COST-SENSITIVE XCS AND SAMPLING ENSEMBLER XCS FOR SOLVING IMBALANCE PROBLEM

NGUYEN HUY THACH

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
FACULTY OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2008

KMITL-2008-EN-M-070-154

หัวข้อวิทยานิพนธ์	การแก้ปัญหาความไม่สมดุลย์ของกลุ่มข้อมูลโดยคอสเซ็นซิทีฟเอ็กซ์ซีเอสและเซมปลิงเอนเซมเบิลเอ็กซ์ซีเอส
นักศึกษา	Nguyen Huy Thach
รหัสนักศึกษา	49060654
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2551
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.ดร. บุญวัฒน์ อัตชู
อาจารย์ผู้ควบคุมวิทยานิพนธ์ร่วม	รศ.ดร. เอื้อน ปิ่นเงิน

บทคัดย่อ

ปัญหาความไม่สมดุลของกลุ่มข้อมูลเป็นปัญหาที่สำคัญในด้านการเรียนรู้ของจักรกลและการทำเหมืองข้อมูล โดยทั่วไประบบเรียนรู้จะมีแนวโน้มในการเรียนรู้โอนเอียงไปทางกลุ่มข้อมูลที่มีจำนวนสมาชิกมากซึ่งจะทำให้การเรียนรู้ของกลุ่มข้อมูลที่มีสมาชิกน้อยนั้นไม่ค่อยมีประสิทธิภาพ

วิทยานิพนธ์ฉบับนี้วิเคราะห์ผลกระทบจากปัญหาความไม่สมดุลของกลุ่มข้อมูลที่มีต่อระบบ XCS ซึ่งเป็นระบบการเรียนรู้ตัวจำแนกประเภทแบบอาศัยความถูกต้องประเภทหนึ่งที่ประสบความเร็จในการใช้งานเป็นอย่างมาก (accuracy-based learning classifier system) โดยได้นำเสนอการแก้ปัญหาสองวิธีคือ วิธีแรก Cost-sensitive XCS และวิธีที่สองคือ Sampling Ensemble XCS ในวิธีแรก Cost-sensitive XCS ค่าตอบแทน (Reward) ที่ได้ในแต่ละกลุ่มจะแตกต่างกัน โดยค่าตอบแทนที่ได้ของกลุ่มที่มีข้อมูลที่มีสมาชิกน้อยจะให้มากกว่าค่าตอบแทนของกลุ่มอื่น และนำเสนอแนวทางการคำนวณค่าตอบแทนแบบ online โดยหาจากข้อมูลที่ถูกรวบรวมในระหว่างเรียนรู้ วิธีที่สอง Sampling Ensemble XCS เป็นการรวมเทคนิคการสุ่มตัวอย่าง (Sampling) และการใช้ตัวจำแนกประเภทหลายตัวร่วมกัน (Ensemble) โดยเทคนิคการสุ่มตัวอย่างจะทำการปรับปรุงการกระจายของข้อมูลที่ใช้ในการเรียนรู้ใหม่และอาศัยเทคนิคการใช้ตัวจำแนกประเภทหลายตัวร่วมกัน สร้างเซตของตัวจำแนกประเภทขึ้นมาเพื่อทำการจำแนกประเภทของข้อมูลโดยคำตอบที่ได้จะได้จากการโหวต (แบบมีน้ำหนัก) จากการทดลองทั้งข้อมูลที่สร้างขึ้นมาเพื่อวิเคราะห์และข้อมูลจริงมาตรฐาน 13 ชนิดจาก UCI ผลการทดลองและการศึกษาเปรียบเทียบแสดงให้เห็นว่า Cost-sensitive XCS และ Sampling Ensemble XCS เป็นเครื่องมือที่มีประสิทธิภาพสำหรับนำมาใช้ในการแก้ปัญหาความไม่สมดุลของกลุ่มข้อมูล

Thesis Title:	Cost-sensitive XCS and Sampling Ensemble XCS for Solving Imbalance Problem.
Student	Nguyen Huy Thach
Student ID	49060654
Degree	Master's Degree
Program	M.Eng. Computer Engineering
Year	2008
Thesis Advisor	Assoc. Prof. Dr. Boonwat Attachoo
Thesis Co-Advisor	Assoc. Prof. Dr. Ouen Pinngern

ABSTRACT

The class imbalance problem has been recognized as a crucial problem in machine learning and data mining. Learning systems tend to be biased towards the majority class and thus have poor performance on the minority class instances. This thesis analyses the imbalance problem in accuracy-based classifier system XCS which is a most successful kind of Learning Classifier System. In particularly, we propose two methods Cost-sensitive XCS and Sampling Ensemble XCS to solve the problem. In Cost-sensitive XCS, the classification reward for each class is set differently. The reward value of correctly identifying the positive (rare) class outweighs the value of correctly identifying the common class. We provide guidelines to set reward value based on the dataset imbalance ratio and a method calculating reward online based on the information collected by XCS during training is also proposed. In the second approach, Sampling Ensemble XCS is a combination of sampling technique and ensemble learning where sampling technique modifies the distribution of the training data and ensemble learning constructs a set of classifiers and then classifies new data points by taking a (weighted) vote of their predictions. An artificial problem and thirteen UCI imbalance datasets are used to verify the feasibility of our proposals. The experimental results and comparative studies indicate that Cost-sensitive XCS and Sampling Ensemble XCS are effective tools in solving imbalance data problem.

Acknowledgments

This thesis could not have been finished without the help and contributions of several important people. First and foremost, I would like to thank my advisor, Assoc. Prof. Dr. Ouen Pinngern, thank you for your trust in me. Thank you for your patience in making me understands, in teaching me how to do research, how not to get caught up in details and how to improve my writing. I learned a lot from you. Thank you for your time, scientific and personal guidance, thank you for supporting me as a research assistant and introducing me to the scientific community. Thank you for your generosity, enthusiasm and patience, and for being a great teacher. Thank you for your careful reading of my thesis and for allowing me to do pure research during the last two years. I will always be proud to have been your student. And thank you for encouraging me in several key moments.

To my supervisors, Assoc. Prof. Dr. Boonwat Attachoo, Engineering Faculty, King Mongkut's Institute of Technology Ladkrabang, Prof. Hidekazu Tsuji, Department of Information Media Technology, School of Information Science and Technology, Tokai University and Dr. Cao Tuan Dung, Faculty of Information Technology, Hanoi University of Technology, who were always available for additional advise, support, and help in my thesis. Thank you for your constructive discussions and for your important support throughout this work.

I am also very grateful to my thesis committee members. Thank you for all the useful comments, suggestions, and additional work. I am also grateful to many other faculty members of King Mongkut's Institute of Technology Ladkrabang including Assoc. Prof. Somsak Mitatha, Asst. Prof. Kritawan Siriboon, Assoc. Prof. Kanchit Maitree, Assoc. Prof. Dr. Boontee Kruatrachue and many others.

Thank you to the people in the Information System laboratory: Phaitoon Srinil, Kreangsak Taemee, Porntep Rojanavasu, Sombut Foitong and Sornchai Udomthanapong. I was blessed with a research environment of really smart people that were always available not only for advice and discussions but also knew when to talk about other issues in life. Warm thanks to Porntep Rojanavasu for your kindness, generosity in helping me with programming questions. Thank you to Kreangsak Taemee for making to me aware of Wilson's XCS. I must also thank many other graduate students in our Computer Engineering Department for their help during the time we studied together.

I am also grateful to the people in Faculty of Information Technology, Hanoi University of Technology for their essential help, in particular Assoc. Prof. Dr. Nguyen Ngoc Binh, Assoc. Prof. Dr. Huynh Quyet Thang, Ms. Nguyen Thanh Phuong and many others.

Specifically, I would like to thank my family for all of their support throughout the years and without whom I would literally not be here today. In particular, I thank my mom, my dad and my younger brother for all of their sage advice and support to help me keep the life balance between Thailand and Vietnam.

Finally, I would like to acknowledge all the financial support I received during my studies from the Japan International Cooperation Agency (JICA) under the ASEAN University Network/ Southeast Asia Engineering Education Development network (AUN/SEED-Net) Project. Thanks also for the additional funds from other resources available to the Information System laboratory that supplied me with computer power and an optimal working environment.

Thach Nguyen Huy

August 08, 2008

Bangkok, Thailand

Contents

Contents (cont.)

Contents (cont.)

Contents (cont.)

Contents (cont.)

List of Tables

List of Tables (cont.)

List of Figures

Chapter 1

Introduction

1.1 Background

The growth in the amount of data collected and generated has exploded in recent times with the widespread automation of various day-to-day activities, advances in high-level scientific and engineering research and the development of efficient data collection tools. This has given rise to the need for automatically analyzing the data in order to extract knowledge from it, thereby making the data potentially more useful. Data mining, or knowledge discovery in databases (KDD), is a process satisfied that requirement. It is an interdisciplinary area that integrates techniques from several fields including machine learning, statistics, and database systems, for the analysis of large volumes of data as described in Figure 1.1 [3]. In this introductory chapter, we present an overview of Data Mining, Artificial Intelligence and outline the key topics to be covered this study.

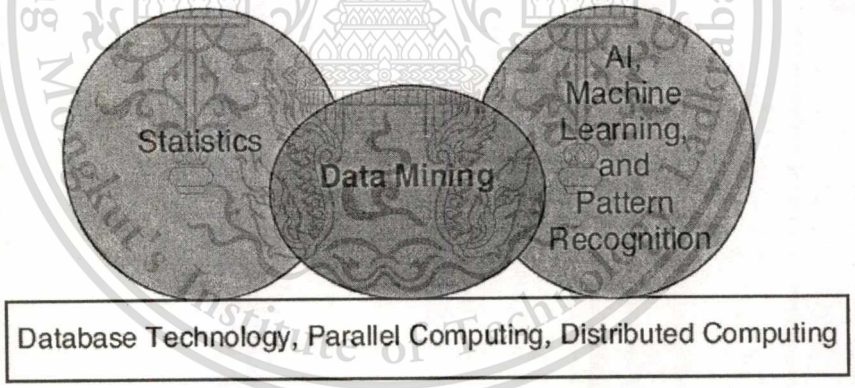


Figure 1.1 Data mining as a confluence of many disciplines

There has been a rapid growth in the successful use of intelligent systems in diverse areas such as science, medicine and commerce [3]. The main contributing factor influencing the development of intelligent systems in data mining has been the demand for a more powerful yet flexible and robust technique. This is in order to cope with the extensive amount of data and knowledge stored in databases, and more importantly, the complexity of interpretation. The successful artificial intelligence tools for data mining are: Rule-based Systems, Artificial Neural Networks, Decision Trees, Fuzzy Systems, Genetic Algorithms and etc. In this study, we would like to introduce

one kind of rule-based system, also successful in data mining that is Learning Classifier System (LCS).

Learning Classifier System is introduced by Holland [4], which is a combination of Reinforcement Learning and Genetic Algorithm. LCS represents a technique by which various characteristics of a given problem space may be deduced and presented to the user in a readable format, that is, the Learning Classifier System could be used within a Knowledge Discovery system. There have been several studies published that have demonstrated Learning Classifier System's and, in particular, the XCS classifier system's capabilities for data-mining through rule induction. Bernado et al [25] describe an experimental comparison of XCS [8] with seven other learning schemes using fifteen UCI repository datasets [74]. The XCS classifier system is shown to be highly competitive when compared with the other learning schemes. In addition, Wilson [17] demonstrated the capabilities of a new encoding when used to induce rules describing the Wisconsin Breast Cancer dataset, where the XCS classifier system improved on the best known performance for that dataset. The XCS classifier system would seem an appropriate technique by which to support decision-makers in the identification of interesting regions of a solution space.

Recently, imbalance problem has been recognized as a crucial problem in data mining and machine learning, where the number of instances in one class is much less than that of the other classes. It occurs frequently in many real-world applications such as fraud detection [40], product inspection, text classification [44], and medical diagnosis [39], [42]. When a classifier is learned from an imbalanced or skewed data set, it will cause bias and, then, the tendency is that the classifier will produce high predictive accuracy over the majority class, but will predict poorly over the minority class [45]. Furthermore, the examples in the minority class can be treated as noise and are ignored completely by the classifier.

As described, XCS is an appropriate technique of data mining and quite robust to imbalance problem [28]. However, there remain various aspects of XCS that can be further improved to achieve satisfactory performance levels on this difficult problem. This thesis represents an investigation into various aspects of the data-mining process using the XCS classifier system on imbalance problem. We will discuss more detail in the next sections.

1.2 Objective of the Study

To solve imbalance problems, there are two approaches: in classifier level and in data level. In this thesis, we would like to study these approaches in XCS classifier system. In more details, we summarize concrete objectives as follows.

- To explore the parameter settings that control XCS. There are many parameters that could be adjusted in XCS. We think it is significant that XCS works well with the default parameter settings. However, it might be possible to substantially improve the performance of XCS with tuning some of these parameters.
- To analyze effects of sampling techniques to XCS performance on imbalance problems
- To find an efficient method to combine sampling techniques and ensemble learning to enhance XCS performance.
- To verify the feasibility of our proposals by getting experimental results and comparing with other methods.

1.3 Assumption of this study

There are some assumptions in this thesis. Firstly, we know that the process of knowledge discovery in data bases (KDD) consists of a series of transformation steps: Data preprocessing, Data Mining and Post-processing. Each process has its own function: the purpose of preprocessing is to transform the raw input data into an appropriate format for subsequent analysis. Beside that, the post-processing integrates data mining results into decision support system. Various techniques are developed to cope with these issues. In this thesis, we concentrate on the classification problems of Data Mining process rather than Dimensionality Reduction, Feature Selection or Clustering techniques, Association Analysis. With this assumption, our system will not focus on removing noise, duplicate instances, missing or redundant features which should be done in the preprocessing step. Particularly, imbalance datasets are considered where the interesting information is usually stored in positive (rare) class.

Secondly, to be able to adapt XCS classifier system suitably for imbalance problem, in this thesis, we will present components of XCS and explain how it works. Some other background knowledge as Evolutionary algorithm, Re-enforcement learning and Rule-based system are only presented briefly and some useful references are provided for interesting readers.

Thirdly, it is interesting to point out that all of the studies we are aware of (including our own) never eliminate members from the minority class. Since there are so few members of the minority class, researchers are very hesitant to eliminate members of the minority class. Instead, the assumption is that each minority class member is very important. This may or may not be the case in practice since some minority class members may represent noise. However, with so few data points, it is difficult to differentiate between noise and minority class members which represent very rarely seen instances (i.e. class disjunctions in the intra class imbalance problem).

Least but not last, our framework supports for artificial and real-life datasets where values of attributes are binary, integer, category and real.

1.4 Theory or Concept to be Used in this Research

There are some crucial theories and concepts described as following:

+ **Niche genetic algorithm** (GA): a set of environmental states is applied genetic operators: crossover and mutation.

In the original XCS, the genetic algorithm was executed in the match set instead of panmictically using the whole population as in many classifier systems. The basic idea of a niche instead of panmictic GA is to eliminate the undesirable competition that otherwise occurs between classifiers in different match sets of a common, cooperative chain of actions. In addition, as Wilson [10] pointed out, crossovers within a niche are more likely to yield useful classifiers than crossovers between potentially unrelated classifiers that match in different niches. XCS first followed Goldberg [5] in performing the GA in the match set [M], but the GA now takes place in the action set [A]. As will be explained in Section 3.5, this change increases the proportion of accurate, maximally general classifiers in the population.

+ **Steady–state GA**: The steady state genetic algorithm is different from the generational genetic algorithm used in many other systems in that there is typically a single new member inserted to the population at any one time in stead of all chromosomes in population are replaced by a new population set.

+ **Macroclassifier** is a combination of all classifiers having the same condition to only one classifier having the most general condition.

+ **Positive class** or **rare class** or **minority class** is the class having the smallest number of instances in a dataset. Other classes are **Negative class** or **common class** or **majority class.**

+ **Imbalance ratio** (*ir*) is the ratio between the number of the majority instances and minority class instances.

1.5 Scope of the Research Work

According to the objective described in section 1.2, the scope of this thesis is illustrated as following:

- **Imbalance Problem**: Classification Imbalance Problem is studied. We analyze its effects to standard classifiers.
- **Classification Algorithm**: XCS Classifier System and all its advantages as well as disadvantages are studied. Although XCS is successful on both multi-step problems and single-step problem, our study only focuses on single-step problems which are classification problems and particularly imbalance problems.
- **Support methods**: Cost-sensitive Learning, Sampling techniques and Ensemble Learning are investigated on conjunction with XCS to enhance performance of system in imbalance problems. There are various algorithms of these three methods proposed by scientific community, but we only study some algorithms which are suitable for XCS in imbalance dataset problem.

1.6 Research Methodology

Firstly, *investigating methods for knowledge representation of classifier in XCS*: XCS evolves a distributed knowledge representation expressed by a population (set) of classifiers (rules) where classifier conditions are usually represented by bit-strings of fixed length on the ternary alphabet {0, 1, #}. Recently, generalization capabilities become the main focus of researches in XCS, so, the representation of classifier conditions turns out to be probably the most interesting topic. The generalization capabilities of classifiers in fact rely on their ability to represent the knowledge that system acquires in a compact form. Many kinds of knowledge representation are proposed: Integer-valued, continuous-valued intervals, messy, s-expression, even neural and fuzzy encodings. We analyze each kind of representation in order to find a way of representing information that is suitable for imbalance dataset problem.

Secondly, *studying components and performance of XCS*: There are three main components in XCS: Performance Component, Reinforcement Component and Discovery Component. These components are considered so that we can know how to adapt XCS and make it work well in our interested problems.

Thirdly, *working with Cost-sensitive Learning, Sampling Techniques and Ensemble Learning*; Cost-sensitive Learning and Sampling techniques are famous algorithms for solving imbalance problems, in while Ensemble learning can reduce both the bias and the variance of single classifier. So, we study these learning methods to find a way to combine or learn advantages of these methods and apply them to XCS.

Fourthly, *proposing Cost-sensitive XCS to solve Imbalance Problem*: After studying components of XCS and Cost-sensitive learning, we must define a combination between them. Based on the combination of reinforcement Component, particularly Q-learning algorithm in updating parameters and cost matrix of Cost-sensitive learning, we can create a new system, Cost-sensitive XCS.

Fifthly, *proposing Sampling Ensemble XCS by combining sampling techniques and ensemble learning to solve Imbalance Problem*: Various algorithms of Ensemble and Sampling are considered. They are adapted and integrated into our ensemble-XCS. This system must get advantages from Ensemble learning and Sampling techniques, and reduce as much disadvantage of this combination as possible.

And finally, *publishing some papers* which present our methods in International Conferences to make sure that our proposals are new ones and new approaches in Data mining as well as Artificial Intelligence field.

1.7 Thesis Organization

We divide this thesis into 8 chapters with title and brief description as following

Chapter 2 Literature Survey

This chapter introduces the nature of imbalance problems. We analyze effects of imbalance problems as well as other factors as small training sample size, distance between classes and the existence of within-class sub-concepts to performance of classifier systems. Then we present methods for dealing with rarity and point out drawbacks with some of these methods. Finally, we discuss the

use of appropriate evaluation metrics to guide the search algorithms employed by data mining systems.

Chapter 3 The XCS Classifier System

The goal of this chapter is to offer an overview of the fundamental aspects of XCS and of the recent developments it is giving rise to. In order to reach that goal, we first present the two mechanisms on which they rely, namely Genetic Algorithm (GA) and Reinforcement Learning (RL). Afterward, we reviews some of the ways in which rules, and especially rule condition, have been represented in classifier systems. Then three main components of XCS: Performance Component, Reinforcement Component and Discovery Component are presented.

Chapter 4 Guidelines for XCS in Imbalance Problems

This chapter analyzes effects of Imbalance problems to XCS's performance on population initialization and on the creation and deletion of classifiers of the minority class. We show that XCS with standard parameter settings is quite robust to class imbalances. For high class imbalances, XCS suffers from biases toward the majority class. Three main points: population initialization, generation of correct classifiers of the minority class and time to extinction of correct classifiers of the minority class are analyzed. Then we extend the theory of computational complexity of XCS to imbalance problems and highlight the impact of learning from class-imbalanced domains on different mechanisms of XCS.

Chapter 5 Cost-sensitive XCS in Imbalance Problems

This chapter presents our first proposed method, Cost-sensitive XCS to solve imbalance dataset problem. We begin with foundations of Cost-sensitive Learning, reward function and value function in XCS. After that is how to combine XCS with cost-sensitive learning to solve Imbalance problems. Finally, experimental results of our first proposal on synthetic and real-life datasets are shown to verify the feasibility of Cost-sensitive XCS.

Chapter 6 Sampling Ensemble XCS in Imbalance Problems

This chapter presents our second proposed method based on adaptation data distribution and multi-classifier to solve imbalance problem, Sampling Ensemble XCS. We begin with foundations of Sampling and Ensemble learning algorithms. We explain why we need sampling and ensemble

method in solving imbalance problems. Then under-sampling, over-sampling of sampling technique and Bagging, Boosting algorithms of Ensemble learning are considered. Finally, the proposed method is described and following that is experimental results on our imbalance datasets

Chapter 7 The Proposed Methods Compared to Other Related Works

This chapter compares the performance, in terms of F1-measure evaluation, of our proposals described in chapter 5 and 6 to six well-known learning algorithms, coming from instance based learning, Rule-learning, Ensemble learning, Decision Tree Induction, Neural Networks, Support Vector Machine and Bayesian Classifier Systems. The experiments verify and show the feasibility of our proposals for imbalance classification tasks.

Chapter 8 Conclusions and Future Researches

This chapter presents main results achieved in the thesis. The conclusions outline limitations with this research and the next steps necessary for further XCS analysis and more competent XCS design.

Chapter 2

Literature Survey

In this chapter, we introduce the nature of imbalance problem. Before proceeding, it is useful to discuss what exactly is meant by rarity. Much of the research on rarity relates to rare classes, or, more generally, class imbalance. This type of rarity requires labeled examples and is associated with classification problems. A second type of rarity concerns minority cases. Informally, minority cases correspond to a meaningful but relatively small subset of the data, or equivalently, define a small region of the instance space. Minority cases depend only on the distribution of data and therefore are defined for both labeled and unlabeled data, and for supervised and unsupervised data mining tasks. Minority cases are naturally defined by the domain and will share common characteristics. In the following of this chapter, the categories of imbalance problems are provided in section 2.2. In Section 2.3 we present the methods for dealing with rarity and point out drawbacks with some of these methods. Finally, in Section 2.4, we discuss the use of appropriate evaluation metrics to guide the search algorithms employed by data mining systems.

2.1 Nature of the Imbalance Problems

In a data set with the class imbalance problem, the most obvious characteristic is the skewed data distribution between classes or degree of imbalance. However, theoretical and experimental studies presented in [32], [33], [75] indicate that the skewed data distribution is not the only parameter that influences the modeling of a capable classifier in identifying rare events. Other influential facts include small training sample size, distance between classes and the existence of within-class sub-concepts. These studies show that classifier systems can perform well in some imbalance domains if the training datasets size is large enough and the distance between classes is appropriate.

2.1.1 Degree of Imbalance

The imbalance degree of a class distribution can be denoted by the imbalance ratio which is the sample size of the majority/large class to that of the minority/small class. In practical applications, the imbalance ratio can be as drastic as 100:1, 1000:1 or even larger [57]. In [33], research was

conducted to explore the relationship between the class distribution of a training dataset and the classification performances of decision trees. Their study indicates that a relatively balanced distribution usually attains a better result. However, at what imbalance degree the class distribution deteriorates the classification performance cannot be stated explicitly, since other factors such as sample size and distance between classes also affect performance. In some applications, a ratio as low as 35:1 can make some methods adequate for building a good model, but in some other cases, ratio 10:1 also makes classifiers difficult to find solutions of minority class [75].

2.1.2 Training Dataset Size

Given a fixed imbalance degree, the sample size plays a crucial role in determining the "goodness" of a classification model. In the case that the sample size is limited, uncovering regularities inherent in small class is unreliable. Experimental observations reported in one of our researches [75] indicate that as the size of the training set increases, the large error rate caused by the imbalanced class distribution decreases. This observation is quite understandable. When more data can be used, relatively more information about the small class benefits the classification modeling, which becomes able to distinguish minority samples from the majority. Hence, we suggest that the imbalanced class distribution may not be a hindrance to classification by providing a large enough data set, assuming that the data set is available and the learning time required for a sizeable data set is acceptable.

2.1.3 Distance between Classes

The difficulty in separating the small class from the prevalent class is the key issue of the small class problem. Assuming that there exist highly discriminative patterns among each class, and then not very sophisticated rules are required to distinguish class objects. However, if patterns among each class are overlapping at different levels in some feature space; discriminative rules are hard to induce. Experiments conducted in [38] vary the degree of overlap between classes. It is then concluded that "the class imbalance distribution, by itself, does not seem to be a problem, but when allied to highly overlapped classes, it can significantly decrease the number of minority class examples correctly classified". A similar claim based on experiments is also reported in [37] as "Linearly separable domains do not sensitive to any amount of imbalance. As a matter of fact, as the degree of concept complexity increases, so does the system's sensitivity to imbalance."

2.1.4 Within-class Concepts

In many classification problems, a single class is composed of various sub-clusters, or sub-concepts. Samples of a class are collected from different sub-concepts. These sub-concepts do not always contain the same number of examples. This phenomenon is referred to as within-class imbalance, corresponding to the imbalanced class distribution between classes [32]. The presence of within-class sub-concepts worsens the imbalance distribution problem (no matter between or within class) in two aspects: (1) the presence of within-class sub-concepts increases the learning concept complexity of the data set; and (2) the presence of within-class sub-concepts is implicit in most cases.

2.2 Categories of Imbalance Problems

There are a number of problems that arise when mining imbalance datasets and there are a number of reasons making an imbalance dataset. Therefore, appropriate categorization of all these imbalance problems may help for us to grasp their basic idea. We divide these reasons into categories and describe each one in detail in separate subsections.

2.2.1 Lack of Data: Absolute Imbalance

The most fundamental problem with imbalance is the associated lack of data. In this section we are concerned with *absolute* imbalance, where the number of examples associated with the rare (minority) class/case is small in an absolute sense. In this situation the lack of data makes it difficult to detect regularities within the rare class/cases.

Figure 2.1 demonstrates the problems that can result from an "absolute" lack of data. The left side of Figure 2.1 shows the original situation, where A3 contains only one positive example, while the right side represents the situation where much more data are available. As can clearly be seen, the learned decision boundaries (shown using dashed lines) much more closely approximate the true decision boundaries when more data are available.

So, minority cases may be due to a lack of data. The impact that these rare cases have on classification performance has been analyzed. In one of our researches, we employed synthetically generated data sets to show that minority cases have a much higher misclassification rate than majority (common) cases [75]. We refer to this as the *problem with minority cases*. This research further demonstrated something that had previously been assumed – that minority cases cause small disjunctions in the learned classifier. The *problem with small disjunctions*, observed in many

empirical studies, is that they (i.e., small disjunctions) generally have a much higher error rate than large disjunctions [55], [56], [67], [75]. We can see that this is again the result of a lack of data. The most thorough empirical study of small disjunctions analyzed thirty real-world data sets and showed that, in the classifiers induced from these data sets, the vast majority of errors are concentrated in the smaller disjunctions [68].

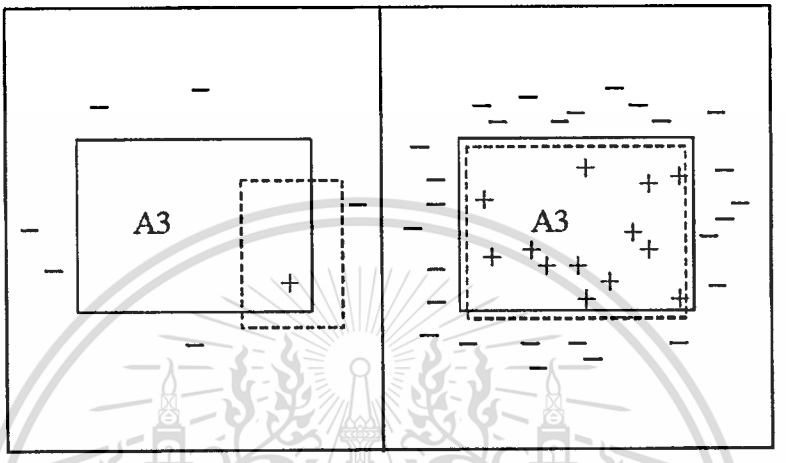


Figure 2.1 The impact of an "absolute" lack of data [3]

The error-prone nature of small disjunctions is a direct result of rarity. Therefore, an understanding of why small disjunctions are so error prone will help explain why rarity is a problem. One explanation is that some small disjunctions may not represent rare, or exceptional, cases, but rather something else—such as noisy data. Thus, only small disjunctions that are "meaningful" should be kept. Most classifier induction systems have some means of preventing over-fitting, to remove sub-concepts (i.e., disjunctions) that are not believed to be meaningful. Statistical significance testing is used by some systems to prevent this over-fitting. Disjunctions that cover few examples will not pass these significance tests. If a data set has two classes and an equal number of training examples in each, then a disjunction is 99% significant if and only if it covers at least 7 training examples [56]. The basic problem is that the significance of small disjunctions cannot be reliably estimated and consequently significant small disjunctions may be eliminated along with the insignificant ones. Empirical results [56] show that the strategy of eliminating all small disjunctions results in an increase in overall error rate and hence is not a good strategy. Error estimation techniques are also unreliable when there are only a few examples, and hence they suffer from the same basic problem. These approaches work well for large disjunctions because in these cases statistical significance and error

rate estimation techniques yield relatively reliable estimates—something they do not do for small disjunctions.

2.2.2 Relative Lack of Data: Relative Imbalance

One problem with imbalance problems is that minority "objects" can be hard to find. This holds true even if imbalance is relative – that is, objects are not rare in an absolute sense but are rare relative to other objects. A popular phrase that illustrates this is "like a needle in a haystack". This phrase is relevant because it is the large number of strands of hay in the haystack that makes it hard to find the needle.

So why is finding/identifying minority objects (patterns, cases, events, etc.) difficult when data mining? One reason is that the minority objects are not easily located using greedy search heuristics and more global methods are, in general, not tractable. Greedy search heuristics have a problem with imbalance problem for several reasons. First, minority objects may depend on the conjunction of many conditions and therefore examining any single condition in isolation may not provide much information, or guidance. While this may also be true of common objects, with minority objects the impact is greater because the common objects may obscure the true signal (the related issue of data fragmentation is covered in Section 2.2.3).

As a specific example of this general problem, consider the association rule mining problem described earlier, where we want to be able to detect the association between *food processor* and *cooking pan*. The problem is that both items are rarely purchased in a supermarket, so that even if the two are often purchased together when either one is purchased, this association may not be found. To find this association, the minimum support threshold for the algorithm must be set quite low. However, if this were done, it would cause a combinatorial explosion because frequently occurring items will be associated with one another in an enormous number of ways. This problem has been called the *rare item problem* [69]. The fact that these random co-occurrences will swamp the meaningful associations between rare items is one example of the problem with relative rarity.

2.2.3 Data Fragmentation

Many data mining algorithms employ a divide-and-conquer approach, where the original problem is decomposed into smaller and smaller problems, which results in the instance space being partitioned into smaller and smaller pieces. Decision tree algorithms are a good example of this approach in that they begin with all of the data (all of the instance space) and repeatedly partition it

into smaller and smaller pieces. This process may lead to data fragmentation [70]. Data fragmentation is a problem because regularities can then only be found within each individual partition, which will contain less data. While data fragmentation is always a concern, it is more of a concern when mining rare classes/cases, because of the existing "lack of data" problem described in Section 2.2.1. Thus all iterative divide-and-conquer approaches may have difficulty in the presence of imbalance problem.

2.2.4 Inappropriate Inductive Bias

Generalizing from specific examples, or induction, requires an extra-evidentiary bias. Without such a bias "inductive leaps" are not possible and learning cannot occur. The bias of a data mining system is therefore critical to its performance. Many learners utilize a general bias in order to foster generalization and avoid over-fitting. This bias can adversely impact the ability to learn the minority class.

Consider a learner with a maximum-generality bias [56]. Once the learner decides to create a disjunction that covers some set of training examples, it selects the most general set of conditions that satisfy those examples but no others. This can be contrasted with a maximum-specificity bias, which will add all possible conditions that satisfy the training examples. The maximum-generality bias works well for large disjunctions but not for small disjunctions, leading to the observed problem with small disjunctions. Thus we infer that the maximum-generality bias is not appropriate for minority cases, since it may make them overly general. This leads to error-prone classification rules for predicting the minority class, which may subsequently be pruned. Most methods for addressing the problem of small disjunctions (and minority cases) operate by adjusting the bias of the learner. Inductive bias also plays a role with respect to imbalance problems. Many induction systems will tend to prefer the more common classes in the presence of uncertainty (i.e., they will be biased in favor of the class priors).

2.2.5 Noise

Noisy data will affect the way any data mining system behaves, but, what is interesting from the perspective of this article is that noise has a greater impact on rare cases than on common cases. To see this, consider Figure 2.2 where A is the minority class and B is the majority class. The right side of this Figure presents a noise-free dataset, but the left side introduces some noisy examples. Noise creates a problem when positive-class examples are found in the negative class (class B) and negative-class examples are found in the positive class (class A).

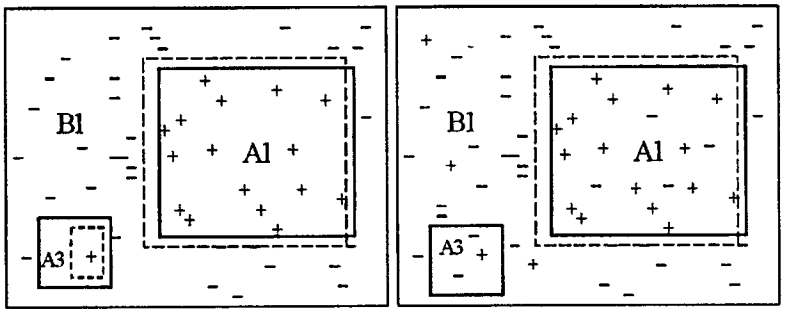


Figure 2.2 The effect of noise on minority instances [39]

Because rare cases have fewer examples to begin with, it will take fewer "noisy" examples to impact the learned sub-concept. As we can see in Figure 2.2, the four noisy data points in A1 have no impact on the learned decision boundary, because of the learner's ability to generalize. However, the two noisy data point in A3 cause the learner to not learn this rare case at all (i.e., there is no decision boundary). In this case the learner cannot distinguish between exceptional (minority) cases and noise, a problem that has been analyzed previously [33]. If the learner were modified to generalize less, so that a portion of A3 were correctly learned, this would most likely have the undesirable effect of having small disjunctions formed to cover the noisy examples in A1 and B1. Thus, noise necessitates the use of over-fitting avoidance techniques (e.g., pruning) to eliminate noise-induced small disjunctions and a consequence of this is that some "true" minority cases will not be learned. Should these minority cases be important enough, one should be able to adjust the bias of the learner to include them, even though this will have some undesirable consequences.

2.3 Related Works

Methods to deal with class imbalances can be applied at the data-level or at the algorithm-level. Those acting at the data-level aim at balancing the a priori probabilities of classes, either by over-sampling the minority class instances or under-sampling the majority class instances. The second class of methods tries to adapt the classifier to class imbalances, e.g., by measuring each classification cost separately.

2.3.1 Data-level Approaches

Solutions at the data level include many different forms of re-sampling, such as randomly over-sampling the small class, randomly under-sampling the prevalent class, informatively over-

sampling the small class (in which no new samples are created, but the choice of samples to resample is targeted rather than random), informatively under-sampling the prevalent class (the choice of samples to eliminate is targeted), over-sampling the small class by generating new synthetic data, and combinations of the above techniques.

Chawla et al [44] devised a method called Synthetic Minority Oversampling Technique (SMOTE). This technique involved creating new instances through "phantom-transduction." For each positive instance, its nearest positive neighbors were identified and new positive instances were created and placed randomly in between the instance and its neighbors. Since this technique creates new positive instances, we found this technique to be more useful for XCS than simple oversampling.

In [49], Kubat and Matwin selectively under-sampled the majority class while keeping the original population of the minority class. They have used the geometric mean as a performance measure for the classifier, which can be related to a single point on the ROC curve where the ROC curve is a popular measurement evaluating the performance of classifier in imbalance problems. We describe detail ROC curve in section 2.4.3. The minority examples were divided into four categories: some noise overlapping the positive class decision region, borderline samples, redundant samples and safe samples. Another related work proposed the SHRINK system that classifies an overlapping region of minority (positive) and majority (negative) classes as positive; it searches for the "best positive region" [49].

Japkowicz [34] discussed the effect of imbalance in a dataset. She evaluated three strategies: under-sampling, re-sampling and a recognition-based induction scheme. We focus on her sampling approaches. She experimented on artificial 1D data in order to easily measure and construct concept complexity. Two re-sampling methods were considered. Random re-sampling consisted of re-sampling the smaller class at random until it consisted of as many samples as the majority class and "focused re-sampling" consisted of re-sampling only those minority examples that occurred on the boundary between the minority and majority classes. Random under-sampling was considered, which involved under-sampling the majority class samples at random until their numbers matched the number of minority class samples; focused under-sampling involved under-sampling the majority class samples lying further away. She noted that both the sampling approaches were effective, and she also observed that using the sophisticated sampling techniques did not give any clear advantage in the domain considered.

Another approach that is similar to our work is that of Domingos [65]. He compares the "metacost" approach to each of majority under-sampling and minority over-sampling. He finds that

metacost improves over either, and that under-sampling is preferable to minority over-sampling. Error-based classifiers are made cost-sensitive. The probability of each class for each example is estimated, and the examples are relabeled optimally with respect to the misclassification costs. The relabeling of the examples expands the decision space as it creates new samples from which the classifier may learn.

The information retrieval domain [35], [43] also faces the problem of class imbalance in the dataset. A document or web page is converted into a bag-of-words representation; that is, a feature vector reflecting occurrences of words in the page is constructed. Usually, there are very few instances of the interesting category in text categorization. This over-representation of the negative class in information retrieval problems can cause problems in evaluating classifiers' performances. Since error rate is not a good metric for skewed datasets, the classification performance of algorithms in information retrieval is usually measured by *precision* and *recall* (detail in section 2.4).

Even though re-sampling is an often-used method in dealing with the class imbalance problem, the matter at issue is what is or how to decide the optimal class distribution given a data set. A thorough experimental study on the effect of a training set's class distribution on a classifier's performance is conducted in [33]. The general conclusion is that, with respect to the classification performance on the small class, a balanced class distribution (class size ratio is 1:1) performs relatively well but is not necessarily optimal. Optimal class distributions differ from data to data.

2.3.2 Algorithm-level Approaches

Generally, a common strategy to deal with the class imbalance problem is to choose an appropriate inductive bias. To develop an algorithmic solution, one needs knowledge of both the corresponding classifier learning algorithm and the application domain, especially a thorough comprehension on why the learning algorithm fails when the class distribution of available data is uneven.

For neural networks, a feed-forward neural network trained on an imbalanced dataset may not learn to discriminate enough between classes [58]. The authors proposed that the learning rate of the neural network be adapted to the statistics of class representation in the data. They calculated an attention factor from the proportion of samples presented to the neural network for training. The learning rate of the network elements was adjusted based on the attention factor. They experimented on an artificially generated training set and on a real-world training set, both with multiple (more than

two) classes. They compared this to the approach of replicating the minority class samples to balance the data set used for training. The classification accuracy on the minority class was improved.

Lewis and Catlett [73] examined heterogeneous uncertainty sampling for supervised learning. This method is useful for training samples with uncertain classes. The training samples are labeled incrementally in two phases and the uncertain instances are passed on to the next phase. They modified C4.5 to include a loss ratio for determining the class values at the leaves. The class values were determined by comparison with a probability threshold of LR/(LR + 1), where LR is the loss ratio.

Association-rule mining systems generally employ an exhaustive search algorithm [36] and are therefore, in theory, capable of finding rare associations. The problem, previously described in Section 2.2, is that these algorithms become intractable if the minimum level of support is set small enough to find rare associations. Thus, such algorithms are heuristically inadequate for finding rare associations. This problem can be solved by specifying multiple minimum levels of support to reflect the frequencies of the associated items in the distribution [59].

For the kNN classifier, we see some interesting general trends. The k nearest neighbor classification algorithm is an instance-based learning method first proposed in [60]. kNN is simple yet efficient and has been used in the past on classification. In particular, kNN is popular since the only parameters one needs to choose are k and an appropriate distance metric. In particular, kNN works quite well on several imbalance datasets. Since kNN classifies input instance based on local neighborhoods of the data, the minority class consisted of instances which were very similar to each other. In this case, simple approaches seem to work best.

2.3.3 In Learning Classifier System Field

In the Learning Classifier System field, there are few studies analyzing LCS's behavior with imbalanced datasets. Holmes [30] addressed this topic in the context of epidemiological datasets, and adapted EpiCS applying a strategy based on disproportionate reinforcement per class.

In [29], A. Orriols and E.B. Mansilla, proposed a solution dealing with the class imbalance problem by finding fitness adaptation based on class-sensitive accuracy in combination with UCS, a supervised LCS derived from XCS, as a useful tool for alleviating the effects of class imbalances. UCS is also studied in [27], this research analyzes that XCS, UCS and some other learning classifier systems have their fitness based on accuracy; they present a high bias toward the majority class instances and evolve easily *over-general* classifiers. Idea of the proposal is restrict classifiers to cover

regions formed by examples of a single class and make accuracy class-sensitive rather than instance-sensitive. Thus, accuracy is modified so that each class is considered equally important regardless of the number of instances representing in each class. The experiments show that this model evolved with imbalance level up to i=7.

In addition, in [28], they proposed to adapt value of learning rate of classifiers and the genetic algorithm triggering rate (θ_{GA}). Their suggestion is to set learning rate according to the proportion of frequencies between the least occurring niche (f_{min}) and the most frequent niche (f_{max}). If the imbalance ratio increases, we should also decrease the learning rate. Beside that, this study analyzed the function of GA triggering rate, if this rate is very low, the genetic algorithm will be activated very often and will favor the most frequent classifiers. Those classifiers occurring infrequently will receive a GA event only when they belong to an action set. A way to counterbalance the reproductive opportunities of niches is to set the triggering GA rate to a value higher than the maximum delay between infrequent niches. This would guarantee that all niches will receive the same opportunities, regardless of the occurrence frequency of each niche.

Another kind of research related to our work is studying Learning Classifier System in Intrusion Detection [31]. An intrusion detection problem is characterized by huge network traffic volumes, difficult to realize decision boundaries between attacks and normal activities and highly imbalanced attack class distribution. Moreover, it demands high accuracy, fast processing times and adaptability to a changing environment. They presented the results and analysis of two classifier systems (XCS and UCS) on a subset of a publicly available benchmark intrusion detection dataset which features serious class imbalances and two very rare classes. They also introduced an approach for handling the situation when no rules match an input on the test set and recommended this be adopted as a standard part of XCS and UCS. By experiments, both systems tend to reach near-best performance in very few passes over the training data.

2.4 Evaluation Measures

The accuracy measure, which is used extensively to compare the performance of classifiers, may not be well suited for evaluating models derived from imbalanced data sets. For example, if 1% of the credit card transactions are fraudulent, then a model that predicts every transaction as legitimate has an accuracy of 99% even though it fails to detect any of the fraudulent activities. This section

presents some of the measures: *precision*, *recall*, *F-measure* and *g-mean* used in our thesis for evaluating the performance of classifiers in imbalance datasets.

2.4.1 Precision, Recall and F-measure

In classification tasks, one class with very few training samples but high identification importance is referred to as the positive class; the other as the negative class. Samples can be categorized into four groups after a classification process as denoted in the confusion matrix presented in Table 2.1 for bi-class problems and table 2.2 for multi-class problems. In table 2.1, we consider all others classes as the negative class.

Table 2.1 A confusion matrix for a binary classification problem in which the classes are not equally important

		Predicted Class	
		+	-
Actual Class	+	*f*++ (TP)	*f*+- (FN)
	-	*f*-+ (FP)	*f*-- (TN)

Table 2.2 A confusion matrix for a multi-classification problem in which the classes are not equally important

		Predicted Class	
		+	Others
Actual Class	+	*f*++ (TP)	*f*+- (FN)
	Others	*f*-+ (FP)	*f*-- (TN)

The following terminology is often used when referring to the counts tabulated in a confusion matrix:

+ **True Positive** (TP) or *f*++, which corresponds to the number of positive examples correctly predicted by the classification model.

+ **False Negative** (FN) or *f*+-, which corresponds to the number of positive examples incorrectly predicted as negative by the classification model.

+ **False Positive** (FP) or *f*-+, which corresponds to the number of negative examples incorrectly predicted as positive by the classification model.

+ **True Negative** (TN) or *f*--, which corresponds to the number of negative examples correctly predicted by the classification model.

Several measures can be derived using the confusion matrix:

• **True Positive Rate:** $TP_{rate} = \frac{TP}{TP + FN}$

• **True Negative Rate:** $TN_{rate} = \frac{TN}{TN + FP}$

• **False Positive Rate:** $FP_{rate} = \frac{FP}{TN + FP}$

• **False Negative Rate:** $FN_{rate} = \frac{FN}{TP + FN}$

Recall and ***precision*** are two widely used metrics employed in applications where successful detection of one of the classes is considered more significant than detection of the other classes. A formal definition of these metrics is given below.

$$Precision,\ p = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall,\ r = \frac{TP}{TP + FN} \tag{2.2}$$

Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class. The higher the *precision* is, the lower the number of false positive errors committed by the classifier. *Recall* measures the fraction of positive examples correctly predicted by the classifier. Classifiers with large *recall* have very few positive examples misclassified as the negative class. In fact, the value of *recall* is equivalent to true positive rate (TPR).

It is often possible to construct baseline models that maximize one metric but not the other. For example, a model that declares every record to be the positive class will have a perfect *recall*, but very poor *precision*. Conversely, a model that assigns a positive class to every test record that matches one of the positive records in the training set has very high *precision*, but low *recall*. Building a model that maximizes both *precision* and *recall* is the key challenge of classification algorithms.

Precision and *recall* can be summarized into another metric known as the *F1-measure*.

$$F_1 = \frac{2rp}{r + p} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2.3}$$

In principle, F_1 represents a harmonic mean between *recall* and *precision*, i.e.,

$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}} \tag{2.4}$$

The harmonic mean F1 of two numbers r and p tends to be closer to the smaller of the two numbers. Hence, a high value of F_1-measure ensures that both *precision* and *recall* are reasonably high.

More generally, the F_β measure can be used to examine the tradeoff between *recall* and *precision*:

$$F\beta = \frac{(\beta^2+1)rp}{r+\beta^2 p} = \frac{(\beta^2+1)\times TP}{(\beta^2+1)\times TP + \beta^2 \times FP + FN} \qquad (2.5)$$

Both *precision* and *recall* are special cases of F_β by setting β=0 and $\beta=\infty$, respectively. Low values of β make F_β closer to *precision*, and high values make it closer to *recall*.

2.4.2 g-mean

When the performance of both classes is concerned, both True Positive Rate (*TPrate*) and True Negative Rate (*TNrate*) are expected to be high simultaneously. Kubat et al. [49] suggested the g-mean defined as

$$g-mean = \sqrt{TP_{rate} \cdot TN_{rate}} \qquad (2.6)$$

g-mean measures the balanced performance of a learning algorithm between these two classes. The comparison among harmonic F1, geometric, and arithmetic means are illustrated in [3] by way of an example. Suppose that there are two positive numbers 1 and 5. Their arithmetic mean is 3, their geometric mean is 2.236, and their F1-measure is 1.667. The F1-measure is the closest to the smaller value and the geometric mean is closer than the arithmetic mean to the smaller number.

2.4.3 ROC Analysis

A receiver operating characteristic (ROC) curve is a graphical approach for displaying the tradeoff between true positive rate and false positive rate of a classifier. In an ROC curve, the true positive rate (*TPR*) is plotted along the y axis and the false positive rate (*FPR*) is shown on the x axis. Each point along the curve corresponds to one of the models induced by the classifier. Figure 2.3 shows the ROC curves for a pair of classifiers, M_1 and M_2.

There are several critical points along an ROC curve that have well-known interpretations:

(TPR = 0, FPR = 0): Model predicts every instance to be a negative class.

(TPR = 1, FPR = 1): Model predicts every instance to be a positive class.

(TPR = 1, FPR = 0): The ideal model.

A good classification model should be located as close as possible to the upper left corner of the diagram, while a model that makes random guesses should reside along the main diagonal, connecting the points (*TPR* = 0, *FPR* = 0) and (*TPR* = 1, *FPR* = 1). Random guessing means that a record is classified as a positive class with a fixed probability *p*, irrespective of its attribute set. For example, consider a data set that contains n_+ positive instances an n_- negative instances. The random classifier is expected to correctly classify pn_+ of the positive instances and to misclassify pn_- of the negative instances. Therefore, the *TPR* of the classifier is $(pn_+)/n_+ = p$, while its *FPR* is $(pn_-)/p=p$. Since the *TPR* and *FPR* are identical, the ROC curve for a random classifier always resides along the main diagonal.

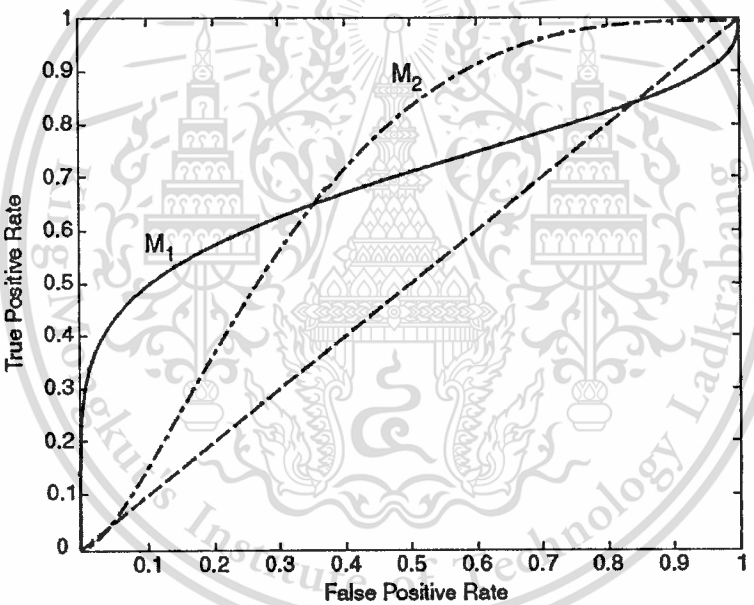


Figure 2.3 ROC curves for two different classifiers [3]

An ROC cure is useful for comparing the relative performance among different classifiers. In Figure 2.3, M_1 is better than M_2 when *FPR* is less than 0.36 is superior when *FPR* is greater than 0.36. Clearly, neither of these two classifiers dominates the other.

The area under the ROC curve (AUC) provides another approach for evaluating which model is better on average. If the model is perfect, then its area under the ROC curve would equal 1. If the model simply performs random guessing, then its area under the ROC curve would equal 0.5. A model that is strictly better than another would have a larger area under the ROC cure.

2.5 Summary

This chapter investigates the issues related to mining in the presence of imbalance problems. The problems that can arise from these two forms of rarity are categorized and described in detail. Methods for addressing these problems are also described. In most instances the descriptions of the problems and solution methods include descriptions of relevant research and data mining programs; thus, this chapter provides a good survey of the literature on data mining with imbalance problems. This chapter also demonstrates that minority classes and minority cases both suffer from the same set of problems and generally can benefit from the same solutions. Thus both forms of rarity can be viewed as being instances of the same basic phenomenon. This realization is described in the next chapters.

Chapter 3

The XCS Classifier System

Learning Classifier Systems (LCSs) are rule-based systems that automatically build their rule-set. LCSs seek to gain reinforcement from their environment based on an evolving set of condition-action rules called classifiers. Via a Darwinian process, useful classifiers in gaining reinforcement are selected and propagate over those less useful, leading to increasing system performance. Recently, LCSs and particularly XCS have arisen as promising methods for classification tasks and data mining. XCS is a most successful kind of learning classifier system that differs from the traditional kind primarily in its definition of classifier fitness and its relation to contemporary reinforcement learning. Advantages of XCS include improved performance and an ability to form accurate maximal generalizations. The goal of this chapter is to offer an overview of the fundamental aspects of XCS and of the recent developments it is giving rise to.

In order to reach that goal, we first present the two mechanisms on which they rely, namely Genetic Algorithm (GA) and Reinforcement Learning (RL). Afterward, we present recent research on XCS with respect to representation and three components: Performance Component, Reinforcement Component and Discovery Component.

3.1 Background Knowledge

3.1.1 Genetic Algorithm

Genetic algorithms (GAs) are part of a class of what is known as *evolutionary algorithms* [5]. Evolutionary algorithms are computational models that solve a given problem by maintaining a changing population of individuals, each with its own level of "fitness". The change in the population is achieved by the reproduction, crossover and mutation procedures within the method. The operation of these three procedures is dependent upon the fitness of the individuals concerned. Given a clearly defined problem to be solved and a binary string representation for candidate solutions, a basic GA can be represented as in Figure 3.1

```
Simple GA() {
        Initialize population;
        Evaluate population;
        While termination criterion not reached {
                Select solution for next population;
                Perform crossover and mutation;
                Evaluate population;
        }
}
```

Figure 3.1 Simple Genetic Algorithm structure

Since research on GAs is now a field in itself, we will not survey it in this article. Though GAs are at their root, LCSs have made limited use of the important extensions of this field. As a consequence, in order to introduce the GAs used in LCSs, it is only necessary to describe the following aspects:

– We must classically distinguish between the *one-point crossover* operator, which cuts two genotypes into two parts at a randomly selected place and builds a new genotype by inverting the sub-parts from distinct parents, and the *multi-point* crossover operator, which does the same after cutting the parent genotypes into several pieces. Historically, most early LCSs were using the one-point crossover operator. Recently, a surge of interest on the discovery of complex "building blocks" in the structure of input data led to a more frequent use of multi-point crossover.

– We must also distinguish between *generational* GAs, where all or an important part of the population is renewed from one generation to the next, and *steady state* GAs, where individuals are changed in the population one by one without notion of generation. Most LCSs use a steady-state GA, since this less disruptive mechanism results in a better interplay between the evolutionary process and the learning process, as explained below.

3.1.2 Markov Chain Model

The second fundamental mechanism in LCSs is Reinforcement Learning. In order to describe this mechanism, it is necessary to briefly present the Markov Decision Process (MDP) framework and the Q-learning algorithm, which is now the learning algorithm most used in LCSs. This presentation is as succinct as possible; the reader who wants to get a deeper view is referred to [66].

An MDP is defined as the collection of the following elements:

– A finite set S of discrete states s of an agent;

– A finite set A of discrete actions a;

– A transition function P: $S \times A \rightarrow \Pi(S)$ where $\Pi(S)$ is the set of probability distributions over S. A particular probability distribution $Pr(s_{t+1}|s_t, a_t)$ indicates the probabilities that the agent reaches the different s_{t+1} possible states when he performs action at in state s_t;

– A reward function R: $S \times A \rightarrow IR$ which gives for each (s_t, a_t) pair the scalar reward signal that the agent receives when he performs action at in state s_t.

The MDP formalism describes the stochastic structure of a problem faced by an agent, and does not tell anything about the behavior of this agent in its environment. It only tells what, depending on its current state and action, will be its future situation and reward.

The above definition of the transition function implies a specific assumption about the nature of the state of the agent. This assumption, known as the *Markov property*, stipulates that the probability distribution specifying the s_{t+1} state only depends on s_t and a_t, but not on the past of the agent. Thus $P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0)$. This means that, when the Markov property holds, knowledge of the past of the agent does not bring any further information on its next state.

The behavior of the agent is described by a policy π giving for each state the probability distribution of the choice of all possible actions.

When the transition and reward functions are known in advance, *Dynamic Programming* (DP) methods such as *policy iteration* and *value iteration* efficiently find a policy maximizing the accumulated reward that the agent can get out of its behavior [66].

In order to define the accumulated reward, a discount factor $\gamma \in [0, 1]$. This factor defines how much the future rewards are taken into account in the computation of the accumulated reward at time t as follows:

$$Rc_{\pi}(t) = \sum_{k=t}^{T\max} \gamma^{(k-t)} r_{\pi}(k) \qquad (3.1)$$

where T_{max} can be finite or infinite and $r_{\pi}(k)$ represents the immediate reward received at time k if the agent follows policy π.

DP methods introduce a *value function* V^{π} where $V^{\pi}(s)$ represents for each state s the accumulated reward that the agent can expect if it follows policy π from state s. If the Markov property holds, V^{π} is solution of the Bellman equation:

$$\forall s \in S, V^{\pi}(s) = \sum_{a} \pi(s_t, a_t)[R(s_t, a_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^{\pi}(s_{t+1})] \qquad (3.2)$$

Rather than the value function V^{π}, it is often useful to introduce an action-value function Q^{π} where $Q^{\pi}(s, a)$ represents the accumulated reward that the agent can expect if it follows policy π after having done action a in state s. Everything that was said of V^{π} directly applies to Q^{π}, given that $V^{\pi}(s) = \max_a Q^{\pi}(s, a)$. The corresponding optimal functions are independent of the policy of the agent; they are denoted V^* and Q^*.

3.1.3 Reinforcement Learning

Learning becomes necessary when the transition and reward functions are not known in advance. In such a case, the agent must explore the outcome of each action in each situation, looking for the (s_t, a_t) pairs that bring it a high reward.

Figure 3.2 In RL problems an adaptive agent interacts with an environment executing actions and receiving state information and reinforcement feedback.

The main RL methods consist in trying to estimate V^* or Q^* iteratively from the trials of the agent in its environment. All these methods rely on a general approximation technique in order to estimate the average of a stochastic signal received at each time step without storing any information from the past of the agent. Let us consider the case of the average immediate reward. Its exact value after k iterations is

$$E_k(s) = \frac{r_1 + r_2 + \ldots + r_k}{k} \tag{3.3}$$

Furthermore,

$$E_{k+1}(s) = \frac{r_1 + r_2 + \ldots + r_k + r_{k+1}}{k+1} \tag{3.4}$$

Thus

$$E_{k+1}(s) = \frac{k}{k+1} E_k(s) + \frac{r_{k+1}}{k+1} \tag{3.5}$$

Which can be rewritten:

$$E_{k+1}(s) = \frac{k+1}{k+1}E_k(s) - \frac{1}{k+1}E_k(s) + \frac{r_{k+1}}{k+1} \quad (3.6)$$

Or

$$E_{k+1}(s) = E_k(s) + \frac{1}{k+1}[r_{k+1} - E_k(s)] \quad (3.7)$$

Formulated that way, we can compute the exact average by merely storing k. If we do not want to store even k, we can approximate $1/(k+1)$ with, which results in equation (3.7) whose general form is found everywhere in RL:

$$E_{k+1}(s) = E_k(s) + \alpha[r_{k+1} - E_k(s)] \quad (3.8)$$

The parameter α, called *learning rate*, must be tuned adequately because it influences the speed of convergence towards the exact average.

We do not detail all of the RL method relying on this estimation principle. We only give the update equation of the Q-learning algorithm, which is the following:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (3.9)$$

3.2 Classifier Rule Encodings

This section reviews some of the ways in which rules, and especially rule condition, have been represented in LCS. There are a number of different types of classifier rule encoding that have been used with classifier systems and, in particular, with the XCS classifier system including binary, continuous-valued and integer-valued intervals, messy, s-expression and even neural and fuzzy encodings have been used.

3.2.1 Binary

Binary appears to represent the first type of encoding used in the classifier system, which may be a result of the close historical development between the classifier system and Genetic Algorithms. A great deal of analytical work has been carried out on binary Genetic Algorithms (the standard encoding for this technique), which will have an indirect relevance to any analysis of binary-encoded classifier systems. Using what we will call the standard ternary LCS language each rule has a single condition and a single action. Conditions are fixed length strings from {0, 1, #}, while rule actions and environmental inputs are fixed length strings from {0, 1}.

A rule's condition c is said to match an environmental input m if for each character m_i the character in the corresponding position c_i is identical or the wildcard (#). For example, the condition

00# matches two inputs: 000 and 001. The wildcard is the means by which rules generalize over environmental states; the more #s a rule contains the more general it is. The most general 3-bit rule condition is ###, and the most specific are those without any #s, e.g., 000, 001, 010 and so on. Since actions do not contain wildcards the system cannot generalize over them.

3.2.2 Messy

In [15], Lanzi presented a version (mXCS) of the XCS classifier system which has a Messy-encoding for classifier rule conditions. The implementation is very similar to the standard Binary implementation except for the classifier-specific issues related to matching and the cover operator. In particular, an under-specified sensor is seen by the classifier system as a don't care for that sensor while positional precedence is used for any over-specified sensor value. The cover operator makes use of this under-specification by generating a random condition of messy genes where the probability of including a given sensor position is P_s (the messy gene's value is taken directly from the environmental state). In addition, a constraint is enforced in which a covered condition must have at least one gene in order to avoid trivial classifier rules. In order to overcome problems related to the proliferation of classifier rules with many under-specified sensors, Lanzi modified the mutation operator and matching procedures. In particular, the mutation operator was extended such that $p_{sensor} = p_{value}$, a messy gene that matches one of current sensory inputs can be added or a messy gene can be removed with probability p_{add} and p_{remove}, respectively. The matching procedure was modified such that a condition only matches the current sensor input if all sensor values (if over-specified) match the sensors. These modifications allowed the Messy XCS classifier system to achieve optimal performance for the test environments used, matching the performance of the Binary XCS system for the same environments.

3.2.3 Continuous-Valued and Integer-Valued Intervals

In [11], Wilson presented a version (XCSR) of the XCS classifier system [8] for problems which can be defined by a vector of bounded continuous real-coded variables. In [11], a condition C is defined as consisting of l interval predicates of the form $\{\{c_1, s_1\},\ldots, \{c_l, s_l\}\}$, where c_i is the interval's range *centre*, s_i is the *spread* from that centre and l is the number of variables. This is termed the *Centre-Spread* encoding throughout this study. Each interval predicate's upper and lower bounds are calculated as follows: $[c_i - s_i; c_i + s_i]$. If an interval predicate goes outside the variable's defined bounds, it is truncated. In order for a classifier rule to match the environmental stimulus, each input

vector value must be within the interval predicate defined for that variable. In [17], Wilson described the XCSI classifier system, which uses a different kind of numerical encoding for multi-variable problems. A solution is defined as a vector of integer-coded interval predicates in the form $\{[l_1, u_1],\ldots, [l_l, u_l]\}$, where l_i and u_i are the intervals' lower and upper bounds, respectively. Although, there is no necessity for integers in this type of interval encoding, a real-coded version would be trivial to implement.

3.2.4 Fuzzy Sets

The idea of a fuzzy set is an alternative to the crisp set defined in standard set theory. In [19], Bonarini et al present a comparison between a crisp and a fuzzy Learning Classifier System ("Michigan-style") for a robot navigation application. The crisp representation used is analogous to the continuous interval encoding described above. Both classifier systems use rules whose condition represents a conjunction of symbolic values for both the input variables and output variables. In the crisp system, each symbol describes an interval of real-values for a given variable while in the fuzzy system, each symbol represents a fuzzy subset of the given variable's range. Conditions may also contain don't care symbols which are used to identify irrelevant variables. In addition, symbols used for each output variable corresponds to a crisp value. The system uses a fuzzy aggregation operator [1] to combine the proposed actions, or turned in to a crisp output value. The interested reader is referred to [18] for an overview of fuzzy-encoded Learning Classifier Systems.

3.2.5 S-expressions

In [16], Lanzi presented a version (XCSL) of the XCS classifier system that uses a general-purpose representation for classifier rule conditions, namely, s-expressions. The implementation of the XCSL classifier system was both clarified and extended in [16] and as such [16] forms the basis of the following description. In particular, the XCSL classifier system works in the same way as the standard XCS system except that it differs in four important ways, that is, during condition matching, the cover operator, the genetic operators used in the discovery component and condensation. Lanzi discovered a tendency in the XCSL system to evolve large populations of over-complicated classifier conditions making analysis of evolved solutions impossible. In fact, Wilson [8] suggested that executing a condensation phase at the end of a period of learning would cause the classifier system to compact its knowledge without loss of performance. The condensation phase involves executing the Genetic Algorithm with no recombination or mutation operators. This causes weaker individuals to be

removed allowing fitter individuals to reproduce preferentially. As a result of these modifications, the XCSL classifier system was shown to outperform C4.5 at high level of statistical significance (P < 0:005) [16].

3.2.6 Neural Nets

In [20], Bull and O'Hara introduced two new versions of the XCS system, that is, X-NCS which uses a Multi-layer Perceptron encoding and X-NFCS which uses a Radial Basis Function encoding. In the first encoding, a single, fully connected Multi-layer Perceptron replaces both the condition and action parts of the standard classifier rule as represented by an arbitrary concatenation of network weights that described an instantiation of an Multi-layer Perceptron. All Multi-layer Perceptron rules have an input node for each input variable, a fixed-size hidden layer and $n + 1$ output nodes where n represents the number of possible actions. The output node with the highest output value represents the classifier rule's proposed action except when the extra output node has the highest output value which signifies that the classifier rule should not be added to the Match Set. In the second encoding, a fully connected Radial Basis Function replaces the standard classifier rule and is represented by an arbitrary concatenation of basis function centers, widths and network weights describing an instantiation of an Radial Basis Function. The system works in the same way as the X-NCS, except a positive response on the extra node is all that is required to stop the rule being added to the Match Set. Bull and O'Hara suggest that the X-NFCS classifier system is capable of building co-evolutionary solutions such that "different rules emerge to handle different regions of the input space together covering the total problem space". An added advantage to both types of neural representation is the capability to describe both discrete and continuous action spaces.

In conclusion, after analysis all kinds of representations and all kinds of UCI datasets used in this thesis, we see that the third representation, continuous-valued and integer-valued intervals are suitable for our imbalance data. So, in this thesis, all experimental results are gotten by using this representation kind. In the next sections, we introduce three main components of XCS.

3.3 Performance Component

Figure 3.3 gives an overall picture of the system, which is shown in interaction with an environment via detectors for sensory input and effectors for motor actions.

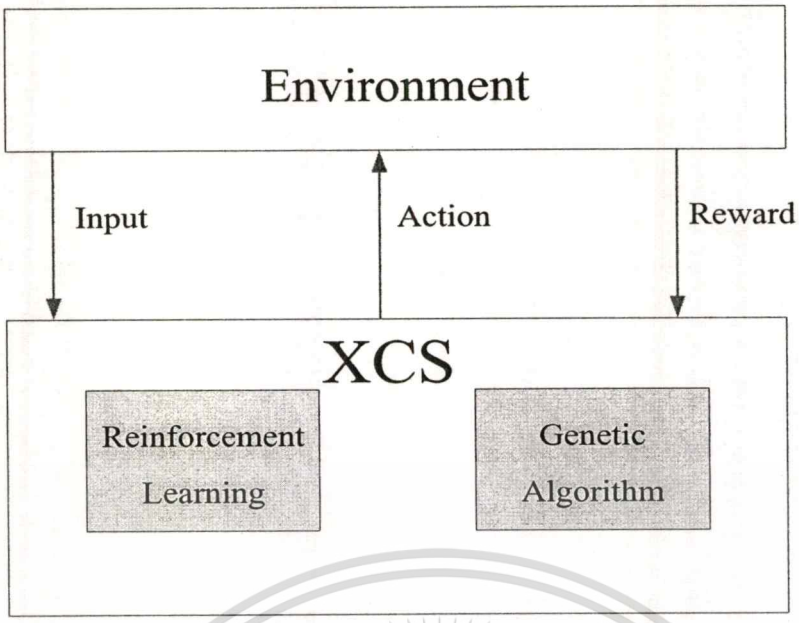


Figure 3.3 Overall schematic illustration of XCS

```
Performance of XCS {
        Step 1: Initialize population
        While termination criterion not reached {
                Step 2: Encoding input by Detectors
                Step 3: Creating Match set [M]
                Step 4: Calculating Prediction Array
                Step 5: Creating Action set [A]
                Step 6: Decoding the output by Effectors
                Step 7: Get reward and update parameters
                Step 8: Using a niche GA to generate new rules
        }
}
```

Figure 3.4 Pseudo code of XCS performance

Given a clearly defined problem to be solved and a string representation as described in section 3.2, XCS performance can be represented as in Figure 3.4 and Figure 3.5. As we see, XCS represents an iterative process. Each iteration is called a generation. The number of generations is based on the complexity of problem needed to solve. The environment at times provides a scalar

reinforcement, here termed reward. The box labeled [P] contains the classifier population, and shows some example classifiers. The left side of each classifier consists of a single condition; the right side codes an environmental action. Associated with each classifier are prediction, prediction error, and fitness parameters, symbolized by p, ϵ, and F, respectively. Where *prediction p* is an average of the payoff received - internal or external - when that classifier's action controlled the system; *prediction error* ϵ is an average of a measure of the error in the prediction parameter; and *fitness F* is an inverse function of the prediction error.

Here, we explain performance of XCS detail step by step as following:

Step 1: *Initialize population*. The population has a fixed maximum size N and may be initialized in a variety of ways: with N randomly generated classifiers; with potentially useful "seed" classifiers; with no classifiers; or with one general (condition consisting of #'s) classifier for each action; etc. The initial values of p, ϵ, and F can be set more or less arbitrarily; there is little effect on performance.

Step 2: *Encoding input by Detectors*. Encoding input is one of three encoding tasks in XCS. The two other encoding tasks are: encoding condition and encoding action. As described in section 3.2, there are several ways to represent condition of classifier as well as input of environment. A simple example will help us to understand how an XCS works. Let us classify a classification problem having 2 input attributes: A1 and A2, each attribute has 4 possible values: {0, 1, 2 and 3}. For simplicity, we may assume that A1 and A2 take only integer values. Thus, the input can be encoded with only four genes:

Table 3.1 Condition Encoding

A1	A2	Binary Code	A1	A2	Binary Code	A1	A2	Binary Code	A1	A2	Binary Code
0	0	0000	1	0	0100	2	0	1000	3	0	1100
0	1	0001	1	1	0101	2	1	1001	3	1	1101
0	2	0010	1	2	0110	2	2	1010	3	2	1110
0	3	0011	1	3	0111	2	3	1011	3	3	1111

Classifier systems employ a "don't care" (#) symbol in the syntax of their conditions and thus permit the formation of generalizations. For example, the condition 00## matches four inputs: 0000,

0001, 0010 and 0011. In this input, we ignore the value of attribute A2 or we can get more general classifiers when using #'s symbol.

For encoding action or class of input problem, XCS usually uses binary code for action presentation. For instance, one classification problem has four classes: 0, 1, 2 and 3, we can encode these actions as following table:

Table 3.2 Action Encoding

Class	Binary Code
0	00
1	01
2	10
3	11

Step 3: *Creating Match set [M]*. At each step, an input from environment used to build a match set [M], which is formed by all the classifiers in [P] whose conditions are satisfied by the input example. The binary input string $S = \{0, 1\}^l$ is matched with the conditions that specify the attributes it requires to be match set. If the condition part is satisfied by the current problem instance, the classifier is said to *match*. Table 3.3 shows an example of a potential problem instance and all conditions that would match this problem instance.

Table 3.3 All classifier conditions whose specified attributes are identical to the corresponding values in the problem instance match the current problem instance. The more general a condition part, the more problem instances it matches.

Instance	Matching conditions		Matching problem instances	Condition
1001	1001		1001	1001
	100# 10#1 1#01 #001		1001 1000	100#
	10## 1#0# #00# 1##1		1011 1010 1001 1000	10##
	#0#1 ##01 ###1 ##0#		1111 1101 ... 0011 0001	###1
	#0## 1###		1111 1110 ... 0001 0000	####
	####			

If the number of actions represented in [M] is less than a threshold θ_{mna}, then covering is triggered to create a new classifier that matches the current input and has a random action from those not present in [M].

Step 4: *Calculating Prediction Array [PA]*. From the resulting match set, an action must be selected and sent to the environment. For this purpose, a payoff prediction $P(a_i)$ is computed for each action a_i in [M] as following equation:

$$P(a_i) = \frac{\sum p_i * F_i}{\sum F_i} \tag{3.10}$$

$P(a_i)$ estimates the payoff that the system will receive if action a_i is chosen. It is computed as fitness-weighted average of the predictions of all classifiers proposing that action. For example, in figure 3.5, the system has 4 actions: 00, 01, 10 and 11, but Match set [M] contains only 2 actions: 01 and 11, so P(a = 00) = P(a = 10) = 0 (nil) while P(a = 01) and P(a = 11) are calculated as following:

$$P(a=01) = \frac{\sum P(a=01) * F(a=01)}{\sum F(a=01)} = \frac{43*99+27*3}{99+3} = 42.5$$

$$P(a=11) = \frac{\sum P(a=11) * F(a=11)}{\sum F(a=11)} = \frac{14*52+18*92}{52+92} = 16.6$$

Step 5: *Creating Action set [A]*. Many action-selection methods are possible. The system may simply pick the action with the largest prediction; for brevity, we shall call this deterministic action selection. In figure 3.5, action 01 is chosen based on deterministic action scheme. Alternatively, the action may be selected probabilistically, with the probability of selection proportional to $P(a_i)$; we shall call this roulette-wheel action selection. In some cases the action may be selected completely at random (from actions with non-null predictions), ignoring the $P(a_i)$. There are of course additional schemes. The chosen action determines the action set [A] which consists of all the classifiers in [M] advocating this action.

In classification, the winning action is usually selected using either pure explore mode or pure exploit mode. In pure explore mode, the action is selected randomly. This makes sense during training, i.e., when the system is learning the consequences of all possible actions for a given input. In pure exploit mode, the action is selected deterministically according to the highest prediction. This is used in test, that is, when the system classifies new unseen instances based on the knowledge it has acquired.

Step 6: *Decoding the output by Effectors*. Decoding is the opposite process - the conversion of an encoded format back into the original sequence of characters. By conversing table 3.2, we have Action Decoding table as following:

Table 3.4 Action Decoding

Binary Code	Class
00	0
01	1
10	2
11	3

Step 7: *Get reward and update parameters*. The technique update parameters used in XCS is closer conceptually and algorithmically to Q-learning [66], a theoretically grounded and widely used technique in contemporary reinforcement learning. This step is described detail in section 3.4.

Step 8: *Using a niche GA to generate new rules*. This is the final step in the performance of XCS, also the most complex. By using accuracy-based fitness and niche GA, two equally accurate classifiers where one matches a subset of the states matched by the other, the more general classifier will win out because it has more reproductive opportunities. We describe detail this step in section 3.5

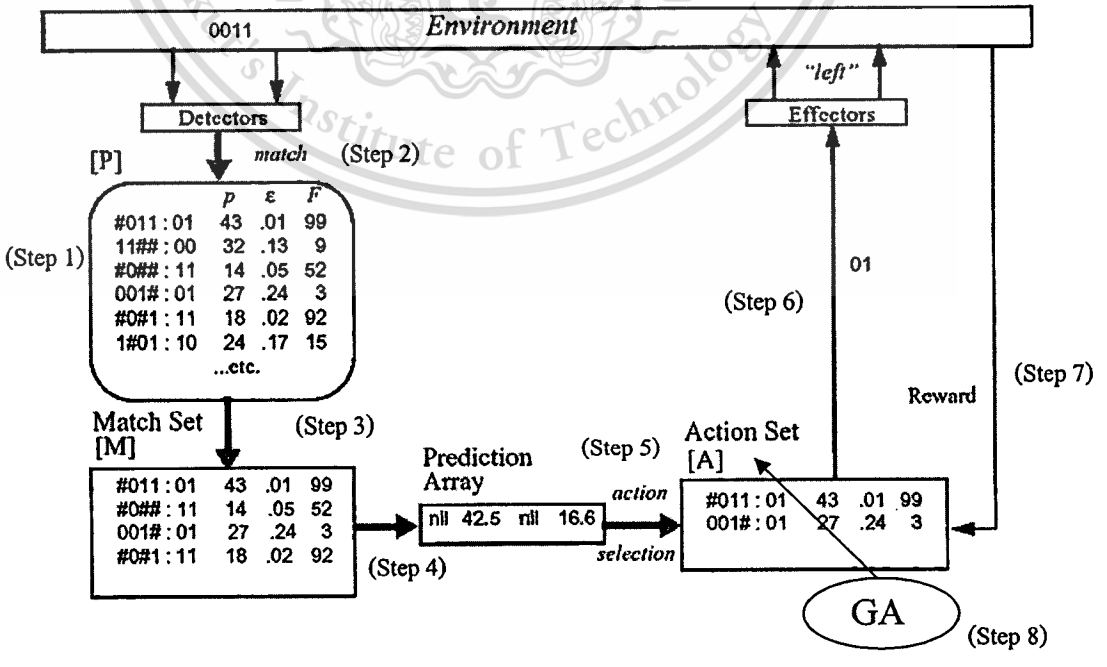


Figure 3.5 Detail schematic illustration of XCS

3.4 Reinforcement Component

XCS's reinforcement component consists of updating the prediction, prediction error and fitness of classifiers in the action set [A], as shown in Figure 3.4.

In classification problems, classifier parameters are updated with respect to the immediate feedback R in the current action set [A]. In multi-step problems, all classifiers in [A] are updated with respect to the immediate reward R plus the estimated discounted future reward as follows:

$$Q = \max_{x \in A} P^{t+1}(x) \tag{3.11}$$

where the $(t + 1)$ term refers to the prediction array in the consequent learning iteration $t + 1$.

The prediction p_j of the classifiers in [A] using standard Widrow-Hoff delta rule [21] is updated by:

$$p_j \leftarrow p_j + \beta(\rho - p_j) \tag{3.12}$$

where $\rho = R$ in classification problems and $\rho = R + \gamma Q$ in multi-step problems. Parameter $\beta \in [0, 1]$ denotes the learning rate influencing accuracy and adaptation of the moving average prediction. Similar to the learning rate dependence in reinforcement learning (see section 3.1.3), a higher learning rate β results in less history dependence and thus faster adaptation but also higher variance if different reward values may be received.

Next, prediction error ϵ of each classifier in [A] is updated by:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta(|\rho - R| - \varepsilon_j) \tag{3.13}$$

with the same notation as in the update of p_j. Note how XCS essentially applies a Q-learning update. However, Q-values are not approximated by a tabular entry but by a collection of rules expressed in the prediction array $P(A)$.

The fitness value of each classifier in [A] is updated with respect to its current scaled relative accuracy κ', which is derived from the current reward prediction error ϵ as follows:

$$\kappa_j = \begin{cases} \alpha\left(\dfrac{\varepsilon_j}{\varepsilon_0}\right)^{-\nu}, & \varepsilon_j \geq \varepsilon_0 \\ 1 & otherwise \end{cases} \tag{3.14}$$

$$\kappa' = \frac{\kappa \cdot num}{\sum_{cl \in [A]} cl.\kappa \cdot cl.num} \tag{3.15}$$

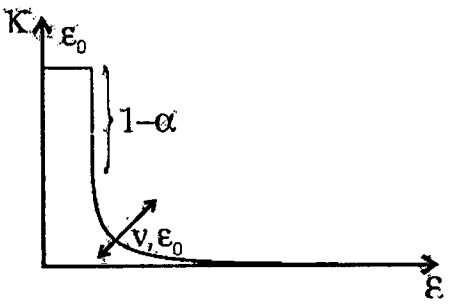


Figure 3.6 The scaling of accuracy K is crucial for successful selection of genetic algorithm

Essentially, K measures the current absolute accuracy of a classifier using a power function with exponent ν to further prefer low error classifiers. Threshold ϵ_0 denotes a threshold of maximal error tolerance. That is, classifiers whose error estimate ϵ drops below threshold ϵ_0 are considered accurate. The derivation of accuracy K with respect to ϵ is illustrated in Figure 3.5. The relative accuracy K' then reflects the relative accuracy with respect to the other classifiers in the current action set. In effect, each classifier in [A] competes for a limited fitness resource that is distributed dependent on $\kappa \cdot num$.

Finally, fitness estimate F is updated with respect to the current action set relative accuracy K' as follows:

$$F \leftarrow F + \beta(\kappa' - F) \tag{3.16}$$

In effect, fitness reflects the moving average, set-relative accuracy of a classifier. As before, β controls the sensitivity of the fitness.

3.5 Discovery Component

The rule discovery system is the third and final of XCS's major subsystems. It is responsible for the generation of new rules and the removal of old rules. Rules may be created in one of three ways: 1) as part of a random initial population, 2) by covering, and 3) by the genetic algorithm. Each of these is discussed in turn, along with the selection of rules for deletion, subset deletion, and the initialization of newly created rules.

Random Initial Populations

One approach to initializing the rule population in XCS is to fill it with random rules at the outset. N rules are generated, each with a random condition and random action. Each bit in a rule's condition is a # with probability $P_{\#}$, otherwise it is equally likely to be a 0 or 1. This produces a

random initial population consisting of more and less general hypotheses. In practice, however, covering has almost always been used instead.

Covering

The alternative to creating a random initial population is to start with an empty initial population and create matching rules as needed through covering. When a classifier is created through covering, its condition is a copy of the current environmental input and it is given a randomly chosen action. Each character in the condition is then mutated with probability $P_{\#}$ into a #. This procedure allows the newly generated rule to generalize, but guarantees that it matches the current input. The covering classifier is then inserted into [P] and if the population size limit is exceeded, a classifier is deleted using the normal method. When covering is used to initialize a population, it normally occurs only a few times at the beginning of the run. This is sufficient to start the system, after which the genetic algorithm takes over.

The Niche Genetic Algorithm

Most LCSs employ a *panmictic* GA, that is, all classifiers are eligible for selection as parents. XCS, however, employs a niche GA, in which only a subset of classifiers participates in the process of selecting parents. In other words, the niche GA is a form of restricted mating scheme. The niche GA was first introduced by Booker [71] as a way of focusing genetic search. Booker reasoned that since the rules in a match set were related (in that they apply to the same input), crossover among them would be more productive than crossover between unrelated rules. In XCS the GA originally operated upon [M] [8] but was moved to [A] when Wilson realized this was more effective [10].

Selection: The probability of a classifier being selected for reproduction is proportional to its fitness [8]. In practice, Wilson and others have used roulette wheel selection.

Genetic Operators: Copies of the parent classifiers are generated and then transformed using standard genetic algorithm operators. One point crossover occurs with probability χ per pair of chromosomes (i.e., per pair of bit-string) and point mutation occurs with probability μ per allele (i.e., per bit) [8]. Crossover only occurs in the conditions, but mutation occurs in both the condition and action [9].

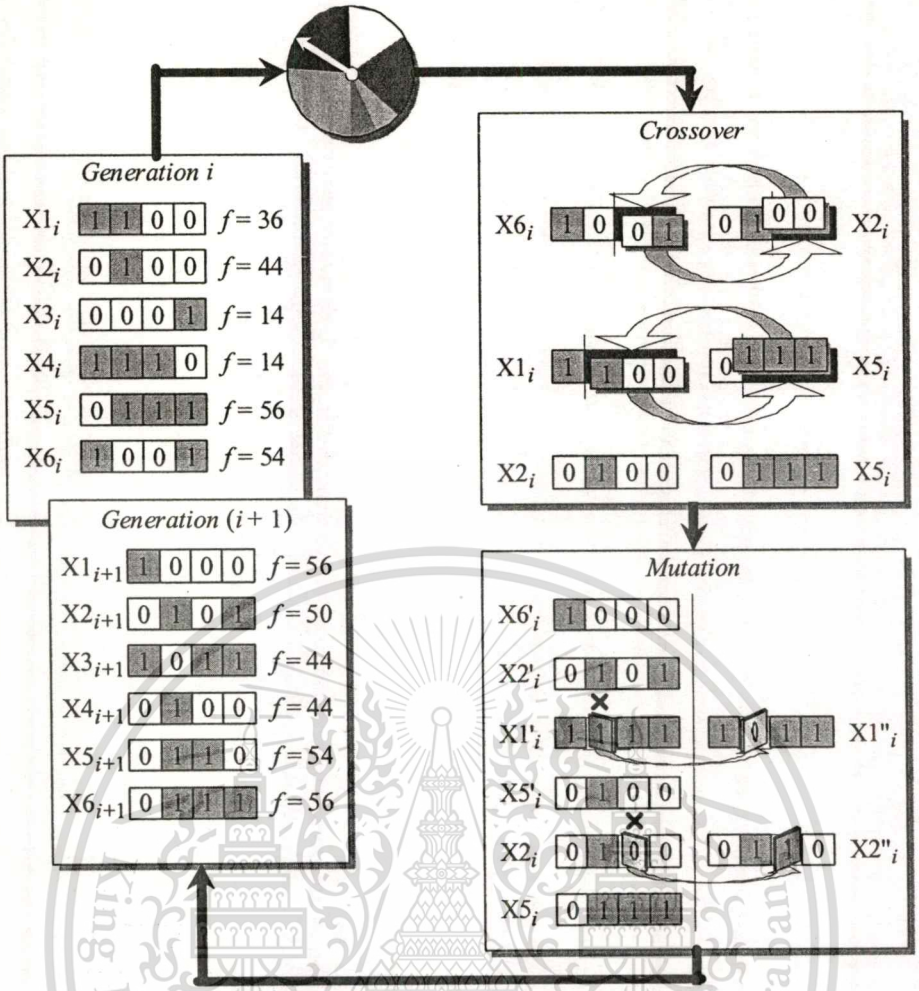


Figure 3.7 The Genetic Algorithm cycle

Deletion: If [P] contains less than N members, the copies are inserted into the population and no compensating deletion occurs. Otherwise, two classifiers are deleted stochastically from [P] to make room. There are two methods of selecting the classifiers to be deleted:

1. Every classifier keeps an estimate of the size of the action sets in which it occurs. The estimate is updated every time the classifier takes part in an [A], using the MAM technique with rate β. A classifier's deletion probability is set proportional to the action set size estimate, which tends to make all action sets have about the same size, so that classifier resources are allocated more or less equally to all niches (action sets). This deletion technique is similar to one introduced by Booker [71] for the same purpose.

2. A classifier's deletion probability is as in (1), except if its fitness is less than a small fraction δ of the average fitness of population. Then the probability from (1) is multiplied by the average/mean fitness divided by the classifier's fitness. If for example δ is 0.1, the result is to delete such low-fitness classifiers with a probability 10 times that of the others.

3.6 Summary

This chapter has given detailed specification of an accuracy-based classifier system called XCS, which add to the classical Reinforcement Learning framework the possibility of representing the state as a vector of attributes and finding a compact expression of the representation so induced. Their formalism conveys a nice interaction between learning and evolution, which makes them a class of particularly rich systems, at the intersection of several research domains. As a result, they profit from the accumulated extensions of these domains.

In effect, XCS is designed to evolve a *complete*, *maximally accurate*, and *maximally general problem solution* represented by a population of classifiers. Each classifier defines a problem subspace in which it predicts the corresponding Q-value accurately.

While the base XCS system introduced in this chapter learns a generalized representation of the underlying Q-value function, it should be noted that XCS is not limited to approximating Q-functions. In fact, XCS can be modified yielding a general, online learning *function approximation technique* [22]. Due the rule-based structure, XCS is designed to partition its space dependent on the representation of classifier conditions. Each classifier then approximates, or predicts, the actual function value in its defined subspace. In the base XCS, conditions define hyper-cubes as subspaces and predict constant reward values-consequently applying piece-wise constant function approximation. Wilson [22] experimented with piecewise linear approximations. Lanzi [16] experimented with S-expressions for conditions. Dependent on the problem structure at hand, other condition representations may be applied. Similarly, other prediction methods are imaginable.

This chapter introduces the XCS classifier system. We provide a concise description of problem representation and all fundamental mechanisms. The algorithmic description found in Appendix A provides an exact description of all problem notation and parameters in XCS facilitating the implementation of the system.

Chapter 4

Guidelines for XCS in Imbalance Problems

This chapter analyzes effects of imbalance problems to XCS's performance on population initialization and on the creation and deletion of classifiers of the minority class. We show that XCS with standard parameter settings is quite robust to class imbalances. For high class imbalances, XCS suffers from biases toward the majority class. We provide guidelines to improve performances of XCS by setting parameters appropriately. Specially, the inheritance procedure of classifiers' parameters mutation and subsumption are analyzed and improvements in XCS's mechanisms are proposed to effectively and efficiently handle imbalance problems. To investigate these points, we undertake a general analysis of all evolutionary pressures of XCS in classification. Evolutionary pressures can be regarded as evolutionary biases that influence or bias learning in XCS.

4.1 XCS in Classification

As introduced in the previous chapter, XCS represents the knowledge extracted from the problem in a set of rules. From [11], XCS is extended to represent continuous state spaces with using hyper-rectangle codification. In the hyper-rectangle representation, the condition of the rule is represented as a concatenation of interval predicates of type $[l_i, u_i]$, where l_i and u_i are the lower and upper extremes of the hyper-rectangle in each dimension i, $l_i \leq u_i$, $1 \leq i \leq n$ and n is the number of attributes. A condition $([l_1, u_1], [l_2, u_2], \ldots [l_n, u_n])$ matches an input example $(x_1, x_2, \ldots, x_n)$ if and only if $\forall\, i=1...n$, $l_i \leq x_i \leq u_i$. XCSI [17] introduced this type of representation for integer attributes, where l_i and u_i were integers. We can see that this representation can be easily adapted to deal with real attributes, by coding l_i and u_i as real.

The hyper-rectangle representation is used to analyze the properties of the decision boundaries evolved by XCS. For this purpose, we will use a graphical analysis that shows the evolved rules and the decision boundaries. To facilitate the graphical display, we restrict the analysis to two-dimensional data.

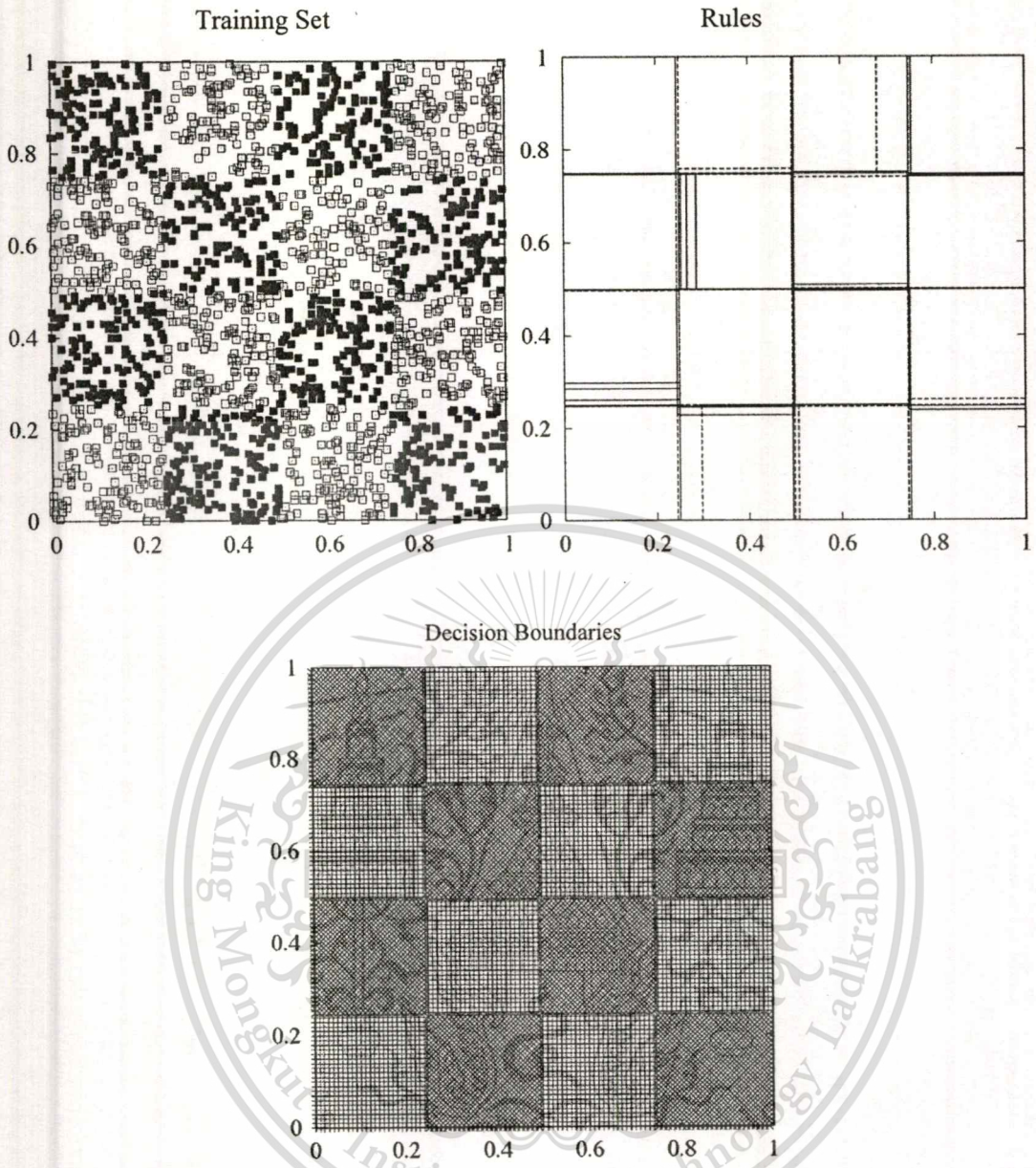


Figure 4.1 Checkerboard problem. The graphs show the training points, the experienced and consistently correct rules evolved by XCS, and the decision boundaries obtained by XCS, respectively.

We select checkerboard domain, a well-known problem in classification, to analyze performance of XCS. The domain has two real attributes ranging in the interval [0,1], and two classes distributed in alternating squares drawing a checkerboard in the feature space. Figure 4.1 shows the checkerboard problem along with the result where (a) is the training set, (b) is the rule-set evolved by XCS and (c) is the classification boundaries determined by XCS. Plot (a) shows the training points in the two-dimensional space depicted differently depending on the class to which they belong. The rule-set evolved by XCS shown in plot (b). The rules are depicted using different line types, depending on

the class they predict, i.e., a solid line is used for rules predicting class 0, a dashed line for rules predicting class 1. Plot (c) represents the classification boundaries induced by XCS. This test is done by using XCS in pure exploit mode, and therefore all the rules evolved by XCS participate in the process.

It would appear, therefore, that the application of a, possibly modified, form of XCS to data mining has the potential to address the significant classification problems. This hypothesis was first identified and examined empirically by [4]. In this work XCS is applied to the Monk data sets, and it is shown that where an appropriate representation is used XCS can produce a classification rate that is as accurate as or better than that provided by the closest competitive Machine Learning algorithm, even within a noisy problem domain. Works by [25] illustrated the competitiveness of XCS on a range of further data mining tasks drawn from the UCI Repository. [12] applied XCS to a data mining problem where the encoding used for the classifiers (a conjunction of range values) was oblique to the representation required. It was demonstrated that even though the encoding was not appropriate, XCS would identify and preserve the necessarily large population of optimally general rules.

Furthermore, in [24], the performance of XCS is analyzed and tested on 30 real datasets which include different characteristics that may imply a factor of complexity for learning systems, such as high dimensionality, small number of available instances, missing attributes, real-valued attributes, high number of classes, etc. The performance of XCS is compared to a variety of learning algorithms: ZeroR, nearest neighbor algorithms: IB1 and IBk, statistical method: Naïve Bayes, induction trees: C4.5, combination of decision trees and the separate-and-conquer rule learning strategy: PART and Support Vector Classifier System: SMO. The experimental results show that the overall performance of XCS is comparable to the other well-known learning algorithms or XCS is completely suitable for solving classification problems.

4.2 XCS on Imbalance Problems

This section analyses effects of imbalance datasets to the performance of XCS. For this purpose, we run XCS with 11-MUX problem for different imbalance levels and seven UCI imbalance datasets [67] (table 4.1) with the number in the parentheses indicates the target class we chose as the positive and all the other classes are regarded as negative. The more detail information of these datasets is presented in Appendix C.2.

Table 4.1 UCI dataset description

DATASET	(+) Inst.	(-) Inst.	*ir*	#Attr.
Echocardiogram	44	88	1:1.2	12
Breast-w	241	458	1:2	9
Wine3	47	129	1:3	8
Glass7	29	185	1:6	9
Vowel3	90	900	1:10	10
Hypothyroid1	93	3679	1:40	21
Abalone19	32	4145	1:130	8

(+) Inst. = number of positive instances, (-) Inst. = number of negative instances, *ir* = imbalance ratio, #Attr = number of attributes.

These Boolean functions are defined for binary strings of length $l=k+2^k$ under which the first k bits index into the remaining 2^k bits, returning the value of the indexed bit. For example, in the 6-multiplexer (l=6), the value for the input string 100010 is 1, since the "address", 10, indexes bit 2 of the remaining four bits, we do not care the other bits (note that we begin counting from 0 or 00 indexes bit 1 and 01 indexes bit 2 of the remaining four bits). The 64-string input space can be covered by exactly eight such maximally general classifiers, each having three #s (don't care symbol) in its condition so it matches eight strings as described in Table 4.2. The more detail information of multiplexer problem is presented in Appendix C.1.

Table 4.2 The 6-multiplexer problem

Majority class	Minority class
000###:0	001###:1
01#0##:0	01#1##:1
10##0#:0	10##1#:1
11###0:0	11###1:1

In disjunctive normal form, the 6-multiplexer is fairly complicated:

$$F_6 = \bar{x}_0\bar{x}_1x_2 + \bar{x}_0x_1x_3 + x_0\bar{x}_1x_4 + x_0x_1x_5 \qquad (4.1)$$

The imbalance complexity is controlled by means of the probability of sampling an instance of the minority class P_{smin}. At each learning interaction, the environment chooses an instance randomly. If it is a majority class instance is passed to XCS with probability P_{smin}. If it is not accepted, a new instance is randomly sampled which undergoes the same decision process. Specially, the minority class is sampled with probability $P_{min}=1/(1+ir)$ and the majority class sampled with probability $P_{maj}=ir/(1+ir)$. Another definition is also used that is imbalance level i where $i=log_2(ir)$. Thus, level i=0 represents the balanced multiplexer. For level $i \geq 1$, there are half of the minority class instances with respect to i-1.

We ran XCS with the following parameter settings (as introduced in [9]): N=1000, β=0.2, α=0.1, ϵ_0=1, ν=5, θ_{GA}=25, χ=0.8, μ=0.04, θ_{del}=20, δ=0.1, θ_{sub}=200, $P_{\#}$=0.6. The meaning of parameters in XCS is provided detail in Appendix A.1. Table 4.3 and table 4.4 show performances of XCS on standard parameter settings, called standard XCS or XCS shortly to distinguish to our approach, Cost-sensitive XCS. All experiments presented are averaged over 10 independent runs of 10-fold cross-validation technique [2]. As expected, when imbalance level increases, the performance of XCS decreases. In 11-MUX problem, XCS's *F1-measure* raises to value 1 for imbalance levels up to i=4. For i=5, *F1-measure* is 0.8 and *F1-measure* is 0.2 for i=6. It means that for lower imbalance levels, XCS classifies correctly, for imbalance level i=5 or higher, XCS begins to find difficulties classifying the minority class examples. XCS also encounters this problem in UCI datasets as results shown in table 4.4. XCS's *F1-measure* reduces from 0.980 to 0 as imbalance ratio increases from 1:1.2 (*Echocardiogram* data) to 1:130 (*Abalone* data).

We analyzed possible reasons of this problem by looking at the classifier population of XCS, we found that when imbalance level is high, the population mainly consists of the two most *overgeneral* classifiers which contain only #'s symbols (don't care symbols) at their condition part. They cover accurately all the instances of the majority class and cover wrongly the instances of minority class, so, XCS has poor performance for the minority class or *F1-measure* is low. In the next chapter, we proposed some methods based on reward setting and ensemble learning to overcome this problem.

Table 4.3 Standard XCS's performance on 11-MUX

i	p	r	F1	g
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	0.99±.01	1	0.99±.01	0.99±.01
5	0.8±.002	0.8±.001	0.8±.015	0.8±.015
6	0.2±.01	0.19±.02	0.2±.01	0.2±.01
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

Table 4.4 Standard XCS's performance on UCI datasets

dataset	p	r	F1	g
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.986±.034
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.964±.020
Wine3	0.971±.090	0.748±.311	0.801±.230	0.831±.193
Glass7	0.900±.161	0.867±.172	0.873±.142	0.919±.099
Vowel3	0.673±.326	0.522±.119	0.590±.233	0.557±.073
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.269±.367
Abalone19	0±.00	0±.00	0±.00	0±.00

(the values following "±" are standard deviation)

4.3 Population Initialization

In this section we analyze the effect of class imbalances on the covering operator, which is responsible for initializing the population. As described in the XCS introduction, covering creates classifiers given a problem instance that is not matched by at least one classifier for each possible classification. Recall, in XCS, reproduction occurs in action sets and deletion occurs in population sets. Since the population is of fixed size *N*, if the population is already filled up with classifiers, other

classifiers are deleted to make space for the new covering classifiers. In the beginning of a run, with a population of classifiers that have a very low experience, the fitness F as well as the action set size estimate *as* of these classifiers is basically meaningless. Consequently, the deletion method chooses classifiers for deletion at random. Dependent on the specificity of classifiers generated by covering (determined by the parameter *don't care probability* $P_{\#}$) the average specificity in the population may be too high to cover the whole problem space. Thus, it may happen that the population fills up with over-specialized classifiers and a *covering-random deletion* cycle continues forever. The issue basically can only happen if parameter $P_{\#}$ is set too low so that the initial specificity in the population is too high.

In our analysis, we assume that $P_{\#}$ is appropriately set to a high value to guarantee the covering challenge. As the imbalance ratio *ir* increases, less instances of the minority class will be sampled during the first iterations. Thus, for high class imbalances, covering will be activated on examples of any class other than the minority class, and so, all classifiers' conditions, regardless the action they advocate, will be mainly a generalization of instances of any class other than the minority class. However, the covering operator should be applied on enough instances of the minority class to initially supply the population with sufficient classifiers of the minority class. In the following, we derive a lower bound of the probability of activating covering when the first instance of the minority class is sampled.

The situation can be formalized by determining the probability *P(cover)* that an input is covered by at least one classifier in a randomly generated population, as a function of the specificity in the population $s([P])$ (initially equal to $1-P_{\#}$) which is the rate between the number of specific bits (0 and 1) and the total bits (0, 1 and #) of all classifiers in the population P. First, according to [13] the probability P(match) that a random classifier matches an input

$$P(match) = \left(\frac{s([P])}{2} + (1 - s(P))\right)^{l} = \left(\frac{2 - s([P])}{2}\right)^{l} \qquad (4.1)$$

where l is the length of the input string. For example, an input 110100 has l=6. Then, the probability P(no match in [P]) that no classifier in a randomly generated population matches the current input string is computed as:

$$P(\text{no match in } [P]) = (1 - P(match))^{N} \qquad (4.3)$$

where N denotes the size of the population. Finally, the probability P(cover) that at least one classifier matches the current input string is derived from the previous equations as

$$\begin{aligned} P(\text{cover}) &= 1 - P(\text{no match in [P]}) \\ &= 1 - (1 - P(\text{match}))^N \\ &= 1 - \left(1 - \left(\frac{2 - s([P])}{2}\right)^l\right)^N \end{aligned} \tag{4.4}$$

In the imbalance problem with a given imbalance ratio ir, let us assume the worst case where a) XCS receives ir instances of the other classes before receiving the first instance of the minority class, and b) the covering operator is triggered for each instance supplying n classifiers per instance (where n is the number of classes). The probability that XCS covers the first instance of the minority class is:

$$P(\text{cover\_on\_minority}) = 1 - \left[1 - \frac{1}{n}\left(\frac{2 - s([P])}{2}\right)^l\right]^{n \cdot ir} \tag{4.5}$$

The equation supposes that $N > n*ir$, i.e., that XCS will not delete any classifier during the first ir iterations. Given a fixed l and $s[P]$, the term in brackets in the right hand of the equation decreases exponentially as the imbalance ratio increases; thus, the probability of covering minority class instances tends to one exponentially with the imbalance ratio.

Provided that the probability of activating covering is 1 - $P(cover)$, and recognizing that $(1 - r/n)^n \approx e^{-r}$, we obtain that the probability of activating covering having sampled a minority class instance is:

$$\begin{aligned} P(\text{activate cov. on. min.}) &= 1 - P(cover\_on\_minority) \\ &= \left[1 - \frac{1}{n}\left(\frac{2 - s([P])}{2}\right)^l\right]^{n \cdot ir} \\ &\approx e^{-ir \cdot \left(1 - \frac{s([P])}{2}\right)^l} \\ &\approx e^{-ir \cdot \left(1 - \frac{s([P]) \cdot l}{2 \cdot l}\right)^l} \\ &\approx e^{-ir \cdot e^{-\frac{ls[P]}{2}}} \end{aligned} \tag{4.6}$$

which decreases exponentially with the imbalance ratio and, in a higher degree, with the condition length and the initial specificity. Figure 4.2 depicts the equation for $l = 20$, $n = 2$ and different initial

specificities, showing that the probability of activating covering on the first sampled instance of the minority class decreases exponentially with the imbalance ratio.

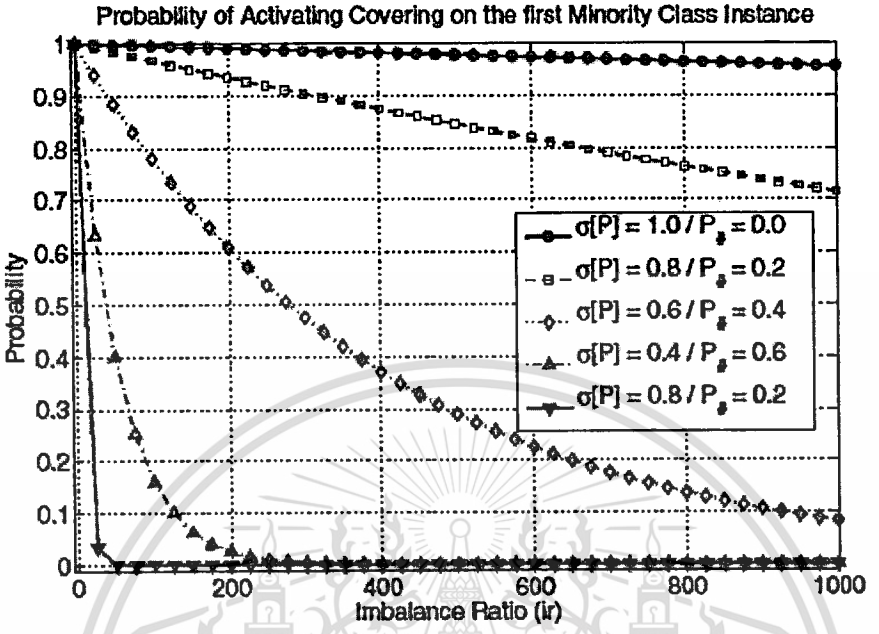


Figure 4.2 Probability of activating covering on a minority class instance given a certain specificity s([P]) and the imbalance ratio ir. The curves have been drawn from formula 4-5 with different specificity s([P]) and setting l=20.

The analysis made above shows up that the covering operator fails to supply classifiers representing correct schemas of the minority class for moderate and high imbalance ratios. In classification tasks, dynamic re-sampling techniques could be applied to overcome this problem [41]. However, in this chapter we are interested in the intrinsic capabilities of XCS to discover the minority class, and so, re-sampling techniques will be remained to consider in Chapter 6. Consequently, XCS will start the search with a high general population that contains few schemas of the minority class. So, the genetic search will be the main responsible for obtaining the first correct classifiers of the minority class. In the next section we analyze the probabilities that the GA generates new classifiers that represent starved niches.

4.4 Generation of Correct Classifiers of the Minority Class

In this section, we analyze the probability of generating correct classifiers of the minority class assuming that covering has not provided any classifier representing any schema of the minority

class. Since mutation is the primary operator for exploring new regions of the input space, we propose a simplified model where only the effect of mutation is considered. We also assume low values of the probability of mutation μ ($\mu < 0.5$) as it is usual in practice.

Accurate classifiers of the minority class can be obtained while exploring any class in the feature space. In the following, we derive the probabilities of generating a correct classifier of the minority class when sampling instances of any class, and gather them together deriving the probability of generating new correct classifiers of the minority class.

4.4.1 Sampling Minority Class Instances

A minority class instance is sampled with probability $P(\text{inst}_{\text{min}}) = 1/(1 + ir)$. As XCS chooses the class to be explored randomly, two possible scenarios are possible: 1) every $1/n$ times, XCS activates high rewarded niches of the minority class, and 2) the other $(n - 1)/n$ times; XCS explores low rewarded niches of another class. Next, we derive the probabilities of obtaining a correct classifier of the minority class in both cases.

First, if the GA is triggered on a niche of the minority class (assuming that there are not correct classifiers in the niche), it will generate a correct classifier of the minority class if all the bits of the schema are set to their correct value, and the class of the classifier is not flipped. We consider the worst case, that is, that all the bits of the schema need to be changed. That gives the following lower bound on the probability of generating a new correct classifier of the minority class:

$$P(cl_{\text{min}} \mid inst_{\text{min}} \Lambda\, niche_{\text{min}}) = \left(\frac{\mu}{2}\right)^{k_m} \cdot (1-\mu) \qquad (4.7)$$

with μ is the probability of mutation and k_m is the order of the schema which is the number of defined bits (non-asterisks) in a schema. The schema H = 1****0, for example, has two defined bits, and thus its order is 2. That is, the formula defines a parabola on the values of μ; for k_m=1, the probability is maximized at μ = 0.5. However, this high value of μ may introduce too much disruption in the genetic search. When we increase the order of the schema k_m, the probability of obtaining a correct classifier of the minority class is decreased.

Second, if the GA is activated on a niche of any class other than the minority class, not only all bits of the schema have to be correctly set, but also the class has to be mutated to the minority class. Thus, the lower bound on the probability of GA activating on a niche of any class other than the minority class to generate a minority class classifier is:

$$P(cl_{\text{min}} \mid inst_{\text{min}} \Lambda \neg niche_{\text{min}}) = \left(\frac{\mu}{2}\right)^{k_m} \cdot \frac{\mu}{n-1} \qquad (6\text{-}8)$$

As above, the probability of generating a new correct classifier of the minority class depends on the mutation probability μ and the order of the schema k_m. Moreover, it also depends inversely on the number of classes of the problem.

4.4.2 Sampling Instances of Other Classes

XCS can also create correct classifiers of the minority class when sampling instances that do not belong to the minority class. The probability of sampling these instances is $P(\neg instmin) = ir/(1 + ir)$. Two possible scenarios are possible when an instance of any class other than the minority class is sampled: 1) a niche of the minority class is triggered, and 2) a niche of another class is triggered. The first case implies that all the bits of the schema must be specified. The second case also requires that the class is flipped to the minority class. Thus, the derived probabilities are:

$$P(cl_{\min} \mid \neg inst_{\min} \wedge niche_{\min}) = \left(\frac{\mu}{2}\right)^{k_m} \cdot (1-\mu) \tag{4.9}$$

$$P(cl_{\min} \mid \neg inst_{\min} \wedge \neg niche_{\min}) = \left(\frac{\mu}{2}\right)^{k_m} \cdot \frac{\mu}{n-1} \tag{4.10}$$

These probabilities are equivalent to the ones obtained in formulas (4.7) and (4.8) respectively. They are guided by the probability of mutation μ and the order of the schema k_m.

4.4.3 Time to Create Correct Classifiers of the Minority Class

Given the derived sampling probabilities and lower bounds of formulas from (4.7) to (4.10), we analyze the minimum time required to generate the first correct representative of the minority class. We assume that the genetic event is always applied ($\theta_{GA} = 0$) and only one classifier is created by the effect of mutation. Under these circumstances, the probability of generating a correct classifier of the minority class is the sum of the following probabilities:

- The probability p_1 of generating a minority class classifier when sampling a minority class instance. That is, $p_1 = \frac{1}{1+ir} \cdot \frac{1}{n}\left(\frac{\mu}{2}\right)^{k_m}$
- The probability p_2 of generating a minority class classifier when sampling an instance of any class other than the minority class. That is, $p_2 = \frac{ir}{1+ir} \cdot \frac{1}{n}\left(\frac{\mu}{2}\right)^{k_m}$

The time required to discover the first representative is derived from the addition of both probabilities:

$$t(cl_{\min}) = \frac{1}{P(cl_{\min})} = \frac{1}{p_1 + p_2} = n\left(\frac{2}{\mu}\right)^{k_m} \quad (4.11)$$

which depends linearly on the number of classes and exponentially on the order of the schema, but does not depend on the imbalance ratio.

Thus, even though covering fails to provide classifiers representing schemas of the minority class, XCS will be able to generate the first correct classifiers of the minority class independently of the imbalance ratio. In the following, we derive the time until the deletion of these classifiers. With both the generation and deletion time, we calculate the minimum population size to maintain these classifiers and ensure the growth of the best representatives of the minority class.

4.5 Deletion Time of Minority Class Classifiers

XCS deletes classifiers depending on their action set size as and their fitness. Having a good estimation of *as*, deletion would remove classifiers that belong to numerous niches and have low fitness; consequently, it would maintain accurate classifiers in starved niches. Nevertheless, overgeneral classifiers, whose presence is numerous in imbalance domains [28], tend to bias the action set size estimate of accurate classifiers that belong to the same action sets. Under these circumstances, deletion may be better approached as a random deletion. As we delete two classifiers every GA application, we obtain that:

$$P(\text{delete cl}_{\min}) = \frac{2}{N} \quad (4.12)$$

From this formula, we derive the time until deletion:

$$\text{t(delete cl}_{\min}) = \frac{N}{2} \quad (4.13)$$

In the following, we use formulas (4.11) and (4.13) to derive the minimum population size that guarantees the discovery, maintenance and growth of starved niches.

4.6 Bounding the Population Size

Herein, we use the formulas of generation and deletion of minority class classifiers to derive two population size bounds. First, the minimum population size to ensure that XCS will be able to create and maintain correct classifiers of the minority class is derived. Then, the population size

bound to ensure the growth of the niches that contain these correct classifiers of the minority class is derived.

4.6.1 Minimum Population Size to Guarantee Representation

Our first concern is to ensure that there would be correct classifiers representing the niches of the minority class. Our sake is to guarantee that, before deleting any classifier of the minority class, another correct classifier of the minority class will be created. Thus, we require that the time until deletion be greater than the time until a correct classifier of the minority class is generated.

$$t(\text{delete } cl_{\min}) > t(cl_{\min}) \tag{4.14}$$

That is:

$$N > 2n\left(\frac{2}{\mu}\right)^{k_m} \tag{4.15}$$

which indicates that, to guarantee that all the minority class niches of the system have at least one correct classifier, the population size have to increase linearly with the number of classes and exponentially with the order of the schema; however, it does not depend on the imbalance ratio.

4.6.2 Population Size Bound to Guarantee Reproductive Opportunities

Above, we discussed the minimum time required to generate the first correct classifiers of the minority class. Now, we are concerned about the requirements to ensure that classifiers of minority class will evolve to better ones.

To ensure the growth of niches of the minority class, we should guarantee that the best classifiers in the niche receive, at least, one genetic opportunity. Otherwise, XCS could be continuously creating and removing classifiers from a niche, but not searching toward better classifiers. As before, we consider $\theta_{GA} = 0$; moreover, we assume that the selection procedure chooses one of the strongest classifiers in the niche. Then, the time required for a minority class classifier to receive a genetic event is inversely proportional to the probability of activation of the niche it belongs to:

$$t(\text{GA } niche_{min}) = n \cdot (1 + ir) \tag{4.16}$$

which depends on the imbalance ratio and the number of classes.

To guarantee that these strong classifiers of the minority class will receive a genetic opportunity before being deleted, we require that:

$$t(\text{delete } niche_{min}) > t(\text{GA } nichemin) \tag{4.17}$$

from which we derive the population size bound:

$$N > 2n \cdot (1 + ir) \tag{4.18}$$

That is, the population size N has to increase linearly with the number of classes and the imbalance ratio to warrant that correct classifiers of the minority class will receive, at least, one genetic event before being deleted.

In this section a theoretical model for learning under class imbalances is derived. The analysis revealed a failure of covering to provide schemas of the minority class for high imbalance ratios, giving the responsibility for generating correct classifiers of the minority class to the GA. Under this scenario, a population size bound is developed that indicated that XCS should be able to create and maintain correct classifiers of the minority class regardless of the imbalance ratio; furthermore, a second bound denoting that the population size should increase linearly with the imbalance ratio to ensure the growth of starved niches is described. In the next section we describe the test problems used to validate the theoretical models developed.

4.7 Summary

In this chapter, we provided theoretical models that analyzed the effects of class imbalances on the covering operator, the creation and deletion of correct classifiers of the minority class, and the number of genetic opportunities that these classifiers receive. In recently, much work based on XCS focuses on data mining, with some excellent results, but there is a need to extend XCS to handle more challenging and real world problems such as imbalance problems. This chapter extends the theory of computational complexity of XCS [14] to imbalance problems, highlighting the impact of learning from class-imbalanced domains on different mechanisms of XCS. In order to achieve an optimal population, XCS has to evolve different niches distributed around the problem subspace. In imbalanced domains, examples of minority class are sampled in a lower frequency than the others; thus, niches activated by these instances are poorly nourished. We are interested in the discovering and maintenance of classifiers belonging to high rewarded niches but poorly nourished; to which we

refer as correct classifiers of the minority class. For that purpose, three main points: population initialization, generation of correct classifiers of the minority class and time to extinction of correct classifiers of the minority class are analyzed. A bound on the population size to ensure that XCS will be able to maintain correct classifiers of the minority class is presented and that these classifiers will receive, at least, one genetic event before being removed. In the analysis, we consider problems that consist of n classes in which one of the classes, addressed as the minority class, is sampled in a lower frequency than the others. Specifically, the minority class is sampled with a probability $1/(1+ir)$. From that, we propose two methods: Cost-sensitive XCS and Sampling Ensemble XCS to solve imbalance problems which are described in the two next chapters. These methods focus on the creation, maintenance and deletion of correct classifiers that advocate the minority class.

Chapter 5

Cost-sensitive XCS in Imbalance Problems

This chapter presents our first proposed method, Cost-sensitive XCS. We begin with foundations of Cost-sensitive Learning, reward function and value function in XCS. After that is how to combine XCS with cost-sensitive learning to solve Imbalance problems. In our approach, the reward value of correctly identifying the positive (rare) class outweighs the value of correctly identifying the common class. This chapter provides guidelines to set reward base on the dataset imbalance ratio and a method to calculate reward online base on the information collected by XCS during training is also proposed. Experimental results on synthetic and real-life datasets show that, with appropriate reward settings, XCS is robust to class imbalances.

5.1 Foundations of Cost-sensitive Learning

In many real-world applications, the costs of different errors are often unequal. For example, in medical diagnosis, the cost of erroneously diagnosing a patient to be healthy may be much bigger than that of mistakenly diagnosing a healthy person as being sick, because the former kind of error may result in the loss of a life. In fact, cost-sensitive learning has already attracted much attention from the machine learning and data mining communities. As it has been stated in the Technological Roadmap of the MLnetII project [51], the inclusion of costs into learning has been regarded as one of the most relevant topics of future machine learning research. During the past years, many cost-sensitive learning methods have been developed [45], [47]. However, although there are many research efforts devoted to making decision trees cost-sensitive [46], [47], [48], only a few studies discuss cost-sensitive learning classifier systems, while usually it is not feasible to apply cost-sensitive decision tree learning methods to learning classifier system directly. For example, the instance-weighting method [48] requires the learning algorithm to accept weighted-examples, which is not a problem for C4.5 decision trees but is difficult for common feed forward learning classifier systems. In this section, we present the background of Cost-sensitive learning and its application to imbalance problems.

5.1.1 Costs versus Benefits

There are two methods to create a confusion matrix: cost matrix and benefit matrix. When thinking in terms of costs, it is easy to posit a cost matrix that is logically contradictory because not all entries in the matrix are measured from the same baseline. For example, consider the so-called German credit dataset that was published as part of the Statlog project [52]. The cost matrix given with this dataset is as follows:

Table 5.1 Cost matrix

	Actual bad	Actual good
Predict bad	0	1
Predict good	5	0

Here examples are people who apply for a loan from a bank. "Actual good" means that a customer would repay a loan while "actual bad" means that the customer would default. The action associated with "predict bad" is to deny the loan. Hence, the cashflow relative to any baseline associated with this prediction is the same regardless of whether "actual good" or "actual bad" is true. In every economically reasonable cost matrix for this domain, both entries in the "predict bad" row must be the same.

Costs or benefits can be measured against any baseline, but the baseline must be fixed. An opportunity cost is a foregone benefit, i.e. a missed opportunity rather than an actual penalty. It is easy to make the mistake of measuring different opportunity costs against different baselines. For example, the erroneous cost matrix above can be justified informally as follows: "The cost of approving a good customer is zero, and the cost of rejecting a bad customer is zero, because in both cases the correct decision has been made. If a good customer is rejected, the cost is an opportunity cost, the foregone profit of 1. If a bad customer is approved for a loan, the cost is the lost loan principal of 5."

To see concretely that the reasoning in quotes above is incorrect, suppose that the bank has one customer of each of the four types. Clearly the cost matrix above is intended to imply that the net change in the assets of the bank is then -4. Alternatively, suppose that we have four customers who receive loans and repay them. The net change in assets is then +4. Regardless of the baseline, any method of accounting should give a difference of 8 between these scenarios. But with the erroneous cost matrix above, the first scenario gives a total cost of 6, while the second scenario gives a total cost of 0.

In general the amount in some cells of a cost or benefit matrix may not be constant, and may be different for different examples. For example, consider the credit card transactions domain. Here the benefit matrix might be

Table 5.2 Benefit matrix

	Fraudulent	Legitimate
Refuse	\$20	-\$20
Approve	$-x$	$0.02x$

where x is the size of the transaction in dollars. Approving a fraudulent transaction costs the amount of the transaction because the bank is liable for the expenses of fraud. Refusing a legitimate transaction has a non-trivial cost because it annoys a customer. Refusing a fraudulent transaction has a nontrivial benefit because it may prevent further fraud and lead to the arrest of a criminal. Research on cost-sensitive learning and decision-making when costs may be example-dependent is only just beginning [53].

5.1.2 Cost-Sensitive Classification

If the costs are known, they can be incorporated into a financial analysis of the decision-making process. In the two-class case where the confusion matrix is like that of Table 5.1, the two kinds of error – false positives and false negatives – will have different costs; likewise, the two types of correct classification may have different benefits. In the two-class case, costs can be summarized in the form of a 2x2 matrix in which the diagonal elements represent the two types of correct classification and the off-diagonal elements represent the two types of error. In the multiclass case this generalizes to a square matrix whose size is the number of classes, and again the diagonal elements represent the cost of correct classification. Table 5.3 (a) and (b) show default cost matrixes for the two- and three-class cases whose values simply give the number of errors: misclassification costs are all 1.

Taking the cost matrix into account replaces the success rate by the average cost (or, thinking more positively, profit) per decision.

Although we will not do so here, a complete financial analysis of the decision-making process might also take into account the cost of using the machine learning tool – including the cost of gathering the training data – an the cost of using the model, or decision structure, that it produces –

that is, the cost of determining the attributes for the test instances. If all costs are known, and the projected number of the different outcomes in the cost matrix can be estimated – say, using cross-validation – it is straightforward to perform this kind of financial analysis.

Table 5.3 Default cost matrixes: (a) a two-class case and (b) a three-class case

		Predicted class	
		yes	no
Actual class	yes	0	1
	No	1	0

(a)

		Predicted class		
		a	B	c
Actual class	a	0	1	1
	b	1	0	1
	c	1	1	0

(b)

Given a cost matrix, we can calculate the cost of a particular learned model on a given test set just by summing the relevant elements of the cost matrix for the model's prediction for each test instance. Hear, the costs are ignored when making predictions, but taken into account when evaluating them.

If the model outputs the probability associated with each prediction, it can be adjusted to minimize the expected cost of the predictions. Given a set of predicted probabilities for each outcome on a certain test instance, one normally selects the most likely outcome. Instead, the model could predict the class with the smallest expected misclassification cost. For example, suppose in a three-class situation the model assigns the classes a, b and c to a test instance with probabilities p_a, p_b and p_c, an the cost matrix is that in Table 5.3 (b). If it predicts a, the expected cost of the prediction is obtained by multiplying the first column of the matrix, [0, 1, 1], by the probability vector, $[p_a, p_b, p_c]$, yielding $p_b + p_c$ or $1 - p_a$ because the three probabilities sum to 1. Similarly, the costs for predicting the other two classes are $1 - p_b$ and $1 - p_c$. For this cost matrix, choosing the prediction with the lowest expected cost is the same as choosing the one with the greatest probability. For a different cost matrix it might be different.

5.1.3 Making Optimal Decisions

In the two-class case, the optimal prediction is class 1 if and only if the expected cost of this prediction is less than or equal to the expected cost of predicting class 0, i.e. if and only if

$$P(j=0|x)c_{10} + P(j=1|x)c_{11} \leq P(j=0|x)c_{00} + P(j=1|x)c_{01} \quad (5.1)$$

which is equivalent to

$$(1-p)c_{10} + pc_{11} \leq (1-p)c_{00} + pc_{01} \quad (5.2)$$

given $p = P(j=1|x)$. If this inequality is in fact equality, then predicting either class is optimal.

The threshold for making optimal decisions is p^* such that

$$(1-p^*)c_{10} + p^*c_{11} = (1-p^*)c_{00} + p^*c_{01} \quad (5.3)$$

Assuming the reasonableness conditions the optimal prediction is class 1 if and only if $p \geq p^*$. Rearranging the equation for p^* leads to the solution

$$p^* = \frac{c_{10} - c_{00}}{c_{10} - c_{00} + c_{01} - c_{11}} \quad (5.4)$$

assuming the denominator is nonzero, which is implied by the reasonableness conditions. This formula for p* shows that any 2x2 cost matrix has essentially only one degree of freedom from a decision-making perspective, although it has two degrees of freedom from a matrix perspective. The cause of the apparent contradiction is that the optimal decision-making policy is a nonlinear function of the cost matrix.

5.2 Guidelines for Reward Setting

In this section, the foundations of reward function, value function from Reinforcement Learning, a component of XCS, and our reward setting method are introduced. Both synthetic and real datasets are employed to verify the feasibility of our proposal. The experiments show that our suggestion can improve performance of XCS on imbalance problems.

5.2.1 Reward Function

As introduced in Chapter 3, XCS is a combination of Reinforcement Learning and Genetic Algorithm. In reinforcement learning, the purpose of classifiers in single-step tasks or goal of the agent in multi-step tasks is formalized in terms of a special signal, called the *reward*, that passes from the training data to classifiers or from the environment to the agent. The reward is just a single number whose value varies from step to step. Informally, the agent's goal is to maximize the total amount of reward it receives. This means maximizing not just immediate reward, but cumulative reward in the long run. The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning. Although this way of formulating goals might at first appear limiting, in practice it has proven to be flexible and widely applicable.

A reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps perceived states (or state-action pairs) of the environment to a single number, a reward, indicating the intrinsic desirability of the state. The reward function defines what the good and bad events for the agent are. In a biological system, it would not be inappropriate to identify rewards with pleasure and pain. They are the immediate and defining features of the problem faced by the agent. As such, the reward function must necessarily be fixed. It may, however, be used as a basis for changing the policy. For example, if an action selected by the policy is followed by low reward then the policy may be changed to select some other action in that situation in the future. In general, reward functions may also be stochastic.

In XCS, when the environment returns a reward r, it is used to update the parameters of the classifiers in action set $[A]$ following order [8]: *prediction*, *prediction error*, and finally *fitness*. Prediction p is updated with learning rate as follows:

$$p \leftarrow p + \beta(r - p) \tag{5.5}$$

where $\beta\,(0 \le \beta \le 1)$ is the learning rate. Then, the *prediction error* is updated as: $\varepsilon \leftarrow \varepsilon + \beta(|r - p| - \varepsilon)$. Finally, classifier *fitness* is updated in two steps: first, the relative accuracy κ' of the classifiers in [A] is computed; then κ' is used to update the classifier fitness as: $F \leftarrow F + \beta(\kappa' - F)$.

5.2.2 Value Function

Whereas a reward function indicates what is good in an immediate sense, a value function specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward a classifier or an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and

evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run. In decision-making and planning, the derived quantity called value is the one with which we are most concerned. Unfortunately, it is much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations during training time of classifier or over its entire lifetime of an agent. In fact, the most important component of almost all reinforcement learning algorithms is a method for efficiently estimating values. XCS uses Q-learning technique to estimate prediction value p which is as a value function. In the next sub-section, we introduce how to set reward of XCS appropriately to solve imbalance data problems.

5.2.3 Achieving Reward Setting Based on Imbalance Ratio

Because XCS's fitness is based on accuracy, an inverse value of the classifier's average prediction error, it presents a high bias towards the majority class instances. In addition, XCS pays equally reward for correctly identifying of all classes, combined with the generalization tendency of the genetic algorithm. This made XCS to evolve easily *overgeneral* classifiers that cover all the feature space as analysis in the previous chapter. Making difference of reward for each class could help in the reduction of *overgeneral* classifiers. The idea is to assign a greater reward to true positive classifying than to true negative classifying which will improve performance with respect to the positive (rare) class. The hope is that the bias introduced on the training data would be able to output a hypothesis that minimizes the overall costs of the decisions for unknown future examples.

In XCS, the reward computation should be modified. Our suggestion is to set the proportion of true positive reward ($r_{true\_positive}$) and true negative reward ($r_{true\_negative}$) according to the proportion of frequencies between the occurring of negative instances (f_{neg}) and the occurring of positive instances (f_{pos}):

$$\frac{r_{true\_positive}}{r_{true\_negative}} = \frac{f_{neg}}{f_{pos}} = ir \tag{5.6}$$

The ratio f_{neg}/f_{pos} tells us how many examples of the majority class (negative class) are provided with respect to those of the minority class (positive class) which is imbalance ratio (ir), so with 11-MUX problem, $i=log_2(ir)$ we can rewrite equation (5.6) as follows:

$$\frac{r_{true\_positive}}{r_{true\_negative}} = \frac{f_{neg}}{f_{pos}} = ir = 2^i \tag{5.7}$$

If the imbalance ratio increases, we should also increase the proportion $r_{true\_positive}/r_{true\_negative}$. In addition, the population size should be assured that no niche will be loss (as analyzed in chapter 4). It should be increased proportionally with the imbalance ratio *ir*.

Table 5.4 Cost-Sensitive XCS's performance on 11-MUX

i	*p*	*r*	F1	g
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	0.99±.010	0.99±.013	1
6	0.90±.013	0.91±.027	0.91±.014	0.91±.012
7	0.821±.040	0.833±.097	0.824±.062	0.827±.053
8	0.71±.121	0.712±.311	0.7±.155	0.69±.219
9	0.51±.092	0.512±.113	0.500±.015	0.49±.182

Table 5.5 Cost-sensitive XCS's performance on UCI datasets

dataset	*p*	*r*	F1	g
Echocardiogram	0.975±.109	1.00±.00	0.983±.062	0.984±.028
Breast-w	0.946±.029	0.971±.044	0.958±.021	0.970±.020
Wine3	0.931±.147	0.842±.295	0.839±.224	0.875±.185
Glass7	0.913±.274	0.900±.161	0.912±.224	0.924±.133
Vowel3	0.812±.147	0.822±.192	0.815±.101	0.817±.153
Hypothyroid1	0.654±.201	0.649±.193	0.655±.178	0.651±.166
Abalone19	0.500±.198	0.593±.271	0.519±.275	0.553±.203
Average	0.819±.158	0.825±.165	0.812±.155	0.825±.127

We show the performance of Cost-sensitive XCS in table 5.4 and table 5.5. Experimental results reveal that Cost-sensitive XCS performs better than standard XCS on almost imbalance datasets. On 11-MUX problem, the proposed system can reach until *i*=8, which is a notable

improvement with respect to the initial experiments in the previous chapter. Recall that, with standard parameter settings, XCS can reach to $i = 4$ only.

With imbalance datasets having small imbalance ratio as *Echocardiogram, Breast-w, Wine3* and $i \leq 4$ in 11-MUX, both standard XCS and Cost-sensitive XCS work well. On *Glass7* and i=5 of 11-MUX, performances of Cost-sensitive XCS are slightly better than that of standard XCS. On extremely imbalance datasets as *Hypothyroid1*, *Abalone19* and $i \geq 7$ of 11-MUX, performances of Cost-sensitive XCS are significantly better. Furthermore, the tending of evolving *overgeneral* rules has been restrained. The population evolved by Cost-sensitive XCS contains accurately and maximally general rules predicting both the minority class and majority class.

5.2.4 Online Reward Adaptation

The provided guidelines assumed that the frequency of minority and majority classes is known. However, in real-world datasets containing class imbalances, the frequencies are unknown. To solve this problem, our approach is to use information collected during XCS's training. One way to estimate this is by averaging the proportion between instances of negative class and instances of positive class. Employing an idea from Q-learning of Reinforcement Learning [66], we averaged online imbalance ration as equation following:

$$ir_{online}(t+1) = ir_{online}(t) + \alpha * \left(\frac{n(-)}{n(+)} - ir_{online}(t) \right) \quad (5.8)$$

where $ir_{online}(0) = 1$, α=0.2 is step-size in Q-learning or learning rate in supervised learning; *n(-)* and *n(+)* are number of negative and positive instances gotten in each iteration, respectively.

We calculated ir_{online} by (5.8), reward calculation as equation (5.6) and we got results as well as table 5.4 and table 5.5 although the training time is longer. That is because of Cost-sensitive XCS needing some time to realize the imbalance ratio of problems. The online reward adaptation simplifies Cost-sensitive XCS's tuning, since it does not require a previous estimation of the imbalance ratio, which is important for Cost-sensitive XCS's performance.

5.3 Cost-Sensitive XCS in Multi-Class Tasks

Some of the classification techniques, described in this chapter, are originally designed for binary classification problems. Yet there are many real-world problems, such as character recognition, face identification, and text classification, where the input data is divided into more than two

categories. Learning with multiclass tasks is more difficult than with two-class tasks and a higher degree of class imbalance may increase the difficulty. First, we overview several approaches and then present our proposal for extending the binary classifiers to handle multiclass problems. To illustrate these approaches, let $Y = \{y_1, y_2, \ldots, y_K\}$ be the set of classes of the input data.

The first approach decomposes the multiclass problem into K binary problems [54]. For each class $y_i \in Y$, a binary problem is created where all instances that belong to y_i are considered positive examples, while the remaining instances are considered negative examples. A binary classifier is then constructed to separate instances of class y_i from the rest of the classes. This is known as the one-against-rest (1-r) approach.

The second approach, which is known as the one-against-one (1-1) approach, constructs $K(K-1)/2$ binary classifiers, where each classifier is used to distinguish between a pair of classes, (y_i, y_j). Instances that do not belong to either y_i or y_j are ignored when constructing the binary classifier for (y_i, y_j). In both 1-r and 1-1 approaches, a test instance is classified by combining the predictions made by the binary classifiers. A voting scheme is typically employed to combine the predictions, where the class that receives the highest number of votes is assigned to the test instance. In the 1-r approach, if an instance is classified as negative, then all classes except for the positive class receive a vote. This approach, however, may lead to ties among the different classes. Another possibility is to transform the outputs of the binary classifiers into probability estimates and then assign the test instance to the class that has the highest probability.

In this section, we present a method extended from our proposal in section 5.2 by using (1-r) approach for solving multi-class cost-sensitive learning problems. We establish some theoretical guarantees concerning the performance of this method. We also empirically evaluate the performance of the proposal method using benchmark datasets and verify our proposed method to performance of standard XCS. We propose that the reward of instance in one class classified correctly should be set according to the proportion of frequencies between the occurring of the minority class and the occurring of instances in that class by following equation:

+ Reward of minority class: $r(minority) = maximum\ reward = 1000$

+ Reward of instance, in class i, classified correctly:

$$r(class\ i) = \frac{\text{n(minority)}}{\text{n(class i)}} * \text{r(minority)} \qquad (5.9)$$

We run XCS with the proposal on eight multi-class UCI datasets shown in table 5.6, there are. The more detail information of these datasets is provided in Appendix C.2. The results on them with standard XCS and Cost-sensitive XCS are shown in table 5.7

Table 5.6 UCI Datasets Used in the Empirical Study

Data set	Size	Attribute	Class	Class distribution
lymphography	148	9B 9N	4	2/4/61/81
glass	214	9C	6	9/13/17/29/70/76
ecoli	336	10C	6	2*2/5/20/35/52/77/143
vowel	990	10C	11	90*11
optdigits	5600	64N	10	552/557/558*2/560/562/566/568/571/572
satellite	6435	36C	6	626/703/707/1358/1508/1533
pendigits	10992	16N	10	1055*4/1056/1142/1143*2/1144*2
letters	20000	16N	26	734*2/736/739/747/748/752/753/755/ 758/761/764/766/768/773/775/783*2/ 786/787/ 789/792/796/803/805/813

(B: Binary, N: Nominal, C: Continuous)

The results from table 5.7 show that, Cost-sensitive XCS performs better than Standard XCS on seven of nine datasets. The most significant improvement is in *e-coli* dataset where F1 increases from 0.772 to 0.821. The second improvement is in *letters* dataset from 0.759 to 0.823. Both Standard XCS and Cost-sensitive XCS encounter problem when classify *lymphograhy* dataset. Standard XCS reaches to 0.286 while Cost-sensitive XCS performs better with 0.333 in term of F1-measure. Although this dataset has small number of instance, the imbalance ratio is chosen to test is high (2.70% of minority instances). There are two datasets: *glass* and *optdigits* where Cost-sensitive XCS performs worse than Standard XCS.

We can conclude that cost-sensitive learning is relatively easy on two-class tasks while hard on multi-class tasks. This is not difficult to understand because an example can be misclassified in more way in multiclass tasks than it might be in two-class tasks, which means the multiclass cost function structure can be more complex to be incorporated in any learning algorithms. Investigating the nature of multi-class cost-sensitive learning and designing powerful learning methods remain important open problems.

Table 5.7 Performance of Standard XCS and Cost-sensitive XCS on multi-class UCI datasets

Datasets	Standard XCS	Cost-sensitive XCS
e-coli	0.722	0.821
glass	0.769	0.750
letters	0.759	0.823
lymphography	0.286	0.333
optdigits	0.957	0.946
pendigits	0.957	0.981
satellite	0.949	0.966
vowel	0.922	0.960
Average	0.790	0.823

5.4 Adapting Other Parameters

In XCS, there are many other parameters that could be adjusted. We think it is significant that XCS works well with the default parameter settings. However, it might be possible to substantially improve the performance of XCS with tuning some of these parameters. In this section, we explore the settings of parameters ϵ_0, ν and α. These parameters affect to calculate accuracies of classifiers. Changing accuracy also changes fitness so that we expect the genetic algorithm to be guided towards classifiers predicting correctly instances of minority class.

5.4.1 Parameter ϵ_0, ν and α Dependence in Imbalance Problems

As noted in Chapter 3, a classifier's fitness is based on the accuracy of its payoff prediction. The fitness is arrived at in steps. First, the classifier's accuracy K_j is computed using the formula:

$$\kappa_j = \begin{cases} \alpha\left(\frac{\varepsilon_j}{\varepsilon_0}\right)^{-\nu}, & \varepsilon_j \geq \varepsilon_0 \\ 1 & otherwise \end{cases} \qquad (5.10)$$

In this expression, ϵ_0 is a threshold such that if the classifier's error is less than ϵ_0, the classifier is deemed "accurate", and given accuracy 1. If, on the other hand, the error is greater than ϵ_0, the resulting lower accuracy is given by the falling exponential of the error that is also employed and controlled by parameter ν.

So, we see that parameters ϵ_0, ν and α are very important to decide whether a classifier is accurate or not. In imbalance problem, these parameters should be adapted approximately then *over-general* classifiers are considered inaccurate and accurate classifiers are protected where over-general classifier is a classifier or a rule covering more than one class. The idea behind adapting these parameters is that in highly imbalanced datasets, an over-general classifier will often be correct more often than wrong. Any change in the classifier's condition will be reflected in a change in its statistical correctness and - thus its error ϵ - and this will tend to guide the system toward accurate classifiers.

Adapting these parameters can be approached mathematically. Let us assume a two-level *R*/0 payoff landscape, where *R* is provided if the prediction is correct, 0, otherwise. According to [14] and [28], the prediction *P* of a classifier can be approximated by:

$$P = P_c(cl) \cdot R_{max} + (1 - P_c(cl)) \cdot R_{min} \tag{5.11}$$

where $P_c(cl)$ is the probability that a classifier classifies the matching input correctly, R_{max} is the maximum reward or $R_{max} = R$, and R_{min} the minimum reward given by the environment. The error of a classifier can be approximated by:

$$\epsilon = |P - R_{max}|P_c(cl) + |P - R_{min}|(1 - P_c(cl)) \tag{5.12}$$

For classification problems, R_{min} is usually 0, so that the prediction of a classifier can be estimated by: $P = P_c(\mathrm{cl})R_{max}$. Substituting *P* into formula (5.11), we get a prediction error estimate:

$$\epsilon = 2R_{max}(P_c(cl) - P_c(cl)^2) \tag{5.13}$$

$P_c(cl)$ can be approximated as:

$$P_c(cl) = \frac{C}{C+!C} \tag{5.14}$$

where *C* is the number of instances that the classifier predicts correctly (i.e., the predicted class agrees with that of the sample) and $!C$ is the number of instances that the classifier predicts incorrectly (i.e., the predicted class does not agree with that of the sample). The sum $C+!C$ corresponds to the total number of examples that the classifier matches. Let's denote as *p* the ratio between $!C$ and *C*:

$$p = \frac{!C}{C} \tag{5.15}$$

We can derive the classifier's probability of being correct as:

$$P_c(cl) = \frac{1}{1+p} \tag{5.16}$$

and its error estimate as:

$$\varepsilon = \frac{2p}{(1+p)^2} R_{\max} \tag{5.17}$$

In an accurate classifier, $p = 0$, which gives an error estimate equal to zero. For the maximally *overgeneral* classifier predicting the majority class, p=1/*ir*. An *overgeneral* classifier would be considered inaccurate as long as:

$$\varepsilon \geq \varepsilon_0 \tag{5.18}$$

Using equation (5.17), we get:

$$\frac{2p}{(1+p)^2} R_{\max} \geq \varepsilon_0 \tag{5.19}$$

Deriving the formula, we obtain:

$$\frac{\varepsilon_0}{R_{\max}} * ir^2 + 2\left(\frac{\varepsilon_0}{R_{\max}} - 1\right) * ir + \frac{\varepsilon_0}{R_{\max}} \leq 0 \tag{5.20}$$

Set $t = \epsilon_0/R_{max}$, where ϵ_0 is quite small when compared with R_{max}, so we only need consider 0 $\leq$ $\leq$1/2.

Substituting t to equation 5.20, we get:

$$t * ir^2 + 2(t-1) * ir + t \leq 0 \tag{5.21}$$

The boundary of *ir* respected to changing in $t = \epsilon_0/R_{max}$:

$$\frac{1-t-\sqrt{1-2t}}{t} \leq ir \leq \frac{1-t+\sqrt{1-2t}}{t} \tag{5.22}$$

Because:

$$\begin{cases} \lim\limits_{t\to 0^+} \dfrac{1-t-\sqrt{1-2t}}{t} = \lim\limits_{t\to 0^+} \dfrac{t}{1-t+\sqrt{1-2t}} = 0 \\ \lim\limits_{t\to 0^+} \dfrac{1-t+\sqrt{1-2t}}{t} = +\infty \end{cases} \tag{5.23}$$

So, if we decrease the value of $t = \epsilon_0/R_{max}$, the boundary of parameter *ir* increases or XCS can solve problems with higher imbalance levels. Our proposal is to set ϵ_0/R_{max} based on imbalance level i, ϵ_0=1/i.

Additionally, in equation (5.10), α causes a strong distinction between accurate and not quite accurate classifiers. The steepness of the succeeding slope is influenced by ν, so we should set α low enough and ν high enough. Moreover, population size should assure that no niche will be lost as suggested in chapter 4, it should be increased proportionally with the imbalance rate i, β and θ_{GA} should set with an appropriate window size as suggested in [27]. We run XCS on imbalanced multiplexer problems with imbalance level from i=1 to i=10 with the following parameter setting: ϵ_0=1/i, N=1000, α=0.05, ν=10, χ=0.8, μ=0.04, θ_{del}=20, δ=0.1, θ_{sub}=200, $P_{\#}$=0.6. The meaning of these parameters is provided in Appendix A.1.

Figure 5.1 shows that XCS is able to solve the imbalance problem if appropriate ϵ_0, ν and α values are chosen. From experimental results, we can see that although adaptive XCS needs some first steps to evolve accurate classifiers, it can reach to 100% performance with $i \leq 7$; 95% with $i = 8$ and 80% with $i = 9$ which is a notable improvement. For the highest imbalance level, $i = 10$ or $ir = 2^{10} = 1024$, XCS can only classify correctly about 5% minority instances. Because this is extremely imbalanced, for i=10 XCS sees about 1000 examples of the minority class in 1,000,000 learning iterations. Under a pure exploration regime, half of these 1000 examples activate niches of class 1 (highly rewarded niches), and the other half activate niches of class 0 (with reward 0). Thus, the system has only approximately 500 examples for discovering the high rewarding niches of the minority class. Such a severe imbalance ratio makes learning of the minority class really difficult. However, we believe that XCS is quite robust to high class imbalances. Imbalance levels $i = 10$ and further $i = 11$ represent two extreme cases in which we suspect that many learners will also suffer from biases. In addition, variant values of ϵ_0, ν and α were tested and $\epsilon_0 = 1/i$, $\nu = 10$ and $\alpha = 0.05$ show a better performance speed and stability in the imbalance problems. If we set $\nu \geq 10$ and $\alpha \leq 0.05$, we can get the results as same as Figure 5.1, but XCS needs a longer time in the beginning steps to evolve a population with accurate classifiers.

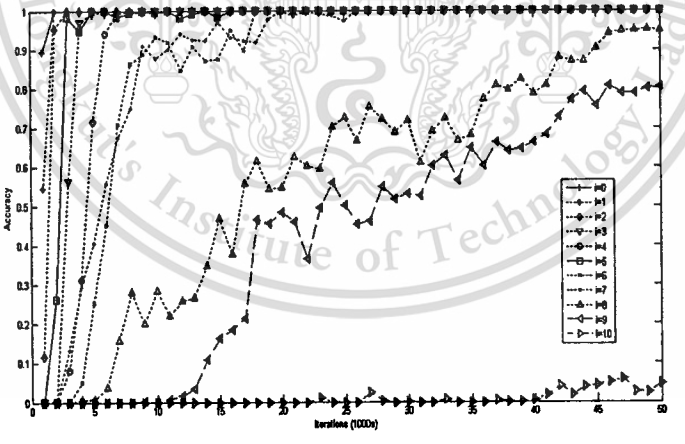


Figure 5.1 XCS in imbalance 11-MUX problem with parameter adaptation

5.4.2 Layered Payoff Benefits for Imbalance Problem

This section investigates XCS's behavior further. We apply XCS to the multiplexer problem with layered reward as well as combination with parameter adaptation analysis in section 5.4. In the imbalance problems, combinatory events are highly beneficial.

The layered reward multiplexer [8] provides useful fitness guidance for XCS. The more position bits are specified (starting from the most significant one) the less different reward levels are available and the closer the values of the different reward levels are together so that classifiers that have more position bits specified have higher accuracy. Consequently, those classifiers get propagated. Thus, the reward scheme somewhat simplifies the imbalance problem.

The exact equation of the layered reward scheme is [8]:

$$reward = (value\ of\ the\ k\ address\ bits + value\ of\ addressed\ bit) * 100 + (correctness) * 300 \quad (5.24)$$

where correctness is 1 if the action is correct, 0, otherwise. For example, in the 6-multiplexer the (incorrect) classification 0 of instance (100010) would result in a payoff of 500 while the (correct) classification 1 would result in a payoff of 800.

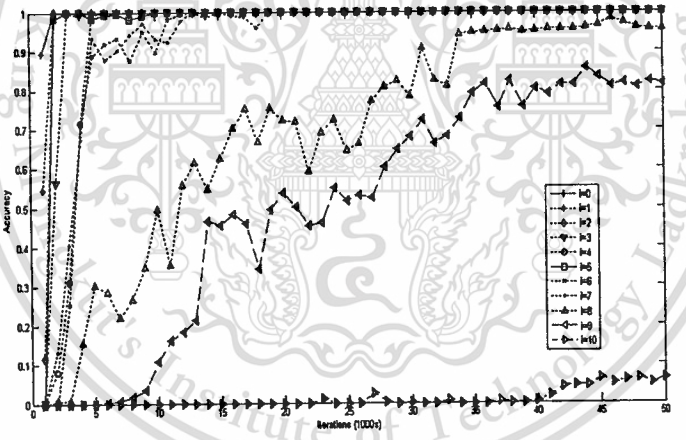


Figure 5.2 XCS in imbalance 11-MUX problem with benefits of Layered Payoff

Figure 5.2 shows experimental results for the layered 11-multiplexer when parameters are set as section 5.4.1. The performances in all imbalanced levels are look quite similar to that on Figure 5.1 only difference on some initial steps where XCS learn on layered reward faster than on original reward setting a little bit.

5.5 Summary

We have introduced a Cost-sensitive XCS learning classifier system to solve imbalance

problems. When implementing classifier tasks, a standard learning classifier system based on accuracy as XCS aims to minimize the overall error rate. For an imbalance dataset, the majority class has dominant influence on the overall error since XCS tends to prioritize the different classes in favor of the class with more training data examples in order to achieve a high overall accuracy. This biased favor makes a poor performance on classifying minority class and achieves a high accuracy on the majority class but a very low, sometimes unacceptable, accuracy on the minority class. Our mechanism for parameter adaptation is a novel approach that takes advantage of XCS's learning mechanism. An online reward adaptation is proposed; it does not need any priori estimation of the imbalance ratio because it uses the information collected by the classifiers themselves.

In this chapter, we first analyzed effects of imbalance datasets to performances of XCS. The results showed that with standard parameter settings, XCS is quite robust to imbalance problems up to imbalance ratio *ir*=16 with 11-MUX and *ir*=6 with real datasets. For higher imbalance levels, XCS's parameters are needed to adapt to improve its performance. This is mainly due to the fact that XCS has a bias towards the majority class. We identified the presence of *overgeneral* rules predicting the majority class which covered almost all the feature space. The constrained reward function and adapting other parameters that we proposed makes it possible for XCS to address imbalance problems. Cost-sensitive XCS attempts not only to maximize the total reward of positive class in long runs but also to increase the performance of XCS. One problem with this approach is that specific cost information is rarely available. So, we proposed to use online reward adaptation which can be yielded during the training time. We tested 11-MUX problem and seven UCI imbalance datasets by XCS first, by Cost-sensitive XCS following the guidelines, and further using the online adaptation method. Results showed that, performances of the proposed method are improved significantly in term of accuracy in positive class.

Experiments reveal that when the percentage of positive class gets larger, the increase of the *g-mean* and *F1-measure* values get smaller. It means that when the data balance issue get less severe (i.e., the datasets are more balanced), the improvement of the Cost-sensitive XCS over standard XCS gets smaller; when the data constitution is relatively balanced, the Cost-sensitive XCS does show significant superiority over standard XCS. And from experiments, we can see that with the same parameter settings, XCS performs on 11-MUX datasets more robustly than on real datasets (imbalance ratio *ir*=16 comparing to *ir*=6). This is because of difference on overlapping and complexity of each dataset. In [75], we investigated on this approach to analyze effects of imbalance

problems in combining with other factors as small disjunction, overlapping degree and complexity of problem to performance of classifiers.

Chapter 6

Sampling Ensemble XCS in Imbalance Problems

There are generally two methods of dealing with imbalance datasets. The first method is to modify the classifier as our first proposal described in the previous chapter. The second general approach to solve the imbalanced dataset problem is to modify the data itself. The most widespread and typical way to modify the dataset is to re-sample the data or sampling-based approach. Re-sampling refers to the process of changing the prior probabilities of the majority and minority class in the training set by changing the number of instances in the majority and minority class. In particular, over-sampling refers to the process of increasing the number of records in the minority class while under-sampling refers to the process of decreasing the number of records in the majority class. In this chapter, we analysis effects of re-sampling methods, ensemble learning algorithms and combining of re-sampling and ensemble to performance of XCS in imbalance datasets, where ensemble learning method is described detail in following.

The classification techniques we have seen so far in the previous chapter predict the class labels of unknown examples using a single classifier induced from training data. This chapter presents techniques for improving classification accuracy by aggregation the predictions of multiple classifiers. These techniques are known as the ensemble or classifier combination methods. An ensemble method constructs a set of base classifiers from training data and performs classification by taking a (weighted) vote on the predictions made by each base classifier. We will explain why ensemble methods tend to perform better than any single classifier and present techniques for combining ensemble methods with re-sampling techniques.

6.1 Rational for Ensemble and Sampling methods

6.1.1 Ensemble-based Learning

The following example illustrates how an ensemble method can improve a classifier's performance. Consider an ensemble of twenty-five binary classifiers, each of which has an error rate of $\epsilon = 0.35$. The ensemble classifier predicts the class label of a test example by taking a majority vote on the predictions made by the base classifiers. If the base classifiers are identical, then the

ensemble will misclassify the same examples predicted incorrectly by the base classifiers. Thus, the error rate of the ensemble remains 0.35. On the other hand, if the base classifiers are independent – i.e., their errors are uncorrelated – then the ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly. In this case, the error rate of the ensemble classifier is

$$e_{ensemble} = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06 \qquad (6.1)$$

which is considerably lower than the error rate of the base classifiers.

Figure 6.1 shows the error rate of an ensemble of twenty-five binary classifiers ($e_{ensemble}$) for different base classifier error rates (ϵ). The diagonal line represents the case in which the base classifiers are identical, while the solid line represents the case in which the base classifiers are independent. Observe that the ensemble classifier performs worse than the base classifiers when ϵ is larger than 0.5.

The preceding example illustrates two necessary conditions for an ensemble classifier to perform better than a single classifier: (1) the base classifiers should be independent of each other, and (2) the base classifiers should do better than a classifier that performs random guessing. In practice, it is difficult to ensure total independence among the base classifiers. Nevertheless, improvements in classification accuracies have been observed in ensemble methods in which the base classifiers are lightly correlated.

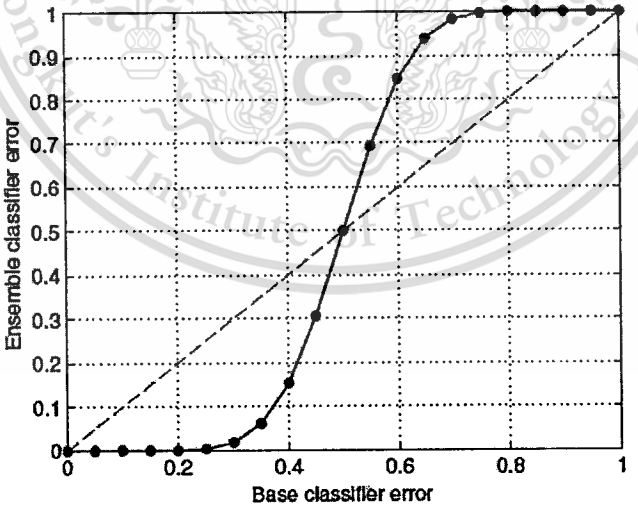


Figure 6.1 Comparison between errors of base classifiers and errors of the ensemble classifier [3].

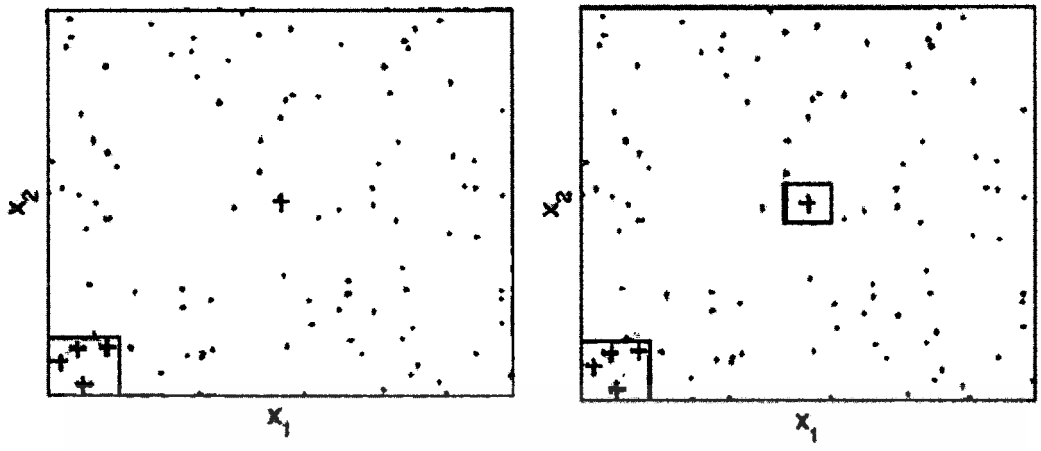
6.1.2 Sampling-based Approaches

Sampling is a widely used approach for handling the class imbalance problem. The idea of sampling is to modify the distribution of instances so that the rare class is well represented in the training set. Some of the available techniques for sampling include under-sampling, over-sampling, and hybrid of both approaches. To illustrate these techniques, consider a data set that contains 100 positive examples and 1000 negative examples.

In the case of under-sampling, a random sample of 100 negative examples is chosen to form the training set along with all the positive examples. One potential problem with this approach is that some of the useful negative examples may not be chosen for training, therefore, resulting in a less than optimal model. To overcome this problem, we propose one method, described in the next section, performing under-sampling multiple times and inducing multiple classifiers similar to the ensemble learning approach. Focused under-sampling methods may also be used, where the sampling procedure makes an informed choice with regard to the negative examples that should be eliminated, e.g., those located far away from the decision boundary.

Over-sampling replicates the positive examples until the training set has an equal number of positive and negative examples. Figure 6.2 illustrates the effect of over-sampling on the construction of a decision boundary. Without oversampling, only the positive examples at the bottom right-hand side of Figure 6.2(a) are classified correctly. The positive example in the middle of the diagram is misclassified because there are not enough examples to justify the creation of a new decision boundary to separate the positive and negative instances. Over-sampling provides the additional examples needed to ensure that the decision boundary surrounding the positive example is not pruned, as illustrated in Figure 6.2(b).

The hybrid approach uses a combination of under-sampling the majority class and over-sampling the rare class to achieve uniform class distribution. Under-sampling can be performed using random for focused sub-sampling. Over-sampling, on the other hand, can be done by replicating the existing positive examples or generating new positive examples in the neighborhood of the existing positive examples. In the latter approach, we must first determine the k-nearest neighbors for each existing positive example. A new positive example is then generated at some random point along the line segment that joins the positive example to one of its k-nearest neighbors. This process is repeated until the desired number of positive examples is reached. Unlike the data replication approach, the new examples allow us to extend the decision boundary for the positive class outward. Nevertheless, this approach may still be quite susceptible to model over-fitting.



(a) Without over-sampling (b) With oversampling

Figure 6.2 Illustrating the effect of over-sampling of the rare class

6.2 Ensemble Learning XCS

In the past few years, vast of experimental studies conducted by the machine learning community show that combining the outputs of multiple classifiers reduce the generalization error (see for instance [63], [64]). Ensemble methods are very effective; mainly due to the phenomenon that various types of classifiers have different "inductive biases" [2]. Indeed, ensemble methods can effectively make use of such diversity to reduce the variance-error without increasing the bias-error. In certain situations, an ensemble can also reduce bias-error, as shown by the theory of large margin classifiers.

Bagging and Boosting are two most popular methods in ensemble learning. In the following sections, we present effects of these methods to performance of XCS on imbalance datasets.

6.2.1 Bagging in XCSs

Figure 6.3 presents the pseudo-code of the bagging algorithm [63]. Each classifier is trained on a sample of instances taken with replacement from the training set. Usually each sample size is equal to the size of the original training set.

1: Let k be the number of bootstrap samples

2: For $i = 1$ to k do

3: Create a bootstrap sample of size N, D_i.

4: Train a base classifier C_i on the bootstrap sample D_i

5: End for

6: $C^*(x) = \arg\max_y \sum_i \delta(C_i(x) = y)$.

{$\delta(.) = 1$ if its argument is true and 0 otherwise}

Figure 6.3 The Bagging Algorithm.

In figure 6.3, since sampling with replacement is used, some of the original instances of D may appear more than once in D_i and some may not be included at all. So the training sets D_i are different from each other, but are certainly not independent. To classify a new instance, each classifier returns the class prediction for the unknown instance. The composite bagged classifier, C^*, returns the class that has been predicted most often (voting method). The result is that bagging produces a combined model that often performs better than the single model built from the original single data. In [62], Breiman notes that this is true especially for unstable inducers because bagging can eliminate their instability.

We test effects of Bagging learning method to performance of XCS on imbalanced datasets described in table 4.1 of chapter 4. The experimental results are shown in table 6.1. For easy to compare, we also present the performance of standard XCS in the imbalance datasets.

It is clear that, over these seven datasets, Bagging XCSs leads to markedly higher *precision* classifiers; but in the process may achieve very poor *recall*. Bagging XCSs reaches to optimal $\geq$97.7 %, in terms of *precision* in all datasets which make Bagging XCSs performs well on the first three datasets. It outperforms standard XCS when *F1* reaches to optimal solutions and reduces the variances (standard deviation is reduced significantly). However, on the last three datasets, Bagging XCSs encounters problem. That is because the weight update scheme shifts the classifier boundary all over the region of positive examples, such that every positive example gets covered by at least some classifier. Because in each iteration, the *recall* is limited by the weak of XCS learner on difficult datasets as *Vowel3*, *Hypothyroid1* and *Abalone19*, in a given number of iterations, each example may get predicted as positive class only by a few number of classifier. Thus, the ensemble-based prediction may assign it a class of negative, thus achieving an overall poor *recall*.

Table 6.1 Results of Bagging XCSs in Imbalance Datasets

Dataset	Standard XCS			Bagging XCSs		
	p	*r*	*F1*	*p*	*r*	*F1*
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	1.0±.0	0.972±0.034	0.989±0.021
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.977±.002	0.974±0.008	0.975±0.005
Wine3	0.971±.090	0.748±.311	0.801±.230	0.999±.001	0.998±0.007	0.998±0.001
Glass7	0.900±.161	0.867±.172	0.873±.142	1.0±.0	0.579±0.176	0.733±0.165
Vowel3	0.673±.326	0.522±.119	0.590±.233	1.0±.0	0.256±0.122	0.407±0.068
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	1.0±.0	0.0±0.0	0.0±0.0
Abalone19	0±.00	0±.00	0±.00	1.0±.0	0.0±0.0	0.0±0.0
Average	0.678±.157	0.614±.15	0.629±.140	0.997±.000	0.540±.050	0.586±.037

6.2.2 Boosting in XCSs

Boosting is an iterative procedure used to adaptively change the distribution of training examples so that the base classifiers will focus on examples that are hard to classify. Unlike bagging, boosting assigns a weight to each training example and may adaptively change the weight at the end of each boosting round.

Let us describe AdaBoost, one popular algorithm in Boosting Approaches. Let $\{(x_j, y_j) \mid j = 1, 2, \ldots, N\}$ denote a set of N training examples. In the AdaBoost algorithm, the importance of a base classifier C_i depends on its error rate, which is defined as

$$\varepsilon_i = \frac{1}{N}\left[\sum_{j=1}^{N} w_j I\left(C_i(x_j) \neq y_j\right)\right] \tag{6.2}$$

where $I(p) = 1$ if the predicate p is true, and 0 otherwise. The importance of a classifier C_i is given by the following parameter,

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right) \tag{6.3}$$

Note that α_i has as large positive value if the error rate is close to 0 and a large negative value if the error rate is close to 1.

The α_i parameter is also used to update the weight of the training examples. To illustrate, let $w_i^{(j)}$ denote the weight assigned to example (x_i, y_i) during the j^{th} boosting round. The weight update mechanism for AdaBoost is given by the equation:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \cdot \begin{cases} e^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases} \tag{6.4}$$

where Z_j is the normalization factor used to ensure that $\sum_i w_i^{(j+1)} = 1$. The weight update formula given in Equation 6.4 increases the weights of incorrectly classified examples and decreases the weights of those classified correctly. The algorithm of AdaBoost in the pseudo-code is presented in figure 6.4 [63].

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots N\}$. {Initialize the weights for all N examples.}

2: Let k be the number of boosting rounds.

3: **For** i =1 to k **do**

4: Create training set D_i by sampling (with replacement) from D according to w.

5: Train a base classifier C_i on D_i.

6: Apply C_i to all examples in the original training set, D.

7: $\varepsilon_i = \frac{1}{N}\left[\sum_j w_j \delta(C_i(x_j) \neq y_j)\right]$ {Calculate the weighted error.}

8: **if** $\epsilon_i > 0.5$ **then**

9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}

10: Go back to Step 4.

11: **end if**

12: $\alpha_i = \frac{1}{2}\ln\frac{1-\varepsilon_i}{\varepsilon_i}$

13: Update the weight of each example according to Equation (6.1)

14: **end for**

15: $C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(x) = y))$

Figure 6.4 AdaBoost algorithm

Table 6.2 shows the results of AdaBoost XCSs in our imbalance data problems. It is clearly to see that AdaBoost XCSs performs as well as or a little better than Standard XCS in the first four datasets, begins to find the difficulty from *Vowel3* dataset and classifies all input data as majority class in *Abalone19* or we can recognize that *Abalone19* is a really difficult problem. That is because in

the first four datasets, the based learner – XCS performs well, it gets accuracy greater than 50%, so AdaBoost XCSs has ability to improve performance by using meta-technique. In *Vowel*3, the accuracy of XCS is only little greater than 50%, meta-technique does not have any improvement, even makes problem worse. To improve this problem, we propose to combine meta-technique and sampling algorithms in solving imbalance data problems. We can use advantages of sampling algorithms to increase performance of based learner, and then use meta-technique to reduce variant error of each based learner and boost performance of all system. In the next section, we describe sampling techniques and following is the combination of these techniques with ensemble learning.

Table 6.2 AdaBoost XCSs

Dataset	Standard XCS			AdaBoost XCSs		
	p	*r*	*F1*	*p*	*r*	*F1*
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.987±.005	0.980±.018	0.983±.008
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.950±.023	0.967±.042	0.958±.019
Wine3	0.971±.090	0.748±.311	0.801±.230	0.972±.032	0.971±.051	0.970±.025
Glass7	0.900±.161	0.867±.172	0.873±.142	0.982±.023	0.759±.046	0.855±.034
Vowel3	0.673±.326	0.522±.119	0.590±.233	0.200±.422	0.009±.021	0.017±.040
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.792±.336	0.143±.039	0.227±.055
Abalone19	0±.00	0±.00	0±.00	0±.00	0±.00	0±.00
Average	0.678±.157	0.614±.15	0.629±.140	0.698±.120	0.547±.031	0.573±.026

6.3 Sampling XCS

As described in section 6.1, re-sampling methods are applicable to many types of classifier. The aim of re-sampling is to balance a priori probabilities of the classes. In this section, we analyze effects of over-sampling and under-sampling methods to XCS on our seven UCI datasets.

6.3.1 Over-sampling XCS

Figure 6.5 represents the idea of over-sampling with replacement method. In this figure, the minority class is over-sampled by increasing amounts. We study different over-sampling techniques simply to see if any of these techniques are promising on XCS to solve imbalance dataset problems. The simplest over-sampling approach is random oversampling. There, one chooses instances from the

minority class at random; these randomly chosen instances are then duplicated and added to the new training set. This method only lightly improves performance of XCS on our imbalance dataset. A promising over-sampling technique used in this study is neighbor Based Over-sampling Technique (NBOTE) [33]. That is, for each minority instance, we find the K nearest neighbors, draw a line between the instance and each of its K nearest neighbors and then randomly select a point on each line to use as a new minority instance. In this way, Kxn new minority instances are added to the training data, where n is the number of minority class instances in original training data and K is based on imbalance ratio ir.

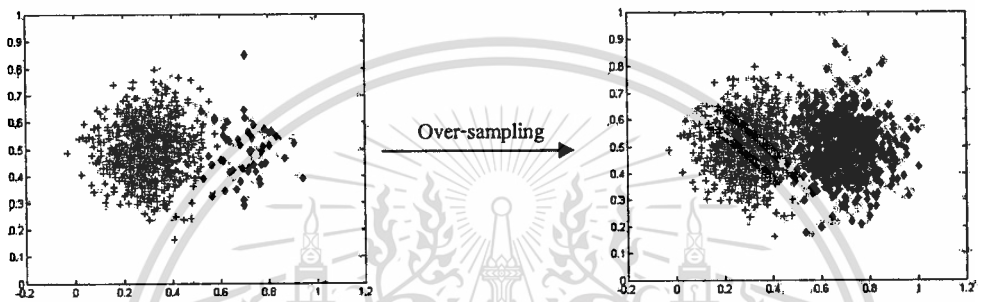


Figure 6.5 Over-sampling method

Table 6.3 Results of Over-sampling XCS

Dataset	Standard XCS			Over-sampling XCS		
	p	*r*	*F1*	*p*	*r*	*F1*
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.953±.048	0.870±.063	0.907±.024
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.997±.002	0.946±.005	0.971±.002
Wine3	0.971±.090	0.748±.311	0.801±.230	0.998±.007	0.960±.018	0.979±.010
Glass7	0.900±.161	0.867±.172	0.873±.142	0.969±.011	0.886±.041	0.925±.020
Vowel3	0.673±.326	0.522±.119	0.590±.233	1.0±.0.0	0.858±.043	0.923±.025
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.997±.005	0.625±.116	0.762±.093
Abalone19	0±.00	0±.00	0±.00	0.997±.010	0.086±.064	0.154±.095
Average	0.678±.157	0.614±.15	0.629±.140	0.987±.014	0.747±.050	0.803±.038

The experimental results are shown in table 6.3. Our experimental result is 10-fold cross-validation method. Table 6.3 shows that XCS works quite well on several imbalance datasets without

sampling. However, with re-sampling, the performance of XCS increases significantly, except *Echocardiogram* data. In general, since generative over-sampling creates new instances of minority class, it is possible that generative over-sampling helps XCS to learn a more accurate model of the minority class, thus increasing *precision*. Recall, *precision* measure determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class, p=TP/(TP+FP). The higher the *precision* is, the lower the number of false positive errors committed by the classifier. And since we are creating new instances based only on the instances in the training set, we are probably over-fitting the training set, thus resulting in lower *recall*. However, it is a bit surprising that the value of *recall* is also improved in *Wine3*, *Vowel3*, *Hypothyroid* and a bit better in *Glass7*.

Under oversampling, XCS sees a dataset with the same number examples per class, but as described in figure 6.5, there are still some gaps in the feature space that are not represented by any example. These gaps are mostly covered by rules predicting the majority class rather than by minority class rules. In fact, what happens is that rules from both classes tend to expand as much as possible into these gaps until they reach points belonging to the opposite classes. That is, rules tend to expand as long as they are accurate. Thus, there are overlapping rules belonging to different classes in the regions that are not covered by any example. When we test XCS in these regions, the majority class rules have higher *numerosities* and their vote into the prediction array is higher. The majority class rules have higher *numerosities* is that their boundaries are less complex, so in many cases a single rule suffices for all the regions. This rule tends to cover all the examples of a majority class square and benefits from long experience and numerosity. On the contrary, minority class regions are more complex, and rules covering these regions tend to have less experience and *numerosity*. Generative oversampling, which creates completely new minority class documents, also does quite well, but there are several datasets on which it performs poorly as *Abalone*.

6.3.2 Under-sampling XCS

Under-sampling seeks to reduce the number of majority class members in the training set. As a result, the overall number of records in the training set is greatly reduced, we can see in figure 6.6. This means that during classification, training time is also greatly reduced. So, there is a significant savings in memory as well if we deal with high dimensional datasets. We test effect of under-sampling to XCS on our seven UCI datasets and get results shown in table 6.4.

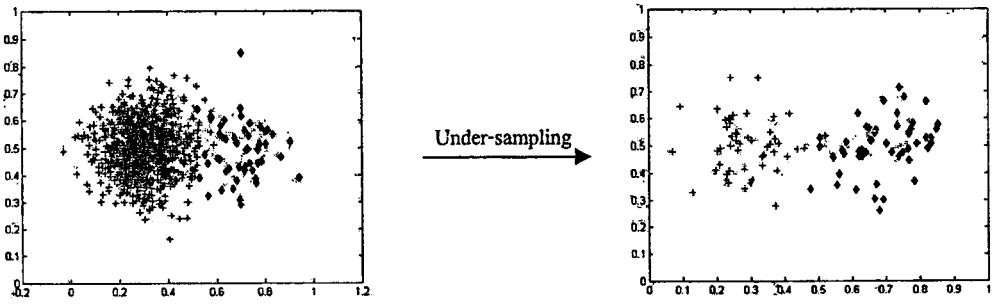


Figure 6.6 Under-sampling method

Table 6.4 Results of Under-sampling XCS

Dataset	Standard XCS			Under-sampling XCS		
	p	*r*	*F1*	*p*	*r*	*F1*
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.948±.079	0.827±.160	0.874±.119
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.913±.043	0.998±.002	0.953±.023
Wine3	0.971±.090	0.748±.311	0.801±.230	0.959±.028	1.0±.0	0.979±.014
Glass7	0.900±.161	0.867±.172	0.873±.142	0.728±.132	0.945±.024	0.815±.089
Vowel3	0.673±.326	0.522±.119	0.590±.233	0.142±.037	1.0±.0	0.247±.055
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.025±.0	1.0±.0	0.048±.0
Abalone19	0±.00	0±.00	0±.00	0.008±.001	1.0±.0	0.016±.001

From the results, we can see that, under-sampling tends to outperform over-sampling in terms of *recall*, but at a very high cost of *precision*, in some datasets as *Vowel3* and *Hypothyroid1*, the value of *precision* is too low. When compared to over-sampling, over-sampling technique yields much better F1-measure. Finally, none of dataset, under-sampling significantly outperforms over-sampling. That is to say, under-sampling seems to be a very poor choice when using XCS classifier system.

Analyzing the behavior of XCS under these two strategies, we found that under under-sampling there is less generalization pressure towards the minority class rules than with oversampling. Under oversampling, minority class instances are always sampled at the same a priori probability as majority class instances, keeping the same generalization pressure towards both classes. This may justify why, with under-sampling, XCS finds more difficulties in the generalization of rules covering the minority class instances, especially for the highest imbalance levels.

6.4 Over-sampling Ensemble XCSs

Figure 6.7 presents the framework of Over-sampling Ensemble XCS whose performance includes 3 steps: Step 1 creates multiple datasets from original Training data using Over-sampling techniques. Given a dataset having two classes: minority class and majority class, Over-sampling is used only on minority class to create minority training subsets. The number of instance in each minority training subset is equal to the number of the majority class instances. This process is repeated *ir* times where *ir* is the imbalance ratio between of majority class instances and minority class instances. After that, each minority subset is combined with all the majority examples to be an individual training data subset. In step 2, each training data subset is used to train one XCS classifier system. And finally, in step 3, all *ir* XCSs are combined to make a prediction on a test example. The outputs of all individual classifiers are aggregated to produce the output of the ensemble.

After each classifier is trained independently, aggregating the results of each classifier are interesting research issues concerning XCS ensembles. Some combination strategies are suggested by previous studies: if only class labels are considered, a majority vote can be used; if continuous-valued outputs like posteriori probabilities are available, the sum of all the output probabilities has been suggested (Sum Rule) [78]. Besides these direct combination strategies, it is possible to "stacked" another learning method on top using the outputs of the input classifiers as new features [78], [79]. A mixture model of XCSs with another neural network (NN) as a gate has been proposed to solve very large-scale classification problem [77]. One simple and effective ensemble method is weighted voting where the output of an individual classifier with small training error contributes to the output of the ensemble more than an individual classifier with large training error. So, in XCS ensemble, we choose weighted voting to aggregate results from individual classifier.

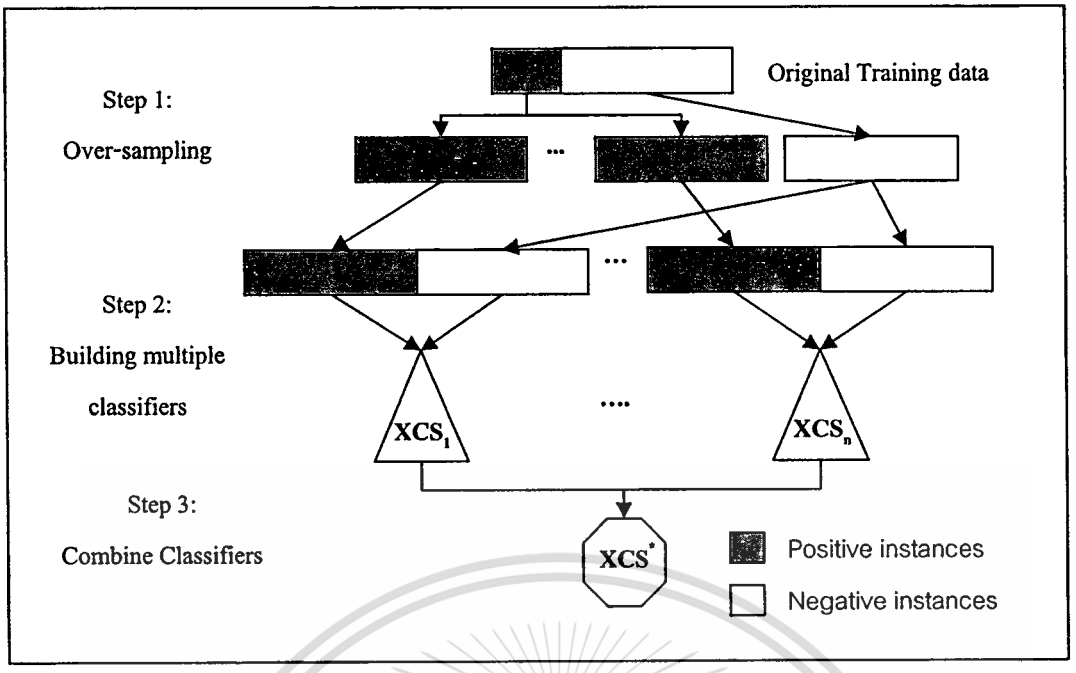


Figure 6.7 Over-sampling Ensemble XCSs Framework

The experimental results of over-sampling ensemble XCS are shown in table 6.5. All experiments presented are averaged over 10 independent runs of 10-fold cross-validation technique [3]. The performance of Over-sampling Ensemble XCS in table 6.5 increases significantly in comparing to the performance of Standard XCS. In general, since generative over-sampling creates new instances of minority class, it is possible that generative over-sampling helps XCS to learn a more accurate model of the minority class, thus increasing *precision*. The higher the *precision* is, the lower the number of false positive errors committed by the classifier. And since we are creating new instances based only on the instances in the training set, we are probably over-fitting the training set, thus resulting in lower *recall* as we discussed in sub-section 6.3.2. However, because of the combination with ensemble learning, the value of *recall* is improved significantly as in *Wine3*, *Vowel3*, *Hypothyroid* and a bit better in *Glass7*.

In addition, we can observe that over-sampling ensemble XCS outperforms on almost datasets compared with Standard XCS, it yields a 20% improvement on the average performance. We present a brief comparison between methods in the next section after getting results of Under-sampling Ensemble XCS.

Table 6.5 Over-sampling Ensemble XCS on imbalance datasets

Dataset	Standard XCS			Over-sampling Ensemble XCS		
	p	*r*	*F1*	*p*	*r*	*F1*
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.993±.048	0.950±.063	0.977±.008
Breast-w	0.931±.041	0.967±.042	0.948±.024	1.0±.0	0.986±.005	0.999±.001
Wine3	0.971±.090	0.748±.311	0.801±.230	1.0±.0	0.980±.012	1.0±.0
Glass7	0.900±.161	0.867±.172	0.873±.142	0.990±.011	0.960±.040	0.981±.053
Vowel3	0.673±.326	0.522±.119	0.590±.233	1.0±.0	0.958±.043	0.961±.062
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.997±.005	0.525±.116	0.690±.133
Abalone19	0±.00	0±.00	0±.00	0.997±.010	0.006±.064	0.0±.01
Average	0.678±.157	0.614±.15	0.629±.140	0.996±.012	0.766±.049	0.801±.038

6.5 Under-sampling Ensemble XCSs

Figure 6.8 presents the framework of Under-sampling Ensemble XCS, as Over-sampling Ensemble XCS, there are three main steps in performance of Under-sampling Ensemble XCS: Creating training subset by using Under-sampling, Building multiple classifiers and Aggregating results from based classifiers. Under-sampling Ensemble XCS is inherited from Over-sampling XCS on step 2 and step 3; only different in step 1 where Under-sampling technique is used to create training subsets instead of Over-sampling technique. Under-sampling is used *ir* times on majority class instances to generate *ir* bootstrap samples so that each bootstrap sample has the same or similar size with the minority instances. Then, each bootstrap sample (of the majority class) is combined with the entire minority instances to form training sets. Finally, the *ir* XCSs are combined to make a prediction on a test example by casting a weighted voting from the ensemble of XCSs.

There are also some under-sampling strategies suggested by the past studies. Like random Over-sampling, random Under-sampling is a simple yet effective approach to re-sampling. The following under-sampling method used in our study is taken from [50] with some adjustment in order to be suited for our imbalance datasets. In particular, for each minority class member in the training

set, we select the *n* closest majority class documents in the training set, where *n*=(target num majority class instances) / (number minority class instances).

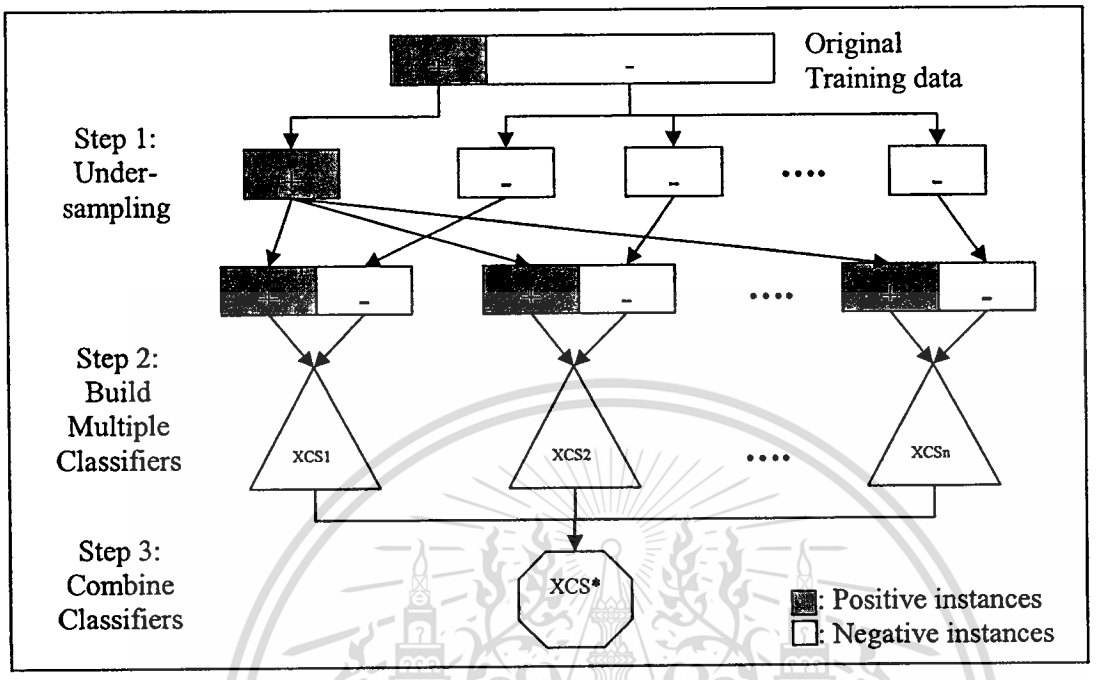


Figure 6.8 Under-sampling Ensemble XCSs Framework

Table 6.6 Results of Under-sampling Ensemble XCS on Imbalance Datasets

Dataset	Standard XCS			Under-sampling Ensemble XCS		
	p	*r*	*F1*	*p*	*r*	*F1*
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.938±.079	0.977±.160	0.959±0.019
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.913±.043	0.998±.002	0.985±0.014
Wine3	0.971±.090	0.748±.311	0.801±.230	0.959±.028	1.0±.0	0.991±0.004
Glass7	0.900±.161	0.867±.172	0.873±.142	0.728±.132	0.945±.024	0.753±0.165
Vowel3	0.673±.326	0.522±.119	0.590±.233	0.142±.037	1.0±.0	0.117±0.068
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.025±.0	1.0±.0	0.0±0.0
Abalone19	0±.00	0±.00	0±.00	0.008±.001	1.0±.0	0.0±0.0
Average	0.678±.157	0.614±.15	0.629±.140	0.530±.046	0.989±.027	0.621±.039

The experimental results of Under-sampling Ensemble XCS are shown in table 6.9. All experiments presented are averaged over 10 independent runs of 10-fold cross-validation technique

[3]. We can observe that, Under-sampling Ensemble XCS tends to outperform over-sampling ensemble XCS in terms of *recall*, but at a very high cost of *precision*. In some datasets as *Vowel3* and *Hypothyroid1*, the value of *precision* is too low. That is because under-sampling tries to reduce the number of majority class instances, then in training phrase, many minority instances will get covered by many classifiers. Thus, the overall ensemble can achieve very good *recall* levels. However, in the process, sufficiently large number of majority class instances can also get covered within minority rules' boundaries. Remember that the total number of minority examples is small, so even if a small fraction of total number of majority examples get captured by sufficiently many classifiers, the *precision* will suffer.

In our imbalance datasets, Under-sampling Ensemble XCS outperforms, in terms of *F1-measure*, standard XCS on the first four datasets: *Echocardiogram*, *Breast-w*, *Wine3* and *Glass7*. However, it begins to find difficulties classifying the minority class examples of the last three datasets where the imbalance ratio increases. With *Hypothyroid1* and *Abalone19*, Under-sampling Ensemble XCS classifies all input data as majority class instances. When compared to over-sampling ensemble, none of dataset, under-sampling ensemble XCS significantly outperforms over-sampling ensemble XCS. That is to say, under-sampling ensemble seems to be a very poor choice when using XCS classifier system.

6.6 Summary

In this chapter, we have introduced the rational of sampling and ensemble method in classification problems. Then, we have analyzed effects of these methods to performance of XCS on imbalance datasets. After that, we present our proposed systems, ensemble XCS.

In our proposed systems, it is found some advantages over the basic methods. First, ensemble XCS uses the information of the entire dataset compared to under-sampling which uses only part of the dataset. Second, ensemble XCS is able to overcome the limitation of the current XCS classifier system. XCS is stable classification methods and expected to learn a global optimum [24]. However, XCS is also an Evolution-Based learning [8] and we cannot guarantee that a single XCS, which is used for the over/under-sampling case, always provides the optimal performance over the test data. XCS ensembles can overcome this limitation by combining those potentially suboptimal solutions and thus achieve better performance.

Results from seven UCI imbalance datasets showed that, performances of the proposed methods are improved significantly in term of accuracy in positive class. We conclude that Sampling Ensemble XCS is good candidates to address the imbalance dataset problem in scene classification, which can simultaneously achieve high effectiveness as well as high efficiency.

Chapter 7

The Proposed Methods Compared to Other Related Works

Chapter 4 through chapter 6, we have described in detail our proposals: Cost-sensitive XCS and Sampling Ensemble XCS to solve imbalance problems. In this chapter, we compare the performance, in terms of F1-measure evaluation, of our proposals to seven well-known learning algorithms, coming from instance based learning, Rule-learning, Ensemble learning, Decision Tree Induction, Neural Networks, Support Vector Machine, Bayesian Classifier Systems and Nearest Neighbor Algorithms. All these algorithms are obtained from the Weka package developed at the University of Waikato in New Zealand. The code is available from the http address: http://www.cs.waikato.ac.nz/ml/weka. These algorithms are run with the default configuration provided by Weka. The experiments, performed on several imbalance datasets, show the suitability of our proposals for imbalance classification tasks.

Before we start the comparison, let us summarize and emphasize some key properties and results, in term of F1-measure, of our proposals.

- In Cost-sensitive XCS, deriving from the idea of Cost-sensitive learning, we propose to set reward for correctly identifying the positive class outweighs the value of correctly identifying the common class. This takes a cost matrix into consideration during model building and generates a model that has the lowest cost or lowest error rate.
- On other hand, Sampling Ensemble XCSs get benefits from sampling techniques and ensemble learning. As sampling techniques, Sampling Ensemble XCSs change the priors in the training set by decreasing points from the majority class. Sampling Ensemble XCSs inherited from Ensemble learning reducing variance, or instability of learning classifiers.
- Table 7.1 summarizes performance of our proposal systems on imbalance datasets. From this table, we can see that, Over-sampling is quite suitable for XCS in solving imbalance problems while Sampling Ensemble XCSs outperform on datasets that standard XCS classifies inputs correctly greater than 50% as *Echocardiogram, Breast-w, Wine3* in Under-sampling ensemble XCS and these datasets including *Galss7, Vowel3* in Over-sampling Ensemble XCS. Cost-sensitive XCS has acceptable results on almost datasets.

Table 7.1 Summarize performances of Under-sampling XCS, Over-sampling XCS, Cost-sensitive XCS and Ensemble XCSs on imbalance datasets

Dataset	Under-sampling XCS	Over-sampling XCS	Cost-sensitive XCS	Un-sampling Ensemble XCS	Over-sampling Ensemble XCS
Echocardiogram	0.874±.119	0.907±.024	0.983±.062	0.959±.019	0.977±.008
Breast-w	0.953±.023	0.971±.002	0.958±.021	0.985±.014	0.999±.001
Wine3	0.979±.014	0.979±.010	0.839±.224	0.991±.004	1.0±.0
Glass7	0.815±.089	0.925±.020	0.912±.224	0.753±.165	0.981±.053
Vowel3	0.247±.055	0.923±.025	0.815±.101	0.117±.068	0.961±.062
Hypothyroid1	0.048±.0	0.762±.093	0.655±.178	0.0±.0	0.690±.133
Abalone19	0.016±.001	0.154±.095	0.519±.275	0.0±.0	0.0±.01
Average	0.562±.043	0.803±.0384	0.812±.155	0.544±.039	0.801±.038

In each the following section, we briefly introduce algorithms chosen to compare and discuss detail performance of these systems in comparison with our proposals. We give a few references for further reading, and anyone who wants to understand any of these methods may find from elsewhere; the references contained here would only serve to get one started.

7.1 The Proposals and Other Rule-based Approaches

A rule-based classifier is a technique for classifying records using a collection of "if... then..." rules. To build a rule-based classifier, we need to extract a set of rules that identifies key relationships between the attributes of a dataset and the class label. There are two broad classes of methods for extracting classification rules: (1) direct methods, which extract classification rules directly from data, and (2) indirect methods, which extract classification rules from other classification models, such as decision trees and neural networks.

The four Rule-based Systems are chosen to compare are: *Conjunctive Rule*, *JRip*, *PART* and *Ridor*. *Conjunctive Rule* learns a single rule that predicts either a numeric or a nominal class value. Uncovered test instances are assigned the default class value (or distribution) of the uncovered training instances. The information gain (nominal class) or variance reduction (numeric class) of each antecedent is computed, and rules are pruned using reduced-error pruning [7]. JRip implements

RIPPER [61] algorithm, including heuristic global optimization of the rule set, where RIPPER is an algorithm particularly suited for building models from datasets with imbalanced class distributions [61]. RIPPER also works well with noisy datasets because it uses a validation set to prevent model over-fitting. Beside that, PART and Ridor are also evaluated, where the former is an alternative approach to rule induction that avoids global optimization but nevertheless produces accurate, compact, rule sets. The method combines the divide-and-conquer strategy for decision tree learning with the separate-and-conquer one for rule learning. The latter learns rules with exceptions by generating the default rule, using incremental reduced-error pruning to find exceptions with the smallest error rate, finding the best exceptions fro each exception and iterating [7].

Table 7.2 Performances of Rule-based Approaches on imbalance datasets

Dataset	Conjunctive Rule	JRip	PART	Ridor
Echocardiogram	0.842	0.827	0.790	0.831
Breast-w	0.888	0.933	0.905	0.924
Wine3	0.863	0.948	0.894	0.899
Glass7	0.897	0.877	0.764	0.787
Vowel3	0	0.681	0.635	0.667
Hypothyroid1	0.771	0.963	0.963	0.968
Abalone19	0	0	0	0
Average	0.609	0.747	0.707	0.725

Table 7.2 summarizes the performances of the different rule-based systems on all the datasets. Several observations can be drawn from this table. Almost the rule-based systems performing on these datasets are quite similar. Comparing all kinds of XCS Classifiers and the rule-based systems, we observe that there are significant differences between them in these datasets. XCS-based classifiers perform better in the first five datasets: *Echocardiogram*, *Breast-w*, *Wine3*, *Glass7* and *Vowel3* (except Under-sampling XCS in *Vowel3* dataset). In *Breast-w* and *Wine3*, Sampling Ensemble XCSs, both under-sampling and over-sampling, outperform significantly nearly all classifier schemes. These systems also show a good performance on *Echocardiogram* dataset, whereas the worst performances of XCS-based classifiers are obtained on *Hypothyroid1* and *Abalone19*. And as expected, as imbalance ratio increases, the performances of classifiers decrease.

Because almost classifiers, including XCS-based and other approaches, perform poorly on *Abalone19* dataset, the most surprising occurs on *Hypothyroid1* dataset. We observe all kind of XCS-based classifiers encounter problem when learning from this dataset, however, almost other learning approaches classify *Hypothyroid1* dataset well, at a confidence level greater than 95%. That is maybe because of geometrical complexity of the problem, such as the degree of clustering of the points of the same class and the proximity between the classes, as discuss in [23]. Nevertheless, the performance of Ridor on *Hypothyroid1* is the best one on table 7.2. From this table, we can observe that JRip algorithm has significantly higher prediction accuracy than Conjunctive Rule, PART and Ridor in almost datasets that is because JRip implements RIPPER algorithm, one algorithm is proposed to solve imbalance problems [61].

7.2 The Proposals and Other Ensemble Approaches

As introduced in chapter 6, ensemble approaches are combination the decisions of different models to reduce the variance-error without increasing the bias-error. This section describes briefly results from applying some ensemble learning algorithms to our imbalance datasets. Three well-known ensemble learning algorithms are chosen: AdaBoostM1, Bagging REPTree and MultiBoostAB.

First, we describe a widely used method called AdaBoostM1 that is designed specifically for classification. AdaBoostM1 can be applied to any classification learning algorithm that can handle weighted instance, where the weight of an instance is a positive number. By weighting instances, the learning algorithm can be forced to concentrate on a particular set of instances, namely, those with weight. Such instances become particularly important because there is a greater incentive to classify them correctly. AdaBoostM1 implements the algorithm described in chapter 6. It can be accelerated by specifying a threshold for weight pruning. AdaBoostM1 resamples if the base classifier cannot handle weighted instances.

Second, Bagging REPTRee uses Fast Tree [3] learner that uses reduced-error pruning as weak leaner. Because Bagging Algorithm is described quite detail in chapter 6, we do not more describe it in this section.

Third, MultiBoostAB is a boosting algorithm combining boosting with a variant of bagging to prevent over-fitting. It is able to harness both AdaBoost's high bias and variance reduction with wagging's superior variance reduction. Using C4.5 as the base learning algorithm, Multi-boostingAB

is demonstrated to produce decision committees with lower error than either AdaBoost or wagging significantly. It offers the further advantage over AdaBoost of suiting parallel execution. It operates by selectively re-sampling from the training data to generate derived training sets to which the base learner is applied.

Table 7.3 Performances of Other Ensemble Approaches on imbalance datasets

Dataset	AdaBoostM1	Bagging REPTree	MultiBoostAB
Echocardiogram	0.850	0.857	0.857
Breast-w	0.918	0.932	0.914
Wine3	0.929	0.959	0.918
Glass7	0.862	0.897	0.897
Vowel3	0.130	0.699	0
Hypothyroid1	0.963	0.957	0.968
Abalone19	0	0	0
Average	0.665	0.757	0.651

Table 7.3 summarizes the performances of the different ensemble learning algorithms on our imbalance datasets. The experimental results show that ensemble approaches are little better than Rule-based approaches. From the observation Bagging REPTree is the best ensemble algorithm on these datasets. Looking closer to training time, the effective training time specifies the number of (effective) generations or epochs until the minimum validation error occurred. As expected, the more complex problems cause more difficulty for classifiers, thus, a longer effective training time. Normally, ensemble learning approaches need longer time to build model than other approaches do.

In term of F1-measure, like our proposals, ensemble approaches classify well on *Echocardiogram*, *Breast-w*, *Wine3* and *Glass7*; begin encounter difficulty on *Vowel3*. Only Bagging REPTree gets 69.9% on *Vowel3*, AdaBoostM1 and MultiBoostAB classify all inputs of this dataset as negative class. Once again, we can see that non-XCS-based classifiers perform well on *Hypothyroid1*; like Ridor of Rule-based system in the previous section, MultiBoostAB even gets the highest performance at 96.8% on this dataset. Recall, XCS is a genetic-based system and *Hypothyroid1* is characterized by a high number of attributes and instances, 21 attributes and 3772 instances, the results in this section confirm the observations made in the previous section; where XCS was hypothesized to perform worse in problems with a high number of classes, so further investigation

should be done to improve this problem. With almost classifiers, *Wine3* gets a high performance and all ensemble approaches considered in this section cannot classify *Abalone* dataset.

7.3 The Proposals and Decision Tree Approaches

This section introduces some decision tree classifiers in imbalance datasets, which are simple yet widely used classification techniques. The attractiveness of decision trees is due to the fact that, like XCS and in contrast to neural networks, decision trees represent rules. Rules can readily be expressed so that humans can understand them or even directly used in a database access language like SQL so that records falling into a particular category may be retrieved [6].

In principle, there are exponentially many decision trees that can be constructed from a given set of attributes. While some of the trees are more accurate than others, finding the optimal tree is computationally infeasible because of the exponential size of the search space. Nevertheless, efficient algorithms have been developed to induce a reasonably accurate, albeit suboptimal, decision tree in a reasonable amount of time. Four well-known Decision Tree Inductions are chosen to compare in this section: Random Tree, Random Forest, REPTree and C4.5 revision 8. In the following, we describe briefly the properties of these classifiers.

Random Tree chooses a test based on a given number of random features at each node, performing no pruning. The strength of the *random tree* method is that it can be used to generate new random sequences in a reproducible and non-centralized fashion. This is valuable, for example, in applications in which new tasks and hence new random generators must be created dynamically [33].

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges to a limit as the number of trees in the forest becomes large. As other ensemble learning, the generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them [62].

REPTree builds a decision tree using information gain/variance reduction and prunes it using reduced-error pruning. Optimized for speed, it only sorts values for numeric attributes once. It deals with missing values by splitting instances into pieces [46].

The most well-known decision tree learner is C4.5. For many domains, the trees produced by C4.5 are both small and accurate, resulting in fast, reliable classifiers. These properties make decision trees a valuable and popular tool for classification. C4.5 and its predecessor, ID3, use formulas based

on information theory to evaluate the "goodness" of a test; in particular, they choose the test that extracts the maximum amount of information from a set of cases, given the constraint that only one attribute will be tested [46].

Table 7.4 Performances of Decision Tree Approaches on imbalance datasets

Dataset	Random Tree	Random Forest	REPTree	C4.5 revision 8
Echocardiogram	0.659	0.857	0.868	0.842
Breast-w	0.919	0.940	0.915	0.930
Wine3	0.894	0.947	0.898	0.894
Glass7	0.659	0.873	0.897	0.842
Vowel3	0.659	0.817	0.601	0.689
Hypothyroid1	0.846	0.904	0.969	0.963
Abalone19	0.037	0	0	0
Average	0.667	0.763	0.735	0.737

The results for the 4 chosen decision tree classifiers are shown in Table 7.4. The first point to note is that the results fall into three categories: less than 70% (*Random Tree*), ~73.5% (*REPTree* and *C4.5 revision 8*) and the best technique (*Random Forest*). Although there are little differences within these categories, it is probably possible to experiment with parameters improve each technique's performance within the categories. This implies there are levels of complexity in the underlying relationship between the attributes and the response, and the level of accuracy attained is determined in part by the level of complexity of model considered. This point is reinforced by the fact that the more complex model Random Forest outperforms Random Tree.

Before our work, Quinlan [6] compared performance of Decision Tree and Classifier Systems. Particularly, he showed that BOOLE, one kind of Classifier System, introduced by Wilson [4], is noted extremely slow convergence rate and obtained much better results when using C4.5 algorithm and post-processing to rules. However, after BOOLE, Wilson proposed NewBOOLE a new version of Boole and especially in 1995, XCS was introduced which converges significantly faster to accurate results. It requires around 800 examples to find an (almost) accurate hypothesis and around 5000 examples to find the minimal set of rules. The same author also used neural nets of different sizes to learn the same concept. He report convergence after 1600 cycles for a reasonable net (6:20-

20-10-10:1). Furthermore, on the 11-multiplexer problem, NewBOOLE requires around 4000 examples to converge, a neural net around 8000.

From table 7.4, Random Forest is the best technique for classifying our imbalance dataset. Using a F1-measure assuming unequal variance, the mean testing accuracy of Random Forest is significantly higher than that of all other techniques at the 3% level. With advantages of decision trees as small and simple to understand, interpret and results gotten, we can conclude that performances of decision trees on imbalance problem are reliable and accuracy.

7.4 The Proposals and Bayesian Classifier Approaches

The Bayesian Classifier techniques are based on the so-called Bayesian theorem and are particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Bayesian classifiers can often outperform more sophisticated classification methods. This section presents results of four well-known Bayesian Classifiers: Bayes Net, Complement Naïve Bayes, Naïve Bayes Simple and Naïve Bayes.

Bayes Net learns Bayesian networks under the assumptions: nominal attributes (numeric ones are prediscretized) and no missing values (any such values are replaced globally). There are two different algorithms for estimating the conditionally probability tables of the network. Search is done using K2 or the TAN algorithm or more sophisticated methods based on hill-climbing, simulated annealing, *tabu* search and genetic algorithm [3].

Complement Naive Bayes builds a Complement Naïve Bayes classifier as described by Rennie et al. [72]. This system is proposed to tackle poor assumptions of Naïve Bayes classifier: features are independent and attributes are nominal. Complement Naïve Bayes is a combination of Naive Bayes classifier and heuristic solutions to address both systemic issues as well as problems of multi-nominal model.

Naive Bayes [7] implements the probabilistic Naïve Bayes classifier while Naïve Bayes Simple uses the normal distribution to model numeric attributes. Naïve Bayes classifiers assume that the effect of a variable value on a given class is independent of the values of other variable. This assumption is called class conditional independence. It is made to simplify the computation and in this sense considered to be "Naïve". This assumption is a fairly strong assumption and is often not applicable. However, bias in estimating probabilities often may not make a difference in practice - it is the order of the probabilities, not their exact values that determine the classifications.

Table 7.5 Performances of Bayesian Classifier Approaches on imbalance datasets

Dataset	Bayes Net	Complement Naïve Bayes	Naïve Bayes Simple	Naïve Bayes
Echocardiogram	0.872	-	-	0.787
Breast-w	0.960	0.791	0.944	0.942
Wine3	0.979	0.596	0.960	0.960
Glass7	0.800	0.658	0.806	0.820
Vowel3	0.566	-	0.568	0.569
Hypothyroid1	0.922	-	0.800	0.810
Abalone19	0	-	0.021	0.018
Average	0.728	0.681	0.683	0.701

The performances of Bayesian Classifier Systems are shown in Table 7.5. Some datasets, not suitable for classifiers, are indicated by symbol "-". An overview, we can see that Bayesian Classifiers are not good for imbalance problems, at least for our datasets. They perform worse than our proposals and Decision Tree approaches. The Bayes Net achieved the best performance with 72.8% and Naïve Bayes is the closest second with 70.1%.

Although Complement Naïve Bayes is expected to perform well in difficult dataset, as imbalance problems, its results are low, even in *Wine3* dataset where all other classifiers reach to optimal solutions; Complement Naïve Bayes only reaches 59.6% of F1-measure. The same results occur when it classifies *Breast-w* and *Galss7*. Beside that, Naïve Bayes Simple is a simple method but it performs quite well. It can not classify *Echocardiogram* but, in *Breast-w*, *Wine3* and *Glass7*, it has compatible results to other methods.

In issue of training time, Bayesian Classifiers learn typically faster than XCS and Decision Trees, because they are straightforward method for supervised learning and are based on probability theory. They provide a flexible way for dealing with any number of attributes and classes. The benefit of using the normal method in Naïve Bayes is that is fast in both learning and classification time and requires little memory.

7.5 The Proposals and Neural Network, SVM and Nearest Neighbor Approaches

This is the last section comparing our work to other related works in classification field which are: *Neural Network, Support Vector Machine and Nearest Neighbor Algorithm.*

Neural networks [52] are predictive models loosely based on the action of biological neurons. There are many types of Neural Networks; in this article, we use Multi-Layer-Perceptron (MLP) that trains using back-propagation. MLP networks are general-purpose, flexible, nonlinear models consisting of a number of units organized into multiple layers. The complexity of the MLP network can be changed by varying the number of layers and the number of units in each layer. Given enough hidden units and enough data, it has been shown that MLPs can approximate virtually any function to any desired accuracy. The principle of the MLP is that when data from an input pattern is presented at the input layer the network nodes perform calculations in the successive layers until an output value is computed at each of the output nodes. This output signal should indicate which the appropriate class for the input data is i.e. we expect to have a high output value on the correct class node and a low output value on all the rest.

A *Support Vector Machine* (SVM) [59] performs classification by constructing an *N*-dimensional hyper-plane that optimally separates the data into two categories. SVM models are closely related to neural networks. In fact, a SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network. The algorithm we choose to analyze Support Vector Machine is Sequential Minimal Optimization (SMO). Training a SVM requires the solution of a very large quadratic programming (QP) optimization problem. SMO breaks this large QP problem into a series of smallest possible QP problems. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets.

The *nearest-neighbor algorithm* [60] is extremely simple to implement and leaves itself open to a wide variety of variations. In brief, the training portion of nearest-neighbor does little more than store the data points presented to it. When asked to make a prediction about an unknown point, the nearest-neighbor classifier finds the closest (according to some distance metric) training-point to the unknown point and predicts the category of that training-point.

The k-nearest-neighbor algorithm is a variation of nearest-neighbor. Instead of looking at only the point that is closest to input, we instead look at the k points in $S = \{x_1, x_2, \ldots, x_k\}$ that are closest to the input. Since we know the categories of each of the k nearest points, we find the majority category in this subset. If it exists, the query input point is predicted to be in that category. If no

majority exists, the result is considered inconclusive. k-nearest-neighbor is more general than nearest-neighbor. Put another way, nearest-neighbor is a special case of k-nearest-neighbor, where $k = 1$. For the tests in this thesis, we chose $k = 1$ and $k = 3$, called IB1 and IB3 where IB stands for instance-based learner. IBk uses the Euclidean distance to find the training instance closest to the input.

Table 7.6 Performances of Multi-Layer-Perceptron, Support Vector Machine and Nearest Neighbor Approaches on imbalance datasets

Dataset	MLP	SMO	IB1	IB3
Echocardiogram	0.805	0.827	0.738	0.795
Breast-w	0.924	0.955	0.929	0.952
Wine3	0.969	0.980	0.96	0.960
Glass7	0.847	0.807	0.842	0.836
Vowel3	0.852	0	0.983	0.939
Hypothyroid1	0.849	0.689	0.757	0.658
Abalone19	0	0	0	0
Average	0.749	0.608	0.744	0.734

Table 7.6 summarizes performances of Multi-Layer-Perceptron, Sequential Minimal Optimization and Nearest Neighbor Algorithms on our imbalance datasets. All 4 algorithms have high accuracy for the first five datasets (except SMO in *Vowel3*). In fact, in *Breast-w* and *Wine3*, all scored > 92%, especially, SMO scored 98% in *Wine3*. *Vowel3* is recorded as a difficult problem for the previous classifiers, but, MLP, IB1 and IB3 can score very high. IB1, a simple method, even performs on *Vowel3* better than all XCS-based classifiers. However, in the first 4 datasets, the XCS-based classifiers have been seen performing better than non-XCS machine learning. The highest performance in this section is Neural Network – MLP, with 74.9% while the second highest is IB1 with 74.4%. Because, IBk considers only k closest points in local area, its performance in imbalance dataset is quite good. We can observe that IB1 classifies minority instances better than IB3 in our datasets due to the number of minority instances in imbalance datasets are small.

In issue of training time, MLP and SMO are complex systems, so, the training time of these classifiers are long. Because SMO is a descendant of SVM, it has properties of SVM as: the training is slow, especially for large problems and training algorithm is complex, subtle and sometimes difficult to implement. Although SMO is fastest for linear SVMs and sparse data sets, it is slowest for

all classifiers we study in this thesis. In the other hand, IBk is a simple classifier; its training time is short even in large data with high number of instances.

7.6 Summary

In this chapter, we have shown that our proposals are capable of learning complex, non-linear, imbalance classification problems and that they can be used to accurately predict unseen cases. We have also directly compared the performance of our proposals with several popular machine learning algorithms and have shown the suitability of the genetic-based learning classifier system for imbalance classification tasks. Our methods significantly achieved better results than that of Decision Trees and Bayesian approaches. Besides, any method could not outperform our methods significantly. However, for LCS, a standard set of benchmark problems still does not exist. Such a set would give researchers the opportunity for a better comparability of their published methods and results. An appropriate benchmark set should be composed of real-world datasets taken from real problem domains as well as artificial problems in which the characteristics of the data are exactly known.

From the experimental results, we also observe that Nearest Neighbor methods are simple methods but get a good results when performing on imbalance problems. It is possible to find some properties of Nearest Neighbor methods and applying them to XCS to improve XCS's performance. Another issue needs to be studied in LCS field is identify what kind of problems XCS is well and poorly suited. This issue is derived when we study performance of XCS on *Hypothyroid1* dataset. While non-LCS approaches classify this dataset well, LCS approaches encounter problem.

Chapter 8

Conclusions and Future Researches

This thesis provides an empirical study of imbalance dataset problem, its impact on XCS Classifier System and we have proposed two methods to solve this problem: Cost-sensitive XCS and Sampling Ensemble XCS. This chapter summarizes the main contributions and lessons learned from this study. Limitations with this study are then discussed, including the reliance on XCS Classifier System throughout this thesis. In this discussion we do provide some reasons to believe that our results may generalize beyond this one important class of learners. Areas of possible future research are then discussed.

8.1 Summary of Contributions

The contributions of the research are best summarized by the evaluation of the research outcome, corresponding to the research objectives as stated in Section 1.2. Although the study of Cost-sensitive XCS and Sampling Ensemble XCS are separated, many of the main contributions for each are related, since the studies share the same general framework. We begin by describing the contribution of each algorithm.

- **Analysis effects of imbalance dataset problem to XCS classifier.** We first analyzed effects of imbalance datasets to performances of XCS. Imbalance dataset makes a poor performance of standard classifiers on classifying minority class and a high accuracy on the majority class but a very low, sometimes unacceptable, accuracy on the minority class. That is because standard learning classifier systems aim to minimize the overall error rate and in imbalance dataset, the majority class has dominant influence on the overall error than minority. From that, we provide guidelines to improve performances of XCS by setting parameters appropriately and combining with sampling techniques as well as ensemble learning.
- **Cost-sensitive XCS for bi-class imbalanced data.** We have introduced a Cost-sensitive XCS learning classifier system to solve imbalanced data by introducing cost items into the reward setting of XCS. In our approach, the reward value of correctly identifying the positive class outweighs the value of correctly identifying the common class. The adaptation reward

based on imbalance ratio allows maximizing the total reward of positive class in long runs, which has demonstrated to be useful in solving imbalance dataset problems.

- **Cost-sensitive XCS for multi-class imbalanced data.** In practice, there are some applications with more than two classes where the imbalanced class distribution hinders the classification performance. For classifying multi-class imbalanced data, the Cost-sensitive XCS for multi-class is developed by expanding Cost-sensitive XCS for bi-class.

  The results suggest that cost-sensitive learning is relatively easy on two-class tasks while it is difficult on multiclass tasks and a higher degree of class imbalance usually results in higher difficulty in cost-sensitive learning.

- **Under-sampling Ensemble XCS for Imbalanced data.** The well-known technique to solve imbalance problems is sampling. Under-sampling Ensemble XCS is a combination of Under-sampling and Ensemble learning with XCS to solve imbalanced data. Firstly, an overview of the system is presented. All components of the classifier are described. Secondly, the performance of system is described to understand how Under-sampling Ensemble XCS works. In which, Under-sampling techniques are employed to change the distribution of training datasets by under-sampling the majority class multiple time and inducing multiple classifiers by ensemble learning algorithm. Finally, seven UCI imbalance datasets described in the previous section are used to verify the feasibility of our proposals.

- **Over-sampling Ensemble XCS for Imbalanced data.** Over-sampling is second well-known sampling technique. Over-sampling Ensemble XCS is developed with a framework similar to that of Under-sampling Ensemble XCS on 2 steps: creating multiple classifiers and aggregating results; only step creating training subsets is different where Over-sampling technique is used to create training subsets instead of Under-sampling technique. The study in this dissertation shows that Over-sampling Ensemble XCS tends to outperform all other algorithms in solving our imbalanced data.

- **Experimental evaluations on the proposed algorithms.** Experiments are conducted on synthetic dataset as well as real-world datasets for evaluating our proposal methods in terms of accuracy improvement for both bi-class and multi-class imbalanced data in terms of the recognition performance on a specific minority class and the balanced performance among all classes.

When compared with other related algorithms, these experimental results indicate that the proposed algorithms are superior in achieving better measurements with respect to the learning objectives.

8.2 Discussion

In this section, we briefly discuss several additional issues of our framework that related to Cost-sensitive Learning, Sampling Technique and Ensemble Algorithms.

- **Cost-sensitive XCS**

A cost-sensitive classification technique takes the cost matrix into consideration during model building and generates a model. In these cases, it is important to create a classifier that minimizes the overall misclassification cost. This tends to cause the classifiers to perform better on the minority class than if the misclassification costs were equal. For highly skewed class distribution, this also ensures that the classifier does not only predict the majority class.

The empirical study presented in chapter 5 reveals that cost-sensitive learning is relatively easy on two-class tasks while hard on multiclass tasks. This is not difficult to understand because an example can be misclassified in more ways in multiclass tasks than it might be in two-class tasks, which means the multiclass cost function structure can be more complex to be incorporated in any learning algorithms. Note that multiclass problems can be converted into a series of binary classification problems, and methods effective in two-class cost-sensitive learning can be used after the conversion. However, this approach might be troublesome when there are many classes and a user usually favors a more direct solution. This is just like that although multiclass classification can be addressed by our cost-sensitive XCS for bi-class via pair-wise coupling; we still attempt to design multiclass cost-sensitive XCS and our future work is improve its performance. Nevertheless, an interesting future issue is to compare the effect of doing multiclass cost-sensitive learning directly and the effect of decoupling multiclass problems.

And from experiments, we can see that with the same parameter settings, XCS performs on 11-MUX datasets more robustly than on real datasets (imbalance ratio ir=16 comparing to ir=6). This is because of difference on overlapping and complexity of each dataset. In [75], we investigated on this approach to analyze effects of imbalance problems in combining with other factors as small disjunction, overlapping degree, training dataset size and complexity of problem to performance of classifiers. We thus suggest that addressing the class imbalance problem is not necessarily the best

way to deal with the problem of performance degradation in the presence of class imbalances. Instead, it may be more effective if we study imbalance problem in combination with studying small disjunction, overlapping degree and training dataset size.

- **Sampling Ensemble XCS**

From the eyes of statistician, sampling consists of selecting part of a population to observe so that one may estimate something about the whole population. In while, the ensemble scheme is a set of classifiers that can make lower the variation of each individual classifier so that the performance of the system can be more stable. Thus, Sampling Ensemble XCS can integrate the strength of both sampling and the ensemble scheme.

Two related tasks in sampling are (1) how best to obtain a sample, and once the sample data are in hand, (2) how best to use them to estimate the characteristics of the whole population. We should note that we might get very different results if we change the level of re-sampling. In particular, some re-sampling methods might perform better with a higher or lower amount of re-sampling. It is also quite likely that re-sampling until the minority class data makes up 50% of the training data does not yield optimal results [37]. However, to reach to optimal results by variety the level of re-sampling is still an open interesting issue.

In ensemble learning, clearly there is no advantage to combining a set of classifiers which are *identical*; *identical* that is, in that they generalize in the same way. The emphasis here is on the similarity or otherwise of the pattern of generalization. In principle, a set of classifiers could vary in terms of their weights, the time they took to converge, and even their architecture and yet constitute essentially the same solution, since they resulted in the same pattern of errors when tested on a test set. There are a number of parameters which can be manipulated in efforts to obtain a set of classifiers which generalize differently. These include the following: initial conditions, the training data, the typology of the nets, and the training algorithm. In XCS, we use different seed in creating new chromosome of genetic algorithm.

From experimental results in chapter 6, we can see that although we used the same ensemble learning algorithm, performance of Over-sampling Ensemble XCS is better than Under-sampling Ensemble XCS, that because XCS performs on over-sampling training dataset better than on under-sampling training datasets. So, we can conclude that for rare class scenarios, the ability of an ensemble algorithm is dependent on the abilities of its base learner.

8.3 Limitations and Future Researches

Although the experiments have provided evidence that the proposed method can be successful for learning from imbalanced data sets, future work is needed to address its possible drawbacks.

- **More investigations on other application domains.** In chapter 2, several application domains where the class imbalance problem prevails are listed. In our experimental study on classifying imbalanced data, most of the data sets are taken from the UCI machine learning repository. The proposed cost-sensitive XCS algorithm can also be applied to other application domains, such as fraud detection and network intrusion, to explore their effectiveness in these specific domains.
- **More investigations on other base classification systems.** In this thesis, XCS classifier system is investigated for classifying imbalanced data by using the proposed cost-sensitive algorithm, sampling techniques and ensemble learning. Other standard classification systems, such as Decision Trees, bayesian network classifier, neural networks, and support vector machines, are all reported to be affected by the class imbalance problem. The proposed cost-sensitive algorithm and combination of sampling and ensemble learning are applicable to any base classifier where Bagging and Boosting can be applied. As ensemble learning performs differently with respect to the base classification systems, further study can investigate how the cost-sensitive algorithm, sampling techniques and ensemble learning affect different base classification systems.
- **XCS classifier system.** XCS still suffers from higher training times, so further investigation should be done to reduce its training time. Some issues in this direction may include the use of reduction techniques that minimize the number of rules during training, especially in problems with real attributes.
- **Sampling Techniques.** Till now, that there are relative fewer sampling schemes used in data mining compared with the large variety of sampling methods with have been existed for a long time in the area of statistics. Furthermore, due to the inherent requirement of data mining, some conclusions of traditional sampling methodology may not apply or not much desirable. Therefore, more theoretical and empirical research is needed before we can

claim a thorough understanding of sampling and further its application in data mining. Further research on sampling in data mining should, include:

- To find the best sampling approach (including new sampling scheme) for varied data mining algorithms;
- Practical methods for determining optimal sample size for different learning tasks;
- Practical techniques for measuring the sample quality and controlling the loss of original information;
- Practical off-line and on-line sampling algorithms (possibly combining with data mining algorithms);
- Sampling from special data structures such as semi-structured data;
- Sampling in parallel or distributed databases;

There are of course many other worthwhile research possibilities that are not included in the list. We believe that because of the challenging topics and tremendous potential applications, the classification of imbalanced data will continue to receive more and more attention in both the scientific and the industrial worlds.

References

[1]. Michael Negnevitsky, Artificial Intelligence A Guide to Intelligent Systems, 2nd ed. Addison- Wesley, 2005.

[2]. Tom M. Mitchell, Machine learning, McGraw-Hill, 1997.

[3]. P.N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison Wesley, USA, 2006.

[4]. P.L. Lanzi, W. Stolzmann, S.W. Wilson, Springer Berlin/Heidelberg, Learning Classifier Systems: From Foundations to Applications, 2000.

[5]. Goldberg, David E., Genetic Algorithms in Search, Optimization and Machine Learning, Kluwer Academic Publishers, Boston, MA, 1989.

[6]. Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[7]. Ian H. Witten, Eibe Frank, Data Mining: Practical Machine Learning Tools and Techniques Second Edition, Morgan Kaufmann Publishers, June, 2005.

[8]. Stewart W. Wilson, "Classifier Fitness Based on Accuracy", *Evolutionary Computation*, Vol.3(2), 1995, pp.149-175.

[9]. Butz, M. and Wilson, S., "An algorithmic description of XCS". *Journal of Soft Computing 6*, 2002, pp.144-153.

[10]. Stewart W. Wilson, "Generalization in the XCS Classifier System", *Genetic Programming 1998: Proceedings of the Third Annual Conference,* J. Koza et al. (Eds.) San Francisco, CA: Morgan Kaufmann, 1998, pp.665-674.

[11]. S. W. Wilson, "Get real! XCS with continuous-valued inputs", in *Festschrift in Honor* of John H. Holland, L. Booker, S. Forrest, M. Mitchell, and R. Riolo, Eds. Ann Arbor, MI: Center for the Study of Complex Systems, Univ. Michigan, 1999, pp. 111–121.

[12]. S. W. Wilson, "Mining oblique data with XCS," *in Lecture Notes in Artificial Intelligence*, P. Lanzi, W. Stolzmann, and S. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2001, vol. 1996, Proc. 3rd Int. Workshop, Adv. Learn. Classifier Syst., pp. 158–176.

[13]. M. V. Butz. "Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design", vol. 109 of *Studies in Fuzziness and Soft Computing*. Springer, 2006.

[14]. M. Butz, T. Kovacs, P.L. Lanzi, W. Stewart, "Toward a theory of generalization and learning in XCS", IEEE Trans. Evolutionary Computation 8(1), pp. 28-46, 2004.

[15]. P. Lanzi. "Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding". *Proceedings of the Genetic and Evolutionary Computation Conference 1999*, pp. 337-344, Orlando, Florida, 1999. Morgan Kaufmann.

[16]. P. Lanzi and A. Perrucci. "Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions". *In Proceedings of the Genetic and Evolutionary Computation Conference 1999*, pp. 345-352, Orlando, Florida, 1999. Morgan Kaufmann.

[17]. S. Wilson. "Compact Rulesets for XCSI". In *the Fourth International Workshop on Advances in Learning Classifier Systems*, vol. 2321 of Lecture Notes in Artificial Intelligence, pp. 197-210. Springer-Verlag, 2001.

[18]. A. Bonarini. "An Introduction to Learning Fuzzy Classifier Systems". In *From Foundations to Applications*, vol. 1813 of Lecture Notes in Artificial Intelligence, pp. 83-104. Springer-Verlag, 2000.

[19]. A. Bonarini, C. Bonacina, and M. Matteucci. "Fuzzy and Crisp Representations of Real-Valued Input for Learning Classifier Systems". *From Foundations to Applications*, vol. 1813 of Lecture Notes in Artificial Intelligence, pp. 107-124. Springer-Verlag, 2000.

[20]. L. Bull and T. O'Hara. "Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems". In *Proceedings of the Genetic and Evolutionary Computation Conference 2002*, pp. 905-911, New York, USA, 2002. Morgan Kaufmann.

[21]. Wilson, S. W. "ZCS: a Zeroth order Classifier System". *Evolutionary Computation*, 2, pp. 1-18, 1994.

[22]. Wilson, S. W. "Function approximation with a classifier system". *Proceedings of the Third Genetic and Evolutionary Computation Conference* (GECCO-2001), 974-981, 2001.

[23]. Ester Bernadó-Mansilla, Tin Kam Ho, "Domain of competence of XCS classifier system in complexity measurement space". *IEEE Trans. Evolutionary Computation* 9(1): 82-104 2005.

[24]. E. Bernadó-Mansilla, J.M. Garrell Guiu. "Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks". *Evolutionary Computation 11*, vol. 3, pp. 209-238, 2003.

[25]. Ester Bernado, Xavier Llora, and Josep M. Garrell. "XCS and GALE: A Comparative Study of Two Learning Classifier Systems on Data Mining". In Advances of Learning Classifier Systems, volume 2321 of LNAI, pp. 115-132, Springer-Verlag, Berlin, 2002.

[26]. Orriols-Puig A., Goldberg D.E., Sastry K., and Bernadó-Mansilla E. "Modeling XCS in Class Imbalances: Population Size and Parameter Settings". *Technical report*, IlliGAl Report No. 2007001, USA, Feb. 2007.

[27]. Albert Orriols and Ester Bernado-Mansilla, "Data Imbalance in UCS Classifier System: Fitness Adaptation", *IEEE Congress on Evolutionary Computation - CEC2005*, vol. 1, pp. 604-611, 2005.

[28]. Albert Orriols-Puig A. and Ester Bernadó-Mansilla, "Bounding XCS's Parameters for Unbalanced Datasets". *GECCO'06*, pp. 1561 - 1568. July 8–12, 2006, Seattle, Washington, USA.

[29]. Albert Orriols-Puig, Ester Bernadó-Mansilla, "The class imbalance problem in learning classifier systems: a preliminary study", *GECCO'05*, pp. 74-78, Washington, D.C, USA, 2005.

[30]. Holmes, J.H. "Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates". In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 635–642. Morgan Kaufmann, 1998.

[31]. Kamran Shafi, Tim Kovacs, Hussein A. Abbass, Weiping Zhu, "Intrusion detection with evolutionary learning classifier systems", *Journal of Natural Computing,* Springer Verlag, in press, expected Publication Date 2008

[32]. N. Japkowicz, "Concept-learning in the presence of between-class and within-class imbalances", in: *Proceedings of the Fourteenth Conference of the Canadian Society for Computational Studies of Intelligence*, Ottawa, Canada, June 2001, pp. 67–77.

[33]. G. Weiss, F. Provost, "Learning when training data are costly: the effect of class distribution on tree induction", *Journal of Artificial Intelligence Res. 19*, 2003. pp. 315–354.

[34]. Japkowicz, N. "The Class Imbalance Problem: Significance and Strategies". In *Proceedings of the 2000 International Conference on Artificial Intelligence - IC-AI'2000*: Special Track on Inductive Learning Las Vegas, Nevada, 2000.

[35]. Cohen, W. "Learning to Classify English Text with ILP Methods". In *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pp. 3–24. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[36]. R. Agarwal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases". In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207-217, 1993.

[37]. N. Japkowicz and S. Stephen, "The Class Imbalance Problem: A Systematic Study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429-450, 2002.

[38]. R.C. Prati, G.E.A.P.A. Batista, and M.C. Monard, "Class Imbalances versus Class Overlapping: An Analysis of a Learning System Behavior", *the Third Mexican International Conference on Artificial Intelligence - MICAI 2004*, LNAI 2972, pp. 312–321, 2004.

[39]. N. Japkowicz, "Learning from imbalanced data sets: a comparison of various strategies," *Learning from Imbalanced Data Sets: The AAAI Workshop*, pp. 10-15, 2000.

[40]. C. Phua, D. Alahakoon, and V. Lee, "Minority report in fraud detection: classification of skewed data," *SIGKDD Explorations*, vol. 6, no.1, pp. 50-59, 2004.

[41]. G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. "A study of the behavior of several methods for balancing machine learning training data". *SIGKDD Explorations*, 6(1):20-29, 2004.

[42]. J. W. Grzymala-Busse, J. Stefanowski, and S. Wilk, "A comparison of two approaches to data mining from imbalanced data," *Lecture Notes in Computer Science*, vol. 3213,pp. 757-763, 2004.

[43]. Mladeni´c, D., & Grobelnik, M. "Feature Selection for Unbalanced Class Distribution and Naive Bayes". In *Proceedings of the 16th International Conference on Machine Learning*, pp. 258–267. Morgan Kaufmann, 1999.

[44]. N. V. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp.321-357, 2002.

[45]. M. A. Maloof, "Learning when data sets are imbalanced and when costs are unequal and unknown," *ICML-2003 Workshop on Learning from Imbalanced Data Sets II*, 2003.

[46]. N. V. Chawla. "C4.5 and imbalanced datasets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure." *In Proceedings of the ICML'03 Workshop on Class Imbalances*, 2003.

[47]. C. Drummond and R. Holte. "C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling." In *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets*, 2003.

[48]. K.M. Ting, "An Instance-weighting Method to Induce Cost-Sensitive Trees", *IEEE Trans. Knowledge and Data Eng.*, vol. 14, no.3, pp. 659-665, Apr./May 2002.

[49]. M. Kubat and S. Matwin. "Addressing the curse of imbalanced training sets: One sided selection". In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179-186, Nashville, Tennesse, 1997. Morgan Kaufmann.

[50]. Zhang, J., Mani, I. "kNN Approach to Unbalanced Data Distributions: A Case Study involving Information Extraction", *ICML*, 2003.

[51]. Machine Learning—"A Technological Roadmap", L. Saitta, ed. The Netherlands: Univ. of Amsterdam, 2000.

[52]. D. Michie, D. J. Spiegelhalter, and C. C. Taylor. "Machine Learning, Neural and Statistical Classification". *Ellis Horwood*, 1994.

[53]. Bianca Zadrozny and Charles Elkan. "Learning and making decisions when costs and probabilities are both unknown". *Technical Report CS2001-0664*, Department of Computer Science and Engineering, University of California, San Diego, January 2001.

[54]. A. Beygelzimer, J. Langford, and P. Ravikumar. "Multiclass classification with filter trees". Manuscript. Available at URL: http://hunch.net/~jl/projects/reductions/mc_to_b/invertedTree.pdf, 2007.

[55]. A. van den Bosch, T. Weijters, H. J. van den Herik, and W. Daelemans. "When small disjuncts abound, try lazy learning: A case study". In *Proceedings of the Seventh Belgian-Dutch Conference on Machine Learning*, pp. 109-118, 1997.

[56]. R. C. Holte, L. E. Acker, and B. W. Porter. "Concept learning and the problem of small disjuncts". In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 813-818, 1989.

[57]. N.V. Chawla, N. Japkowicz, A. Kolcz, "Editorial: special issue on learning from imbalanced data sets", *SIGKDD Explorations Special Issue on Learning from Imbalanced Datasets 6* (1), 2004 1–6.

[58]. DeRouin, E., Brown, J., Fausett, L., & Schneider, M. "Neural Network Training on Unequally Represented Classes". In *Intelligent Engineering Systems through Artificial Neural Networks*, pp. 135–141 New York. ASME Press, 1991.

[59]. B. Liu, W. Hsu, and Y. Ma. "Mining association rules with multiple minimum supports". In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 337-341, 1999.

[60]. Cover, T. M., Hart, P. E. "Nearest neighbor pattern classification", *IEEE Trans. on Information Theory 13*, pp. 21-27, 1967.

[61]. Cohen, W. W. 1995. "Fast effective rule induction". In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA. San Francisco: Morgan Kaufmann, pp. 115-123.

[62]. Breiman L. "Random forests". *Machine Learning 2001*, 45(1): pp. 5-32

[63]. Quinlan, J. R., "Bagging, Boosting, and C4.5". In *Proceedings of the Thirteenth Naional Conference on Artificial Intelligence*, pp. 725-730, 1996

[64]. Bauer, E. and Kohavi, R., "An Empirical comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants". *Machine learning*, 35: pp. 1-38, 1999.

[65]. Domingos, P. "Metacost: A General Method for Making Classifiers Cost-sensitive". In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 155–164 San Diego, CA. ACM Press., 1999.

[66]. Sutton, R. & Barto, R., Reinforcement Learning. MIT Press, 1998.

[67]. K. M. Ting. "The problem of small disjuncts: its remedy in decision trees". In *Proceeding of the Tenth Canadian Conference on Artificial Intelligence*, pp. 91-97, 1994.

[68]. G. M. Weiss and H. Hirsh. "A quantitative study of small disjuncts". In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pp. 665-670, AAAI Press, 2000.

[69]. B. Liu, W. Hsu, and Y. Ma. "Mining association rules with multiple minimum supports". In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp. 337-341, 1999.

[70]. J. H. Friedman, R. Kohavi, and Y. Yun. "Lazy decision trees". In P*roceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 717-724, 1996.

[71]. Lashon B. Booker. "Intelligent behavior as an adaptation to the task environment". PhD Thesis, The University of Michigan, 1982.

[72]. Rennie, J. D. M., L. Shih, J. Teevan, and D. R. Karger. "Tackling the poor assumptions of Naïve Bayes text classifiers". In *Proceedings of the Twentieth International Conference on Machine Learning*, Washington, DC., CA: AAAI Press, pp. 616-623, 2003.

[73]. Lewis, D., & Catlett, J. Heterogeneous Uncertainty Sampling for Supervised Learning. In Proceedings of the Eleventh International Conference of Machine Learning, pp. 148–156 San Francisco, CA. Morgan Kaufmann, 1994.

[74]. UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets.html.

[75]. Thach Huy Nguyen, Sombut Foitong, Sornchai Udomthanapong and Ouen Pinngern, "Effects of Distance between Classes and Training Datasets Size to the Performance of XCS: Case of Imbalance Datasets", *IAENG International Conference on Artificial Intelligence and Applications*, vol. I, pp. 468-473, Hong Kong, 19-21 March, 2008.

[76]. Thach Huy Nguyen, Porntep Rojanavasu and Ouen Pinngern, "Cost-sensitive XCS Classifier System Addressing Imbalance Problems", *The 5th International Conference on Fuzzy Systems and Knowledge Discovery – FSKD'08*, China, 18-20 October, 2008 (To appear)

[77]. Hai H. Dam, Hussein A. Abbass, Chris Lokan, Xin Yao: "Neural-Based Learning Classifier Systems". *IEEE Transactions Knowledge Data Engineering* 20(1): 26-39, 2008

[78]. R.P.W. Duin J. Kittler, M. Hater and J. Mates, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 20, no. 3, 1998.

[79]. D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, 1992.

Appendix A

Notation and Parameters of XCS

A.1 Problem Notation

S	problem space
$s \in S$	problem instance
$S = \{0, 1\}^l$	binary problem space of length l
l	problem length l (that is, number of features in a problem instance)
A	problem classes
a $\in$ A	current class
A = {0, 1}	binary (two-class) problem
n	number of problem classes

A.2 XCS Classifier System

LCS Parameters

N	maximal population size
$P_{\#}$	don't care probability
P_I, ε_I, F_I	default parameter initialization values
ε	e-greedy strategy parameter
β	learning rate
$\alpha, \varepsilon_0, \nu$	accuracy determination parameters
θ_{GA}	threshold that controls GA invocation
μ	probability of mutating a condition attribute (or the action)
χ	probability of applying the chosen crossover operator
θ_{del}	threshold that requires minimal experience for fitness influence during deletion
δ	fraction of mean fitness below which deletion probability is further decreased
θ_{sub}	threshold that requires minimal experience for subsumption

Classifier Parameters

C	condition part – in binary domains, $C \in \{0, 1, \#\}^l$
A	action part $A \in$ A
R	reward prediction

ε	mean absolute reward prediction error
F	fitness (in macro classifiers)
as	the mean action set size the classifier is part of
ts	the time the classifier was part of an action set in which the GA was applied
exp	the number of evaluation steps the classifier underwent so far
num	the numerosity, that is, the number of micro-classifiers represented by this (macro-) classifier

Other Notations

$[P]$	classifier population
$[M]$	match set
$[A]$	action set
$\sigma(cl)$	specificity of condition part of classifier *cl*; in binary domains, specificity equals to the number of specified attributes over *l*
$\sigma[X]$	average specificity in classifier set X
κ	the current accuracy of a classifier
κ'	the current set-relative accuracy of a classifier
ρ	the (combined) reward received by the current action set
$P(\mathrm{A})$	prediction array estimating the value of each possible action A
$P(A)$	the predicted value of action A

Appendix B

Implementation of the XCS Classifier System in C/C++

This appendix explains how to download, compile and use the XCS code version in C/C++. It discusses how to select various parameter settings, how to add and remove certain procedures in the XCS, how to apply the XCS in the imbalanced multiplexer environment and real-world dataset environments. All instructions are suited for UNIX operating systems. But, as the code is written in C/C++ it should not cause problems to compile and run the code on other machines.

B.1 How to Download, Extract, Compile and Run the Code

The package with the source code and some examples is available at the Illinois Genetic Algorithms Laboratory, Anonymous FTP site in ftp://ftp-illigal.ge.uiuc.edu/pub/src/XCS/XCS.tar.Z.

- **Download and Extract**

After downloading the package (XCS.tar.Z) into a directory, the directories and files can be extracted by typing:

```
uncompress XCS.tar.Z
tar xvf XCS.tar
```

When extraction the files a new directory, called XCS, will be created that contents the following:

Environments	classifierList.c	xcs.c
Makefile	classifierList.h	xcs.h
actionSelection.c	env.c	xcsMacros.h
actionSelection.h	env.h	

The implementations of the multiplexer and the woods environment can be found in the Environments subdirectory. The file Woods1.txt and Woods2.txt in this subdirectory represent the

Woods1 and Woods2 environment. Because in this thesis we only study XCS in classification (single-step), we do not describe how to implement XCS in multi-step tasks, as Woods1 and Woods 2, here.

- **Compile**

The compilation works with the provided Makefile that can be found in the created XCS directory together with the source code. A simple call of make will compile the files and create the executable program xcs.out. A call of make clean deletes all *.o, *.out, and *~ files. A previous call of make clean before make is not necessary, though.

The Makefile can be modified as follows:

+ Line 17: In the statement 'CC = ...' the preferred C compiler can be chosen. By default the 'gcc' compiler is chosen here.

+ Line 18: In the statement 'CC_OPTS = ...' diverse options can be chosen dependent on the compiler.

- **Choosing an Environment**

The Makefile chooses by default the environment with source code env.c and env.h from the XCS directory. By default env.c and env.h represent the Multiplexer environment. To change this to other environment, simply copy from the Environments subdirectory the files woodsXCS.c and woodsXCS.h to env.c and env.h, respectively and recompile the code. Then the XCS learns in that environment. To change back to the Multiplexer environment then again, copy the files multipXCS.c and multipXCS.h from the Environments subdirectory to env.c and env.h and recompile again.

- **Run the Code**

After the code is compiled with the intended environment, the program can be executed by typing:

```
xcs.out
```

B.2 Constants and Macros

This section describes how to modify the XCS program easily by the provided constants. Furthermore, how to modify the environments to match perfectly into the XCS program is presented.

- **The xcsMacros.h File**

The XCS classifier system uses a few constants for the different methods. The constants are all defined in the xcsMacros.h file where they can all be modified. Moreover, xcsMacros.h provides some other constants with that certain methods in the XCS mechanism can be selected or deselected.

The following manipulations are possible:

+ Line 13: 'TABOUTFILE' is the file where the statistics are going to be printed out.

+ Line 15: 'NR EXPS' is the number of experiments that are successively taken.

+ Line 16: 'MAX NR STEPS' is the number of problems to solve in one experiment.

+ Line 18: 'MAX POP SIZE' is the maximum size of the population.

+ Line 20: 'BETA' is the used learning rate for prediction, prediction error, fitness and peer set size updates.

+ Line 22: 'MEAN LAST GATIME' is the threshold to do a GA is the average number of time-steps in a set since the last GA is greater than the threshold.

+ Line 23: 'MINIMUM ERROR' is equivalent to ε_0 in Wilson's parameters.

+ Line 24: 'FALL OFF RATE' is equivalent to α in Wilson's parameters.

+ Line 26: 'CROSSPX' is the probability of doing crossover, if applicable.

+ Line 27: 'MUTPMUE' is the probability to mutate one bit.

+ Line 29: 'DELTA' is part of the second deletion method. It is used to enforce the deletion of low fitness classifiers.

+ Line 30: 'DELETION EXPERIENCE' is the minimum experience required to use the second deletion method.

+ Line 32: 'PREDICTION THRESHOLD' is the threshold to do covering, if the average prediction decreases under it.

+ Line 34: 'DONTCARE PROBABILITY' is the probability of choosing a don't care sign in generating a new classifier with covering.

+ Line 36: 'PRE INI' is the initial prediction in new classifiers.

+ Line 37: 'PREER INI' is the initial prediction error in new classifiers.

+ Line 38: 'FIT INI' is the initial fitness in new classifiers.

+ Line 40: 'PRE ERROR REDUCTION IN NEW CLASSIFIERS' is the amount of reducing the prediction error in a classifier that was generated with the genetic algorithm.

+ Line 41: 'FITNESS REDUCTION IN NEW CLASSIFIERS' is the amount of reducing the fitness in a classifier that was generated with the genetic algorithm.

+ Line 42: 'KAPPA MULTIPLIER' is the multiplier to reduce the accuracy of a classifier, if its prediction error is less than MINIMUM ERROR*PAYMENT_RANGE.

+ Line 44: 'REDUCTION THRESHOLD' is the threshold to emphasize the usage of more experienced classifiers in the prediction array (currently set to 0).

+ Line 45: 'FITNESS REDUCTION' is the reduction that is used to reduce the contribution of a classifier to the prediction array if its experience is lower than the reduction threshold.

+ Line 47: 'SUBSUME EXPERIENCE' is the experience required to serve for the subsumption deletion method.

+ Line 49: 'INITIALIZE POP' is a Boolean that specifies if the population should be generated randomly at first (1) or if an experiment should start with an empty population.

+ Line 51: 'DECREASE FITNESS IN M NOT A' is a Boolean that specifies if a classifiers fitness is reduced, if it is in the match set but not in the action set (currently set to 0).

+ Line 52: 'FITNESS REDUCTION IN M NOT A' is the amount a classifier's fitness will be reduced, if it is in a match set but not in the action set and if the Boolean 'DECREASE FITNESS IN M NOT A' is set to 1.

+ Line 54: 'DECREASE FITNESS IF MUTATED' is a Boolean that specifies if a classifier's fitness should be decreased if it was mutated (not changed by crossover).

- **The Multiplexer Constants**

The constants of the Multiplexer environment are found in the header of the environment (either multipXCS.h or env.h once copied to the XCS directory). The size of the environment can be specified and the type of payment given.

+ Line 14: 'CONDITION LENGTH' is the length of the situations generated by the environment. The multiplexer environment is defined for length 6, 11, 20, 37....

+ Line 15: 'ACTION LENGTH' is the length of the action string that is possible to generate. In the multiplexer environment this should always be 1.

+ Line 16: 'NUMBER OF ACTIONS' is the different number of actions possible. This macro is used e.g. for the prediction array.

+ Line 17: 'NRBITS' is the number of bits that determine the position of the result in a string. (e.g. 6 mp. = 2 bits, 11 mp. = 3 bits ...)

+ Line 19: 'PAYMENT RANGE' is the maximum payment receivable from the environment.

+ Line 20: 'PAYOFF LANDSCAPE' is a Boolean that determines, if a payoff landscape for the rewards should be constructed similar to Wilson [8] or rather a simple reward should be given when the action was correct.

+ Line 22: 'IS MULTI STEP' is a Boolean that determines whether this is a multi-step environment. Therefore, this Boolean should be 0 in the multiplexer environment and 1 in the woods environment.

B.3 An Introduction to the Code

This section summarizes the different source code files of the XCS code. Furthermore, it gives an introduction to the implementation of the multiplexer tasks. Finally, it outlines how to plug-in other test environments.

- **The XCS code**

+ xcs.c, xcs.h files

In the xcs.c / xcs.h files the general flow of the XCS classifier system is implemented. It contains the 'main' method. In 'startExperiments' the specified number of experiments are executed and the execution distinguishes between single-step and multi-step environments. In 'startOneSingleStepExperiment' the execution of one single-step experiment is implemented and subsequently divided into the execution of one problem in explore or exploit mode in the functions 'doOneSingleStepProblemExplore' and 'doOneSingleStepProblemExploit'. Similar, in 'startOneMultiStepExperiment' the execution of one multi-step experiment is implemented and subsequently the execution of one explore and one exploit problem in the functions 'doOneMultiStepProblemExplore' and 'doOneMultiStepProblemExploit'. Furthermore, the 'writePerformance' function writes the performance out to the file specified in xcsMacros.h. 'randomize' randomizes the pseudo random generator.

+ classifierList.c, classifierList.h files

In these files, all functions related to the classifier set are implemented. At first, the creation of a random classifier list is specified. Then, the match set procedures and the action set procedures are coded. After that, the discovery component is implemented where the genetic algorithm is applied. The different subsume functions realize the procedure of subsumption deletion. Then the actual crossover and the mutation functions are given. How to add a classifier or a classifier pointer is realized in the three 'addClassifier' functions. Then, deletion functions

are realized. First, functions that select a certain classifier for deletion and then functions that actually delete a classifier or a whole set of classifiers. Finally, output functions and other utilities are implemented.

+ actionSelection.c, actionSelection.h files

These files handle action procedures. The first function generates a prediction array. Then, diverse functions for determining the action winner are given. Finally, two functions provide the conversion of a represented action from a string to an integer and backwards.

- **The Implemented Environments**

Provided with the code are the Multiplexer environment and an implementation of a Woods environment, which offers the application of diverse Woods environments. As outlined in B.1 the two environments are provided in the subdirectory Environments in the directory XCS. In order to apply the XCS in either of them the environment with the corresponding, the header must be copied to the XCS directory to the file env.c / env.h.

+ The Multiplexer Environment

The function 'resetState' generates a new randomly generated situation. 'doAction' executes an action on the actual situation and returns the received reward and if the action was the correct (better) one. 'initEnv' and 'freeEnv' normally initialize the environment and have always to be provided. In this case they don't initialize / free anything.

+ The Woods Environment

We do not concern this environment kind in classification tasks.

- **How to Plug-In other Test Environments**

It is very simple to plug-in other test environments. Any environment needs to be copied to the env.c and the corresponding header to the env.h file. The header must provide the following constants:

+ CONDITION LENGTH: This macro must specify the number of bits that specify a state in the environment.

+ ACTION LENGTH: This macro must specify the number of bits that are used for an action.

+ NUMBER OF ACTIONS: Here, the number of different possible actions must be specified. This is needed for the construction of the prediction array.

+ PAYMENT RANGE: This macro specified the highest amount that can be received as a reward.

+ IS MULTI STEP: This Boolean macro specifies if the environment is a single-step or multi-step environment (0: single-step, 1: multi-step).

Moreover, the environment must include the following functions:

+ int initEnv(FILE *fp): This function initializes the environment and is always called from the xcs program before the experiments start. It returns true (1) if the initialization was sucessfull.

+ void freeEnv(): is always called in the end of all experiments and is supposed to free the memory which was allocated by the environment.

+ void resetState(char *state): This function should execute a reset in the environment. In a single-step environment this function is called before each problem. In a multi-step environment it is called in the beginning of an experiment and when a new trial starts (i.e. when the 'doAction' function returns that a reset must be executed, e.g. in the woods environment, when reward was received).

+ double doAction(char *state, char *action, int *ret2): This function executes the specified action in the environment. It sets the 'state' array to the perceived string after the execution of the action. Moreover, it returns the received reward. 'ret2' in a single-step environment returns if the correct (better) action was executed. In a multi-step environment 'ret2' should specify, if a reset should be executed or not.

Once these functions are properly implemented, the new environment can be plugged in to the XCS mechanism. Even the output produced by the XCS should give a correct picture of the performance of the XCS in the new environment.

B.4 The Structure of the Output

The XCS code always produces an output of the performance in the environment in that it is applied in. The output is written to the file specified in 'TABOUTFILE' in the xcsMacros.h file (line 13).

Every 50 problems the XCS code produces some output using the 'writePerformance' function. Each line in the output represents the performance at a certain point in time. The different values are separated by ';'.

In a single-step environment it writes out:

+ The number of the so far examined problems.

+ The percentage of correctness in the last 50 problems.

+ The error in the predictions averaged over the last 50 problems.

+ The current size of the population divided by the payment range.

Appendix C

Imbalance Dataset Problems

C.1 Multiplexer Problem

The multiplexer problem has been the subject of a good deal of research in the machine learning field in general and in the classifier system field in particular. The idea of multiplexer problem is from digital circuit. It is introduced to apply in classifier system by Wilson [8]. There are many versions of the multiplexer problem, each with a different size of the form $k + 2^k$, for $k \geq 1$.

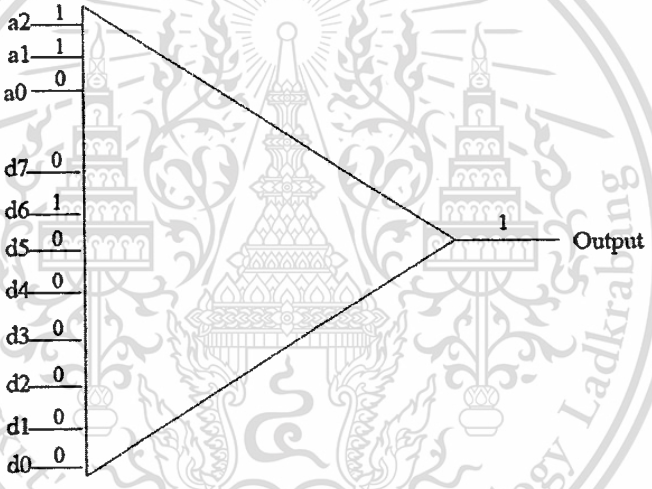


Figure C.1 Boolean 11-multiplexer with input of 11001000000 and output of 1.

Examples of multiplexer sizes, beginning with the smallest and increasing in size, are the 3-multiplexer, the 6-multiplexer, the 11-multiplexer, the 20-multiplexer, the 37-multiplexer, the 70-multiplexer, etc. For brevity, we call these problems as the "3-MP", the "6-MP", the "11-MP", and so forth, respectively. The difficulty of a multiplexer problem is correlated with its size. The smaller multiplexer problems are fairly simple to solve. For researchers who have used classifier systems to solve the multiplexer problem, problems with interesting levels of difficulty begin with the 11-MP. The multiplexer has 11 inputs and one output. The first three inputs, a_0, a_1, and a_2, can be considered as address lines. They describe the binary representation of an integer between 0 and 7. This integer chooses one of the 7 remaining inputs, which are labeled d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7. The correct output for the multiplexer is the input on the line specified by the address lines. A picture is shown in Figure

C.1. In this picture, the address lines specify the integer 6, so the output should be the input on line d_6, which is 1. There are $2^{11} = 2048$ possible combinations for the 11 arguments of the Boolean 11-multiplexer function.

So in generally, a multiplexer problem generates messages – lists of bits of length $k + 2^k$, where k is a positive integer. The size of a multiplexer problem (MP) is the length of the messages it generates. An algorithm learning to solve a standard MP receives such messages from the MP and learns whether to respond with a "1" or a "0" to any messages it may receive. The MP decides what the correct reply is in this way:

1. The MP treats the first k bits (called the address bits) of its message as the encoding of an integer in base 2. The MP converts the address bits of the message to i, the integer that the address bits encode in base 2.

2. The MP treats the last 2^k bits (the data bits) of its message as a list of possible responses. Bit number i in this list is the data bit for a message with address bits encoding i in binary.

3. If the learning algorithm provides a response equal to the value of signal bit number i, the MP provides positive reinforcement to the learning algorithm. If the learning algorithm provides a response different from the value of signal bit number i, the MP provides negative reinforcement to the learning algorithm.

Here are some examples of messages and the correct response to them for the 11-MP. (11 is of the form $3 + 2^3$, so each message generated by an 11-MP will have three address bits and eight data bits.) Note that here and in what follows, extra spaces are inserted to improve readability between the address bits and the data bits. These spaces are not represented in the signals themselves.

1 1 1 1 1 1 1 1 1 1 0: the 3 address bits encode 7 in base 2. The relevant data bit is the last one, so the correct response is "0". (Note that the counting starts from zero: the first data bit is bit number zero, and the last is data bit number 7).

0 1 1 1 1 1 1 0 0 0 0: the 3 address bits encode 3 in base 2. The relevant data bit is the final one of the initial block of 1s, so the correct response is "1".

0 0 1 0 1 0 1 0 1 0 1: the 3 address bits encode 1 in base 2. The relevant data bit is the first with the value of 1, so the correct response is "1".

The messages generated by a standard MP are randomly generated. A standard MP of size m will generate any message of length m with equal probability.

C:2 UCI Datasets

In this thesis, we have used 13 UCI datasets both bi-class and multi-class which are: *Echocardiogram*, *Breast-w*, *Wine*, *Glass*, *Vowel*, *Hypothyroid*, *Abalone*, *E-coli*, *Letters Recognition*, *Lymphography*, *Opt-digits*, *Pen-digits* and *Satellite*. Following is description of these datasets:

1. Echocardiogram (Year: 1989)

Concept: All the patients suffered heart attacks at some point in the past. Some are still alive and some are not. The survival and still-alive variables, when taken together, indicate whether a patient survived for at least one year following the heart attack.

Table C.1 Echocardiogram

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	survival	Real	[0, 57]	Class 0	88 (66.7%)
2	still-alive	Integer	[0, 86]	Class 1	44 (33.3%)
3	age-at-heart-attack	Category	{0, 1}	*Total*	132 (100%)
4	pericardial-effusion	Real	[0, 1]		
5	fractional-shortening	Real	[0, 40]		
6	epss	Real	[0, 7]		
7	lvdd	Real	[0, 39]		
8	wall-motion-score	Real	[0, 3]		
9	wall-motion-index	Real	[0, 2]		
10	mult	Category	{name}		
11	name	Integer	[0, 2]		
12	group	Integer	[0, 2]		

2. Breast-w (Year: 1984)

Concept: Each record represents follow-up data for one breast cancer case.

These are consecutive patients seen by Dr. Wolberg since 1984, and include only those cases exhibiting invasive breast cancer and no evidence of distant metastases at the time of diagnosis.

Table C.2 Breast-w

No.	Attribute	Class

	Name	Type	Range/Values	Name	Distribution (%)
1	Sample code number	Integer	[1, 10]	Benign	458 (65.5%)
2	Clump Thickness	Integer	[1, 10]	Malignant	241 (34.5%)
3	Uniformity of Cell Size	Integer	[1, 10]	*Total*	699 (100%)
4	Uniformity of Cell Shape	Integer	[1, 10]		
5	Marginal Adhesion	Integer	[1, 10]		
6	Single Epithelial Cell Size	Integer	[1, 10]		
7	Bare Nuclei	Integer	[1, 10]		
8	Bland Chromatin	Integer	[1, 10]		
9	Normal Nucleoli	Integer	[1, 10]		
10	Mitoses	Integer	[1, 10]		

3. **Wine** (Year: 1991)

Concept: These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars.

Minority class: Class 3 (26.97%)

Table C.3 Wine

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	Alcohol	real	[11, 15]	Class 0	59 (33.14%)
2	Malic acid	real	[0, 6]	Class 1	71 (39.89%)
3	Ash	real	[1, 4]	class 3	48 (26.97%)
4	Alcalinity of ash	real	[10, 30]	*Total*	178 (100%)
5	Magnesium	integer	[70, 162]		
6	Total phenols	real	[0, 4]		
7	Flavanoids	real	[0, 6]		
8	Nonflavanoid phenols	real	[0, 1]		
9	Proanthocyanins	real	[0, 4]		
10	Color intensity	real	[1, 13]		
11	Hue	real	[0, 2]		

12	OD280/OD315 of diluted wines	real	[1, 4]	
13	Proline	integer	[278, 1686]	

4. **Glass** (Year: 1987)

Concept: This data determines whether the glass was a type of "float" glass or not,

Minority class: Class 5 (4.21%)

Table C.4 Glass

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	Id	Integer	[1, 214]	Class 1	70 (32.71%)
2	RI	REAL	[1.5112, 1.5339]	Class 2	76 (35.51%)
3	Na	REAL	[10.73, 17.38]	Class 3	17 (7.94%)
4	Mg	REAL	[0, 4.49]	Class 4	13 (6.07%)
5	Al	REAL	[0.29, 3.5]	Class 5	9 (4.21%)
6	Si	REAL	[69.81, 75.41]	Class 6	29 (13.55%)
7	K:	REAL	[0, 6.21]	*Total*	214 (100%)
8	Ca	REAL	[5.43, 16.19]		
9	Ba	REAL	[0, 3.15]		
10	Fe	REAL	[0, 0.51]		

5. **Vowel Recognition** (Year 1989)

Concept: Speaker independent recognition of the eleven steady state vowels of British English using a specified training set.

Minority class: Class 0 (9.09%)

Table C.5 Vowel Recognition

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	Feature 1	REAL	[-6, 0]	Class 0	90 (9.09%)
2	Feature 2	REAL	[-2, 6]	Class 1	90 (9.09%)
3	Feature 3	REAL	[-3, 2]	Class 2	90 (9.09%)
4	Feature 4	REAL	[-2, 3]	Class 3	90 (9.09%)

5	Feature 5	REAL	[-3, 2]	Class 4	90 (9.09%)
6	Feature 6	REAL	[-1, 3]	Class 5	90 (9.09%)
7	Feature 7	REAL	[-2, 2]	Class 6	90 (9.09%)
8	Feature 8	REAL	[-2, 3]	Class 7	90 (9.09%)
9	Feature 9	REAL	[-2, 2]	Class 8	90 (9.09%)
10	Feature 10	REAL	[-2, 2]	Class 9	90 (9.09%)
11				Class 10	90 (9.09%)
12				*Total*	990 (100%)

6. **Hypothyroid** (Year _)

Concept: This is a medical diagnosis dataset to indicate hypothyroid patient based on input attributes

Minority class: Class 1 (2.45%)

Table C.6 Hypothyroid

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	Feature 1	Real	[0, 1]	Class 0	3697 (97.55%)
2	Feature 2-16	Binary	{0, 1}	Class 1	93 (2.45%)
3	Feature 17	Real	[0, 1]	*Total*	3790 (100%)
4	Feature 18	Real	[0, 1]		
5	Feature 19	Real	[0, 1]		
6	Feature 20	Real	[0, 1]		
7	Feature 21	Real	[0, 1]		

7. **Abalone** (Year 1995)

Concept: Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task.

Minority class: Class 19 (0.77%)

Table C.7 Abalone

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)

1	Sex	Category	{Male, Female, Infant}	Class 1, 2, 25, 26, 29	1 (0.02%) (each class has 1 instance)
2	Length	Real	[0,1]	Class 24, 27	2 (0.04%)
3	Diameter	Real	[0,1]	3	15 (0.36%)
4	Height	Real	[0,2]	4	57 (1.36%)
5	Whole weight	Real	[0,3]	5	115 (2.75%)
6	Shucked weight	Real	[0,2]	6	259 (6.20%)
7	Viscera weight	Real	[0,1]	7	391 (9.36%)
8	Shell weight	Real	[0,2]	8	568 (13.60%)
9	Rings	integer	[0,1]	9	689 (16.50%)

Class	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Dist.	634	487	267	203	126	103	67	58	42	32	26	14	6	9
(%)	15.18	11.66	6.39	4.86	3.02	2.47	1.60	1.39	1.01	0.77	0.62	0.34	0.14	0.22

Total: 4177 (100%)

8. **E-coli** (Year)

Concept: To determine the site of a protein within a cell by its amino acid sequence. There are 8 classes: inner membrane lipoproteins (imL), outer membrane lipoproteins (omL), inner membrane proteins with a cleavable signal sequence (imS), other outer membrane-proteins (om), periplasmic proteins (pp), inner membrane proteins with an uncleavable signal sequence (imU), inner membrane proteins without a signal sequence (im), and cytoplasmic proteins (cp).

Minority Class: Class imS (0.60%)

Table C.8 E-coli

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	mcg	Real	[0, 1]	cp	143 (42.56%)
2	gvh	Real	[0, 1]	im	77 (22.92%)
3	lip	Real	[0, 1]	pp	52 (15.48%)
4	chg	Real	[0, 1]	imU	35 (10.42%)
5	aac	Real	[0, 1]	om	20 (5.95%)
6	alm1	Real	[0, 1]	omL	5 (1.49%)
7	alm2	Real	[0, 1]	imL	2 (0.60%)

8		imS	2 (0.60%)
9		Total	336 (100%)

9. **Lymphography** (Year 1988)

Concept: This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature.

Minority Class: Class fibrosis (2.70%)

Table C.9 Lymphography

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	lymphatics	Category	{1, 2, 3, 4}	normal find	2 (1.35%)
2	block of affere	Category	yes, no	metastases	81 (54.73%)
3	bl. of lymph. c	Category	yes, no	malign lymph	61 (41.22%)
4	bl. of lymph. s	Category	yes, no	fibrosis	4 (2.70%)
5	by pass	Category	yes, no	*Total*	148 (100%)
6	extravasates	Category	yes, no		
7	regeneration	Category	yes, no		
8	early uptake in	Category	yes, no		
9	lym.nodes dimin	Category	{0, 1, 2, 3}		
10	lym.nodes enlar	Category	{1, 2, 3, 4}		
11	changes in lym.	Category	{1, 2, 3}		
12	defect in node	Category	{1, 2, 3, 4}		
13	changes in node	Category	{1, 2, 3, 4}		
14	changes in stru	Integer	[1, 8]		
15	special forms	Category	{1, 2, 3}		
16	dislocation of	Category	yes, no		
17	exclusion of no	Category	yes, no		
18	no. of nodes in	Integer	[1, 8]		

10. **Opt-digits** (Year 1998)

Concept: The data is to extract normalized bitmaps of handwritten digits from a preprinted form. From a total of 43 people, 30 contributed to the training set and different 13 to the test set. 32x32 bitmaps are divided into non-overlapping blocks of 4x4 and the number of on pixels is counted in each block. This generates an input matrix

of 8x8 where each element is an integer in the range 0...16. This reduces dimensionality and gives invariance to small distortions.

Minority Class: Class 0 (9.86%)

Table C.10 Opt-digits

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	All 64 attributes	Integer	[0, 16]	Class 0	554 (9.86%)
2				Class 1	571 (10.16%)
3				Class 2	557 (9.91%)
4				Class 3	572 (10.18%)
5				Class 4	568 (10.11%)
6				Class 5	558 (9.93%)
7				Class 6	558 (9.93%)
8				Class 7	556 (10.07%)
9				Class 8	554 (9.86%)
10				Class 9	562 (10.00%)
11				*Total*	5610 (100%)

11. **Satellite** (Year 1993)

Concept: The database consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood. The aim is to predict this classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number.

There are 6 decision classes: 1, 2, 3, 4, 5 and 7.

Minority Class: Class 4 (9.73%)

Table C.11 Satellite

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	All 36 attributes	Integer	[0, 255]	Class 1	1533 (23.82%)
2				Class 2	703 (10.92%)
3				Class 3	1358 (21.10%)
4				Class 4	626 (9.73%)

5		Class 5	707 (10.99%)
6		Class 7	1508 (23.43%)
7		*Total*	6435 (100%)

12. **Letter Recognition** (Year 1991)

Concept: The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli.

Minority Class: Class H (3.67%)

Table C.12 Letter Recognition

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	x-box	Integer	[1, 15]	A	789 (3.95%)
2	y-box	Integer	[1, 15]	B	766 (3.83%)
3	width	Integer	[1, 15]	C	736 (3.68%)
4	high	Integer	[1, 15]	D	805 (4.03%)
5	onpix	Integer	[1, 15]	E	768 (3.84%)
6	x-bar	Integer	[1, 15]	F	775 (3.88%)
7	y-bar	Integer	[1, 15]	G	773 (3.87%)
8	x2bar	Integer	[1, 15]	H, Z	734 (3.67%)
9	y2bar	Integer	[1, 15]	I	755 (3.78%)
10	xybar	Integer	[1, 15]	J	747 (3.74%)
11	x2ybr	Integer	[1, 15]	K	739 (3.70%)
12	xy2br	Integer	[1, 15]	L	761 (3.81%)
13	x-ege	Integer	[1, 15]	M, Q	792 (3.96%)
14	xegvy	Integer	[1, 15]	N	783 (3.92%)
15	y-ege	Integer	[1, 15]	O	753 (3.77%)
16	yegvx	Integer	[1, 15]	P	803 (4.02%)
17				R	758 (3.79%)
18				S	748 (3.74%)
19				T	796 (3.98%)
20				U	813 (4.07%)

21		V	764 (3.82%)
22		W	752 (3.76%)
23		X	787 (3.94%)
24		Y	786 (3.93%)
25		*Total*	20009 (100%)

13. **Pen Recognition** (Year 1998)

Concept: This data is a digit database by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training, cross-validation and writer dependent testing, and the digits written by the other 14 are used for writer independent testing

Minority Class: Class 3 (9.60%)

Table C.13 Pen Recognition

No.	Attribute			Class	
	Name	Type	Range/Values	Name	Distribution (%)
1	All 16 attributes	Integer	[0, 100]	Class 0, 1	1143 (10.40%)
2				Class 2, 4	1144 (10.41%)
3				Class 3, 5, 8, 9	1055 (9.60%)
4				Class 6	1056 (9.61%)
5				Class 7	1142 (10.39%)
				Total	8705 (100%)

Appendix D

Author Biography

1	Personal Information						
1.1	Family Name	NGUYEN HUY					
	Given Name(s)	THACH					
1.2	Nationality	Vietnamese					
1.3	Date of Birth	Date	19	Month	10	Year	1983
1.4	Gender	Male	√	Female			

2	Academic Qualifications (I)				
2.1	Bachelors' Degree	Bachelor of Software Engineering			
2.2	Program/Faculty	Faculty of Information Technology			
2.3	University	Hanoi University of Technology, Hanoi, Vietnam			
2.4	Year of Start	Month	September	Year	2001
2.5	Year of Completion	Month	June	Year	2006
2.6	Grade Point Average	8.15			(10-scale grading system)

3	Academic Qualifications (II)				
3.1	Master's Degree	Master of Computer Engineering			
3.2	Program/Faculty	Department of Computer Engineering, Faculty of Engineering			
3.3	University	King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand.			
3.4	Year of Start	Month	November	Year	2006
3.5	Year of Completion	Month	October	Year	2008
3.6	Grade Point Average	4.0			(4.0-scale grading system)

4	Employment Record

- July 2006 – October 2006, worked at Silver Software Development Investment Joint Stock Company, No. 17, 151A/1 Thai Ha Street, Dong Da District, Hanoi, Vietnam with Leader Mr. Dinh Cong Thanh.
- Worked as an internship at the end of my bachelor study for six months in Library and Information Network Centre (LINC) (January – June 2006). Director: Assoc. Prof. Dr. Nguyen Ngoc Binh.
- Participated in training newcomers for some projects.

5	Publications

1) Ouen Pinngern and Thach Huy Nguyen, "Application of XCS to Cart-pole Problem", *The 10th International Symposium on Electrical – Electronics Engineering 2007 - ISEE 2007*, Automatic Control Section, pp. 30-35, Ho Chi Minh city, Vietnam, October 23-25, 2007.

2) Thach Huy Nguyen, Sombut Foitong, Sornchai Udomthanapong and Ouen Pinngern, "Effects of Distance between Classes and Training Datasets Size to the Performance of XCS: Case of Imbalance Datasets", *IAENG International Conference on Artificial Intelligence and Applications*, vol. I, pp. 468-473, Hong Kong, 19-21 March, 2008.

3) Thach Huy Nguyen, Sombut Foitong and Ouen Pinngern, "Rough Set and XCS in Classification Problems", *International Conference on Computer and Communication Engineering – ICCCE 08*, vol. II, pp. 806-811, Malaysia, 13-15 May 2008.

4) Thach Huy Nguyen, Porntep Rojanavasu and Ouen Pinngern, "Cost-sensitive XCS Classifier System Addressing Imbalance Problems", *The 5th International Conference on Fuzzy Systems and Knowledge Discovery – FSKD'08*, China, 18-20 October, 2008. (accepted for publication)

5) Thach Huy Nguyen, Sombut Foitong, Phaitoon Srinil and Ouen Pinngern, "Towards Adapting XCS for Imbalance Problems", *Tenth Pacific Rim International Conference on Artificial Intelligence - PRICAI 2008 - LNAI 5351*, pp. 1028–1033, Vietnam, 15-19 December 2008. (accepted for publication)

THE 10th CONFERENCE ON SCIENCE AND TECHNOLOGY

BK
TP.HCM

THE INTERNATIONAL SYMPOSIUM ON ELECTRICAL – ELECTRONICS ENGINEERING

ISEE 2007

TRACK 3

POWER AND SYSTEMS ENGINEERING

Organized by

Ho Chi Minh City University of Technology

Faculty of Electrical and Electronics Engineering

October 24-25, 2007

HOCHIMINH CITY, VIETNAM

APPLICATION OF XCS TO CART-POLE PROBLEM

O. Pinngern[1] and T.H. Nguyen[2]

Department of Computer Engineering, Faculty of Engineering
Information Science Laboratory
Research Center for Communication and Information Technology (ReCCIT)
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520 THAILAND

[1]kpouen@kmitl.ac.th, [2]nguyenhuythach1983@yahoo.com

ABSTRACT

The problems of centering a cart and balancing a pole by applying force are well-known as inverted pendulum problem in control theory. It is well referenced and has successfully addressed by control methods including genetic algorithms, artificial neural networks, and reinforcement learning. In our research, we propose a novel approach by using XCS to solve this inverted pendulum problem. The objective is to analyze the behavior of the XCS classifier system on balancing the pole and centering the cart. We show that based on the generalization property, XCS significantly reduces number of rules of the cart-pole problem without any effect to its overall performance. Initial test was conducted to balance the pole while subsequent test included centering the cart. Preliminary results have motivated a promising alternative.

1. INTRODUCTION

A learning classifier system (LCS) is an adaptive system that learns to achieve a task through interacting with environment. Introduced in 1995 by Wilson [1], XCS is one of the most successful classifier systems. The main advantage of XCS is that its generalization property over the state-action space creates a full map of the problem space XxA=>P from inputs and actions to payoff predictions [13][14]. Other attractive feature is that XCS executes the niche genetic algorithm in combination with accuracy-based fitness. In addition, XCS is capable of avoiding problematic *overgeneral* rules which receive a high optimal payoff for some inputs but are sub-optimal for other, lower payoff, inputs.

In these ways, XCS has been shown to be extremely effective in a number of domains [2]. An important application of LCS commonly and XCS specifically is autonomous robotics where classifier systems have been applied most during the last ten years and gotten many successes. XCS also gets useful results in other fields such as data mining and data analysis.

The inverted pendulum (or cart-pole system) is a benchmark for trainable controllers [4]. It is a typical non-linear dynamic control system. Learning to balance the cart-pole was the subject of one of the simplest experiments and easiest to understand conducted in machine learning. The inverted pendulum is a textbook example of an inherently unstable control system [4]. It is well-referenced and has been already successfully addressed by many control methods. The earliest work on learning for this problem was carried out by Michie and Chambers (1968) [1]. In the training time, they discretized the state space into non-overlapping region and applied two opposite constant forces to train cart-pole balancing. In other approach, two neuron-like elements were used to solve this problem by Barto et al. [3]. Recently, Chiang et al. [9] and Liu et al. [10] proposed to use genetic reinforcement fuzzy logic controller to balancing cart-pole via digital simulation.

After survey existed cart-pole controllers, there are some controllers that avoid failure but

do not center the cart while others would do a very good job of balancing and centering if they were allowed to run in a slightly longer track. In this paper, the proposed schemes make the pole upright and regulate the cart to the specific position.

This paper is organized as follows. In section 2, we will briefly review the XCS classifier system, classifiers, performance and properties of XCS. In section 3, the cart-pole system will be described with all its equations and parameters. The details of implementations of XCS in cart-pole problem will be presented in section 4. In section 5, the simulation results and comparison with other methods will be presented to verify the feasibility of XCS in inverted pendulum problem. Section 6 concludes this work and directs the future works.

2. XCS CLASSIFIER SYSTEM

Like other classifier system, XCS seeks a gain reinforcement from its environment based on an evolving set of condition-action rules called classifiers. Via genetic algorithm process, classifiers useful in gaining reinforcement are selected and propagate over those less useful leading to increasing system performance [3]. Classifiers, performance and properties of XCS are explained as follows.

2.1. Classifier representation

As same as classifier representation in traditional LCS, classifier representation in XCS takes the form of ternary strings. The system should evolve a classifier <condition> : <action> => <prediction> or ({x}, a) => p, where the condition, {x}, is a string from {1, 0, #}, a condition can match more than one state if it contains #'s, the action a $\in \{a_1, a_2 \ldots a_n\}$ specifies the action that the classifier proposes, the prediction p estimates the payoff expected if the classifier matches and its action is taken by the system. There are other parameters of classifier as prediction error ε , fitness F, *numerosity* n. More details can be found in [12].

Classifiers in XCS are *macroclassifiers*. XCS used *macroclassifiers* concept in order to eliminate the redundancy as classifier system populations contain many classifiers having the same conditions and actions [11]. In a population of *macroclassifiers*, each newly generated classifier is examined to see if a structurally identical classifier already exits. If so, the existing classifier's *numerosity* parameter is increased by one and the new classifier is abandoned. If not, the new classifier is added to the population with its *numerosity* initialized at one. The *macroclassifier* idea reduces processing time and saves storage. Classifiers with high *numerosity* tend to be those with maximal accurate generalization, so the most significant knowledge can be determined readily by sorting the population by *numerosity*.

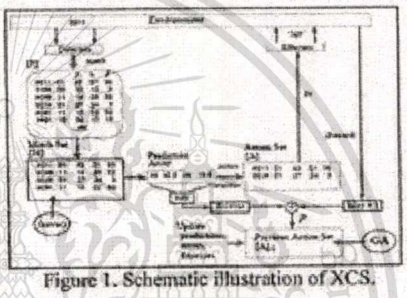


Figure 1. Schematic illustration of XCS.

2.2. Performance of XCS

Figure 1 illustrates a broad picture of XCS [11]. Given an input, it is matched with classifiers of population [P] in the condition part to form matched set [M]. From the match set, the prediction value $P(a_i)$ for each action a_i appearing in [M] is calculated. The $P(a_i)$ values are placed in a prediction array and an action is selected by some methods as: selecting action with the largest prediction is called deterministic action selection or exploit method, roulette-wheel action selection or selecting completely random. Once an action is selected, the action set [A] is formed. The action in [A] will be sent to environment by the effectors and a reward r be returned by the environment. The Q-learning method of Reinforcement learning [16] and MAM technique use the reward r to update values of classifiers in the previous action set $[A]_{-1}$.

2.3. Properties of XCS

XCS has three main properties: Fitness based on the accuracy, using niche genetic algorithm, subsumption and MAM technique.

2.3.1. Accuracy-based approach

In XCS, rule fitness for the genetic algorithm is not based on rule prediction (or strength) but on the accuracy of these predictions. It forms a complete and accurate mapping of the search space rather than to simply focus on the higher payoff niches in the environment. As noted in the Introduction, XCS is capable of avoiding problematic *overgeneral* rules, since the payoffs received by *overgenral* rules typically have high variance (they are in-accurate predictors) and so have low fitness in XCS. [11]

2.3.2. Niche genetic algorithm (GA)

XCS restricts GA activity to within niches defined by the previous action set $[A]_{-1}$ (or [A] in a single-step problem). XCS has been shown to evolve rules that are maximally general, subject to an accuracy criterion [13]. In addition, crossovers within a niche are more likely to yield useful classifiers than crossovers between potentially unrelated classifiers that match in different niches.

2.3.3. Subsumption and MAM technique

As mentioned in [13], there are two subsumption procedures. The first one is "GA subsumption" applied when an offspring is created to check if its condition is logically subsumed by the condition of its parent. If so, the offspring is not added to population, but the parent's numerosity is increased (macroclassifires). The second one is "action set subsumption" taking place in every action set. In this procedure, the action set is searched for the most general classifier. Then all classifiers in the set are subsumed of the most general one are eliminated from the population.

Besides, MAM (Moyenne Adaptive Modifee) technique is used to adjust the classifier's fitness F_j. If the finess has been adjusted $1/\beta$ times, $F_j = F_j + \beta(K_j' - F_j)$. Otherwise, F_j is set to the average of the current and previous values of K_j'. This method results in a stronger robustness against inaccurate classifiers.

3. CART-POLE PROBLEM WITH XCS

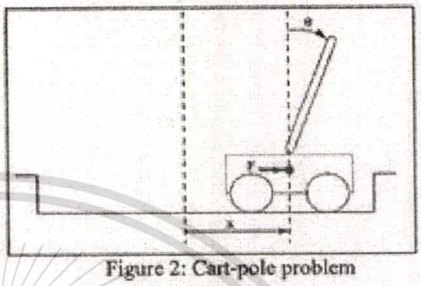


Figure 2: Cart-pole problem

The cart-pole balance system is really a well-known nonlinear system [4]. Fig. 2 shows the cart-pole balance system which includes a pole hinged to a motor-driven cart and the cart is moved back and forth on a track of limited length. The goal of the controller task is to apply the forces of fixed magnitude to the cart such that the pole is balanced and the cart does not hit the edge of the track.

The system gets the failure signal when either the pole falling exceed $\pm 12^0$ or the cart hitting the bounds of the track ± 2.4 m. At each time, the system state is specified by 4 real-valued variables:

x – the position of the cart

$\dot{x}$ – the velocity of the cart

θ – the angle of the pole with respect to the vertical line

$\dot{\theta}$ – the angular velocity of the pole

And F – the force lies in [-10, 10] newtons, applied to the cart. The dynamics of the cart-pole system are modeled by the following nonlinear differential equations [7]:

$$\theta(t+1) = \theta(t) + \Delta\dot{\theta}(t) \quad (1)$$

$$\dot{\theta}(t+1) = \dot{\theta}(t) + \Delta\ddot{\theta}(t) \quad (2)$$

$$\ddot{\theta}(t) = \frac{g\sin\theta(t) + \cos\theta(t)\left(\frac{-F - (m_c + m_p)l\dot{\theta}^2(t)\sin\theta(t)}{m_c + m_p}\right)}{l\left(\frac{4}{3} - \frac{m_p\cos^2\theta(t)}{m_c + m_p}\right)} \quad (3)$$

$$x(t+1) = x(t) + \Delta\dot{x}(t) \quad (4)$$

$$\ddot{x}(t+1) = \ddot{x}(t) + \Delta \frac{F + m_p l(\dot{\theta}^2(t)\sin\theta(t) - \ddot{\theta}(t)\cos\theta(t))}{m_c + m_p}$$

(5)

Where F is the amount of force (in Newton) applied to the cart, l is the length of the pole, m_c is the combined mass of the pole and the cart, m_p is the mass of the pole, g is the acceleration due to gravity, and Δ or Δt is the sampling interval time.

4. IMPLEMENTATION DETAILS

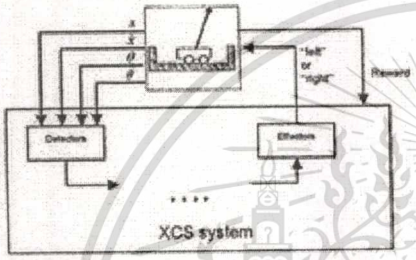


Figure 3: XCS in cart-pole problem

The fig. 3 illustrates the XCS as applied to cart-pole problem. Next, we consider the rule representation of the XCS in cart-pole system.

4.1. Conditions

The XCS is supposed to have 4 input units to detectors. The function of detectors is discretizing input continuous values to binary values. As the first time, Michie and Chambers proposed their BOXES algorithm, we divided distance x, angle θ, velocity $\dot{x}$ and angle velocity $\dot{\theta}$ to 3:6:3:3 parts as following:

$x \in [-2.4, -0.8)U[-0.8, +0.8]U(+0.8, +2.4]$ (m)

$\theta \in [-12,-6) \cup [-6,-1) \cup [-1,0) \cup [0,1) \cup [1,6) \cup [6,12]$ (degree)

$\dot{x} \in [-\infty, -0.5)U[-0.5, +0.5]U(+0.5, +\infty]$ (m/s)

$\dot{\theta} \in [-\infty, -50)U[-50, +50]U(+50, +\infty]$ (degree/s)

We encoded $x, \dot{x}, \dot{\theta}$ by 2 bits: 00, 01, 10 and do not use 11; θ by 3 bits: 000, 001, 010, 011, 100 101 and do not use: 110 or 111. So, we have totally 3x6x3x3 = 162 states or 162 cases of conditions. The condition part of each classifier is encoded by $<x, \theta, \dot{x}, \dot{\theta}>$ in ternary alphabet {1, 0, #}. Due to crossover and mutation properties of XCS, we do not need worry about cases $x, \dot{x}, \dot{\theta}$ are 11 or θ is 110 or 111 (The more details can be found in [12]).

In other approach, 2 bits of x, $\dot{x}, \dot{\theta}$ and 3 bits of θ are used fully, we have 4x8x4x4 = 512 states, the results of simulation in this case are better than that of 162 states a little bit, this view is analyzed carefully in 5.2.

4.2. Actions

The classifiers of XCS in cart-pole problem take two actions: 0 and 1 which are dimensions of the force F applied to the cart, 0:left, 1:right. Each classifier has only one action, the XCS software uses only the mutation operator for actions and it randomly generates a new action from the remained action.

5. EXPRIMENTS AND RESULTS

The simulated program used the standard values for the parameters of cart-pole in table 1 [6] and parameters of XCS in table 2.

After the training time, the best rules based on *numerosity* are selected to be used in the testing time. By experiment, the enough number is 13-15. In the following sections, the simulation results and comparison with other methods will be presented.

Table 1: Standard cart-pole values

Parameter	Value
Track Length	± 2.4 m
Failure Angles	± 12°
Gravity (g)	9.8 m/s²
Length of Pole (2l)	1 m
Mass of cart (m_c)	1.0 kg
Mass of pole (m_p)	0.1 kg
Magnitude of Control Force (F)	10 N
Integration time step (Δt)	0.02 s

Table 2: Parameters of XCS

Parameter	Value
Maximum size of population (N)	162
Crossover probability in GA (χ)	1.0
Mutation probability in GA (μ)	0.01
Learning rate (β)	0.2
Discount factor (γ)	0.95
GA threshold (θ_{GA})	25
Covering probability of using # when covering ($P_\#$)	0.33

5.1. With standard parameters

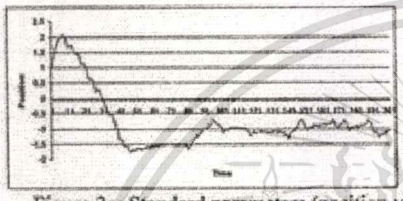


Figure 3a: Standard parameters (position v time)

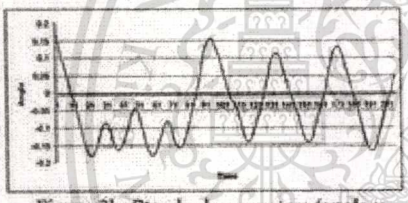


Figure 3b: Standard parameters (angle v time)

The fig. 3a, 3b present the operation of system with standard parameters. Balancing the pole needs 11 steps and centering the cart needs 31 steps. These results show that XCS has high potential to solve the inverted pendulum problems.

5.2. Adapting parameters

To explore capabilities of XCS in controller systems, we adapted some parameters of the system such as: condition of classifier and the way to calculate integration time step.

In adapting condition of classifier, we used full 9-bit condition, so the state space of problem is smoother with 2^9=512 states instead of 162 states. The results in fig. 4a, 4b are not much different from those in fig. 3a, 3b. We can see that the time to center the cart is shorter and angle mutation is less oscillation a little bit but the training time is longer. So, 162 states are enough.

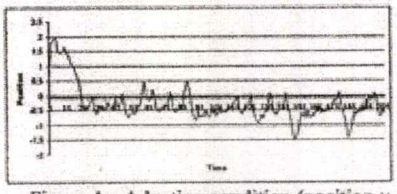


Figure 4a: Adapting condition (position v time)

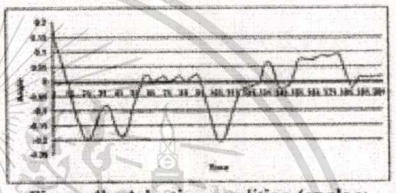


Figure 4b: Adapting condition (angle v time)

When the integration time step Δt is changed, the time to balance pole is changed as fig. 6. With $\Delta t \leq 0.013$s or $\Delta t \geq 0.027$s, the system is fail and cannot learn. So, the best Δt that should be chosen is $0.014 \leq \Delta t \leq 0.027$ (s).

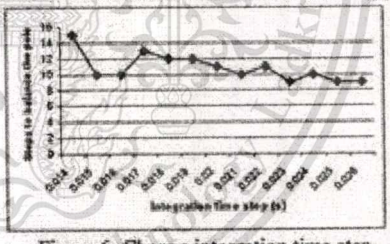


Figure 6: Change integration time step

5.3. Comparison

From information in [6], we have a comparison result: XCS-cartpole system performances better than BOXES controller but worse than genetic algorithm controller in terms of balancing pole time and centering cart position. The large oscillation exhibited by XCS-cartpole controller was also observed by BOXES controller.

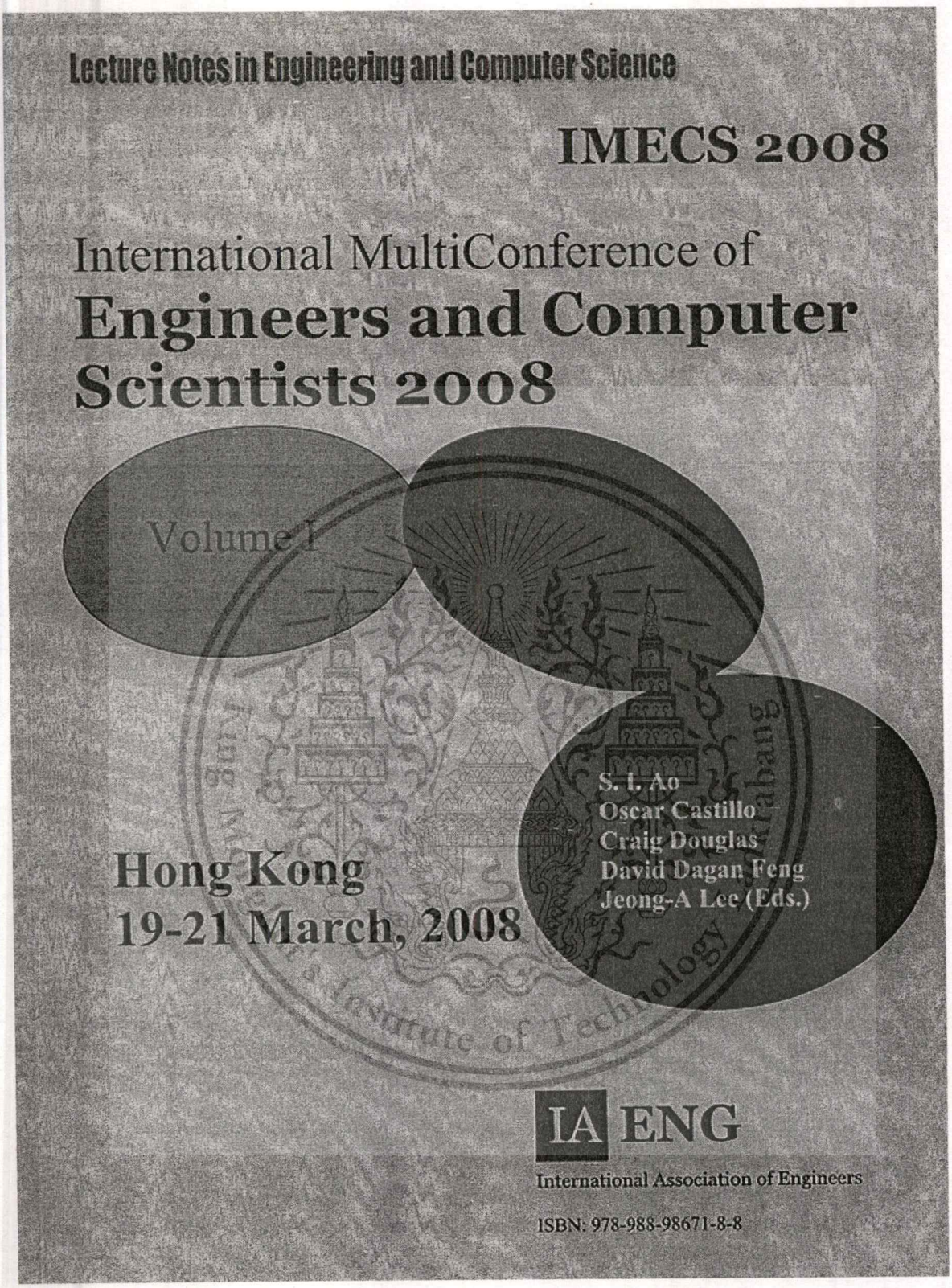
6. CONCLUSION AND FUTURE DIRECTIONS

This paper explored the intelligent capability of XCS in the invert pendulum problem that the number of rules in this rule-base system was reduced significantly. All the simulation results reveal that the XCS is indeed effectively in the cart-pole balancing system. XCS presented a property of a self-learning controller: Generated the control rules automatically without expert knowledge because of using a niche genetic algorithm, subsumption technique and accuracy-based fitness.

The approach described may be viewed as a step in the development of better understanding of how to improve the operation of controllers by XCS classifier system. Furthermore, due to advantages of the mapping from continuous inputs to outputs of fuzzy systems combined with the self-learning property of classifier systems, our future research will dealing with problems using Fuzzy-XCS which is introduced in [15].

REFERENCES

1. S. Russell and P. Norvig, Artificial Intelligence: A modern approach, 2nd ed. Prentice Hall (2003), pp.763-789.
2. P.L. Lanzi, W. Stolzmann, S.W. Wilson, Springer Berlin/Heidelberg, Learning Classifier Systems: From Foundations to Applications (2000)
3. A. Barto, R. Sutton, and C. Anderson, Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Trans. on Sys.*, Vol.13, No.5, (1983), pp.834-846.
4. Geva, S. and Sitte, J. A cart-pole experiment benchmark for trainable controllers, *IEEE Control Systems Magazine*, Vol.13, No.5, (1993), pp.40-51.
5. O. Michel, M. Clergue and Ph. Collard, Artificial neurogenesis: Applications to the cart-pole problem and to an autonomous mobile robot, *International Journal on Artificial Intelligence Tools* Vol.6, No.4, (1997), pp.613-634.
6. M. Randall, C. Thorne, C. Wild, A standard comparison of adaptive controllers to solve the cart pole problem, *Proceedings of the Second IEEE Australian and New Zealand Conference on Intelligent Information Systems*, (1994), pp.61-65.
7. R.V. Florian, Correct equations for the dynamics of the cart-pole system, Center for Cognitive and Neural Studies (Coneural), Romania.
8. A. P. Wieland. Evolving neural network controllers for unstable systems, *IEEE International Joint Conference on Neural Networks*, Vol.2, (1991), pp.667-673.
9. C.K. Chiang, H.Y. Chung, and J.J. Lin, A Self-Learning Fuzzy Logic Controller Using Genetic Algorithms with Reinforcements, *IEEE Trans. on fuzzy sys.*, Vol.5, No.3, (1997), pp.460-467.
10. B.D. Liu, C.Y. Chen, and J.Y. Tsao, Design of Adaptive Fuzzy Logic Controller Based on Linguistic-Hedge Concepts and Genetic Algorithms, *IEEE Trans. on sys.*, Part B: cybernetics, Vol.31, No.1, (2001), pp.32-53.
11. Stewart W. Wilson, Classifier Fitness Based on Accuracy, *Evolutionary Computation*, Vol.3(2), (1995), pp.149-175.
12. Butz, M. and Wilson, S., An algorithmic description of XCS. *Journal of Soft Computing 6*, (2002), pp.144-153.
13. Stewart W. Wilson, Generalization in the XCS Classifier System, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. Koza et al. (Eds.) San Francisco, CA: Morgan Kaufmann, (1998), pp.665-674.
14. Pier Luca Lanzi. A Study on the Generalization Capabilities of XCS. *Evolutionary Computation*, 7(2), (1999), pp.125-149.
15. J. Casillas, B. Carse, and L. Bull, Fuzzy-XCS: a Michigan Genetic Fuzzy System, *IEEE Trans. on fuzzy sys.*, Vol.XX, No.Y, (2007)
16. Sutton, R. & Barto, R., Reinforcement Learning. MIT Press, (1998).



Effects of Distance between Classes and Training Datasets Size to the Performance of XCS: Case of Imbalance Datasets

Thach H. Nguyen, Sombut Foitong, Somchai Udomthanapong and Ouen Pinngern

***Abstract*—** This paper analyzes the effects of distance between classes and training datasets size to XCS classifier system on imbalanced datasets. Our purpose is to answer the question whether the loss of performance incurred by the classifier faced with class imbalance problems stems from the class imbalance per se or it can be explained in some other ways. The experiments from 250 artificial imbalanced datasets show that XCS can perform well in some imbalance domains if the training datasets size is large enough and the distance between classes is appropriate. Thus, it dose not seem fair to correlate imbalance datasets directly to the loss performance of XCS. Through this research, we also know what kinds of datasets are suitable for training XCS and dealing with class imbalances alone will not always help improve performance of classifiers.

Keywords - Data mining, XCS, imbalance dataset, distance between classes.

I. Introduction

XCS is an accuracy-based learning classifier system (LCS) that is shown to perform very competitively with respect to other machine learning methods in classification problems. Different from other learning classifier systems executing genetic algorithm panmictically, XCS executes a genetic algorithm in niches defined by the action sets [3] that produces a more complete model of the problem space.

XCS has already attracted much attention from machine learning and data mining communities [1], [5]. There are some performance comparisons of strength-based fitness systems and accuracy-based fitness systems showing that accuracy-based fitness systems, including XCS, have a significant impact on data mining field. In [6], Mansilla and Garrel-Guiu studied the performance in a set of well-known machine learning problems demonstrating that XCS is able to produce a classification performance and rule set which exceeds the performance of most current Machine Learning techniques.

Recently, the class imbalance problem has been recognized as a crucial problem in machine learning and data mining. This problem has been reported to hinder the XCS's performance on many types of problems [6][7][8]. However, the main problem with these proposals is that they ignored the effects of training datasets, distance between classes and no study has made a point of linking the class imbalance problem directly to this loss. As a matter of fact, although the performance of XCS may decrease on many imbalance domains, that does not prove that it is imbalance domains per se, that causes that decrease. Rather it is quite possible that class imbalances yield certain conditions that hamper classification, which would suggest that 1) class imbalances are not necessarily always a problem and, perhaps even more importantly, 2) dealing with class imbalances will not always help improve performance of classifiers. Our experiments show that the distance between classes affects extremely to accuracy of learning systems on high class imbalance levels. And in this research, we also analyze the effects of training dataset size and determine how it affects to classifier learning system, XCS.

The purpose of this paper is to answer the question whether class imbalances are truly to blame for the reported losses of XCS performance or whether these deficiencies can be explained in some other ways. We show that class imbalances are, actually, not a problem by themselves, but that, in small and complex datasets, they come accompanied with the problem of distance between classes which in turn causes degradation in XCS's performance.

The remainder of this paper is organized as follows. The application of data mining techniques to class imbalance problems are discussed in section 2 which is followed by a brief description of XCS. From section 4, dataset generation with all parameters: imbalance rate – *ir*, distance between classes – *dis* and training dataset size – *s* will be described. In section 5, the results and experiments of XCS in different dataset domains will be presented. Section 6 concludes this work and directs the future works.

II. Related Work

To solve class imbalance problems, many methods and

Manuscript received October 15, 2007. This work was sponsored by the AUN/SEED-Net organization and the Information Science Laboratory, Research Center for Communications and Information Technology: ReCCIT, King Mongkut's Institute of Technology Ladkrabang, Thailand.

T. H. Nguyen, is a master student in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand (e-mail: nguyenhuythach1983@yahoo.com).

S. Foitong is a doctoral student in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand (e-mail: sombutfi@yahoo.com).

S. Udomthanapong is a master student in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand (e-mail: u.somchai@gmail.com).

O. Pinngern is an Associate Professor in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Thailand (e-mail: kpouen@kmitl.ac.th).

algorithms have been proposed. Some of the best well-known approaches are applied at the sampling level. In [10], Chawla et al. presented SMOTE which can be done either over-sampling minority class or under-sampling the majority class. Both methods can be applied in any concept learning system, since they act as a preprocessing phase, allowing the learning system to receive the training instances as if they belonged to a well-balance dataset. Thus, any bias of the system towards the majority class due to the different proportion of examples per class would be expected to be suppressed. In addition, in [12], the effects of sampling method, probabilistic estimate and decision tree structure of C4.5 on imbalance datasets were investigated. They also proposed an average improvement of the Area under the ROC curve (AUC) measure over the other sampling schemes. Beside that, G.M. Weiss and F. Provost [16] analysis the effects of imbalance datasets to classifier learning systems and how they affect through the evaluation of learned classifiers by using two performance measures: AUC and classification accuracy. In [13], a new approach in cost-sensitive to neural networks is proposed instead of decision trees to solve multiclass tasks. And in [17], G.M. Weiss indicated the learning from imbalance and rarity datasets can be handled in a similar manner. Recently, Chris S. et al. [9] evaluated the performance of seven commonly-used sampling techniques on imbalance and noise datasets to study the effects of noise to imbalance problem. The most notice researches are [14] and [15]. They proposed a new sight in imbalance problem by proposing to consider the effects of overlapping datasets and small disjunction problem. This approach is also focused in our research.

In learning classifier systems field, there are 3 different approaches: Fitness adaptation, population size and parameter settings. In [8], Albert O.P. and Ester B.M showed that XCS with standard parameter settings for 10 imbalance levels, the True Negative (TN or majority class) rate quickly reaches 100%, but the True Positive (TP or minority class) rate raises to 100% for imbalance levels up to i=4 (i is the imbalance level used to calculate the imbalance ratio ir=2^i). For i=5, XCS needs a long training time to reach 100% correct of TP rate and, for $i \geq 6$, the system classifies all input instances belonged to the majority. At that time, the population mainly consists of the two *overgeneral* rules: ######:0 and ######:1 (in case of 6-multiplexer).

To overcome this problem, tuning XCS's parameters based on the dataset imbalance ratio, is proposed by 2 methods: manual (offline) and automatic (online) adjustment. The experiments show that XCS can solve the unbalanced 11-bit multiplexer problem until i=8, which is a notable improvement with respect to the initial experiments.

Another attribution deals with the class imbalance problem by finding fitness adaption [7] based on class-sensitive accuracy in combination with UCS, a supervised LCS derived from XCS, as a useful tool for alleviating the effects of class imbalances. In fact, because XCS, UCS and some other learning classifier systems have their fitness based on accuracy, they present a high bias toward the majority class instances and evolve easily *overgeneral* classifiers. Idea of the proposal is restrict classifiers to cover regions formed by examples of a single class and make accuracy class-sensitive rather than instance-sensitive. Thus, accuracy is modified so that each class is considered equally important regardless of the number of instances representing in each class. The experiments show that this model evolved with imbalance level up to i=7.

III. XCS Overview

Like other classifier systems, XCS seeks a reinforcement reward from its environment based on an evolving set of condition-action rules called classifiers. Via genetic algorithm process, classifiers useful in gaining reinforcement are selected and propagate over those less useful leading to increasing system performance. Figure 1 illustrates a broad picture of XCS.

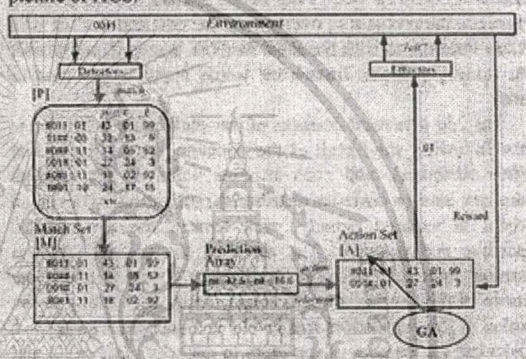


Figure 1: Schematic illustration of XCS in classification application.

In the training time, datum is read from the training data set and encoded to a classifier by detectors and it is matched with classifiers of population [P] in the condition part to form match set [M]. From the match set, the prediction value $P(a_i)$ for each action a_i appearing in [M] is calculated. The $P(a_i)$ values are placed in a prediction array and an action is selected by some methods as: selecting action with the largest prediction is called deterministic action selection or exploit method, roulette-wheel action selection or selecting completely random. Once an action is selected, the action set [A] is formed. The action in [A] will be sent to environment by the effectors and a reward r be returned by the environment. The Q-learning method of Reinforcement learning and MAM technique use the reward r to update values of classifiers in the action set [A]. Because classification problem is a single step problem, Genetic Algorithm (GA) is applied in [A] instead of $[A]_{-1}$, previous action set, as multi-step problems [3]. The more details are referred to [3] and a detailed algorithmic description can be found in [4].

IV. Dataset Generation

Because our purpose is to understand how class imbalances influence to the performance of XCS, we chose to run our experiments on a series of artificial domains whose characteristics could be controlled carefully rather than conducting our study on real-world domains whose results would be difficult to decipher.

The artificial datasets employed in the experiments have three major controlled parameters. The first one is distance between means of two classes in Gaussian distribution (*dis*), the second one is the training dataset size level (*s*) and the last one is the grade of imbalance (*i*). In particular, we created 250 domains belong to change of the three parameters by employing the method represented in [14]. In [14], Nathalie Japkowics designed a similar framework for testing the effect of the small disjunction to the imbalance dataset problem. In addition, in [15], R.C. Prati et al. studied the relationship between the problem of overlapping the data and dealing with class imbalances with the same method. It seemed reasonable to assume that this framework would apply to our case as well.

The 250 generated domains of our study were generated in the following way: each of the domains is two-dimensional (two attributes), and each attribute value is generated at random, using a Gaussian distribution, with standard deviation equal to value 1. Jointly, each domain has two classes: positive and negative. The mean of the positive class is created firstly and based on the distance between classes; the mean of the negative class is found. After that, belong to the value of training dataset size level *s* and imbalance level *i*, we calculate the number of instances in majority class (n_{maj}) and minority class (n_{min}). Actual training datasets are generated from this backbone model.

Ten differences of distance between classes were considered (*dis*=*0..9*). For the first domain, the means of the Gaussian function for both classes are similar. For the following domains, we stepwise add 1 standard deviation to the mean of the negative class, up to 9 standard deviations.

Five training dataset sizes were considered (*s*=*1..5*) where each size, *s*, corresponds to the number of instances in the majority class: $round((8192/2^5)*2^s)$. We will explain why we chose the number 8192 later.

Finally, five levels of class imbalance were also considered. Because, with standard parameter settings, XCS can solve well problems with imbalance level $i \leq 4$ as introduced in the previous section, in this paper, the imbalance level *i* is considered only with *i*=*5..9*. Each level *i* corresponds to the situation that minority class contains only $1/2^i$ of all instances in the majority class. So with given values of *s* and *i*, we can calculate the number of instances in majority and minority are $n_{maj}=round((8192/2^5)*2^s)$ and $n_{min}=round((8192/2^5)*2^s/2^i)$, respectively. For example, for *s*=*1* and *i*=*5*, the majority class is represented by $n_{maj}=round((8192/2^5)*2^1)=512$ instances and the minority class is represented by $n_{min}=round((8192/2^5)*2^1/2^5)=16$ instances; If *s*=*5*, *i*=*9* then $n_{maj}=8192$ and $n_{min}=16$.

Now, we explain why we chose the number 8192 in formulation $n_{maj}=round((8192/2^5)*2^s)$. In the initial time, we did not know that number, we called it is *x*, and we had to find it. By this framework, we have number of instances in minority class depending on *x*: $n_{min}=round((x/2^5)*2^1/2^i)$. To ensure XCS can learn the condition $n_{min} \geq 1$ must be approved. We see that, in the most extremely imbalance level case and the smallest training dataset size: *s*=*1*, *i*=*9*, $n_{min} = round((x/2^5)*2^1/2^9) = round((x/2^5)*2^1/2^9) = round(x*2^{-13}) \geq 1$, or $x \geq 8912$. In our research, we chose *x*=8912.

V. Experiments and Results

To answer the question whether the class imbalance problem always causes degradation in performance of XCS or it does so only in certain cases, we ran XCS [3], [4] on the artificial datasets described in the previous section and XCS's parameters settings as table 1. These values are described in [4]:

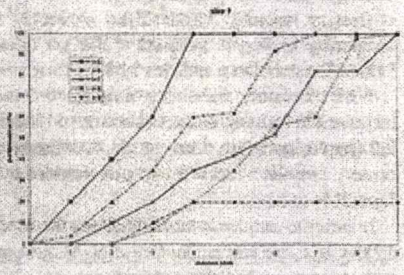
Table 1: XCS's parameters settings

Symbol	Parameter	Value
N	Maximum size of population	2000
χ	Crossover probability in GA	0.75
μ	Mutation probability in GA	0.01
β	Learning rate	0.2
γ	Discount factor	0.95
θ_{GA}	GA threshold	25
$P_\#$	Covering probability of using # when covering	0.66

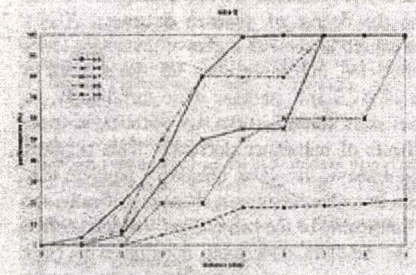
A. The effects of imbalance datasets to XCS

The results of our experiments are displayed in Figure 2 which plots the performance of XCS obtained for each combination of training dataset size, imbalance level and distance between classes. Each plot of this figure represents XCS performance obtained at a difference training set size. The top-leftmost plot corresponds to the smallest size (*s*=1) and progress until the below-center plot which corresponds to the largest size (*s*=5). Within each of these plots, each line connected by 10 points represents the concept of imbalance level from *i*=5 to *i*=9. Within each line, finally, each point corresponds to a particular distance between classes. The leftmost (or starting) point of each line corresponds to the nearest distance between two classes (*dis*=0) and progress until the rightmost point which corresponds to the farthest distance between classes (*dis*=9). The height of each point represents the average percent performance obtained by XCS on the training dataset size, imbalance level and distance between classes that this point represents.

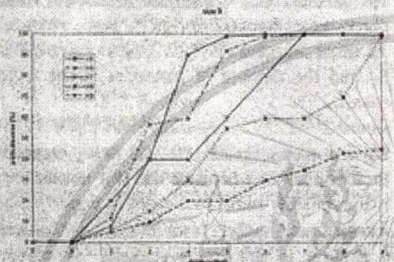
Our results reveal several points of interest: first, no matter what size of the training set is, linearly separable domains with domains of distance level *dis*=9 do not appear sensitive to any amount of imbalance (except *i*=9, we will analysis this case in the next section). As a matter of fact, as degree of



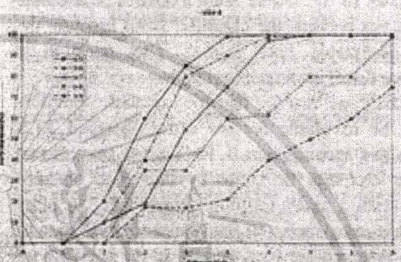
(a) Size 1



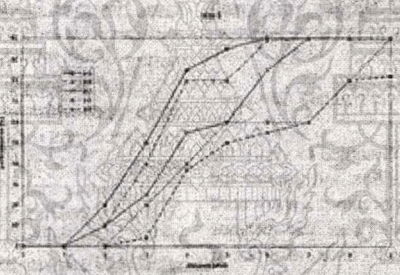
(b) Size 2



(c) Size 3

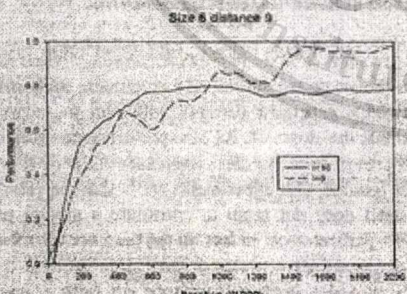


(d) Size 4

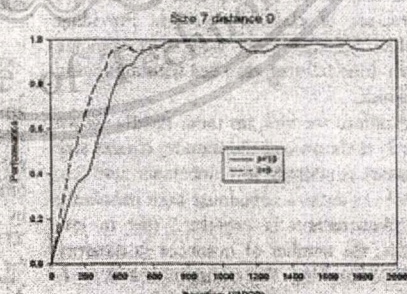


(e) Size 5

Figure 2: XCS and the Class Imbalance Problem



(a) Size 6



(b) Size 7

Figure 3: XCS and the training dataset size

distance between classes decreases, so does the XCS's sensitivity to imbalances. Indeed, we can clearly see in the Figure 2 that as the degree of distance decreases, XCS's performances are caused lower with higher of imbalance level.

As could be expected, imbalance rates are also a factor in the performance of XCS and, perhaps more surprisingly, so are the training set sizes. Indeed, as the size of the training set increases, the effects of imbalance decreases. This suggests that in very large domains, the class imbalance problem may not be a hindrance to a classification system. Specifically, the issue of relative cardinality of the two classes – which is often assumed to be the problem underlying domains with class imbalanced – may in fact be easily overridden by the use of a large enough dataset (if, of course, such a dataset is available and its size does not prevent the classifier from learning the domain in an acceptable time frame). This question is considered in more detail in the next section.

From our research, we can see that the imbalance problem may not always be to blame for the often observed XCS performance loss that accompanies it. Rather, we suggest that this performance loss may be cause by the small distance between classes with a small training dataset size. In other words, a huge class imbalance will not hinder classification of a domain whose distance between classes is far nor will we see a problem if the training dataset is very large. Conversely, a small dataset imbalance can greatly harm a very small dataset or one representing a very small distance between classes.

B. Class imbalance versus Training dataset size and Distance between classes

The experiments of this section were designed to verify the hypothesis proposed in the previous section and answer the question whether or not the loss of performance experienced by XCS is caused by the training dataset size and distance between classes rather than the class imbalance. In order to test this hypothesis in our artificial domains, we kept the same generation scheme with respect to concept of distance between classes, class imbalance and emphasized on the training set size. The amount of training data is known to affect on the performance of classifier systems. Providing additional training data typically leads to a more complex model (e.g. more rules, time training etc.) but with improving classification performance.

For a better visualization, we look up these results shown graphically in Figure 2, it shows that the training dataset size affects slightly to dataset containing low imbalance rate (i.e. i=5, 6) but significantly to dataset containing high imbalance rate (i=8, 9). This phenomenon is explained that in low imbalance rate datasets, the number of instances in minority class is enough for XCS learning. For example, if i=5, s=1, the number of instances in minority class is n_{min} = $round(2^{13}*2^1/(2^5*2^1))$ = $round(2^{13}*2^1/(2^5*2^5))$=16 comparing to number of instances in majority class is n_{maj} = $round(2^{13}*2^1/2^5)$ = $round(2^{13}*2^1/2^5)$ = 512. However, in high imbalance rate datasets, the number of instances in minority class is very small and XCS encounters obstacle during training time so the performance is low. For example, if i=9, s=1; n_{min}= $round(2^{13}*2^1/(2^5*2^1))$ = $round(2^{13}*2^1/(2^5*2^5))$ = 1 comparing to n_{maj} = $round(2^{13}*2^1/2^5)$ = $round(2^{13}*2^1/2^5)$ = 512, or the imbalance ratio ir=1:512.

With extremely imbalance datasets: i=9 when dis=9, we increase the training dataset size from 513 instances (the top-left graph), correlative to s=1, to 8208 instances (the below-center), correlative to s=5, the performance of XCS increases from 20% to 82%.

In order to emphasis more detail on the function of training dataset size, we test performance of XCS in imbalance levels with respect to distance between classes, dis =9 and s=6 or s=7. In this case, we have the number of instances in minority class: n_{min}= $round(2^{13}*2^1/(2^5*2^1))$ = $round(2^{13}*2^6/(2^5*2^6))$ = 32 or n_{min}= $round(2^{13}*2^1/(2^5*2^1))$ = $round(2^{13}*2^7/(2^5*2^6))$ = 64 and the number of instances in the majority class n_{maj}= $round(2^{13}*2^6/2^5)$ = $round(2^{13}*2^6/2^5)$ = 16384 or n_{maj}= $round(2^{13}*2^6/2^5)$ = $round(2^{13}*2^7/2^5)$ = 32768 to implement the imbalance levels i=9 and i=10, respectively. We chose these values of i because the two imbalance levels represent two extremely imbalance cases in which we believe that many learners will suffer from biases to the majority class and according to [8], with standard or adjustment parameter settings, XCS encounters problem to solve these imbalance rates. However, by experiments showing in Figure 3, XCS can perform well in these cases. In all runs, XCS was trained during 4,000,000 learning iterations, but only the first 2,000,000 are shown for a better visibility. These results once again agree with our assertion. In training size level s=6 with imbalance level i=10, XCS reaches to 80% and with i=9, XCS reaches to 95% after $2*10^6$ iterations, stabilizing at 100% after $4*10^6$ explore trials. We increase the training size level to s=7, XCS even reaches to 100% performance with i=9 only after $6*10^5$ iterations and with i=10, XCS nearly reaches 100% after $2*10^6$ explore trials.

The goal of our experiments in the previous section and this section is to show that the performance of XCS, through hindered by class imbalance, is repaired as the training dataset size and distance between classes increase.

VI. Conclusions and Future Directions

In this work, we developed a systematic study using a set of artificially generated datasets with aim to answer question whether the loss of XCS's performance with imbalance problems is from the class imbalance itself or it is caused by other reasons. Our experiments show that the class imbalance by itself does not seem to constitute a crucial problem for XCS's performance. In fact, in the presence of imbalance with far distance between classes and large training dataset size, XCS provides high performance on both classes even in extremely imbalance datasets. In contrast, the combination of class imbalance, small distance between classes and small training dataset size make a degradation of XCS's classification accuracy.

Our future works are to extend the study to a more general class of problems, effects of injecting various degrees and types of noises to XCS's performance. Another important consideration has to do that is studying sampling techniques [10] in conjunction with XCS to enhance quality classification performance. Least but not last, it is imperative that we conduct experiments on a number of real-world datasets to verify that the hypothesis we posited on simple artificial datasets actually does apply to general and actual data.

ACKNOWLEDGMENT

We would like to thank to AUN-SEED/Net about the financial support for doing this research. We are also grateful to Research Center for Communications and Information Technology (ReCCIT) and Department of Computer Engineering, KMITL Ladkrabang for the help in research equipments.

REFERENCES:

[1] P.L. Lanzi, W. Stolzmann, S.W. Wilson, *Learning Classifier Systems: From Foundations to Applications*, Springer Berlin/Heidelberg, 2000.

[2] P.N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison Wesley, USA, 2006.

[3] Stewart W. Wilson, "Classifier Fitness Based on Accuracy," *Evolutionary Computation*, Vol.3 (2), (1995), pp.149-175.

[4] Butz, M. and Wilson, S., "An algorithmic description of XCS", *Journal of Soft Computing 6*, (2002), pp.144-153.

[5] E. Bernadó-Mansilla, J.M. Garrell Guiu, "Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks", *Evolutionary Computation 11*, vol. 3, pp. 209-238, 2003.

[6] Orriols-Puig A., Goldberg D.E., Sastry K., and Bernadó-Mansilla E. "Modeling XCS in Class Imbalances: Population Size and Parameter Settings". *Technical report, IlliGAL Report No. 2007001, USA*. Feb. 2007.

[7] Albert Orriols and Ester Bernadó-Mansilla, "Data Imbalance in UCS Classifier System: Fitness Adaptation", *IEEE Congress on Evolutionary Computation - CEC2005*, vol. 1, pp. 604-611, 2005.

[8] Orriols-Puig A. and Ester Bernadó-Mansilla, "Bounding XCS's Parameters for Unbalanced Datasets", *GECCO'06*, pp. 1561 - 1568, July 8-12, 2006, Seattle, Washington, USA.

[9] Chris S., Taghi M.K., Jason V.H., Andres F., "An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data", *Information Reuse and Integration, 2007. IEEE International Conference*, pp. 651-658, August 2007.

[10] Nitesh V.C., Kevin W.B., Lawrence O.H., W. Philip K., "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research* 16 pp. 321–357, 2002.

[11] N. Japkowicz and S. Stephen, "The Class Imbalance Problem: A Systematic Study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429-450, 2002.

[12] N. V. Chawla, "C4.5 and imbalanced datasets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure", *in Proceedings of the ICML'03 Workshop on Class Imbalances*, 2003.

[13] Zhi-Hua Zhou and Xu-Ying Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem", *Knowledge and Data Engineering, IEEE Transactions*, vol. 18(1), pp. 63-77, January 2006.

[14] Jo, T. and Japkowicz, N., "Class Imbalances versus Small Disjuncts", *SIGKDD Explorations* 6(1), June 2004.

[15] R.C. Prati, G.E.A.P.A. Batista, and M.C. Monard, "Class Imbalances *versus* Class Overlapping: An Analysis of a Learning System Behavior", *the Third Mexican International Conference on Artificial Intelligence - MICAI 2004*, LNAI 2972, pp. 312–321, 2004.

[16] Weiss, G. & Provost, F. "The Effect of Class Distribution on Classifier Learning: An Empirical Study", *Technical Report ML-TR-44, Department of Computer Science*, Rutgers University, 2001.

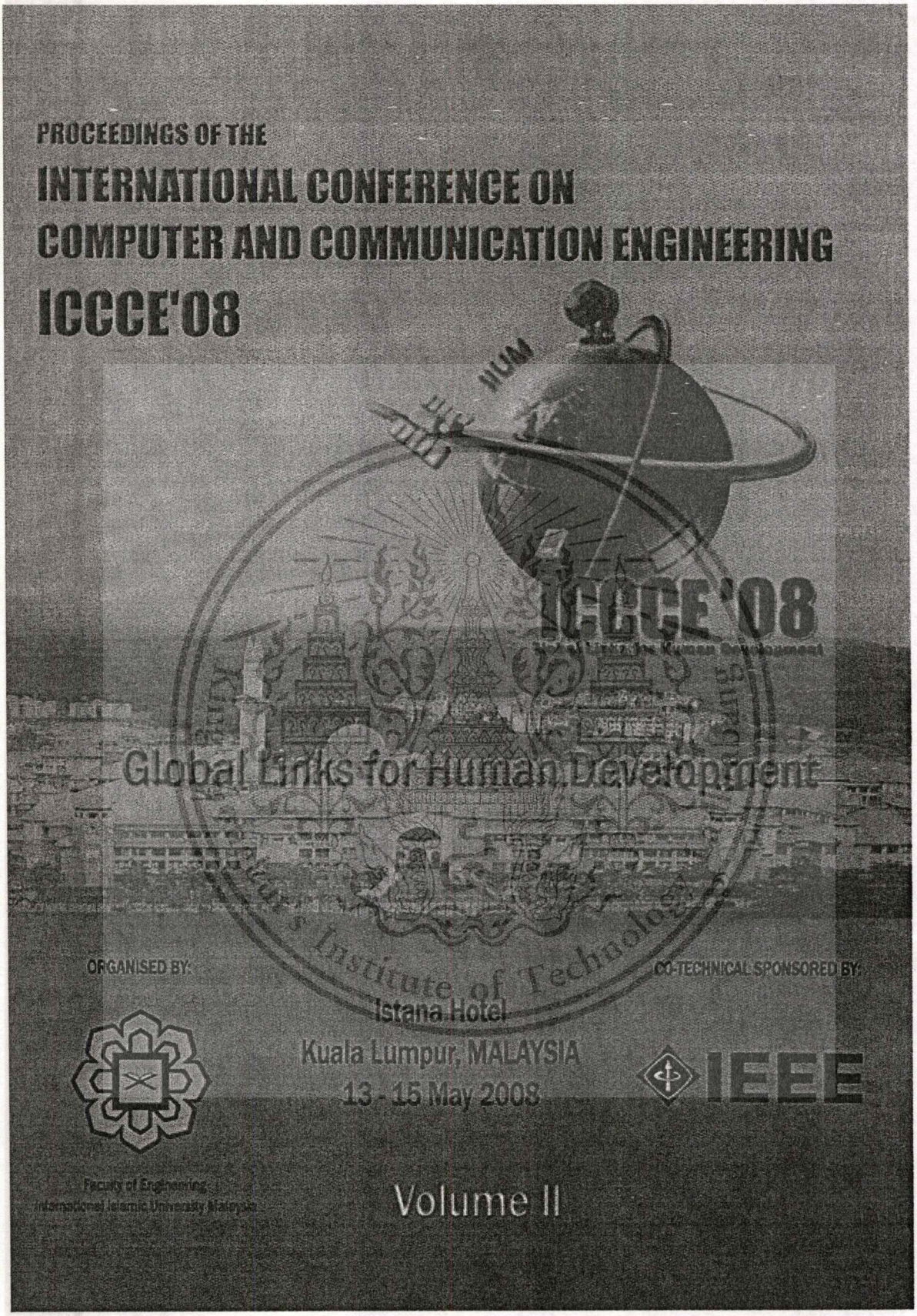
[17] G.M. Weiss, "Mining with Rarity - Problems and Solutions: A Unifying Framework," *SIGKDD Explorations*, vol. 6, no. 1, pp. 7-19, 2004.

Thach Huy Nguyen received B.S. degree in information technology from Hanoi University of Technology, Vietnam, in 2006. Currently, he is a Master student under scholarship of AUN/SEED-Net, JICA, Japan, in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok 10520 THAILAND. His research interests include Learning Classifier System, Genetic Algorithm, Rough Set and Classification.

Sombut Foitong received the B.S. degrees in mathematical – computer from Ramkhamhaeng University and the M.Eng degrees in computer engineering from King Mongkut's Institute of Technology Ladkrabang, Bangkok, THAILAND. Currently, he is a PhD student in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, THAILAND. His research interests include fuzzy classification, rule-based generation, rough set and soft computing.

Somchai Udomthanapong received the B.S. degree in computer engineering from King Mongkut's Institute of Technology Ladkrabang, Bankok, Thailand. Currently, he is a master degree student in the Department of Computer Engineering from King Mongkut's Institute of Technology Ladkrabang, Bankok 10520 Thailand. His research interests include classification Learning Classifier System, Genetic Algorithm, Neural Network, Data Mining.

Ouen Pinngern received the MS degree from Oregon State University in USA in 1983 and the PhD degree from University of Nebraska at Lincoln in USA in 1986 both in computer science. He is an associated professor in the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, THAILAND. His current research interests lie in the areas of soft computing, machine learning, data mining, information retrieval, and model of computation. He has been appointed as a Director of Research Center for Communications and Information Technology (ReCCIT) which is promoted to be a Center of Excellence in ICT under KMITL.



Proceedings of the International Conference on Computer and Communication Engineering 2008 May 13-15, 2008 Kuala Lumpur, Malaysia

Rough Set and XCS in Classification Problems

Thach H. Nguyen*, Sombut Foitong† and Ouen Pinngern‡
Department of Computer Engineering, Faculty of Engineering,
Research Center for Communication and Information Technology (ReCCIT)
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520 THAILAND.
E-mail: nguyenhuythach1983@yahoo.com, s8060022@ kmitl.ac.th†, kpouen@kmitl.ac.th‡*

Abstract

XCS is known to degrade in classification performance when faced with many features that are redundant for rules discovery. In this paper, we propose a novel system combining of rough sets and XCS to deal with the mentioned problem. Firstly, rough set theory is used to handle inconsistent input datasets. The purpose of feature reduction by rough set is to identify the most significant attributes and eliminate the irrelevant ones to form a good feature subset for classification. Secondly, the reduced datasets are used to create a set of rules by using XCS. The main contribution of XCS to learning theory is its rules generation without experts. Finally, by applying the set of rules, we can classify unseen datasets into their specific classes. Experimental results on real-life datasets show that the proposed method can reduce storage space as well as can preserve and may also improve solution accuracy. Beside that, the rule retrieval time is also greatly reduced because the use of Rough-XCS classifier contains a smaller amount of instances with fewer features. Furthermore, the proposed method has a high potential to be used as a mean to construct a classifier system that copes with incomplete, noisy and chaotic data.

Keywords: Classification, Learning Classifier System, XCS, Rough set, redundant datasets.

I. INTRODUCTION

Learning classifier systems (LCSs) are adaptive systems, often called Genetics Based Machine Learning Tools, that learn to achieve a task through interacting with environment. Introduced in 1995 by Wilson, accuracy-based learning XCS is one of the most successful learning classifier systems [2]. XCS classifier system has recently shown a high degree of competence on a variety of data mining problems. There are some performance comparisons of strength-based fitness systems and accuracy-based fitness systems showing that accuracy-based fitness systems, including XCS, have a significant impact on data mining field [3], [4].

Datasets containing redundant attributes appear in many applications of data mining and machine learning. These datasets may be of very high dimensionality and consist of much complex structure that even most well planned data mining or analysis techniques might have difficulty extracting meaningful patterns from it.

Feature Reduction (FR) is commonly applied as preprocessing step to overcome the curse of dimensionality, where many types of data analysis become significantly harder as the dimensionality of the data increases. FR removes non-informative features and preserves informative ones. The related work to FR involves reduction of pattern dimensionality through feature selection or feature extraction methods. Many methods have been proposed for feature reduction, such as Principal Component Analysis and Genetic Algorithms. Another often used approach in FR is rough sets (RS) [5]. Rough sets theory provides a formal and robust of manipulating the roughness of the knowledge in information systems. It is a powerful tool for data analysis and knowledge discovery from imprecise and ambiguous data. The effectiveness of RS is which has been demonstrated in many different domains as medicine, economics, finance and business [7].

In this paper, we propose a combination of Rough set and XCS, called Rough set-XCS or RS-XCS, to solve datasets containing redundant attributes. Employing an idea from [6], the approximate reduct is

978-1-4244-1692-9/08/$25.00 ©2008 IEEE

used in our approach. By generating different approximate reducts in FR, the "best" subset of features is obtained. We use this subset to train XCS to increase classification accuracy. Eleven UCI datasets were used in the empirical study. The results suggest that our approach improves performance of the classifier system significantly.

The rest of paper is organized as follows: In section II, XCS classifier system is introduced briefly. Section III analyzes the performances of XCS on original datasets. In section IV, the approximate Rough set-based attribute reduction is described. The framework of our proposed method, experimental results and comparisons will be presented on section V. Finally, we present our main conclusions and discuss future works.

II. XCS Overview

XCS represents the knowledge extracted from the problem in a set of rules. This ruleset is incrementally evaluated by means of interacting with the environment through a reinforcement learning scheme and is improved by a search mechanism based on a genetic algorithm. XCS evolves a population [P] of classifiers, where each classifier consists of a condition, an action and four main parameters: prediction p, prediction error ε, fitness F and numerosity *num*. This section gives a short introduction of XCS. The more details are referred to [2].

A. Performance Component

At each step, an input x is presented to the system. Given x, the system builds a match set [M], which is formed by all the classifiers in [P] whose conditions are satisfied by the input example. If the number of actions represented in [M] is less than a threshold θ_{mna}, then covering is triggered to create a new classifier that matches the current input and has a random action from those not present in [M]. From the resulting match set, an action must be selected and sent to the environment. For this purpose, a payoff prediction $P(a_i)$ is computed for each action a_i in [M]. $P(a_i)$ estimates the payoff that the system will receive if action a_i is chosen. It is computed as fitness-weighted average of the predictions of all classifiers proposing that action. The system chooses the winning action based on these prediction values. The chosen action determines the action set [A] which consists of all the classifiers in [M] advocating this action.

B. Reinforcement Component

Once the action is sent to the environment, the environment returns a reward r, which is used to update the parameters of the classifiers in [A] following order [2]: prediction, prediction error, and finally fitness. Prediction p is updated with learning rate as follows:

$$p \leftarrow p + \beta(r - p) \qquad (5)$$

Where $\beta\,(0 \le \beta \le 1)$ is the learning rate. Then, the prediction error is updated as: $\varepsilon \leftarrow \varepsilon + \beta(|r - p| - \varepsilon)$. Finally, classifier fitness is updated in two steps: first, the *relative accuracy* κ' of the classifiers in [A] is computed; then κ' is used to update the classifier fitness as: $F \leftarrow F + \beta(\kappa' - F)$.

III. XCS with Datasets Containing Redundant Attributes

This section analyses effects of datasets containing redundant attributes to accuracy and speed performance of XCS. For this purpose, we ran XCS on eleven UCI datasets [8] shown in table I. These datasets are individually significant to the evaluation of XCS for redundant datasets. We ran XCS with the following parameter setting (as introduced in [2]): N= 6400, β=0.2, α=0.1, ε_0=1, ν=5, θ_{GA}=25, χ=0.8, μ=0.04, θ_{del}=20, δ=0.1, θ_{sub}=200, $P_\#$=0.6.

TABLE I. UCI DATASET DESCRIPTION

Dataset	#Attr.	#Inst.	#Class	XCS accuracy
Balloon1	4	20	2	100%
Corral	6	64	2	83.3%
SPECTF	45	267	2	74.07%
Monk3	6	432	2	95%
Balance-scale	4	625	3	81.6%
m-of-n	13	1000	2	72%
Exactly	13	1000	2	69%
Parity5+2	10	1024	2	50%
Car	6	1728	4	98.2%
Led(10%n)	24	2000	10	61.8%
Mushroom	22	8124	2	99.2%
Average				80.38%

#Attr. = number of attributes, #Inst. = number of instances, #Class = number of classes.

The first test sets are small datasets and no noise as *Ballom1*, *Corral*, *balance-scale* and *SPECTF*. Experiments show that the training time of XCS on these datasets are short and population is small. The second experiments require complex sets in number of instances. We test XCS with larger datasets (the number of instances is greater than or equal to 1000).

These experiments will test the ability of XCS to maintain a large population size in order to represent the classification accurately. By experiments, the explore time and population size of XCS on these datasets increase significantly. The last experiments test XCS on noisy datasets which are *Monk3* (5% noise) and *Led* (10% noise). Datasets containing noise could be preventing XCS from forming accurate generalizations and thereby hinder the search process of the GA within XCS.

Experiments in table I show performances of XCS on original datasets. We can see that XCS performs best on *Ballooon1* where 100% data are classified correctly. *Parity5+2* is classified correctly only 50% that is dataset classified worst. However, the notice datasets are *Mushroom* and *Car*. Although these datasets contain a large number of instances, their accuracies are still high. This is because not only the number of attributes and instances affect to performance of XCS, but also there are other factors affecting to performance of XCS such as: complexity of problem and distribution of dataset. This may be our future works.

IV. Approximate Reduct

In this section, we focus on our approach, approximate reduct. The more details of Rough set foundation are referred to [5]. Our combination of feature reduction uses β-reduct or approximate reduct with the definition as follows:

Let IS=(U, A, f) be an information system, where U is a finite nonempty set of N objects $\{x_1, x_2, \ldots, x_N\}$; A is a finite nonempty set of n attributes (features) $\{a_1, a_2, \ldots, a_n\}$; f_a: $U \rightarrow V_a$ for any $a \in A$, where V_a is called the domain of attribute a.

B is called a β-reduct or approximate reduct of conditional attribute set A if B is the minimal subset of A such that $\delta_B^D \geq \beta * \delta_A^D$. Where $D=\{d\}$ is the decision attribute and δ_A^D is the relative dependency degree of A with regard to D [5]. The rough set-based attribute reduction algorithm in our developed approach is given as Fig.1.

In algorithm of Fig. 1, the parameter β, $\beta \in [0, 1]$, is

called the consistency measurement. β represents how consistent the sub-decision table (with respect to the considered subset of attributes) is relative to the original decision table (with respect to the original attribute set). It also reflects the relationship of the approximate reduct and the exact reduct.

For selecting feature subsets from datasets with a lot of redundant attributes, we select the best attributes one by one by using the evaluation criterion of dependency γ, until a reduct is found. A feature subset, good or not, depends on the dependency of *decision D* on that feature subset. To select a strong feature subset, the following selection strategies are used:

- Selecting the features that cause the number of consistent instances increases faster, to obtain the subset of features as small as possible.
- The size of maximal subset in $POS_R(D)/IND(R, D)$ should be considered. In general, the more the number of attribute values in a feature in R, the more the number of subsets, and the smaller the size of the maximal subset. Selecting a feature, by which a bigger subset can be acquired, is a way for our purpose.
- Next, we consider the consistent instances. Let *largest* $POS_{R\cup\{a\}}(D)/IND(RU\{a\}, D)$ denote the size of the maximal subset of the lower approximation of the set $POS_{RU\{a\}}(D)$.

```
//step 1
R = {};
x = γ_C(D);
While (γ_R(D) < β*x)
{
    ∀a ∈ C − R
        Choose a with largest γ_{R∪{a}}(D);
        If ties occur, select a with largest
        POS_{R∪{a}}(D) / IND({R ∪ {a}, D});
        If further ties occur then select a with smallest
        |POS_{R∪{a}}(D) / IND({R ∪ {a}, D})|;
    R = R ∪ {a};
    U = U − POS_R (D); //Remove all consistent instances
    C = C − {a};
}
//step 2
∀ a ∈ R
    If γ_{R−{a}}(D) = γ_R(D) then R = R − {a};
Return R;
```

Figure 1: Approximate Reduct Algorithm

- When two features have the same performance described above (ties occur), the one that contains a less number of different values will

be selected.

- After performing all steps above, we should have a last step to remove all redundant features by re-computing the dependency of decision D on each feature.

V. Combining Rough Set and XCS

A. The Framework

We address problems discussed in section III by combining rough set-based FR approach with XCS learning classifier system as Fig. 2. Feature importance is taken into account through reduct generation. The features in the reduct are considered to be important while other features are considered to be irrelevant. Reduct computation by RS does not require any domain knowledge and the computational complexity is only linear with respect to the number of attributes and instances [5]. After combining the FR method and XCS, the knowledge representation is still the same as that the original one. This form of knowledge representation is easier to understand and more convenient for classifying unseen inputs. Furthermore, since only the features in the reduct are evolved in the rule generation of XCS, the running time is also reduced.

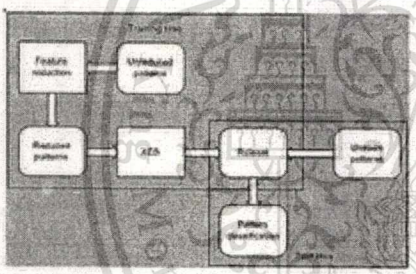


Figure 2: the RS and XCS combination framework

In redundant-datasets classification problems, there are three main benefits from combining RS and XCS. 1) Classification accuracy can be preserved or even improved by removing non-informative features, redundant and noisy instances, 2) Storage requirements are reduced by deleting irrelevant features and redundant instances, and 3) the classification decision response time can be reduced because fewer features and instances will be examined when an unseen instance occurs. As shown in Fig. 2, the system in this paper includes the following modules.

+ Rough Set-based Feature Reduction: Reads a set of feature patterns and outputs another subset with reduced dimensionality, implemented with the Rough set-based Feature Reduction algorithm.

+XCS-based Rule induction: Reads a sequence feature patterns and outputs a set of *if-then* rules connecting features and the implied classes.

B. Experimental Results

1) Attribute reduction using RS

In this section, we test and analyze the feature reduction capability of the rough set-based algorithm proposed in the section IV. The results of this algorithm on 11 datasets described on the previous section are shown on the second and third columns of table II.

Step 1: **Initialize** $Acc = \phi$ and β=1

Step 2: **While** (β>0)

+ **Implement** Approximate Reduction Algorithm; (Output the reduced dataset RD)

+ **Training** XCS with RD.

+ **Implement** test XCS with unseen instances; (Output the current accuracy, a)

$Acc \Leftarrow Acc \cup \{a\}$;

$\beta = \beta - \lambda$;

Step 3: **Find** a^*, $a^* = max\{a \in Acc\}$; and find the corresponding β^* and R^*.

Step 4: **Output** a^*, β^* and R^*.

Figure 3: The RS and XCS combination algorithm

The *SPECTF* dataset achieves the greatest attribute reduction, reducing the number to attributes to 3 (i.e. only 1/15 of 45 unreduced attributes), however only one instance is reduced (from 267 instances to 266 instances). With *Mushroom* dataset, when number of attributes is reduced from 22 to 5, the number of instances is reduced significantly from 8124 to 47 instances. This also occurs with *Parity5+2*, *m-of-n*, *Exactly* and *Led(10%n)* datasets. On datasets where there is no redundant or noise attribute as *Car* and *Balance-scale*, there is no reduction on any value of β. The remained datasets achieve the attribute reduction and instance reduction slightly.

In Fig. 1, we can see that the larger the value of β, the less number of attributes and instances are reduced. However, our objective is to find the value of β in which accuracy of XCS is highest. We ran the proposed system on 11 datasets with various values of

β as algorithm in Fig. 3. We found that the best β value after applying both RS and XCS for these real-life datasets is 0.95. This is why the β value is set at 0.95 in this paper.

2) XCS on reduced dataset

TABLE II: XCS AND RS-XCS PERFORMANCES

DATASET	#Red. Attributes	#Red. Instances	RS-XCS accuracy
Balloon1	2	4	100%
Corral	4	16	85%
SPECTF	3	266	80%
Monk3	3	36	99%
Balance-scale	4	625	81.6%
m-of-n	6	64	76%
Exactly	6	64	74%
Parity5+2	5	32	75%
Car	6	1728	98.2%
Led(10%n)	5	10	75%
Mushroom	5	47	100%
Average			85.80%

#Red. Attributes = Reduct attributes, #Red. Instances = Reduct instances,

Experiments in table II show that after attribute reduction, all the XCS performances were able to preserve or even improve classification accuracy. The most improved classification accuracy is occurred on two datasets *Monk3* and *Led*(10%n). This is because these datasets contain noisy instances and after applying Rough set-based attribute reduction, not only redundant attributes are eliminated but also noisy instances are. Note that since Rough set-XCS specializes in the removal of redundant attributes and noises, it might not improve performance on complex datasets as *Car* and *Balance-scale*. In such settings, we might expect Rough set-XCS to throw out perfectly good data. On average, the classification accuracy after applying the combination of RS and XCS increase by 5.42 percent.

To conclude, when Rough set-XCS is applied, the results of almost datasets are positive. The classification accuracy and speed performance show a notable improvement.

VI. CONCLUSIONS AND FUTURE WORKS

We have introduced a combination of an approximate Rough set-based feature reduction approach and XCS learning classifier system to solve datasets containing redundant attributes. In the FR approach, the concept of a reduct is generalized to an approximate reduct, which makes the reduct computation faster and more flexible. In some situations, the crisp reduct (β=1) is the best subset of features in term of the classification accuracy. In some other situations, the crisp reduct is not the optimal subset of feature; in this paper, the β value is determined to optimize the classification accuracy. The developed approach can remove not only the redundant attributes but also the noise instances. We first analyzed effects of these datasets to performance of XCS. The results show that XCS is quite robust to datasets containing redundant attributes although the speed is slow, especially with large datasets. This is mainly the fact that when implementing classifier tasks, XCS tries to cover all space of problem. The combination of approximate Rough set-based feature reduction and XCS that we proposed makes it possible to address this problem. By this combination, we could further enhance the accuracy; reduce training time and the storage. We tested 11 UCI datasets by XCS first and Rough set-XCS following. The results show that performances of the proposed method are improved significantly in term of accuracy.

There are still some limitations of our developed FR and XCS approaches, which may need to be tackled in our future work: 1) The determination of using dummy variables or nominal categorical variables in rule representation of XCS is empirical and heuristic based during the testing and depends on dataset. 2) The rough set-based FR method works better with symbolic data. The numerical data needs to be discretized before applying FR process. We plan to expand the framework to deal with real-value inputs with multiple classes, and to test the system on other datasets.

ACKNOWLEDGEMENT

This work was sponsored by the AUN/SEED-Net organization. We thank Kreangsak Tamee for making to us aware of Wilson's XCS, to Phaitoon Srinil for pointing out important properties of redundant datasets and to Somchai Udomthanapong for essential help in guiding during our simulation.

REFERENCES

[1] P.N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison Wesley, USA, 2006.

[2] Stewart W. Wilson, "Classifier Fitness Based on Accuracy," *Evolutionary Computation*, Vol.3 (2), pp.149-175, 1995.

[3] E. Bernadó-Mansilla, J.M. Garrell Guiu. "Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks". *Evolutionary Computation 11*, vol. 3, pp. 209-238, 2003.

[4] A. J. Bagnall, and G. C. Cawley, "*Learning classifier systems for data mining: A comparison of XCS with other classifiers for the Forest Cover dataset*", In Proceedings of the IEEE/INNS

International Joint Conference on Artificial Neural Networks (IJCNN-2003), vol. 3, pp. 1802-1807, 2003.

[5] Komorowski, J., Pawlak, Z., Polkowski, L., et. al., "Rough Sets: A Tutorial," in *Rough Fuzzy Hybridization: A New Trend in Decision-Making* (ed. S. K. Pal, A. Skowron), Springer-Verlag, Singapore, pp. 3-98, 1999.

[6] Dominik Slezak, "Approximate Entropy Reducts", *Journal of Fundamenta Informaticae*, vol. 53 (3-4), pp.365 - 390, 2002.

[7] Pawlak, Z., "Rough set theory for intelligent industrial applications", *Proceedings of the Second IEEE International Conference on Manufacturing of Materials*, vol. 1, pp.37-44, 1999.

[8] http://archive.ics.uci.edu/ml/datasets.html.

山东大学

SHANDONG UNIVERSITY

27 Shanda Nanlu, Ji'nan
P. R. China 250100
TEL: +86-(0)531-88391367
FAX: +86-(0)531-88391367
http://www.sdu.edu.cn
E-mail: nc2008@sdu.edu.cn

8 May 2008

Dear Nguyen Huy Thach, Porntep Rojanavasu and Ouen Pinngern:

Paper ID : P2620
PaperTitle : Cost-sensitive XCS Classifier System Addressing Imbalance Problems

On behalf of the organizing committee and with great pleasure, we invite you to participate in the 4th International Conference on Natural Computation (ICNC'08) and the 5th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'08), to be held jointly in Jinan, Shandong, China during 18–20 October 2008. (For more information regarding the joint conferences, please visit http://www.icnc-fskd2008.sdu.edu.cn/).

We look forward to meeting you in Jinan in October!

Yours sincerely,

Dr. Jun Ma and Dr. Yilong Yin
General Chair, ICNC'08-FSKD'08
Professor of Computer Science
Tel: +86-531-88391367
Fax: +86-531-88391367
Email: majun@sdu.edu.cn
ylyin@sdu.edu.cn

P.S.: For some unexpected reasons, the dates of the conferences have been changed from August 25-27, 2008 to the new time period October 18-20, 2008. We sincerely apologize for the possible inconvenience to you.

Accepted by FSKD'08, China, 18-20 October, 2008

Cost-sensitive XCS Classifier System Addressing Imbalance Problems

Nguyen Huy Thach*, Porntep Rojanavasu† and Ouen Pinngern‡
Department of Computer Engineering, Faculty of Engineering,
Research Center for Communication and Information Technology (ReCCIT)
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520 THAILAND.
E-mail: s9060654@kmitl.ac.th*, s8060022@kmitl.ac.th†, kpouen@kmitl.ac.th‡

Abstract

The class imbalance problem has been recognized as a crucial problem in machine learning and data mining. Learning systems tend to be biased towards the majority class and thus have poor generalization for the minority class instances. This paper analyses the imbalance problem in accuracy-based learning classifier systems. In particular, we propose a novel approach based on XCS classifier system and cost-sensitive learning. In our approach, the reward value of correctly identifying the positive (rare) class outweighs the value of correctly identifying the common class. This research provides guidelines to set reward base on the dataset imbalance ratio and a method to calculate reward online base on the information collected by XCS during training is also proposed. Experimental results on synthetic and real-life datasets show that, with appropriate reward settings, XCS is robust to class imbalances.

Keywords - Classification, Learning Classifier System, XCS, cost-sensitive learning, imbalance problem.

1. Introduction

Learning classifier systems (LCSs) are adaptive systems, often called Genetics Based Machine Learning Tools, which learn to achieve a task through interacting with environment. Introduced in 1995 by Wilson, accuracy-based learning XCS is one of the most successful learning classifier systems [3]. XCS classifier system has recently shown a high degree of competence on a variety of data mining problems [2]. There are some performance comparisons of strength-based fitness systems and accuracy-based fitness systems showing that accuracy-based fitness systems, including XCS, have a significant impact on data mining field [4].

Recently, the class imbalance problem, where examples in dataset belonging to one class heavily outnumber the examples in the other classes, has been recognized as a crucial problem in machine learning and data mining. There are many methods and algorithms proposed to solve this problem. A survey of methods solving imbalanced datasets is provided by N. Japkowicz [10].

Cost-sensitive learning has been already attracted much attention from the machine learning and data mining communities. Many cost-sensitive learning methods have been developed and they are shown effective methods for solving imbalance problems [7], [8]. For example, in medical diagnosis, the cost of erroneously diagnosing a patient to be healthy may be much bigger than of mistakenly diagnosing a healthy person as being sick, because the former kind of error may result in the loss of a life.

In this paper, we propose an approach based on XCS and cost-sensitive learning, called Cost-sensitive XCS, to solve imbalance datasets. Cost-sensitive XCS allows one to assign different rewards (costs) to different types of misclassification. It is noteworthy that this technique does not need modify the architecture or training algorithms of the standard XCS, therefore, it is very easy to use. A synthetic dataset and seven UCI datasets were used in the empirical study. The results suggest that our approach improves performance of classifier system significantly on highly imbalanced datasets and slightly on lowly imbalanced datasets.

The rest of paper is organized as follows: The application of data mining techniques to class imbalance problems are discussed in section 2. In section 3, we introduce the performance evaluation measure used in our method. XCS classifier system is described briefly in Section 4. In section 5 and section 6, the experimental results and comparison with standard XCS will be presented to verify the feasibility of Cost-sensitive XCS. Section 7 discusses extensions of the current work and finally, we provide conclusions and future works.

2. Related works

To solve class imbalance problem, many methods and algorithms have been proposed. Some of the best well-known approaches are applied at the sampling level. In [12], Chawla et al. presented SMOTE which can be done either over-sampling minority class or under-sampling the majority class. Both methods can be applied in any concept learning system, since they act as a preprocessing phase, allowing the learning system to receive the training

Accepted by FSKD'08, China, 18-20 October, 2008

instances as if they belonged to a well-balance dataset. Thus, any bias of the system towards the majority class due to the different proportion of examples per class would be expected to be suppressed. In addition, in [13], the effects of sampling method, probabilistic estimate and decision tree structure of C4.5 on imbalance datasets were investigated. Beside that, G.M. Weiss and F. Provost [14] analysis the effects of imbalance datasets to classifier learning systems and how they affect through the evaluation of learned classifiers by using two performance measures: AUC and classification accuracy. A new approach in cost-sensitive, cost-sensitive neural networks are proposed instead of decision trees [9] to solve multiclass tasks. And in [15], G.M. Weiss indicated the learning from imbalance and rarity datasets can be handled in a similar manner. Recently, Chris S. et al. [11] evaluated the performance of seven commonly-used sampling techniques on imbalance and noise datasets to study the effects of noise to imbalance problem. Or in [19], the authors proposed a novel algorithm on neural network classification to calculate a direction to decrease the error of each class in imbalance datasets based on weight-space. Beside that, the paper provided a detail anaylysis of standard backpropagation neural network.

In learning classifier systems field, A. Orriols and E.B. Mansilla [6] proposed a solution dealing with the class imbalance problem by finding fitness adaptation based on class-sensitive accuracy in combination with UCS, a supervised LCS derived from XCS, as a useful tool for alleviating the effects of class imbalances. In fact, because XCS, UCS and some other learning classifier systems have their fitness based on accuracy, they present a high bias toward the majority class instances and evolve easily *overgeneral* classifiers. Idea of the proposal is restrict classifiers to cover regions formed by examples of a single class and make accuracy class-sensitive rather than instance-sensitive. Thus, accuracy is modified so that each class is considered equally important regardless of the number of instances representing in each class. The experiments show that this model evolved with imbalance level up to i=7.

3. Evaluation Measure

Since the accuracy measure, a fraction of records classified correctly and total input records, treats every class as equally important, it may not be suitable for analyzing imbalanced datasets, where the rare class is considered more interesting than the majority class. So, in our paper, we employ the *Precision, Recall, F1-measure* and *g-mean* to evaluate the Cost-sensitive XCS performance.

The confusion matrix, as shown in Table I, is typically used to evaluate the performance of the machine learning algorithms in a two-class problem. Based on the confusion matrix, the *Precision (Pr), Recall (Re), F1-mean (F1)* and *g-mean (g)* can be defined as follows:

$$Pr = TP/(TP+FP) \quad (1)$$

$$Re = TP/(TP+FN) \quad (2)$$

$$F1 = 2*Re*Pr/(Re+Pr) = 2*TP/(2*TP+FP+FN) \quad (3)$$

$$g = \sqrt{acc+ * acc-} \quad (4)$$

Where $acc+ = TP/(TP+FN) = Re$, $acc- = TN/(TN+FP)$.

TABLE I: CONFUSION MATRIX

		Predicted Class	
		+	-
Actual class	+	f+(TP)	f-(FN)
	-	f-(FP)	f-(TN)

TP=True Positive, FN=False Negative, FP=False Positive and TN=True Negative.

From the equation (3) and (4), it is obvious that when both *precision* and *recall* are high the *F1-measure* and *g-mean* will be high. Thus the *F1-measure* and *g-mean* are fit for evaluating the performance of a machine learning algorithm on the minority class.

4. XCS overview

XCS represents the knowledge extracted from the problem in a set of rules. This ruleset is incrementally evaluated by means of interacting with the environment through a reinforcement learning scheme and is improved by a search mechanism based on a genetic algorithm (GA). XCS evolves a population [P] of classifiers, where each classifier consists of a condition, an action and four main parameters: prediction p, prediction error ε, fitness F and numerosity num. This section gives a short introduction of XCS. The more details are referred to [3].

4.1. Performance Component

At each step, an input x is presented to the system. Given x, the system builds a match set [M], which is formed by all the classifiers in [P] whose conditions are satisfied by the input example. If the number of actions represented in [M] is less than a threshold θ_{mna}, then covering is triggered to create a new classifier that matches the current input and has a random action from those not present in [M]. From the resulting match set, an action must be selected and sent to the environment. For this purpose, a payoff prediction $P(a_i)$ is computed for each action a_i in [M]. $P(a_i)$ estimates the payoff that the system will receive if action a_i is chosen. It is computed as fitness-weighted average of the predictions of all classifiers proposing that action. The system chooses the

Accepted by FSKD'08, China, 18-20 October, 2008

winning action based on these prediction values. The chosen action determines the action set [A] which consists of all the classifiers in [M] advocating this action.

4.2. Reinforcement Component

Once the action is sent to the environment, the environment returns a reward r, which is used to update the parameters of the classifiers in [A] following order [3]: prediction, prediction error, and finally fitness. Prediction p is updated with learning rate as follows:

$$p \leftarrow p + \beta(r - p) \qquad (5)$$

Where $\beta\,(0 \le \beta \le 1)$ is the learning rate. Then, the prediction error is updated as: $\varepsilon \leftarrow \varepsilon + \beta(|r - p| - \varepsilon)$. Finally, classifier fitness is updated in two steps: first, the *relative accuracy* κ' of the classifiers in [A] is computed; then κ' is used to update the classifier fitness as: $F \leftarrow F + \beta(\kappa' - F)$.

5. XCS on imbalance problems

This section analyses effects of imbalance datasets to the performance of XCS. For this purpose, we ran XCS with 11-MUX problem for different imbalance levels where 11-MUX is an 11-bit version of the well-known, single-step multiplexer task. These boolean functions are defined for binary strings of length $l = k + 2^k$ under which the first k=3 bits index into the $2^k = 8$ remaining bits, returning the indexed bit [3]. We also ran XCS with seven UCI imbalance datasets [17] (table II) with the number in the parentheses indicates the target class we chose as the positive and all the other classes are regarded as negative. The complexity of 11-MUX problem can be tuned along to the imbalance level (i) between two classes where the ratio between the majority instances and minority instances is 2^i. Thus, level i=0 represents the balanced multiplexer. For level $i \ge 1$, there are half of the minority class instances with respect to i-1.

TABLE II: UCI DATASET DESCRIPTION

DATASET	(+) Inst.	(-) Inst.	ir	#Attr.
Echocardiogram	307	383	1:2	12
Breast-w	241	457	1:2	9
Wine3	268	500	1:3	8
Glass7	29	185	1:6	9
Vowel3	90	900	1:10	10
Hypothyroid1	93	3681	1:40	21
Abalone19	32	4145	1:130	8

(+) Inst. = number of positive instances, (-) Inst. = number of negative instances, *ir* = imbalance ratio, #Attr = number of attributes.

We ran XCS with the following parameter settings (as introduced in [3]): N=1000, β=0.2, α=0.1, ε_0=1, ν=5, θ_{GA}=25, χ=0.8, μ=0.04, θ_{del}=20, δ=0.1, θ_{sub}=200, $P_\#$=0.6.

Table III and table IV show performances of XCS on standard parameter settings, called standard XCS or XCS shortly to distinguish to our approach, Cost-sensitive XCS. All experiments presented are averaged over 10 independent runs of 10-fold cross-validation technique [1]. As expected, when imbalance level increases, the performance of XCS decreases. In 11-MUX problem, XCS's *F1-measure* raises to value 1 for imbalance levels up to i=4. For i=5, *F1-measure* is 0.8 and *F1-measure* is 0.2 for i=6. It means that for lower imbalance levels, XCS classifies correctly, for imbalance level i=5 or higher, XCS begins to find difficulties classifying the minority class examples. XCS also encounters this problem in UCI datasets as results shown in table IV. XCS's *F1-measure* reduces from 0.980 to 0 as imbalance ratio increases from 1:2 (*Echocardiogram* data) to 1:130 (*Abalone* data).

TABLE III: STANDARD XCS'S PERFORMANCE ON 11-MUX

i	p	r	F1	g
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	0.99±.01	1	0.99±.01	0.99±.01
5	0.8±.002	0.8±.001	0.8±.015	0.8±.015
6	0.2±.01	0.19±.02	0.2±.01	0.2±.01
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

(the values following "±" are standard deviation)

We analyzed possible reasons of this problem by looking at the classifier population of XCS, we found that when imbalance level is high, the population mainly consists of the two most *overgeneral* classifiers which contain only #'s symbols (don't care symbols) at their condition part. They cover accurately all the instances of the majority class and cover wrongly the instances of minority class, so, XCS has poor performance for the minority class or *F1-measure* is low. In the next section, we proposed a method based on reward setting to overcome this problem.

TABLE IV: STANDARD XCS'S PERFORMANCE ON UCI DATASETS

dataset	p	r	F1	g
Echocardiogram	0.980±.063	0.980±.063	0.978±.047	0.986±.034
Breast-w	0.931±.041	0.967±.042	0.948±.024	0.964±.020
Wine3	0.971±.090	0.748±.311	0.801±.230	0.831±.193
Glass7	0.900±.161	0.867±.172	0.873±.142	0.919±.099
Vowel3	0.673±.326	0.522±.119	0.590±.233	0.557±.073
Hypothyroid1	0.290±.418	0.217±.343	0.215±.301	0.269±.367
Abalone19	0±.00	0±.00	0±.00	0±.00

(the values following "±" are standard deviation)

6. Cost-sensitive XCS

6.1. Guidelines for reward setting

Accepted by FSKD'08, China, 18-20 October, 2008

XCS's fitness is based on accuracy, an inverse value of the classifier's average prediction error, which presents a high bias towards the majority class instances. In addition, XCS pays equally reward for correctly identifying of all classes, combined with the generalization tendency of the GA. This made XCS to evolve easily *overgeneral* classifiers that cover all the feature space as analysis in the previous section. Making difference of reward for each class could help in the reduction of *overgeneral* classifiers. The idea is to assign a greater reward to true positive classifying than to true negative classifying which will improve performance with respect to the positive (rare) class.

In XCS, the reward computation should be modified. Our suggestion is to set the proportion of true positive reward ($r_{true\_positive}$) and true negative reward ($r_{true\_negative}$) according to the proportion of frequencies between the occurring of negative instances (f_{neg}) and the occurring of positive instances (f_{pos}):

$$\frac{r_{true\_positive}}{r_{true\_negative}} = \frac{f_{neg}}{f_{pos}} = ir \qquad (6)$$

The ratio f_{neg}/f_{pos} tells us how many examples of the majority class are provided with respect to those of the minority class which is imbalance ratio (*ir*), so with 11-MUX problem, we can rewrite equation (6) as follows:

$$\frac{r_{true\_positive}}{r_{true\_negative}} = \frac{f_{neg}}{f_{pos}} = 2^i = ir \qquad (7)$$

If the imbalance ratio increases, we should also increase the proportion $r_{true\_positive}/r_{true\_negative}$. In addition, the population size should be assured that no niche will be loss (as introduced in [4]). It should be increased proportionally with the imbalance ratio *ir*.

TABLE V: COST-SENSITIVE XCS'S PERFORMANCE ON 11-MUX

i	p	r	F1	g
0	1	1	1	1
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	0.99±.010	0.99±.013	1
6	0.90±.013	0.91±.027	0.91±.014	0.91±.012
7	0.821±.040	0.833±.097	0.824±.062	0.827±.053
8	0.71±.121	0.712±.311	0.7±.155	0.69±.219
9	0.51±.092	0.512±.113	0.500±.015	0.49±.182

i = imbalance level, *p*=precision, *r*=recall, *F1* = F1 measure, *g* = g-mean.

We show the performance of Cost-sensitive XCS in table V and table VI. Experimental results reveal that Cost-sensitive XCS performs better than standard XCS on almost imbalance datasets. On 11-MUX problem, the proposed system can reach until i=8, which is a notable improvement with respect to the initial experiments in table III.

With imbalance datasets having small imbalance ratio as *Echocardiogram*, *Breast-w*, *Wine3* and $i \leq 4$ in 11-MUX, both standard XCS and Cost-sensitive XCS work well. On *Glass7* and i=5 of 11-MUX, performances of Cost-sensitive XCS are slightly better than that of standard XCS. On extremely imbalance datasets as *Hypothyroid1*, *Abalone19* and $i \geq 7$ of 11-MUX, performances of Cost-sensitive XCS are significantly better. Furthermore, the tending of evolving *overgeneral* rules has been restrained. The population evolved by Cost-sensitive XCS (not detailed for brevity) contains accurately and maximally general rules predicting both the minority class and majority class.

TABLE VI: COST-SENSITIVE XCS'S PERFORMANCE ON UCI DATASETS

dataset	p	r	F1	g
Echocardiogram	0.975±.109	1.00±.00	0.983±.062	0.984±.028
Breast-w	0.946±.029	0.971±.044	0.958±.021	0.970±.020
Wine3	0.931±.147	0.842±.295	0.839±.224	0.875±.185
Glass7	0.913±.274	0.900±.161	0.912±.224	0.924±.133
Vowel3	0.812±.147	0.822±.192	0.815±.101	0.817±.153
Hypothyroid1	0.654±.201	0.649±.193	0.655±.178	0.651±.166
Abalone19	0.400±.198	0.493±.271	0.419±.275	0.453±.203

6.2. Online Reward Adaptation

The provided guidelines assumed that the frequency of minority and majority classes is known. However, in real-world datasets containing class imbalances, the frequencies are unknown. To solve this problem, our approach is to use information collected during XCS's training. One way to estimate this is by averaging the proportion between instances of negative class and instances of positive class. Employing an idea from Q-learning of Reinforcement Learning [18], we averaged online imbalance ration as equation following:

$$ir_{online}(t+1) = ir_{online}(t) + \alpha * \left(\frac{n(-)}{n(+)} - ir_{online}(t) \right) \qquad (8)$$

Where $ir_{online}(0) = 1$, α=0.2 is step-size in Q-learning or learning rate in supervised learning; $n(-)$ and $n(+)$ are number of negative and positive instances gotten in each iteration, respectively.

We calculated ir_{online} by (8), reward calculation as equation (6) and we got results as same as table V and table VI, so we do not show them for brevity, although the training time is longer. That is because of Cost-sensitive XCS needing some time to realize the imbalance ratio of problems. The online reward adaptation simplifies Cost-sensitive XCS's tuning, since it does not require a previous estimation of the imbalance ratio, which is important for Cost-sensitive XCS's performance.

Accepted by FSKD'08, China, 18-20 October, 2008

7. Conclusions and Future Works

We have introduced a Cost-sensitive XCS learning classifier system to solve imbalance problems. We first analyzed effects of imbalance datasets to performances of XCS. The results showed that with standard parameter settings, XCS is quite robust to imbalance problems up to imbalance ratio *ir*=16 with 11-MUX and *ir*=6 with real datasets. For higher imbalance levels, XCS's parameters are needed to adapt to improve its performance. This is mainly due to the fact that XCS has a bias towards the majority class. We identified the presence of *overgeneral* rules predicting the majority class which covered almost all the feature space. The constrained reward function that we proposed makes it possible for XCS to address imbalance problems. Cost-sensitive XCS attempts not only to maximize the total reward of positive class in long runs but also to increase the performance of XCS.

As future works, we would like to study effects of injecting various degrees and types of noises to XCS's performance. Another important consideration has to do that is studying sampling techniques [16] in conjunction with XCS to enhance quality performance. We also would like to explore the parameter settings that control XCS. There are many parameters that could be adjusted in XCS. We think it is significant that XCS works well with the default parameter settings. However, it might be possible to substantially improve the performance of XCS with tuning some of these parameters. And from experiments, we can see that with the same parameter settings, XCS performs on 11-MUX datasets more robustly than on real datasets (imbalance ratio *ir*=16 comparing with *ir*=6). This is because of difference on overlapping and complexity of each dataset. We are currently investigating on this approach to analyze effects of imbalance problems in combining with other factors as small disjunction, overlapping degree and complexity of problem to performance of classifiers. Least but not last, we should compare performance of our system to several popular classification techniques such as: SMOTE [12], Support Vector Machine, Decision Tree [13], k-Nearest Neighbor.

8. References

[1]. P.N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison Wesley, USA, 2006.

[2]. P.L. Lanzi, W. Stolzmann, S.W. Wilson, Springer Berlin/Heidelberg, *Learning Classifier Systems: From Foundations to Applications*, 2000.

[3]. Stewart W. Wilson, "Classifier Fitness Based on Accuracy," *Evolutionary Computation*, Vol.3 (2), 1995, pp.149-175.

[4]. E. Bernadó-Mansilla, J.M. Garrell Guiu. "Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks". *Evolutionary Computation 11*, vol. 3, pp. 209-238, 2003.

[5]. Orriols-Puig A., Goldberg D.E., Sastry K., and Bernadó-Mansilla E. "Modeling XCS in Class Imbalances: Population Size and Parameter Settings". *Technical report, IlliGAl Report* No. 2007001, USA, Feb. 2007.

[6]. Albert Orriols and Ester Bernado-Mansilla, "Data Imbalance in UCS Classifier System: Fitness Adaptation", *IEEE Congress on Evolutionary Computation - CEC2005*, vol. 1, pp. 604-611, 2005.

[7]. C.Elkan, "The Foundations of Cost-Sensitive Learning" *Proceeding of the 17th International Joint Conference on Artificial Intelligence*, pp. 973-978, 2000.

[8]. K.M. Ting, "An Instance-weighting Method to Induce Cost-Sensitive Trees", *IEEE Trans. Knowledge and Data Eng.*, vol. 14, no.3, pp. 659-665, 2002.

[9]. Zhi-Hua Zhou and Xu-Ying Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem", *Knowledge and Data Engineering, IEEE Transactions*, vol. 18(1), pp. 63-77, January 2006.

[10]. N. Japkowicz, "Learning from Imbalanced Data Sets: A Comparison of Various Strategies," *Proc. Assoc. Advancement of Artificial Intelligence Workshop Learning from Imbalanced Data Sets (AAAI '00)* pp. 10-15, 2000.

[11]. Chris S., Taghi M.K., Jason V.H., Andres F., "An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data", *Information Reuse and Integration, 2007, IEEE International Conference*, pp. 651-658, August 2007.

[12]. Nitesh V.C., Kevin W.B., Lawrence O.H., W. Philip K., "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research* 16 pp. 321–357, 2002.

[13]. N. V. Chawla. "C4.5 and imbalanced datasets: Investigating the effect of sampling method, probabilistic estimate, and decision tree structure". *In Proceedings of the ICML '03 Workshop on Class Imbalances*, 2003.

[14]. Weiss, G. & Provost, F. "The Effect of Class Distribution on Classifier Learning: An Empirical Study", *Technical Report ML-TR-44, Department of Computer Science*, Rutgers University, 2001.

[15]. G.M. Weiss, "Mining with Rarity - Problems and Solutions: A Unifying Framework," *SIGKDD Explorations*, vol. 6, no. 1, pp. 7-19, 2004.

[16]. F.H.Gaohua Gu and H. Liu. "Sampling and Its Application in Data Mining: A survey". *Technical Report TRA6/00, National University of Singapore*, Singapore, 1999.

[17]. UCI Machine Learning Reposity http://archive.ics.uci.edu/ml/datasets.html.

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

[19]. Anand, R., Mehrotra, K.G., Mohan, C.K., Ranka, S.: An improved algorithm for neural network classification of imbalanced training sets. IEEE Transactions on Neural Networks 4, pp. 962 – 969, 1993.

Second Call for Papers

PRICAI 2008

Tenth Pacific Rim International Conference on Artificial Intelligence

Hanoi, Vietnam, 15-19 December 2008

The Pacific Rim International Conference on Artificial Intelligence (PRICAI) is a biennial international event which concentrates on AI theories, technologies and their applications in the areas of social and economic importance for countries in the Pacific Rim. In the past conferences have been held in Nagoya (1990), Seoul (1992), Beijing (1994), Cairns (1996), Singapore (1998), Melbourne (2000), Tokyo (2002), Auckland (2004) and Quilin (2006).

The Program Committee invites technical papers on substantial, original, and unpublished research in all aspects of Artificial Intelligence (AI). PRICAI aims to bring together a large and diverse community, which includes practitioners, researchers, educators, and users. The conference URL is http://www.jaist.ac.jp/PRICAI-08.

Conference Chairs

Hiroshi Motoda, AOARD/Osaka University, Japan
Bach Hung Khang, Vietnamese Academy of Science and Technology, Vietnam

Program Committee Chairs

Tu Bao Ho, Japan Advanced Institute of Science and Technology, Japan
Zhi-Hua Zhou, Nanjing University, China

Organizing Chairs

Tu Bao Ho, Japan Advanced Institute of Science and Technology, Japan
Nguyen Ngoc Binh, College of Technology, VNU-HN, Vietnam
Pham Hoang Luong, Hanoi University of Technology, Vietnam
Luong Chi Mai, Institute of Information Technology, VAST, Vietnam

Workshop Chairs

Duc Nghia Pham, NICTA/Griffith, Australia
Takashi Washio, Osaka University, Japan

Tutorial Chairs

Aditya K. Ghose, University of Wollongong, Australia
Tru Hoang Cao, Ho Chi Minh City University of Technology, Vietnam

Publicity Chair

Saori Kawasaki, Japan Advanced Institute of Science and Technology, Japan

Industrial Chair

Minh B. Do, Palo Alto Research Center, USA

For further information, please contact the Organizing Chair and/or PC Chairs

Tu Bao Ho, Japan Advanced Institute of Science and Technology (JAIST)
Phone & Fax : (81) 761-51-1730
Email: bao@jaist.ac.jp

Zhi-Hua Zhou, Nanjing University
Phone & Fax: (86) 25-8368-6268
Email: zhouzh@nju.edu.cn

Paper Submission

Proceedings will be published by Springer Verlag as a volume of LNAI series. PRICAI'08 will have a special issue in International Journal of Software and Informatics (IJSI, http://www.ijsi.org/) and is planning to have another special issue in Journal of Intelligence Data Analysis (IDA, http://www.iospress.nl/loadtop/load.php?isbn=1088467x). Authors are encouraged to use Springer's manuscript submission guidelines (http://www.springer.de/comp/lncs/authors.html). All papers must be submitted electronically in PDF format only, using the conference management tool. Detailed instructions will be available at the conference Website http://www.jaist.ac.jp/PRICAI-08. The submitted papers must not be published or under consideration to be published elsewhere.

Call for Workshops and Tutorials

The PRICAI'08 organizers invite workshop and tutorial proposals in the area of Artificial Intelligence in various topics. See conference Website for details.

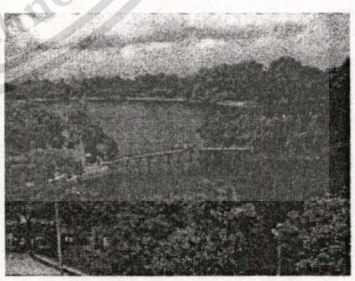
Important Dates

Paper Submission Deadline	26 May 2008
Paper acceptance notifications	25 July 2008
Camera Ready Papers & Copyright	15 August 2008
Registration and payment of author	15 August 2008
Workshop Proposals	6 June 2008
Workshop Notifications	30 June 2008
Tutorial Proposals	6 May 2008
Tutorial Notifications	30 June 2008
Conference	15-19 December 2008

Conference Location

The conference venue will be Hanoi University of Technology, located in the center of Hanoi (http://www1.hut.edu.vn/en/index.php?c2)

Welcome to Hanoi and Vietnam



Hanoi, the capital of Vietnam with a millennium-long history, has become one of the most beguiling cities in Asia. The city located in the Red River Delta in the center of North Vietnam has a lovely landscape of lakes, shaded boulevards, verdant public parks, colonial French houses and astounding modern skyscrapers. Hanoi is political and cultural center of Vietnam, gathered with the intellectual human resources of the country.

Towards Adapting XCS for Imbalance Problems

Thach Huy Nguyen[1], Sombut Foitong[2], Phaitoon Srinil[3], and Ouen Pinngern[4]

Department of Computer Engineering, Faculty of Engineering,
Research Center for Communication and Information Technology (ReCCIT)
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
{s9060654[1],s8060058[2],s8060020[3],kpouen[4]}@kmitl.ac.th

Abstract. The class imbalance problem has been recognized as a crucial problem in machine learning and data mining. Learning systems tend to be biased towards the majority class and thus have poor performance in classifying the minority class instances. This paper analyzes the imbalance problem in accuracy-based learning classifier system XCS. XCS has shown excellent performance on some data mining tasks, but as other classifiers, it also performs poorly on imbalance data problems. We analyze XCS's behavior on various imbalance levels and propose an appropriate parameter tuning to improve performance of the system. Particularly, XCS is adapted to eliminate *over-general* classifiers[1] and protect accurate classifiers of minority class. Experimental results in Boolean function problems show that, with proposal adaptations, XCS is robust to class imbalance.

Keywords: Classification, Learning Classifier System, XCS, Genetic Algorithm, Imbalance problem.

1 Introduction

Introduced in 1995 by Wilson, accuracy-based learning XCS is one of the most successful learning classifier systems [1]. XCS classifier system has recently shown a high degree of competence on a variety of data mining problems. There are some performance comparisons of strength-based fitness systems and accuracy-based fitness systems showing that accuracy-based fitness systems, including XCS, have a significant impact on data mining field. In [3], Mansilla and Garrel-Guiu studied performances of a set of well-known machine learning algorithms demonstrating that XCS exceeds the performance of most current Machine Learning techniques. In this paper, we study XCS in imbalance data problem.

In classification, data imbalance occurs when the number of patterns of a class is much larger than that of the other class [8]. Many real world data mining applications encounter how to deal with the imbalance problem, such as fraudulent credit card transactions identification, fault detection, target detection oil spill detection or medical diagnosis [8]. In classical machine learning or data mining setting, the classifiers that are designed to optimize overall predictive accuracy tend to produce high predictive

[1] *Over-general* classifier is a classifier or a rule covering more than one class.

T.-B. Ho and Z.-H. Zhou (Eds.): PRICAI 2008, LNAI 5351, pp. 1028–1033, 2008.

accuracy the majority class but ignore the minority class. This comes from two major causes. The first cause comes from the distribution of the classes. Since the number of majority class patterns exceeds that of the minority class, the majority class is likely to invade the territory the minority class so that the class boundary becomes vulnerable to be distorted. Second, the simple accuracy as an objective function used in most classification tasks is inadequate for the task having data imbalance. From that, there are two classes of methods proposed to solve imbalance problems: at the sampling level and at the classifier level. Methods at the sampling level aim at balancing the a priori probabilities of classes, while methods at the classifier level try to adapt the classifier to class imbalances, e.g. parameter adapting and cost sensitive learning.

This paper studies in the former approach by extending XCS to able to handle imbalance problems. Our study analysis effects of parameters to performance of XCS and provides guidelines to set them appropriately to solve imbalance problems. We restrict our analysis to Boolean classification problems because its characteristics could be controlled carefully rather than on real-world domains whose results would be difficult to decipher. The aim of this study is twofold: first, to make the XCS classifier system more robust on imbalance problems and second, to contribute to the understanding of the functioning of XCS in general to enable further improvement of XCS as well as to understand its limitations.

The rest of paper is organized as follows: XCS classifier system is described briefly in Section 2. In section 3, the proposed parameter adaptation, experimental results and comparison with original XCS will be presented to verify the feasibility of our research. Section 4 provides conclusions and future works.

2 Brief Description of XCS

As other reinforcement learning systems, after XCS sends an action to environment, a reward r is returned, which is then used to update the parameters of classifiers following order [2]: prediction p - an estimate of the payoff for that classifier if its condition matches to the input and its action is selected, prediction error ε - estimates the average error made in the predictions, and finally fitness F - an estimate of the accuracy of the payoff prediction given by p. Prediction p is updated with learning rate as follows:

$$p \leftarrow p + \beta(r - p) \tag{1}$$

where $\beta\ (0 \le \beta \le 1)$ is the learning rate. Next, the prediction error is updated as:

$$\varepsilon \leftarrow \varepsilon + \beta(|r - p| - \varepsilon) \tag{2}$$

Then, the classifier's accuracy is computed as an inverse function of the classifier's prediction error.

In equation (3) and Fig. 1, ε_0 determines the threshold error under which a classifier is considered to be accurate, i.e., it has an accuracy of 1; otherwise, the classifier accuracy κ drops off. α ($0<\alpha<1$) and ν ($\nu>0$) control the degree of decline in accuracy if the classifier is inaccurate [1]. Then, the relative accuracy over the action set is computed:

$$\kappa = \begin{cases} 1 & if \quad \varepsilon \le \varepsilon_0 \\ \alpha\left(\frac{\varepsilon}{\varepsilon_0}\right)^{-\nu}, & otherwise \end{cases} \qquad (3)$$

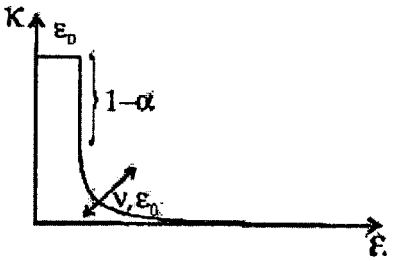


Fig. 1. The scaling of accuracy κ is crucial for successful selection of genetic algorithm

$$\kappa' = \frac{\kappa}{\sum_{cl \in [A]} \kappa_{cl}} \qquad (4)$$

where [A] is action set that consists of all the classifiers advocating the action sent to environment. And finally, classifier fitness is updated according to the classifier's relative accuracy:

$$F \leftarrow F + \beta(\kappa' - F) \qquad (5)$$

3 Adaptations of XCS for Imbalance Problems

In this section, we analysis effects of imbalance problems and our proposed parameter adaptation to performance of XCS. For this purpose, XCS was applied to version of the well-known multiplexer tasks.

3.1 The Imbalance Multiplexer Dataset Generation

The idea of multiplexer problem is from digital circuit. It is introduced to apply in classifier system by Wilson [1]. There are many versions of the multiplexer problem, each with a different size of the form $k + 2^k$, for $k \ge 1$. For example, in 11-Multiplexer problem, there are 11 inputs and one output (11 is of the form $3 + 2^3$). The first three inputs, a_0, a_1, and a_2, can be considered as address lines. They describe the binary representation of an integer between 0 and 7. This integer chooses one of the 7 remaining inputs, which are labeled d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7. The correct output for the multiplexer is the input on the line specified by the address lines. For instance, the value for the input string 01111110000 is 1.

The imbalance complexity is controlled by means of the probability of sampling an instance of the minority class P_{smin}. At each learning interaction, the environment chooses an instance randomly. If it is a minority class instance, it is passed to XCS with probability P_{smin}. If it is not accepted, a new instance is randomly sampled which undergoes the same decision process. In the remainder of this paper, we use the imbalance ratio *ir* to denote the ratio between the number of the majority and minority class instances that are given to XCS. Specially, the minority class is sampled with probability $P_{min}=1/(1+ir)$ and the majority class is sampled with probability $P_{maj}=ir/(1+ir)$. Another definition is also used that is imbalance level *i* where $i=log_2(ir)$.

3.2 Parameter ε_0, ν and α Dependence in Imbalance Problems

From Fig. 1, we can see that parameters ε_0, ν and α are very important to decide whether a classifies is accurate or not. In imbalance problem, these parameters should be adapted approximately then *over-general* classifiers are considered inaccurate and accurate classifiers are protected. The idea behind adapting these parameters is that in highly imbalanced datasets, an *over-general* classifier will often be correct more often than wrong. Any change in the classifier's condition will be reflected in a change in its statistical correctness – thus its error ε – and this will tend to guide the system toward accurate classifiers.

Parameter adaptation can be approached mathematically. Let us assume a two-level $R_{max}/0$ payoff landscape, where R_{max} is provided if the prediction is correct, 0, otherwise. According to [4] and [7], an *over-general* classifier would be considered inaccurate or its accuracy will fall down on the inaccurate part in Fig.1 as long as:

$$\frac{\varepsilon_0}{R_{\max}} * ir^2 + 2\left(\frac{\varepsilon_0}{R_{\max}} - 1\right) * ir + \frac{\varepsilon_0}{R_{\max}} \le 0 \tag{6}$$

Set $t = \varepsilon_0/R_{max}$, where ε_0 is quite small when compared with R_{max}, so we only need consider $0 \le t \le 1/2$.

From (6), we obtain the boundary of *ir* respected to changing in ε_0/R_{max}:

$$\frac{1-t-\sqrt{1-2t}}{t} \le ir \le \frac{1-t+\sqrt{1-2t}}{t} \tag{7}$$

Because:

$$\begin{cases} \lim\limits_{t\to 0^+} \dfrac{1-t-\sqrt{1-2t}}{t} = \lim\limits_{t\to 0^+} \dfrac{t}{1-t+\sqrt{1-2t}} = 0 \\ \lim\limits_{t\to 0^+} \dfrac{1-t+\sqrt{1-2t}}{t} = +\infty \end{cases} \tag{8}$$

So, if we decrease the value of $t = \varepsilon_0/R_{max}$, the boundary of parameter *ir* increases or XCS can solve problems with higher imbalance levels. Our proposal is to set ε_0/R_{max} based on imbalance ratio *ir*, $\varepsilon_0 = 1/ir$, other parameters are set as with standard XCS.

Additionally, in Fig.1, α causes a strong distinction between accurate and not quite accurate classifiers. The steepness of the succeeding slope is influenced by ν, so we should set α low enough and ν high enough. Moreover, population size should assure that no niche will be lost as suggested elsewhere [6], it should be increased proportionally with the imbalance rate *ir*, β and θ_{GA} should set with an appropriate window size as suggested in [7]. First, we run XCS on imbalanced multiplexer problems with imbalance level from $i=0$ to $i=10$ with the standard parameters setting (as introduced in [2]): N=1000, β=0.2, α=0.1, ε_0=1, ν=5, θ_{GA}=25, χ=0.8, μ=0.04, θ_{del}=20, δ=0.1, θ_{sub}=200, $P_\#$=0.6. Second, the result of our proposed method is gotten by setting: $\varepsilon_0 = 1/ir$.

Table 1 shows that the performance is improved significantly if appropriate ε_0, ν and α values are chosen. With standard parameters setting, XCS performs well up to imbalance level $i = 4$, it begins to encounter problem with $i = 5$ and $i = 6$. For higher imbalance levels, from $i = 7$ to $i = 10$, the systems classifies all the input instances as if they belonged to the majority class. In while, adaptive XCS can reach to 100% performance with $i \leq 7$; 95% with $i=8$; 80% with $i=9$ which is a notable improvement with respect to the performance of standard XCS. For the highest imbalance level, $i=10$ or $ir = 2^{10}=1024$, XCS can only classify correctly about 5% minority instances. Variant values of ε_0, ν and α were tested as $\nu=10$ and $\alpha=0.05$ show a better performance speed and stability in the imbalance problems. If we set $\nu \geq 10$ and $\alpha \leq 0.05$, we can get the results as same as Table 1 (not shown for brevity), but XCS needs a longer time in the beginning steps to evolve a population with accurate classifiers.

Table 1. Performance of Standard XCS and our proposal system in imbalance 11-MUX problem

Imbalance level	Standard XCS	Our proposal
$i=0$	100%	100%
$i=1$	100%	100%
$i=2$	100%	100%
$i=3$	100%	100%
$i=4$	100%	100%
$i=5$	70%	100%
$i=6$	20%	100%
$i=7$	0%	100%
$i=8$	0%	95%
$i=9$	0%	80%
$i=10$	0%	5%

4 Conclusions

We have introduced an XCS with adapting parameters to solve imbalance problems. We first analyzed effects of imbalance multiplexer problems to performances of XCS. The results show that with standard parameter settings, XCS is quite robust to imbalance problems up to imbalance ratio $ir=16$ ($i=4$) of multiplexer problem. For higher imbalance levels, XCS's parameters are needed to adapt to improve its performance. The parameter setting that we proposed makes it possible for XCS to address imbalance problems. It attempts to eliminate *over-general* classifiers and protect accuracy classifiers of minority class, so the performance of XCS is improved. The value of ε_0 guides the selection pressure within genetic algorithm since it represents the steepness of the fitness curve. Therefore, choosing the correct value of ε_0 gives the system the ability to create the required amount of pressure that can solve the problems used by [4] including the imbalanced ones. It also controls the GA's ability to cope with the low values for other parameters.

As future works, we would like to study effects of injecting various degrees and types of noises to XCS's performance. Another important consideration has to do that is studying sampling techniques [9] in conjunction with XCS to enhance quality

performance. A real world problem that we are investigating is intrusion detection where the problem is to realize decision boundaries between attacks and normal activities and highly imbalanced attack class distribution [10]. With the on-line learning probability, XCS is shown as a potential tool to solve intrusion detection.

References

1. Wilson, S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3(2), 149–175 (1995)
2. Butz, M., Wilson, S.: An algorithmic description of XCS. Journal of Soft Computing 6, 144–153 (2002)
3. Bernadó-Mansilla, E., Garrell Guiu, J.M.: Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. Evolutionary Computation 11 3, 209–238 (2003)
4. Butz, M., Kovacs, T., Lanzi, P.L., Stewart, W.: Toward a theory of generalization and learning in XCS. IEEE Trans. Evolutionary Computation 8(1), 28–46 (2004)
5. Kovacs, T.: XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In: Soft Computing in Engineering Design and Manufacturing, London, U.K, pp. 59–68. Springer, Heidelberg (1997)
6. Orriols-Puig, A., Goldberg, D.E., Sastry, K., Bernadó-Mansilla, E.: Modeling XCS in Class Imbalances: Population Size and Parameter Settings. Technical report, IlliGAl Report No. 2007001, USA (February 2007)
7. Orriols-Puig, A., Bernadó-Mansilla, E.: The class imbalance problem in learning classifier systems: a preliminary study. In: GECCO 2005, Washington, D.C, USA, pp. 74–78 (2005)
8. Chris, S., Taghi, M.K., Jason, V.H., Andres, F.: An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data. In: Information Reuse and Integration, IEEE International Conference, pp. 651–658 (August 2007)
9. Nitesh, V.C., Kevin, W.B., Lawrence, O.H., Philip, W.K.: SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research 16, 321–357 (2002)
10. Lee, W., Stolfo, S., Mok, K.: A data mining framework for building intrusion detection models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, Oakland, CA, pp. 120–132 (May 1999)