a large number of classifiers anyway in order to cover the whole input space. In this case, a longer time is normally required for the population to generalize well.

6.6.3 Learning at the Server

Table 6.1: The mean and standard deviation of accuracy at the server of EANCS, DANCS, and DUCS

Noise	DANCS	EANCS	DUCS
0.00	0.6932 ± 0.0030	0.7071 ± 0.0021	0.6928 ± 0.0021
0.10	0.6923 ± 0.0021	0.7072 ± 0.0017	0.6941 ± 0.0035
0.20	0.6920 ± 0.0029	0.7066 ± 0.0023	0.6904 ± 0.0036

This section discusses the predictive accuracy at the server. Table 6.1 presents

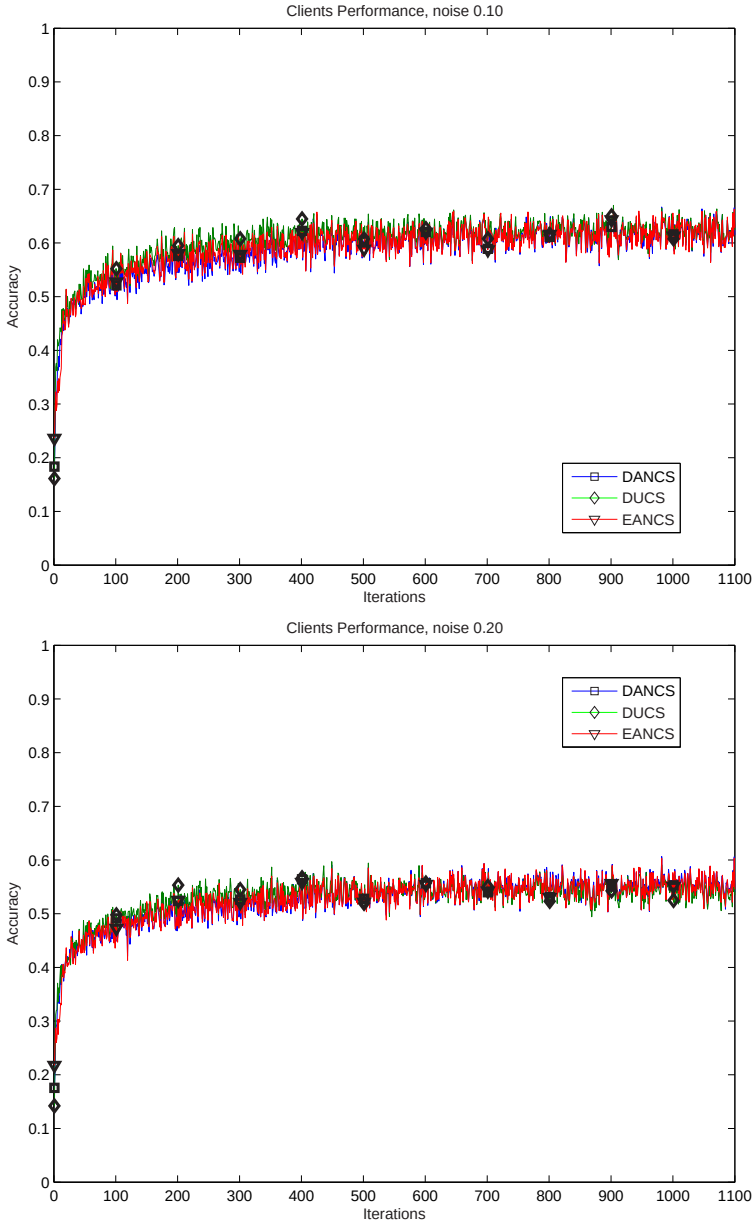

Figure 6.19: The training accuracy at the client on the forest problem in noisy environments: noise level 0.1 (upper) and noise level 0.2 (lower)

the mean accuracy and standard deviation at the server of EANCS, DANCS, and DUCS. A statistical t-test is carried out to test the significance of differences in the accuracy between three systems.

The accuracy of EANCS appears slightly better than both DUCS and DANCS, even though no statistically significant differences are observed. The use of different parameter setups helps to diversify local ANCSs and therefore increases the

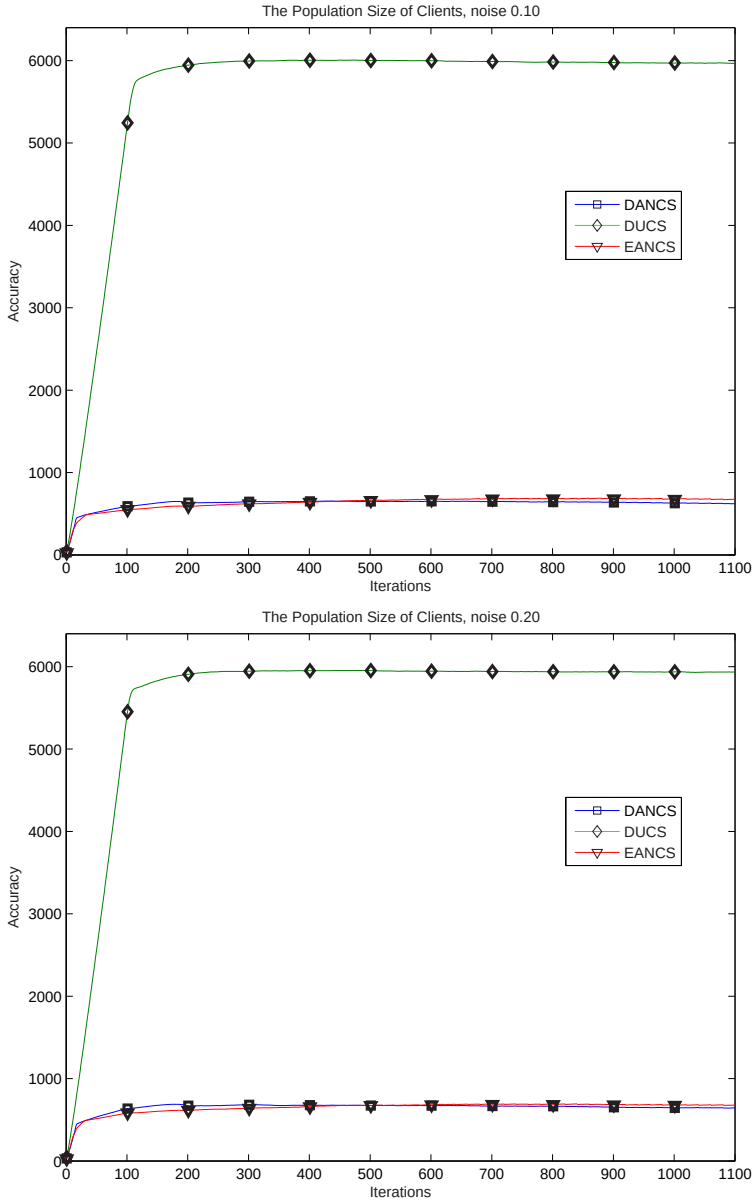

Figure 6.20: The population size at the client on the forest problem in noisy environments: noise level 0.1 (upper) and noise level 0.2 (lower)

accuracy at the server after the combination.

6.6.4 Processing Time

Table 6.2 shows the training time (in seconds) of EANCS, DANCS, and DUCS. The time is estimated by averaging the total real time needed during the training.

DUCS requires much longer to process an instance, because it has a larger

Table 6.2: The training time required for each data instance in noisy environments

Noise	DANCS	EANCS	DUCS
0.00	0.0445	0.0882	0.4531
0.10	0.0492	0.0807	0.4481
0.20	0.0516	0.0839	0.3351

population size than both EANCS and DANCS. Experiments in Chapter 3 showed that the matching time to form a match set is the step which takes most of the processing time in learning classifier systems. Reducing the population size from 6000 classifiers to less than 800 classifiers means much less time is required for matching for each training instance.

EANCS seems to require more time than DANCS. It is because the population size of DANCS is smaller than that of EANCS in both noise-free and noisy environments. A slight difference in the population size results in a large difference of the processing time (as much as double the time needed in EANCS in comparison to DANCS).

Again, it comes to the trade-off between the accuracy and the population size in order to decide if DANCS or EANCS is better. EANCS is able to offer better accuracy at the server, but requires longer processing time than DANCS. Moreover the distribution time on top of the processing time to divide the data into smaller subset also needs to be considered.

However, both DANCS and EANCS are designed for two different environments: physically and logically distributed environments. The comparison between two systems is not really a big concern in this thesis. Experiments in this chapter confirm that the neural representation is able to reduce the traffic in distributed environments, to be able to handle more data instances in comparison to DUCS. These are two main concerns in stream data mining.

Table 6.3 shows the testing time for an instance (in percentage) of EANCS, DANCS in comparison to DUCS.

It is clearly shown that both DANCS and EANCS require much less time than DUCS during the testing phase. The testing time of DANCS and EANCS is

Table 6.3: The testing time required for each data instance in noisy environments

Noise	DANCS/DUCS	EANCS/DUCS
0.00	13.52%	19.89%
0.10	13.21%	18.45
0.20	14.15%	19.51%

measured around 13-14% and 19-20% of the time needed for DUCS, respectively.

6.7 Chapter Summary

The aim of this chapter is to compare the traditional UCS and enhanced UCS in stream data mining. Three issues of stream data mining: (i) processing time, (ii) dynamic and noisy environments and (iii) distributed environments, are investigated in this chapter.

Three systems are studied in this chapter: the Distributed Supervised Classifier System (DUCS), the Distributed Adaptive Neural-based Classifier System (DANCS), and the Ensemble Adaptive Neural-based Classifier System (EANCS). The first system has been proposed in Chapter 3, using traditional UCS in distributed environments. The last two systems employ the enhanced UCS with the neural representation in distributed environments. DANCS is proposed in this chapter for physically distributed environments. EANCS has been discussed in the previous chapter and extended in this chapter for logically distributed environments. Both synthetic and real data sets are tested in these systems.

The first experiment was taken on DANCS alone as both DUCS and EANCS have been partially tested in previous chapters. Several issues were investigated, such as the number of clients, and noisy environment, on a synthetic data set – the multiplexer problem. The results show that DANCS is able to achieve similar accuracy as DUCS.

The second experiment tests three systems in dynamic and noisy environments. Two cases of dynamic environments were simulated: small change and severe change. The results reveals that DUCS, DANCS, EANCS are capable of

adapting to environment changes and reusing information gained from the past. Although DUCS seems to converge faster than both DANCS and EANCS, all three systems are able to achieve quite similar accuracy at the server in the end. Moreover all three systems show the tolerance to noise as the learning accuracy does not seem to be affected by noise.

The last experiment was taken on a large real-world data set: the forest cover types, with more than half a million instances. The results showed that EANCS, DANCS and DUCS are able to perform as well as each other at the client. EANCS is able to get slightly higher accuracy than both DUCS and DANCS. DANCS is able to obtain a smaller model than EANCS. Both DANCS and EANCS are able to get much smaller population size than DUCS. Moreover, the testing time of DANCS and EANCS is reduced to approximately 14% and 20% of the one of DUCS, respectively.

Testing on the forest data set again confirms that the neural representation and the adaptive framework is able to reduce the population size in comparison to non-enhanced version. By reducing the population size, both the traffic load in the system and the processing time of each training instance are dropped dramatically. However, it is not always a case as shown on the multiplexer problem. The nature of this problem is likely to favour the representation (with lower and upper bounds) of UCS, and therefore smaller population is observed on this problem.

Experiments in this chapter confirm the last research sub-question in Chapter 1 that ANCS/EANCS can be effective in a distributed environment and also suitable for stream data mining.

Chapter 7

Conclusions and Future Research

7.1 Summary of Results

This thesis presents a study of the evolutionary-based classifier on distributed stream data mining problems. The Michigan-style classifier system is chosen in this thesis because it is widely accepted in the literature as a more useful approach in an online, real time environments than the Pittsburgh approach. UCS was chosen as a base system for investigation because the thesis focuses on supervised learning.

A comprehensive analysis of UCS in distributed stream data mining indicates that it is potentially useful, as it is robust to noise and is able to recover quickly after a concept change. The study also reveals that the large population size required to cover the whole search space is a key drawback in UCS. Most of the time needed for processing an instance is used for matching it against all classifiers in the population. As a result, longer processing time is a bottleneck of UCS when faced with data streams.

In order to reduce the population size, a neural representation is employed in UCS in order to capture more information within a classifier. The traditional action in UCS is replaced by an artificial neural network. A very simple neural network is used so that the expressiveness of the rule-based knowledge can be protected.

The enhanced UCS is improved further by employing an adaptive framework

that internally controls several parameters' values by observing the training performance. An ensemble framework is applied to reduce the effect of initial parameters' setups.

Finally, the proposed system is tested on distributed data streams. Two distributed frameworks are proposed to help the system in distributed environments. The first framework simulates a physically distributed environment using a client-server architecture. The second framework employs an ensemble framework to logically distribute the data streams.

Overall, contributions of this thesis can be summarized as follows:

1. An empirical study of UCS on distributed stream data mining is presented in Chapter 3. The results confirm that UCS is able to handle several issues of data streams, such as concept change and noisy environments. The population size is the key drawback of UCS as it is very sensitive: too small a population might delay or sometimes prevent the convergence of UCS. Expanding the population size will affect the processing time of UCS. Moreover UCS normally requires a large population size to cover the whole search space. This factor affects directly the processing time of UCS, as most of the time is used for matching.

   The client-server framework helps UCS perform as well as the centralized framework in distributed environments, in terms of the predictive accuracy and convergence rate. Furthermore, traffic can be reduced within the system and also the raw data is protected from being transferred through the network.

   The aim of Chapter 3 is to answer the first research sub-question discussed in the first chapter. The results showed that UCS is useful for distributed data streams in terms of robustness, ability to recover after concept changes, and ability to work in distributed environments. However, the results reveal that UCS requires a large population to cover the whole search space. Also, the overall learning performance of the system is very sensitive to some parameters, such as the maximum population size.

2. The neural representation was proposed in Chapter 4 to replace classifiers' ac-

tion. Experiments on UCI data sets confirmed that the neural-based learning classifier system (NLCS) is better than UCS since a much smaller population is required in order to achieve equivalent accuracy.

The main findings in this chapter are NLCS is able to achieve equivalent or even better accuracy than UCS. The incorporation of negative correlation learning during the training of NLCS improves further the predictive accuracy of the system because the networks in the same regions are localized.

This chapter is motivated by the findings in the previous chapter. The study in this chapter aims at answering the second research sub-question that "is a neural network's representation beneficial for evolutionary learning classifier systems?" Indeed it is found to be beneficial for UCS because the new representation in general requires a significantly smaller population size than the traditional representation.

3. NLCS is extended to the adaptive version in Chapter 5. The adaptive neural learning classifier system (ANCS) allows the population size to vary depending on the training performance. Moreover, an ensemble framework is proposed to reduce the effect of initial choice of parameters.

   The main findings in this chapter are that ANCS is able to learn as well as NLCS without fixing the population size. By allowing regular insertion/deletion, ANCS takes advantage of the condition part to decompose the learning problem. Visualization showed that ANCS is able to find more useful patterns in the underlying information of the data than NLCS. Moreover the ensemble framework of several ANCSs with different setups produces higher predictive accuracy and also avoids the bias of each setup. The ensemble framework avoids the need of setting up parameters. This is extremely useful when faced with a new problem with little knowledge about it.

   The findings in this chapter solve the third research sub-question of "how to reduce the bias caused by the initial choice of parameters' values?". Overall, the chapter has proposed both the internally and externally adaptive frameworks to control several parameters' value dynamically. The results show that

the adaptive framework is useful in terms of less parameters setup and better accuracy.

4. The distributed framework proposed in Chapter 3 and also the ensemble framework in Chapter 5 are constructed in Chapter 6. ANCS is used in these frameworks as the base learner. DANCS and EANCS are tested against DUCS in distributed data streams. The findings are that both DANCS and EANCS are able to recover as quickly as DUCS in dynamic environments. Moreover, they are also very robust to noise. Testing the three systems on a large data set reveals that DANCS and EANCS are much faster than DUCS because their populations are smaller. The testing time of DANCS and EANCS requires approximately 14% and 20%, respectively, of the time needed in DUCS.

   The results in this chapter answer positively the last research sub-question that the proposed system is appropriate for distributed stream data mining.

In conclusion, experiments and findings have answered all sub-questions derived from the main research question stated in Chapter 1: **Can an evolutionary-based classifier system meet the challenges imposed by stream data mining?** With support from experimental results, the study in this thesis found that DANCS and EANCS are evolutionary techniques which indeed are efficient for distributed and stream data mining.

7.2 Future Research

Through out the development of this research, a number of areas for future work have been identified. Some open research questions have already been discussed through the thesis; this section gives more philosophical future research directions.

The use of NNs would certainly make the learning algorithm more flexible in terms of handling different learning tasks. Beside classification, other tasks such as prediction, function approximation, etc can be easily implemented. It will be interesting to expand ANCS to other data mining tasks.

ANCS was tested on single-step problems. The algorithm can be easily extended to multi-step problems using a similar Q-Learning approach as in XCS for delaying the immediate reward. The error for back propagating in ANCS can be estimated or delayed until the destination is reached. Several forms of delayed reward tasks used in XCSF Lanzi *et al.* (2007) can be applied in ANCS.

The distributed framework in this thesis could be modified to handle heterogeneous distributed environments efficiently. A more sophisticated and complex method could be employed at the server to combine sources that are partitioned vertically.

Experiments in this thesis were limited to simple neural networks so that expressiveness does not degrade dramatically. The visualization tools proposed in Chapter 5 are able to draw a search boundary only for simple neural networks. However, if more complicated visualization tools for the networks' boundary can be implemented, more complicated neural networks can be employed to allow LCS to have more complex behaviours with smaller population sizes.

Appendix A

Supervised Classifier Systems (UCS)

UCS aims to evolve a population of classifiers during training through its learning and searching components. Each classifier consists of a rule and a set of parameters. The rule is made up from two main components: **a condition** (a body of the rule), **an action** (a prediction of the rule). The condition refers to several environmental states, to which the classifier may match. The action is an outcome if the classifier is fired/activated.

The algorithm of UCS is presented in Algorithm 5.

A.1 Parameters

The parameters it uses are listed in Table A.1.

The key parameter is the fitness F, which measures how good the classifier is relative to the rest in the population. Another two important parameters are *numerosity* and *experience*. A classifier of LCS is a macro–classifier, which holds a distinct rule (a unique pair of *Condition:Action*) within the population. The *numerosity* parameter records the number of copies of the classifier in [P]. Whenever a new classifier is introduced, the population is scanned through to check if a copy of

Initialize UCS parameters;
Initialize $[P]$ to empty;
repeat
  Given the training dataset $D = (x_1, y_1), ..., (x_N, y_N)$ from the env;
  for *the training instance* I_trn: $(x_i : y_i)$ **do**
    Form a match set [M] of those classifiers in [P] that match x_i;
    Form the correct set [C] of those classifiers in [M] which predicted y_i correctly;
    if *[C] is empty* **then**
      Create a classifier C_{cover} that matches x_i and assigned y_i to its action ;
      Insert C_{cover} to [P] and [C];
    end
    Update the fitness of those classifiers in [M] by:

$$\begin{aligned} acc &= \frac{number\_of\_correct\_classification}{number\_of\_matches} \\ F &= (acc)^v \end{aligned}$$

    if *The average experiences of classifiers in [C] is higher than* θ_{GA} **then**
      Select and reproduce (mutation and crossover) two classifiers in [C];
      Offspring inherit the *fitness* from the parents;
      Insert offspring in $[P]$;
      while *the current population size is larger than* N **do**
        Remove a classifier from $[P]$ wrt the fitness;
      end
    end
  end
  for *each testing instance* I_{tst} *from* env **do**
    Form a match set $[M]$;
    Select the best prediction from the vote (weighted by fitness) of all classifiers in [M];
    Return the prediction to the environment;
  end
until *the termination conditions are met* ;

Algorithm 5: The simple algorithm of UCS

it already exists. If it does not, the classifier is added to the population; otherwise, the *numerosity* value is incremented by one. The *experience* parameter indicates how often the classifier is chosen for making the prediction; in other words, how general the classifier is.

Table A.1: Parameters of UCS

Parameter	Meaning
N	maximum population size
v	A constant of the fitness function
θ_{GA}	threshold to activate GA
χ	probability of applying crossover
μ	probability of applying mutation
θ_{del}	minimum experience to be considered during deletion
θ_{sub}	minimum experience to be considered for subsumption
r_0	Covering range in interval codification
m_0	Mutation range in interval codification

A.2 The Learning Component

During the learning cycle, the system repeatedly receives an input from the environment. After processing the input, an appropriate outcome is produced and returned to the environment. Feedbacks by the environment is given immediately in terms of a reward (in the case of XCS) or a simple year/no answer (in the case of UCS) reflecting how good the decision was.

A *match set* [M] is formed for each input, containing all classifiers in the population [P] whose *condition* matches the input. Classifiers in [M] will work together to decide on the system's outcome.

In exploitation phase, a *prediction array* [PA] is formed for estimating the value of each possible action in [M] with regards to their fitness. An action having the highest value in [PA] will be selected to be exported to the environment.

The exploration phase, on the other hand, is more complicated. A *correct set* [C] is formed, containing those classifiers in [M] that have the same *action* as the input. If [C] is empty, *covering* is applied, where a classifier that matches the input, is created and assigned the same class with the input.

A.3 The Searching Component

The parameters of classifiers in [M] are updated incrementally. GA is invoked in [C] when the average *experience* of classifiers in [C] is higher than a threshold defined by the user. Two parents are selected from [C] with probability proportional to their fitness. Two offspring are generated by reproducing, crossing–over, and mutating the parents with certain probabilities. Offspring are inserted in [P]. The parents also remain in [P] to compete with the offspring. If the population size exceeds the predefined limit, some inaccurate classifiers are removed from [P].

Appendix B

Data Sets

B.1 Synthetic Data Sets

The multiplexer and checkerboard problems were studied in this research. Results from the multiplexer problem are given in this thesis. Results from the checkerboard problem are given in (Dam *et al.*, 2005a) and (Dam *et al.*, 2005c). The multiplexer problem is one of the most popular testing benchmarks in the literature of learning classifier systems.

A data instance of this problem is a binary string of length $k + 2^k$. The first k bits determine the position of an output bit in the last 2^k bits. The length of an input can be varied by the variable k. For example, the 6-bits multiplexer problem has 6 binary attributes such as 110000, 101010, 010000, etc. The first two attributes decide the position of the output bit such as: 11000→0, 101010→1, 010000→0. The 11-bits multiplexer has 11 binary attributes, where the first three attributes decide the final output. Similarly, the 20-bits multiplexer has 20 binary attributes and the first four attributes are used to calculate the position of the output. The 37-bits multiplexer uses the first five attributes in 37 attributes to decide on the outcome.

In order to make this problem closer to the real world ones, Wilson (2000) extended the multiplexer problem to a continuous domain by introducing a real-threshold to convert a real number to a binary number. For example, assume a real

number is in the range $[0, 1)$ and the real-threshold is 0.5. The real input number r will be converted to 0 in binary if $r < 0.5$, otherwise it is set to 1 in binary. The real multiplexer problem then becomes a traditional binary multiplexer problem.

B.2 Real Data Sets

B.2.1 Small Data sets

This thesis tests the proposed framework on thirteen data sets available from the University of California at Irvine (UCI) repository Blake and Merz (1999) and one artificial data set. All represent classification problems. This set of problems was selected to give us a range of important characteristics, such as (i) number of features: low (up to 10 features) or high (more than 10 features)); (ii) features' representations: real, integer, nominal, or a mixture; (iii) number of outcomes: binary (2 possible outcomes), small (3 or 4 outcomes), or large (5 or more outcomes).

Table B.1 shows the properties of the data sets used in this paper: $Inst$ (the number of instances in the data set), Fs (the number of features), R (the number of real-valued features), I (the number of integer-valued features), N (the number of nominal-valued features), and C (the number of possible outcomes).

The chosen test problems:

- are well known data sets that have been used in many research papers (e.g. breast cancer, diabetes, ionosphere, and iris);
- include some with a large number of features (e.g. heart statlog, lymph and sonar) and some with a small number (e.g. balance scale, iris, and tao);
- include some with a high number of classes (e.g. glass, segment, vowel);
- include different representations (e.g. breast cancer, Australian credit card, lymph, and vowel);
- include non-linear decision boundaries (e.g. Tao).

Table B.1: The properties of testing data sets

Problem	# Inst	# Fs	# R	# I	# N	# C
balance-scale	625	4	4	0	0	3
breast-w	699	9	0	9	0	2
bupa	345	6	6	0	0	2
credit-a	690	15	10	0	5	2
diabetes	768	8	8	0	0	2
glass	214	9	0	0	0	6
heart-statlog	270	13	13	0	0	2
ionosphere	351	34	34	0	0	2
iris	150	4	4	0	0	3
lymph	148	18	3	6	9	4
segment	2310	19	19	0	0	7
sonar	208	60	60	0	0	2
tao	1888	2	2	0	0	2
vowel	990	13	10	0	3	11

By considering those factors, this small set of problems is an appropriate set to study.

- Balance scale: this data set was generated to model psychological experimental results of weight and distance.
- Breast cancer: this data set was obtained from the University of Wisconsin Hospitals, Madison. The problem is to distinguish malignant (cancer) from benign (non-cancer) samples. The data contains 65.5% for benign class and 34.5% for malignant class. Those 16 instances with missing values are removed, left the data set with 683 instances.
- Bupa contains the information of blood tests which were thought to be sensitive to liver disorders that might arise from alcohol consumption.
- The Australian credit card has two classes with distribution 44.5% and 55.5%.
- Diabetes: The data has two classes, positive or negative for diabetes.
- Glass contains information to identify types of glass.
- Heart Statlog is a heart disease database donated by Ross King.

- Ionosphere contains 351 instances for classifying of radar returns from ionosphere.

- Iris: The data contains three classes referring to three types of Iris plants: Iris Setosa, Iris Versicolour, and Iris Virginica. The first class is linearly separable from the other two. The latter are not linearly separable from each other. The data is distributed equally for each class (33.3%).

- Lymph is the medical data in lymphography domain.

- Segment: The data is drawn randomly from a database of 7 outdoor images. The images were hand-segmented to create a classification for each pixel.

- Sonar: This data set has 60 features, representing a potential challenge for LCSs.

- TAO: This data set was previously investigated with LCSs. The data set contains samples from the Tao figure where white areas are assigned class zero and black areas class one. This problem is well known in machine learning, but the particular data set, used in this thesis was first introduced in Llorà and Garrell (2001). Although there are only two features, it is a very hard problem for LCSs (using interval encoding) due to the non-linear decision boundaries.

- Vowel: This data set has 11 classes equally distributed.

B.2.2 Large Data Set

The forest cover type data set of the Roosevelt national forest in northern Colorado, available at (Blake and Merz, 1999), is chosen for testing. According to Blackard (1998), the collected data covers an area of 70 miles northwest of Denver in Colorado (30m x 30m cells obtain from US Forest Service (USFS) Region 2 Resource Information System (RIS)), which has 7 major forest cover types (or seven classes). The data was obtained from the U.S. Geological Survey.

The data set has 581,012 observations, 54 attributes, and no missing values. Each observation is labeled as one of 7 different classes (forest cover types).

Bibliography

Abbass, H. A. (2003). Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'03)*, volume 3, pages 2074–2080. IEEE Press.

Abbass, H. A., Bacardit, J., Butz, M. V., and Llora, X. (2004). *Online Adaptation in Learning Classifier Systems: Stream Data Mining*. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign. IlliGAL Report No. 2004031.

Aggarwal, C. C. (2007). An introduction to data streams. In C. C. Aggarwal, editor, *Data Streams, Models and Algorithms*, Advances in Database Systems, chapter 1, pages 1–7. Springer-Verlag, USA.

Ahluwalia, M. and Bull, L. (2009). A genetic programming-based classifier system. In *Proceedings of the 2009 GECCO conference companion on Genetic and Evolutionary Computation (GECCO'09)*, pages 11–18.

Alpaydin, E. (1993). Multiple networks for function learning. In *Proceedings of the 1993 IEEE International Conference on Neural Networks*, pages 27–32, San Francisco USA. IEEE Press.

Andrews, R., Diederich, J., and Tickle, A. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, **8**(6), 373–389.

Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R., and Taylor, P. (1996). Asset

pricing under endogenous expectation in an artificial stock market. Technical Report 96-12-093, Santa Fe Institute.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *PODS '02: Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, New York, NY, USA. ACM Press.

Bacardit, J. and Butz, M. V. (2007). Data mining in learning classifier systems: Comparing XCS with GAssist. In *Lecture Notes in Computer Science*, volume 4399 of *Learning Classifier Systems*, pages 282–290. Springer-Verlag Berlin / Heidelberg. Revised Selected Papers of the International Workshop on Learning Classifier Systems 2003-2005.

Bacardit, J. and Garrell, J. M. (2007). Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach learning classifier system. In *Lecture Notes in Computer Science*, volume 4399 of *Learning Classifier Systems*, pages 59–79. Springer-Verlag Berlin / Heidelberg. Revised Selected Papers of the International Workshop on Learning Classifier Systems 2003-2005.

Bernadó-Mansilla, E. and Garrell-Guiu, J. M. (2003). Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary Computation*, **11**(3), 209–238.

Bernadó-Mansilla, E., Llorà, X., and Garrell-Guiu, J. M. (2002). Xcs and gale: A comparative study of two learning classifier systems on data mining. In *IWLCS '01: Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*, pages 115–132. Springer-Verlag.

Berthold, M. R. and Hall, L. O. (2003). Visualizing fuzzy points in parallel coordinates. *IEEE Transactions on Fuzzy Systems*, **11**(3), 369–374.

Berthold, M. R. and Holve, R. (2000). Visualizing high dimensional fuzzy rules. In *Proceedings of the 19th International Conference of the North American (NAFIPS)*, pages 64–68. IEEE Press.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Blackard, J. A. (1998). *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. Ph.D. thesis, Department of Forest Sciences. Colorado State University, Fort Collins, Colorado.

Blake, C. and Merz, C. (1999). UCI repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/mlrepository.html. *University of California, Irvine, Dept. of Information and Computer Sciences*.

Bonarini, A. (2000). An introduction to learning fuzzy classifier systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications*, pages 83–106. Springer-Verlag, London, UK.

Bottou, L., Soulie, F. F., Blanchet, P., and Lienard, J. S. (1990). Speaker-independent isolated digit recognition: multilayerer perceptrons vs. dynamic time warping. *Neural Networks*, **3**(4), 453–465.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (1999). Pasting small votes for classification in large databases and on-line. *Machine Learning*, **36**(1–2), 85–103.

Brown, G. (2004). *Diversity in Neural Network Ensembles*. Ph.D. thesis, School of Computer Science, The University of Birmingham.

Brown, G., Wyatt, J., Harris, R., and Yao, X. (2005). Diversity creation methods: a survey and categorisation. *Information Fusion*, **6**(1), 5–20.

Browne, W. (1999). *The Development of an Industrial Learning Classifier System for Application to a Steel Hot Strip Mill*. Ph.D. thesis, University of Wales, Cardiff.

Browne, W. and Scott, D. (2005). An abstraction agorithm for genetics-based reinforcement learning. In *Proceedings of the 2005 Conference on Genetic and*

Evolutionary Computation (GECCO '05), pages 1875–1882, New York, NY, USA. ACM Press.

Browne, W. N. and Ioannides, C. (2007). Investigating scaling of an abstracted LCS utilising ternary and s-expression alphabets. In *Proceedings of the 2007 GECCO conference companion on Genetic and Evolutionary Computation (GECCO'07)*, pages 2759–2764, New York, NY, USA. ACM Press.

Bull, L. (2002). On using constructivism in neural classifier systems. In *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 558–567, London, UK. Springer-Verlag.

Bull, L. and Hurst, J. (2000). Self-adaptive mutation in ZCS controllers. In *Proceedings of Real-World Applications of Evolutionary Computing, EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoROB, and EvoFlight*, pages 339–346, London, UK. Springer-Verlag.

Bull, L. and Hurst, J. (2002). ZCS redux. *Evolutionary Computation*, **10**(2), 185–205.

Bull, L. and Kovacs, T. (2005). Foundations of learning classifier systems: An introduction. In L. Bull and T. Kovacs, editors, *Foundations of Learning Classifier Systems*, pages 1–17. Springer-Verlag.

Bull, L. and O'Hara, T. (2002). Accuracy-based neuro and neuro-fuzzy classifier systems. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Bull, L., Sha'Aban, J., Tomlinson, A., Addison, P., and Heydecker, B. (2004). Towards distributed adaptive control for road traffic junction signals using learning classifier systems. In L. Bull, editor, *Applications of Learning Classifier Systems*, pages 276–299. Springer-Verlag.

Bull, L., Studley, M., Bagnall, T., and Whittley, I. (2005). On the use of rule-sharing in learning classifier system ensembles. In *Proceedings of IEEE Congress*

on Evolutionary Computation (CEC'05), pages 612–617, Edinburgh, Scotland. IEEE.

Bull, L., Lanzi, P. L., and O'Hara, T. (2007a). Anticipation mappings for learning classifier systems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07)*, pages 2133–2140, Singapore. IEEE.

Bull, L., Studley, M., Bagnall, A., and Whittley, I. (2007b). Learning classifier system ensembles with rule-sharing. *IEEE Transactions on Evolutionary Computation*, **11**(4), 496–502.

Butz, M. (2006). *Rule-Based Evolutionary Online Learning Systems, A Principled Approach to LCS Analysis and Design*, volume 191 of *Studies in Fuzziness and Soft Computing*. Springer-Verlag.

Butz, M., Lanzi, P., and Goldberg, D. (2007). Effect of pure error-based fitness in XCS. In *Learning Classifier Systems: International Workshop (IWLCS'05)*, pages 104–114. Springer-Verlag Berlin Heidelberg.

Butz, M. V. (2004). *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Butz, M. V. (2005). Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 1835–1842, New York, NY, USA. ACM Press.

Butz, M. V. and Pelikan, M. (2006). Studying XCS/BOA learning in boolean functions: structure encoding and random boolean functions. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO'06)*, pages 1449–1456, New York, NY, USA. ACM Press.

Butz, M. V. and Wilson, S. W. (2001). An algorithmic description of XCS. In *Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems (IWLCS'00)*, pages 253–272. Springer-Verlag.

Butz, M. V., Goldberg, D. E., and Tharakunnel, K. (2003a). Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, **11**(3), 239–277.

Butz, M. V., Kovacs, T., Lanze, P. L., and Wilson, S. W. (2003b). Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, **7**(6), 28–46.

Butz, M. V., Lanzi, P. L., and Wilson, S. W. (2006). Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 1457–1464, New York, NY, USA. ACM Press.

Cantu-Paz, E. and Kamath, C. (2002). On the use of evolutionary algorithms in data mining. In H. A. Abbass, R. A. Sarker, and C. S. Newton, editors, *Data Mining: A Heuristic Approach*, chapter 3, pages 48–71. Idea Group Publishing.

Chan, P. K. and Stolfo, S. J. (1993). Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Knowledge Discovery in Databases*, pages 227–240, Washington, DC.

Chen, K.-Y., Dam, H. H., Lindsay, P., and Abbass, H. A. (2007). Biasing XCS with domain knowledge for planning flight trajectories in a moving sector free flight environment. In *Proceedings of the 1st IEEE Symposium on Artificial Life*, Piscataway, NJ. IEEE Press.

Cliff, D. and Ross, S. (1994). Adding temporary memory to ZCS. *Adaptive Behavior*, **3**(2), 101–150.

Colombetti, M., Dorigo, M., and Borghi, G. (1996). Robot shaping: The HAMSTER Experiment. In *Proceedings of ISRAM'96, Sixth International Symposium on Robotics and Manufacturing* , Montpellier, France.

Dam, H., Lokan, C., and Abbass, H. (2007). Evolutionary online data mining: An investigation in a dynamic environment. In Y.-S. O. Shengxiang Yang and Y. Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*,

volume 51 of *Studies in Computational Intelligence*, pages 153–178. Springer-Verlag Berlin / Heidelberg.

Dam, H. H., Abbass, H. A., and Lokan, C. (2005a). Be real! XCS with continuous valued inputs. In *Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation, the Eighth International Workshop on Learning Classifier Systems (IWLCS'05)*, pages 85–87, New York, NY, USA. ACM Press.

Dam, H. H., Abbass, H. A., and Lokan, C. (2005b). DXCS: an XCS system for distributed data mining. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1883–1890, New York, NY, USA. ACM Press.

Dam, H. H., Abbass, H. A., and Lokan, C. (2005c). Investigation on DXCS: An XCS system for distribution data mining, with continuous-valued inputs in static and dynamic environments. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'05)*, volume 1, pages 618–625. IEEE.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London.

De Jong, K. A., Spears, W. M., and Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, **13**(2-3), 161–188.

Delany, S., Cunningham, P., Tsymbal, A., and Coyle, L. (2005). A case-based technique for tracking concept drift in spam filtering. *Journal of Knowledge Based Systems*, **18**(4–5), 187–195.

Dixon, P. W., Corne, D., and Oates, M. J. (2001). A preliminary investigation of modified XCS as a generic data mining tool. In *Advances in Learning Classifier Systems: 4th International Workshop (IWLCS'01)*, pages 133–150. Springer-Verlag Berlin Heidelberg.

Dixon, P. W., Corne, D. W., and Oates, M. J. (2003). A ruleset reduction algorithm for the XCS learning classifier system. In P. L. Lanzi, W. Stolzmann, and

S. W. Wilson, editors, *Learning Classifier Systems: Fifth International Workshop (IWLCS'02) (LNAI 2661)*, pages 20–29. Springer-Verlag Berlin / Heidelberg.

Dorigo, M. (1993). Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation*, **1**(2), 151–164.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. *Communication of the ACM*, **39**(11), 27–34.

Ferrandi, F., Lanzi, P. L., and Sciuto, D. (2003). Mining Interesting Patterns from Hardware-Software Codesign Data with the Learning Classifier System XCS. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'03)*, pages 1486–1492. IEEE.

Fogel, D. B. (1992). *Evolving artificial intelligence*. Ph.D. thesis, University of California at San Diego, La Jolla, CA, USA.

Frawley, W. J., Piatetsky-Shapiro, G., and Matheus, C. J. (1992). Knowledge discovery in databases: an overview. *AI Magazine*, **13**(3), 57–70.

Freitas, A. A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing: Theory and Applications*, pages 819–845. Springer-Verlag New York.

Frey, P. W. and Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, **6**(2), 161–182.

Fu, C. and Davis, L. (2002). A modified classifier system compaction algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pages 920–925, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2007). A survey of classification methods in data streams. In C. C. Aggarwal, editor, *Data Streams,*

Models and Algorithms, Advances in Database Systems, chapter 3, pages 39–53. Springer-Verlag, USA.

Gao, Y., Huang, J. Z., Rong, H., and Gu, D. (2005). Learning classifier system ensemble for data mining. In *Proceedings of the 2005 Workshop on Genetic and Evolutionary Computation (GECCO'05)*, pages 63–66, New York, NY, USA. ACM Press.

Gao, Y., Wu, L., and Huang, J. Z. (2006). Ensemble learning classifier system and compact ruleset. In *Proceedings of the 6th International Conference on Simulated Evolution and Learning (SEAL'06)(LNCS 4247)*, pages 42–49. Springer-Verlag Berlin / Heidelberg.

Garner, S. R. (1995). Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, pages 57–64.

Gershoff, M. and Schulenburg, S. (2007). Collective behavior based hierarchical XCS. In *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation (GECCO'07)*, pages 2695–2700, New York, NY, USA. ACM Press.

Giannella, C., Bhargava, R., and Kargupta, H. (2004). Multi-agent systems and distributed data mining. In *Cooperative Information Agents VIII: 8th International Workshop, CIA 2004 (LNCS 3191)*, pages 1–15. Springer-Verlag Berlin / Heidelberg.

Goldberg, D. E. (1983). *Computer-Aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning*. Ph.D. thesis, The University of Michigan, USA.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Company, Boston, MA, USA.

Goldberg, D. E. (2002). *The Design of Innovation, lessons from and for competent genetic algorithms*. Kluwer Academic Publishers, Norwell, Massachusetts, USA.

Guerra-Salcedo, C. and Whitley, D. (1999). Genetic approach to feature selection for ensemble creation. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 236–243, Orlando, Florida, USA. Morgan Kaufmann.

Guo, Y. and Sutiwaraphun, J. (2000). Distributed classification with knowledge probing. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 115–132. The MIT Press, Cambridge, MA, USA.

Guo, Y., Rueger, S., Sutiwaraphun, J., and Forbes-Millott, J. (1997). Meta-learning for parallel data mining. In *Proceedings of the Seventh Parallel Computing Worksop (PCW'97)*, Canberra. Australian National University.

Han, J. (2005). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Hand, D., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining*. MIT Press.

Hansen, L. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(10), 993–1001.

Harries, M. B., Sammut, C., and Horn, K. (1998). Extracting hidden context. *Machine Learning*, **32**(2), 101–126.

Hartley, A. R. (1999). Accuracy-based fitness allows similar performance to humans in static and dynamic environments. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 266–273, Orlando, Florida, USA. Morgan Kaufmann.

Hashem, S. (1997). Optimal linear combinations of neural networks. *Neural Networks*, **10**(4), 599–614.

Haykin, S. (1999). *Neural networks, a comprehensive foundation*. Prentice-Hall, Upper Saddle River, New Jersey, USA, 2 edition.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor. Republished by the MIT press, 1992.

Holland, J. H. (1985). Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7, Mahwah, NJ, USA. Lawrence Erlbaum Associates.

Holland, J. H. (1986). Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine Learning, an Artificial Intelligence Approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann.

Holland, J. H. and Reitman, J. S. (1977). Cognitive systems based on adaptive algorithms. *SIGART Bulletin*, **63**, 49–49.

Holmes, J. H. (1996). *Evolution-assisted discovery of sentinel features in epidemiologic surveillance*. Ph.D. thesis, Drexel University, Philadelphia, PA, USA.

Horn, J., Goldberg, D. E., and Deb, K. (1994). Implicit Niching in a Learning Classifier System: Nature's Way. *Evolutionary Computation*, **2**(1), 37–66.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**(5), 359–366.

Hurst, J. and Bull, L. (2001). A self-adaptive classifier system. In *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 70–79. Springer-Verlag Berlin / Heidelberg.

Hurst, J. and Bull, L. (2002). A self-adaptive XCS. In *IWLCS '01: Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*, pages 57–73. Springer-Verlag Berlin / Heidelberg.

Hurst, J. and Bull, L. (2004). A self-adaptive neural learning classifier system with constructivism for mobile robot control. In *Parallel Problem Solving from Nature - PPSN VIII*, pages 942–951. Springer-Verlag Berlin / Heidelberg.

Islam, M. M., Yao, X., and Murase, K. (2003). A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, **14**(4), 820–834.

Jones, C., Hall, J., and Hale, J. (2000). Secure distributed database mining: Principles of design. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 277–294. The MIT Press USA.

Jong, K., Mary, J., Cornuéjols, A., Marchiori, E., and Sebag, M. (2004). Ensemble feature ranking. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '04)*, pages 267–278, New York, NY, USA. Springer-Verlag.

Kargupta, H. and Chan, P., editors (2000). *Advances in Distributed and Parallel Knowledge Discovery*. The MIT Press USA.

Kharbat, F., Bull, L., and Odeh, M. (2007a). Mining breast cancer data with XCS. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO'07)*, pages 2066–2073, New York, NY, USA. ACM Press.

Kharbat, F., Odeh, M., and Bull, L. (2007b). New approach for extracting knowledge from the XCS learning classifier system. *International Journal of Hybrid Intelligent Systems*, **4**(2), 49–62.

Klinkenberg, R. and Joachims, T. (2000). Detecting concept drift with support vector machines. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 487–494, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Kolter, J. and Maloof, M. (2003). Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 123–130, Los Alamitos, CA. IEEE Press.

Kovacs, T. (1997). XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag.

Kovacs, T. (1999). Deletion schemes for classifier systems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 329–336, Orlando, Florida, USA. Morgan Kaufmann.

Kovacs, T. (2000). Strength or accuracy? Fitness calculation in learning classifier systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications*, volume 1813, pages 143–160. Springer-Verlag, London, UK.

Kovacs, T. (2002). What should a classifier system learn and how should we measure it? *Journal of Soft Computing*, **6**(3–4), 171–182.

Krogh, A. and Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. Cambridge, MA. MIT Press.

Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, **51**(2), 181–207.

Lanzi, P. L. (2001). Mining interesting knowledge from data with the XCS classifier system. In L. Spector, E. D. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 958–965, San Francisco, CA 94104, USA. Morgan Kaufmann.

Lanzi, P. L. and Colombetti, M. (1999). An extension to the XCS classifier system for stochastic environments. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon,

V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 353–360, Orlando, Florida, USA. Morgan Kaufmann.

Lanzi, P. L. and Loiacono, D. (2007). Classifier systems that compute action mappings. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO'07:)*, pages 1822–1829, New York, NY, USA. ACM Press.

Lanzi, P. L. and Perrucci, A. (1999a). Extending the representation of classifier conditions. part I: From from binary to messy coding. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 1, pages 337–344. Morgan Kaufmann.

Lanzi, P. L. and Perrucci, A. (1999b). Extending the representation of classifier conditions. part II: From messy coding to S-expressions. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 1, pages 345–352. Morgan Kaufmann.

Lanzi, P. L. and Wilson, S. W. (2006). Using convex hulls to represent classifier conditions. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 1481–1488, New York, NY, USA. ACM Press.

Lanzi, P. L., Loiacono, D., Wilson, S. W., and Goldberg, D. E. (2007). Generalization in the XCSF classifier system: Analysis, improvement, and extension. *Evolutionary Computation*, **15**(2), 133–168.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1**(4), 541–551.

Lincoln, W. P. and Skrzypek, J. (1990). Synergy of clustering multiple back propagation networks. In D. S. Touretzky, editor, *Advances in neural information*

processing systems 2, pages 650–659. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Liu, Y. (1999). *Negative Correlation Learning and Evolutionary Design of Neural Network Ensembles*. Ph.D. thesis, Australian Defence Force Academy at the University of New South Wales, Northcott Drive, Canberra ACT 2600 Australia.

Liu, Y. and Yao, X. (1997). Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems*, **4**(3/4), 176–185.

Liu, Y., Yao, X., and Higuchi, T. (2000). Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, **4**(4), 380–387.

Llorà, X. (2002). *Genetics-based machine learning using fine-gained parallelism for data mining*. Ph.D. thesis, Enginyeria i Arquitectura La Salle. Ramon Llull University, Barcelona, Catalonia, European Union.

Llorà, X. and Garrell, J. M. (2001). Evolving partially-defined instances with evolutionary algorithms. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, pages 337–344, San Francisco, CA, USA. Morgan Kauffmann.

Llorà, X. and Sastry, K. (2006). Fast rule matching for learning classifier systems via vector instructions. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 1513–1520, New York, NY, USA. ACM Press.

Llorà, X., Sastry, K., and Goldberg, D. E. (2005a). Binary rule encoding schemes: a study using the compact classifier system. In *Proceedings of the 2005 Workshop on Genetic and Evolutionary Computation (GECCO'05)*, pages 88–89, New York, NY, USA. ACM Press.

Llorà, X., Sastry, K., and Goldberg, D. (2005b). The Compact Classifier System:

Scalability Analysis and First Results. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'05)*, pages 596–603. IEEE.

Llorà, X., Priya, A., and Bhargava, R. (2006). *Observer-invariant histopathology using genetics-based machine learning*. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (IlliGAL TR No 200627).

Loiacono, D., Marelli, A., and Lanzi, P. L. (2007). Support vector machines for computing action mappings in learning classifier systems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07)*, pages 2141–2148. IEEE.

Maclin, R. (1998). Boosting classifiers regionally. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 700–705, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Maclin, R. and Opitz, D. (1997). An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551. Cambridge, MA. AAAI Press/MIT Press.

Mani, G. (1991). Lowering variance of decisions by using artificial neural network portfolios. *Neural Computation*, **3**, 484–486.

McKay, R. and Abbass, H. (2001). Anti-correlation: A diversity promoting mechanism in ensemble learning. *The Australian Journal of Intelligent Information Processing Systems*, **7**(3/4), 139–149.

Mellor, D. (2005). A first order logic classifier system. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 1819–1826, New York, NY, USA. ACM Press.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York, USA.

Naisbitt, J. (1988). *Megatrends : Ten New Directions Transforming Our Lives*. Warner Books.

Nguyen, M., Abbass, H., and McKay, R. (2004). Diversity and neuro-ensemble. In A. Ghosh, L. C. Jain, A. Ghosh, and L. C. Jain, editors, *Evolutionary Computing in Data Mining*, volume 163 of *Studies in Fuzziness and Soft Computing*, chapter 7. Springer-Verlag Berlin / Heidelberg.

Nguyen, M. H. (2006). *Cooperative Coevolutionary Mixture of Experts. A neural ensemple approach for automatic decomposition of classification problems*. Ph.D. thesis, Australian Defence Force Acamedy at the University of New South Wales, Northcott Drive, Canberra ACT 2600 Australia.

Nilsson, N. J. (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, New York, USA.

O'Hara, T. and Bull, L. (2005). Building anticipations in an accuracy-based learning classifier system by use of an artificial neural network. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'05)*, pages 2046–2052. IEEE Press.

O'Hara, T. and Bull, L. (2007). Backpropagation in accuracy-based neural learning classifier systems. In T. Kovacs, X. Llora, K. Takadama, P.-L. Lanzi, W. Stolzmann, and S. Wilson, editors, *Advances at the Frontier of Learning Classifier Systems*, pages 26–40. Springer-Verlag.

Opitz, D. W. and Shavlik, J. W. (1996a). Actively searching for an effective neural-network ensemble. *Connection Science*, **8**(3/4), 337–353.

Opitz, D. W. and Shavlik, J. W. (1996b). Generating accurate and diverse members of a neural-network ensemble. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 535–541. The MIT Press.

Orriols-Puig, A. and Bernadó-Mansilla, E. (2006). A further look at UCS classifier system. In *Proceedings of the 9th International Workshop on Learning Classifier Systems (IWLCS'06)*. ACM Press.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, **6**(3), 21–45.

Prodromidis, A. L., Chan, P. K., and Stolfo, S. J. (2000). Meta-learning in distributed data mining systems: Issues and approaches. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 81–114. The MIT Press USA.

Provost, F. (2000). Distributed data mining: Scaling up and beyond. In H. Kargupta and P. Chan, editors, *Advances in Distributed and Parallel Knowledge Discovery*, pages 3–29. The MIT Press USA.

Quartz, S. and Sejnowski, T. (1997). The neural basis of cognitive development: A constructivist manifesto. *Brain and Behavioral Sciences*, **20**(4), 537–596.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Rosen., B. (1996). Ensemble learning using decorrelated neural networks. *Special Issue of Connection Science on Combining Artificial Neural: Ensemble Approaches*, **8**(3/4), 373–384.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 318–362. MIT Press, Cambridge, MA, USA.

Saxon, S. and Barry, A. (2000). XCS and the Monk's problems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications (LNCS 1813)*, pages 223–242. Springer-Verlag, London, UK.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, **5**(2), 197–227.

Schlimmer, J. C. and Fisher, D. H. (1986). A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501, Philadelpha, PA. Morgan Kaufmann.

Schulenburg, S. and Ross, P. (2001). Strength and money: An lcs approach to increasing returns. In *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 114–137, London, UK. Springer-Verlag.

Shafi, K., Abbass, H., and Zhu, W. (2006). The role of early stopping and population size in XCS for intrusion detection. In *6th International Conference on Simulated Evolution and Learning (SEAL'06)*, pages 50–57, Hefei, China.

Shafi, K., Kovacs, T., Abbass, H. A., and Zhu, W. (2007). Intrusion detection with evolutionary learning classifier system. *Natural Computing an International Journal, Special Issue on Learning Classifier Systems*.

Sharkey, A. (1996). On combining artificial neural nets. *Connection Science*, **8**, 299–313.

Skurichina, M., Kuncheva, L., and Duin, R. (2002). Bagging and boosting for the nearest mean classifier: Effects of sample size on diversity and accuracy. In *International Workshop on Multiple Classifier Systems*, pages 62–71. Springer-Verlag.

Smith, R. E. and Jiang, M. K. (2007a). A learning classifier system with mutual-information-based fitness. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07)*, pages 2173–2180. IEEE Press.

Smith, R. E. and Jiang, M. K. (2007b). MILCS: a mutual information learning classifier system. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, pages 1877–1877, New York, NY, USA. ACM Press.

Smith, S. F. (1980). *A Learning System Based on Genetic Adaptive Algorithms*. Ph.D. thesis, University of Pittsburgh.

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

Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, New York, NY, USA. ACM Press.

Towell, G. and Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, **13**(1), 71–101.

Tsumoto, S. and Hirano, S. (2003). Visualization of rule's similarity using multi-dimensional scaling. In *ICDM '03: Proceedings of the 3th IEEE International Conference on Data Mining*, pages 339–346. IEEE Press.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 226–235, New York, NY, USA. ACM Press.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, **23**(1), 69–101.

Wilson, S. W. (1985). Knowledge growth in an artificial animal. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 16–23, Mahwah, NJ, USA. Lawrence Erlbaum Associates.

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), 149–175.

Wilson, S. W. (1998). Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Proceedings of the Third Annual Conference on Genetic Programming*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann.

Wilson, S. W. (2000). Get real! XCS with continuous-valued inputs. In P. Lanzi, W. Stolzmann, and S. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications (LNAI'1813)*, pages 209–219, Berlin. Springer-Verlag.

Wilson, S. W. (2001a). Compact rulesets from XCSI. In *IWLCS '01: Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*, pages 197–210, London, UK. Springer-Verlag.

Wilson, S. W. (2001b). Function approximation with a classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 974–981, San Francisco, California, USA. Morgan Kaufmann.

Wilson, S. W. (2001c). Mining oblique data with XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems (IWLCS-2000), Lecture Notes in Artificial Intelligence*, pages 158–176, London, UK. Springer-Verlag.

Wilson, S. W. (2002). Classifiers that approximate functions. *Natural Computing*, **1**(2–3), 211–233.

Wilson, S. W. (2004). Classifier systems for continuous payoff environments. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'04)(LNCS 3103)*, pages 824–835. Springer Berlin / Heidelberg.

Wilson, S. W. and Goldberg, D. E. (1989). A critical review of classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255, San Francisco, CA, USA. Morgan Kaufmann.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco, CA USA, second edition.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, **5**(2), 241–259.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for searchs. *IEEE Transactions on Evolutionary Computation*.

Wyatt, D., Bull, L., and Parmee, I. (2004). Building compact rulesets for describing continuous-valued problem spaces using a learning classifier system. In I. Parmee, editor, *Adaptive Computing in Design and Manufacture VI*, pages 235–248. Springer-Verlag.

Zhang, G. P. (2000). Neural networks for classification: A survey. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, **30**(4), 451–462.