Universitat Ramon Llull

TESI DOCTORAL

Títol	**Facing Online Challenges Using Learning Classifier Systems**
Realitzada per	**Andres Sancho Asensio**
en el Centre	**Escola Técnica Superior d'Enginyeria Electrònica i Informàtica La Salle**
i en el Departament	**Informàtica**
Dirigida per	**Dra. Elisabet Golobardes i Ribé** **Dr. Jorge Casillas Barranquero**

Universitat Ramon Llull

Aquesta Tesi Doctoral ha estat defensada el dia ____ d __________________ de ____

al Centre ____

de la Universitat Ramon Llull

davant el Tribunal format pels Doctors sotasignants, havent obtingut la qualificació:

President/a

____

Vocal

____

Vocal

____

Vocal

____

Secretari/ària

____

Doctorand/a

C. Claravall, 1-3
08022 Barcelona
Tel. 936 022 200
Fax 936 022 249
E-mail: urlsc@sec.url.es
www.url.es

FACING ONLINE CHALLENGES USING LEARNING CLASSIFIER SYSTEMS

By

Andreu Sancho-Asensio

Ramon Llull University & Granada University
for the degree of
Doctor of Philosophy
Grup de Recerca en Sistemes Intel·ligents

Supervisors:
Dr. Jorge Casillas Barranquero
Dr. Elisabet Golobardes i Ribé

April 2014

Abstract

Last advances in machine learning have fostered the design of competent algorithms that are able to learn and extract novel and useful information from data. Recently, some of these techniques have been successfully applied to solve real-world problems in distinct technological, scientific and industrial areas; problems that were not possible to handle by the traditional engineering methodology of analysis either for their inherent complexity or by the huge volumes of data involved. Due to the initial success of these pioneers, current machine learning systems are facing problems with higher difficulties that hamper the learning process of such algorithms, promoting the interest of practitioners for designing systems that are able to scalably and efficiently tackle real-world problems.

One of the most appealing machine learning paradigms are *Learning Classifier Systems* (LCSs), and more specifically Michigan-style LCSs, an open framework that combines an apportionment of credit mechanism with a knowledge discovery technique inspired by biological processes to *evolve* their internal knowledge. In this regard, LCSs mimic human experts by making use of rule lists to choose the best action to a given problem situation, acquiring their knowledge through the experience. LCSs have been applied with relative success to a wide set of real-world problems such as cancer prediction or business support systems, among many others. Furthermore, on some of these areas LCSs have demonstrated a learning capacity that exceed those of human experts for that particular task.

The purpose of this thesis is to explore the online learning nature of Michigan-style LCSs for mining large amounts of data in the form of continuous, high speed and time-changing streams of information. Most often, extracting knowledge from these data is key, in order to gain a better understanding of the processes that the data are describing. Learning from these data poses new challenges to traditional machine learning techniques, which are not typically designed to deal with data in which concepts and noise levels may vary over time. The contribution of this thesis takes the *extended classifier system* (XCS), the most studied Michigan-style LCS and one of the most competent machine learning algorithms, as the starting point. Thus, the challenges addressed in this thesis are twofold: the first challenge is building a competent supervised system based on the guidance of Michigan-style LCSs that learns from data streams with a fast reaction capacity to changes in concept and noisy inputs. As many scientific and industrial applications generate vast amounts of unlabeled data, the second challenge is to apply the lessons learned in the previous issue to continue with the design of unsupervised Michigan-style LCSs that handle online problems without assuming any a priori structure in input data.

Resum

Els grans avenços en el camp de l'aprenentatge automàtic han resultat en el disseny de màquines competents que són capaces d'aprendre i d'extreure informació útil i original de l'experiència. Recentment, algunes d'aquestes tècniques d'aprenentatge s'han aplicat amb èxit per resoldre problemes del món real en àmbits tecnològics, mèdics, científics i industrials, els quals no es podien tractar amb tècniques convencionals d'anàlisi ja sigui per la seva complexitat o pel gran volum de dades a processar. Donat aquest èxit inicial, actualment els sistemes d'aprenentatge s'enfronten a problemes de complexitat més elevada, el que ha resultat en un augment de l'activitat investigadora entorn sistemes capaços d'afrontar nous problemes del món real eficientment i de manera escalable.

Una de les famílies d'algorismes més prometedores en l'aprenentatge automàtic són els sistemes classificadors basats en algorismes genètics (LCSs), el funcionament dels quals s'inspira en la natura. Els LCSs intenten representar les polítiques d'actuació d'experts humans amb un conjunt de regles que s'empren per escollir les millors accions a realitzar en tot moment. Així doncs, aquests sistemes aprenen polítiques d'actuació de manera incremental a mida que van adquirint experiència a través de la informació nova que se'ls va presentant durant el temps. Els LCSs s'han aplicat, amb èxit, a camps tan diversos com la predicció de càncer de pròstata o el suport a la inversió en borsa, entre altres. A més en alguns casos s'ha demostrat que els LCSs realitzen tasques superant la precisió dels éssers humans.

El propòsit d'aquesta tesi és explorar la naturalesa de l'aprenentatge *online* dels LCSs d'estil Michigan per a la mineria de grans quantitats de dades en forma de fluxos d'informació continus a alta velocitat i canviants en el temps. Molt sovint, l'extracció de coneixement a partir d'aquestes fonts de dades és clau per tal d'obtenir una millor comprensió dels processos que les dades estan descrivint. Així, aprendre d'aquestes dades planteja nous reptes a les tècniques tradicionals d'aprenentatge automàtic, les quals no estan dissenyades per tractar fluxos de dades continus i on els conceptes i els nivells de soroll poden variar amb el temps de forma arbitrària. La contribució de la present tesi pren l'*eXtended Classifier System* (XCS), el LCS d'estil Michigan més estudiat i un dels algoritmes d'aprenentatge automàtic més competents, com el punt de partida. D'aquesta manera els reptes abordats en aquesta tesi són dos: el primer desafiament és la construcció d'un sistema supervisat competent sobre el *framework* dels LCSs d'estil Michigan que aprèn dels fluxos de dades amb una capacitat de reacció ràpida als canvis de concepte i entrades amb soroll. Com moltes aplicacions científiques i industrials generen grans quantitats de dades sense etiquetar, el segon repte és aplicar les lliçons apreses per continuar amb el disseny de LCSs d'estil Michigan capaços de solventar problemes online sense assumir una estructura a priori en els dades d'entrada.

Resumen

Los grandes avances en el campo del aprendizaje automático han resultado en el diseño de máquinas capaces de aprender y de extraer información útil y original de la experiencia. Recientemente alguna de estas técnicas de aprendizaje se han aplicado con éxito para resolver problemas del mundo real en ámbitos tecnológicos, médicos, científicos e industriales, los cuales no se podían tratar con técnicas convencionales de análisis ya sea por su complejidad o por el gran volumen de datos a procesar. Dado este éxito inicial, los sistemas de aprendizaje automático se enfrentan actualmente a problemas de complejidad cada vez más elevada, lo que ha resultado en un aumento de la actividad investigadora en sistemas capaces de afrontar nuevos problemas del mundo real de manera eficiente y escalable.

Una de las familias más prometedoras dentro del aprendizaje automático son los sistemas clasificadores basados en algoritmos genéticos (LCSs), el funcionamiento de los cuales se inspira en la naturaleza. Los LCSs intentan representar las políticas de actuación de expertos humanos usando conjuntos de reglas que se emplean para escoger las mejores acciones a realizar en todo momento. Así pues estos sistemas aprenden políticas de actuación de manera incremental mientras van adquiriendo experiencia a través de la nueva información que se les va presentando. Los LCSs se han aplicado con éxito en campos tan diversos como en la predicción de cáncer de próstata o en sistemas de soporte de bolsa, entre otros. Además en algunos casos se ha demostrado que los LCSs realizan tareas superando la precisión de expertos humanos.

El propósito de la presente tesis es explorar la naturaleza *online* del aprendizaje empleado por los LCSs de estilo Michigan para la minería de grandes cantidades de datos en forma de flujos continuos de información a alta velocidad y cambiantes en el tiempo. La extracción del conocimiento a partir de estas fuentes de datos es clave para obtener una mejor comprensión de los procesos que se describen. Así, aprender de estos datos plantea nuevos retos a las técnicas tradicionales, las cuales no están diseñadas para tratar flujos de datos continuos y donde los conceptos y los niveles de ruido pueden variar en el tiempo de forma arbitraria. La contribución del la presente tesis toma el *eXtended Classifier System* (XCS), el LCS de tipo Michigan más estudiado y uno de los sistemas de aprendizaje automático más competentes, como punto de partida. De esta forma los retos abordados en esta tesis son dos: el primer desafío es la construcción de un sistema supervisado competente sobre el *framework* de los LCSs de estilo Michigan que aprende de flujos de datos con una capacidad de reacción rápida a los cambios de concepto y al ruido. Como muchas aplicaciones científicas e industriales generan grandes volúmenes de datos sin etiquetar, el segundo reto es aplicar las lecciones aprendidas para continuar con el diseño de nuevos LCSs de tipo Michigan capaces de solucionar problemas online sin asumir una estructura a priori en los datos de entrada.

Acknowledgements

The present thesis is the result of three years of painstaking work, and it would not have been possible without the guidance of many individuals who contributed in the completion of the herein presented dissertation.

First, I would like to thank Elisabet Golobardes and Jorge Casillas for their valuable support and guidance as supervisors. I owe my deepest gratitude and respect to Albert Orriols who introduced me to the exciting world of research. Definitely, without his help and patience I would not have come this far. Also, I am grateful to the *Research Group in Intelligent Systems*[1] (GRSI) and the *Soft Computing and Intelligent Information Systems*[2] (SCI^{2}S) group for letting me grow as a researcher.

I would like to show my gratitude to the *Interdisciplinary Computing and Complex Systems*[3] (I(CO)$_2$S) group for giving me the pleasure of visiting them, and specially to Jaume Bacardit for receiving me and for his valuable support and guidance.

The present work is the result of the collaboration with many researchers of distinct areas. In this regard I would like to thank Joan Navarro, Álvaro García, Germán Terrazas, Salvador García, María Martínez, Núria Macià, Nunzia Lopiccolo, Xaver Solé, Isaac Triguero, María Franco, Agustín Zaballos, Albert Fornells, José Enrique Armendáriz, José Antonio Moral, Xavier Vilasís, Rosa Sanz, Miquel Beltrán, Rubén Nicolás, Francesc Xavier Babot, Francesc Teixidó, Joan Camps, Xavier Canaleta, David Vernet, Joaquim Rios, Carles Garriga, and all the others I forgot—thank you!

Last, but not least, I would like to thank the unconditional support that all my family and friends have given me over these time. Specially, I want to thank my parents Andreu and María Isabel, and my brother Sergi for their unconditional support. Also, I would like to especially thank María José for her great support despite the distance.

The research done in this thesis is framed on the graduate program in Information Technology and Management from La Salle at Ramon Llull University. It has been developed in the GRSI, which is a research group created in 1994 and recognised as a consolidated research group by the Government of Catalonia since 2002 (2002-SGR-00155, 2005-SGR-00302, and 2009-SGR-183). During the development of my PhD thesis I have had the opportunity of relating my research to three projects, two national and one European, which are detailed in the following:

[1]http://salleurl.edu/GRSI
[2]http://sci2s.ugr.es
[3]http://icos.cs.nott.ac.uk

1. **KEEL-III: Knowledge Discovery based on Evolutionary Learning: Current Trends and New Challenges (TIN2008-06681-C06-05).** Focused on the knowledge extraction from data using evolutionary algorithms, KEEL III aims at (1) to continue with the development of the KEEL software tool, (2) to continue with the development of evolutionary learning models and/or their improvement and adaptation to specific contexts associated to the current trends on knowledge extraction based on evolutionary learning, (3) the development of studies on new challenges in knowledge extraction, and (4) the characterisation of specific real problems and the applicability of evolutionary learning algorithms.

2. **INTEGRIS: Intelligent Electrical Grid Sensor Communications (FP7-ICT-ENERGY-2009-1)**. INTEGRIS proposes the development of a novel and flexible ICT infrastructure based on a hybrid Power Line Communication-wireless integrated communications system able to completely and efficiently fulfil the communications requirements foreseen for the Smart Electricity Networks of the future.

3. **PATRICIA: Pain and Anxiety Treatment based on social Robot Interaction with Children to Improve pAtient experience (TIN2012-38416-C03-01)**. A major focus for children's quality of life programs in hospitals is improving their experiences during procedures. In anticipation of treatment, children may become anxious and during procedures pain appears. The challenge of the coordinated project is to design pioneering techniques based on the use of social robots to improve the patient experience by eliminating or minimising pain and anxiety. According to this proposed challenge, this research aims to design and develop specific human-social robot interaction with pet robots. Robot interactive behaviour will be designed based on modular skills using soft-computing paradigms.

My research has been cosupervised by Dr. Jorge Casillas and Dr. Elisabet Golobardes i Ribé, and it has been supported by the *Generalitat de Catalunya*, the commission for Universities and Research of the DIUE and European Social Fund under the FI grant (with references 2011FI_B 01028, 2012FI_B1 00158, and 2013FI_B2 00089). This thesis would not have been possible without the financial support of the *Departament d'Universitats, Recerca i Societat de la Informació* (DURSI) and the European Social Fund (ESF) under a scholarship in the FI research program.

Contents

List of Figures

List of Tables

1 Introduction

The field of machine learning has fostered the interest of scientists and engineers since its conception in the second half of the fifties of the last century. Deeply rooted in artificial intelligence, machine learning is concerned with the design and development of computer programs that learn from the past experience without being explicitly programmed for solving problems that are far too complex for human experts to unravel (Mitchell, 1997). In this regard, machine learning is very attractive for extracting useful information out of the ever increasing, massively collected data of many industrial and scientific applications. The main feature of this kind of environment lies in the fact that data are potentially unbounded in size since current real-world applications output continuous flows or streams of information with a fast arrival rate. Another feature is that its data distribution may change over time and hence a fixed distribution cannot be assumed. Also, as most of these data come from sensor networks, large and varying amounts of noise are expected. Therefore, the processing of such information requires the use of learning techniques with a high degree of *plasticity*.

To tackle the above-mentioned challenges, Nature has recurrently inspired practitioners to design and develop a large variety of learning algorithms: individual cell biomechanics, group behaviour or population genetics—just to mention a few—have been commonly used as analogies to obtain computer programs that solve the aforementioned challenges. One of these studies gave rise to LCSs as cognitive systems that received perceptions from their environment and, in response to these perceptions, performed actions to achieve certain goals (Holland, 1992, Orriols-Puig, 2008). The learning process of LCSs is guided by the general principles of Darwinian evolution (Darwin, 1859) and cognitive learning (Butz, 2006), and their goal is to provide human-readable rules that model the unknown structure of the problems faced by the system. Despite the fact that the family of algorithms that integrate LCSs have demonstrated to be mature, flexible and competitive machine learning techniques, the challenge of mining streams of time-changing information is just starting to being addressed by researchers (Orriols-Puig and Casillas, 2010b).

Therefore, the purpose of this thesis is to further investigate the online learning architecture of LCSs by analysing their behaviour when applied to data stream problems not in a single branch of machine learning, but covering distinct issues from different disciplines. In this chapter we present the framework of this thesis, detailing the dissertation scope to set the reader in the appropriate context to follow the presented work. Finally, we provide the road map of the thesis.

1.1 Thesis Scope

Strongly inspired by the general principles of Darwinian evolution and cognitive learning, Holland (1976) envisaged LCSs as cognitive systems that receive perceptions from an environment and perform actions to achieve goals, thus imitating human experts. In this regard, the very first LCS—the *cognitive system one* or CS-1 (Holland and Reitman, 1977)—was designed as a computer program capable of being aware of the environment it was embedded on and able to take decisions online that could affect such environment. Another important characteristic is that it automatically learns from the experience inferring patterns, behaviours or conclusions from data by modelling the unknown structure of the problems faced. As the pioneer CS-1 had, modern day LCSs are characterised by three fundamental aspects (Orriols-Puig, 2008): (1) a knowledge representation made up by individuals that enables the system to map sensorial states to actions, (2) an apportionment of credit mechanism which shares the credit obtained among individuals, and (3) a search algorithm, typically a genetic algorithm (GA (Holland, 1976)), to discover new promising individuals. GAs, developed as the core knowledge-discovery component of LCSs, are search methods based on the mechanics of natural selection and genetics, and they rely on two fundamental concepts: (1) survival of the fittest and (2) the genetic effects of selection and recombination.

Shortly after the appearance of CS-1, Smith (1980) took a workaround to design a new kind of LCS—the *learning system one* or LS-1—focused on evolving rule sets instead of individual rules, as opposed to Holland's idea: this way the complex rule sharing mechanism is avoided and therefore the learning process is hugely simplified, being the learning architecture much more similar to traditional offline GAs (Bacardit, 2004). Algorithms resulting of both approaches were distinguished, being the Holland's approach coined as Michigan-style LCSs, whilst the Smith's approach labeled as Pittsburgh-style LCSs (Orriols-Puig, 2008).

Although the early success of these pioneers, no complete learning theory was developed for LCSs (Butz, 2004): neither learning nor convergence could be assured and the learning interactions of primeval LCSs appeared to be too complex and therefore these were not well-understood. It was not until the mid-1990s, with the conception of XCS (Wilson, 1995), that the interest in the field was renewed. XCS simplifies the learning architecture of primeval LCSs while being (1) accurate and (2) robust—i.e., XCS is based on a general-purpose framework that performs well in any type of problem. Shortly after its appearance, several theoretical analyses of XCS were published, resulting in the renaissance of the field (Butz, 2004). Consequently, XCS has become the standard framework for Michigan-style LCSs. So far, this framework has demonstrated a competitive behaviour in several applications, being at the same level or even surpassing the most well-studied machine learning techniques (Orriols-Puig, 2008).

Machine learning techniques are specially practical when the problem to solve is too complex for traditional engineering methods, when the problem requires a high degree of adaptation to changing environments, and when requiring the processing of huge amounts of data to extract useful information. As reported by Aggarwal (2007), Angelov (2012), Gama (2010) and others, today's data of most industrial and scientific applications are generated online in the form of dynamic streams of information. This issue has rocketed the attention of machine learning practitioners, leading to the emergency of the field of data streams. Differently from traditional data mining processes, data streams come in continuous flows of information and posses the following characteristics: (1) these flows of data are potentially unbounded in size and therefore there are strong constraints in memory usage, (2) these data can only be read in a single pass (i.e., one *epoch*) due to its large size and fast arrival rate, and (3) as these data are inherently dynamic and consequently a fixed data distribution cannot be assumed. This dynamic nature is the distinctive trait of data streams, hence hampering the learning process of traditional machine learning algorithms, which are ill-suited for the task. Another important issue of data streams is that, as data come mostly from sensor networks, large and varying amounts of noise are present, therefore algorithms that are robust against noise are required in order to mine from real-world scenarios.

In this regard, the scope of the present work is focused on Michigan-style LCSs as a well-suited framework for mining data streams. Even if this framework has demonstrated to be mature, flexible and competitive, the problem of mining streams of time-changing information is just starting to being addressed by practitioners (Orriols-Puig and Casillas, 2010b). The purpose of this thesis is to go beyond this pioneering exploratory study and further investigate the online learning architecture of LCSs by analysing their behaviour when applied to data stream problems not in a single branch of machine learning (e.g., *label prediction*), but covering distinct issues from different disciplines. In the subsequent sections the thesis objectives and the roadmap followed in this work are detailed.

1.2 Thesis Objectives and Contributions

The fundamental objective of the present thesis is to explore the online learning nature of the Michigan-style LCS architecture for mining data streams without limiting our analysis to a single learning paradigm nor technique (i.e., *supervised* or *unsupervised* learning; *classification* or *clustering*), but covering a broader, global vision. As there is no recipe to guide practitioners in order to tackle a new, previously unsolved real-world problem, having a broader vision is of the upmost importance. Regarding this issue, some problems require the application of supervised techniques whereas some others the unsupervised paradigm. Further, often a mixture of both learning strategies may lead researchers towards their goal.

Setting the mature and open framework of XCS, by far the most well-studied and influenced Michigan-style LCS (Orriols-Puig, 2008), as a departure point we propose to:

1. Revise and improve the characteristics of Michigan-style LCSs for supervised learning in data stream classification tasks.

2. Revise, extend and improve the characteristics of Michigan-style LCSs for unsupervised learning in clustering data streams.

3. Explore and enrich the characteristics of Michigan-style LCSs for unsupervised learning by introducing association streams.

A more detailed discussion for each one of the three objectives is provided in the following.

Revise and improve the characteristics of Michigan-style LCSs for supervised learning in data stream classification tasks. In spite of the ever-increasing interest of researchers in obtaining novel and useful knowledge out of data streams little research has been done using Michigan-style LCSs (Abbass et al., 2004, Orriols-Puig and Casillas, 2010b). These studies suggest that, although XCS and its derivatives—specifically the *supervised classifier system*, UCS (Bernadó-Mansilla and Garrell-Guiu, 2003, Orriols-Puig and Bernadó-Mansilla, 2008)—are able to tackle such problems satisfactorily they suffer from requiring a huge population to obtain accurate results and also their responsiveness to a sudden concept drift is not competitive enough under high dimensional spaces. This issue is mainly caused by the overlapping rule-based representation these algorithms use, which is not the best individual representation for these kinds of problems as preliminary studies suggest. Therefore, this thesis starts by identifying the common difficulties that hamper the learning phase of supervised techniques when applied to data stream problems with the aim of obtaining a very robust system with a high reaction capacity to concept changes and noisy inputs, even in problems with high dimensional variables.

Revise, extend and improve the characteristics of Michigan-style LCSs for unsupervised learning in clustering data streams. The first objective shows that Michigan-style LCSs are indeed a competitive choice for mining prediction problems when applied to data streams—i.e., supervised tasks. However, many real-world problems require unsupervised learning schemes due to the dearth of training examples and the associated costs of labelling the information to train the system. Moreover, an interesting application of unsupervised learning that has received a special attention of the community is clustering data streams (Bifet et al., 2010, Gama, 2012). Therefore, in this second objective we propose the study and application of the Michigan-style LCSs framework for clustering data streams by means of a specialisation of XCS: the *extended classifier system for clustering*, XCSc (Shi et al., 2011, Tamee et al., 2007a). So far, XCSc has not been applied to online problems because of its internal architecture. Thus, we propose a revision of this learning architecture to handle the strict requirements of clustering data streams. Also, we deploy this enhanced version of the algorithm to tackle a real-world scenario.

Explore tand enrich the characteristics of Michigan-style LCSs for unsupervised learning by introducing association streams. Current trends in knowledge discovery are fostering practitioners to extract potentially useful knowledge out of unlabelled data composed of continuous features that flow over time while providing a high degree of interpretability (Orriols-Puig and Casillas, 2010a, Orriols-Puig et al., 2012). Sound examples of this kinds of environments are found in Smart Grids and in network security monitoring, where it is of the utmost importance to detect intrusion attacks when these are happening without assuming any *a priori* underlying structure. A realistic response to this challenge is in association stream mining, a field closely related to frequent pattern mining, but that differently to this latter is focused on obtaining rules directly out of the streams and adapting to concept drift in a pure online fashion—thus it does not allow to use a classic offline process for rule generation. Due to its relevance, in the last objective of this thesis we explore the new field of association streams by means of the Michigan-style LCS architecture. Furthermore, we propose ways for tackling these kinds of problems. Also, with the goal of having a high interpretability, we explore and enrich the crossbreeding of fuzzy logic with LCSs.

In addition, we take the first steps towards a theory of generalisation and learning for Fuzzy-CSar departing from the existing formalisms of XCS and its related algorithms. These resulting models of the analysis provide several configuration recommendations for applying Fuzzy-CSar.

Figure 1.1 depicts the distinct representations explored with Michigan-style LCSs in this thesis: the highly-flexible, highly-accurate and non-human readable neural network, the accurate and human-friendly clustering rule and finally the linguistic fuzzy association rule, the most readable of them all. At this point it is important to highlight that as we approach to more real data stream problems, we improve the readability of the representation used. Also, we refine the applicability of our framework: as supervised learners are a replacement for human operators in daunting tasks, their applicability in real environments is limited as we will discuss in later chapters. On the other side of the spectrum, unsupervised learners are support tools designed for helping experts. Consequently, readability is a desirable feature in the unsupervised field. Figure 1.2 conceptualises the aforementioned applicability versus readability of the techniques presented in this thesis in a purely qualitative way.

1.3 Roadmap

In this section we detail the overall structure of the present thesis, which is composed of, in addition to the present chapter, seven chapters plus a complementing appendix.

Chapter 2 provides the required theoretical background to follow the present work. It starts with a brief introduction to machine learning, starting with a formal definition and providing a taxonomy of the three main branches of learning of algorithms. Afterwards, evolutionary computation—the core field of the learning algorithms of this thesis—is described in detail providing the theoretical background on designing competent algorithms. Finally, a survey on LCSs is presented which provides to the reader the big picture.

Chapter 3 reviews the Michigan-style LCS framework by concisely detailing the XCS classifier system. It departs from an overview of the system, introduces the knowledge representation used by this system, details the learning organisation, shows its action inference, and briefly details the theoretical framework on which XCS (and hence every Michigan-style LCS) sustains. Also, UCS, a supervised extension of XCS, is described in detail.

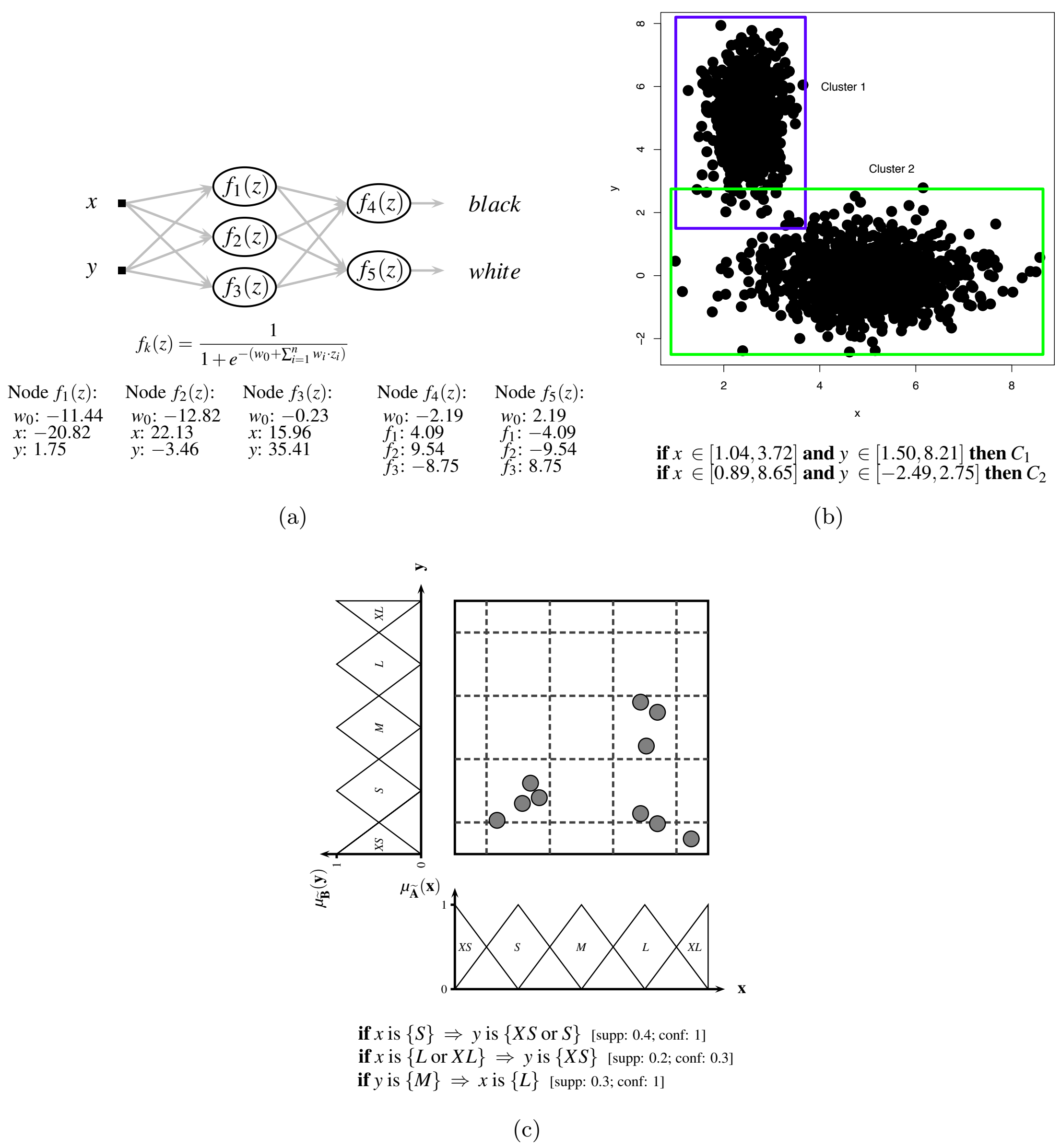

x
y
$f_1(z)$
$f_2(z)$
$f_3(z)$
$f_4(z)$
$f_5(z)$
black
white
$f_k(z) = \frac{1}{1+e^{-(w_0+\sum_{i=1}^{n} w_i \cdot z_i)}}$
Node $f_1(z)$: w_0: −11.44 x: −20.82 y: 1.75
Node $f_2(z)$: w_0: −12.82 x: 22.13 y: −3.46
Node $f_3(z)$: w_0: −0.23 x: 15.96 y: 35.41
Node $f_4(z)$: w_0: −2.19 f_1: 4.09 f_2: 9.54 f_3: −8.75
Node $f_5(z)$: w_0: 2.19 f_1: −4.09 f_2: −9.54 f_3: 8.75
(a)
Cluster 1
Cluster 2
x
y
if $x \in [1.04, 3.72]$ **and** $y \in [1.50, 8.21]$ **then** C_1
if $x \in [0.89, 8.65]$ **and** $y \in [-2.49, 2.75]$ **then** C_2
(b)
XS S M L XL
$\mu_{\tilde{A}}(\mathbf{x})$
$\mu_{\tilde{B}}(\mathbf{y})$
if x is $\{S\} \Rightarrow y$ is $\{XS$ or $S\}$ [supp: 0.4; conf: 1]
if x is $\{L$ or $XL\} \Rightarrow y$ is $\{XS\}$ [supp: 0.2; conf: 0.3]
if y is $\{M\} \Rightarrow x$ is $\{L\}$ [supp: 0.3; conf: 1]
(c)

FIGURE 1.1: The distinct representations explored in this thesis: (a) a multilayer perceptron for data classification in the *tao* problem, (b) two clustering rules in a two-dimensional problem, and (c) three fuzzy association rules in a two-dimensional problem using five fuzzy sets per variable.

Chapter 4 starts detailing the challenges of learning when applied to data streams in supervised tasks and the difficulties that XCS, the referent Michigan-style LCS, has at handling these environments. This chapter presents the supervised neural constructivist system (SNCS), a brand new member of the Michigan-style LCS family that is specifically designed to learn from data streams with a fast reaction capacity to concept changes and a remarkably robustness to noisy inputs. The behaviour of SNCS on data stream problems with different characteristics is carefully analysed and compared with other state-of-the-art techniques in the field. Furthermore, SNCS is also compared with XCS and UCS using the same testbed to highlight their differences. This comparison is extended to a large

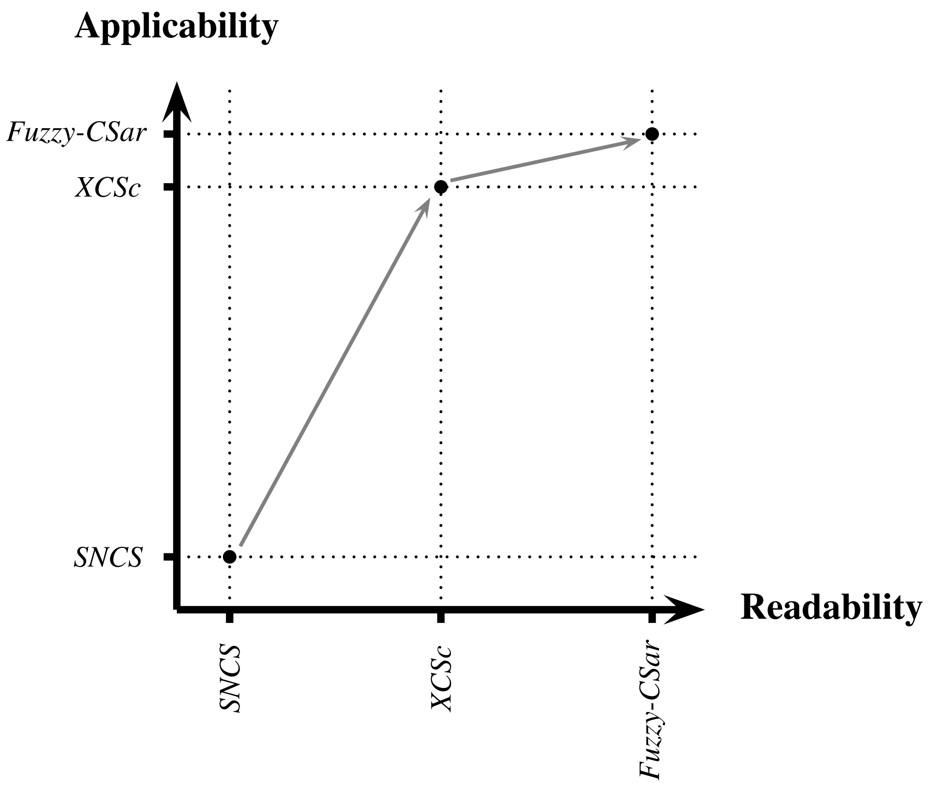


Figure 1.2: Conceptual chart displaying the applicability of the studied algorithms to real-world data stream problems versus readability of the representation used. Notice that the metric is not to scale.

collection of real-world problems.

Chapter 5 exploits the Michigan-style LCS framework beyond supervised tasks when applied to data streams. Despite the good results obtained in the supervised field, these techniques assume an *a priori* underlying structure for the set of features of the problem: supervised techniques require the class of each training example in order to build a reliable model. This issue is often unrealistic, specially in real-world problems, making the direct application of pure supervised learners ill-suited. With this issue in mind this chapter is dedicated to improve XCSc to face the challenges of clustering data streams.

Chapter 6 presents the new field of association streams, devoted to modelling dynamically complex domains via production rules without assuming any *a priori* structure. This chapter discusses the challenges that association streams poses to learning algorithms and presents *fuzzy-classifier system for association rules* (Fuzzy-CSar), an algorithm that inherits the main architecture of XCS and Fuzzy-UCS for extracting useful knowledge out of continuous streams of unlabelled data.

Chapter 7 goes further and takes the first steps towards a theory of generalisation and learning of Fuzzy-CSar departing from the existing formalisms of XCS and its related algorithms. Also, the lessons learned from this analysis result in several configuration recommendations for applying this algorithm on any type of problems.

Chapter 8 recapitulates the contributions of this thesis by summarising, providing key conclusions, and presenting a proposal of the future work lines.

The material presented in the distinct chapters is complemented with an appendix (A) that details the required background on the statistic tests employed in the different chapters of this work.

2 Theoretical Background

Over the past decades the study of machine learning has grown from the early efforts of engineers exploring whether computers could learn to perform some *intelligent* actions to a broad discipline (Mitchell, 2006). This discipline, strongly related to artificial intelligence and statistics, is concerned with the development of computer programs that can learn from the past experience. This chapter provides all the necessary background to understand the theoretical foundations described in this thesis, starting with the formal definition of machine learning, arguing why is such an interesting field, and then showing the classical taxonomy, i.e., supervised learning, unsupervised learning and reinforcement learning. Following that, we further classify machine learning into offline and online learning, where this latter leads to the field of data streams. Afterwards, this chapter presents the core field of the thesis: Evolutive Computation (EC), a family of learning algorithms inspired in natural selection and genetics. Next, the theoretical foundations on which this family is based and the background on designing competent EC algorithms are described. Finally, the two main models of EC learning systems—Michigan-style and Pittsburgh-style LCSs—and its derivatives are detailed showing their differences.

2.1 Machine Learning, a Brief Tour

Machine Learning is a field of computer science closely related to AI that is concerned with the development of computer programs that *learn* from data obtained, for example, from input sensors or databases, and that automatically *improve* its results with the *experience* gained when manipulating these data. Mitchell (1997) proposed a more precise and formal definition of machine learning: *"a computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at task T, as measured by P, improves with experience E."*

So, in general, to have a well-defined learning problem, the algorithm designer have to identify three key features: (1) the class of the task to be performed by the machine, (2) the measure of performance to be improved, (3) and the source of experience. For example, a computer program that learns to drive a robot through a maze can be formally defined as follows:

- **Task *T***: driving a robot through the maze.
- **Performance measure *P***: average distance traveled before an error occurs.
- **Training experience *E***: a set of sensor data classified by an expert.

There are a vast number of useful applications of machine learning in distinct fields such as search engines, medical diagnosis, pattern recognition or recommender systems, just to mention a few. As mentioned by Orriols-Puig (2008), the most important reasons that may lead to the application of machine learning to solve problems are enumerated as follows:

1. Problems that are too complex to manually design and code an algorithm to solve it. This is the case of computer vision: it turns out that it is very complex to write an algorithm for recognising human faces using traditional software engineering tools.
2. Necessity of programs that continuously adapt to changing environments. This is the case of robot control, for instance the Mars Science Laboratory *Curiosity*[1].
3. Necessity of processing huge amounts of data to extract novel, interesting and useful knowledge from patterns hidden in these data. This is the objective, in fact, of *data mining* (Witten et al., 2011).

From the classic point of view, machine learning algorithms are classified based on the task to perform. In general, there are three fundamental types of learning, which are: *supervised learning*, where an expert or teacher provides feedback in the learning process, *unsupervised learning*, where there is no expert or teacher when the learning process is running, and *reinforcement learning*, where the program learns interacting with the environment. Also, these families of learners can be further classified into offline and online methods, depending on the strategy followed by the algorithm. These concepts are elaborated in more detail in what follows.

2.1.1 Supervised Learning

Supervised learning consists in the process for extracting a *function* or *model* from training data. These training data is composed of a set of input features—often also called attributes—and the desired output, as is shown in Table 2.1. The main characteristic of the supervised learning approach is that the computer program needs an expert or teacher that provides feedback in the learning process. The supervised paradigm assumes an *a priori* underlying structure and thus require the use of existing information to obtain its knowledge (i.e., the learner uses the output of the training data as a guidance).

[1]More information available at `http://mars.jpl.nasa.gov/msl`

Orriols-Puig (2008) gives a more precise definition: *"supervised learning is the process of extracting a function or model that maps the relation between a set of descriptive input attributes and one or several output attributes."*

outlook	**temperature**	**humidity**	**windy**	**play tennis?**
sunny	85	85	false	**no**
sunny	80	90	true	**no**
overcast	83	86	false	**yes**
rainy	70	96	false	**yes**
rainy	68	80	false	**yes**
rainy	65	70	true	**no**
overcast	64	65	true	**yes**
sunny	72	95	false	**no**
sunny	69	70	false	**yes**
rainy	75	80	false	**yes**
sunny	75	70	true	**yes**
overcast	72	90	true	**yes**
overcast	81	75	false	**yes**
rainy	71	91	true	**no**

Table 2.1: Training data of the *weather problem* taken from the UCI machine learning repository (Bache and Lichman, 2013). We can see the input features *outlook, temperature, humidity* and *windy*, and the desired output *play tennis?*

We can further classify supervised learning depending on the type of the output features as *data classification* or *data regression.* In data classification the goal is to find a model that predicts the *class* (that is, the output feature) of new input instances not previously seen by the algorithm during the training phase. These output features are said to be *categorical* (i.e., the output represents the classes of the examples). Table 2.1 shows a classic example of a data classification problem. In data regression, the goal is to find a function that predicts the output value of new and previously unseen input instances, that is, modeling a function. In this case, the output features are said to be *continuous.*

2.1.2 Unsupervised Learning

Unsupervised learning encompasses a range of techniques for extracting a *representation* from data—these data consist only of input features—thus these techniques do not assume any *a priori* underlying structure in data (i.e., the learner does not know what it is looking for and, hence, has no feedback from the environment).

The kind of problems that unsupervised learning approaches handle are those in which it is required to determine how the data are organised. The main unsupervised techniques are *clustering*, *dimensionality reduction* and *association rules.* The objective of clustering is to separate a finite unlabelled set into a discrete finite set of hidden data structures. The aim of dimensionality reduction is to find a subset of the input space without any loss of information, that is, to reduce the number of input features of the problem. Association rules are methods for discovering interesting relations hidden among the variables of the problem.

2.1.3 Reinforcement Learning

Reinforcement learning is a family of techniques that envisage learning *what to do* by interacting with the environment. In this interaction the agent[2] receives what is called *perceptions* from the environment and performs actions with the aim of achieving one or several goals. The agent receives, then, positive—or negative—rewards as a consequence of its actions. Sutton and Barto (1998) describe reinforcement learning as the problem faced by an agent that must learn behaviour through trial-and-error interactions with a dynamic environment.

Sutton and Barto (1998) define reinforcement learning somewhat more formally: *"reinforcement learning is how to map situations to actions so as to maximise a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and, though that, all subsequent rewards."*

In this regard, reinforcement learning lies between supervised and unsupervised learning, and thus the agent has not any kind of examples to learn from, so it must be able to learn from its own experience. The agent has to *exploit* what it already knows in order to obtain a reward, but also has to *explore* in order to make better action selections in the future.

2.1.4 Offline and Online Learning

So far we presented the classic taxonomy that identifies the three types of learning. Moreover, the aforementioned families can be further classified into offline and online methods. Offline algorithms require all the data to be analysed in order to build a comprehensive system model and infer any kind of knowledge from it, whether we are in the supervised, unsupervised or reinforcement learning paradigm. On the other hand, online learning approaches build a dynamic system model that attempts to adapt itself to the environment specificities.

Offline techniques can be exported to online environments by means of windowing techniques. It this manner, the algorithm can continuously train and thus get nearly-online results.

2.2 Data Streams

In our Information Age, data of most industrial and scientific applications are generated online and collected continuously and massively, in which a static model cannot be assumed. Moreover, these data present the following characteristics (Angelov, 2012, Gama, 2012, 2010, Gama and Gaber, 2007, Lughofer and Angelov, 2011, Núñez et al., 2007):

- Potentially unbounded in size.
- Limited usage of memory.
- Data can only be handled once.
- Fast arrival rate.

[2] In reinforcement learning literature the learner is called an *agent*.

- Continuous flow of information.
- Target concepts may vary over time (*concept drifts*).
- Noise levels may change.
- A fixed data distribution cannot be assumed.

The field that monitors, tracks and controls such data under these constrains is coined as *data streams* and has attracted the attention of machine learning practitioners in recent years. One of the main challenges that present data streams is the dynamic nature of the target concept inside data that may change over time (referred to as concept drift), which makes the learning process difficult. Typically, concept drifts are classified as abrupt, incremental and recurring (Gama, 2010), which characterize the type of change inside the data. Also, most often examples can be seen only once, hence limiting the accuracy of the models generated by the learners. Some real-world examples of data streams can be found in sensor monitoring, Smart Grids, stock market analysis, web click-stream analysis, road traffic congestion analysis, market basket mining or credit card fraud detection, among many others (Orriols-Puig and Casillas, 2010b).

So far we presented the classic taxonomy of machine learning based on the task they do. Distinct machine learning techniques have been developed to perform some of the aforementioned tasks. One of the most successful approaches to face the distinct types of problems in machine learning are *learning classifier systems* (Holland, 1971, 1976, 1992, Holland et al., 1999), originally designed for reinforcement learning tasks (Holland and Reitman, 1977). Since then, the LCSs family has been extended to deal with supervised (Bacardit and Butz, 2007, Bernadó-Mansilla and Garrell-Guiu, 2003) and unsupervised (Orriols-Puig and Casillas, 2010a, Tamee et al., 2007a) learning tasks, hence rendering as a flexible and trustworthy learning architecture (Orriols-Puig, 2008). The remainder of this chapter is focused on *evolutionary computation*, the core set of techniques used by most LCSs to discover new knowledge and, thus, to LCSs themselves.

2.3 Nature-inspired Learning Algorithms

Strongly inspired by the way Nature solves complex problems, Evolutionary Computation (EC) is a field of study that solve problems using procedures inspired by natural processes (Bacardit, 2004). EC does not refer to a single type of learning algorithm, but a series of distinct techniques that share the same principles of natural selection and evolution for competent problem solving (Orriols-Puig, 2008).

In Nature, all living entities are made of cells, and each contains *chromosomes*—strings of DNA—that define the *blueprint* for the organism. A chromosome is divided into *genes*, the functional blocks of DNA, each of which encodes a particular protein (Mitchell, 1998). In order to understand the principles that guide Nature, two fundamental concepts have to be defined: the concept of *genotype* and the concept of *phenotype*, which are elaborated in the following.

Genotype. It is defined as the gene—or set of genes—responsible for a particular trait. That is, the information within the genes. For example the gene responsible for the eye colour.

Phenotype. It is defined as the expression of a particular trait, and it is influenced by the environment in which the organism lives. The eye colour of a particular individual is an example of phenotype.

Evolution, the *force* that drives Nature, can be defined as the change in the inherited characteristics of populations of living entities over successive generations (Hall and Hallgrímason, 2008). There are two key mechanisms that enforce evolution (Futuyma, 2005): (1) the process of natural selection (Darwin, 1859, Mendel, 1865) and (2) genetic drift. The former results in a process that generates the traits that are heritable and useful for surviving and thus reproducing. Over many generations, new adaptations flourish as combinations of many small successive random changes in these traits, and natural selection *favors* those that are best adapted to their environments. The latter consist of a process that causes random changes in the frequency of an inheritable trait in a population. These changes are small but steady, causing important changes in organisms after several generations.

These ideas were taken as inspiration by different researchers as a way of solving complex optimisation problems. Consequently, several authors started their ways on designing optimisation methods that simulate different aspects of evolution, which have been grouped under the EC term (Orriols-Puig, 2008). Freitas (2002) discusses a taxonomy of these methods. These are classified in four main families: *genetic algorithms* (Holland, 1971), *evolution strategies* (Rechenberg, 1973, Schwefel, 1981), *evolutionary programming* (Fogel et al., 1966) and *genetic programming* (Koza, 1992). From these families, genetic algorithms (GAs) are the most fundamental for this thesis. Due to their importance, the next section explains GAs in detail. The reader is referred to (Freitas, 2002) for a comprehensive review of the other families.

2.3.1 Genetic Algorithms

Originally ideated by Holland (1971, 1976) with the aim of understanding the underlying principles of adaptive systems, and later extended by Goldberg (1989), GAs are a family of techniques that use mechanisms inspired by biological evolution to approximate solutions to the problem of learning.

As pointed out by Orriols-Puig (2008), GAs differ from other optimisation techniques in the following:

- They learn from the objective function without assuming any structure or underlying distribution.
- They search from a population of candidate solutions.
- They code potential solutions instead of directly tuning the decision variables of the problem.
- They use stochastic, local operators instead of deterministic, global rules.

Similarly to the occurring natural processes, GAs evolve a *population* of *individuals*, where each individual in the population represents a potential solution to the problem to be solved. Individuals are represented by *chromosomes*, which encode the variables of the problem with a finite-length string. Each of the basic constituents of the chromosome is referred to as *genes*, and the values that the gene can take are addressed as *alleles* (Bacardit, 2004, Orriols-Puig, 2008).

Natural selection is simulated by incorporating an *evaluation function* that is responsible for assessing the degree of adaptation of each solution. This quality term is made explicit with a *fitness* value that is given to the individual.

GAs are a search and optimisation family of techniques that use evolutionary operators such as inheritance, mutation, selection and crossover to find solutions. Figure 2.1 depicts this process graphically. The core idea is that a group of individuals reproduce themselves forming new individuals with crossed characteristics so that the diversity of species grows and obtain a solution by searching the problem space through successive generations. Fitter individuals (i.e., with a fitness counting above the average) will generate more descendants. The inheritance mechanism will keep fit solutions through successive generations while mutation and crossover will explore new ones. This process is repeated until the stop criteria is met. Thus, the aforesaid evolutionary operators play a key role in GAs. These are further elaborated:

- **Selection:** This process chooses the fittest individuals in the population for, later in the GA cycle, mating. It simulates the *survival of the fittest* mechanism found in Nature. Several selection mechanisms exist, but the most common ones are (Freitas, 2002): *fitness-proportionate selection* (also known as *roulette-wheel*) (Goldberg, 1989, Holland, 1976), *tournament selection* (Goldberg, 1989) and *ranking selection* (Mühlenbein and Schlierkamp-Voosen, 1993).

- **Crossover:** The crossover operator is responsible for the process of reproduction between the selected parents. This operator combines the genetic information of two or more parents to create new offspring. Recombination is a key element in GAs, since it should detect important traits of parental solutions and exchange them with the aim of generating fittest individuals that are not identical to their respective parents (Orriols-Puig, 2008). As in the case of the selection operator, distinct crossover operators can be found elsewhere in the literature (Goldberg, 1989, 2002, Herrera et al., 1998, Holland, 1976), being the most common the *two-point crossover*.

- **Mutation:** The mutation operator is responsible for an alteration of the genetic material of an individual. This alteration is stochastic and, jointly with the crossover operator, introduces new solutions. Again, several mutation operators can be found elsewhere (Freitas, 2002, Herrera et al., 1998), but they share the same idea: to introduce one or more (small) changes in individual genes.

- **Replacement:** This operator replaces unfit individuals of the old population with the newer offspring which are expected to be fitter. Several replacement schemas exist, being the common ones the *elitist replacement*, where the fittest individuals of the population are copied to the new population, and the *steady state replacement*, where the best individuals of the offspring are copied to the original population, removing the unfit individuals (Orriols-Puig, 2008).

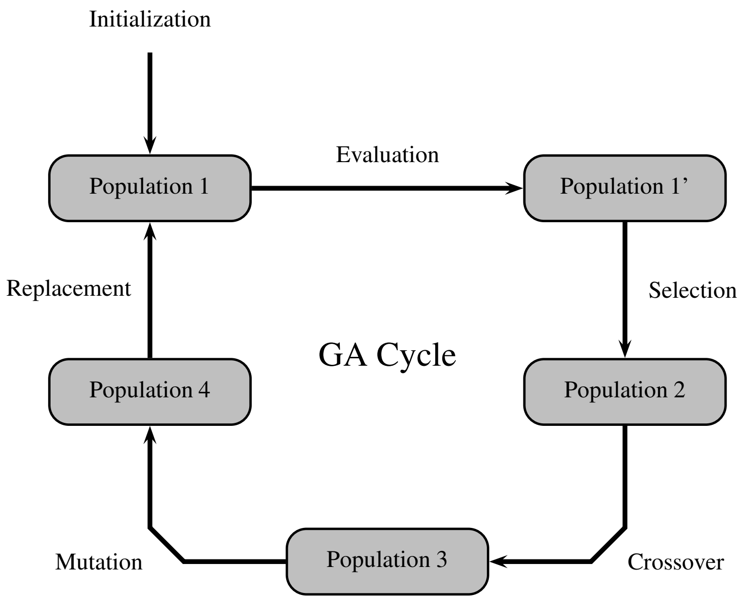


FIGURE 2.1: Evolution of a GA population through the GA cycle.

Goldberg (2002) fostered the idea that, while selection, crossover and mutation can be shown to be ineffective when applied individually, they might produce a useful result when working together. That is, selection and crossover operators introduce a process of innovation, whereas the combination of selection and mutation presents a continuous improvement process (Orriols-Puig, 2008).

Several authors have developed a formal theory to explain the behaviour of GAs and give some insights of why they work. The first attempts were done by Holland (1976) by introducing the notions of the *schema theorem*, which makes use of the concept of *building block* (BB). BBs can be understood by the analogy of a child's set of building blocks; pieces or assemblies of pieces that contribute at generating a final structure. Later, Goldberg (2002) proposed the methodology for designing competent selecto-recombinative GAs using Holland's work as fundamental basis. The next section briefly outlines these ideas, thus providing a formal background that help to explain why GAs work and gives some hints on competent GA design.

2.3.2 The Theory behind GA Design

The schema theorem is based on the idea of BBs, that is, a template that identifies a subset of individuals (Orriols-Puig, 2008). A schema is represented by a binary string $s = (s_1, \ldots, s_\ell)$ of length ℓ, where each bit s_i can take a value of the ternary alphabet $\{0, 1, \star\}$. Values 0 and 1 represent fixed values and $\star$ represents a "don't care", that is, a position that can take either 0 or 1. For example, provided the schema $00\star 11$, instances 00011 and 00111 belong to this template. The schema theorem uses the concepts of *order* and *length*, which are defined in what follows:

Order of a schema. The order $o(h)$ of a schema h is the number of fixed positions in the template, in other words, the number of bits that are either 0 or 1. For example, $o(01\star\star) = 2$.

Length of a schema. The length $\delta(h)$ of a schema h is the distance between the first and the last specific positions in the template. For example, $\delta(01\star\star) = 1$.

The schema theorem describes how the frequency of schema instances changes with the iterations. Then, the schema theorem demonstrates that the expected number of offspring that belong to schema h at iteration $t+1$ (i.e., $E[N_h(P(t+1))|P(t)]$) satisfies that (Goldberg, 2002):

$$E[N_h(P(t+1))|P(t)] \geq N_h(P(t)) \cdot \phi(h, N_h, P, F_i, t) \cdot P_s(h, N_h, P, F_i, t), \tag{2.1}$$

where ϕ is the selection or reproduction factor and P_s is the survival probability under the genetic operators applied. Assuming a fitness-proportionate selection, one point crossover and gene-wise mutation, the schema theorem models how the different templates evolve along a GA run, that is:

$$E[N_h(P(t+1))|P(t)] \geq N_h(P(t)) \cdot \frac{\overline{F}(h,t)}{\overline{F}(t)} \cdot \left(1 - \frac{\delta(h) \cdot P_\chi}{\ell - 1}\right) \cdot (1 - P_\mu)^{o(h)}, \tag{2.2}$$

where $N_h(P(t+1))$ is the number of individuals in the population $P(t)$ that belong to schema h at time t, $\overline{F}(h,t)$ is the average fitness of the individuals that belong to h at time t, $\overline{F}(t)$ is the average fitness of the population, P_χ is the probability of crossover, and P_μ is the probability of mutation. Equation 2.2 clearly denotes the effects of the three main components of GAs:

1. The effect of the fitness-proportionate selection, given by the term $\frac{\overline{F}(h,t)}{\overline{F}(t)}$, which increments the expectation of the number of individuals in the next generation if the average fitness of the individuals that belong to the schema h is greater than the average fitness of the population.

2. The effect of the crossover operator is represented by the term $1 - \frac{\delta(h) \cdot P_\chi}{\ell-1}$, which indicates that the probability that a schema survives depends on the length of the schema and the crossover probability. Note that a string of length ℓ can be crossed by $\ell - 1$ sites. The shorter the schema, the more chances of surviving the disrupting effects of crossover.

3. The effect of mutation is defined by the term $(1 - P_\mu)^{o(h)}$, which denotes that the probability that the schema is preserved with each one of the $o(h)$ fixed positions *intact* to the next generation is inversely proportional to the mutation probability and exponentially proportional to the order of the schema.

Therefore, the schema theorem supports that the expected number of individuals that belong to the schema h at time $t+1$ that (1) have a fitness above the average, (2) are short in length and (3) have a low order grows in the subsequent generation. For a more detailed derivations of the schema theorem the reader is referred to (Goldberg, 1989, Michalewicz, 1996, Mitchell, 1998).

Moreover, Goldberg (2002) using the aforementioned concepts suggested thinking of BB as a kind of matter and to ensure four conditions, namely ensure (1) that we have an initial stock of BB, (2) that good ones grow in the *market share*, (3) that *good decisions* are made among them, and (4) that they are *exchanged well* to solve a large number of complex problems (Orriols-Puig, 2008). In order to satisfy these conditions, Goldberg (2002) decomposes the problem of designing competent selecto-recombinative GAs in the following seven aspects:

1. Know what GA process—BBs.
2. Know the BB challenges—BB-wise difficult problems.
3. Ensure an adequate supply of raw BBs.
4. Ensure increased market share for superior BBs.
5. Know BB takeover and convergence times.
6. Make decisions well among competing BBs.
7. Mix BBs well.

Answering these questions leads to a facet-wise analysis of the GA—i.e., analysing separately each one of these elements assuming that the other ones behave in an ideal manner—, and the proposal of several models that all together have the objective of guaranteeing the success of a GA for problems of bounded difficulty (Bacardit, 2004).

As suggested by Goldberg (2002), selecto-recombinative GAs work through a mechanism of *decomposition* and *reassembly*, thus identifying the fitter BB. Then, these BB have to be correctly processed. Complex problems, however, are those whose BB are difficult to acquire. This could be a result of having large, complex BB, having BB that are hard to separate, or having a deceptive guidance toward high-order BB (Goldberg, 2002, Orriols-Puig, 2008). In order to ensure and adequate supply of BB and that these evolve in a *market economy of ideas*, increasing the population to obtain more variability is a typical solution.

Two important aspects are (Orriols-Puig, 2008): (1) the best BBs have to grow and take over a dominant market share of the population and (2) that this growth should be neither too slow nor too quick—to avoid falling in a local optimum. Different approaches have been taken to understand time and convergence, and the most important approaches are takeover time models, which model the dynamics of the best individuals, and selection-intensity models, where the dynamics of the average fitness of the population are modelled.

As the increase in the population size the likelihood of making the best possible decisions increase, decision making among competing BBs has been studied from the perspective of population sizing.

As pointed out by Orriols-Puig (2008), the correct identification and exchange of BBs is critical to innovative success. In other words, when designing a competent GA, one of the key challenges that needs to be addressed is how to identify BBs and exchange them effectively.

GAs have been broadly used as the core heuristic for knowledge discovery in machine learning. To this crossbreeding has been referred to as *genetics-based machine learning* (GBML) in the literature (Bacardit and Llorà, 2013, Goldberg, 1989). The following section reviews the most popular models of GBML learning systems, that is, Michigan-style, Pittsburgh-style LCSs and the related ones.

2.4 Learning Classifier Systems, a Quick Survey

Proposed by Holland (1976) and later simplified by Goldberg (1989) and Wilson (1994, 1995), LCSs are a family of machine learning algorithms that are strongly inspired on the observation of how natural selection and evolution solve complex tasks. The original purpose was to create *true artificial intelligence* mimicking the adaptive behaviour of Nature. The purpose of this section is to introduce the reader to the main branches of GBML. To do so, we begin with a description of LCSs as a whole. Next, Michigan-style and Pittsburgh-style LCSs, the two most well known models of GBML, are detailed. Finally, we present other forms of GBML that have been researched more recently.

According to Goldberg (1989), a classifier system—later renamed as LCS (Holland et al., 1999)—is a machine learning system that learns production rules (called *classifiers*) to guide its performance in an arbitrary environment. Any LCS consist of three main components:

1. A knowledge representation system based on production rules.

2. A rule evaluation mechanism, which estimates the usefulness of rules.

3. A rule discovery mechanism, which discovers new and promising rules.

LCSs are, typically, rule-based evolutionary systems (Butz, 2006). The population of classifiers codes the current knowledge of the LCS program. The evaluation mechanism, usually a reinforcement learning technique, estimates and propagates the rule utility at solving the task. Based on these estimates, the rule evolution mechanism, which is implemented by a GA, generates a new set of classifiers, the offspring, and then deleting the less useful classifiers.

There are two main LCS flavours, namely Michigan-style and Pittsburgh-style LCSs. The main differences are (1) how they code the population, being an individual a rule in Michigan-style and being an individual a set of rules in Pittsburgh-style, (2) the way the solution is obtained, being all the population the solution in Michigan-style and being the best individual the solution in Pittsburgh-style, and (3) Michigan-style is an online learning system whereas Pittsburgh-style is an offline one. These are discussed in more detail in the subsequent sections.

2.4.1 Michigan-style LCSs

As it occurs in Nature, living entities are not a global optimisation but a set of partial optimisations—or local specialisations—that in association solve, more or less successfully, the problem of life. Similarly, in the Michigan-style model the population is formed by a set of classifiers in which each classifier solves a concrete, specialised task of the given problem. Thus the solution to the problem is the entire population. In this approach the evolutive mechanism is not the main learning component, it only introduces new rules to the system (Llorà, 2002). Michigan-style LCSs are online systems and they learn by interacting with the environment adding new rules—and deleting older ones—on demand. Also, the quality of the solutions is evaluated online by a cognitive-inspired mechanism.

Since the first successful implementation (Holland and Reitman, 1977), the learning architecture of Michigan-style LCSs has been updated to tackle several drawbacks that early LCSs suffered. Decades after the invention of the first Michigan-style LCSs, *truly-working* architectures started to flourish, generating a renaissance in the field. The most important of those systems is *extended classifier system* (XCS) (Wilson, 1995, 1998), a competitive, accurate, and general-purpose machine learning technique. It is worth noting that XCS is the current state-of-the-art in the Michigan-style LCS community.

2.4.2 Pittsburgh-style LCSs

Another approach was envisaged with Pittsburgh-style LCSs. This model considers each individual as a complete solution to the given problem. That means that a single individual contains all the rules needed to solve the given problem. Thus, the objective is to find the individual with the best fitness of all the population of classifiers. The main learning component in this model is the evolutive mechanism, which is a straight and generational GA. Due to that issue, Pittsburgh-style systems converge in a single solution. These systems—typically supervised—are purely offline, so they first need the learning mechanisms to be completed before tackling the problem.

Smith (1980) developed the first implementation of Pittsburgh-style LCS and since then there have been a variety of successful designs (Orriols-Puig, 2008). One of the most competitive Pittsburgh approach algorithm, in terms of accuracy and scalability, can be found in GAssist (Bacardit, 2004).

More recently, other strategies inspired by traditional LCSs (i.e., Michigan-style and Pittsburg-style LCSs) have flourished. These are the *iterative rule learning* (IRL) (Venturini, 1993) and the *genetic cooperative-competitive learning* (GCCL) (Greene and Smith, 1993), and are briefly reviewed in the following.

2.4.3 Iterative Rule Learning

Differently from the aforementioned LCSs, the IRL approach follows a *separate-and-conquer* methodology to iteratively learn rules that cover a subset of the input examples given to the learning algorithm. IRL iteratively performs the following two steps until no training examples remain: (1) learn a new rule that covers part of the training examples via the genetic discovery mechanism and (2) remove the covered examples from the training set. Similarly to Michigan-style LCSs, in IRL each individual is a single production rule, and similarly to Pittsburgh-style LCSs, the genetic search is driven by a generational GA. A typical characteristic of IRL algorithms is that they make the rules available as a decision list.

The first IRL implementation can be found in (Venturini, 1993). More recently, other successful algorithms have been cradled under the IRL paradigm (Aguilar-Ruiz et al., 2003, Franco et al., 2013, Martínez-Ballesteros et al., 2011a,b).

2.4.4 Genetic Cooperative-Competitive Learning

Designed with the purpose of addressing the goal of constructing highly accurate and as-simple-as-possible decision models from a set of examples, GCCL systems assume that the training set correspond to niches in an *ecology* (Greene and Smith, 1993, Orriols-Puig, 2008). The exact number of niches is not known a priori, but it is assumed to be less than the total number of training examples. Then, as in the case of Michigan-style LCSs, the population is considered to be the whole model, which represents all the niches of the ecology, and each individual is a representation of a particular niche. Afterwards, the objective is to learn the minimum number of niches—i.e., individuals—that cover all the input space accurately.

Recently, the GCCL approach has been successfully applied for mining association rules that model consumers' behaviour (Casillas and Martínez-López, 2009).

2.5 Summary

This chapter has introduced machine learning as programs that learn by improving with experience at some task using a measure of the performance. machine learning techniques are very adequate to solve problems that are too complex to manually design and code such as human face recognition, very adequate to handle problems that require adapting to changing environments, and very adequate to process huge amounts of data to mine them and extract useful knowledge. Then, the classic taxonomy of machine learning techniques, that is, supervised, unsupervised and reinforcement learning, has been reviewed.

Afterwards, GAs have been detailed, describing the theoretical background on designing competent algorithms using these family of techniques. Then LCSs have been briefly introduced as rule-based evolutionary systems and showed its intrinsics, from the very beginning to the current standards.

The present work focuses on Michigan-style LCSs and starts with XCS, by far the most influential LCS in the field. In the next chapter, the Michigan-style LCS framework is concisely described by detailing XCS.

3 The Michigan-style LCS Framework Through XCS

Traditional Michigan-style LCSs are rule-based evolutionary learning algorithms that receive perceptions from the environment and, in response to these perceptions, perform actions to solve complex problems. These cognitive-inspired techniques have been successfully applied in different types of learning tasks. Michigan-style LCSs are open frameworks that foster crossbreeding between different learning paradigms. Three key aspects are required for every implementation of Michigan-style LCS: (1) a knowledge representation based on classifiers that enables the system to map sensorial status with actions, (2) an apportionment of credit mechanism which shares the credit obtained by the system among classifiers, and (3) an algorithm to evolve the knowledge base, which typically is a GA. The most successful technique that implements the Michigan-style LCS framework is XCS (Wilson, 1995, 1998). XCS is specifically designed to evolve a complete and accurate *payoff map* of all possible solutions for all possible problem instances (Butz, 2006) by basing the fitness function on the accuracy of the predictions the system performs. XCS is the most well-known Michigan-style LCS, and since its first versions a large amount of research has been conducted on formalising facetwise models for a better understanding of its underlying processes (Butz, 2006, Orriols-Puig et al., 2009a, 2010). The purpose of this chapter is to provide the Michigan-style LCS framework by concisely detailing the XCS classifier system. To do so this chapter departs from an overview of the XCS system, presented here in section 3.1. It introduces the knowledge representation used by this system, details the learning organisation of XCS, which is of the upmost importance for the correct understanding of the Michigan-style LCS framework, shows the action inference of XCS and briefly details the theoretical framework on which XCS (and hence every Michigan-style LCS) sustains. Also, a supervised extension of XCS is described in detail in section 3.2. Finally, section 3.3 summarises and concludes the chapter.

3.1 A Concise XCS Overview

As traditional Michigan-style LCSs, XCS evolves a population of maximally general and accurate classifiers online by means of interacting with an environment. Classifiers evolved by XCS consist of a two main components: a production rule and a set of parameters that estimate the quality of the rule. To such discovery, a steady-state GA is used from time to time. The main characteristic of XCS—and of the related systems, referred to as *accuracy-based systems*—, is the way in which fitness is computed: XCS computes classifier's fitness based on the accuracy of the reward prediction instead of calculating the fitness from the reward itself (Orriols-Puig, 2008), thus creating a *complete action map*, that is, a set of maximally general and accurate classifiers that map the problem space completely.

XCS learns new rules by interacting with an environment which provides a new training example at each iteration. In this regard, XCS has two operation modes: the *exploration* mode (i.e., the training stage), where XCS discovers new rules, and the *exploitation* mode (i.e., the test stage), where XCS uses its current knowledge to decide the best action for the new, previously unseen example.

At the very beginning XCS starts with an empty population running in exploration mode. At each iteration, the environment provides a new training example. Then the system builds a match set $[M]$, a structure containing all the classifiers in the population that whose conditions match the given instance. If the number of actions represented by the classifiers in the match set is less than the total number of possible actions of the problem, a covering mechanism is triggered generating as many new classifiers as required to cover all the possible actions. Next, the system makes a prediction of the payoff to be expected for each possible action in $[M]$. These predictions are stored in the prediction array PA. Once the prediction array is formed, the system has to select an action, the action that will be used to form the action set $[A]$, a structure which consists of all classifiers in $[M]$ advocating the action chosen. If the system is in exploration mode, the action is chosen randomly, exploring the consequences of all actions for each possible input, and if the system is in test mode, the action with more votes is chosen. Once the action is selected and the action set formed, the system executes the chosen action and the environment returns a reward quantifying the decision of the system.

Following that, all classifiers in $[A]$ get its parameters updated following a reinforcement learning scheme. After the evaluation, a GA is applied from time to time to the classifiers in the action set to discover new and interesting knowledge. A steady-state—only two members of the action set are changed at a time—niche-based—the GA is applied only in $[A]$ and not in the entire population—GA is used (Wilson, 1995, 1998). The first action the GA does is to select two parents from $[A]$ for reproduction. Individuals with higher fitness are more likely to be selected, but there is a chance to select lower fitness ones. This is an advantage because weak individuals may include some components that could prove successful when combined with fitter ones.

Next, this pair of parents are crossed to generate offspring. Afterwards, mutation may take place. If it is the case, for each variable, the mutation operator randomly decides whether the variable changes its value, choosing a new one. The action of the classifier can change following the same pattern. The resulting offspring is introduced in the population via subsumption, that is, if there exists an experienced and accurate classifier in the current population whose condition is more general than the new offspring, this latter is discarded. Otherwise, the new offspring are introduced into the population. The final solution of XCS consists of a set of rules that cover all the problem space. This means that the system will evolve two types of rules: (1) rules with high prediction and low error—that is, correct action—, and (2) rules with low prediction and low error—that is, wrong action.

In the subsequent sections, we introduce the learning architecture of XCS in detail. First, we review the knowledge representation assuming that classifiers are represented by real-coded rules (Wilson, 2000) using the unordered-bound interval representation (Stone and Bull, 2003). Also, single step tasks, which are the common under real-world classification problems, are assumed. Then, we explain the learning interaction of the system. Finally, we give some insights of the theory that explains why XCS is able to generalise and learn accurate classifiers.

3.1.1 XCS Knowledge Representation

XCS evolves a population $[P]$ of highly fit classifiers. The core of each classifier consists of a production rule, which identifies the set of values that define the domain of action of the classifier, and a set of parameters that estimate the quality of the rule. A rule takes the form

$$\textbf{if } x_1 \in [\ell_1,\, u_1] \textbf{ and } x_2 \in [\ell_2,\, u_2] \textbf{ and } \ldots \textbf{ and } x_k \in [\ell_k,\, u_k] \textbf{ then } a_j,$$

where the leftmost part contains k input variables—the input features—which take values of the interval $[\ell_i,\, u_i]^k$, where ℓ_i and u_i are the lower and upper limits, respectively, of each interval ($\forall_i \in \mathbb{N} :\ \ell_i \leq u_i$), and the rightmost part denotes the predicted action a_j (for the purpose of this thesis, we consider the action to be the same as the class of the problem). The parameters used by a classifier for evaluating the quality of the rule are the following: (1) p, an estimate of the reward the system will receive if the advocated action a_j of the rule is selected as output, (2) ϵ the prediction error, (3) the fitness F of the rule, (4) the experience exp, (5) the numerosity num or number of copies of this particular classifier in $[P]$, (6) as, an estimate of the average size of the actions sets in which the classifier has participated, and (7) the time stamp $timeOfCl$ of the classifier.

3.1.2 XCS Learning Organisation

XCS learns incrementally from a stream of examples that are provided by the environment. That is, at each learning iteration the environment provides a new training example $e = (e_1, e_2, \ldots, e_k)$ and XCS takes the following steps to learn from the environment. First, XCS creates a match set $[M]$ of classifiers consisting on those whose conditions match the input example. A rule matches an input example if $\forall_i :\ \ell_i \leq e_i \leq u_i$. If $[M]$ contains less than θ_{mna} classifiers (where θ_{mna} is a configuration parameter) with different actions, a covering mechanism is triggered which generates many different and matching classifiers (by means of using r_0 configuration parameter to tune the different intervals) as actions not previously present in $[M]$. For each of these, the system computes a prediction of the payoff to be expected if the rule is executed and stores them in the prediction array PA. It is computed as the fitness-weighted average of all matching classifiers that specify action a_i, that is

$$P(a_i) \leftarrow \frac{\sum_{cl.a=a_i \wedge cl \in [M]} cl.p \cdot cl.F}{\sum_{cl.a=a_i \wedge cl \in [M]} cl.F}, \tag{3.1}$$

where $cl.a$, $cl.p$, and $cl.F$ refer to the action, the reward prediction, and the fitness of the classifier respectively. If the system is in training mode, XCS chooses the action randomly out of PA (this is referred to as *exploration*). All the classifiers in $[M]$ advocating the selected action are put in the action set $[A]$ and a is sent to the environment, which returns a reward ρ. Next, the parameters of classifiers in $[A]$ are evaluated. Finally, if the average time since the last application of the GA of classifiers in $[A]$ is greater than the θ_{GA} threshold (a user-defined parameter), the genetic rule discovery is triggered.

Three elements are needed to be further elaborated in order to understand how XCS works. Those are (1) the covering operator, (2) the parameter update procedure, and (3) the rule discovery mechanisms are described in more detail in the following.

XCS Covering Operator

Given the input example $e = (e_1, e_2, \ldots, e_k)$—it is important to highlight that these values are normalised—, the covering operator generates a new classifier cl that fully matches the input instance e. For this purpose, the interval of each variable i of the new classifier is initialised as

$$\begin{aligned} p_i &\leftarrow e_i - \text{rand}(0, r_0) \quad \text{and} \\ q_i &\leftarrow e_i + \text{rand}(0, r_0), \end{aligned} \tag{3.2}$$

where r_0 is a configuration parameter, and $\text{rand}(0, \text{r}_0)$ returns a random number between 0 and r_0 using an uniform distribution. Therefore, this operator creates an interval that includes the value of the corresponding attribute, and r_0 controls the generalisation in the initial population. In this regard, li takes the lower value between p_i and q_i, and u_i the higher value. If e_i is unknown or missing, the covering operator replaces it with 0.5 and proceeds as usual. Classifier's parameters are set to initial values; that is, $\epsilon = 0$, $F = 0.01$, $num = 1$, $exp = 0$, as is set to the size of the action set where the covering has been applied, and $timeOfCl$ to the actual learning time stamp.

XCS Parameter Update Procedure

In training mode, and after the creation of $[A]$, the parameters of all the classifiers that belong to such set are updated. First, the experience of each classifier is incremented. Second, the reward prediction is updated following the Widrow-Hoff delta rule (Widrow and Lehr, 1990) using the full gradient (Butz, 2006) as

$$cl.p \leftarrow cl.p + \beta \left(\rho - cl.p\right) \frac{cl.F}{\sum_{cl_j \in [A]} cl_j.F}, \tag{3.3}$$

where β is the learning rate set by the user and ρ is the reward given by the environment after applying the action predicted by the rule. Third, the system updates the error prediction ϵ using a similar procedure as

$$cl.\epsilon \leftarrow cl.\epsilon + \beta \left(|\rho - cl.p| - cl.\epsilon\right). \tag{3.4}$$

Fourth, the niche size estimate is updated following as

$$cl.as \leftarrow cl.as + \beta(\sum_{cl_j \in [A]} cl_j.num - cl.as). \tag{3.5}$$

Finally, the fitness of the classifier is updated. In the first place, the accuracy $cl.k$ of the classifier is computed as

$$cl.k \leftarrow \begin{cases} \alpha \left(\frac{cl.\epsilon}{\epsilon_0}\right)^{-\nu} & \text{if } cl.\epsilon \geq \epsilon_0; \\ 1 & \text{otherwise,} \end{cases} \tag{3.6}$$

where α is a user-defined scaling factor, ϵ_0 is the error threshold defined by the user, and ν is the exponent of the power function used to tune the pressure towards highly fit classifiers. Afterwards, the classifier fitness is updated as follows

$$cl.F \leftarrow cl.F + \beta \left(\frac{cl.k \cdot cl.num}{\sum_{cl_i \in [A]} cl_i.k \cdot cl_i.num} - cl.F \right). \tag{3.7}$$

With the aim of letting the classifier parameters move quickly to their real values at the beginning of the classifier life, the *moyenne adpative modifiée* technique is used in XCS (Venturini, 1994). This technique sets the parameters of the classifiers directly to the average value computed with the instances that the classifier has matched. The parameters affected by this procedure are p, ϵ and as. This process is applied while the experience of the classifier is less than β^{-1}.

XCS Rule Discovery Mechanism

Two different mechanisms are used by XCS to discover new knowledge: the covering operator and a niche-based, steady-state GA (Bull and O'Hara, 2002, Wilson, 1998). XCS, as most Michigan-style LCSs, uses a steady-state niche-based GA to discover new promising rules. The GA is triggered in the current action set if the average time since its last application to the classifiers in $[A]$ is greater than the user-defined threshold θ_{GA}. If this happens, the evolutive component is triggered. First, the time stamp of each classifier in $[A]$ is updated. Next, the GA selects two parents from the current $[A]$ following a selection strategy. There are several ways to perform this selection and the most common are, in Michigan-style LCSs, proportionate selection (Wilson, 1995) and tournament selection (Butz et al., 2005).

Under proportionate selection, each classifier has a probability $P_{sel}(cl)$ to be selected proportional to its fitness

$$P_{sel}(cl) \leftarrow \frac{cl.F}{\sum_{cl_i \in [A]} cl_i.F}. \tag{3.8}$$

Under tournament selection, a proportion τ (an user-defined parameter) of $[A]$ is selected to participate in the tournament. In this scheme, the classifier with maximum fitness is chosen.

Then, two copies of these parent classifiers are made. These undergo crossover with probability P_χ and mutation with probability P_μ per allele. XCS typically uses uniform crossover: this operator decides, for each input variable, from which parent the information is copied. If crossover is not applied, the offspring remain as exact copies of the parents.

After this, mutation is applied: for each input variable, the mutation operator randomly decides whether the variable needs to be changed by adding a random value of the interval $[-m_0, m_0]$, where m_0 is a configuration parameter. The action predicted by a classifier also undergoes the mutation process.

The offspring parameters are initialised as follows: if no crossover is applied, the error and the fitness are copied directly from the selected parent. Otherwise, these parameters are set to the average value between the corresponding parameters in the parents. In all cases, the fitness is decreased to 10% of the parental fitness. Experience and numerosity are initialised to 1, and the average size of the niches in which the classifier has participated is set to the value of the selected parent.

The resulting offspring are introduced into $[P]$ via a subsumption mechanism: if there exists a sufficiently experienced and accurate classifier cl in $[P]$; that is, if $cl.exp > \theta_{sub}$, where θ_{sub} is a user-defined threshold, and $cl.\epsilon < \epsilon_0$, whose condition is more general than the new offspring, the numerosity of this classifier is increased (and, consequently, the offspring discarded). Otherwise, the new offspring is introduced into $[P]$. At this step, until the population is full, classifiers in $[P]$ are deleted proportionally to (1) their fitness and (2) their numerosity as

$$cl.P_{del} \leftarrow \frac{cl.d}{\sum_{\forall cl_i \in [P]} cl_i.d}, \tag{3.9}$$

where

$$cl.d \leftarrow \begin{cases} cl.num \cdot cl.as \cdot F_{[P]} & \text{if } cl.exp > \theta_{del} \text{ and } cl.F < \delta F_{[P]}; \\ cl.as \cdot cl.num & \text{otherwise,} \end{cases} \tag{3.10}$$

where $F_{[P]}$ is the average fitness of the population, θ_{del} is the classifier deletion threshold, and δ is and user-defined scaling factor. This deletion scheme biases the search towards highly fit classifiers and, at the same time, balances the classifiers' allocation in the different niches (Butz, 2006).

Once the learning stage finishes, the population is used to infer the action of new unlabelled instances. This process is detailed in the following section.

3.1.3 XCS Action Inference in Test Phase

In the test phase, the XCS action inference is performed using the knowledge acquired during the previous training stage. This process is performed in the following way: a new example, previously not known by XCS, is given to the system and all the matching classifiers in $[P]$ vote for the action they predict. This voting scheme is performed using the PA, as mentioned earlier. The most voted action is returned as the output. Notice that, during the test stage, the population is never modified.

It is worth mentioning that during the action inference XCS always chooses an action, but what happens when two or more actions are tied in the PA? In this particular case a common heuristic is to choose the action which appears most often in the data set.

3.1.4 Limitations and Further Improvements to XCS

XCS is a competent learner with a flexible architecture that allows the system to accurately handle a variety of situations. However, empirical studies (Butz et al., 2004, Orriols-Puig, 2008) detected two issues that may hamper the learning performance of XCS, namely *the covering challenge* and the *schema challenge*, which are summarised in the following.

Covering challenge. The covering challenge occurs when there is a cover-delete cycle and the system is unable to generate accurate and maximal-general classifiers during the covering stage. This issue can be overcome if the user-defined parameter r_0 is set *large enough*[1].

Schema challenge. The schema challenge happens when there is no fitness guidance. This issue can be handled by setting θ_{GA} *large enough* and r_0 *small enough.*

As the reader can deduce, solving the covering challenge may trigger the schema challenge (and vice versa), thus degrading the results obtained by XCS. In addition from cleverly setting the parameter r_0 to a tradeoff value between *too much* general and *too much* specific, more recently Orriols-Puig et al. (2010) developed a rather simple strategy to further improve the performance of XCS minimising the effects of the aforementioned issues. This strategy consist of the following: at the end of each learning iteration and after the GA has been triggered, an online covering operator is applied, generating a new matching classifier with the correct action a_c if the prediction of this action was not present in the PA. To do so, a new user defined parameter is introduced, c_0, which determines the generality of the variable conditions of the new classifier.

Another way of improving the system by means of condensation runs (kovacs, 1997, Wilson, 1995). It consists of running the evolutionary event forcing the crossover and mutation rates to zero, thus suspending the genetic search but allowing the classifier selection and deletion dynamics in the GA, resulting in a gradual shift towards more fit and general classifiers.

XCS is a complex online system which codifies its knowledge as classifiers that contain single rules, employs an apportionment of credit mechanism to evaluate the quality of these rules, and uses a steady-state, niche-based GA to discover new rules. In the subsequent section, we give some insights on how XCS works through a series of evolutionary pressures.

3.1.5 Theoretical Insights on Why XCS Works

Butz et al. (2004) identified five main pressures that guide XCS towards highly accurate and maximally general solutions. The explanations are maintained as simple as possible, and the mathematical foundations are skipped for the sake of clarity. The reader is referred to (Butz, 2006, Butz et al., 2004) for the technical details. These pressures are:

1. *Fitness pressure*, the main pressure towards higher accuracy.

2. *Set pressure*, which causes classifier generalisation.

3. *Mutation pressure*, which causes diversification in the search for better solutions.

[1]This issue is problem-specific; for more information see (Butz, 2006).

4. *Deletion pressure*, which emphasises the maintenance of a complete solution.
5. *Subsumption pressure*, which propagate accurate classifiers that are *syntactically* more general.

In the following, these five pressures are further elaborated following the summary given by Orriols-Puig (2008).

Fitness pressure. This pressure is present in all the mechanisms of XCS, influencing all the other pressures since selection, mutation and subsumption depend on the fitness of the classifier. In general, fitness results in a pressure which pushes the population from the over general side towards accurate classifiers. Late in the run, when the optimal solution is mainly found, it prevents overgeneralisation.

Set pressure. This pressure is mainly due to the combination of the application of the evolutive events in niches (the distinct $[A]$'s) and the deletion in the whole population. The result is a tendency towards generality favouring more general classifiers, that is, the most general and accurate classifiers will take over their niches, displacing both over-general and most specific, accurate classifiers.

Mutation pressure. Differently from the set pressure, mutation pressure causes the population to tend to more specific classifiers, thus diversifying the individuals.

Deletion pressure. Due to the resulting bias towards deleting classifiers that occupy larger action sets, deletion pressure pushes the population towards an equal distribution of classifiers in each environmental niche. Also, this pressure pushes towards removing classifiers with low fitness, driving the search towards the fittest individuals.

Subsumption pressure. This pressure pushes towards generalisation inside the niche. Once several accurate classifiers have been found, subsumption deletion causes the system to prefer the maximally general classifiers over the most specific ones. In this sense, subsumption produces an additional pressure towards generalisation.

Figure 3.1 summarises the interaction of the aforementioned pressures: the fitness pressure pushes [P] towards more accurate classifiers; the set pressure pushes towards more general classifiers; the subsumption pressure pushes [P] towards classifiers that are accurate and maximally general; the mutation pressure pushes towards a fixed proportion of volumes in classifier conditions. Deletion pressure is implicitly included in the notion of set pressure. Overall, these pressures lead $[P]$ towards a population of accurate maximally general classifiers (Butz, 2006, Butz et al., 2004).

3.2 Specializing XCS for Supervised Tasks: UCS

Although XCS has a general-purpose architecture which allows its usage in a broad variety of problems, it can be specialized to rocket the system capabilities at certain tasks, i.e., evolve a

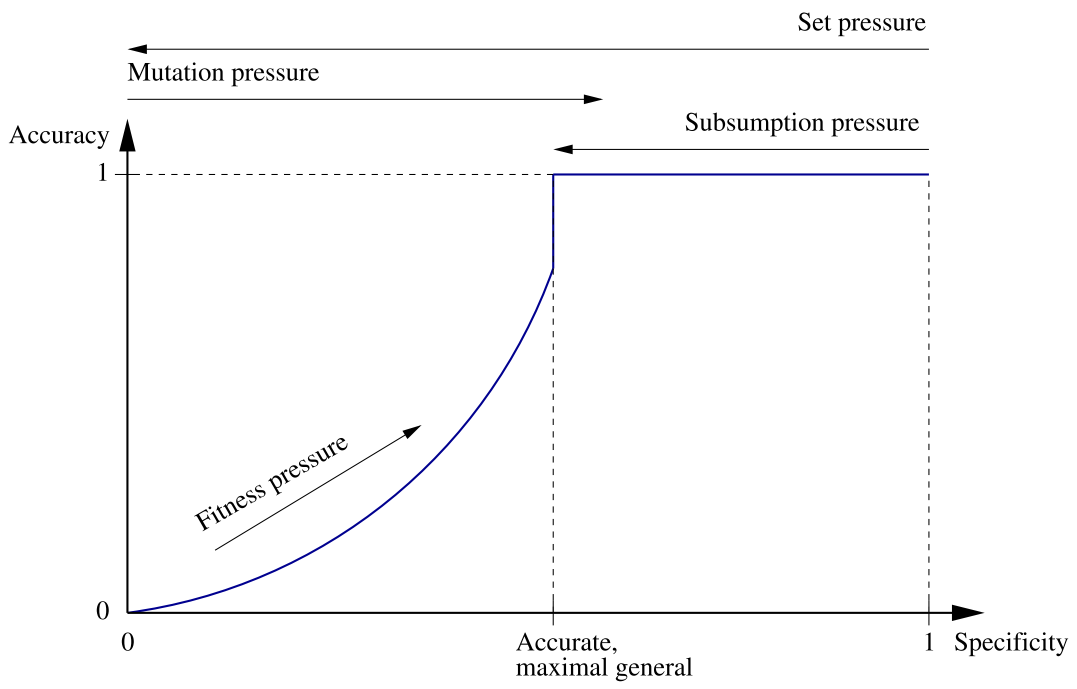


Figure 3.1: Interaction of the distinct pressures in XCS (Butz, 2006, Butz et al., 2004).

solution quicker and require less population to store the solution. The most successful architecture specialisation is found in the *supervised classifier system* (UCS) (Bernadó-Mansilla and Garrell-Guiu, 2003, Orriols-Puig and Bernadó-Mansilla, 2008). UCS is an accuracy-based Michigan-style LCS that takes advantage of knowing the class of the training instances, thus minimising the explore phase by searching for the *best action map*, which consist of the set of maximally general and accurate classifiers that predict the correct class (Orriols-Puig, 2008).

In the following, the learning architecture of UCS is described and the changes with respect to XCS are detailed.

3.2.1 Knowledge Representation in UCS

UCS, as it does XCS, evolves a population of classifiers which, together, cover the input space, learning from streams of examples. The main difference with respect to XCS is that UCS explicits the class instead of an action in every classifier, that is

$$\textbf{if } x_1 \in [\ell_1, u_1] \textbf{ and } x_2 \in [\ell_2, u_2] \textbf{ and } \ldots \textbf{ and } x_k \in [\ell_k, u_k] \textbf{ then } c_j,$$

where c_j is the class predicted by the condition part. As it happens in XCS, each classifier has a set of parameters that evaluate the quality of the rule. These parameters are (1) the rule accuracy acc, (2) the fitness F of the rule, (3) the experience exp, (4) the numerosity num or number of copies of this particular classifier in $[P]$, (5) cs, an estimate of the average size of the correct sets in which the classifier has participated, and (6) the time stamp $timeOfCl$ of the classifier.

3.2.2 Learning Organization in UCS

UCS receives input instances from the environment in the form of streams. Differently from XCS, UCS also takes the class c of the current training example (i.e., it receives $e = (e_1, e_2, \ldots, e_k)$ and c). Then, $[M]$ is created in the same way as XCS, that is, this set contains all the classifiers in $[P]$ whose condition matches the example given by the environment. Afterwards, the correct set $[C]$ is generated out of all classifiers in $[M]$ that predict the class c. If $[C]$ is empty, the covering operator is activated generating a single classifier with a generalised condition matching the input instance e and predicting the class c. Following that, the parameters of all the classifiers in $[M]$ are evaluated in a similar way as XCS. The main differences are (1) the evaluation is done in $[M]$ instead of $[C]$ because the accuracy of all the matching classifiers that predict the input instance wrongly needs to be decreased (Orriols-Puig, 2008), and (2) UCS does not use a prediction of the accuracy but the classification accuracy itself. Finally, in the same way as XCS, if the average time since the last application of the GA of classifiers in $[C]$ is greater than the θ_{GA} threshold, the genetic rule discovery is triggered.

The covering operator, the parameter update procedure and the rule discovery mechanism are briefly reviewed in the subsequent sections.

UCS Covering Operator

UCS uses the same covering operator as XCS, thus generating a new classifier cl that fully matches the input instance e. The difference with respect to XCS is that a single classifier is generated predicting the class c given by the environment. UCS classifier's parameters are set to initial values; that is, $acc = 1$, $F = 1$, $num = 1$, $exp = 1$, $cs = 1$ $timeOfCl$ to the actual learning time stamp.

UCS Parameter Update Procedure

In training mode, the parameters of all the classifiers that belong to $[M]$ set are updated. First, the experience of each classifier is incremented. Second, the accuracy is computed as the percentage of correct classifications:

$$cl.acc \leftarrow \frac{\text{number of correct classifications}}{cl.exp}. \tag{3.11}$$

Third, the niche size estimate is updated as the average of the sizes of the correct sets in which the classifier has taken part:

$$cl.cs \leftarrow cl.cs + \frac{\sum_{cl_j \in [C]} cl_j.num - cl.cs}{cl.exp}. \tag{3.12}$$

Finally, the fitness of the classifier is updated. In the first place, the relative accuracy $cl.k$ of each classifier is computed. For classifiers belonging to $[M]$ but not to $[C]$, $cl.k$ is set to zero; that is $\forall\, cl \notin [C] : cl.k \leftarrow 0$. For each classifier belonging to $[C]$, $cl.k$ is computed as follows:

$$cl.k \leftarrow \begin{cases} 1 & \text{if } cl.acc > acc_0; \\ \alpha \left(\frac{cl.acc}{acc_0}\right)^{\nu} & \text{otherwise,} \end{cases} \tag{3.13}$$

where acc_0 is the accuracy threshold defined by the user. Afterwards, the classifier fitness is updated using the classic Widrow-Hoff delta rule:

$$cl.F \leftarrow cl.F + \beta \left(\frac{cl.k \cdot cl.num}{\sum_{cl_i \in [C]} cl_i.k \cdot cl_i.num} - cl.F \right). \tag{3.14}$$

UCS Rule Discovery Mechanism

UCS uses the same rule discovery mechanism as XCS, that is, a steady-state niche-based GA. The GA is applied to $[C]$ following the same procedure as in XCS. UCS slightly differs from XCS in the selection operator and in the rule deletion mechanism. In the former, classifiers with low experience (i.e., $cl.exp < \theta_{del}$) got their selection probability scaled down by a factor of $cl.exp^{-1}$.

In the case of the deletion scheme, the offspring are introduced into $[P]$ via the subsumption mechanism: if there exists a sufficiently experienced and accurate classifier cl in $[P]$; that is, if $cl.exp > \theta_{sub}$ and $cl.acc > acc_0$, whose condition is more general than the new offspring, the numerosity of this classifier is increased and the offspring discarded. Otherwise, the new offspring is introduced into $[P]$. At this step, until the population is full, classifiers in $[P]$ are deleted proportionally to (1) their fitness and (2) their numerosity as

$$cl.P_{del} \leftarrow \frac{cl.d}{\sum_{\forall cl_i \in [P]} cl_i.d}, \tag{3.15}$$

where

$$cl.d \leftarrow \begin{cases} cl.num \cdot cl.cs \cdot F_{[P]} & \text{if } cl.exp > \theta_{del} \text{ and } cl.F < \delta F_{[P]}; \\ cl.cs \cdot cl.num & \text{otherwise,} \end{cases} \tag{3.16}$$

As fitness is computed from the proportion of correct classifications of a classifier, classifiers that predict wrong classes are not maintained in $[P]$, and so the best action map is evolved (Orriols-Puig, 2008).

3.2.3 UCS Class Inference in Test Phase

As it happens in XCS, in the test phase, the UCS class inference is performed using the knowledge acquired during the previous training stage. This process, very similar to the one of XCS, is performed in the following way: a new example, previously not known by UCS, is given to the system and all the matching classifiers in $[P]$ vote for the class they predict proportional to their fines and accuracy, that is:

$$P(c_i) \leftarrow \sum_{cl.c = c_i} cl.F \cdot cl.acc. \tag{3.17}$$

The most voted class is selected as the output. Notice that, during the test stage, the population is never modified.

3.3 Summary and Conclusions

The appearance of the XCS classifier system sparked a renaissance for the Michigan-style LCS field. Its simplified structure with respect to Holland's original version allowed XCS to avoid the generation of an excessive number of over-general classifiers and to provide high generalisation capabilities due the combination of a niche-based GA and a population-wise deletion operator (Orriols-Puig, 2008).

In this chapter, the general Michigan-style LCS framework has been detailed through XCS. A gentle description of the XCS classifier system has been provided, detailing its general learning scheme. Afterwards, the theoretical insights into why XCS works have been shown, which consists on a set of pressures that guide Michigan-style LCS towards highly accurate and maximally general solutions.

In the second part of this chapter, a supervised specialisation of XCS, the UCS classifier system, has been described. UCS differs from XCS in two aspects, namely (1) the way the fitness is computed and (2) the exploration stage. In this regard, UCS minimizes the exploration phase by searching for the best action map, which consist of the set of maximally general and accurate classifiers that predict the correct class, thus it requires less classifiers to represent its knowledge.

In the forthcoming chapters the Michigan-style LCS architecture is extended to handle dynamic situations effectively. First, the supervised paradigm is carefully analyzed departing from XCS/UCS architecture. A new algorithm is designed to deal with continuous, high-speed and time-changing flow of information. The purpose of this new algorithm in the LCS family is twofold: (1) being robust to noise and (2) having a fast reaction to concept drifts, while being accurate. Finally, we revise the unsupervised paradigm, facing the challenges of (1) online clustering and (2) association streams.

4 Supervised Algorithms for Data Streams

The increasing integration of technology in the different areas of science and industry has resulted in the design of applications that generate large amounts of data online. Most often, extracting novel and useful information from these data is key, in order to gain a better understanding of the processes that the data are describing. Learning from these data poses new challenges to traditional machine learning techniques, which are not typically designed to deal with problems in which concepts and noise levels may vary over time. Due to the online nature of Michigan-style LCSs, XCS, the most studied of its kind, its capable of handling the aforementioned environments. However, previous studies (Abbass et al., 2004) have revealed that although XCS works under dynamic streams, its reaction capacity is not as quick as desirable. In this regard, the purpose of this chapter is to present the *supervised neural constructivist system* (SNCS), an accuracy-based neural-constructivist LCS that makes use of multilayer perceptrons to learn from data streams with a fast reaction capacity to concept changes and a remarkable robustness to noisy inputs. The behaviour of SNCS on data stream problems with different characteristics is carefully analysed and compared with other state-of-the-art techniques in the field. Furthermore, SNCS is also compared with XCS and UCS using the same testbed to highlight their differences. In order to obtain a more accurate view of how SNCS behaves, this comparison is also extended to a large collection of real-world problems. The results obtained support that SNCS can function in a variety of problem situations producing accurate classification of data, whether the data are static or in dynamic streams, surpassing the reaction capacity of current Michigan-style LCSs.

The remainder of this chapter is organised as follows. Section 4.1 introduces the necessary background to follow the chapter. Section 4.2 highlights the contributions of supervised data stream mining. SNCS is described in detail in section 4.3. In section 4.4, SNCS is tested in a wide set of online environments with distinct characteristics. To show the robustness of SNCS section 4.5 shows the results of SNCS under a large collection of real-world problems with static concepts. Finally, section 4.7 summarised and concludes the chapter.

4.1 Supervised Learning from Data Streams

In modern times, the need of systems that are able to monitor, track and control massive amounts of data, usually as a continuous, high-speed, noisy, and time-changing flow or stream of information has increased dramatically (Aggarwal, 2007, Gama, 2012, 2010, Gama and Gaber, 2007, Gama et al., 2013, Khan, 2010, Zhu et al., 2010). Stock markets, smart networks and security sensors, among many others, are the primary targets of this kind of knowledge extraction. Also, this learning paradigm presents the following challenges: (1) the concept inside data may change over time (also referred to as concept drift) and (2) the noise levels may change per concept (Lughofer and Angelov, 2011, Masud et al., 2011, Vivekanandan and Nedunchezhian, 2011). Classical machine learning methods, typically offline, are not able to extract accurate models when the information is as described (Lakshmi and Reddy, 2010, Núñez et al., 2007). Because of the importance of this problem there were several contributions to adapt these learners in order to properly handle data streams mainly making use of (1) time windows or (2) algorithm tuning to *transform* them into online learners. The use of time windows to store the data stream—or at least parts of these—make those batch learners suitable (Maloof and Michalski, 2004, Widmer and Kubat, 1996). However, managing time windows is not exempt of decision challenges: the size of the window is the key aspect to determine because it controls the ability of forgetting past examples that are no longer useful. Also, deciding which examples are no longer useful is not a trivial task. Is in data stream mining where incremental, online learners take advantage by handling directly the streams. Although the contributions in recent years in the online field (Orriols-Puig and Casillas, 2010b, Torres et al., 2012), there are still challenges to be faced: (1) fast reaction to concept changes, (2) robustness of the learner against noise, and (3) obtaining accurate results under high dimensional spaces.

The purpose of this chapter is twofold: (1) design a system based on the guidance of Michigan-style LCSs (Holland, 1992) that learns from data streams with a fast reaction capacity and adaptability to changes in concept with noisy inputs and (2) demonstrate its competitiveness with respect to the state-of-the-art methods in both data-stream problems and traditional supervised learning problems. Inheriting the intrinsically online fashion of LCSs, the here presented *supervised neural constructivist system* (SNCS)—a *Michigan-style neural-learning classifier system* (N-LCS) (Bull, 2002) that evolves a population of multilayer perceptrons (MLP) online by means of interacting with an environment—is analysed under a variety of data stream environments which contain the common difficulties of the field. These difficulties are identified as

1. abrupt concept changes,
2. varying concept drift speeds,
3. varying noise levels,
4. changes in the distribution of the input examples without a concept change (referred to as *virtual drifts*),
5. padding variables under high dimensional spaces, and

6. non-linearities.

We compare SNCS results with the ones of the instance-based classifier IBk (Aha et al., 1991), the statistical classifier Naïve Bayes (NB) (John and Langley, 1995), both tuned to handle data streams, and the results of the tree-based algorithm CVFDT (Hulten et al., 2001), one of the most accurate and relevant algorithms in the field. To show the advantages of the architecture designed, SNCS is also compared to XCS and UCS using the same testbed, performing the analysis separately for clarity. In addition, in order to demonstrate the excellence of SNCS on general classification tasks, we also extend the study of SNCS by experimenting with a set of real-world problems with static concepts, comparing its results with the ones of the most significant machine learning methods: (1) the decision tree C4.5 (Quinlan, 1993), (2) the sequential minimal optimization (SMO) (Platt, 1998) support vector machine (Vapnik, 1995), (3) the multilayer perceptron (MLP) (Rumelhart et al., 1986, Widrow and Lehr, 1990) neural network, (4) the statistical classifier Naïve Bayes (NB), and (5) the instance-based classifier IBk.

This chapter provides the following contributions:

- It provides a study of the problem of data streams with concept drift from a practical point of view.
- It details a supervised stream miner with a fast reaction capacity and high adaptability to harsh situations.
- It explores the online capacities of Michigan-style LCSs under extreme environments.

4.2 Related Work

LCSs are online machine learning techniques that use *genetic algorithms* (GAs) (Goldberg, 2002, Holland, 1992) and an *apportionment of credit mechanism* to evolve a rule-based knowledge. LCS have been successfully applied to a broader kind of problems such as data classification, reinforcement learning and data clustering. However, it has been shown that the lack of flexibility of LCS may hinder from learning complex domains (Wilson, 2008). In order to improve flexibility and obtain accurate models beyond the limitations of rule-based systems, the use of neural networks, a more complex representation, flourished. Neural networks have the following advantages (Zhang, 2000): (1) they provide a data driven self-adaptive representation, (2) they can approximate any function with arbitrary accuracy, and (3) they make use of flexible, non-linear models, capable of modelling real-world complex relationships. The N-LCS scheme was first proposed by Bull (2002) as a way to improve traditional rule-based LCS, such as ZCS (Wilson, 1994), using the flexibility of the representation provided by artificial neural networks on a coevolutionary approach. Later on, this N-LCS scheme fused with accuracy-based LCSs (Bull and O'Hara, 2002, Howard et al., 2009, 2010) such as XCS (Wilson, 1995) and XCSF (Wilson, 2001).

Those systems used neural constructivism-inspired mechanisms (Quartz and Sejnowski, 1999) to adapt the structure of neural networks dynamically through interactions with the environment and solve reinforcement learning tasks in an online fashion. The results obtained so far show that, in fact, LCS can benefit from using a richer and more complex representation. However, this area is still young and it has not been fully explored. Thus, our purpose with SNCS is to follow these steps by exploiting the concepts of N-LCS for data stream mining tasks.

A major challenge of learning from data streams lies in the detection of changes in the target concepts. Recent contributions tackle this issue (Lughofer and Angelov, 2011, Mozafari et al., 2011) by performing some incremental statistical test over the incoming streams of data. Mining huge volumes of data under dynamic streams is a hot research topic an there are several recent contributions in both supervised (Angelov and Xiaowei, 2008, Orriols-Puig and Casillas, 2010b) and unsupervised (Baruah and Angelov, 2012a, Chandra and Bhaskar, 2011) learning paradigms.

The intrinsics of the knowledge representation, the evaluation mechanism and the knowledge discovery procedure of SNCS—the three key factors for every Michigan-style LCS algorithm—are detailed in the next section.

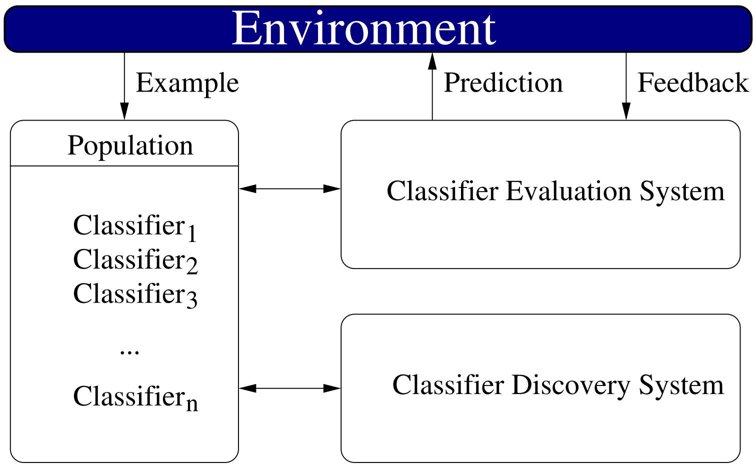


Figure 4.1: Michigan-style LCS Framework.

4.3 Description of SNCS

Michigan-style LCSs are open frameworks that foster crossbreeding between different learning paradigms (Butz, 2006). Three key aspects are required for every implementation of Michigan-style LCS: (1) a knowledge representation based on classifiers that enables the system to map sensorial status with actions, (2) an apportionment of credit mechanism which evaluates the classifiers, and (3) a knowledge discovery mechanism. These are depicted in Figure 4.1.

As detailed in chapter 3, Butz (2006) detected a set of pressures that help Michigan-style LCSs to obtain accurate results and that explain why Michigan-style LCSs work. Despite the fact that these studies are referred to XCS, the most studied LCS and a particular implementation of Michigan style-LCS, they can be extrapolated to other systems that follow the same framework.

We have designed a new Michigan-style LCS which (1) uses MLPs as classifiers (a more flexible representation than classical rule lists), (2) uses its own apportionment of credit mechanism and (3) adapts the GA to the new requirements of the representation.

SNCS is an accuracy-based N-LCS that evolves a population of classifiers online by means of interacting with an environment. The main difference between SNCS and other N-LCSs is that SNCS specialises its online learning architecture for supervised data stream mining and classification tasks. Thus, SNCS is designed using the Michigan-style framework to acquire its knowledge model incrementally and to quickly adapt to concept changes.

In what follows, the learning mechanisms of the SNCS classifier system are shown. First, the knowledge representation is discussed in detail. Then, the apportionment of credit mechanism of the learner is detailed: how SNCS interacts with its online environment and how the classifiers are evaluated following the framework of Michigan-style LCS. Thereafter, the evolutive component of SNCS is shown. Following that, the inference system is detailed. Finally, an analysis of the complexity of the algorithm is performed.

4.3.1 Knowledge Representation

SNCS uses a *population* [P] of *classifiers*, but differently from *traditional* LCSs, it does not contain a rule and an action, but an MLP with a *feed forward* topology (i.e., the signal propagates from inputs toward the output layer). This topology consists of several neurons (also referred to as perceptrons) that are interconnected. In this regard, perceptrons are organised in three distinct layers:

- An input layer consisting of the input attributes of the problem.
- A hidden layer that varies in size. This feature is exposed later.
- An output layer with a number of neurons equal to the number of possible labels or classes of the problem, where each neuron in this layer maps their output as the label they predict.

In each layer, there is, at least, one *sigmoid unit*, that is, the activation function used by the neuron is a sigmoid:

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}, \tag{4.1}$$

where w_0 is the bias weight, w_i is the weight of the *ith* input and x_i is the *ith* input. Learning is understood as a modification of the network weights in order to minimize the error between the expected output and the computed one.

The architecture of two "true" layers (hidden and output layers) plus the input vector has been chosen due to the following Theorem (Cybenko, 1989):

Theorem 1 *Let f be any continuous discriminatory*[1] *function. Given a function F and a*

[1]Discriminatory functions allow to tell whether a measure is zero or not.

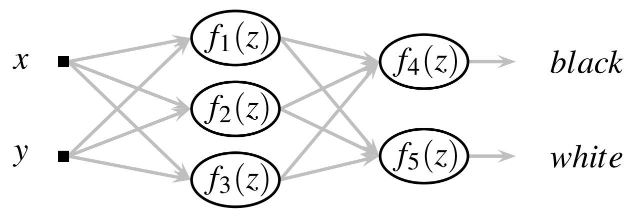
positive number ε there exists a finite sum of the form

$$G(x) = \sum_{i=0}^{n} \alpha_i f(w_i \cdot x + \mu_i) \tag{4.2}$$

such that

$$\|G(x) - F(x)\| < \varepsilon, \forall x. \tag{4.3}$$

Theorem 1 states that we can get arbitrarily close to any given function. Since sigmoids happen to be discriminatory functions the above theorem can be understood as the action of a two layer perceptron on the inputs. All this means that MLPs with two layers are universal classifiers.



$$f_k(z) = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i \cdot z_i)}}$$

Node $f_1(z)$:	Node $f_2(z)$:	Node $f_3(z)$:	Node $f_4(z)$:	Node $f_5(z)$:
w_0: -11.44	w_0: -12.82	w_0: -0.23	w_0: -2.19	w_0: 2.19
x: -20.82	x: 22.13	x: 15.96	f_1: 4.09	f_1: -4.09
y: 1.75	y: -3.46	y: 35.41	f_2: 9.54	f_2: -9.54
			f_3: -8.75	f_3: 8.75

FIGURE 4.2: The knowledge representation of SNCS in the *tao* problem.

As in any MLP, the predicted label is obtained by performing feed forward (Widrow and Lehr, 1990), which consists in giving the input instance vector $\mathbf{x}$ to the network and, for each neuron on the hidden layer, calculating the output of this vector (according to Equation 4.1) and passing it to the output layer, which computes, again, its output using the same Equation. The output neuron with the most intense signal is the winner and determines the label. Figure 4.2 illustrates the representation of SNCS.

For example, if we have a problem with four input attributes and two classes (the positive class $\{+\}$ and the negative class $\{-\}$), the input vector $\mathbf{x}$ will consist of these four input attributes and will be the input of the MLP. If we assume that in this example the hidden layer consist of three neurons, for each one of these, the formula 4.1 is applied and its results are passed to the output layer. Thus, the neurons on the output layer will have, as input, the output computed by the hidden layer. With these data, each unit in the output layer will compute the strength of the class they predict using the same formula. If the MLP has the following output $< 0.82, 0.16 >$, that is, a signal of 0.82 for the class $\{+\}$ and a signal of 0.16 for the class $\{-\}$, the predicted output will be class $\{+\}$ due to the strength of this signal.

SNCS contains, also, a set of parameters that estimate the quality of the classifier. These parameters maintain different statistics of the quality of the MLPs. These are:

- The fitness F, which is computed as a function of the accuracy.
- The experience *exp* of the classifier, which counts the number of times since its creation that the classifier has been in a species set.
- The time stamp t, which denotes the time-step of the last occurrence of a GA in the correct species set to which the classifier has belonged.
- The accuracy *acc*, which maintains an average of the number of examples correctly classified.

4.3.2 Interaction with the Environment

SNCS has two online operation modes, the *training* mode and the *test* mode. In training mode, SNCS discovers and/or evolves new MLPs that accurately predict the desired label. In test mode, SNCS uses its current knowledge to determine the best label for a new input example (and the model is not updated).

At the very beginning of the execution, the system starts the training phase with an empty population. At each iteration, the environment provides a new example with its label. Then, the system creates the different *species sets* [S] containing the classifiers that predict the label of the associated $[\mathrm{S}]_i$—each $[\mathrm{S}]_i$ predicts a different label, i.e., $[\mathrm{S}]_1$ predicts the label 1, $[\mathrm{S}]_2$ predicts the label 2, and so on. To obtain that prediction, the MLP of the classifier performs a feed forward step with the example provided by the environment. The system has the label of the input example so if the correct species set $[\mathrm{S}]_c$ is empty and there is free space for a new classifier, a covering operator is triggered, which creates a new classifier that predicts the label given by the environment. This covering mechanism is described as follows.

A blank MLP is generated with the number of neurons in the hidden layer chosen in the following way:

$$n_{hid} \leftarrow \tau_h + \mathrm{rand}(-\tau_h, \tau_h), \tag{4.4}$$

where

$$\tau_h = \frac{1}{2} \cdot (\text{number of inputs} + \text{number of labels}). \tag{4.5}$$

The value of τ_h in Equation 4.5 follows an heuristic based on the observation that MLPs try to maximise the accuracy on average from the beginning if the hidden layer contains a number of units that is proportional to the number of input features and to the number of labels of the problem. Other heuristics are possible, for instance picking the number of hidden units at random with the restriction of having at least a single unit and Max_{hid} units (a configuration parameter) as the upper bound, but experiments with several datasets following our heuristic seemed to maximise the accuracy of the MLPs from the beginning of the run.

The number of neurons in the output layer is equal to the number of labels. All weights of the MLP are initialised randomly between 0 and 1. After this, the system triggers a subsumption mechanism: if there exists another classifier in [P] with the same number of neurons in the hidden layer, the new classifier is automatically discarded. This way SNCS speeds up by not allowing the existence of repeated structures in [P].

To fit the example, the system performs stochastic backpropagation (Mitchell, 1997, Werbos, 1990) iterations with the provided example and label until the MLP is able to correctly predict the desired output. Stochastic backpropagation is based on minimising the cost error function for each training instance, in an incremental way:

$$E_d \leftarrow |\mathbf{o} - \mathbf{y}|^2, \tag{4.6}$$

where **o** is the expected output of the MLP and **y** is the real output of the MLP for the actual example. To minimise the E_d, weights in the MLP are updated following an incremental gradient method:

$$\Delta w_{ij}(t) \leftarrow -\beta \frac{\partial E_d}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1), \tag{4.7}$$

where $\Delta w_{ij}(t)$ is the increase (or decrease) of the weight value form input i to unit j at time t, β is the learning rate (an user-defined parameter), α is the momentum (another configuration parameter), and $\Delta w_{ij}(t-1)$ is the weight update at the previous time step. In our case, we tuned this learning scheme in order to adjust the novel classifier to the actual label provided by the environment. This training scheme fits (1) with the idea of neural constructivism and (2) with the online nature of data streams because it is an incremental and relatively fast method and also it is simple to implement for arbitrary network topologies (Koščak et al., 2010).

We have experimented with this approach and it respond-ed adequately in online environments due to the fact that, to just predict the desired label, this strategy only needs, in general, a few iterations. To avoid a possible excess in the number of covering iterations, a maximum of φ iterations, where φ is a configuration parameter, are used. Other strategies can be found for example in Howard et al. (2009) work, where covering is performed by repeatedly generating random MLPs with a single hidden layer neuron until the desired label matches.

Covering aims at discovering a single classifier that predicts the input example correctly. The different parameters of the new classifier are set to: $F = 1$, $exp = 1$, and $acc = 1$.

After this, the parameters of all classifiers in the different $[\mathrm{S}]_i$ are evaluated, and eventually, $[\mathrm{S}]_c$—which defines a niche of classifiers with the same predicted label as the one of the example provided by the environment—is updated with the evolutive mechanism. The parameter update procedure is applied to all the different $[\mathrm{S}]_i$ since the accuracy of all the classifiers that predict the output instance wrongly needs to be decreased.

4.3.3 Classifier Evaluation

Differently from most LCSs, all classifiers are evaluated each time due to the fact that they participate in the different $[\mathrm{S}]_i$. After each learning step, their experience, accuracy and fitness are updated in an incremental way. Firstly, the experience is increased:

$$cl.exp \leftarrow cl.exp + 1. \tag{4.8}$$

Next, the classifier is trained with the example and label provided by the environment by performing a single stochastic backpropagation iteration.

Then, the accuracy is computed as the percentage of correct classifications, that is:

$$cl.acc \leftarrow \frac{number\,of\,correct\,classifications}{cl.exp}. \tag{4.9}$$

While updates of experience and accuracy are straightforward, the update of the fitness is more complex, due to its mechanism. First, a new accuracy $cl.k$ is calculated, which discriminates between accurate and inaccurate classifiers. For classifiers not belonging to the correct species set, $cl.k$ is set to zero. For each classifier belonging to the correct species set, $cl.k$ is computed as follows:

$$cl.k \leftarrow \begin{cases} 1 & \text{if } cl.acc > \overline{acc}_{[P]}; \\ (max_{acc} \cdot cl.acc)^{\nu} & \text{otherwise,} \end{cases} \tag{4.10}$$

where $\overline{acc}_{[P]}$ is the mean accuracy of the classifiers in [P] which indicates the accuracy threshold, max_{acc} is the accuracy of the fittest classifier in $[\mathrm{S}]_c$, and ν is the constant controlling the rate of decline in accuracy. Finally, $cl.k$ is employed to update the fitness using the user-defined η parameter as:

$$cl.F \leftarrow cl.F + \eta\,(cl.k - cl.F)\,. \tag{4.11}$$

Once the parameters of the classifiers in its different $[\mathrm{S}]_i$ have been evaluated, the GA can be applied to $[\mathrm{S}]_c$. This process is explained in what follows.

4.3.4 Evolutive Component

A steady-state niche-based GA (Bull and O'Hara, 2002) is used to evolve topologies in $[\mathrm{S}]_c$. This GA is applied only if the average time since the last application to the classifiers in $[\mathrm{S}]_c$ is greater than the configuration parameter θ_{GA}.

The first thing to do, once the GA can be applied, is to select an MLP from $[\mathrm{S}]_c$ and clone them. There are several ways to perform this selection and the most common are, in Michigan-style LCS, proportionate selection (Wilson, 1995) and tournament selection (Butz et al., 2005). Under proportionate selection, each classifier cl has a probability $p_{sel}(cl)$ to be selected proportional to its fitness. Under tournament selection, a proportion τ (an user-defined parameter) of $[\mathrm{S}]_c$ is selected to participate in the tournament. The classifier with maximum fitness is chosen. We tested both selection mechanisms and tournament gave better results on average (not shown for brevity). Thus we show the results of SNCS using tournament selection for now on.

SNCS does not let young classifiers ($cl.exp < \theta_{exp}$) have a strong participation in the selection by scaling their fitness as follows:

$$cl.F' \leftarrow \frac{1}{\theta_{exp}} cl.F. \tag{4.12}$$

Next, the constructivist event takes place by randomly adding or deleting a certain amount of neurons in the hidden layer of the offspring MLP with probability ω (ω is a configuration parameter). The process is as follows: a random number is generated in the following form:

$$v \leftarrow \text{rand}(1, \textit{number of labels}). \tag{4.13}$$

After this the system generates another random value p ($p \in [0, 1]$) and, if p is less than the value of ω ($p < \omega$) the v new neurons are added to the hidden layer of the offspring. Otherwise the v neurons are subtracted. The minimum number of neurons in the hidden layer is 1 and the maximum is Max_{hid}, where Max_{hid} is another configuration parameter. The remaining neurons of the MLP, if any, are left untouched due to the fact that the small change in the topology seems not to dramatically affect the learned weights while being accurate. This effect has been observed empirically.

After this process, the system triggers the subsumption mechanism to assure that there is no repeated topology in [P]. If there is no repeated topology, the new MLP updates its parameters as follows: *exp* is set to 1, F is set to 1, and *acc* is set to 1.

Next, to properly tune the new topology, the system triggers a stochastic backpropagation iteration with the example and label provided by the environment. This process can be understood as a mutation since the remaining original weights will vary a small quantity because of the change on the topology.

Finally, the new classifier is added into the population. If the population is full, excess classifiers are deleted in the following way: if the classifier is experienced enough ($cl.exp > \theta_{exp}$) and its fitness is a fraction of the average fitness of the classifiers in [P] ($cl.F < \delta\overline{F}_{[P]}$, where δ is a scaling factor and $\overline{F}_{[P]}$ is the average fitness of the classifiers in [P]), the classifier is deleted.

4.3.5 Inference System

In test mode, all the population is used to predict the label of a new input example. To maintain a compact and accurate population, once the training phase is finished, the final population is reduced deleting those classifiers that have low experience ($cl.exp < \theta_{exp}$) or low fitness ($cl.F < \theta_{del}$, where θ_{del} is a configuration parameter). This reduction strategy does not significantly affect the final accuracy of the population because it deletes those classifiers with low participation. After this process of compaction is performed, the inference system is triggered. The most voted label is returned as output to the environment. Firstly, $[S]_i$ are created with the current knowledge. The update and search mechanisms are disabled. After the different $[S]_i$ are created, the classifiers emit a vote for the class they predict, weighted by their accuracy and fitness. The vote of young classifiers is decreased by scaling its vote by $^1/_{\theta_{exp}}$. With this scheme, classifiers with a low experience will have less participation in the voting than experienced classifiers.

4.3.6 Algorithm Complexity

As Michigan-style LCSs, SNCS incrementally evolves its knowledge model, that is, SNCS learns from streams of examples. This enables the system to learn from time-changing flows of information. Thus, SNCS does not need to process all the training data to produce a working model. In that matter, the cost of the algorithm increases linearly with (1) the maximum population size N, (2) the number input attributes n_{in}, (3) the maximum number of neurons per classifier $max_{neurons}$, and (4) the maximum number of interconnections between neurons max_{con}. That is summarised in the following:

$$Cost_{SNCS} = O(N \cdot n_{in} \cdot max_{neurons} \cdot max_{con}), \tag{4.14}$$

but it does not depend directly on the number of examples.

In respect to the memory demands of SNCS, the main cost of the algorithm lies in the population. SNCS, as most Michigan-style LCSs, does not store the examples in main memory. SNCS has a maximum population size of N MLPs, each composed of $max_{neurons}$. We can further analyse this: each MLP contains a maximum of max_{hidden} neurons in the hidden layer and n_{output} neurons in the output layer. SNCS represents MLPs as vectors of real numbers storing (1) the weights of the two layers, (2) the previous weights for computing the momentum for the two layers, (3) the weights for calculating the deltas for the two layers (Mitchell, 1997) and (4) the outputs for the two layers. In this regard, every MLP requires $4 \cdot (max_{hidden} + n_{output})$ floats.

Having explained in detail the intrinsics of SNCS, in the next section, we analyze the system in a set of different environments: (1) data streams with different complexities and (2) real-world problems.

4.4 Experiments on Data Stream Problems

We wanted to test SNCS under problems with similar complexities to the ones of real-world data streams. Because of the lack of a public repository of these kind of problems, we selected two of the most used families of data stream benchmarks: (1) the rotating hyperplane problem (Hulten et al., 2001), and (2) the SEA problem (Street and Kim, 2001) and some of its variants (Núñez et al., 2007, Orriols-Puig and Casillas, 2010b). These problems allowed us to test different types of difficulties that online systems have to deal with. More precisely, these difficulties are: (1) abrupt concept changes, (2) concept drifts at varying speeds (3) varying noise levels, (4) changes in the distribution of the input examples without a concept change (virtual drifts), (5) high dimensional spaces with padding variables, and (6) non-linear decision bounds. Therefore, we selected the following problems:

1. The rotating hyperplane problem.

2. The SEA problem.

3. The SEA problem with varying noise levels.

4. The SEA problem with virtual drifts.

5. The SEA problem with padding variables under high dimensional spaces.

6. The SEA problem with non-linearities.

Each one of these problems is presented in what follows.

4.4.1 The Rotating Hyperplane Problem

The first benchmark problem used to test SNCS is the rotating hyperplane problem, proposed by Hulten et al. (2001). This problem contains abrupt concept changes and noisy inputs. In this problem there is a multi-dimensional space divided by a hyperplane and where examples are labelled according to the position with respect to this hyperplane: if an example is below it, this example will be labelled as positive {+}, and negative {-} otherwise. Every certain number of time steps this hyperplane rotates, resulting in an abrupt concept change. There is, also, a certain amount of noise inside the training data. There are several variants of this environment (Fan, 2004, Núñez et al., 2007). In our experiment we used the configuration described in what follows.

The problem consists of three continuous variables $\{x_1,\ x_2,\ x_3\}$ ranging in $[0,1]$, from which only x_1 and x_2 are relevant to the problem. The hyperplane is defined by Equation 4.15:

$$\sum_{i=1}^{2} w_i \cdot x_i = w_0, \tag{4.15}$$

where w_j, $j \in \{0,1,2\}$ are the weights used to rotate the hyperplane. In each concept these values change as follows: in the first concept weights have the values $\{0.5, 0, 1\}$, that is, in the first concept $w_0 = 0.5, w_1 = 0$ and $w_2 = 1$. In the rest of concepts weight values are, respectively: $\{0.5, 0.5, 0.5\}$, $\{0.5, 1, 0\}$, $\{0, -1, 1\}$, $\{0.5, 0, 1\}$, $\{0.5, 0.5, 0.5\}$, $\{0.5, 1, 0\}$, and finally $\{0, -1, 1\}$. Thus, if an example holds the relation $\sum_{i=1}^{2} w_i \cdot x_i > w_0$, it will be labelled as positive; otherwise it will be labelled as negative.

The data stream consist of 80 000 data samples and the concept is changed every 10 000. Thus, the system has 10 000 randomly created training instances per concept, and those are affected by 10% of noise. The pseudo-random generator uses a uniform distribution.

To test the models, independent, noise-free sets consisting of 2 000 test examples are generated per concept. Test sets have the same number of positive labels than negative ones (i.e., classes are *balanced*). As before, the pseudo-random generator uses a uniform distribution.

The learner is trained for 400 data samples and then its model is evaluated by the corresponding test set. This train-test pattern is constantly repeated during the entire problem.

4.4.2 The SEA Problem

When benchmarking systems against concept drift one of the most representative environments, besides the rotating hyperplane problem, is the SEA problem, originally proposed by Street and Kim (2001). The SEA problem has smaller changes in concepts than the rotating hyperplane one, but at different speeds and combining these with noisy inputs. This problem, very similar to the rotating hyperplane one, is defined by an artificial data set consisting of uniformly distributed random points in a continuous three-dimensional feature space represented by variables x_1, x_2 and x_3 ($x_i \in [0,1]$, $i \in \{1,2,3\}$), from which only the first two are relevant to determine the label of the given example and the last one is left. These points are divided into four blocks with different concepts. In each block, a data point belongs to the positive label $\{+\}$ if $x_1 + x_2 < b$ and to the negative label $\{-\}$ otherwise, where b is a threshold value between the two classes, as it will be discussed soon in this subsection. The concept is changed, then, by modifying the value of this last parameter.

The training set consists of 50 000 instances in which the concept is changed every 12 500 by giving b one of the following values: $\{0.8, 0.9, 0.7, 0.95\}$. These data are affected by noise: the label of each training instance is randomly assigned with 10% probability.

The test set consists of 10 000 noise-free instances for each concept, that is, for each one of the four concepts there are 2 500 test instances correctly labelled according to the concept. The test set contains the same proportion of positive labels than negative ones.

Every 500 time steps the system is evaluated using the corresponding test set.

4.4.3 The SEA Problem with Varying Noise Levels

In both the original SEA problem and the rotating hyperplane problem description the system has to confront a certain factor of noise. The noise factor of these problems is, however, constant in the entire experiment. With the objective of analysing the adaptation capabilities of SNCS under a much harder problem, suffering of changes in the noise level, and following the work done by Núñez et al. (2007), the SEA problem is modified such that the level of noise changes with each concept drift. Thus, the percentages of noise level at each concept are: 20%, no noise, 40% and 10% respectively.

4.4.4 The SEA Problem with Virtual Drifts

The concept of *virtual drift* can be understood as a change on the distribution of data categories when there is no change in the concept being studied (Núñez et al., 2007). Following the work done by Núñez et al. (2007), we have extended the SEA problem in such way that, with each concept, two virtual drifts occur. To do so, the training data of each concept are divided into three equally-sized parts using different data distributions: (1) in the first third the values of the features for each instance are uniformly distributed, (2) in the second third these values follows a Normal distribution $N(b/2, b/4)$, and (3) in the last third these values are distributed uniformly again. The training data set has 10% level of noise.

4.4.5 The SEA Problem with Padding Variables under High Dimensional Spaces

Another extension to the original SEA problem is done by adding some extra input variables with the aim of studying how the systems perform with high dimensional spaces with padding variables. Following the work done by Orriols-Puig and Casillas (2010b), the problems SEA5, with five input variables, and SEA7, with seven input variables, are considered. The label of an example is assigned as positive when $\sum_{i=1}^{4} x_i < b$ in SEA5 and when $\sum_{i=1}^{6} x_i < b$ in SEA7; otherwise, the example is labelled negative. The concept is changed in the usual way, that is, every 12 500 data samples, giving b the values $\{1.6, 1.8, 1.4, 1.9\}$ for SEA5 and $\{3, 3.4, 2.8, 3.6\}$ for SEA7. The training data set has 10% level of noise.

4.4.6 The SEA Problem with Non-Linearities

In order to test the response of SNCS under non-linear environments, the original SEA problem is extended by adding an extra input variable (in this problem there are four variables) and performing a non-linear decision boundary when determining the output label. It is performed as follows: an example belongs to the positive label if $x_1^2 + x_2^2 + x_3^2 < b$ and otherwise the example is labelled negative. As it happens with the other SEA variants, the last variable (i.e., x_4) is left. The concept is changed every 12 500 data samples, giving b the values $\{0.6, 1.8, 1.2, 2.0\}$. The training data set has 10% level of noise.

4.4.7 Methodology of Experimentation

To get a reasonable view of how SNCS performs we compared the results obtained by our system with other three competitive learning systems prepared for mining data streams: (1) CVFDT (Hulten et al., 2001), a tree-based algorithm, with a global window of 10 000 examples in the rotating hyperplane problem and with a global window of 12 500 examples in the SEA problem and variants, (2) IBk (Aha et al., 1991), a nearest-neighbour classifier algorithm, with $k = 1$ and a global window of 10 000 examples in the rotating hyperplane problem and with a global window of 12 500 examples in the SEA problem and variants, and (3) Naïve Bayes (NB) (John and Langley, 1995), a statistical classifier algorithm, using the same windowing scheme as IBk. CVFDT is the current state-of-the-art in the data stream mining field and comparing our results with the ones of CVFDT is a must. This algorithm was run using the VFML toolkit (Hulten and Domingos, 2003). On the other hand, IBk using a global window is also very interesting due to its properties: we should expect an optimal behaviour having seen 10 000 examples in the rotating hyperplane problem (and 12 500 in the SEA problem and variants), that is, after receiving the maximum amount of examples IBk will predict the output based on input instances that all belong to the current concept. NB is a well-known algorithm that is able to perform accurately is a variety of situations, including data streams using a windowing scheme.

To work with CVFDT, a previous discretization of numerical features is needed. This was achieved by dividing each variable x_i into 25 bins using Weka (Witten et al., 2011).

SNCS has several configuration parameters that allow to adjust the algorithm to evolve competitive models. In order to select an appropriate configuration for SNCS, we used SEA, SEA with varying noise levels, SEA with virtual drifts, SEA5, SEA7 and SEA with non-linearities for experimenting with the configurations shown in Table 4.1. The values used during the online experimentation with SNCS (referred to as Cd) are based on the LCS literature (the reader is referred to (Orriols-Puig, 2008) for more information), and adjusted by experimenting with the different parameters. In this table, Cd is the default configuration used and $C1$ to $C17$ are the changes with respect to Cd. Finally, the Friedman's test rank (Friedman, 1937) for the results with the configurations is shown (see appendix A for more information about this statistical procedure). To obtain the ranking, SNCS was executed 10 times with a different random seed using the different configurations in each problem and the obtained accuracy results at the end of each concept were averaged.

The Friedman's test rejected the null hypothesis that all the configurations performed

Id.	**Parameters**	**Rank**
Cd	$\alpha = 0.8$, $\beta = 0.09$, $\eta = 0.1$, $\delta = 0.1$, $\omega = 0.5$, $\nu = 5$, $\theta_{GA} = 50$, $\theta_{exp} = 15$, $\theta_{del} = 0.9$, $\varphi = 18$, $\tau = 0.4$ and $Max_{hid} = 10$	4.33
C1	$\theta_{GA} = 100$	9.08
C2	$\theta_{GA} = 25$	8.08
C3	$\theta_{exp} = 7$	10.00
C4	$\theta_{exp} = 30$	8.25
C5	$\theta_{del} = 0.1$	9.83
C6	$\omega = 0.25$	9.42
C7	$\omega = 1$	5.50
C8	$\tau = 0.9$	14.83
C9	$\alpha = 0.3$	5.50
C10	$\beta = 0.9$	14.83
C11	$\beta = 0.3$	12.08
C12	$\varphi = 10$	7.92
C13	$Max_{hid} = 2$	10.00
C14	$\nu = 10$	9.83
C15	$\nu = 2$	10.17
C16	$\delta = 1$	11.42
C17	$\eta = 0.9$	10.58

Table 4.1: Configurations used to test the sensitivity of SNCS to configuration parameters in data stream problems.

the same. Therefore, we applied the Holm's (Holm, 1979) and Shaffer's (Shaffer, 1986) procedures (not shown for brevity). For $\alpha = 0.05$, on average, Cd is significantly better than $C10$, and the rest of configurations are not significantly different from Cd. This issue is logical because the parameter β controls the learning rate of MLPs and setting it too high distorts its generalisation capacities.

The study conducted empirically showed the robustness of SNCS to configuration parameters and that the most critical parameter is the learning rate of the MLP. We acknowledge that better results could be obtained if we tuned SNCS for each particular problem, but as we are interested in robust systems that perform competently on average, we use the default configuration for all the experiments.

After this procedure, SNCS was finally configured using the following parameters (Cd): $\alpha = 0.8$, $\beta = 0.09$, $\eta = 0.1$, $\delta = 0.1$, $\omega = 0.5$, $\nu = 5$, $\theta_{GA} = 50$, $\theta_{exp} = 15$, $\theta_{del} = 0.9$, $\varphi = 18$, $\tau = 0.4$, $Max_{hid} = 10$, and the population size was set to 10 individuals.

We performed ten runs for each algorithm, using each time a different random seed, and the results provided here are the averages of these runs.

Following the recommendations given by Demšar (2006) and Derrac et al. (2011), the accuracy results of the algorithms used in the present analysis were statistically compared using the software freely provided by García and Herrera (2008)[2]. First, the non-parametric Friedman's test (Friedman, 1937) was performed. Then, if the null hypothesis that every classifier performs the same was rejected, the post-hoc Nemenyi's test (Nemenyi, 1964), the Holm's procedure (Holm, 1979) and finally the Shaffer's procedure (Shaffer, 1986) were performed. To run these tests, we only considered the accuracy achieved by each learner at the

[2]`http://sci2s.ugr.es/sicidm`

end of each concept. Again, the reader is referred to the appendix A for more information about these statistical procedures.

Afterwards, SNCS was compared to XCS and UCS using the same procedure. To configure XCS and UCS we followed a similar procedure as with SNCS (not shown for brevity). XCS was configured using the following parameters: $\alpha = 0.1$, $\beta = 0.01$, $\delta = 0.1$, $\nu = 5$, $\epsilon_0 = 0.1$, $\theta_{GA} = 25$, $\theta_{del} = 10$, $\theta_{sub} = 50$, $\tau = 0.4$, $P_\chi = 0.8$, $P_\mu = 0.167$, $m_0 = 1.0$, $r_0 = 1.0$, and the population size was set to 6400 individuals in all the problems. UCS used the same configuration except for the following parameters. $\nu = 10$, $acc_0 = 0.999$, $\theta_{del} = 20$, and $\theta_{sub} = 25$, and the population size was set to 2000 individuals in all the problems except for the SEA5, SEA7 and SEA with non-linearities, which was set to 4000.

4.4.8 Analysis of the Results

In the following we show and carefully analyse the behaviour of SNCS for each one of the aforementioned problems. First, we analyse the behaviour of SNCS against the state-of-the-art data stream miners. Afterwards, and for each problem, we compare the results of the Michigan-style LCSs against themselves in a merely qualitative way.

Analysis of the Results on the Rotating Hyperplane Problem

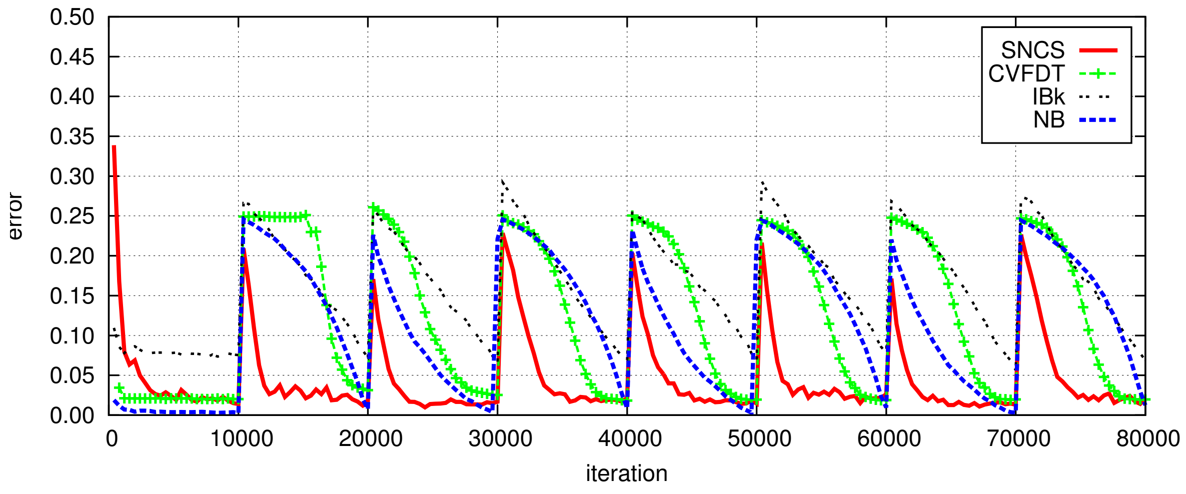


FIGURE 4.3: Rotating hyperplane problem: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 10 000 data samples there is a concept drift. Results are averages of ten runs.

In this problem, the reaction capacity of SNCS under abrupt concept changes and noisy inputs is analysed. There are eight abrupt concept drifts, each lasting 10 000 data samples each and those are affected by a 10% of noise. Figure 4.3 shows the test error obtained by SNCS, CVFDT, IBk, and NB. The horizontal axis represents the number of data samples and the vertical axis the percentage of misclassification error. Each vertical line references to a concept change. Here we can see different phenomenons: (1) the time the algorithms take to "learn" at the first stages of the problem, (2) the reaction of the algorithms to sudden and abrupt concept drifts, and (3) the reaction of the algorithms to the noisy data. These three aspects are elaborated in what follows.

- **Time required to learn at the first stages of the problem**. While NB, CVFDT, and IBk start with a low error rate (less than 2% for NB, less than 5% for CVFDT and less than 11% for IBk), SNCS starts at 34%. This phenomenon is due to the time required by the MLP to learn the first rough approximation of the hidden patterns in the data stream. However, the learning curve is very steep and at the end of the first concept SNCS reaches similar error rates as CVFDT, near 2%. Notice the impressive achievement of NB, being below 0.3%.

- **Reaction to concept drifts**. We observe the same pattern around the problem: a high error peak followed by a recovery time. IBk has, by far, the worst reaction capacity, being almost linear, starting over 25% and ending at 8% approximately. CVFDT results are better than the ones of IBk, showing a soft, non-linear error curve, recovering much quicker than the former. CVFDT ends around at a impressive 2.5%. SNCS has the best reaction capacity showing a very quick recovery time, being much better than the ones shown by NB, CVFDT and IBk. SNCS shows a narrow peak in error rate and then quickly moving to 5%, ending below 2% approximately. On the other hand, NB has the lowest error rate, being below 1% at the end of the concept, although its reaction capacity is not than good.

- **Reaction to noisy data**. Another key issue is the robustness to noisy data. IBk shows a poor performance, being close to the 10% of noise added to the problem. NB, CVFDT and SNCS show a good robustness to noisy inputs, being far below this 10%.

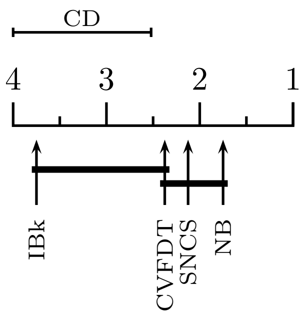


Figure 4.4: Nemenyi's test at $\alpha = 0.1$ on the rotating hyperplane problem. Classifiers that are not significantly different are connected.

- **Statistical tests**. Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 21 degrees of freedom equal to 5.8736 and a computed p-value

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**IBk vs. NB**	3.0984	0.0019	0.0083	0.0083
5	IBk vs. SNCS	2.5174	0.0118	0.0100	0.0167
4	IBk vs. CVFDT	2.1301	0.0333	0.0125	0.0167
3	CVFDT vs. NB	0.9682	0.3329	0.0167	0.0167
2	NB vs. SNCS	0.5809	0.5613	0.0250	0.0250
1	CVFDT vs. SNCS	0.3873	0.6985	0.0500	0.0500

TABLE 4.2: Holm / Shaffer Table for $\alpha = 0.05$ on the rotating hyperplane problem. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

of 0.0045. Therefore, we applied post-hoc tests, starting with Nemenyi's test, shown in Figure 4.4. We can see that both SNCS and NB outperformed IBk, however the tests cannot reject the hypothesis that SNCS, CVFDT and NB perform the same on average, at $\alpha = 0.1$. These results were confirmed by the Holm's and the Shaffer's procedures, as is shown in Table 4.2. Nevertheless note that this conclusion is reached by only considering the accuracy at the end of each concept. Figure 4.3 clearly shows the superiority of SNCS through all the learning process.

- **Conclusions**. We can conclude that SNCS has a good performance which specially demonstrates an outstanding reaction time, one of the most interesting characteristics of an online learner.

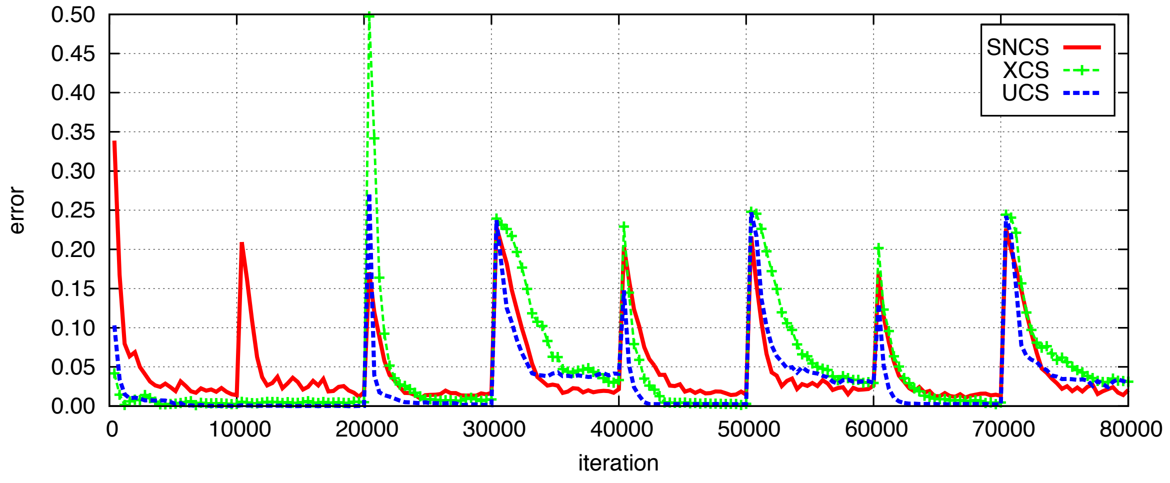


FIGURE 4.5: Rotating hyperplane problem: comparison of the test error achieved by SNCS, XCS and UCS. Every 10 000 data samples there is a concept drift. Results are averages of ten runs.

Figure 4.5 shows the results of the distinct Michigan-style LCSs on the rotating hyperplane problem. Although XCS is very competent in the first two concepts, it degenerates in the rest of the run, having the worst performance of the analysed algorithms. It is worth mentioning the good results shown by UCS, being the best classifier in this particular problem, closely followed by SNCS.

Analysis of the Results on the SEA Problem

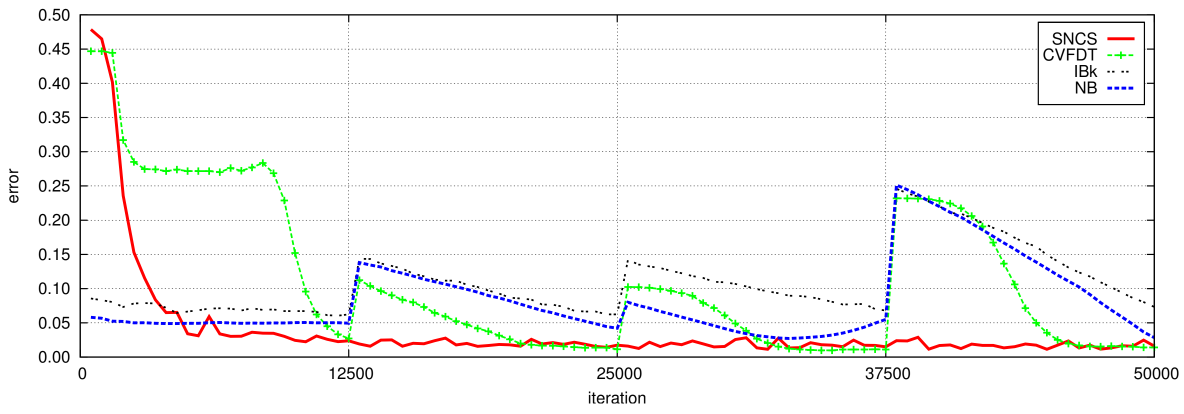


Figure 4.6: SEA problem: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

The SEA problem is the most widely used benchmark problem in data streams. It has smoother concept changes than the rotating hyperplane problem, but they change at different speeds. It has noisy inputs as well. Figure 4.6 shows the test error obtained by SNCS, CVFDT, IBk and NB. The horizontal axis represents the number of data samples and the vertical axis the percentage of misclassification error. As in the rotating hyperplane problem, we are interested in three different aspects: (1) the time the algorithms take to “learn” at the first stages of the problem, (2) the reaction of the algorithms to concept drifts, and (3) the reaction of the algorithms to the noisy data. These aspects are elaborated in the following.

- **Time required to learn at the first stages of the problem**. As in the previous experiment, SNCS starts with a high error rate and then it quickly recovers, finishing below 4% error in the first concept. CVFDT has a similar behaviour but the error curve is somehow more smoother and it takes more time to get below 5% error. IBk behaves in the same way as in the hyperplane problem, starting with a low error rate. NB starts and ends the first concept at 5% error.

- **Reaction to concept drifts**. The reaction capacity of SNCS in this problem is surprisingly good: it is not affected by concept drifts and does not show the error peaks that characterises changes in concept. That is due to the changes in concept are very gentle and the MLPs used by SNCS allow this technique to easily adapt to them. CVFDT, IBk and NB do not show this effect on reaction capacity because of they internal knowledge representation do not allow it. More precisely, CVFDT behaves relatively well, showing error peaks and recovering quickly. IBk, again, has the worst reaction capacity due to the effects of maintaining outdated examples, closely follower by NB.

- **Reaction to noisy data**. SNCS, CVFDT and NB show a high robustness to noise, ending with error rates far below the 10% noise.

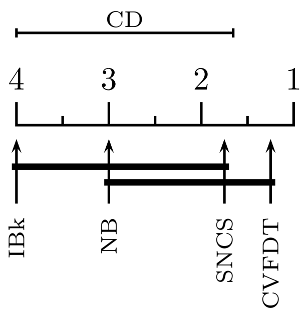


FIGURE 4.7: Nemenyi's test at $\alpha = 0.05$ on the SEA problem. Classifiers that are not significantly different are connected.

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**IBk vs. CVFDT**	3.0125	0.0026	0.0083	0.0083
5	IBk vs. SNCS	2.4647	0.0137	0.0100	0.0167
4	CVFDT vs. NB	1.9170	0.0552	0.0125	0.0167
3	NB vs. SNCS	1.3693	0.1709	0.0167	0.0167
2	IBk vs. NB	1.0954	0.2733	0.0250	0.0250
1	CVFDT vs. SNCS	0.5477	0.5839	0.0500	0.0500

TABLE 4.3: Holm / Shaffer Table for $\alpha = 0.05$ on the SEA problem. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

- **Statistical tests**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 9 degrees of freedom equal to 37 and a computed p-value of $2.1717 \cdot 10^{-5}$. Therefore, we applied post-hoc tests, starting with Nemenyi's test, shown in Figure 4.7. From this figure we can observe that CVFDT outperformed IBk, however SNCS, CVFDT and NB are not significantly different at $\alpha = 0.05$. These results were confirmed by the Holm's and the Shaffer's procedures, as it is shown in Table 4.3.

- **Conclusions**. SNCS has a remarkable robustness against concept drifts with varying speeds. Also, it may seem surprising that statistical tests do not reveal this effect, and that is because we measured the final values at each concept. As is shown in Figure 4.6 we can observe that, in fact, SNCS behaves better than CVFDT on the average.

Figure 4.8 shows the results of the distinct Michigan-style LCSs on the SEA problem. Both XCS and UCS show a competitive behaviour, being close to SNCS results. Notice that, although the similarities in the curves, XCS requires three times more population than UCS

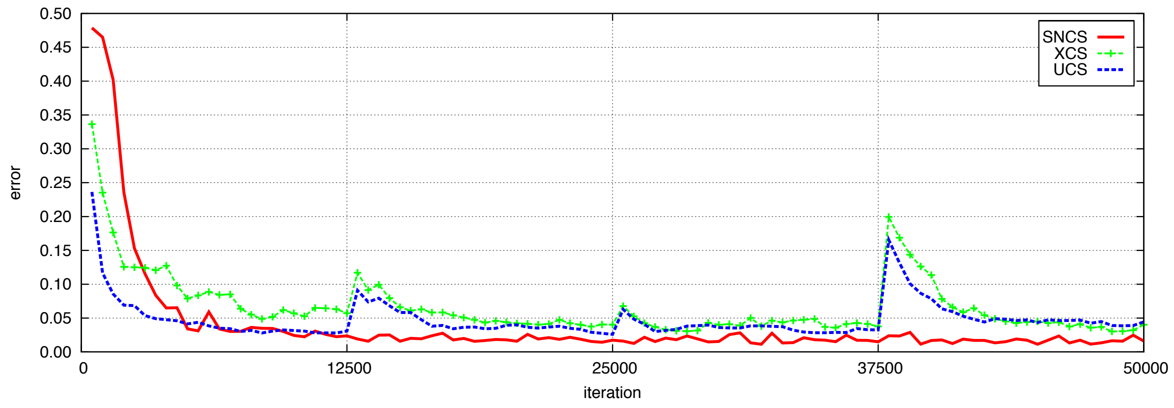


Figure 4.8: SEA problem: comparison of the test error achieved by SNCS, XCS and UCS. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

to solve the problem. The supervised architecture of UCS has a clear advantage in this kind of environments by making the algorithm faster and also by requiring less memory. If we set the population of XCS to 2000 individuals, as in the case of UCS, the error curve grows dramatically (not shown for brevity).

Analysis of the Results on the SEA Problem with Varying Noise Levels

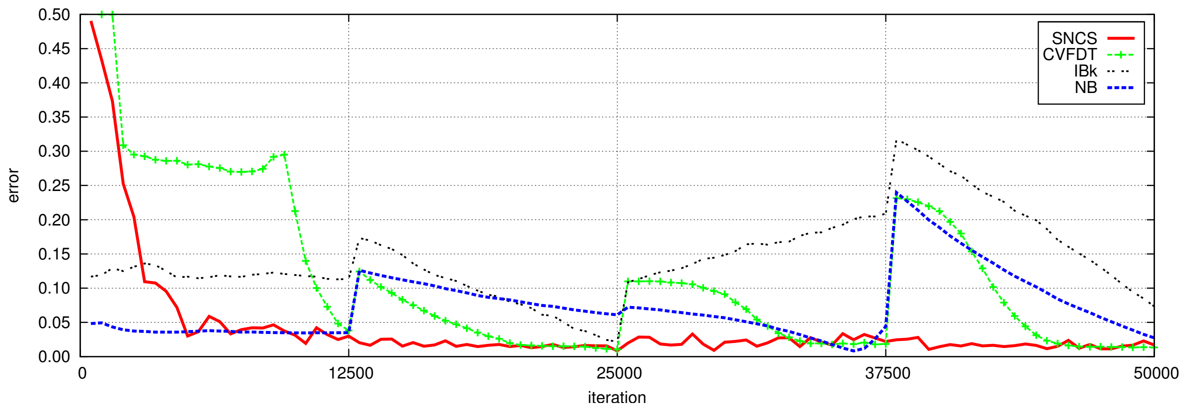


Figure 4.9: SEA problem with varying noise levels for each concept: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

The noise factor of the two latter problems laid constant in the entire runs. The SEA problem with varying noise levels has the added difficulty of varying noise levels (20%, 0%, 40% and 10% respectively for each concept) and hence it is much harder for the learners. Figure 4.9 shows the results of varying the level of noise in each concept.

- **Reaction to noise**. In this particular case we can observe than both SNCS, CVFDT and NB behave with high robustness, being almost unaffected by the distorting effects of noise. This behaviour is specially noticeable under the third concept, where the noise raises up to 40%. This is a very desirable feature for a learner.

- **Reaction to concept drifts**. SNCS remained unaffected by concept drifts. In the other side, IBk has the worst reaction of the classifiers analysed. This is due to the sensitivity to noise of distance measuring.

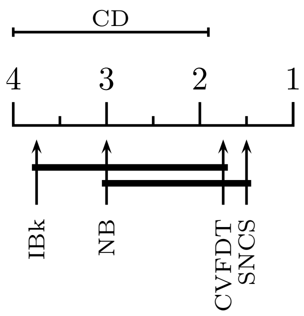


Figure 4.10: Nemenyi's test at $\alpha = 0.1$ on the SEA problem with varying noise levels. Classifiers that are not significantly different are connected.

Table 4.4: Holm / Shaffer Table for $\alpha = 0.1$ on the SEA problem with varying noise levels. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**IBk vs. SNCS**	2.4647	0.0137	0.0167	0.0167
5	IBk vs. CVFDT	2.1909	0.0285	0.0200	0.0333
4	NB vs. SNCS	1.6432	0.1003	0.0250	0.0333
3	CVFDT vs. NB	1.3693	0.1709	0.0333	0.0333
2	IBk vs. NB	0.8216	0.4113	0.0500	0.0500
1	CVFDT vs. SNCS	0.2739	0.7842	0.1000	0.1000

- **Statistical tests**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 9 degrees of freedom equal to 6.2308 and a computed p-value of 0.0141. Therefore, we applied post-hoc tests, starting with Nemenyi's test as shown in Figure 4.10. This test show that SNCS outperformed IBk, however SNCS, CVFDT and NB are not significantly different at $\alpha = 0.1$. We applied Holm's and Shaffer's tests (see the results in Table 4.4). These tests confirmed Nemenyi's one.

- **Conclusions**. Both SNCS and CVFDT show high robustness and it is worth noticing that SNCS remains unaffected by concept changes under high noise rates.

Figure 4.11 shows the results of the distinct Michigan-style LCSs on the SEA problem with varying noise levels for each concept. Both XCS and UCS show high robustness against noise, being almost unaffected by it.

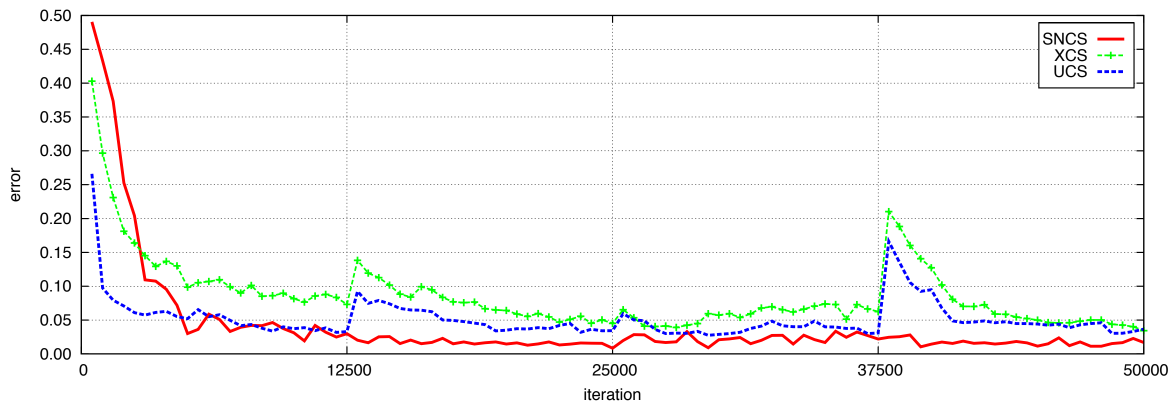


Figure 4.11: SEA problem with varying noise levels for each concept: comparison of the test error achieved by SNCS, XCS and UCS. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

Analysis of the Results on the SEA Problem with Virtual Drifts

Modifying the SEA data set, we introduced virtual drifts in order to study the behaviour the algorithms analysed in this chapter, that is, we are interested in how the algorithms behave when there are changes in the distribution of data and there is no change in the concept being studied. The results are depicted in Figure 4.12.

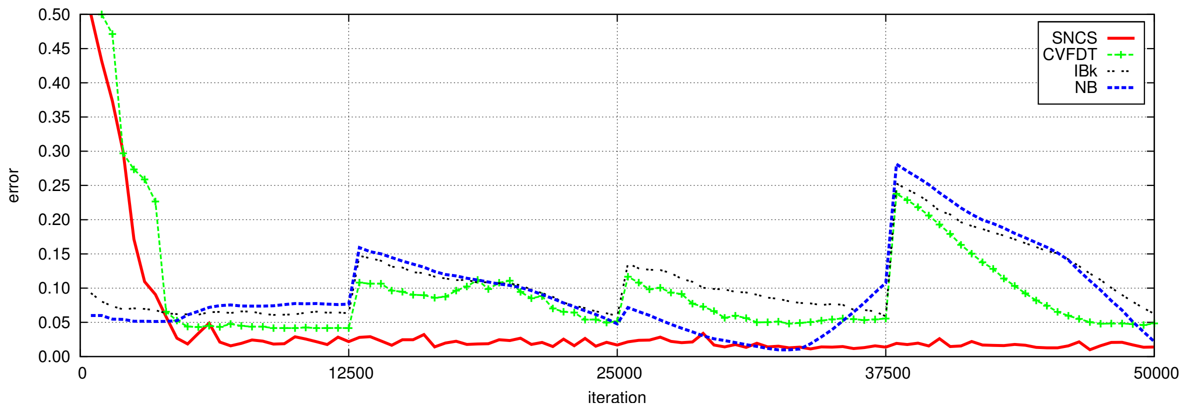


Figure 4.12: SEA problem with virtual drifts: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

- **Reaction to virtual drifts.** SNCS show high robustness to concept drift and virtual drift, being unaffected. It is worth noticing that, in this experiment, CVFDT is clearly affected by virtual drift in two aspects: almost never lowers 5% error and the curve is somehow less smooth than in the other experiments. IBk is affected too, but its effects

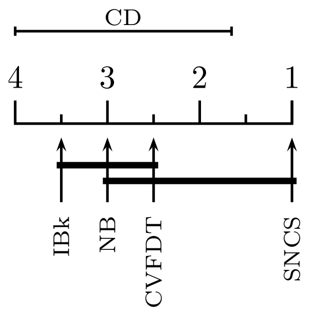


FIGURE 4.13: Nemenyi's test at $\alpha = 0.05$ on the SEA problem with virtual drifts. Classifiers that are not significantly different are connected.

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**IBk vs. SNCS**	2.7386	0.0062	0.0083	0.0083
5	NB vs. SNCS	2.1909	0.0285	0.0100	0.0167
4	CVFDT vs. SNCS	1.6432	0.1003	0.0125	0.0167
3	IBk vs. CVFDT	1.0954	0.2733	0.0167	0.0167
2	IBk vs. NB	0.5477	0.5839	0.0250	0.0250
1	CVFDT vs. NB	0.5477	0.5839	0.0500	0.0500

TABLE 4.5: Holm / Shaffer Table for $\alpha = 0.05$ on the SEA problem with virtual drifts. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

are more gentle than in the case of CVFDT. NB is affected in a similar way as IBk.

- **Statistical tests**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 9 degrees of freedom equal to 7 and a computed p-value of 0.0099. Therefore, we applied post-hoc testes. Nemenyi's test identified a significant difference between SNCS and IBk as Figure 4.13 shows this graphically. These results were confirmed by the Holm's and the Shaffer's procedures, as is shown in Table 4.5.

- **Conclusions**. This experiment confirms that SNCS is not only robust against noise, but against concept and virtual drifts. The flexibility of the representation used by this technique made all this possible.

Figure 4.14 shows the results of the distinct Michigan-style LCSs on the SEA problem with virtual drifts. Virtual drifts seems to affect little to XCS and UCS as the experiment supports. As in the previous experiments, XCS requires of a larger population to achieve lower error rates.

Analysis of the Results on the SEA Problem with Padding Variables under High Dimensional Spaces

Despite the fact that the original SEA problem is a representative environment for data streams, we were also interested in the response of the algorithms compared here under high

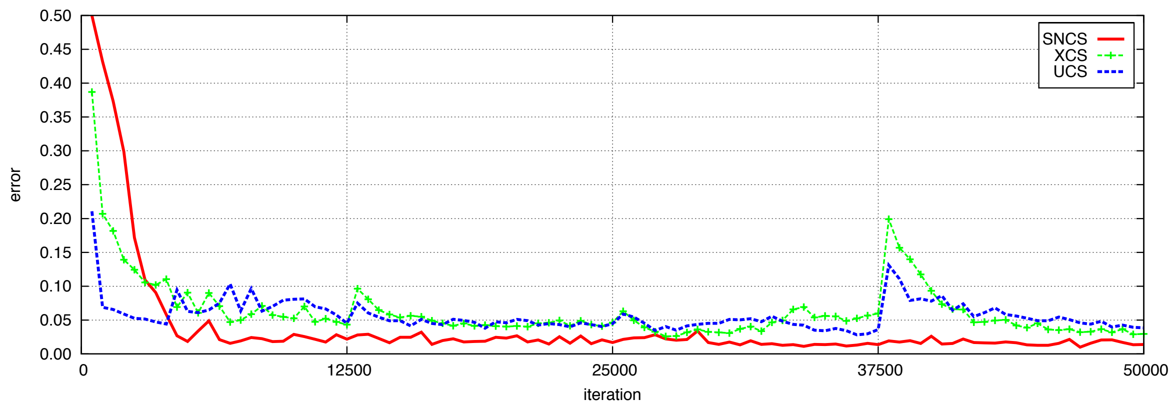


FIGURE 4.14: SEA problem with virtual drifts: comparison of the test error achieved by SNCS, XCS and UCS. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

dimensional stress situations, where each learner has to discover that certain variables are just for *padding*: they do not participate to determine the label (there are spurious relationships between these padding variables and the label values). Therefore, the more padding variables, the more difficult the problem. For this reason we implemented SEA5 and SEA7 problems, increasing the number of input variables to five and seven, respectively. Figure 4.15 and Figure 4.16 show the results of these problems, respectively.

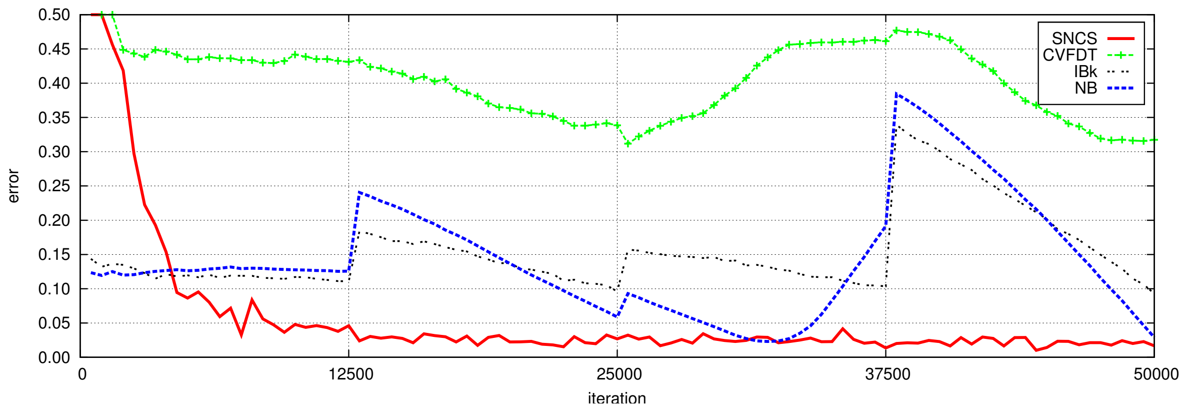


FIGURE 4.15: SEA5 problem: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

- **SNCS behaviour in SEA5**. SNCS shows a remarkable behaviour: after the initial training phase during the first concept period, which is larger than the other experiments, its mean error rapidly falls below 5%.
- **SNCS behaviour in SEA7**. The behaviour of SNCS is similar as in SEA5, but the error curve is more abrupt. After this initial learning, the error rate keeps moving around a 5%, a low error rate for this problem.

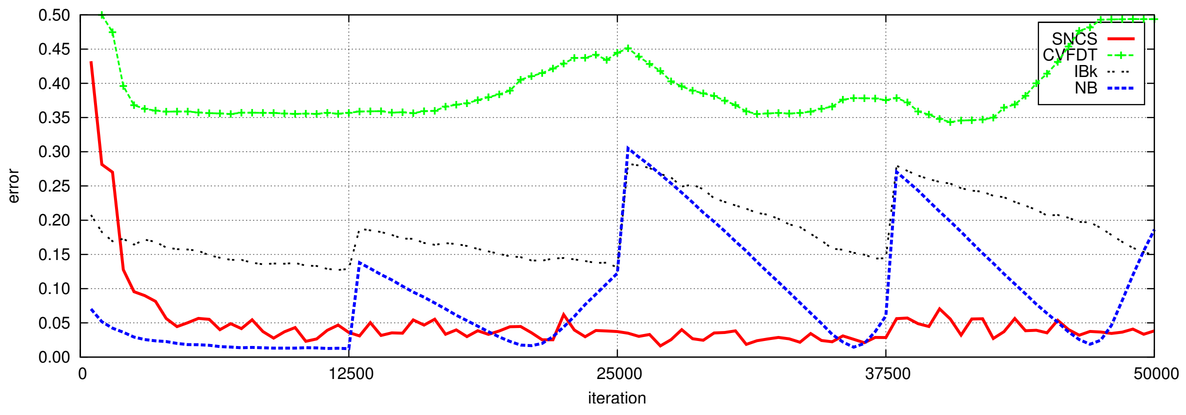


FIGURE 4.16: SEA7 problem: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

- **CVDT behaviour in SEA5**. We observe an unexpected behaviour of CVFDT: it had a considerable amount of error, never falling below 30% in SEA5. We hypothesise that the algorithm cannot identify the padding variables and that is misguided by spurious relationships between these variables and the label values.
- **CVDT behaviour in SEA7**. The behaviour of CVDT is even worse in SEA7: it never falls below 35% error.
- **IBk behaviour in both problems**. IBk shows a more discrete behaviour in both SEA5 and SEA7 problems. Its accuracy degrades only a tiny fraction with respect to the original SEA environment.
- **NB behaviour in both problems**. NB, on the other hand, seems to be more affected by SEA5 than by SEA7. In the former problem we can observe a distinctive curve in the third concept. In the latter one, we can observe these curves from the second concept to the fourth one. This phenomenon seems to be accentuated due to the windowing mechanism.

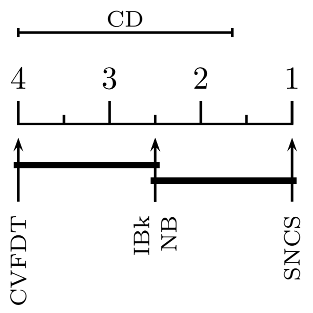


FIGURE 4.17: Nemenyi's test at $\alpha = 0.05$ on the SEA5 problem. Classifiers that are not significantly different are connected.

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**CVFDT vs. SNCS**	3.2863	0.0010	0.0083	0.0083
5	IBk vs. CVFDT	1.6432	0.1003	0.0100	0.0167
4	IBk vs. SNCS	1.6432	0.1003	0.0125	0.0167
3	CVFDT vs. NB	1.6432	0.1003	0.0167	0.0167
2	NB vs. SNCS	1.6432	0.1003	0.0250	0.0250
1	IBk vs. NB	0.0000	1.0000	0.0500	0.0500

Table 4.6: Holm / Shaffer Table for $\alpha = 0.05$ on the SEA5. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

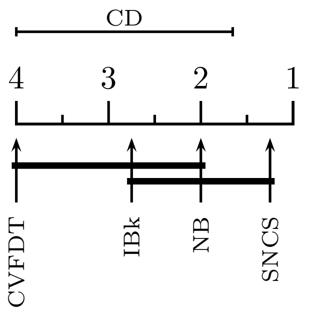


Figure 4.18: Nemenyi's test at $\alpha = 0.05$ on the SEA7 problem. Classifiers that are not significantly different are connected.

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**CVFDT vs. SNCS**	3.0125	0.0026	0.0083	0.0083
5	CVFDT vs. NB	2.1909	0.0285	0.0100	0.0167
4	IBk vs. SNCS	1.6432	0.1003	0.0125	0.0167
3	IBk vs. CVFDT	1.3693	0.1709	0.0167	0.0167
2	IBk vs. NB	0.8216	0.4113	0.0250	0.0250
1	NB vs. SNCS	0.8216	0.4113	0.0500	0.0500

Table 4.7: Holm / Shaffer Table for $\alpha = 0.05$ on the SEA7. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

- **Statistical tests in SEA5**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 9 degrees of freedom equal to 27 and a computed p-value of $7.8384 \cdot 10^{-5}$. Therefore, we applied post-hoc testes. Therefore, we applied Nemenyi's test (Figure 4.17) and then Holm's and Shaffer's procedures (Table 4.6) for the SEA5 problem.

- **Statistical tests in SEA7**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 9 degrees of freedom equal to 14.1429 and a computed p-value of 0.0009. Therefore, we applied Nemenyi's test (Figure 4.18) and then Holm's and Shaffer's procedures (Table 4.7) for the SEA7 problem.

- **Conclusions**. From SEA5 and SEA7 experiments we can conclude that SNCS successfully handles high dimensional spaces when concept drift occurs, showing a remarkable behaviour when the concept changes. This effect is due to SNCS's flexible representation and the LCS paradigm in which it is embedded.

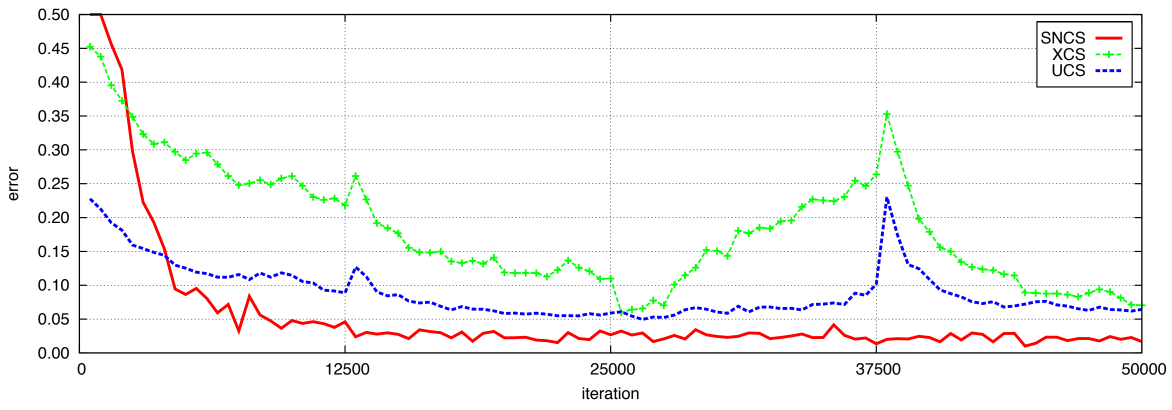


FIGURE 4.19: SEA5 problem: comparison of the test error achieved by SNCS, XCS and UCS. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

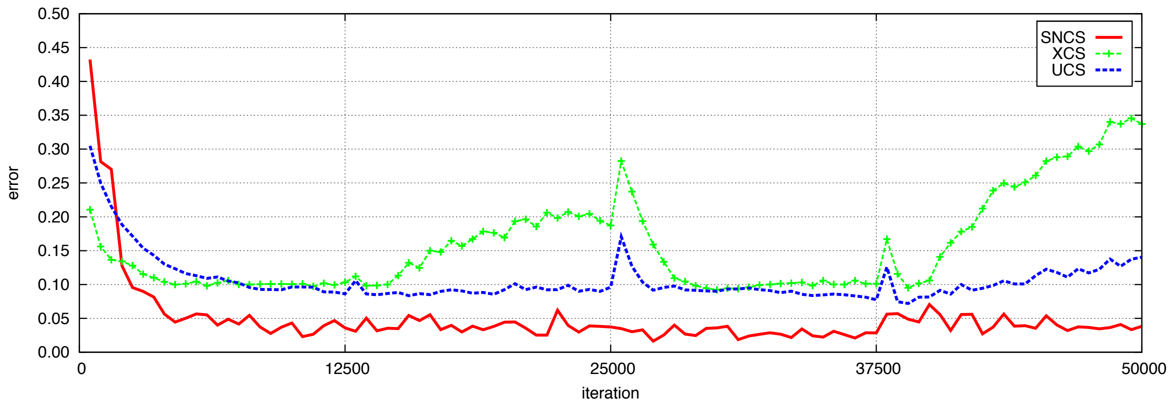


FIGURE 4.20: SEA7 problem: comparison of the test error achieved by SNCS, XCS and UCS. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

Figure 4.19 shows the results of the distinct Michigan-style LCSs on the SEA5 problem and Figure 4.20 the results on SEA7. In both SEA5 and SEA7 XCS shows a degradation in performance with respect to UCS and SNCS. UCS shows a good performance in both SEA5 and SEA7, having an error rate below the 10% in general.

Analysis of the Results on the SEA Problem with Non-Linearities

In the previous SEA problem and variants the decision boundaries were lineal, that is, an hyperplane could be enough to separate the two underlying classes. However, real-world problems are, often, non-lineal. Thus, we extended the SEA problem in order to incorporate non-linearities in the decision boundaries. The results are depicted in Figure 4.21.

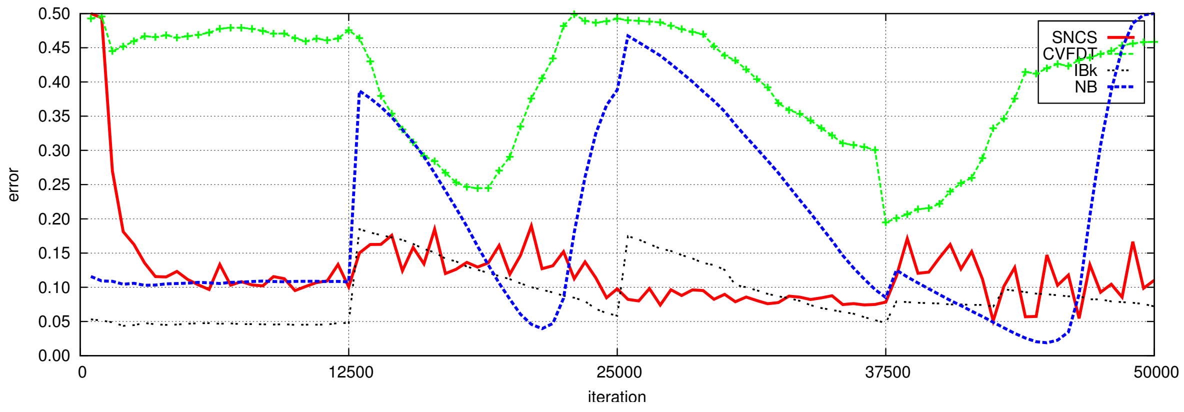


Figure 4.21: SEA problem with non-linearities: comparison of the test error achieved by SNCS, CVFDT, IBk (with $k = 1$), and NB. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

- **Behaviour with non-linearities**. This problem is quite tough, especially for CVFDT and NB. The non-linearities among multiple dimensions in the case of CVFDT and the windowing mechanism in NB cause the low accuracy in the results. We can observe that this problem is complicated for SNCS too, ending over 10% error in each concept. IBk have the best results in this problem showing a good behaviour, ending over 5% error in each concept. This algorithm has the advantage of not depending on decision boundaries.

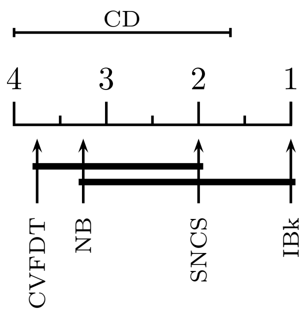


Figure 4.22: Nemenyi's test at $\alpha = 0.05$ on the SEA problem with non-linearities. Classifiers that are not significantly different are connected.

i	Algorithms	z	p	p-Holm	p-Shaffer
6	**IBk vs. CVFDT**	3.0125	0.00259	0.0083	0.0083
5	IBk vs. NB	2.4647	0.01371	0.0100	0.0167
4	CVFDT vs. SNCS	1.9170	0.05523	0.0125	0.0167
3	NB vs. SNCS	1.3693	0.17090	0.0167	0.0167
2	IBk vs. SNCS	1.0954	0.27332	0.0250	0.0250
1	CVFDT vs. NB	0.5477	0.58388	0.0500	0.0500

TABLE 4.8: Holm / Shaffer Table for $\alpha = 0.05$ on the SEA problem with non-linearities. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

- **Statistical tests**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 3 and 9 degrees of freedom equal to 37 and a computed p-value of $2.1717 \cdot 10^{-5}$. Therefore, we applied post-hoc Nemenyi's test as is shown in Figure 4.22 and then Holm's and Shaffer's procedures, shown in Table 4.8. These tests confirmed that the difference in IBk's performance against CVFDT's one is significative, but these cannot reject the null hypothesis over SNCS, CVFDT and NB.

- **Conclusions**. The experiments performed in this section have served to confirm the high adaptivity of SNCS as a data stream miner, being robust to noise and to concept drift and showing a competitive behaviour, which was defined as a primary objective for SNCS.

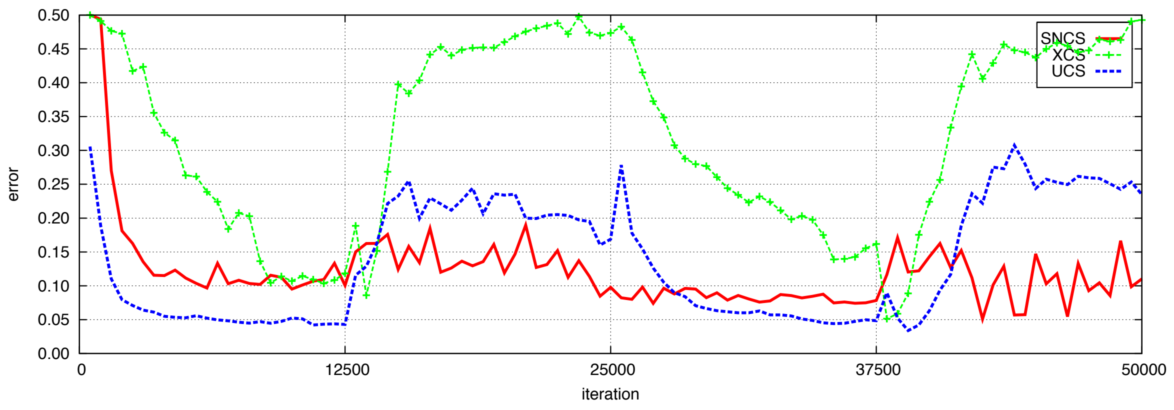


FIGURE 4.23: SEA problem with non-linearities: comparison of the test error achieved by SNCS, XCS and UCS. Every 12 500 data samples there is a concept drift. Results are averages of ten runs.

Figure 4.23 shows the results of the distinct Michigan-style LCSs on the SEA problem with non-linearities. While UCS showed a robust behaviour against non-linearities, XCS oscillated between low error rates and high ones, showing that this algorithm has issues with non-linear boundaries.

4.4.9 Summary and Discusion

Table 4.9 reflects the distinct positions in the ranking for each algorithm according to Friedman's test. SNCS ranked as first in four problems and as second—very close to the first—in the three other problems, showing that SNCS excelled in data stream problems compared with methods specially designed to deal with data streams. These results support the robustness of SNCS, which appears as a competent learner for online data classification, specially under environments with high volumes of noise and virtual drifts. Also, SNCS showed a competent behaviour under problems with high dimensional spaces.

Problem	**Position**			
	CVFDT	**NB**	**IBk**	**SNCS**
Rotating Hyperplane	3	**1**	4	2
SEA	**1**	3	4	2
SEA with Varying Noise Levels	2	3	4	**1**
SEA with Virtual Drifts	2	3	4	**1**
SEA5	4	2	3	**1**
SEA7	4	2	3	**1**
SEA with Non-linearities	4	3	**1**	2

Table 4.9: Summary of the experiments with data streams. For each problem it shows its description and the different positions in the ranking for each algorithm according to Friedman's test. Lower values are better.

Compared with XCS and UCS, the current state-of-the-art Michigan-style LCSs, SNCS reacted much quicker to concept changes and noisy inputs and requiring much less population, supporting the hypothesis that its learning architecture is more adequate for handling data stream problems.

4.5 Experiments on Real-world Problems

SNCS demonstrated a competitive behaviour under concept-changing on-line environments, showing a high adaptivity to sudden changes and being robust to noisy inputs. In this section we want to go further and experiment with a set of real-world problems, without concept drift, to show that the accuracy of SNCS in these kind of problems is at the same level as the accuracy of the models created by some of the most significant machine learning techniques specially designed to mine static data. We show, in what follows, the experimental methodology used and then the results obtained.

4.5.1 Methodology

We took 30 real-world data sets from the UCI repository (Bache and Lichman, 2013) and from Keel (Alcalá-Fdez et al., 2009), except for μCA and *tao*, which are taken from a local repository. These have different characteristics, which are summarised on Table 4.10.

Id.	**#Inst.**	**#Feat.**	**#Num.**	**#Nom.**	**#Class.**
ann	898	38	6	32	6
aut	205	25	15	10	7
ban	539	19	19	0	2
bpa	345	6	6	0	2
col	368	22	7	15	2
gls	214	9	9	0	7
h-c	303	13	6	7	5
h-s	270	13	13	0	2
hov	435	16	0	16	2
ion	351	34	34	0	2
irs	150	4	4	0	3
k-p	3196	36	0	36	2
lab	57	16	8	8	2
lym	148	18	3	15	4
mam	830	5	5	0	2
μCA	216	21	21	0	2
mnk	432	6	6	0	2
msm	8124	22	0	22	2
pim	768	8	8	0	2
pri	339	17	0	17	22
son	208	60	60	0	2
tao	1888	2	2	0	2
thy	215	5	5	0	3
veh	846	18	18	0	4
vot	435	16	0	16	2
vow	990	13	10	3	11
wav	5000	40	40	0	3
wbc	699	9	9	0	2
wne	178	13	13	0	3
zoo	101	17	1	16	7

TABLE 4.10: Summary of the properties of the data sets used. The columns describe: the identifier of the data set (Id.), the number of instances (#Inst.), the total number of features (#Feat.), the number of numeric features (#Num.), the number of nominal features (#Nom.), and the number of classes (#Class.).

SNCS was configured with the following parameters: $\alpha = 0.8$, $\beta = 0.09$, $\eta = 0.1$, $\delta = 0.1$, $\omega = 0.5$, $\nu = 10$, $\theta_{GA} = 50$, $\theta_{exp} = 100$, $\theta_{del} = 0.92$, $\varphi = 15$, $\tau = 0.4$, $Max_{hid} = 100$, the number of iterations was set to 100 000 and the population size was set to 100 individuals.

Following the experimental methodology used earlier, we wanted to compare SNCS with some of the most significant machine learning techniques (Wu et al., 2007), so we chose: (1) the decision tree C4.5 (Quinlan, 1993), (2) the sequential minimal optimization (SMO) (Platt, 1998) support vector machine (Vapnik, 1995), (3) the multilayer perceptron (MLP) (Rumelhart et al., 1986, Widrow and Lehr, 1990) neural network, (4) the statistical classifier Naïve Bayes (NB) (John and Langley, 1995), and (5) the instance-based classifier IBk (Aha et al., 1991). We used the implementations provided by Weka (Witten et al., 2011).

In the case of SMO we chose a polynomial kernel with an exponent out of the following $\{1, 3, 5\}$ and a radial basis function kernel. To select the appropriate polynomial kernel, we used the Friedman's statistical test with the aforementioned data sets and configurations for

Id.	**C4.5**			**IB3**			**NB**			**SMO$_{p1}$**			**SMO$_{rbf}$**			**MLP**			**SNCS**		
	A	K	F	A	K	F	A	K	F	A	K	F	A	K	F	A	K	F	A	K	F
ann	98.57	96.35	98.50	97.29	93.17	97.25	86.59	72.46	88.40	97.46	92.84	97.10	91.81	77.31	89.40	98.91	96.92	98.75	99.37	98.42	99.41
aut	81.76	79.33	84.05	67.22	56.70	66.95	57.41	44.27	55.45	71.32	60.11	68.95	44.78	19.79	32.60	72.92	71.82	78.30	73.59	66.14	72.85
ban	72.91	36.67	69.15	63.82	21.32	61.85	62.26	20.29	61.40	65.19	23.76	62.75	57.88	0.00	42.40	68.24	35.16	68.45	65.29	27.14	69.70
bpa	65.83	29.45	65.85	62.49	23.08	62.50	54.87	15.45	54.25	57.97	0.40	42.85	57.97	0.00	42.50	67.58	37.15	69.40	72.65	42.68	76.08
col	85.16	67.42	85.00	80.95	60.23	81.65	78.70	55.07	78.85	82.66	63.56	83.05	84.02	64.17	83.55	81.12	59.25	81.05	80.75	58.57	77.11
gls	68.74	54.85	66.45	70.19	59.26	69.40	47.34	32.23	45.55	57.80	37.33	52.45	35.51	0.00	18.60	63.61	56.06	66.80	68.49	53.27	68.29
h-c	76.93	47.54	74.05	81.81	63.40	81.85	83.33	65.98	83.15	83.86	65.45	82.90	82.87	64.37	82.35	81.72	64.72	82.55	81.90	63.41	80.68
h-s	78.15	57.03	78.80	79.11	56.22	78.35	83.59	67.31	83.85	83.89	66.50	83.45	82.78	65.15	82.80	79.74	60.38	80.35	80.48	60.33	78.90
hov	96.32	92.72	96.55	92.64	84.25	92.45	90.02	79.76	90.30	95.84	91.32	95.90	94.74	89.16	94.85	94.71	87.97	94.25	94.90	89.31	95.47
ion	89.74	80.36	91.05	86.01	66.60	84.90	82.16	63.41	82.60	88.06	74.16	88.35	75.90	38.63	71.75	90.46	80.35	91.05	90.08	77.74	91.83
irs	94.73	92.00	94.70	95.20	93.00	95.30	95.53	93.00	95.30	96.27	95.00	96.70	88.07	89.00	92.65	96.73	96.00	97.35	96.33	94.50	95.29
k-p	99.44	98.90	99.45	96.55	93.37	96.70	87.79	75.51	87.80	95.79	91.41	95.75	91.34	82.72	91.40	99.13	98.62	99.30	98.24	96.48	98.20
lab	78.77	45.06	75.05	92.81	88.44	94.75	93.51	81.26	91.35	92.98	88.31	94.70	64.91	0.00	51.10	90.97	69.19	86.00	76.80	45.01	82.19
lym	75.81	58.43	77.95	81.69	64.10	81.10	83.11	69.01	83.65	86.49	75.47	87.05	80.27	62.18	79.20	82.26	68.49	83.00	84.50	69.88	85.21
mam	83.98	66.76	83.40	77.11	53.34	76.70	82.52	64.15	82.05	79.71	59.15	79.50	79.46	59.56	79.60	80.66	61.66	80.85	79.95	59.96	80.08
μCA	61.30	24.15	61.85	66.11	30.43	65.60	65.28	28.59	64.85	67.22	29.55	65.40	59.72	13.66	54.95	65.55	28.32	64.80	68.14	35.18	65.47
mnk	90.16	73.13	88.40	83.56	54.02	80.30	66.97	0.23	53.85	67.13	0.00	53.90	67.13	0.00	53.90	77.96	54.58	79.55	93.53	85.64	91.98
msm	100.00	100.00	100.00	100.00	100.00	100.00	95.76	91.45	95.70	100.00	100.00	100.00	99.85	99.70	99.90	100.00	100.00	100.00	99.98	99.95	99.97
pim	74.49	40.62	73.25	73.87	41.00	73.55	75.75	46.23	75.80	76.80	45.63	75.80	65.10	0.00	51.30	72.47	44.70	75.15	76.31	47.22	70.79
pri	41.39	33.06	37.45	44.98	35.98	40.30	49.70	43.14	46.30	47.08	38.10	42.30	24.78	0.00	9.80	51.65	31.83	38.35	53.92	43.96	55.44
son	73.61	43.59	71.90	83.75	65.97	83.05	67.69	36.52	67.40	76.59	53.07	76.70	68.12	33.05	63.95	83.09	65.68	82.95	83.03	65.66	84.22
tao	95.69	90.94	95.45	96.21	92.74	96.40	80.89	61.60	80.80	83.92	67.85	83.95	83.61	67.37	83.65	76.25	72.24	86.15	85.00	70.00	85.06
thy	92.60	84.07	92.55	94.33	89.15	95.00	96.93	93.89	97.20	89.30	75.15	89.00	69.77	0.00	57.30	96.28	92.04	96.30	96.56	92.58	95.47
veh	72.28	62.09	71.50	70.21	60.58	70.35	44.68	28.92	42.85	74.08	65.42	72.90	40.91	22.80	32.80	77.75	75.09	81.20	79.46	72.61	83.18
vot	96.57	92.72	96.55	93.08	85.41	93.05	90.02	79.76	90.30	95.77	91.56	96.00	94.69	88.93	94.70	95.20	87.70	94.15	94.41	88.17	93.49
vow	80.20	77.78	79.75	96.99	96.39	96.75	62.90	60.50	64.00	70.61	68.28	71.00	30.98	26.55	32.30	96.11	92.22	92.90	91.52	89.72	90.49
wav	75.25	63.13	75.40	77.67	66.59	77.75	80.01	70.05	78.75	86.48	79.73	86.45	85.38	78.30	85.30	76.42	74.51	83.00	85.44	78.16	85.99
wbcd	95.01	89.73	95.35	96.59	92.90	96.80	96.07	91.41	96.10	96.75	92.89	96.80	96.02	91.24	96.05	95.78	88.92	95.00	96.61	92.55	95.86
wne	92.81	90.59	93.80	96.24	94.06	96.00	97.36	97.44	98.35	98.48	98.72	99.15	41.46	2.49	25.65	96.64	96.59	97.70	97.91	96.81	97.53
zoo	92.57	90.85	92.90	92.57	90.85	92.85	94.95	93.52	94.70	96.04	94.77	95.80	72.38	61.64	64.15	96.95	94.77	95.80	96.56	94.88	95.09
Rank	**3.87**	**3.80**	**3.93**	**4.20**	**4.05**	**3.95**	**4.87**	**4.65**	**4.60**	**3.13**	**3.42**	**3.28**	**5.82**	**5.88**	**5.85**	**3.42**	**3.23**	**3.22**	**2.7**	**2.96**	**3.17**
Pos	*4*	*4*	*4*	*5*	*5*	*5*	*6*	*6*	*6*	*2*	*3*	*3*	*7*	*7*	*7*	*3*	*2*	*2*	*1*	*1*	*1*

Table 4.11: Comparison table of the average test performance of the ten times stratified ten-fold cross-validation obtained with the different data mining algorithms analyzed. Columns describe: the identifier of the data set (*Id.*), the test accuracy (*A*), the Cohen's kappa statistic (*K*) and the F-Measure (*F*), for each algorithm. The last two rows show the Friedman's average ranking and the position for (1) the test accuracy, (2) the Cohen's kappa statistic and (3) the F-Measure.

the exponent. SMO with exponent 1 was the best ranked on these tests, so we picked a SMO with polynomial kernel with exponent 1 and a SMO with radial basis function kernel. The same happened with IBk: we chose the k parameter from $\{1, 3, 5, 7\}$ following the same procedure. IBk with $k = 3$ was the best ranked. We did not introduce the same system with different configurations in the comparison to avoid biasing the statistical analysis of the results.

To compare the different techniques we followed (Dietterich, 1998) and applied ten times stratified ten-fold cross validation to obtain a measure of the accuracy of each method, that is, we averaged over ten runs for each algorithm. By performing this way sampling biases are avoided. Further, we also performed the same using the Cohen's kappa score, precision, recall and the F-Measure. These results were statistically compared to check whether the null hypothesis that the different algorithms perform the same on average.

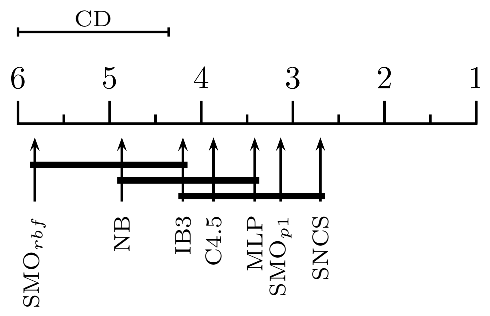


FIGURE 4.24: Nemenyi's test at $\alpha = 0.05$ on the classification problems using test accuracy. Classifiers that are not significantly different are connected.

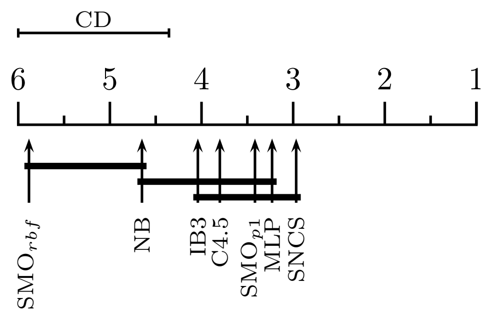


FIGURE 4.25: Nemenyi's test at $\alpha = 0.05$ on the classification problems using Cohen's kappa statistic. Classifiers that are not significantly different are connected.

4.5.2 Results

Table 4.11 shows the results obtained by each algorithm. Note that, due to space limitations, precision and recall are not shown. The last two rows show the Friedman's average ranking and the position for the test accuracy, the Cohen's kappa statistic and the F-Measure,

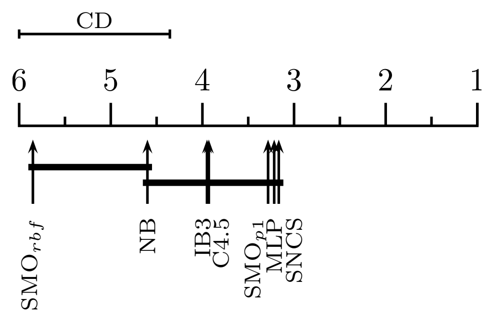


Figure 4.26: Nemenyi's test at $\alpha = 0.05$ on the classification problems using F-Measure. Classifiers that are not significantly different are connected.

respectively, for each method.

The different statistical tests performed are discussed in what follows. These concern (1) test accuracy, (2) Cohen's kappa statistic and (3) F-Measure.

- **Test accuracy**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 6 and 174 degrees of freedom equal to 9.4662 and a computed p-value of $5.3948 \cdot 10^{-9}$, therefore we applied post-hoc Nemenyi's test to check the differences as shown in Figure 4.24. This test resulted in the following conclusions at $\alpha = 0.05$: (1) although SNCS, SMO$_{p1}$, MLP, C4.5 and IB3 were not significantly different, SNCS was the best ranked method, (2) SNCS and SMO$_{p1}$ were significantly better than NB and SMO$_{rbf}$, and (3) MLP, C4.5 and IB3 were significantly better than SMO$_{rbf}$. We performed pairwise comparisons using the Holm's and the Shaffer's procedures and they reflect the same conclusions, visible on Table 4.12. Figure 4.27 illustrates the same results using the Wilcoxon signed-ranks test.

- **Cohen's kappa statistic**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 6 and 174 degrees of freedom equal to 7.9221 and a computed p-value of $1.4679 \cdot 10^{-7}$, therefore we applied post-hoc Nemenyi's test to check the differences as shown in Figure 4.25. Similar conclusions as in the case of the test accuracy can be drawn at $\alpha = 0.05$: (1) SNCS was the best ranked method, but followed closely by MLP, SMO$_{p1}$, C4.5 and IB3, which were not significantly different, (2) SNCS was significantly better than NB and SMO$_{rbf}$, and (3) MLP, SMO$_{p1}$, C4.5 and IB3 were significantly better than SMO$_{rbf}$. We performed pairwise comparisons using the Holm's and the Shaffer's procedures and they reflect the same conclusions, visible on Table 4.13. Figure 4.28 illustrates the same results using the Wilcoxon signed-ranks test.

- **F-Measure**. The Friedman's test rejected the null hypothesis that the algorithms performed the same on average with an F_F statistic distributed according to the F distribution with 6 and 174 degrees of freedom equal to 7.2680 and a computed p-value of $6.1097 \cdot 10^{-7}$, therefore we applied post-hoc Nemenyi's test to check the differences as shown in Figure A.1. Again, similar conclusions can be drawn at $\alpha = 0.05$: (1) SNCS was the best ranked method, but followed very closely by MLP, SMO$_{p1}$, C4.5, IB3, and NB which were not significantly different, and (2) SNCS, MLP, SMO$_{p1}$, C4.5 and IB3 were significantly better than SMO$_{rbf}$. We performed pairwise comparisons using the Holm's and the Shaffer's procedures and they reflect the same conclusions, visible on Table 4.14. Figure 4.29 illustrates the same results using the Wilcoxon signed-ranks test.

i	Algorithms	z	p	p-Holm	p-Shaffer
21	**SMO$_{rbf}$ vs. SNCS**	5.5877	$2.3010 \cdot 10^{-8}$	0.0024	0.0024
20	**SMO$_{p1}$ vs. SMO$_{rbf}$**	4.8108	$1.5033 \cdot 10^{-6}$	0.0025	0.0033
19	**SMO$_{rbf}$ vs. MLP**	4.3028	$1.6864 \cdot 10^{-5}$	0.0026	0.0033
18	**NB vs. SNCS**	3.8845	$1.0254 \cdot 10^{-4}$	0.0028	0.0033
17	**C4.5 vs. SMO$_{rbf}$**	3.4960	$4.7221 \cdot 10^{-4}$	0.0029	0.0033
16	**NB vs. SMO$_{p1}$**	3.1076	0.0019	0.0031	0.0033
15	IB3 vs. SMO$_{rbf}$	2.8984	0.0037	0.0033	0.0033
14	IB3 vs. SNCS	2.6893	0.0072	0.0036	0.0034
13	NB vs. MLP	2.5996	0.0093	0.0038	0.0038
12	C4.5 vs. SNCS	2.0917	0.0365	0.0042	0.0042
11	IB3 vs. SMO$_{p1}$	1.9124	0.0558	0.0045	0.0045
10	C4.5 vs. NB	1.7928	0.0730	0.0050	0.0050
9	NB vs. SMO$_{rbf}$	1.7032	0.0885	0.0056	0.0056
8	IB3 vs. MLP	1.4044	0.1602	0.0062	0.0062
7	C4.5 vs. SMO$_{p1}$	1.3147	0.1886	0.0071	0.0071
6	MLP vs. SNCS	1.2849	0.1988	0.0083	0.0083
5	IB3 vs. NB	1.1952	0.2320	0.0100	0.0100
4	C4.5 vs. MLP	0.8068	0.4198	0.0125	0.0125
3	SMO$_{p1}$ vs. SNCS	0.7769	0.4372	0.0167	0.0167
2	C4.5 vs. IB3	0.5976	0.5501	0.0250	0.0250
1	SMO$_{p1}$ vs. MLP	0.5080	0.6115	0.0500	0.0500

TABLE 4.12: Holm / Shaffer Table for $\alpha = 0.05$ on the classification problems using test accuracy. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

4.6 Discussion

With the results shown on Table 4.11 we can draw two interesting observations. First, since SNCS is based on MLP it is very interesting to compare these two. As expected, both perform the same on average although SNCS being better in the overall ranking (Friedman's test) for the three measures used in this study. This is due to the Michigan-style LCS nature of SNCS and the effectiveness of neural constructivism which allows for evolving the neural structure that best fits the training data. In this regard, the set of pressures detected by Butz (2006) in Michigan-style LCSs can be extrapolated to SNCS (e.g., the mechanics of this LCSs framework tackle the effects of overfitting), as the large experimentation support. Second, the on-line architecture of SNCS performs at the same level as the most significant

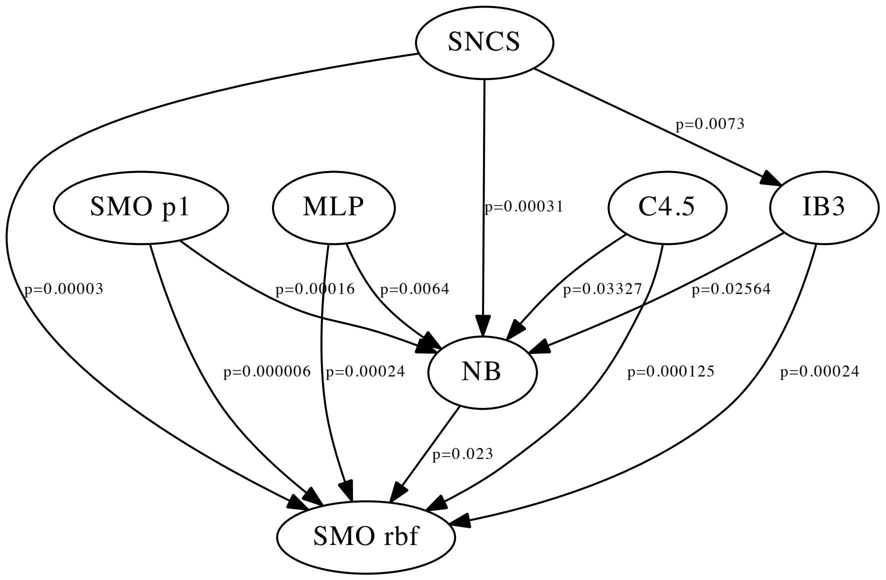


Figure 4.27: Illustration of the significant differences (at $\alpha = 0.05$) among classifiers using accuracy as a test measure. An edge $L_1 \rightarrow L_2$ indicates that the learner L_1 outperforms L_2 with the corresponding p-value.

i	Algorithms	z	p	p-Holm	p-Shaffer
21	**SMO$_{rbf}$ vs. SNCS**	5.2291	$1.7032 \cdot 10^{-7}$	0.0024	0.0024
20	**SMO$_{rbf}$ vs. MLP**	4.7510	$2.0237 \cdot 10^{-6}$	0.0025	0.0033
19	**SMO$_{p1}$ vs. SMO$_{rbf}$**	4.4223	$9.7635 \cdot 10^{-6}$	0.0026	0.0033
18	**C4.5 vs. SMO$_{rbf}$**	3.7351	$1.8765 \cdot 10^{-4}$	0.0028	0.0033
17	**IB3 vs. SMO$_{rbf}$**	3.2869	0.0010	0.0029	0.0033
16	**NB vs. SNCS**	3.0179	0.0025	0.0031	0.0033
15	NB vs. MLP	2.5398	0.0111	0.0033	0.0033
14	NB vs. SMO$_{p1}$	2.2112	0.0270	0.0036	0.0036
13	NB vs. SMO$_{rbf}$	2.2112	0.0270	0.0038	0.0038
12	IB3 vs. SNCS	1.9422	0.0521	0.0042	0.0042
11	C4.5 vs. NB	1.5239	0.1275	0.0045	0.0045
10	C4.5 vs. SNCS	1.4940	0.1352	0.0050	0.0050
9	IB3 vs. MLP	1.4641	0.1431	0.0055	0.0055
8	IB3 vs. SMO$_{p1}$	1.1355	0.2562	0.0062	0.0062
7	IB3 vs. NB	1.0757	0.2821	0.0071	0.0071
6	C4.5 vs. MLP	1.0159	0.3097	0.0083	0.0083
5	SMO$_{p1}$ vs. SNCS	0.8068	0.4198	0.0100	0.0100
4	C4.5 vs. SMO$_{p1}$	0.6872	0.4919	0.0125	0.0125
3	MLP vs. SNCS	0.4781	0.6326	0.0167	0.0167
2	C4.5 vs. IB3	0.4482	0.6540	0.0250	0.0250
1	SMO$_{p1}$ vs. MLP	0.3287	0.7424	0.0500	0.0500

Table 4.13: Holm / Shaffer Table for $\alpha = 0.05$ on the classification problems using Cohen's kappa statistic. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

machine learning techniques, emphasising the flexibility of SNCS as a competent classifier for static data sets. It is important to highlight that the system excelled in data stream problems compared with methods specially designed to deal with data streams. These results support the robustness of SNCS, which appears as a competent learner for data classification,

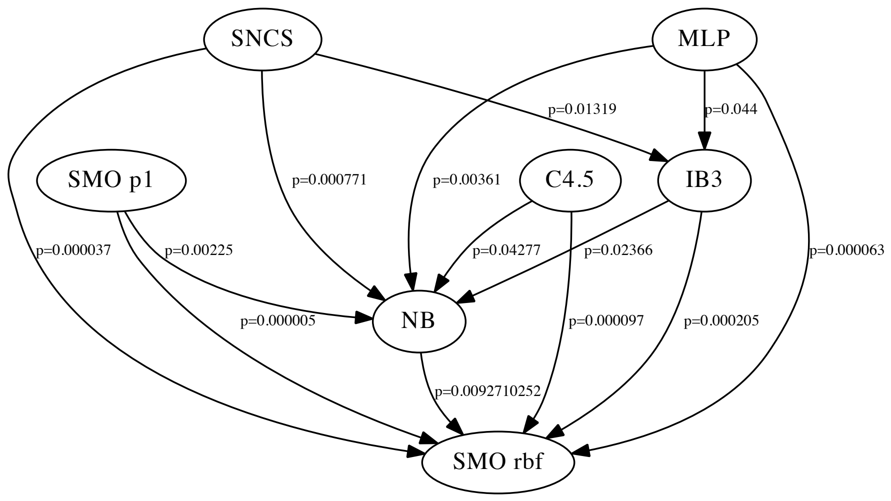


FIGURE 4.28: Illustration of the significant differences (at $\alpha = 0.05$) among classifiers using Cohen's kappa statistic. An edge $L_1 \rightarrow L_2$ indicates that the learner L_1 outperforms L_2 with the corresponding p-value.

i	Algorithms	z	p	p-Holm	p-Shaffer
21	**SMO$_{rbf}$ vs. SNCS**	4.8107	$1.5033 \cdot 10^{-6}$	0.0024	0.0024
20	**SMO$_{rbf}$ vs. MLP**	4.7211	$2.3451 \cdot 10^{-6}$	0.0025	0.0033
19	**SMO$_{p1}$ vs. SMO$_{rbf}$**	4.6016	$4.1920 \cdot 10^{-6}$	0.0026	0.0033
18	**C4.5 vs. SMO$_{rbf}$**	3.4363	$5.8976 \cdot 10^{-4}$	0.0028	0.0033
17	**IB3 vs. SMO$_{rbf}$**	3.4060	$6.5825 \cdot 10^{-4}$	0.0029	0.0033
16	NB vs. SNCS	2.5697	0.0102	0.0031	0.0033
15	NB vs. MLP	2.4801	0.0131	0.0033	0.0033
14	NB vs. SMO$_{p1}$	2.3606	0.0182	0.0036	0.0036
13	NB vs. SMO$_{rbf}$	2.2410	0.0250	0.0038	0.0038
12	IB3 vs. SNCS	1.4044	0.1602	0.0042	0.0042
11	C4.5 vs. SNCS	1.3745	0.1693	0.0045	0.0045
10	IB3 vs. MLP	1.3147	0.1886	0.0050	0.0050
9	C4.5 vs. MLP	1.2849	0.1988	0.0055	0.0055
8	C4.5 vs. NB	1.1952	0.2320	0.0062	0.0062
7	IB3 vs. SMO$_{p1}$	1.1953	0.2320	0.0071	0.0071
6	IB3 vs. NB	1.1653	0.2439	0.0083	0.0083
5	C4.5 vs. SMO$_{p1}$	1.1653	0.2439	0.0100	0.0100
4	SMO$_{p1}$ vs. SNCS	0.2092	0.8343	0.0125	0.0125
3	SMO$_{p1}$ vs. MLP	0.1195	0.9049	0.0167	0.0167
2	MLP vs. SNCS	0.0896	0.9286	0.0250	0.0250
1	C4.5 vs. IB3	0.0299	0.9762	0.0500	0.0500

TABLE 4.14: Holm / Shaffer Table for $\alpha = 0.05$ on the classification problems using the F-Measure. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

regardless of wether the data is static or in the form of a data stream.

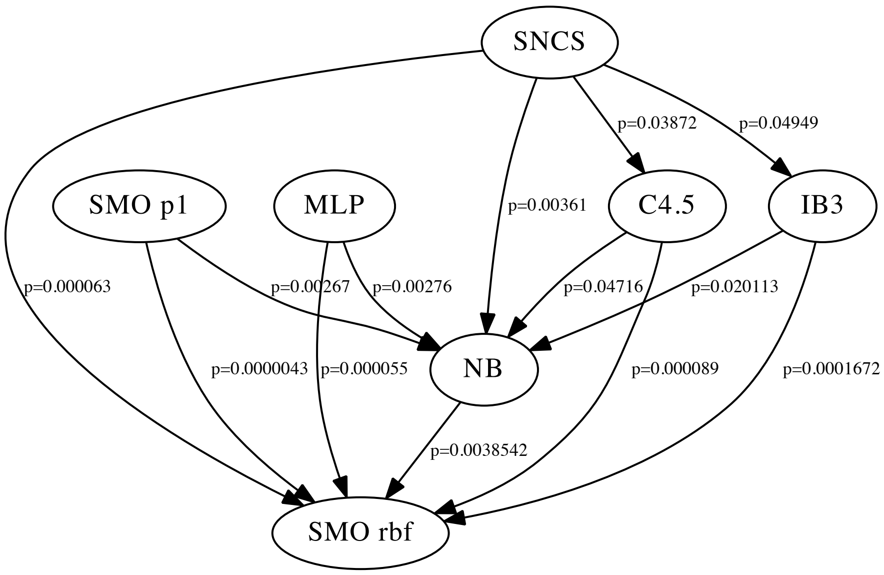


Figure 4.29: Illustration of the significant differences (at $\alpha = 0.05$) among classifiers using the F-Measure. An edge $L_1 \rightarrow L_2$ indicates that the learner L_1 outperforms L_2 with the corresponding p-value.

4.7 Summary, Conclusions and Critical Analysis

4.7.1 Summary and Conclusions

A brand new Michigan-style neural-learning classifier system, SNCS, which is designed to deal with data streams has been presented. We identified the common difficulties of learning form data streams as abrupt concept changes, varying concept drift speeds, varying noise levels, virtual drifts, spurious relationships among padding variables under high dimensional spaces and non-linear boundaries. We selected a set of widely-used benchmark problems that allowed us to test those complexities and we analysed in detail the results obtained by SNCS. We also included three other well-known data stream mining algorithms for comparisons. Furthermore, we have extended the analysis of the competitiveness of SNCS by experimenting with a set of real-world classification problems, all containing stationary concepts, by comparing the accuracy results of SNCS with the ones of the most significant machine learning methods.

From the first series of experiments performed in this work, SNCS has empirically shown a high degree of robustness against the data stream mining complexities. That robustness is specially noticeable on the SEA problem and variants, where SNCS showed a remarkable reaction capacity to concept changes and noisy inputs, even with padding variables in high dimensional spaces, obtaining better overall results than CVFDT, a well known competitive data stream miner.

From the second series of experiments, SNCS has empirically shown the competitiveness with stationary classification tasks, resulting in models that were not significantly different from that of the ones created by some of the most successful machine learning techniques.

4.7.2 Critical Analysis of SNCS

To finish the study of SNCS a final analysis is performed in the following. Table 4.15 summarises the analysis, where *strengths* represent the main advantages of SNCS, *weaknesses* show the drawbacks of this system, *opportunities* suggest the further lines of research to enhance the system performance, and *threads* indicate issues that other machine learning approaches may take advantage of.

Strengths	**Weaknesses**
- It gives highly accurate solutions, excelling in data streams. - It deals with large data sets. - It demonstratea a fast reaction capacity. - It requires a small number of classifiers. - It handles all types of *functions*.	- It outputs solutions that are not human friendly. - Categorical features have to be preprocessed before using them. - It is a *pure* supervised algorithm; it may require an unsupervised technique when tackling real-world problems with concept drift.
Opportunities	**Threats**
- The good results obtained in classification tasks suggest that it can be exported to other kinds of learning, like online regression. - The evolute component can be further investigated to obtain a better response of the overall system. - Other representations can be added to the system.	- It is a young system with relatively little experience. - Other learning approaches can deal with data streams in a similar way.

TABLE 4.15: Critical analysis of SNCS.

SNCS has five main strengths: first, it has demonstrated a remarkable behaviour in both kinds of problems, those with dynamic concepts and the traditional problems with static concepts. Second, SNCS has shown that it can deal with problems that have a huge number of instances. Third, SNCS has a fast reaction capacity against drifting concepts—one of our main design goals with the algorithm. Fourth, in comparison with other Michigan-style LCSs, it requires a reduced number of classifiers to efficiently solve the problem at hand. And fifth, as SNCS uses a flexible knowledge representation, it can handle many types of problems without any loss of generality and accuracy.

However, SNCS has three main weaknesses: the main one is that the solution given by SNCS, although accurate, it is not human readable. This issue may hinder the application of SNCS in some areas where the readability is of the utmost importance. Another issue lies in the fact that categorical features have to be preprocessed before using these: this is due to the representation used which only accepts numeric data naturally. It is worth mentioning that the preprocess is straightforward, and consists in mapping binary features to the set of categorical attributes. Also, as SNCS is a *pure* supervised learning algorithm it may require an unsupervised method for labelling when tackling real-world data streams with unknown dynamics and when concept drift happens.

The possible threats to SNCS are twofold: in the one hand it is a young systems with little experience and it lacks a in-depth analysis for a better understanding of the generalisation and learning of the algorithm. In the other hand, we honestly mention that other learning approaches such as Fuzzy-UCS (Orriols-Puig et al., 2009b) can obtain results which may rival the ones obtained by SNCS (Orriols-Puig and Casillas, 2010b)—it is important to highlight that in this particular case Fuzzy-UCS shares the Michigan-style LCS architecture.

Finally, SNCS shows some interesting opportunities to be developed in further work: we are interested in the study of different evolutive mechanisms to obtain optimal topologies in the population structure and check how these affect the behaviour of the system, as well as testing the algorithm in new challenging environments, like in extremely imbalanced domains. We are also interested in extending SNCS with different types of knowledge representation and check the tradeoff between flexibility and interpretability: on the one hand, a radial basis function (RBF), a potentially very flexible representation, can be explored with minimal changes to the architecture. On the other hand, a neuro-fuzzy representation can be explored as well for improving the readability of the solutions obtained. Also, the competent results obtained by SNCS suggest that we can export it to other kinds of problems, like online regression, without modifying the general architecture.

Despite the good results obtained in the online supervised field, these techniques assume an *a priori* underlying structure for the set of features of the problem. That is, supervised techniques require the *real* output—the class of each training example—in order to build a reliable model from data. This issue is often far reaching, specially in real-world problems, making the direct application of *pure* supervised learners barely practical in most industrial and scientific applications and requiring the use of hybrid methods which combine supervised with unsupervised techniques in order to deal with the problem (Shi et al., 2010).

In the upcoming chapters we apply the lessons learned and introduce two unsupervised algorithms that follow the Michigan-style LCS paradigm.

5 Clustering through Michigan-Style LCS

Data mining techniques are traditionally divided into two distinct disciplines depending on the task to be performed by the algorithm; these two disciplines are named supervised learning and unsupervised learning. While the former aims at making accurate predictions after deeming an underlying structure in data which requires the presence of a *teacher* during the learning phase, the latter aims at discovering regular-occurring patterns beneath the data without making any *a priori* assumptions concerning their underlying structure. The *pure* supervised model can construct a very accurate predictive model from data streams. However, in many real-world problems this paradigm may be ill-suited due to (1) the dearth of training examples and (2) the costs of labelling the required information to train the system. A sound use case of this concern is found when defining data replication and partitioning policies to store data emerged in the Smart Grids domain in order to adapt electric networks to current application demands (e.g., real time consumption, network self adapting). As opposed to classic electrical architectures, Smart Grids encompass a fully distributed scheme with several diverse data generation sources. Current data storage and replication systems fail at both coping with such overwhelming amount of heterogeneous data and at satisfying the stringent requirements posed by this technology: dynamic nature of the physical resources, continuous flow of information and autonomous behaviour demands.

The purpose of this chapter is to improve *eXtended Classifier System for Clustering* (XCSc) (Shi et al., 2011, Tamee et al., 2007a) to face the aforementioned challenges and, thus, present a hybrid system that mixes data replication and partitioning policies in an unsupervised learning by means of an online clustering approach. This enhanced version of the algorithm is coined to as *eXtended Classifier System for Clustering Data Streams* (XCScds) to be distinguished from the original XCSc. Conducted experiments show that the proposed system outperforms previous proposals and truly fits with the Smart Grid premises.

Te remainder of this chapter is organised as follows. Section 5.1 points out the characteristics of Smart Grids and the necessity of obtaining patterns to optimise the processes in which this new form of energy delivery system is involved. Section 5.2 describes the related work in Smart Grids and in clustering data streams, stressing the critical concerns of data storage, replication, and partitioning in the specific context of Smart Grids Section 5.3 presents the deployed system architecture to handle the aforementioned requirements. Section 5.4 depicts the obtained results with our approach and compares them with previous work. Finally, Section 5.5 concludes the chapter.

5.1 Introduction

Power electric distribution and transport networks belong to a deeply established but poorly evolved market (Gungor et al., 2011) that fails to provide advanced functionalities (Gungor et al., 2013, Rusitschka et al., 2010) to both consumers and producers (also referred to as *prosumers*). According to the latest European directives, this situation must change radically in order to meet new standards concerning energy efficiency and sustainability (i.e., reducing greenhouse gas emissions, promoting energy security, fostering technological development and innovation, and limiting the amount of imported energy). In this regard, a new form of energy delivery system, coined as Smart Grid, has emerged as an alternative to handling new-generation power grid functionalities, which include real-time consumption monitoring, network self-healing, advanced metering infrastructure, or overload detection (Gungor et al., 2013).

To successfully migrate from the traditional centralised power delivery infrastructures to the distributed nature required by Smart Grids, several disciplines must be integrated (Monti and Ponci, 2010, Yan et al., 2013): communication networks—to enable interactions between every device of the grid—, cyber security—to ensure that the system is safely operated—, and distributed data storage—to effectively deal with the data generated by this novel scheme (Navarro et al., 2013b). While the interaction of these disciplines is being explored under the context of some European-funded research projects (Gungor et al., 2013, Navarro et al., 2013b), there is still not a mature framework to manage the vast amount of heterogeneous data generated by the ever-growing number of devices that populate the Smart Grid (Navarro et al., 2011, Rusitschka et al., 2010). In fact, there are several factors that prevent—recently proposed in the literature—highly scalable (cloud) data repositories from meeting the stringent requirements of Smart Grids (Yan et al., 2013): (1) limited communication facilities in terms of delay (Selga et al., 2013), (2) massive amount of data streams (22 GB per day) that need to be effectively processed (Gungor et al., 2013, Rusitschka et al., 2010), (3) limited computing resources at smart devices (Navarro et al., 2013b), and (4) incompetence to store big sets of structured data (Stonebraker and Hong, 2012).

When addressing data storage and replication in the context of Smart Grids, practitioners are forced to select a convenient tradeoff between data consistency, data availability, and network partitioning as stated by the CAP theorem (Brewer, 2012). An effective way to deal with the CAP theorem in the specific context of Smart Grids consists of establishing a proper data partitioning layout (i.e., classifying and confining data in “logical islands” in order to limit the scope of replication and user requests) and, thus, keep data availability and consistency while leveraging system scalability. Hence, every data island (i.e., data partition) is able to scale up independently according and meet the system requirements at every moment with little penalty on other partitions. Therefore, initial approaches to address these concerns in distributed storage aim at replicating data in a partitioned scheme (Curino et al., 2010, Navarro et al., 2011). In fact, designing the optimal data partition configuration according to the demands posed by smart applications and the nature of these data has emerged as a hot research topic.

In this context, real-world industrial applications generate large amounts of data that are complex, ill-defined, directly unstructured and which contain hidden information beneath that is potentially useful and exploitable for strategic business decision making (Orriols-Puig et al., 2012). Building useful model representations from these data is a task that requires the use of unsupervised learning because this paradigm does not assume any *a priori* structure in the data. This approach is based on algorithms that automatically explore the data in order to uncover subjacent patterns. A cornerstone of unsupervised learning is found in data clustering, which consists in grouping examples into a set of groups—the clusters—according to some proximity criterion with the aim of searching for hidden patterns (García-Piquer, 2012). The concept of making clusters out of input data is extrapolated to data streams in the field of *clustering data streams*.

The purpose of this chapter is to introduce an online unsupervised Michigan-style LCS algorithm in order to be able to build a data partitioning scheme that adapts itself to the specific needs of the Smart Grid. More specifically, this system is targeted at analysing the data streams generated by smart devices in order to build a set of clusters, each containing the most frequently retrieved datum patterns, and to minimise the amount of accesses to multiple sites. This valuable information is used by the data replication protocol of the Smart Grid when selecting the proper device to place every datum. With this novel approach, the data management system of the Smart Grid is able to build a dynamic set of partitions and, thus, minimise the overhead associated to data movement over a distributed system (Brewer, 2012).

The contributions of this chapter are listed the following:

- It introduces the problem of data replication and partitioning policies under the Smart Grids.
- It applies XCScds, an online unsupervised Michigan-style LCS algorithm, to implement the partitioning policies of a Smart Grid data storage system.
- It goes beyond the state-of-the-art definition of XCSc to foster the usage of this algorithm in online domains. Specifically, it details the internal behaviour of XCScds for implementation and replication purposes.
- It details the online and evolving behaviour capacities of the proposed XCScds version.
- It demonstrated the competitive behaviour of the presented approach by conducting a series of experiments on (1) a classic data stream synthetic environment, (2) an extended data stream synthetic environment with evolving components, and (3) a realistic scenario using the standard benchmarks proposed by the *Yahoo!* Cloud Serving Benchmark (YCSB).

5.2 Data Concerns in Smart Grids and Framework

Data play a key role in the Information and Communication Technology infrastructure that supports the Smart Grid (Navarro et al., 2013b). In fact, smart functions that provide advanced functionalities on the electric domain rely on the quality and richness of the gathered information. In addition, these collected data are aggregated and computed at very geographically distinct points with different requirements—according to every smart function constraint (Gungor et al., 2013)—in terms of consistency, availability and network partitioning, which drives distributed systems practitioners into a critical compromise referred to as CAP theorem (Brewer, 2012).

Classic relational databases fail at finding an optimal trade-off between these features while providing scalable solutions (Brewer, 2012). Therefore, the latest cloud-based repositories, devoted to addressing the storage challenges of what has been recently coined as Big Data (Stonebraker and Hong, 2012), have relaxed the relational properties (commonly by relying on key-value stores) and consistency constraints (using weak consistency models such as eventual consistency) of data in order to build highly scalable storage systems (Gulisano et al., 2012). Although this strategy has been successfully used to cope with the ever-growing amounts of data generated by several real-world applications (e.g., Twitter, Facebook, Google), preliminary results obtained in the Smart Grid domain Rusitschka et al. (2010) are far from acceptable (Yan et al., 2013). Indeed, it has been shown that these general purpose storage repositories are unable to handle the specificities of Smart Grids in terms of variable consistency (Gungor et al., 2013), response time (Selga et al., 2013), and resource adaptability (Navarro et al., 2011).

5.2.1 Data Partitioning

In order to (1) meet the aforesaid storage requirements, (2) overcome scalability limitations of fully replicated solutions, and (3) exploit data locality, the construction of a proper data partitioning scheme has become necessary (Curino et al., 2010), especially on those situations where availability and consistency cannot be relaxed (Gungor et al., 2013). Actually, when data are partitioned (i.e., data from a set of nodes are rarely used at any other set), it is possible to keep an arbitrary set of—reasonably small—groups of federated servers storing every data partition. In this way, each partition owns all data that is going to be used together and, thus, interdependencies are minimal, which allows every partition to independently scale up and feature a particular set of facilities (Navarro et al., 2011) although, there is no notion of global consistency across partitions.

Nevertheless, partitioning requires a deep analysis and knowledge of data and their access patterns, which is not feasible in the Smart Grid because (1) they are constantly changing due to the dynamic conditions of this domain and, (2) smart functions that access data are in the early stages of standardisation. Thus, existing solutions in the literature that use offline approaches (Curino et al., 2010) to partition data are unable to meet the Smart Grid demands. Therefore, we propose the use of an online clustering system to dynamically infer access patterns and discover data dependencies without any previous knowledge about the Smart Grid. The information extracted from this system is used by the data storage infrastructure to properly place every datum and build the best partitioning scheme (i.e., relation between a data set and its location). Note that decisions concerning when to perform partition adjustments (i.e., joining or splitting partitions), the cost analysis of performing a partition reconfiguration, and techniques to keep data consistency while partitions are being rearranged (also referred to as live migration (Das et al., 2011, Kikuchi and Matsumoto, 2012)), are out of the scope of this work. Hence, this chapter (1) aims to provide an online partitioning layout built upon a continuous analysis of the data access patterns and (2) claims that it is feasible to adapt the data partitioning layout to incoming workloads.

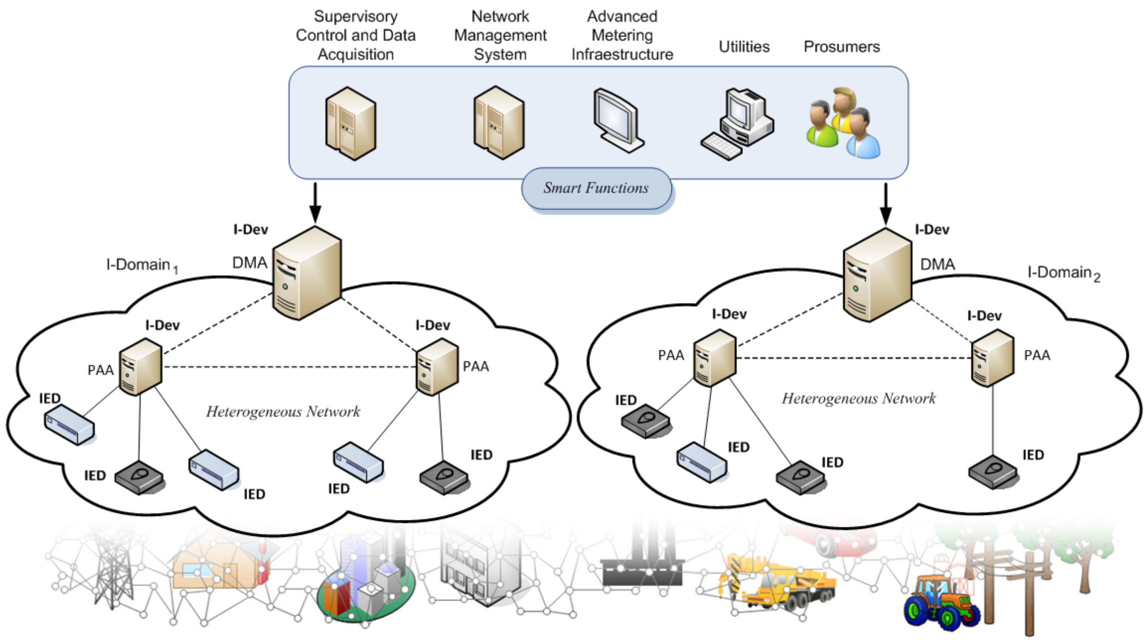


Figure 5.1: Architecture of the deployed XCScds system over the Smart Grid holding two data partitions.

To achieve such commitment, we have deployed on top of the multi-agent system depicted in Figure 5.1 an XCScds for clustering every datum (i.e., building the aforementioned partitions). As shown in Figure 5.1, the Smart Grid is built by means of Intelligent Electronic Devices (IEDs) that are continuously collecting information from the physical domain. Then, this information is delivered to what has been coined as I-Devs, which are highly-reliable devices with general-purpose reduced storage and computing capabilities (Navarro et al., 2013b). In this way, I-Devs communicate among each other through a heterogeneous network (Gungor et al., 2013, Selga et al., 2013) to share the information obtained at every corner of the grid. Likewise, at the very top of this process, smart functions access I-Devs to obtain the requested data.

5.2.2 Clustering Data Streams

Clustering data streams is an appealing paradigm that complements its supervised learning counterpart. Despite the fact that both fields can be merged to exploit the capabilities of the two facets at the same time (i.e., a first unsupervised grouping process and later a supervised prediction), this section focuses on the unsupervised learning state-of-the-art. The reader is referred to (Navarro et al., 2013b) for a practical example on mixing both strategies. In what follows we discuss the current trends on clustering data streams and also the contributions of unsupervised Michigan-style LCSs to offline clustering that served as basis for our purpose of designing an online approach. The most representative technique in the field of clustering data streams is online K-Means (Lu et al., 2008, Zhong, 2005), heavily based on the well-known K-Means clustering algorithm (MacQueen, 1967), and also its recent extension Streaming K-Means++ (Ackermann et al., 2012). These algorithms consist in grouping the incoming instances into k partitions or clusters (k is a user-defined parameter) according to some criteria, typically the Euclidean distance. The major drawback of this family of algorithms is that they require the data analyst to properly set the number of partitions in advance. Another technique that is also popular in clustering data streams (Bifet et al., 2010, Gama, 2012) is CoWeb (Fisher, 1987), which uses the concept of a classification tree where each node represents a cluster. Each node is labeled by a probabilistic method that summarises the distribution of attribute-value pairs of the objects in it. CluStream (Aggarwal et al., 2003), an extension of the BIRCH system (Zhang et al., 1996) for handling data streams, makes use of *micro-clusters*, that is, a minimal representation of a set of points with similarities. In this regard, each micro-cluster is used to store the sufficient statistics for computing the desired distance metric. DenStream (Cao et al., 2006), similarly to CluStream, uses the concept of dense micro-cluster to summarize clusters. ClusTree (Kranen et al., 2011) uses a self-adaptive index structure for maintaining stream summaries. D-Stream (Zaharia et al., 2012) maps each input data into a grid and it computes the grid density. Then, the grids are clustered based on the density.

An interesting and recent extension to clustering data streams are the so called *evolving* clustering systems (Angelov and Xiaowei, 2008) (not to be confuse with evolutive algorithms): these are online systems that adapt themselves to new patterns that appear suddenly in the middle of a run and that were previously unknown by the algorithm. Mean-shift (Baruah and Angelov, 2012b) is such an evolving clustering system.

The Michigan-style LCS framework has been widely used for unsupervised learning purposes as well (Shi et al., 2011, Tamee et al., 2006, 2007a,b). In spite of the success of the framework at clustering tasks, these focused exclusively on *purely* offline approaches. More precisely, the authors modified the original online framework for obtaining very accurate results by introducing *pure* offline rule compaction mechanisms.

The following section details how the proposed system fits in this architecture.

5.3 An Effective Online Clustering for Smart Grids

Considering the ever-growing nature of the Smart Grid, it is essential to build a scalable approach (Monti and Ponci, 2010). To this concern, we have split the XCScds intelligent system into two layers (see Figure 5.1): a low layer composed by a set of Perception Action Agents (PAAs) running at every I-Dev that perceives information from the system, and an upper layer composed by a set of Domain Management Agents (DMAs) that aggregate the information from PAAs and build the knowledge model. Hence, every DMA is in charge of a single partition and manages its own PAAs. Likewise, DMAs report the information to the storage layer which decides the roles of every I-Dev in order to apply the proposed partitioning scheme.

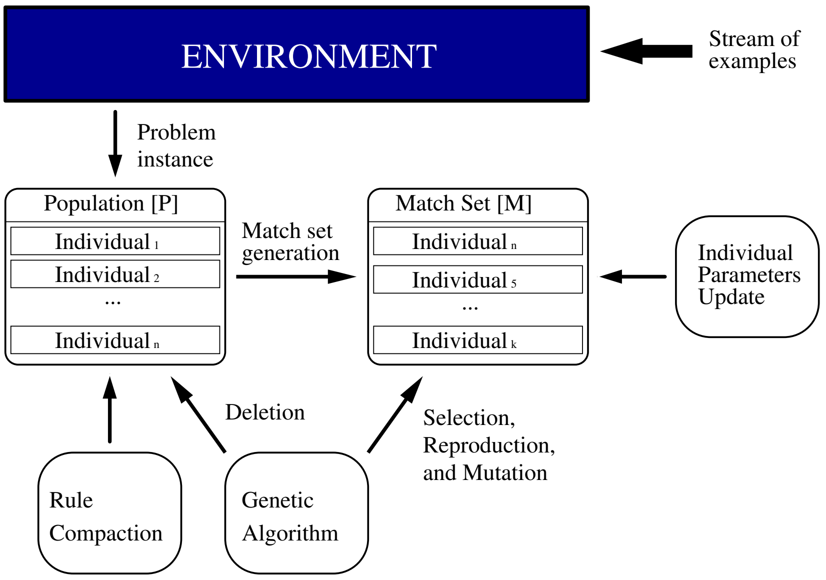


Figure 5.2: Detailed schema of the XCScds system.

XCSc (Shi et al., 2011, Tamee et al., 2007a) is a Michigan-style LCS specifically designed for extracting clusters without any *a priori* underlying structure in these data. XCSc inherits the main traits of XCS, the most well-known Michigan-style LCS, and hence the evolutionary pressures that assure that XCS works are present. However, due to how the original XCSc implements the rule reduction mechanism, it only works under offline problems. Thus, XCSc has never been tested in online environments. We upgraded this learning architecture to allow XCSc to mine online streams of information by (1) modifying the rule reduction mechanism and (2) by integrating an *evoling* component. Also, we improved certain internal details of XCSc in the representation and in the distance metric used by the algorithm. We coined to as XCScds to this new upgrated version. XCScds combines GAs with apportionment of credit mechanisms, hence evolving a population of clusters in an incremental way, as shown in Figure 5.2.

Recently, Baruah and Angelov (2012b) identified three essential requirements for an *evolving* online clustering algorithm, which are the following: (1) capacity of handling outliers,

(2) capacity of creating new clusters if needed and merging existing ones, and (3) the system does not assume a predefined number of clusters. Our version of XCScds possesses these requirements and, hence, its online nature enables the system to adapt quickly to changes in concept and it is also robust to noise.

In what follows, the knowledge representation and the online learning organisation of XCScds, jointly with the modifications carried out in the learning architecture, are detailed.

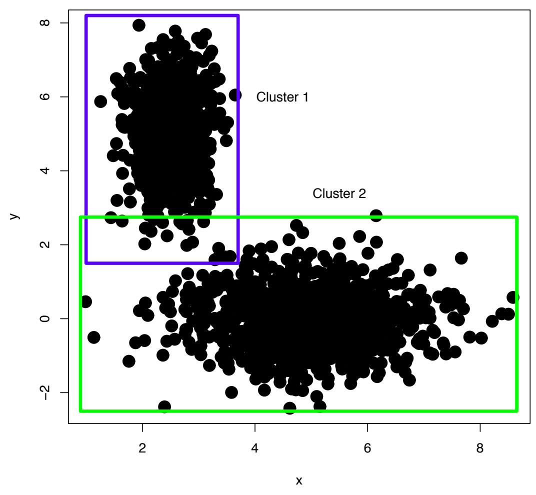
5.3.1 Knowledge Representation

Typically, traditional Michigan-style LCSs evolve a set of highly fit individuals to solve a particular problem. The core of each individual consists of a production rule, which identifies the set of values that define the domain of action of the individual, and a set of parameters that estimate the quality of the cluster.

Our version of XCScds uses the unordered-bound hyper rectangle representation instead of the original centre-spread one (Stone and Bull, 2003, Tamee et al., 2007a), which has been proven more accurate (Orriols-Puig, 2008). In that sense, a rule is represented as

$$\textbf{if}\, x_1 \in [\ell_1,\, u_1]\, \textbf{and}\, \ldots\, \textbf{and}\, x_n \in [\ell_n,\, u_n]\, \textbf{then}\, C_k,$$

where x_i, $\forall i \in \{1, n\}$, are the set of input features, ℓ_j and u_j are the respective upper and lower limits of each interval ($\forall j\, \ell_j \leq u_j$), and C_k is the cluster identified by the rule. The number of features per rule is fixed, that is, all the n input variables are used to identify the different sets of clusters. Figure 5.3 illustrates this in a two-dimensional problem. It is worth mentioning that using this representation different rules may identify the same cluster—i.e., overlapping rules. This issue is later solved by a compaction routine that fuses similar individuals.



if $x \in [1.04, 3.72]$ **and** $y \in [1.50, 8.21]$ **then** C_1
if $x \in [0.89, 8.65]$ **and** $y \in [-2.49, 2.75]$ **then** C_2

Figure 5.3: Knowledge representation used by XCScds in a two-dimensional problem.

Each individual has a set of parameters that evaluate the quality of the rule. These parameters are (1) the average error ϵ in the rule's matching process, (2) the fitness estimate F which is computed as a power function of the error in order to reflect the individual accuracy, (3) the experience exp which counts the number of times that the antecedent part of the rule has matched an input example, (4) the numerosity num, which reckons the number of copies of the individual in the population, (5) the average size of the niches (referred to as match sets) σ in which the individual has participated, and (6) the time stamp of the individual $timeOfCl$.

5.3.2 Learning Organisation

In this section, the learning scheme of our version of XCScds is detailed. It is worth noting that although it shares many aspects with the original XCSc we have performed several improvements in order to allow the algorithm to handle data streams. XCScds learns incrementally from a stream of examples that are provided by the environment. That is, at each learning iteration XCScds receives an input example e from the environment and takes the following steps to incrementally update the individual's parameters and also to discover new promising rules. First, the system creates the match set $[M]$ with all the individuals in the population that match the input example. If $[M]$ is empty—i.e., there are no matching individuals—, the covering operator is triggered to create a new fully-matching individual cl. Following that, XCScds updates individuals present in $[M]$: experience, error, fitness, and the average size of the niches in which every cl has participated are updated. Next, if the average time since the last application of the GA of individuals in $[M]$ is greater than the θ_{GA} threshold (a user-defined parameter), the genetic rule discovery is triggered. Finally, the rule compaction is executed.

Four elements are needed to be further elaborated in order to understand how XCScds works: (1) the covering operator, (2) the parameter update procedure, (3) the rule discovery mechanism, by means of a GA, and (4) the rule compaction mechanism. In the following, each of these elements is described in more detail.

Covering Operator

Given the input example $e = (e_1, e_2, \ldots, e_n)$, the covering operator generates a new individual cl that fully matches e. For this purpose, the interval of each variable i of the new individual is initialized in the usual way:

$$p_i \leftarrow e_i - \mathrm{rand}(0, r_0) \quad \text{and} \quad q_i \leftarrow e_i + \mathrm{rand}(0, r_0), \tag{5.1}$$

where r_0 is a configuration parameter, and $\mathrm{rand}(0, r_0)$ returns a random number between 0 and r_0. Therefore, this operator creates an interval that includes the value of the corresponding attribute, and r_0 controls the generalisation in the initial population. In this regard, ℓ_i takes the lower value between p_i and q_i, and u_i the higher value. If e_i is unknown or missing, the covering operator replaces it with 0.5 and proceeds using Eq. 5.1 as usual. Individual's parameters are set to initial values; that is, $\epsilon = 0, F = 0.01, num = 1, exp = 0, \sigma$ is set to the size of the match set where the covering has been applied, and $timeOfCl$ to the actual learning time stamp.

Parameter Update Procedure

After the creation of $[M]$, the parameters of all the individuals that belong to such set are updated. First, the experience of each individual is incremented. Second, the error ϵ is updated following the Widrow-Hoff delta rule (Widrow and Lehr, 1990). In the particular case of XCScds, ϵ is derived from a distance measure—the Clark distance in our case[1]—with respect to the given example e and the centroid of each individual cl in $[M]$ (Tamee et al., 2007a). Recall that other distances are possible, but we investigated with the Clark distance due to the fact that (1) it is competitive, giving, on average, better results than the Euclidean one, and (2) it is straightforward to compute (Fornells, 2006). Other distances will we explored in a further work. The centroid $cl.c_i$ is simply computed as the centre of the hyper rectangle represented by the rule. Eq. 5.2 shows this update.

$$cl.\epsilon \leftarrow cl.\epsilon + \beta \left(\sqrt{\sum_{i=1}^{n} \frac{(e_i - cl.c_i)^2}{(e_i + cl.c_i)^2}} - cl.\epsilon \right), \tag{5.2}$$

where β is the learning rate, a user-defined parameter. Third, the niche size estimate is updated following a similar procedure. That is

$$cl.\sigma \leftarrow cl.\sigma + \beta \left(|[M]| - cl.\sigma \right), \tag{5.3}$$

where $|[M]|$ is the size of the current match set. Following that, the fitness is updated as in standard XCS: first, the accuracy $cl.k$ of each individual cl in $[M]$ is calculated as:

$$cl.k \leftarrow \begin{cases} \alpha \left(\frac{cl.\epsilon}{\epsilon_0} \right)^{-\nu} & \text{if } cl.\epsilon \geq \epsilon_0; \\ 1 & \text{otherwise,} \end{cases} \tag{5.4}$$

where α is a user-defined scaling factor, ϵ_0 is the error threshold defined by the user, and ν is the exponent of the power function used to tune the pressure towards highly fit individuals. Next, the individual fitness is updated following Eq. 5.5:

$$cl.F \leftarrow cl.F + \beta \left(\frac{cl.k \cdot cl.num}{\sum_{cl_i \in [M]} cl_i.k \cdot cl_i.num} - cl.F \right). \tag{5.5}$$

Finally, XCScds adds a local search scheme to help the guidance towards highly fit clustering solutions. This is done by using the Widrow-Hoff delta rule with the cluster centers c_{ij} found in $[M]$:

$$c_{ij} \leftarrow c_{ij} + \beta \left(|[M]| - c_{ij} \right). \tag{5.6}$$

Rule Discovery Mechanisms

Two different mechanisms are used by XCScds to discover new knowledge: the aforementioned covering operator and a niche-based GA. XCScds, as most Michigan-style LCSs, uses a steady-state niche-based GA (Wilson, 1995) to discover new promising rules. The GA is triggered in the current match set if the average time since its last application to the individuals in $[M]$ is greater than the user-defined threshold θ_{GA}.

[1]Note that the original XCSc uses the Euclidean distance metric (Tamee et al., 2007a).

First, the time stamp of each individual in $[M]$ is updated. Next, the GA selects two parents from the current $[M]$ following a tournament selection scheme (Butz et al., 2004). Eq. 5.7 indicates the probability of selecting the individual *cl* out of [M], which is proportional to its fitness, that is:

$$P_{del}(cl) \leftarrow \frac{cl.F}{\sum_{cl_i \in [M]} cl_i.F}. \tag{5.7}$$

Then, two copies of these parent individuals are made. These undergo crossover with probability P_χ and mutation with probability P_μ per allele. XCScds uses uniform crossover Goldberg (2002): this operator decides, for each input variable, from which parent the information is copied. If crossover is not applied, the offspring remain as exact copies of the parents. After this, mutation is applied: for each input variable, the mutation operator randomly decides whether the variable needs to be changed by adding a random value of the interval $[-m_0, m_0]$, where m_0 is a configuration parameter. The identifier of the cluster also undergoes the mutation process.

The offspring parameters are initialised as follows: if no crossover is applied, the error and the fitness are copied directly from the selected parent. Otherwise, these parameters are set to the average value between the corresponding parameters in the parents. In all cases, the fitness is decreased by a 10% of the parental fitness. Experience and numerosity are initialised to 1, and the average size of the niches in which the individual has participated is set to the value of the selected parent.

The resulting offspring are introduced into $[P]$ via a subsumption mechanism: if there exists a sufficiently experienced and accurate individual *cl* in $[P]$; that is, if $cl.exp > \theta_{sub}$, where θ_{sub} is a user-defined threshold, and $cl.\epsilon < \epsilon_0$, whose condition is more general than the new offspring, the numerosity of this individual is increased (and, consequently, the offspring discarded). Otherwise, the new offspring is introduced into $[P]$. At this step, and as in the case of the original XCS, until the population is full, individuals in $[P]$ are deleted proportionally accordingly to (1) their fitness and (2) their numerosity, as described by Eq. 5.8:

$$cl.P_{del} \leftarrow \frac{cl.d}{\sum_{\forall cl_i \in [P]} cl_i.d}, \tag{5.8}$$

where

$$cl.d \leftarrow \begin{cases} cl.num \cdot cl.\sigma \cdot F_{[P]} & \text{if } cl.exp > \theta_{del} \text{ and } cl.F < \delta F_{[P]}; \\ cl.\sigma \cdot cl.num & \text{otherwise,} \end{cases} \tag{5.9}$$

where $F_{[P]}$ is the average fitness of the population, θ_{del} is the individual deletion threshold, and δ is and user-defined scaling factor. This deletion scheme biases the search towards highly fit individuals and, at the same time, balances the individuals' allocation in the different niches (Butz et al., 2004).

5.3.3 Rule Compaction Mechanism

Due to the intrinsic Michigan-style LCS nature of XCScds, the system evolves a large set of overlapping rules that have to be incrementally processed. The first step is to compact the discovered cluster identifiers by merging the individuals that are close by means of the Clark distance:

$$\text{distance}(cl_1, cl_2) = \sqrt{\sum_{i=1}^{n} \frac{(cl_1.c_i - cl_2.c_i)^2}{(cl_1.c_i + cl_2.c_i)^2}}.$$

If rule i and rule j have a distinct label and if the distance between them is lower than the user-defined threshold θ_{dis}, these two labels are merged. Notice that, after this compaction, two or more individuals may identify the same cluster (i.e., their centroids are near). After this first label compaction, the final step is to fuse rules that: (1) have the same cluster identifier and (2) overlap. It is important to highlight that this rule compaction, differently from the ones found in the literature (Shi et al., 2011, Tamee et al., 2006, 2007a,b), works in an incremental, online way.

5.3.4 Cost of the Algorithm

In common with other Michigan-style LCS (Orriols-Puig et al., 2009b), the cost of our algorithm increases linearly with (1) the maximum population size N and (2) with the number of variables Num_{var} used per rule, as depicted in Eq. 6.27 using the well-known big-O notation:

$$Cost_{XCScds} = O(N \cdot Num_{var}). \tag{5.10}$$

It is important to highlight that our version of XCScds does not depend directly on the number of transactions, which makes it very competent for mining huge databases. Online learners can stall the learning stage whenever required and the evolved rule set can be used to model the environment. In this regard, the more learning iterations XCScds has performed, the more general and accurate the clusters obtained should be.

5.3.5 Insights on Why XCScds Works

Michigan-style LCSs are open frameworks that foster crossbreeding between different learning paradigms. As it was discussed in chapter 3, Butz (2006) identified a set of pressures that guide Michigan-style LCSs to obtain accurate results and that explain why Michigan-style LCSs work. These are the following: the fitness pressure which pushes [P] towards more accurate individuals, the set pressure and the subsumption pressure which pushes [P] towards generalisation, the mutation pressure which pushes towards more specific solutions and the deletion pressure which pushes [P] towards fittest individuals. Notice that this deletion scheme erases individuals that are no longer useful due to a concept drift. Despite the fact that these studies are referred to XCS—the most studied LCS and a particular implementation of Michigan-style LCS—these can be extrapolated to other systems that follow the same framework.

5.4 Experiments

XCSc is a competent clustering algorithm that has an inherent online architecture that makes it scalable, as it can be found elsewhere (Shi et al., 2011, Tamee et al., 2007a). Despite the attempts on facing online clustering under dynamic situations, non-fully successful approaches regarding XCSc have been presented so far. In this section, we test XCScds under *pure* data stream problems, supporting its competitive behaviour. In order to test the proposed algorithm, two sets of experiments have been set. The first set consisting on two distinct environments, is devoted to test the XCScds behaviour under a synthetic and controlled situation. Hence, XCScds is connected to a synthetic online environment that incorporates changes in the target concept—typical in data streams—and also, in the second environment, in the middle of the run, a new cluster is forced to appear, making the learning process much more complex. The second set uses XCScds to determine the optimal data partition configuration for the real distributed storage system used in the Smart Grid domain (Navarro et al., 2011, 2013b). These experiments are further elaborated in subsequent sections.

5.4.1 Methodology of Experimentation

As any Michigan-style LCSs, XCScds has several configuration parameters which enable it to adjust the behaviour of the system to evolve models of maximal quality. The configuration parameters have been obtained experimentally (not shown for brevity) following the recommendations found in (Orriols-Puig, 2008). This study detected that Michigan-style LCSs are sensitive to the generalisation in initialisation r_0 and the fitness pressure ν whereas the setting of the other parameters had little effect on the final behaviour. With this information in mind we selected values that, on average, allow XCScds to perform well on all the problems. Related to his issue, Stalph et al. (2012) recently demonstrated via formal piecewise modelling that every XCS-related algorithm can be configured properly following a series of recommendations relying mostly in (1) r_0, (2) θ_{GA}, and (3) the population size. For further information on this concern the reader is referred to (Stalph et al., 2012). In this regard XCScds was configured with the following parameters for all the experiments: $\delta = 0.1$, $\alpha = 0.1$, $\beta = 0.01$, $\nu = 5$, $P_\chi = 0.8$, $P_\mu = 0.04$, $\theta_{GA} = 60$, $\theta_{del} = 15$, $\{\theta_{sub}, \theta_{exp}\} = 30$, $\theta_{dis} = 0.167$, $\epsilon_0 = 0.1$, $m_0 = 1$, $r_0 = 0.167$, and the population size was set to 400 individuals. This configuration was selected after an empirical test with multiple configurations. We performed 10 distinct runs keeping the configuration parameters constant, using each time a different random seed, and the results provided are the average of these runs.

5.4.2 Experiment 1: Clustering Sythetic Data Streams

Following the classic literature of clustering data streams (Bifet et al., 2010), to test XCScds we coded the online test-bed synthetic environment in which target concepts drift at different speeds. This environment consists of two continuous variables x_1 and x_2 ranging in $[0, 1]$. The stream lasts for 30 000 data samples. Every 10 000 instances there is a concept drift, thus changing the centers of the patterns. Variables x_1 and x_2 take values using a Gaussian pseudo-random generator in the following way:

1. During the first concept, the first pattern (C_1) is centered at $(0.35, 0.65)$ and the second pattern (C_2) is centered at $(0.75, 0.35)$.
2. Then, a drift occurs and centers are moved in the following way: C_1 is displaced and centered at $(0.35, 0.45)$, and C_2 is centered at $(0.78, 0.45)$.

3. Afterwards, a major drift happens; thus C_1 center is displaced to $(0.35, 0.25)$ and C_2 center is displaced to $(0.80, 0.80)$.

Also, the spread of the distinct C_i patterns varies from time to time: in the first and second concepts it is set to 0.04, and in the final one it is set to 0.02, adding an extra difficulty to the learner. It is important to mention that every pattern is generated entirely at random for each different concept and the distinct samples do not appear in an ordered way.

This first experiment was set to evaluate the following two aspects of our version of XCScds: (1) the capacity of the algorithm to discover the correct number of patterns, and (2) the adaptivity of the technique to distinct concepts. These are further elaborated in the following.

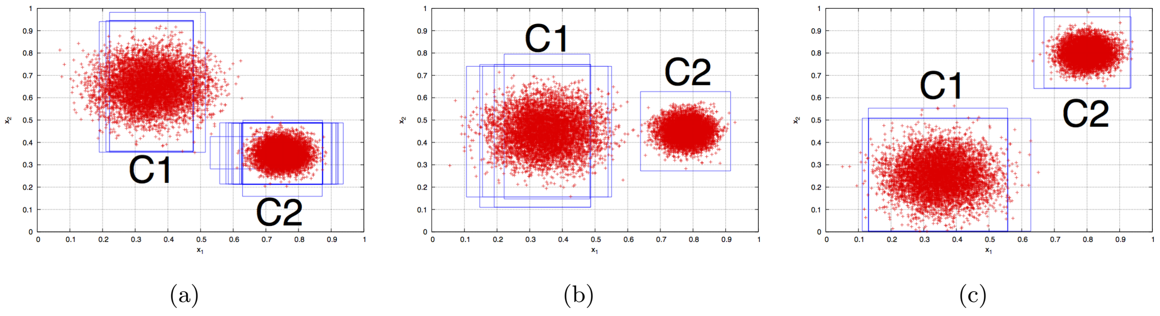


FIGURE 5.4: Results of the data stream experiment at the end of (a) the first concept, (b) the second concept, and (c) third concept drift. Results are averages of 10 runs. Blue lines are the boundaries of the discovered clusters.

Analysis of the Results

Results for this experiment are depicted in Figure 5.4. The aspects to validate are listed in the following.

- First, it is important to highlight that XCScds does not know the correct number of clusters *a priori* and consequently the algorithm has to discover them by itself. As depicted in Figure 5.4, XCScds was able to correctly identify the two distinct patterns. Notice that more than a single classifier points to the same label; that is an issue of the online compactation mechanism.

- Second, XCScds was completely able to detect the drifting concepts (the moving C_i and varying spreads) and adapt to these, as Figure 5.4 shows.

5.4.3 Experiment 2: Evolving Component in Clustering Synthetic Data Streams

We adapted the aforementioned experiment by adding an evolving component in it; that is: in the middle of the run a new pattern emerges in a sudden drift and XCScds has to properly detect and label it. As before, this problem consists in two primary and distinct Gaussian patterns that vary over time, and a third one that suddenly appears at the beginning of the first drift. These patterns are defined by two continuous variables x_1 and x_2 ranging in $[0, 1]$, and the data stream lasts for 50 000 data samples. Likewise, every 12 500 instances there is a concept drift, thus changing the centers of the patterns. Variables x_1 and x_2 take values in the following way:

1. During the first concept, the first pattern (C_1) is centered at $(0.1, 0.8)$ and the second pattern (C_2) is centered at $(0.9, 0.2)$.
2. During the second concept, C_1 is displaced and hence centered at $(0.1, 0.6)$. The same happens with C_2, which in this drift is centered at $(0.9, 0.4)$. In this new arrangement, a new cluster appears (C_3) centered at $(0.5, 0.5)$.
3. Next, during the third concept, C_1 is displaced again and centered at $(0.1, 0.4)$. C_2 is also displaced and centered at $(0.9, 0.6)$. C_3 displaces as well to the center $(0.5, 0.4)$.
4. Finally, in the fourth and last concept, C_1 is displaced and centered at $(0.1, 0.2)$, C_2 is displaced and centered at $(0.9, 0.8)$, and C_3 is displaced and centered at $(0.5, 0.6)$.

In all cases, the spread of the distinct C_i patterns is 0.02. It is important to highlight two distinct aspects of this problem:

1. Both C_1 and C_2 have 6 250 examples per concept during the first concept—C_3 has no examples during this concept—whereas in the rest of the experiment C1 and C2 have 4 167 examples per concept and C3 has 4 166 examples per concept.
2. Every pattern is generated entirely at random for each different concept and without assuming any order in the appearance of the distinct samples.

This second experiment is intended to be validated by means of evaluating three distinct aspects: (1) the capacity of the algorithm to discover the correct number of patterns, (2) the adaptivity of the technique to distinct concepts, and (3) the ability of the method to handle new patterns that appear suddenly in the middle of the run. These are further elaborated in the following.

Analysis of the Results

Figure 5.5 shows the results obtained by XCScds in this experiment. The three aspects to evaluate are in what follows:

- XCScds was capable of discovering and identifying by itself the total number of patterns in the data stream. As Figure 5.5 depicts, there is more than a single classifier labelling the same pattern. This issue is due to the soft rule compaction mechanism which respects the online nature of the problem.
- This problem models an extreme environment where all the concepts drift from one position to another. The results of XCScds shown in Figure 5.5 tell us that the algorithm adapted nicely to the different changes in concept.

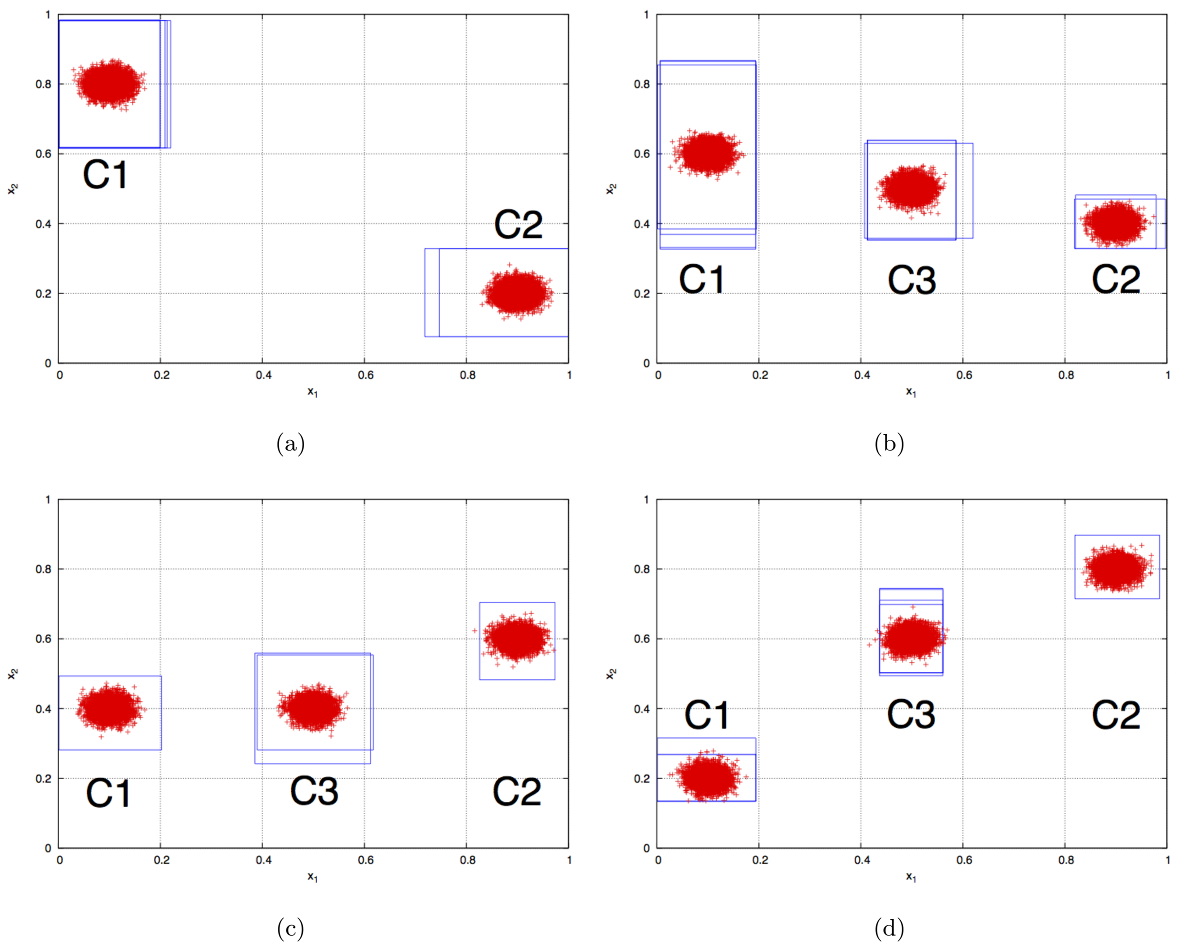


FIGURE 5.5: Results of the evolving data stream experiment at the end of (a) the first concept, (b) the second concept, (c) third concept and (d) the fourth concept. Results are averages of 10 runs. Blue lines are the boundaries of the discovered clusters.

- The evolving behavior of XCScds is clearly displayed in this experiment: Figures 5.5 (a) and 5.5 (b) show that the new pattern is detected and labeled accordingly without assuming any number of predefined clusters.

This experiment supported the adaptivity of XCScds to stream clustering and also its evolving component, concepts that are inherent to the Michigan-style LCS family. It is also important to mention the readability of the output rules obtained by the system. This architecture is validated on a real environment in the subsequent experiment.

5.4.4 Experiment 3: Online Clustering in a Real Environment

The distributed storage architecture specifically designed for the Smart Grid proposed in (Navarro et al., 2011) and deployed in (Navarro et al., 2013b) relies on a user-defined data partitioning layout. However, when (1) the amount of data rockets, (2) data access patterns change suddenly, and (3) an effective solution is requested at anytime—as typically happens in Smart Grids—establishing this predefined configuration becomes unfeasible. Therefore, this experiment aims to use the presented online XCScds system to continuously analyse the data access patterns (i.e., object identifier and operation type) to discover which data items can be placed together inside every partition of the aforesaid distributed storage system (Navarro et al., 2011).

In order to obtain comparable results with the offline approach conducted in (Louis-Rodríguez et al., 2013), we have used the Yahoo Cloud Serving Benchmark (YCSB), which is a benchmark used to evaluate the performance of different key-value and cloud serving stores that are very similar to our Smart Grids' distributed storage system (Navarro et al., 2011). Hence, data generated by the YCSB models the behaviour of any smart device collecting information from the Smart Grid and delivering it to the PAA.

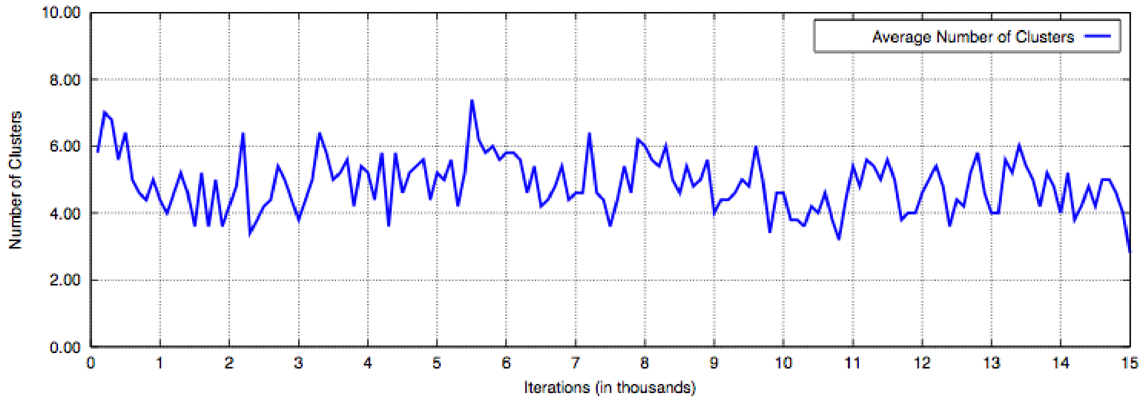


Figure 5.6: Results of the data stream experiment in the real environment. The curve is the average of 10 runs.

Analysis of the Results

Figure 5.6 shows the number of clusters proposed (i.e., partitioning layout) by the system according to what it is continuously learned from the incoming data. Note that the sharpness of the plot is due to the fact that the system is continuously learning from the incoming data and attempting to obtain the best partitioning layout. Note that the solutions proposed by our XCScds (from three to seven partitions) are above the optimal solution (three partitions) achieved by the offline strategies proposed in (Louis-Rodríguez et al., 2013). This is due to the fact that offline strategies used in (Louis-Rodríguez et al., 2013) are not aware of concept drifts nor permit any multi-partition operations, thus resulting in a too restrictive solution—that coincides with the minimum number of partitions discovered by the XCScds online approach. Hence, the layout proposed by the optimal solution results in a worst case scenario, which validates the correctness of our approach. Nonetheless, the discovered solutions suggest that if a dynamic partitioning scheme was allowed in the distributed storage system, smaller partitions could be used, which may derive on an improved throughput (Brewer, 2012) and a better usage of storage resources. Additionally, note that Figure 5.6 does not take into account the overhead associated to the cost of reconfiguring every partition (i.e., it is assumed that partitions are instantaneously rearranged). Therefore, when deploying the proposed XCScds in a real-world scenario, the data storage repository should stop delivering data to the online classifier system while conducting partition readjustments (Das et al., 2011, Kikuchi and Matsumoto, 2012).

5.5 Summary, Conclusions and Critical Analysis

5.5.1 Summary and Conclusions

This chapter presents XCScds—an online version of XCSc—, a successful clustering approach deployed on top of the Smart Grid following a multi-agent system layout, aimed at establishing a dynamic data partitioning scheme for the storage repository. We have shown that previous works based on offline strategies (e.g., Metis, hMetis, Round Robin) (Louis-Rodríguez et al., 2013) derive into too restrictive solutions with very few and large partitions, which limits the performance of the storage system. Hence, considering the dynamic partitioning scheme proposed here that relies on the XCScds's ever-evolving knowledge model, the number of partitions can vary according to the workload demands, which results in smaller partitions and increased throughput. Indeed, we have seen that this evolving strategy of the XCScds fits very well with the dynamic nature of distributed systems. To the best of our knowledge this is the first approach to obtain approximate online clusters by using XCScds.

As continuously reconfiguring the data partitioning scheme (i.e., moving data objects from one partition to another) may be a time costly operation (Das et al., 2011, Kikuchi and Matsumoto, 2012), we plan to include some penalty metrics to the XCScds in order to make it aware of this concern. Conducted preliminary experiments envisage that this strategy would smooth the results of the clustering process and improve the overall throughput of the storage system. Also we envisage the use of other kinds of representation for XCScds in order to test whether the system is capable of compacting the population of individuals even further without a significant loss on the precision of the whole system.

5.5.2 Critical Analysis of XCScds

To finish the study of XCScds a final analysis is performed in the following. Table 5.1 summarises it, where *strengths* represent the main advantages of XCScds, *weaknesses* show the drawbacks of the system, *opportunities* suggest the further lines of research to enhance the system performance, and *threads* indicate issues that other machine learning approaches may take advantage of.

Our implementation of XCScds has four main strengths: first, XCScds was capable of learning the right number of clusters without assuming any a priory number of these in an online environment. Second, the evolving component makes XCScds capable of adapting itself to previously unknown patterns. Third, XCScds' online architecture has demonstrated to be capable of handling large data sets efficiently with a relatively low cost. And Fourth, XCScds uses interval-based rules which are human-friendly (i.e., readable), which makes the algorithm suitable in almost any problem, and specially in those that require of a high degree of readability.

Strengths	Weaknesses
- It demonstrates a competitive behaviour, without assuming any *a priory* number of clusters. - It deals with large data sets. - It has an evolving component that allows the algorithm to adapt to new clusters. - It uses interval-based rules which are human-friendly.	- It generates a large amount of rules, which may hamper the readability of the result. - It may suffer from the *fitness dilemma* as it happens in XCS. - It has several parameters that have to be properly configured.
Opportunities	**Threats**
- The experimentation performed suggests that XCScds can be applied to a large number of real-world data stream problems. - It can be extended to avoid using all the variables in its rules. - Its flexible architecture allows the hybridisation towards a *semisupervised* learning system.	- The large amount of parameters to be properly set may discourage practitioners.

Table 5.1: Critical analysis of XCScds.

The main weaknesses of XCScds are the following: first, it, depending on the problem and configuration used, may generate a large amount of rules. This issue may hamper the readability of the result. Second, as our proposed algorithm is heavily based on XCS it may suffer form the *fitness dilemma*. And third, is has several configuration parameters that have to be properly set in order to obtain accurate results. Configuring XCScds—and many traditional Michigan-style LCS—can require an expert.

As a complex system, XCScds has the main threat that it requires the user to properly set a large amount of parameters to obtain accurate results. Often, this is not a trivial task and requires expertise in these systems. This issue may discourage practitioners–specially those unexperienced—as opposed to much simpler although not as accurate nor flexible systems.

Finally, the experimentation performed suggest that XCScds has the opportunity of being applied to a large number of real-world data stream problems, making the system robust. Also, with the addition of variable-sized rules–in opposition of the fixed sized rules—, the proposed algorithm can be further enhanced for tackling high-dimensional data sets. Moreover, the flexible architecture of XCScds allows the hybridisation towards a semisupervised learning system without much effort, which is of great interest since combining the supervised and unsupervised paradigms leads to a method that minimises their individual limitations.

Clustering data streams is an appealing field due to its applicability in real-world stream problems without suffering the dearth of training examples when a concept drift occurs. Moreover, the interval-based clustering rules used by XCScds is attractive from the point of view of readability. However, in certain problems this kind of representation may be not the desired one, specially when a large amount of rules are involved, thence requiring a much more readable system. In fact, in these problems is more important to describe the knowledge obtained from data in a very readable manner than having a high degree of accuracy. In the following chapter we introduce the field of *association streams* by means of Michigan-style LCSs, where the readability is a critical part of the learning process.

6 A Prospective Approach to Association Streams

The uprising bulk of data generation in industrial and scientific applications has fostered the interest of practitioners for mining large amounts of unlabelled data in the form of continuous, high speed and time-changing streams of information. An appealing field is association stream mining, which regards on modelling dynamically complex domains via production rules without assuming any a priori structure. Differently from the related frequent pattern mining field, its goal is to extract interesting associations among the forming features of such data adapting these to the ever changing dynamics of the environment in a pure online fashion—i.e., without the typical two-step process for rule generation. This chapter describes Fuzzy-CSar, a Michigan-style fuzzy-classifier system designed to extract interesting fuzzy association rules from streams of examples. This algorithm evolves its internal model online, so it is able to quickly adapt its knowledge in the presence of drifting concepts and noisy inputs. The different complexities of association stream mining are presented in a set of novel synthetic benchmark problems. Thus, the behaviour of the online learning architecture presented here is carefully analyzed under these conditions. Furthermore, the analysis is extended to real-world problems with static concepts, showing the competitiveness of this technique in terms of the quality of the rules produced. Experiments support the advantages of applying Fuzzy-CSar to extract information from large volumes of information.

The remainder of this chapter is organised in the following way: Section 6.2 provides the basic concepts of association rule mining (in both qualitative and quantitative environments), details the ways of inferring association rules from data and describes the difficulties of learning from data streams and association streams. Section 6.3 gives a detailed description of the proposed Fuzzy-CSar algorithm. Sections 6.4, 6.5 and 6.6 provide the results of the experiments done with Fuzzy-CSar, which show its competitiveness in a variety of problem situations, whether the data are static or in dynamic streams. Finally, Section 6.7 summarises and concludes the chapter.

6.1 Introduction to Association Streams

As discussed earlier in this thesis, the invention of digital computers allowed industry to collect and store massive amounts of information of the business processes in which it was involved for a subsequent analysis and exploitation. Since then, the analysis and exploitation of large databases is a recurrent topic and there have been several contributions in a wide set of disciplines (Aggarwal and Yu, 2001, Motamedi et al., 2012), ending in the field of data streams (Angelov, 2012).

Despite the need of addressing the complexities of mining new, potentially useful information from data streams, current online mining field research focus mainly on supervised methods, which assume an *a priori* relational structure for the set of features that define the problem. As depicted in the previous chapter, this issue is often distant from real-world situations, and it is specially emphasized when changes in concept occur: the undefined and ill-structured nature of the situation jointly with the dearth of labelled examples—or complete absence of them—from this new concept makes the application of a *pure* supervised algorithm ill-suited, hence requiring the use of hybrid methods which combine supervised with unsupervised techniques in order to deal with the problem (Shi et al., 2010). A real case that entails a sound example of this issue is in detecting potential threats to web sites and network infrastructures (Corral et al., 2011). In this real-world scenario there are a set of features that indicate suspicious acts upon the infrastructure (i.e., strange characters in login interfaces or strange traffic flows, among others) by malicious users trying to identify the vulnerabilities of the system to take possession of it. It is important to highlight that other anomaly detection strategies exist, such as statistical or based on data density. However these are typically based on labelled data and, therefore, they do not adapt themselves to concept changes. In our particular case we are interested not in directly detecting—and thus informing of the attack—but in adapting to the new trends that data are describing, which help experts deciding if the system is under attack. In this regard, the unsupervised approach becomes a feasible alternative to the aforesaid issues. More specifically, *association stream mining*, focused on extracting associations among variables via production rules online, is specially attractive from the point of view of practitioners due to (1) the demand of interpretability of the patterns discovered in data—the human-readable rules obtained fit that requirement (e.g., *every time X occurs Y also happen*)—, (2) the need for discovering patterns while they are happening (e.g., the new steps of the web attack) and hence adapt to them, and (3) the high and continuous volumes of data to be processed, which demand highly scalable learners.

Although the importance on facing association streams, it is a new area and consequently, to the best of our knowledge, no fully operational approaches tackling the challenges discussed in this thesis have been presented so far. The most similar algorithms for knowledge extraction under these conditions focuses mainly on the identification of frequent variables—referred to as *online frequent pattern mining*—, relegating the generation of rules in a second place in an offline process (Gama, 2010), which make these mostly unpractical for handling association streams premises, where the rules discovered have to be present immediately and adapt to the changing dynamics of the continuous flow of data. Several investigations have been presented so far on frequent pattern mining, making this subject into a *heavy hitter* (Chi et al., 2004), but the majority of these ignore the presence of association drifts. Moreover, most of those algorithms are only able to deal with problems described by categorical features (Cheng et al., 2008, Cormode and Hadjieleftheriou, 2010, Deypir and Sadreddini, 2012, Fan et al., 2009, Farzanyar et al., 2012, Li and Lee, 2009, Li et al., 2008, 2009, Singh et al., 2011, Tsai et al., 2009, Wang et al., 2004, Wang and Chen, 2011). Even so, a few quantitative algorithms for frequent pattern mining have been proposed, most of them based on fuzzy logic (Chandra and Bhaskar, 2011, Leung and Boyu, 2009, Peng et al., 2010), focusing on the identification of frequent variables and not on the final rules. Likewise, these systems are not designed for the harsh conditions of continuous flow and time-changing concepts that association streams stresses.

Association streams research area is rooted in traditional, offline association rule mining, one of the most well known Data Mining fields. It is used to extract interesting information from large data bases by means of describing the properties of data in the form of production rules, e.g., $X \Rightarrow Y$, where both X and Y are sets of features and $X \cap Y = \emptyset$ (Agrawal et al., 1993, Cios et al., 2007, Han et al., 2006). Association rule mining algorithms do not assume any *a apriori* structure about the problem. Thus, these methods automatically uncover connections between features (Orriols-Puig et al., 2012). Two main types of representations are found in the literature: qualitative and quantitative association rules. Qualitative association rules are only able to handle problems defined by binary sets of items. A typical qualitative rule example is "[*butter*], [*bread*] $\Rightarrow$ [*milk*]" (Agrawal and Srikant, 1994, Agrawal et al., 1993). In order to deal with continuous features, practitioners switched to quantitative association rules, by exploring two different strategies: (1) discretize the features and then deal with them in a purely qualitative fashion (Miller and Yang, 1997, Srikant and Agrawal, 1996, Wang et al., 1998), which may lead to poor results due to the loss of information, and (2) using an interval-based representation (Martínez-Ballesteros et al., 2011a, Mata et al., 2002, Orriols-Puig et al., 2008a, Yan et al., 2009). A typical quantitative rule example using the interval-based representation is "$weight \in [70, 90]\, kg \Rightarrow height \in [170, 190]\, cm$." Later on, fuzzy modeling (Zadeh, 1965) was introduced in order to create highly legible models from domains with uncertainty and imprecision and to avoid unnatural boundaries produced by the interval-based representation. A typical fuzzy rule example is "*quality* **is** *middle* $\Rightarrow$ *price* **is** *high*" (Alcalá-Fdez et al., 2008, Dubois et al., 2006, Hong et al., 2001, 2008, Kuok et al., 1998).

The purpose of this chapter is to follow up the work started in Orriols-Puig and Casillas (2010a) by addressing the challenges of association streams extending the analysis of Fuzzy-CSar, a *Michingan-style learning classifier system* (LCS) (Holland, 1992) that makes use of *genetic algorithms* (GAs) (Goldberg, 1989, 2002) to evolve *independent-fuzzy-rule* sets online from streams of data. Inheriting the learning architecture of XCS (Wilson, 1995, 1998) and Fuzzy-UCS (Orriols-Puig et al., 2009b), Fuzzy-CSar is a general purpose unsupervised learning algorithm specialised in the extraction of highly interesting fuzzy association rules in a single step, that is, Fuzzy-CSar does not generate lists of frequent itemsets but it directly evolves the set of association rules, which results in an effective way to tackle association streams.

The behaviour of Fuzzy-CSar under a set of online benchmark problems is first studied. Each of these problems face a different aspect of the aforementioned challenges in the field of association streams. Thereafter, the analysis is extended by comparing algorithm scalability and quality of results of Fuzzy-CSar by the ones of Fuzzy-Apriori (Hong et al., 2001), one of the most well known fuzzy association rule mining algorithms, in a set of real-world problems with static concepts. These experiments will demonstrate how competitive Fuzzy-CSar is in both online and offline environments.

This chapter provides the following contributions:

- An introduction to association streams is provided: a new field inside data streams strongly related to online frequent pattern mining. In this regard and due to the dearth of benchmark problems, a framework for algorithm evaluation is proposed.
- An algorithm for handling association streams is detailed based on the Michigan-style LCS framework, an online and mature architecture that has demonstrated that it can be successfully applied to data streams (Orriols-Puig and Casillas, 2010b).

In the forthcoming section, the required background to follow the chapter is detailed providing the basic definitions of association rule mining, quantitative association rules, fuzzy logic and fuzzy association rules. Also, it details the ways of inferring association rules from data and describes the difficulties of learning from data streams. Finally, a formal description of association streams is given.

6.2 Framework

This section introduces the important concepts required to follow this work. These are related to (1) association rule mining and fuzzy logic, (2) rule discovery mechanisms, (3) learning from data streams, and (4) association streams. These concepts are elaborated in the following.

6.2.1 Association Rules: A Descriptive Introduction

Association rule mining consists in extracting interesting associations among input variables rules typically from large databases in the form of rules. The problem of obtaining these associations was first formalized by Agrawal et al. (1993) in the following way: Let $\mathcal{I} = \{i_1, i_2, \cdots, i_\ell\}$ be a set of ℓ items—an item represents a column in a database—and let $\mathcal{D} = \{d_1, d_2, \cdots, d_N\}$ be a set of N transactions, where each d_j contains a subset of items, that is, $d_j \subseteq \mathcal{I}$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$ and $X \cap Y = \emptyset$.

Traditionally, association rules are assessed by two measures of interestingness: their *support*—the frequency of occurring patterns—and their *confidence*—the strength of the implication denoted in the rule. Support is defined as follows:

$$\sup(X) = \frac{n(X)}{|\mathcal{D}|}, \tag{6.1}$$

where $n(X)$ is the number of times X appear in $\mathcal{D}$, and the confidence of a rule is defined as:

$$\text{con}(X \Rightarrow Y) = \frac{\sup(X \cup Y)}{\sup(X)}. \tag{6.2}$$

Therefore, support indicates the frequency of occurring patterns and confidence evaluates the strength of the implication denoted in the association rule. It is worth noting that other quality measures can be used instead of the confidence to represent the reliability of the rule. The most popular ones are lift (Brin et al., 1997) and accuracy (Geng and Hamilton, 2006). Lift measures how many times more often X and Y appear together in the data than expected if they where statistically independent. Lift is reckoned as

$$\text{lif}(X \Rightarrow Y) = \frac{\sup(X \Rightarrow Y)}{\sup(X) \cdot \sup(Y)}. \tag{6.3}$$

Accuracy measures the degree of matching between the rule and the different transactions in $\mathcal{D}$, and it is computed as

$$\text{acc}(X \Rightarrow Y) = \sup(X \Rightarrow Y) + 1 - \Bigg(\sup(X) + \sup(Y) - \sup(X \Rightarrow Y)\Bigg). \tag{6.4}$$

The most well-known classic association rule miner is Apriori (Agrawal and Srikant, 1994). Apriori is focused on identifying strong relationships between the occurrence of two or more items on collections of binary data—coined to as *qualitative association rules*—by a generate-and-test procedure. Apriori performs multiple scans to the database in order to obtain the frequent itemset candidates. Most often, there is a high number of candidates, so support counting for candidates can be time expensive. After the frequent itemsets are found, the algorithm combines these to produce the desired rules.

However, most industrial and scientific applications make use of continuous data that Apriori and its derivates cannot handle. In order to deal with continuous features, researchers concentrated to quantitative association rules, by exploring three different strategies: (1) discretise the features and then deal with them in a purely qualitative fashion (Srikant and Agrawal, 1996), (2) using an interval-based representation (Mata and Riquelme, 2001) and (3) by using fuzzy logic. Because the first strategy may lead to poor results due to the loss of information of the discretisation process practitioners focused on the other two alternatives.

6.2.2 Quantitative Association Rules by Means of Intervals

In order to mine numeric data, the aforementioned concepts were extended for quantitative association rules that make use of a representation consisting of intervals: Let $\mathcal{F} = \{f_1, f_2, \cdots, f_\ell\}$ be a set of ℓ continuous features and let the following two disjoint subsets $\mathcal{A}$ and $\mathcal{C}$—for the antecedent and for the consequent part of the rule, respectively—be subsets of $\mathcal{F}$, that is $A \subset \mathcal{F}$, $C \subset \mathcal{F}$ and $\mathcal{A} \cap \mathcal{C} = \emptyset$. A quantitative association rule is then defined as an implication of the form $X \Rightarrow Y$ in which:

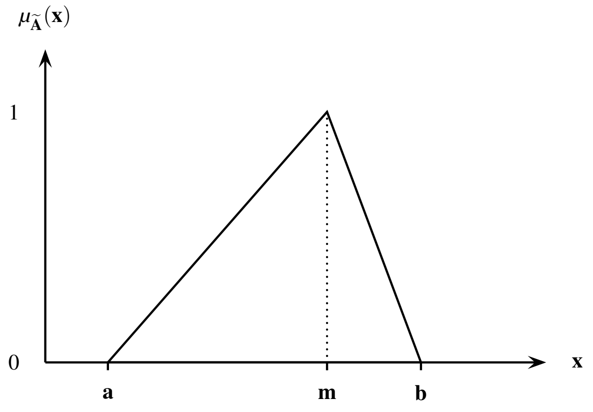
$$X = \bigwedge_{f_i \in \mathcal{A}} f_i \in [\ell_i, u_i] \quad \text{and} \quad Y = \bigwedge_{f_j \in \mathcal{C}} f_j \in [\ell_j, u_j], \tag{6.5}$$

where ℓ_i and u_i are the lower and upper interval bounds of antecedent features (f_i) respectively, and ℓ_j and u_j are the lower and upper interval bounds of consequent features (f_j) respectively. It is worth mentioning that the measures of interestingness are the same as in the case of qualitative association rules.

6.2.3 Fuzzy Logic and Association Rules

The ultimate goal of knowledge discovery and data mining is to find useful patterns from data. However, real-world problems are *flooded* with imprecision and uncertainty that hamper the learning of traditional machine learning algorithms. In this sense, fuzzy logic is concerned with imprecision and approximate reasoning, allowing practitioners to represent subjective concepts in a formal, mathematical way.

In fuzzy set theory, each fuzzy set A of a bivaluated logic set X (i.e., a *crisp* set) is characterised by giving a *degree of membership* to each of its elements x ∈ X. This is addressed by means of a *fuzzy membership function*, which returns the degree of truth in the range $[0, 1]$ of the given element to the fuzzy set A, being 0 the absolute falsity and 1 the absolute certainty, that is: $\mu_{\widetilde{A}} : X \to [0, 1]$ (Orriols-Puig, 2008). There are a myriad of possible membership functions to choose from, but typically practitioners select simple functions to avoid complicated calculations. For that reason, the most common are the triangular membership functions (Cordón et al., 2001). Figure 6.1 show the typical definition of such membership function.



$$\mu_{\widetilde{A}}(x) = \begin{cases} 0 & \text{if } x \le a; \\ \frac{x-a}{m-a} & \text{if } a < x \le m; \\ \frac{b-x}{b-m} & \text{if } m < x < b; \\ 0 & \text{otherwise.} \end{cases}$$

Figure 6.1: Triangular-shaped membership function.

In the same way as crisp sets several propositions can be combined by connectives under fuzzy logic (e.g., conjunction, disjunction, negation and complement). In this sense fuzzy logic gives a mathematical interpretation to these so that a new membership degree can be calculated from several propositions joined by connectives (Orriols-Puig, 2008).

In the fuzzy logic domain each variable has a fuzzy partition representing the fuzzy set associated to each of its linguistic terms[1]. Figure 6.2 depicts a representation of a fuzzy partition for a variable with three uniformly distributed triangular-shaped membership functions using three fuzzy labels (*small*, *medium* and *large*).

Fuzzy rule-based systems are traditionally composed by three main components: an input interface, a database and an output interface (Orriols-Puig, 2008):

- The input interface transforms the crisp input data into fuzzy sets by means of a fuzzification process.

[1]For a more detailed discussion on fuzzy logic the reader is referred to (Cordón et al., 2001).

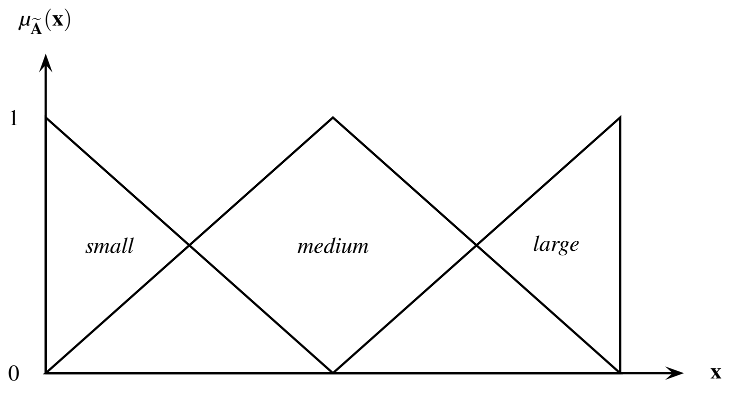


Figure 6.2: Representation of a fuzzy partition for a variable with three uniformly distributed triangular-shaped membership functions using the fuzzy labels *small*, *medium* and *large*.

- The database contains the definition of all the linguistic terms and membership functions defining the semantic of the linguistic labels.
- The output interface translates the fuzzy rule action to a real action by means of a defuzzification process.

The use of fuzzy logic in association rule mining allows the creation of highly legible models from both qualitative and quantitative data. Let $\mathcal{F} = \{f_1, f_2, \cdots, f_\ell\}$ be a set of ℓ features (either continuous or categorical), and let $A \subset \mathcal{F}$ and let $C \subset \mathcal{F}$, $A \cap C = \emptyset$. A fuzzy association rule is an implication of the form $X \Rightarrow Y$ where:

$$X = \bigwedge_{f_i \in A} \mu_{\widetilde{A}}(f_i) \quad \text{and} \quad Y = \bigwedge_{f_j \in C} \mu_{\widetilde{C}}(f_j), \tag{6.6}$$

where $\mu_{\widetilde{A}}(f_i)$ is the membership degree of features in the antecedent part of the rule and $\mu_{\widetilde{C}}(f_j)$ is the membership degree of features in the consequent part of the rule.

In this regard, support is typically extended by using the product T-norm (Dubois et al., 2006) as

$$\sup(X \Rightarrow Y) = \frac{1}{|T|} \sum \mu_{\widetilde{A}}(X) \cdot \mu_{\widetilde{C}}(Y), \tag{6.7}$$

where $\mu_{\widetilde{A}}(X)$ is the membership degree of the antecedent part of the rule and $\mu_{\widetilde{C}}(Y)$ is the membership degree of the consequent part of the rule. Similarly, confidence is extended by using the *Dienes implication* (Dubois et al., 2006):

$$\text{con}(X \Rightarrow Y) = \frac{\sum \mu_{\widetilde{A}}(X) \cdot \max\{1 - \mu_{\widetilde{A}}(X), \mu_{\widetilde{C}}(Y)\}}{\sum \mu_{\widetilde{A}}(X)}. \tag{6.8}$$

Lift and accuracy have each a fuzzy equivalent as well. Lift is extended as

$$lif(X \Rightarrow Y) = \frac{sup(X \Rightarrow Y)}{\sum \mu_{\widetilde{A}}(X) \cdot \sum \mu_{\widetilde{C}}(Y)}. \tag{6.9}$$

Accuracy is extended as

$$acc(X \Rightarrow Y) = sup(X \Rightarrow Y) + 1 - \left(\sum \mu_{\widetilde{A}}(X) + \sum \mu_{\widetilde{C}}(Y) - sup(X \Rightarrow Y) \right). \tag{6.10}$$

Fuzzy-Apriori (Hong et al., 2001) is the most well known extension of the original Apriori for mining fuzzy association rules.

6.2.4 Obtaining Rules from Data

There are mainly three different strategies to obtain association rules: (1) by means of candidate generate-and-test procedures, (2) by means of divide-and-conquer methodologies, and (3) by means of metaheuristics. In what follows, these strategies are briefly detailed.

The candidate generate-and-test family of procedures exploit the *downward closure* property which states that if an itemset is frequent, then all of its subsets are also frequent itemsets. These kinds of algorithms perform multiple scans to the database in order to obtain the frequent itemset candidates. Most often, there is a high number of candidates, so support counting for candidates can be time expensive. After the frequent itemsets are found, the algorithm combines these to produce the desired rules. The most well known algorithm of this family is Apriori.

In order to scale up the generation of frequent itemsets and to avoid unnecessary candidate generation, divide-and-conquer procedures build an auxiliary structure to get statistics about the itemsets, thus avoiding multiple scans of the data. This auxiliary structure is typically based on a k-ary tree. As in the case of candidate generate-and-test, after the frequent itemsets are found, the algorithm combines these to produce the desired rules. The most well known algorithm of this family is FP-Growth (Han et al., 2004).

Another way of avoiding the breeding of candidate generation is by making use of metaheuristics. One of the most successful families of such techniques are Evolutionary Algorithms (EAs). EAs can evolve the desired association rules directly from data without the two-step process of classic association rule mining algorithms. This is the case of LCSs, which are rule-based evolutionary learning systems. As it was detailed in chapter 2, LCSs are classified into two main branches: Michigan-style LCSs and Pittsburgh-style LCSs.

- Michigan-style LCSs are online cognitive-inspired systems that combine a credit apportionment algorithm with EAs. Each individual is a single production rule, whose quality is evaluated online by the cognitive system. An EA is applied periodically to the population to discover new rules. Michigan-style LCSs have attracted the interest of practitioners for mining association rules recently (Orriols-Puig and Casillas, 2010a).

- Pittsburgh-style LCSs are offline EAs. Every individual is a set of production rules that represent the complete solution for the given problem, thus all the solutions compete in the population. The genetic search is usually driven by a generational GA. An example of the use of a Pittsburgh-style LCS for mining association rules is found in (Yan et al., 2009).

More recently, other EA strategies inspired by traditional LCSs have flourished: the Iterative Rule Learning (IRL) (Venturini, 1993) approach which has been broadly used in association rule mining (Martínez-Ballesteros et al., 2011a). Similarly to Michigan-style LCSs, in IRL each individual is a single production rule, and similarly to Pittsburgh-style LCSs, the genetic search is driven by a generational GA. IRL iteratively performs two steps: (1) learn a rule that covers part of the training examples via the genetic discovery mechanism and (2) remove the covered examples from the training set. Another popular strategy for mining association rules is the Genetic Cooperative-Competitive Learning (GCCL) branch (Greene and Smith, 1993). This approach combines the offline rule processing of Pittsburgh-style LCSs with the idea of Michigan-style LCSs that the solution is the whole population, and so, that rules need to collaborate to cover all the input space. Recently, the GCCL approach has been successfully applied for mining association rules that model consumers' behaviour (Casillas and Martínez-López, 2009).

6.2.5 Learning from Data Streams

Learning from online streams of data presents a series of difficulties due to the continuous, potentially unbounded in size, high speed, noisy, and time-changing nature of data streams. Classical machine learning methods are not able to extract accurate models when the information comes in the form of data streams, and these have to be adapted. Typically, the use of time windows to store parts of the stream is the most common way to adapt an offline algorithm. However, managing time windows has some inherent decision challenges that have to be addressed. For example, a key aspect lies in deciding the size of the time window because it controls the ability of forgetting past examples that are no longer useful (Núñez et al., 2007). Also, deciding which examples have to be forgotten is not a trivial task. On the other hand, incremental, online learners take advantage by handling the streams directly. Another major challenge of data streams lies in the detection of changes in the target concepts (*concept drifts*), which can be abrupt, incremental or recurring (Gama, 2010). All the characteristics of data streams are included in the *massive online analysis* (MOA) framework (Bifet et al., 2010), which supplies a large amount of algorithms for stream classification and clustering.

Recent contributions (Lughofer and Angelov, 2011, Mozafari et al., 2011) tackle the issues of data streams by performing some incremental statistical test over the incoming streams. Due to the noisy characteristics of data streams, the use of fuzzy logic is broadly extended (Bouchachia et al., 2013). In this regard, there are several recent contributions in both supervised (Angelov and Xiaowei, 2008, Lughofer and Angelov, 2011, Orriols-Puig and Casillas, 2010b), and unsupervised (Baruah and Angelov, 2012a, Chandra and Bhaskar, 2011, Peng et al., 2010) learning paradigms.

6.2.6 Association Streams in a Nutshell

Association stream mining is a novel field closely related to both data streams and association rule mining that aims at extracting associations among variables in the form of rules. The main characteristics of this field are:

- Continuous flow of information that has to be processed in a single pass.
- There are restrictions in memory usage.
- The concept to be learnt may change over time.

- Does not assume an underlying structure nor distribution in incoming fluxes of data.

In addition, a high degree of interpretability is desirable. The main difference with frequent pattern mining is that rules are generated in an online fashion and in a single step, hence limiting the range of action of the learner. Notice that the notion of concept drift is also present: the learnt model $\mathcal{M}$ (that is, the set of rules that the algorithm has discovered so far) at time t_i may not hold at time t_j. We formalise these concepts in the following manner: let $t \in \{t_1, t_2, \ldots, t_T\}$ be the distinct time indexes. Let $\mathcal{F}^t = \{f_1^t, f_2^t, \ldots f_\ell^t\}$ be the set of ℓ features (both continuous and categorical). A fuzzy association stream rule is a relation $X^t \Rightarrow Y^t$, $X^t \cap Y^t = \emptyset$:

$$X^t = \bigwedge_{i_i^t \in A^t} \mu_{\widetilde{A}}(i_i^t) \quad \text{and} \quad Y = \mu_{\widetilde{C}}(i_c^t). \tag{6.11}$$

For convenience, we define the concept of an association stream rule simplifying the consequent part by forcing a single variable. The change on rule matching degree is directly related to the change in concept.

The measures of interestingness have to be adapted to the incremental nature of association streams as well. The incremental support is computed as follows:

$$\sup(X \Rightarrow Y)^{t+1} = \sup(X \Rightarrow Y)^t + \frac{matchingEstimate^{t+1} - \sup(X \Rightarrow Y)^t}{experience^{t+1}}, \tag{6.12}$$

where *matchingEstimate* is the incremental matching estimate of both the antecedent and the consequent part of the rule with respect to the examples seen so far and *experience* is the number of times that this particular rule has been triggered. The incremental confidence is calculated in the following way:

$$\mathrm{con}(X \Rightarrow Y)^{t+1} = \mathrm{con}(X \Rightarrow Y)^t + \frac{matchingEstimate^{t+1} - \mathrm{con}(X \Rightarrow Y)^t}{antecedentMatchingEstimate^{t+1}}, \tag{6.13}$$

where *antecedentMatchingEstimate* is the incremental matching estimate of the antecedent part of the rule with respect to the examples seen so far. The other measures of interestingness follow the same pattern. Notice that the measures of interestingness reflect concept drifts as well because rules that have high support and confidence at time t_i may have no longer that interestingness at time t_j, $i \neq j$.

This type of dynamic environment turns out to be of the upmost importance in many real-world applications (e.g., network security monitoring), being more practical than *pure* supervised methods when concept drift occurs. It is important to highlight that despite association stream mining is closely related to frequent pattern mining, this latter field is focused exclusively in the identification of the frequent variables, relegating the generation of the final rules in a second place using a classic offline approach, hence making it unfeasible for the challenges that association streams miners have to handle.

A similar challenging environment is in mining frequent itemsets from data streams due to the imposed constraints—single-pass limitation, memory limitations, combinational explosion of itemset candidates and handling drifting concepts (Gama, 2012). In frequent pattern mining, a fast data structure—typically a tree (Liu et al., 2011)—is used for identifying the frequencies of streaming itemsets and storing those with higher frequencies.

To the best of our knowledge, the closest related work to association streams is done in Fan et al. (2009), where a incremental principal component analysis method jointly with a multiple regression technique were used to mine ratio rules—a special case of quantitative association rules—under concept drifting data streams. Also related, in HewaNadungodage et al. (2013) an FP-tree to mine uncertain frequent patterns from data streams and obtain fuzzy association rules. However, when a concept drift occurs the rules obtained so far by the algorithm are all discarded and induced from scratch.

In the present work we present Fuzzy-CSar with the aim of exploiting the online nature and highly adaptive behaviour of fuzzy Michigan-Style LCSs to mine high-quality association rules from streams of unlabelled data. The learning architecture of Fuzzy-CSar is detailed in the next section.

6.3 Description of Fuzzy-CSar

Fuzzy-CSar is a Michigan-stye LCS designed for mining fuzzy association rules from data that contain both quantitative and categorical attributes by means of combining GAs and apportionment of credit mechanisms in an online fashion. Hence, Fuzzy-CSar evolves a population of fuzzy association rules in an incremental way and it is able to quickly adapt to concept changes. In what follows, the knowledge representation and the learning organisation are described in detail.

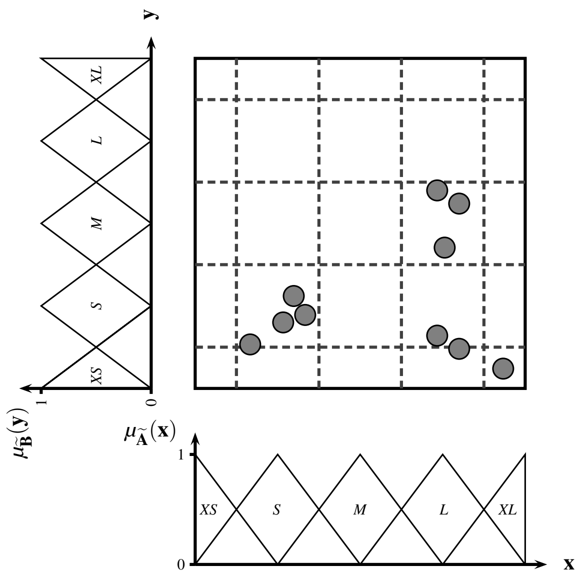
6.3.1 Knowledge Representation

Fuzzy-CSar evolves a *population* [P] of *individuals*, where each consists of (1) a *fuzzy association rule* and (2) a set of parameters that evaluate the quality of the rule. The fuzzy association rule is represented as

$$\textbf{if } x_i \text{ is } \widetilde{A}_i^k \textbf{ and } \ldots \textbf{ and } x_j \text{ is } \widetilde{A}_j^k \textbf{ then } x_c \text{ is } \widetilde{A}_c^k,$$

in which the antecedent contains a set of ℓ_a input variables $x_i, \ldots, x_j$ ($0 < \ell_a < \ell$, where ℓ is the number of variables) and the consequent consist of a single variable x_c which is not present in the antecedent. It is worth noting that Fuzzy-CSar allows rules to have an arbitrary number of input variables in the antecedent. As depicted, each variable is represented by a disjunction of *linguistic terms* or *fuzzy labels* $\widetilde{A}_i^k = \{A_{i1} \vee \ldots \vee A_{in_i}\}$. As an illustrative example, consider the following rule that relates the users' electricity consumption with its respective price: **if** *Demand* is {Medium or Large} and *Transfer* is {Very Small or Small or Medium} **then** *Price* is {Medium or Large}. To avoid creating largely over-general rules the system permits the restriction of the maximum number of linguistic terms per variable via the configuration parameter *maxLingTermsPerVariable*. All variables share the same semantics, which are defined by means of Ruspini's strong fuzzy partitions that satisfy the equality:

$$\sum_{j=1}^{n_i} \mu_{\widetilde{A}_{ij}}(x) = 1, \; \forall x_i. \tag{6.14}$$



if x is $\{S\} \Rightarrow y$ is $\{XS \text{ or } S\}$ [supp: 0.4; conf: 1]
if x is $\{L \text{ or } XL\} \Rightarrow y$ is $\{XS\}$ [supp: 0.2; conf: 0.3]
if y is $\{M\} \Rightarrow x$ is $\{L\}$ [supp: 0.3; conf: 1]

Figure 6.3: Knowledge representation used by Fuzzy-CSar in a two-dimensional problem using five linguistic labels per variable.

Each partition is a set of uniformly distributed triangular-shaped membership functions due to its interpretability tradeoff (Ishibuchi et al., 2009).

The matching degree of an example e with a individual k is computed as follows: for each antecedent variable x_i, the membership degree $\mu_{\widetilde{A}^k}(e)$ of each of its linguistic terms is computed, and then these are aggregated by means of a T-conorm (disjunction). In this work, the *bounded sum* ($\min\{1, \mu_{\widetilde{A}^k_{ai}} + \mu_{\widetilde{A}^k_{aj}}\}$) is utilized as T-conorm, where $\mu_{\widetilde{A}^k_a} = \bigwedge \mu_{\widetilde{A}^k_i}$ is the matching degree of all the variables of the antecedent part of the rule. The system is able to deal with missing values by considering that $\mu_{\widetilde{A}^k_i}(e_i) = 1$ if the value of the feature given by the environment e_i is not known. Next, the matching degree of the antecedent part of the rule is determined by the T-norm (conjunction), in this work using the *product* ($\mu_{\widetilde{A}^k_{ai}} \cdot \mu_{\widetilde{A}^k_{aj}}$), of the matching degree of all the input variables. Following that, the membership degree of each of the linguistic terms of the consequent variable ($\mu_{\widetilde{A}^k_c}$) is computed using the T-conorm. Finally, the matching degree of the whole rule is calculated using the *Dienes implication* ($\max\{1 - \mu_{\widetilde{A}^k_a}, \mu_{\widetilde{A}^k_c}\}$) of the antecedent and consequent matching degrees.

Each individual has nine main parameters that evaluate the quality of the rule:

1. the support *sup*, an indicator of the occurring frequency of the rule,

2. the confidence *con*, which denotes the strength of the implication,

3. the lift *lif*, a measure of how many times the antecedent and the consequent appear together more often than expected,

4. the accuracy *acc*, which indicates the matching degree between the rule and the data,
5. the fitness F, which is computed as a power of the lift and accuracy, so reflecting the quality of the rule,
6. the experience *exp*, which counts the number of times that the antecedent part of the rule has matched an input instance,
7. the numberosity *num*, which reckons the number of copies of the individual in the population,
8. the average size of the action sets *as* in which the individual has participated, and
9. the time stamp of the individual $timeOfInd$.

6.3.2 Learning Interaction

Fuzzy-CSar incrementally learns from stream of examples provided by the environment. That is, at each learning iteration Fuzzy-CSar receives an input example from the environment and takes the following actions to update incrementally the individual's parameters and discover new and promising rules. First, the system creates the *match set* [M] with all the individuals in the population that match the input example with a degree greater than 0 in both the antecedent and the consequent parts of the rule. If [M] contains less than θ_{mna} individuals, where θ_{mna} is a configuration parameter, the *covering operator* is triggered to create as many new matching individuals as required to have θ_{mna} individuals in [M]. Then, individuals in [M] are organised into different *association set candidates* $[A]_i$ grouping individuals by the meaning of their antecedent. Each $[A]_i$ is given a probability to be selected that is proportional to the average confidence of the individuals that belong to it. The selected association set [A] goes through a *subsumption* process which aims at reducing the number of rules that express similar associations among variables. Then, the parameters of all the individuals in [M] are updated in an incremental fashion. At the end of the iteration, a niche-based, steady-state GA (Wilson, 1995) is applied to [A] if the average time since its last application is greater than θ_{GA} (another user-defined parameter). It is important to highlight that the GA is not generational (i.e., a single GA iteration is triggered from time to time), so it functions in an online way. This process is repeatedly applied, therefore, updating the parameters of existing individuals and creating new promising rules online. Six elements are needed to be further detailed in order to understand how the system works: (1) the covering operator, the (2) the procedure to create association set candidates, (3) the association set subsumption mechanism, (4) the parameter update procedure, (5) the rule discovery by means of a GA and (6) the replacement mechanism. In the following, each one of these elements is described in more detail.

Covering Operator

Given the sample input example e, the covering operator generates a new individual that matches e with maximum degree. That is, for each variable e_i, the covering operator randomly decides with probability $1 - P_\#$ (where $P_\#$ is a configuration parameter) whether the variable has to be in the antecedent of the rule, with the following two constraints: (1) that, at least, a variable has to be selected and (2) that, at most, $\ell - 1$ variables can be included in the antecedent. Then, one of the remaining variables is selected to be in the rule consequent. Each one of these variables is initialised with the linguistic label that maximises the matching degree with the corresponding input value. In addition, rule generalisation is introduced by permitting the addition of any other linguistic term with probability $P_{\#}$. with the restriction that each variable in the antecedent and consequent contains *maxLingTermsPerVariable* linguistic terms at most. Fuzzy-CSar deals with missing values in e by ignoring the corresponding input variable e_j. Individual's parameters are set to initial values; that is, $sup = 0$, $con = 1$, $lif = 1$, $acc = 1$, $F = 0.01$, $num = 1$, $exp = 0$, as is set to the size of [A] where the covering has been applied, and $timeOfInd$ to the actual time stamp.

Creation of Association Set Candidates

The purpose of generating $[A]_i$ or niches (Goldberg, 1989) is to group rules that express similar associations to establish a competition among them. This strategy lets the best individuals take over their niche hence evolving highly fit individuals. Whilst the creation of these niches of similar rules is quite immediate in reinforcement learning and classification tasks, several approaches could be used to form groups of similar rules in association rule mining. Fuzzy-CSar considers that two rules are similar if they have exactly the same variables in their antecedent, regardless of their corresponding linguistic terms. Therefore, this grouping strategy creates N_a association set candidates, where N_a is the number of rules in [M] with different variables in the antecedent. The underlying principle is that rules with the same antecedent variables may express similar knowledge. To achieve this purpose Fuzzy-CSar sorts [M] according to the number of variables of the contained individuals. Once sorted, the different association set candidates are generated by grouping those sharing the same variables in the antecedent part of the rule. Note that, under this strategy, rules with different variables in the consequent can be grouped in the same association set candidate. After the generation of $[A]_i$, the final set [A] is selected following a roulette-wheel strategy, where each $[A]_i$ has a probability of being selected proportional to its accumulated confidence:

$$p_{sel}^{k} \leftarrow \frac{\sum_{i \in [A]_k} w_i \cdot con_i}{\sum_{j \in [M]} w_j \cdot con_j}, \tag{6.15}$$

where:

$$w_i \leftarrow \begin{cases} 1 & \text{if } exp_i > \theta_{exp}, \\ \frac{1}{10} & \text{otherwise.} \end{cases} \tag{6.16}$$

The computational cost for creating association set candidates are guided by the cost of sorting [M]. A quicksort strategy is applied for this purpose, which has a cost of $\mathrm{O}(n \cdot log\ n)$, where in our case n is the match set size.

Association Set Subsumption

A subsumption mechanism inspired by the one present in XCS (Wilson, 1998) was designed with the aim of reducing the number of different rules that express the same knowledge. The process works as follows: each rule in [A] is checked for subsumption with each other rule of the set. A rule r_i is a candidate subsumer of r_j if it satisfies two conditions:

1. r_i has a similar confidence than r_j and it is experienced enough (that is, con^i is, at least, a ninety percent of con^j and $exp^i > \theta_{exp}$, where θ_{exp} is a user-set parameter), and

2. r_i is more general than r_j: A rule r_i is more general than r_j if all the input and the output variables of r_i are also defined in r_j and r_i has, at least, the same linguistic terms than r_j for each one if its variables. Each time a rule r_i subsumes a rule r_j, r_i increases its numerosity count and the subsumed one r_j is deleted from population.

Parameter Update

At the end of each learning iteration, the parameters of all individuals that belong to [M] are updated. First, the experience of each individual is incremented. second, the support of each rule is updated as:

$$sup_{t+1} \leftarrow sup_t + \frac{\mu_{\widetilde{A}}(e) \cdot \mu_{\widetilde{C}}(e) - sup_t}{exp}, \tag{6.17}$$

where $\mu_{\widetilde{A}}(e)$ and $\mu_{\widetilde{C}}(e)$ are the matching degree of the antecedent and the consequent respectively. Then, the confidence is computed as:

$$con_{t+1} \leftarrow \frac{imp_{t+1}}{ant\_mat_{t+1}}, \tag{6.18}$$

where:

$$imp_{t+1} \leftarrow imp_t + \mu_{\widetilde{A}}(e) \cdot \max\{1 - \mu_{\widetilde{A}}(e), \mu_{\widetilde{C}}(e)\}, \tag{6.19}$$

and

$$ant\_mat_{t+1} \leftarrow ant\_mat_t + \mu_{\widetilde{A}}(e). \tag{6.20}$$

Initially, both imp_t and ant_mat_t are set to 0. Next, the lift is calculated as:

$$lift_{t+1} \leftarrow \frac{sup_{t+1}}{ant\_mat_{t+1} \cdot con\_mat_{t+1}}, \tag{6.21}$$

where:

$$con\_mat_{t+1} \leftarrow con\_mat_t + \mu_{\widetilde{C}}(e). \tag{6.22}$$

Similarly to ant_mat_t, con_mat_t is initially set to 0. Next, the accuracy is updated as:

$$acc_{t+1} \leftarrow sup_{t+1} + \Bigg(1 - (ant\_mat_{t+1} + con\_mat_{t+1} - sup_{t+1})\Bigg). \tag{6.23}$$

Thereafter, the fitness of each rule in [M] is calculated as:

$$F \leftarrow \left(\frac{sup_{t+1} \cdot lift_{t+1} + acc_{t+1}}{2}\right)^{\nu}, \tag{6.24}$$

where ν is a user-set parameter that permits to control the pressure towards highly fit individuals. It is worth noting that the fitness function can result in individuals with fitness values greater than one, so if it happens, the fitness value is truncated to one. The fitness computation has been designed using the recommendations found in (Martínez-Ballesteros et al., 2011a). Finally, the association set size—notice that this size is computed after the subsumption mechanism—estimate of all rules that belong to the selected [A] is computed as the average size of all the [A]'s in which it has participated.

Discovery Component

Fuzzy-CSar uses a steady-state, niche-based, incremental GA to discover new and promising rules. The GA is applied to the selected [A]. Therefore, the niching is intrinsically provided since the GA is applied to rules that are similar according to one of the heuristics for association set formation. The GA is triggered when the average time from its last application upon the individuals in [A] exceeds the threshold θ_{GA}, a user-defined parameter. It selects two parents p_1 and p_2 from [A] using tournament selection (Butz et al., 2005). The two parents are copied into offspring ch_1 and ch_2, which undergo crossover and mutation. Fuzzy-CSar uses a uniform crossover operator which crosses the antecedents of the rules by two points, permitting cross-points within variables. It is applied with probability P_χ. If crossover is not applied, the children are an exact copy of the parents. The resulting offspring may go through three different types of mutation: (1) mutation of the antecedent variables (with probability $P_{I/R}$), which randomly chooses whether a new antecedent variable has to be added to or one of the antecedent variables has to be removed from the rule; (2) mutation of the consequent variable (with probability P_C), which selects one of the variables of the antecedent and exchanges it with the variable of the consequent; and (3) mutation of the linguistic terms of the variable (with probability P_μ), which selects one of the existing variables of the rule and mutates its value in one of the three possible ways: *expansion*, *contraction* or *shift*. Expansion chooses a linguistic term not represented in the corresponding variable and adds it to this variable; thus, it can be applied only to variables that do not have all the linguistic terms. Contraction selects a linguistic term represented in the variable and removes it; so, it can be applied only to variables that have more than one linguistic term. Shift changes a linguistic term for its immediate inferior or superior. The parameters of the offspring are initialised as follows: if the crossover is not applied, the parameters are copied from the selected parent. Otherwise, these are set to the average value between the corresponding parameters in the parents. In both cases, the fitness is decreased to 10% of the parental fitness. Experience is set to 0 and numerosity to 1.

Replacement Mechanism

The new offspring are introduced into the population, but first each individual is checked for subsumption with their parents. To decide if any parent can subsume the offspring, the same procedure for association set subsumption is followed. If any parent is identified as a possible subsumer for the offspring, the offspring is not inserted into the population and the numerosity of the parent is increased by one. Otherwise, [A] is checked for the most general rule that can subsume the offspring and if no subsumer can be found, the individual is finally inserted into [P].

If the population is full, exceeding individuals are deleted from [P] with probability directly proportional to their association set estimate and inversely proportional to its fitness.

Moreover, if a individual k is sufficiently experienced ($exp^k > \theta_{del}$, where θ_{del} is a user-defined parameter), and its fitness F^k is significantly lower than the average fitness of the individuals in [P] (that is, $F^k < \delta F_{[P]}$, where δ is a configuration parameter, typically set to 0.1, and $F_{[P]} = \frac{1}{N}\sum_{i \in [P]} F^i$), its deletion probability is further increased. That is, each individual has a deletion probability p_k of

$$p^k \leftarrow \frac{d^k}{\sum_{\forall j \in [P]} d^j}, \tag{6.25}$$

where

$$d^k \leftarrow \begin{cases} \frac{as \cdot num \cdot F_{[P]}}{F^k} & \text{if } exp^k > \theta_{del} \text{ and } F^k < \delta F_{[P]}, \\ as \cdot num & \text{otherwise.} \end{cases} \tag{6.26}$$

Thus, the deletion algorithm balances the individual's allocation in the different [A]'s by pushing towards the deletion of rules belonging to large association sets. At the same time, it favors the search towards highly fit individuals, since the deletion probability of rules whose fitness is much smaller than the average fitness is increased.

6.3.3 Cost of the Algorithm

Similarly to other Michigan-style LCS (Orriols-Puig et al., 2009b), the cost of the algorithm increases linearly with the maximum population size N, the maximum number of variables Max_{var} used per rule, and semi-logarithmically with the cost of sorting the match set, as depicted in Equation 6.27:

$$Cost_{Fuzzy-CSar} = O(N \cdot Max_{var} \cdot |[M]| \cdot \log |[M]|), \tag{6.27}$$

where $|[M]|$ is the size of the match set. It is important to highlight that Fuzzy-CSar does not depend directly on the number of transactions, which makes it very competent for mining huge databases. Online learners can stall the learning stage whenever required and the evolved rule set can be used for modeling the environment. In this regard, the more learning iterations Fuzzy-CSar has performed, the more general and accurate rules (in terms of support, confidence, lift and accuracy) should be.

6.3.4 Insights on Why Fuzzy-CSar Works

Michigan-style LCSs are open frameworks that foster crossbreeding between different learning paradigms. As discussed in chapter 3, Butz (2006) identified a set of pressures that guide Michigan-style LCSs to obtain accurate results and that explain why Michigan-style LCSs work. These are the following: the fitness pressure which pushes [P] towards more accurate individuals, the set pressure and the subsumption pressure which pushes [P] towards generalisation, the mutation pressure which pushes towards more specific solutions and the deletion pressure which pushes [P] towards fittest individuals. Notice that this deletion scheme erases individuals that are no longer useful due to a concept drift. Despite the fact that these studies are referred to XCS, the most studied LCS and a particular implementation of Michigan-style LCS, these can be extrapolated to other systems that follow the same framework.

Knowing the intrinsics of Fuzzy-CSar, in the subsequent sections, the system is analysed in a set of different environments: (1) association streams with different complexities and (2) offline real-world problems.

6.4 Experiments on Association Streams

Our hypothesis is that Fuzzy-CSar is able to accurately model what the stream of information hides (in terms of rule support, confidence, lift and accuracy) and that is able to adapt to the distinct drifting concepts due to its adaptive architecture. For that purpose we designed a series of experiments with distinct complexities that model extreme cases of real-world scenarios. These are detailed in the following.

6.4.1 On the Difficulty of Evaluating Association Streams

We want to test our algorithm under problems with similar complexities as those that would appear in real-world association stream environments. However, unlike in supervised and clustering stream fields (see SEA-related problems (Fan, 2004, Orriols-Puig and Casillas, 2010b, Street and Kim, 2001) and MOA for instance), under association streams there is no formal way to quantitatively evaluate what happens with the learned model when a concept drift occurs. That is mainly due to the fact that in association streams we do not rely on a class label or cluster to perform evaluations (i.e., Kappa statistic and F-measure, just to mention two in the supervised case and the validation indexes in the case of clustering). Typically, in the closely related online frequent pattern mining field concept drifts are either avoided by the use of statistics over the incoming data thus eliminating the old model once it has occurred and re-learning a new one or not handled under the assumption that the stream will not have any drift. In this regard, the difficulty is how to evaluate the performance of the algorithm.

To test Fuzzy-CSar under association streams we have designed an environment that allows us to generate online crisp data streams from a series of predefined rules. The data is then generated from uniformly distributed and triangular-shaped membership functions. Because the cut-off points are known from the predefined rules, variables are set using a pseudo-random generator forcing to have a membership degree greater than 0.5 according to the respective fuzzy label in the predefined rule. The idea behind is that the algorithm should discover the rules out of the streams of data and adapt to drifting concepts. Our proposal uses a hold out strategy generating two distinct sets (train and test sets) of examples out of the predefined rules in an online fashion. The procedure to evaluate the system is the following: after a train stage the system enters in test mode. During the test stage the environment send previously unseen examples. We seek for the best matching rule out of [M], that is, the rule that matches both antecedent and consequent with *maximum matching degree*. It there is a tie, we select the most general rule. Finally, we perform a comparison by means of the Hamming distance with the rule selected by Fuzzy-CSar and the predefined rule that has generated the test example: for every one of the variables used in the predefined rule, if the variable is found in the individual's rule, the Hamming distance is computed between the linguistic terms of the predefined rule and the linguistic terms of the individuals's rule. Error is set to 1 for that variable otherwise. If there is no example, the error is set to 1 for that instance. Therefore, for the quantitative evaluation we use the mean accumulated Hamming distance over the test stage.

6.4.2 Methodology Of Experimentation

As any Michigan-style LCSs, Fuzzy-CSar has several configuration parameters which enable it to adjust the behaviour of the system to evolve models of maximal quality. The configuration parameters have been obtained experimentally following the recommendations found in (Orriols-Puig, 2008). This study detected that Michigan-style LCSs are sensitive to the the generalisation in initialisation $P_{\#}$ and the fitness pressure ν and the setting of the other parameters had little effect on the final behaviour. With this information in mind we selected values that, on average, allow Fuzzy-CSar to perform well on all the problems. In this regard Fuzzy-CSar was configured with the following parameters for all the experiments: $\nu = 1$, $P_{\#} = 0.2$, $P_{\chi} = 0.8$, $\{P_{I/R}, P_{\mu}, P_C\} = 0.1$, $\delta = 0.1$, $\theta_{GA} = 12$, $\theta_{exp} = 24$, $\{\theta_{del}, \theta_{sub}\} = 20$, $maxLingTermsPerVariable = 3$, θ_{mna} automatically sets to the number of variables, and the population was set to 1 000 individuals. In addition, all the variables use Ruspini's strong fuzzy semantics with five linguistic terms (XS, S, M, L and XL). Results are averages of 30 runs, using each time a different random seed.

6.4.3 Experiment 1

Description

In the first experiment we want to check the reaction capacity of Fuzzy-CSar under distinct rule drifts. This environment is inspired on the Hulten et al. rotating hyperplane problem (Hulten et al., 2001) and contains abrupt concept drifts and noisy inputs. The problem is described by a multi-dimensional space defined by a set of eight rules which make use of three continuous variables $\{x_1, x_2, x_3\}$ ranging in $[0, 1]$ from which only two are relevant at a given concept. The three variables swap positions in each concept in the following way: during the first concept the variable x_1 is in the antecedent part, x_2 is in the consequent part, and is left x_3 taking random values (i.e., being noise). In the second concept, however, a swap in the antecedent/consequent variable position happens and the variable x_2 now appears in the antecedent part of the rule and x_1 in the consequent. In the third concept another change happens and x_1 returns to the antecedent part of the rule. The variable x_3 is in the consequent during the third concept. This time, the variable x_2 takes random values. The fourth and final drift happens in the same ways as in the simple rule determination problem, that is: the variable x_1 is in the antecedent part of the rule, x_2 is in the consequent part, and x_3 takes random values. Table 6.1 shows the new rule base for problem.

The association stream consists of 50 000 examples and the concept is changed every 12 500. Thus, the system has four different concepts and 12 500 randomly generated training instances per concept. The pseudo-random generator uses a uniform distribution. To test the models, independent sets consisting of 2 500 test examples are generated per concept. The learner is trained for 500 data sambples and then its model is evaluated by the corresponding test set.

Three key aspects are interesting to analysed in the results: (1) the time the algorithm requires to learn at the first stages of the problem, (2) the reaction of the algorithm to sudden concept drifts, and (3) the number of rules and its *quality* at the end of each concept.

Concept 1			x_1					$\mathbf{x_2}$		
	VS	S	M	L	VL	VS	S	M	L	VL
$\mathbf{R_1}$			X	X	X			**X**	**X**	**X**
$\mathbf{R_2}$	X	X				**X**	**X**			
Concept 2			x_2					$\mathbf{x_1}$		
	VS	S	M	L	VL	VS	S	M	L	VL
$\mathbf{R_1}$	X	X	X			**X**	**X**	**X**		
$\mathbf{R_2}$				X	X				**X**	**X**
Concept 3			x_3					$\mathbf{x_1}$		
	VS	S	M	L	VL	VS	S	M	L	VL
$\mathbf{R_1}$	X	X				**X**	**X**			
$\mathbf{R_2}$				X	X				**X**	**X**
Concept 4			x_1					$\mathbf{x_2}$		
	VS	S	M	L	VL	VS	S	M	L	VL
$\mathbf{R_1}$			X	X	X	**X**	**X**	**X**		
$\mathbf{R_2}$	X	X							**X**	**X**

Table 6.1: Predefined rule base used to generate the first experiment stream. The consequent variable is accentuated in bold. Notice that, from concept to concept, one variable is left taking random values.

Results

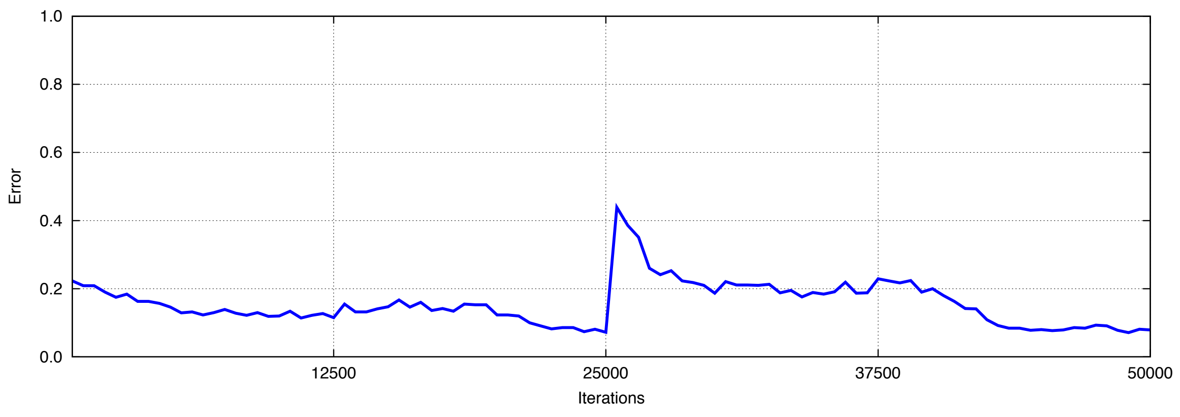


Figure 6.4: Results obtained in the first experiment. Every 12 500 data samples there is a concept drift. Curves are averages of 30 runs.

In the following we elaborate the interesting aspects of the experiment results.

- **Time required to learn at the first stages of the problem.** Fuzzy-CSar starts near a 20% error, a relatively low value. Moreover, the learning curve is step and at the end of the first concept it reaches near a 15% error.

- **Reaction against sudden concept drifts.** As shown in Figure 6.4, the first concept drift causes no trouble to the proposed algorithm. That is partly due to the heuristics of Fuzzy-CSar when generating the covering individuals and partly due to the genetic discovery. The third concept is also very interesting because x_3, the noisy variable until this time, is suddenly swapped with x_2. Also, x_1 returns to the antecedent of the rule.

Is precisely in this concept where the differences with the previous problem arise: at the beginning it rockets the error curve above the 40%. Fuzzy-CSar has a fast reaction capacity and it a few data samples it lowers the error curve to near the 20%. Again, the two main sources of learning make it adapt quickly. Finally, in the fourth concept, the variable x_2 is swapped with x_3 again. In this last part of the problem, Fuzzy-CSar returns to a very low error rate, ending near the 10%.

- **Number of rules and its *quality* at the end of each concept.** Interestingly the number of rules discovered are far less than the population limit of 1 000 individuals for this problem, as depicted in Table 6.2. It is worth mentioning the high values of average confidence and lift obtained by the search engine of Fuzzy-CSar.

Concept	**Rules**	**Sup**	**Con**	**Lif**	**Acc**
1	$64.03\pm_{6.86}$	$0.35\pm_{0.01}$	$0.72\pm_{0.02}$	$1.17\pm_{0.07}$	$0.59\pm_{0.02}$
2	$64.70\pm_{4.21}$	$0.35\pm_{0.01}$	$0.71\pm_{0.02}$	$1.13\pm_{0.06}$	$0.57\pm_{0.02}$
3	$64.20\pm_{5.27}$	$0.31\pm_{0.01}$	$0.65\pm_{0.03}$	$1.04\pm_{0.09}$	$0.52\pm_{0.02}$
4	$64.13\pm_{5.67}$	$0.33\pm_{0.01}$	$0.69\pm_{0.02}$	$1.09\pm_{0.06}$	$0.55\pm_{0.02}$

Table 6.2: Number of rules and its average quality in terms of support, confidence, lift and accuracy at the end of each concept.

6.4.4 Experiment 2

Description

Differently from the previous problem, this environment generates variables in which their generality grow or shrink per concept, that is, an increase or a reduction in the range of all variables, respectively, and also each concept has a different number of rules and length. It uses of three continuous variables $\{x_1, x_2, x_3\}$ ranging in $[0, 1]$ from which only the first two variables are relevant. Table 6.3 show the rule base used to generate the data of this problem. The data stream consists of 31 000 data samples and the concept is changed in

Concept 1	x_1					$\mathbf{x}_2$				
	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$		X	X					**X**	**X**	
Concept 2	x_1					$\mathbf{x}_2$				
	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$		X	X	X			**X**	**X**	**X**	
Concept 3	x_1					$\mathbf{x}_2$				
	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$	X				X	**X**				**X**

Table 6.3: Rule base used to generate the second problem. The consequent variable is accentuated in bold. Notice that the variable x_3 is left taking random values.

the following way: the first concept lasts for 1 000 examples, the second concept lasts for 20 000 data samples, and the third concept lasts for 10 000 examples. As before, the training instances are randomly generated following the corresponding rule base. To test the models, independent sets consisting of 2 500 test examples are generated per concept using a uniform distribution. The learner is trained for 500 data samples and then its model is evaluated by the corresponding test set.

Results

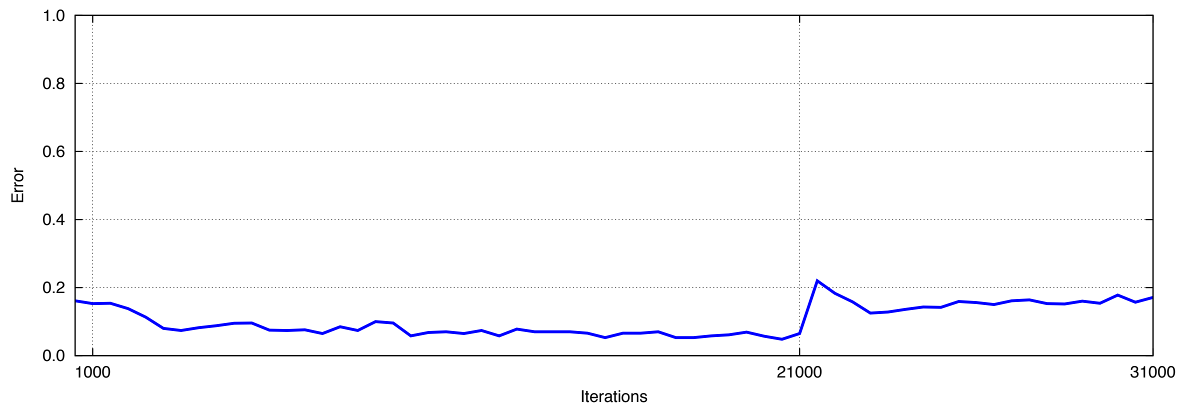


Figure 6.5: Results of Fuzzy-CSar on the second problem. Concept drifts happen at iteration 1 000 and 21 000. The resulting curve is the average of 30 runs.

In the following we provide the analysis of the experiment results.

- **Time required to learn at the first stages of the problem.** As depicted in Figure 6.5, Fuzzy-CSar starts with a moderate error rate, near the 17% during the first concept. It is important to highlight that the concept lasts just 1 000 data samples.
- **Reaction against sudden concept drifts.** Abruptly, the concept changes and Fuzzy-CSar is fostered to learn a set of different rules. Throughout the entire concept (it lasts for 20 000 data samples), the algorithm recovers quickly, ending the error curve below the 10%. Finally, the third—and last concept drift—happens at the iteration 21 000. In this case, the error peak is clearly visible, being slightly higher than the 20%. Fuzzy-CSar recovers from that error peak but we see that it ends near the 17%, a moderate error rate. This change in concept seems to be difficult for the algorithm.
- **Number of rules and its *quality* at the end of each concept.** Again, the number of rules discovered are far less than the population limit of 1 000 individuals for this problem. Table 6.4 shows the quality measures of the rules obtained by Fuzzy-CSar at the end of the run.

6.4.5 Experiment 3

Description

This problem has the particularity that rules with same antecedent have different consequents. This issue makes the problem difficult for any association stream algorithm. Also, the number

Concept	Rules	Sup	Con	Lif	Acc
1	35.97±7.79	0.52±0.07	0.76±0.05	1.0±0.01	0.59±0.04
2	39.07±3.85	0.58±0.03	0.78±0.03	1.0±0.01	0.62±0.02
3	41.33±8.04	0.56±0.04	0.78±0.03	1.0±0.01	0.61±0.02

Table 6.4: Number of rules and its average quality in terms of support, confidence, lift and accuracy at the end of each concept.

of variables is increased in the last concept. The problem uses of four continuous variables $\{x_1, x_2, x_3, x_4\}$ ranging in $[0, 1]$. During the first concept, two distinct rules are generated. While these two rules have distinct consequent variables (x_2 and x_3, respectively), both share the variable x_1, which is always in the antecedent part of the rule. Then an abrupt concept drift occurs by changing completely the fuzzy labels of each variable. It is worth mentioning that in the two first concepts the variable x_4 take random values. Finally, in the last concept, another major concept drift happens adding the variable x_4, jointly with x_1 in the antecedent part of the rules. Table 6.5 depicts the rule base used to generate the data of this problem. The data stream consists of 37 500 data samples and the concept is changed every 12 500

Concept 1	x_1					$\mathbf{x_2}$					$\mathbf{x_3}$									
	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL					
R$_1$			X	X	X	**X**	**X**	**X**												
R$_2$			X	X	X								**X**	**X**	**X**					
Concept 2	x_1					$\mathbf{x_2}$					$\mathbf{x_3}$									
	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL					
R$_1$	X	X			X			**X**	**X**	**X**										
R$_2$	X	X			X						**X**	**X**	**X**							
Concept 3	x_1					x_4					$\mathbf{x_2}$					$\mathbf{x_3}$				
	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$	X	X						X						**X**	**X**					
R$_2$	X	X						X								**X**	**X**			

Table 6.5: Predefined rule base used to generate the third experiment stream.

examples. The training instances are randomly generated following the corresponding rule base and using an uniform distribution. To test the models, independent sets consisting of 2 500 test examples are generated per concept. The learner is trained for 500 data samples and then its model is evaluated by the corresponding test set.

Results

The time Fuzzy-CSar requires to learn at the first stages of the problem, its the reaction to concept drifts, and the number of rules and its quality at the end of each concept are analysed in the following.

- **Time required to learn at the first stages of the problem.** The algorithm starts near a 25% error, a low error value, and it maintains close to this value during the first concept.

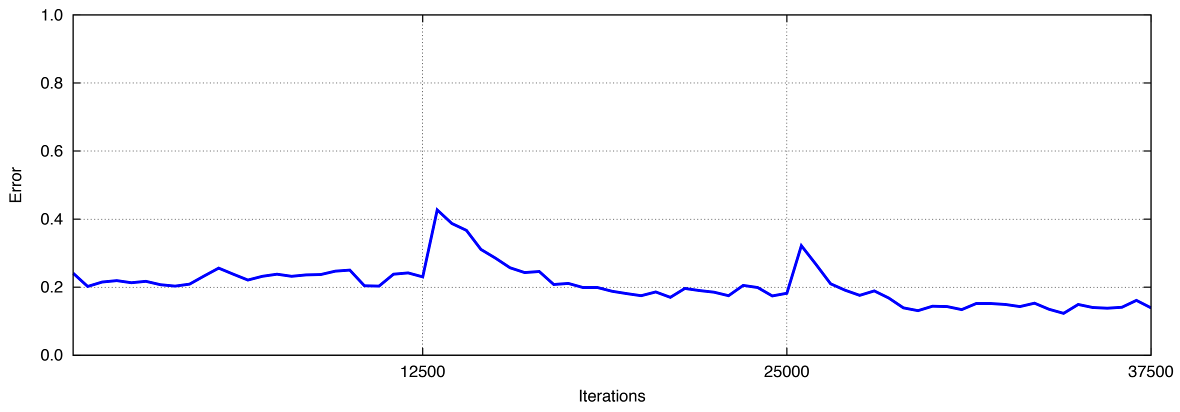


Figure 6.6: Results obtained in the third experiment. Every 12 500 data samples there is a concept drift. Curves are averages of 30 runs.

- **Reaction against concept drifts.** When the first concept drift occurs the error rockets beyond the 40%. Fuzzy-CSar has a fast reaction capacity and a few data samples it lowers the error curve below the 20%. Finally, in the last concept, a major concept drift occurs and Fuzzy-CSar first increases its error above the 35% for, in a few thousand data samples later, end near the 15%, a low error value, as depicted in Figure 6.6.

- **Number of rules and its *quality* at the end of each concept.** As expected the number of rules at the end of each concept is higher than in the previous one due to its difficulty, shown on Table 6.6. Notice the values of the quality measures in the second concept: these are the effects of the mashup of rules. Interestingly, in the subsequent concept they raise, showing the high adaptivity of Fuzzy-CSar.

Concept	Rules	Sup	Con	Lif	Acc
1	101.90$\pm$0.45	0.41$\pm$0.04	0.75$\pm$0.03	1.04$\pm$0.01	0.54$\pm$0.02
2	100.83$\pm$10.38	0.15$\pm$0.02	0.49$\pm$0.11	0.70$\pm$0.22	0.46$\pm$0.02
3	104.63$\pm$7.30	0.42$\pm$0.03	0.74$\pm$0.03	1.01$\pm$0.01	0.55$\pm$0.01

Table 6.6: Number of rules and its average quality in terms of support, confidence, lift and accuracy at the end of each concept.

6.4.6 Experiment 4

Description

This problem extends some of the ideas from the previous problems but in a higher dimensional search space and in which each concept has a different number of rules and length. This problem makes use of five continuous variables $\{x_1, x_2, x_3, x_4, x_5\}$ ranging in $[0, 1]$ from which, depending on the concept, some of these variables are relevant. In the first concept, the rule to be studied uses the first four variables. In the second concept the following variables $\{x_1, x_2, x_4\}$ are the relevant ones. Finally, in the third concept, the following variables

$\{x_1, x_2, x_3, x_4\}$ are the relevant ones, but in distinct rules. Table 6.7 shows the rule base used to generate the problem. In addition, the pseudo-random generator uses the normal distribution N(0, 1) instead of the uniform one. The data stream consists of 50 000 data samples and

Concept 1	x_1					x_2					x_3					$\mathbf{x_4}$				
	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$	X									X	X					**X**				
R$_2$		X							X			X					**X**			
R$_3$			X					X					X					**X**		
R$_4$				X					X			X							**X**	
R$_5$					X					X	X									**X**
Concept 2						x_2					x_3					$\mathbf{x_4}$				
						VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$						X									X					**X**
R$_2$							X							X				**X**		
R$_3$								X					X			**X**				
Concept 3	x_1					x_2					x_3					$\mathbf{x_4}$				
	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL	VS	S	M	L	VL
R$_1$	X															**X**				
R$_2$								X										**X**		
R$_3$															X					**X**

Table 6.7: Rule base used to generate the fourth problem. The consequent variable is accentuated in bold. Notice that, from concept to concept, some variables are left taking random values.

the concept is changed in the following way: the first concept lasts for 20 000 data samples, the second lasts for 15 000 examples, and the third concept lasts for 15 000 instances. The training instances are randomly generated following the corresponding rule base. To test the models, independent sets consisting of 2 500 test examples are generated per concept. The learner is trained for 500 data samples and then its model is evaluated by the corresponding test set.

Results

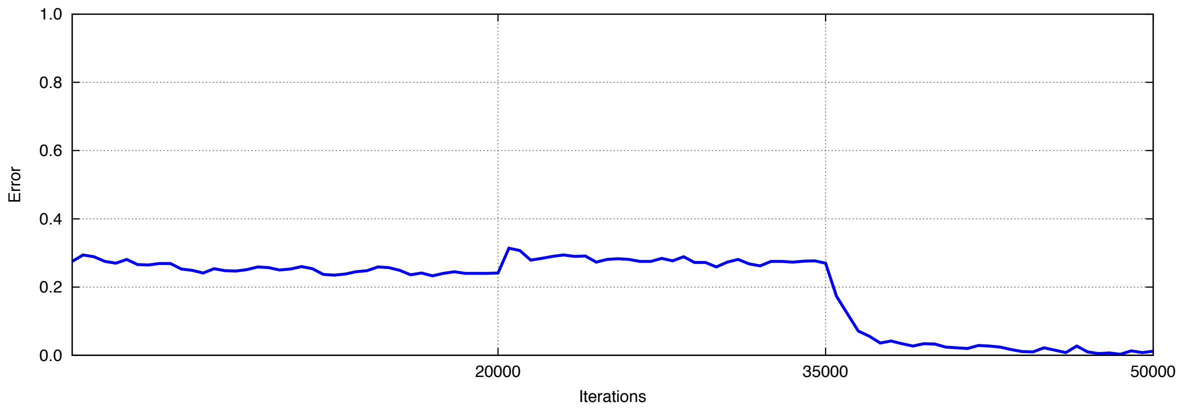


Figure 6.7: Results obtained in the fourth experiment. Concept drift happens at iteration 20 000 and at 35 000. Curves are averages of 30 runs.

In the following the key aspects of the results obtained by Fuzzy-CSar are analysed.

- **Time required to learn at the first stages of the problem.** As Figure 6.7 depicts, the algorithm starts just below the 30% and quickly lowers that value to close the 25% and remains near this value.
- **Reaction against concept drifts.** The first concept drift happens and Fuzzy-CSar shows a very small error peak. This change is soft enough for Fuzzy-CSar to adapt to it. During this stage the algorithm improves its error. However, the noise added by x_3 and x_5 seems to affect Fuzzy-CSar negatively. Finally, in the third concept, Fuzzy-CSar is able to recover to near the 1% error in a step curve. Notice the flat appearance of the error curve in the second concept. Despite Fuzzy-CSar seems to have trouble at finding the exact rules, its generalisation capacity is actuating and the number of individuals used is reduced during the entire concept.
- **Number of rules and its *quality* at the end of each concept.** Table 6.8 shows the number and quality of the rules obtained by Fuzzy-CSar. It is noticeable the low value of average support and the large variation in lift.

Concept	**Rules**	**Sup**	**Con**	**Lif**	**Acc**
1	112.73$\pm$6.63	0.08$\pm$0.03	0.66$\pm$0.02	1.93$\pm$1.15	0.71$\pm$0.01
2	107.17$\pm$7.21	0.02$\pm$0.01	0.47$\pm$0.12	1.36$\pm$3.01	0.82$\pm$0.01
3	137.43$\pm$8.41	0.02$\pm$0.01	0.48$\pm$0.11	1.27$\pm$2.75	0.79$\pm$0.02

Table 6.8: Number of rules and its average quality in terms of support, confidence, lift and accuracy at the end of each concept.

6.4.7 Discussion

Fuzzy-CSar has shown a remarkable response in a series of online environments that model extreme situations that we could find in real-world situations where a quick response is critical (e.g., Smart Grids monitoring). These experiments allowed us to obtain quantitative results on how well the algorithm perform under association streams. As we hypothesised, the online architecture is able to accurately model the hidden information out of the streams of data. Moreover, the experiments done support that the Michigan-style LCS framework in which our proposal relies is able to quickly adapt to drifting concepts.

In the following section, Fuzzy-CSar is carefully analyzed under a real data stream environment with unknown dynamics

6.5 Experiment on a Real Data Stream Problem

First described by Harries et al. (Harries et al., 1998), the New South Wales electricity market problem has been used broadly in data streams literature (Gama et al., 2004, Núñez et al., 2007) as a real problem with unknown dynamics. The problem consists in the modelling of the electric market of a particular area of Australia to extract key knowledge that enable experts to create a compelling online environment for these market. The data set contains 45 312 examples, and each one of these instances refers to a period of 30 minutes, and they have eight fields: the time stamp, the day of the week, the period, the New South Wales electricity price, the New South Wales electricity demand, the Victoria electricity price, the Victoria electricity demand and the scheduled electricity transfer between states. The class has been removed from the data set. The procedure used in this experiment is the following: Fuzzy-CSar is trained for 336 examples (that is: every *week*), and then the quality of the output rules is analysed. The hypothesis is that Fuzzy-CSar will be able to find high quality rules in terms of support and confidence and, from week to week, the algorithm will conserve much of its previously inferred rules, adapting itself to the unknown dynamics of the problem. To do so, after a *week* (336 examples), the output rules obtained by Fuzzy-CSar are compared with the ones obtained in the previous week using the Hamming distance between the rules. For this experiment, rules with a distance 0 (meaning that they are conserved with respect to the new ones: same variables in the antecedent and consequent parts of the rule and same linguistic terms per variable) are contemplated.

6.5.1 Methodology of Experimentation

In this experiment, Fuzzy-CSar was configured with a population size of 400 rules and the following parameters: $\nu = 1$, $P_{\#} = 0.2$, $P_{\chi} = 0.8$, $\{P_{I/R}, P_{\mu}, P_C\} = 0.1$, $\delta = 0.1$, $\theta_{GA} = 25$, $\theta_{exp} = 50$, $\{\theta_{del}, \theta_{sub}\} = 20$, $maxLingTermsPerVariable = 3$, and $\theta_{mna} = 2$. All the variables use Ruspini's strong fuzzy semantics with five linguistic terms (XS, S, M, L and XL). The experiment has been repeated for 30 runs, each with a different random seed. Results are the averages of these runs.

6.5.2 Analysis of the Results

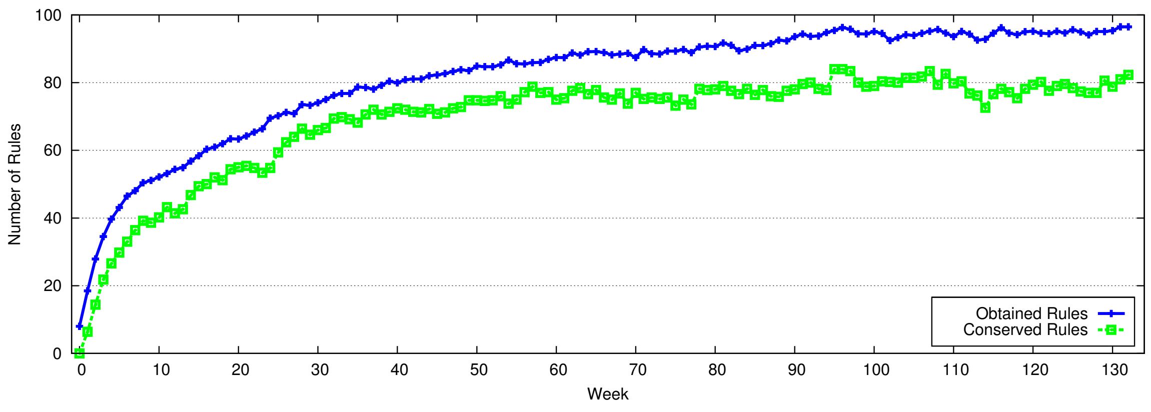


Figure 6.8: Number of rules obtained by Fuzzy-CSar on the New South Wales Electricity market problem.

Fuzzy-CSar has been tested under a real data stream with unknown dynamics with the aim of searching for interesting knowledge in a real Electricity market. Figure 6.8 depicts the number of rules obtained by the presented algorithm. It is clearly visible that, as expected, a large part of the rules are conserved between week periods.

To show the knowledge provided by Fuzzy-CSar, an interesting couple of the discovered rules are depicted in what follows. The first rule describes the feature *transfer*, which refers to the scheduled electricity transfer between Victoria and New South Wales states.

R$_1$: **IF** *timeStamp* **is** {XS or S or XL} **and** *NSWPrice* **is** {XS or S or M} **and** *NSWDemand* **is** {S or M or L} **THEN** *transfer* **is** {S or M or L} [sup: 0.61; con: 0.99; lif: 1.0; acc: 0.61].

R$_1$ shows that the degree of presence of the price and demand in New South Wales (features *NSWPrice* and *NSWDemand*, respectively) are key factors for the electricity transfer between the two states, with a confidence of 0.99. The second rule describes the feature *NSWPrice*, which refers to the electricity price in New South Wales.

R$_2$: **IF** *NSWDemand* **is** {S or M or L} **and** *VICPrice* **is** XS **and** *transfer* **is** {XS or S or M} **THEN** *NSWPrice* **is** {XS or S or M} [sup: 0.78; con: 1.0; lif: 1.0; acc: 0.78].

R$_2$ indicates that the price of electricity in New South Wales not only depends on its demand, but on the price of electricity in Victoria and in the degree of transfer between these two states, with a confidence of 1.0. The results obtained by Fuzzy-CSar are very competitive in terms of rule quality (support, confidence, lift and accuracy). This experiment supports the fact that Fuzzy-CSar can be applied to real environments to obtain information that is potentially useful to experts in the field.

6.6 Experiments on Real-World Data Sets with Static Concepts

Fuzzy-CSar has demonstrated a competitive behaviour under synthetic online environments, showing adaptivity to the challenges of learning under association streams and in a real-world data stream problem. In this section, Fuzzy-CSar's behaviour is analysed under a variate set of real-world problems with static concepts to test its robustness. In what follows, the study of the behaviour of Fuzzy-CSar is extended using a large collection of real-world problems that do not contain any kind of concept drift (i.e., concepts do not change over time). The goal of the analysis is twofold: (1) demonstrate the computational complexity and scalability of Fuzzy-CSar, and (2) show the quality of the models created by Fuzzy-CSar, comparing the results with the ones of Fuzzy-Apriori, the most well-known fuzzy association rule mining algorithm. These problems have different characteristics and they have been taken from the KEEL public repository (Alcalá-Fdez et al., 2011), except for the *fam* data set which was obtained from the UCLA Statistics Data Sets Archive website[2] and μca, which is from a local repository.

[2]http://www.stat.ucla.edu/data/fpp

Id.	**#Inst**	**#Fe**	**#Re**	**#In**
app	106	7	7	0
bpa	345	6	1	5
fam	63 756	10	0	10
gls	214	9	9	0
h-s	270	13	1	12
irs	150	4	4	0
mag	19 020	10	10	0
μca	216	21	21	0
pho	5 404	5	4	1
pim	768	8	8	0
rng	7 400	20	20	0
seg	2 310	19	19	0
son	208	60	60	0
spa	4 597	57	57	0
thy	215	5	4	1
veh	846	18	0	18
wav	5 000	40	40	0
wdbc	569	30	30	0
wne	178	13	13	0
wpbc	198	33	33	0
yst	1 484	8	8	0

Table 6.9: Properties of the data sets considered for the experimental study. Columns describe: the identifier (Id.), the number of instances (#Inst), the number of features (#Fe), the number of continuous features (#Re) and the number of integer features (#In).

Table 6.9 shows the properties of the real-world data sets considered for the experimental study. The aforementioned data sets were intended for classification tasks and hence these have been adapted by removing the class feature. All the experiments were run using a cluster of Core-i5, 2.8 GHz CPUs with 4 Gb of memory machines and running Linux. Both Fuzzy-CSar and Fuzzy-Apriori were programmed in C++. Fuzzy-CSar was configured with a population size of 6 400 individuals and was run during 100 000 data samples. The remaining parameters were configured as follows: $\nu = 1$, $P_{\#} = 0.5$, $P_{\chi} = 0.8$, $\{P_{I/R}, P_{\mu}, P_C\} = 0.1$, $\delta = 0.1$, $\theta_{GA} = 50$, $\theta_{exp} = 1000$, $\{\theta_{del}, \theta_{sub}\} = 20$, $maxLingTermsPerVariable = 3$, and $\theta_{mna} = 2$. Fuzzy-Apriori was configured using standard values (Hong et al., 2001): $\{minSupport, minConfidence\} = 0.05$. All the variables use Ruspini's strong fuzzy semantics with five linguistic terms (XS, S, M, L and XL). The experiments have been repeated for 30 runs, each with a different random seed. Results are the averages of these runs.

6.6.1 Analysis of the Computational Complexity and Scalability

To check the computational complexity and scalability of Fuzzy-CSar, a series of experiments have been performed using the *wav* data set. This data set consists of 5 000 examples and has 40 features, which makes it very appropriate for scalability tests due to its size. Two different experiments have been performed to analyse the complexity and scalability of Fuzzy-CSar: (1) analysis of the relationship between the runtime and the number of variables when using all the 5 000 transactions of the data set, and (2) analysis of the relationship between the runtime and the number of transactions using all the 40 features of the data set.

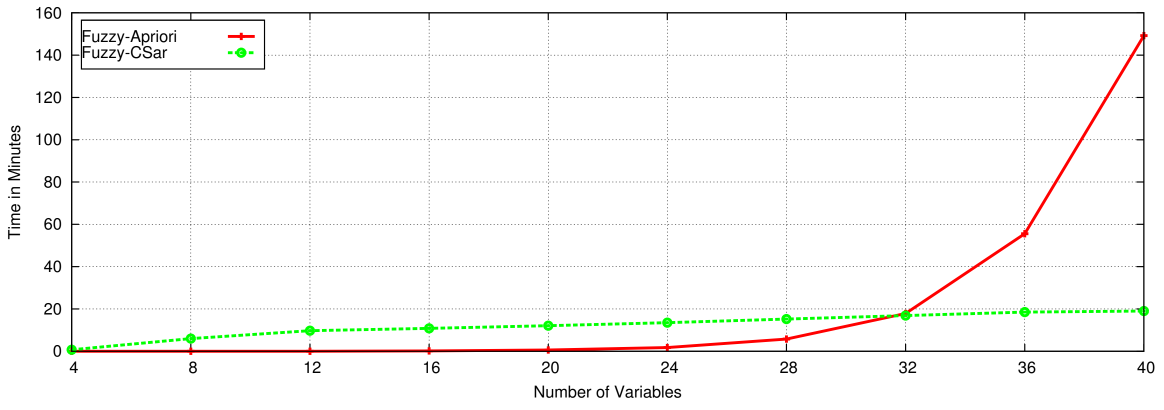


Figure 6.9: Relationship between the runtime (in minutes) and the number of variables used with the 100% of transactions and five linguistic terms.

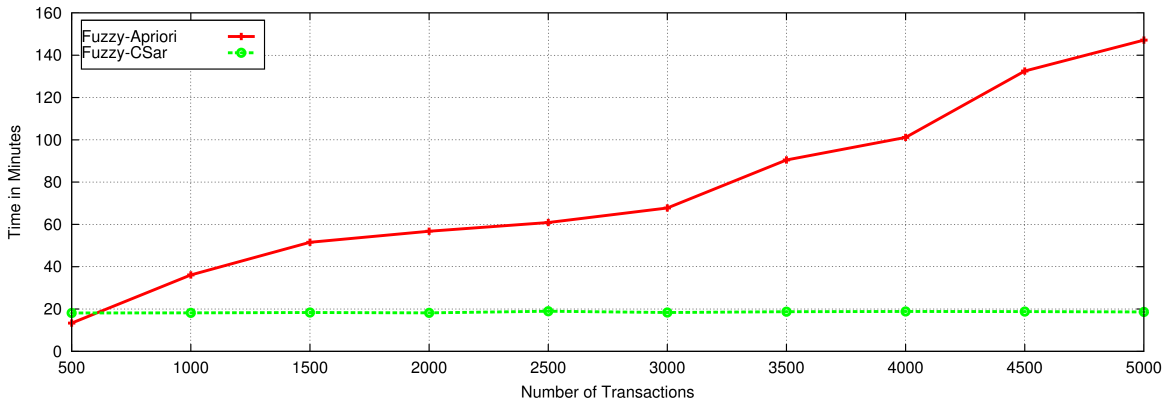


Figure 6.10: Relationship between the runtime (in minutes) and the number of transactions using all the 40 features of the problem and five linguistic terms.

- **Relationship between the runtime and the number of variables.** Figure 6.9 depicts the results of the first experiment. The exponential nature of Fuzzy-Apriori is clearly visible, specially when using 28 (and more) variables its running time starts increasing abruptly. On the other side, Fuzzy-CSar are not dramatically affected for the number of variables, demonstrating a high scalability.

- **Relationship between the runtime and the number of transactions.** Figure 6.10 shows the results of the second experiment. Here it is noticeable that Fuzzy-Apriori is sensitive to the number of transactions in the data base, and its runtime increases almost lineally with the number of transactions. In comparison, Fuzzy-CSar shows an outstanding behaviour.

These experiments support that Fuzzy-CSar has a high-grade of scalability, which is one of the strengths of incremental, online learning algorithms. These kinds of algorithms can stall the learning stage whenever required and the evolved rule set can be used for modelling the environment. In this regard, the more learning iterations Fuzzy-CSar has performed, the more general and accurate rules (in terms of support, confidence, lift and accuracy) should be.

6.6.2 Analysis of the Quality of the Models

To show the quality of the models generated by Fuzzy-CSar, we compared the results with the ones of Fuzzy-Apriori in terms of (1) number of rules, (2) average number of rules obtained in the antecedent part of the rules, (3) support, (4) confidence, and (5) average time required for obtaining such results. In order to make a fair comparison, the output rules of Fuzzy-CSar and Fuzzy-Apriori have been filtered allowing only rules with a high quality (i.e., whose confidence is greater than 0.75).

Table 6.10 shows the results obtained by both algorithms. It is clearly noticeable that Fuzzy-CSar outperform Fuzzy-Apriori in terms of support and confidence. It is worth mentioning the overwhelming number of rules obtained by Fuzzy-Apriori in some of the problems, product of the generate-and-test procedure: this family of techniques suffer from an exponential behaviour. Notice that the average time required for solving such problems is also huge in the case of Fuzzy-Apriori. Also notice how Fuzzy-Apriori is not able to finish the *spa* problem. In comparison Fuzzy-CSar evolves a much more equilibrated population of rules in terms of support and confidence, and also the average time required for obtaining such solutions is much lower.

A statistical analysis has been conducted to evaluate the significance of the results following the recommendations given by Demšar (2006). More specifically, we compared the supports and the confidences of both Fuzzy-CSar and Fuzzy-Apriori by means of the non-parametric Wilcoxon signed-ranks test (Wilcoxon, 1945). The approximate p-values resulting from these pairwise comparisons were provided in the analysis. The reader is referred to appendix A for more information about this statistical procedure.

The results obtained by the statistical tests at $\alpha = 0.05$ are summarised in the following.

- **Support**. In the case of the support we obtain that $z = -4.014$, hence rejecting the hypothesis that both algorithms perform the same on average (H_0). The computed p-value is $5.956 \cdot 10^{-5}$.

- **Confidence**. In the case of the confidence we obtain that $z = -3.945$, hence rejecting the hypothesis that both algorithms perform the same on average (H_0). The computed p-value is $1.204 \cdot 10^{-4}$.

In this regard, Fuzzy-CSar results are significantly better than Fuzzy-Apriori in both support and confidence.

	Fuzzy-CSar Results					**Fuzzy-Apriori Results**				
Id.	**#R**	**#AV**	**sup**	**con**	**#T**	**#R**	**#AV**	**sup**	**con**	**#T**
app	661.200±27.491	3.387±1.816	0.534±0.054	0.910±0.002	99.377±2.782	713	2.804±0.682	0.086±0.002	0.787±0.002	0.230±0.030
bpa	308.700±17.595	2.846±1.619	0.728±0.052	0.919±0.002	49.036±3.145	614	2.651±0.523	0.093±0.003	0.748±0.003	0.220±0.040
fam	1 397.900±40.852	4.548±2.920	0.679±0.049	0.914±0.003	237.626±4.733	7 441	3.718±1.183	0.094±0.003	0.872±0.005	71.563±0.501
gls	1 176.100±37.095	4.186±2.738	0.678±0.033	0.914±0.002	186.504±5.601	4 608	3.863±1.443	0.105±0.004	0.865±0.005	0.181±0.000
h-s	1 769.400±62.764	4.640±3.191	0.568±0.042	0.896±0.003	302.307±7.623	17 026	3.926±1.197	0.080±0.002	0.871±0.005	5.204±0.018
irs	165.900±12.501	2.026±0.548	0.402±0.022	0.906±0.003	16.964±1.267	89	1.876±0.288	0.076±0.001	0.729±0.003	0.230±0.170
mag	1 706.600±25.192	4.754±2.936	0.594±0.041	0.913±0.002	330.333±8.890	7 164	3.358±0.753	0.082±0.002	0.796±0.003	14.464±0.010
µca	2 039.500±35.183	4.351±2.929	0.382±0.036	0.923±0.003	453.461±8.143	7 162 405	8.151±1.667	0.061±0.010	0.930±0.025	51 520.736±2 328.668
pho	196.100±15.788	2.435±1.095	0.678±0.036	0.895±0.003	31.530±2.182	183	1.945±0.227	0.088±0.002	0.693±0.003	0.061±0.000
pim	786.400±25.227	3.872±2.308	0.696±0.032	0.915±0.001	149.680±3.372	724	2.898±0.451	0.077±0.001	0.807±0.003	0.102±0.000
rng	2 131.100±86.244	5.474±6.149	0.486±0.055	0.920±0.003	417.035±20.010	1 727 145	5.753±1.412	0.075±0.001	0.848±0.001	7 854.716±234.547
seg	1 854.800±67.865	4.273±2.758	0.439±0.046	0.913±0.003	363.935±11.729	596 772	6.214±3.468	0.083±0.001	0.911±0.006	896.718±22.663
son	2 030.000±55.596	3.709±2.396	0.399±0.045	0.898±0.003	839.944±20.936	1 941 347	3.250±0.295	0.061±0.000	0.779±0.001	40 737.684±2 958.617
spa	2 207.300±59.057	20.084±9.903	0.642±0.014	0.983±0.000	952.786±38.115	-	-	-	-	-
thy	209.100±29.063	2.317±1.049	0.670±0.086	0.935±0.002	23.307±3.291	182	2.440±0.730	0.138±0.011	0.826±0.005	0.080±0.020
veh	2 098.300±44.445	4.466±2.650	0.411±0.038	0.914±0.003	363.354±10.156	40 779	3.408±0.780	0.068±0.001	0.799±0.003	19.065±1.008
wav	2 132.300±41.075	4.489±4.404	0.415±0.054	0.891±0.005	545.313±8.845	1 262 758	3.372±0.457	0.066±0.000	0.761±0.002	9 417.744±621.681
wdbc	2 049.800±49.859	4.800±5.074	0.458±0.059	0.929±0.003	455.763±13.764	4 273 614	4.757±0.908	0.064±0.000	0.823±0.002	60 643.766±3 704.684
wne	2 146.100±27.012	5.016±3.015	0.359±0.030	0.896±0.002	336.199±4.969	4 292	2.440±0.324	0.069±0.000	0.728±0.002	0.291±0.006
wpbc	2 149.600±23.711	4.015±2.577	0.480±0.043	0.917±0.002	497.874±7.220	852 845	3.643±0.404	0.063±0.000	0.796±0.002	5 274.364±32.128
yst	608.000±14.669	3.743±2.474	0.815±0.017	0.926±0.002	95.588±4.809	2 447	3.631±1.227	0.120±0.010	0.897±0.010	0.307±0.000

Table 6.10: Comparisons of results between Fuzzy-CSar and Fuzzy-Apriori. Columns describe: the identifier of the data set (id.), the average number of rules (#R), the average number of variables in the antecedent (#AV), the average support (sup), the average confidence (con), and the average time in seconds (#T). These values are extracted from rules with a confidence ≥ 0.75.

6.7 Summary, Conclusions and Critical Analysis

6.7.1 Summary and Conclusions

In this chapter we addressed the problem of knowledge discovery from streams of unlabelled real-world data with continuous features from the particular point of view of association stream mining. The goal of this novel field is to extract associations among input variables from streams of information and its main premises are: (1) it consists of continuous flows of data that have to be processed in a single pass, (2) there are restrictions in memory usage, (3) the concept to be learnt may drift over time, and (4) it does not assume any underlying structure nor fixed data distribution. Also, a high degree of interpretability is desirable. It is relevant to highlight that association stream mining differs from frequent pattern mining because this latter field is focused in the identification of frequent variables, relegating the generation of the final rules in a second place using a classic offline approach, which make it mostly ill-suited for our purposes.

The challenges of association streams have been addressed by introducing Fuzzy-CSar, a Michigan-style Learning Fuzzy-Classifier System for extracting association rules. Differently from the classic association mining strategies, the proposed technique performs a single pass over data to obtain the output rules by defining a maximum number of rules to undercover. Our approach handles problems that deliver a continuous flow of unlabelled data, which is usual in many industrial and scientific applications (e.g., Smart Grids) and where the examples provided to the system have to be processed on the fly in order to mine useful information.

An extensive analysis has been performed in order to check the behaviour of the proposed algorithm under association stream problems. First by making use of synthetic environments, each one with a different complexity, and following that, with a real data stream problem with unknown dynamics. The experimentation has been extended by carefully analysing the complexity and scalability of Fuzzy-CSar and the quality of the models evolved by the algorithm. The incremental nature of the presented algorithm is very noticeable when compared to a standard association rule mining method such as Fuzzy-Apriori. The relation between runtime and number of variables and the relation between runtime and number of transactions has shown an outstanding behaviour. Finally, the analysis of the quality of the models under real-world data sets with static concepts has shown that Fuzzy-CSar is very competitive in terms of both support and confidence.

6.7.2 Critical Analysis of Fuzzy-CSar

To finish the study of Fuzzy-CSar a final analysis is performed in the following. Table 6.11 summarises the critical analysis, where *strengths* represent the main advantages of Fuzzy-CSar, *weaknesses* show the drawbacks of the algorithm, *opportunities* suggest the further lines of research to enhance the system performance, and *threads* indicate issues that other machine learning approaches may take advantage of.

Fuzzy-CSar has three main strengths: first, it is a fast algorithm that adapts itself to association drifts and that only needs a single pass over the data to obtain the desired fuzzy association rules. Second, as it is strongly based on XCS (and UCS), Fuzzy-CSar inherits the online architecture and thus it is capable of mining large amounts of data. Third, as the field of association streams require a highly readable output, Fuzzy-CSar exploits its fuzzy representation offering highly legible models to users.

Strengths	**Weaknesses**
- It is a fast algorithm that adapts itself to concept changes and that only needs a single pass over the data to obtain the desired rules. - It deals with large data sets. - It outputs rules that are highly legible.	- It may generate a large amount of rules. - It has many parameters that have to be properly configured. - It requires, in general, a large population to capture the desired patterns out of streams of data.
Opportunities	**Threats**
- It showed a competitive performance which opens opportunities for further research on rule reduction mechanisms and new fuzzy knowledge representations. - Its rule deletion mechanism can be further improved by adding rule-aging mechanisms and a unsupervised association drift detector. - Association streams is a field that remains virtually unexplored.	- It has a large number of parameters that have to be correctly set and this issue may discourage practitioners. - Association streams is a new field and many researchers may not know how to take advantage of it.

Table 6.11: Critical analysis of Fuzzy-CSar.

The main weakness of Fuzzy-CSar is that it may generate a large amount of rules, which may hinder the interpretability of the results given by the algorithm. Related with this issue, the proposed algorithm has many parameters that have to be properly configured in order to obtain the desired output. Also, Fuzzy-CSar requires, in general, a large population to capture the desired patterns out of the streams in which it is applied.

A possible threat is that, although being competent in both dynamic and static environments, the proposed algorithm is a complex system and therefore practitioners have to know how to configure it correctly. This issue may discourage newcomers to explore the possibilities of Fuzzy-CSar. Also, as the field of association streams is very recent, many researcher may not know how to take advantage of the possibilities of the algorithm.

Fuzzy-CSar shows some interesting opportunities which will be developed in future work. Four main lines are considered as future work: (1) the inclusion of rule reduction mechanisms without significant loss of model quality, (2) a more detailed research of other fuzzy knowledge representations, checking the tradeoff between flexibility and interpretability, (3) a new rule deletion mechanism that contains rule-aging and a unsupervised association drift detector to improve the responsiveness of the system, and (4) as association streams is a very novel field and consequently it remains virtually unexplored, Fuzzy-CSar may become a referent to the field.

7
A Deeper Look at Fuzzy-CSar

Fuzzy-CSar is a Michigan-style LCS that has been designed to mine association streams, obtaining interesting relations in the form of fuzzy association rules out of streams of information. Recall that Fuzzy-CSar is a complex system in which several components interact within themselves in order to obtain the desired modelling of the stream of examples. In the previous chapter the behaviour of Fuzzy-CSar was carefully analysed in a set of synthetic and real-world problems that ended showing the competitiveness of the algorithm. However no theoretical framework were given; we simply assumed that the same evolutionary pressures of XCS, the most well-known Michigan-style LCS, apply. In this chapter we take the theoretical framework from XCS (Butz, 2004, 2006, Stalph et al., 2012) and extrapolate them onto Fuzzy-CSar, analysing critical bounds on the system behaviour. The main difficulty on performing this analysis lies in the notion of what is a building block under fuzzy logic: whereas in the original ternary alphabet of XCS is straightforward, in the case of fuzzy logic is not so clear. For this purpose an ideal behaviour of the algorithm is assumed and design decomposition (Goldberg, 2002) is then utilised.

The purpose of this chapter is to take the first steps towards a theory of generalisation and learning of Fuzzy-CSar departing from the existing formalisms of XCS and its related algorithms. Also, the lessons learned from this analysis result in several configuration recommendations for applying Fuzzy-CSar on any type of problems.

This chapter is organised as follows: First, section 7.1 introduces the necessity of deriving facet-wise models for understanding complex systems as Michigan-style LCSs. Next, section 7.2, details the formal framework for fuzzy-CSar, introducing the challenges that every Michigan-style LCS has to overcome in order to obtain competitive results. Afterwards, section 7.3 give guidelines on the task of configuring the algorithm to obtain accurate results. Finally, section 7.4 summarises and concludes the chapter.

7.1 Introduction

Michigan-style LCSs are complex algorithms that integrate a knowledge representation based on individuals that enables the system to map sensorial status with actions, an apportionment of credit mechanism which evaluates the subsolutions, and a search mechanism for discovering new knowledge in order to obtain maximally-general and accurate solutions (Butz, 2006). Further, implementations of this family of techniques have demonstrated its competence in several machine learning tasks such as classification, regression, clustering or association rule mining (Butz, 2006, Orriols-Puig et al., 2008a,b, Tamee et al., 2006).

However, this family of techniques suffer from a large number of parameters that practitioners have to properly configure in order to obtain the desired solutions. This issue has been handled in XCS, the most well-known implementation of the Michigan-style LCS framework: with the goal of understanding better these complex systems and XCS in particular, Butz (2004) introduced a theoretical framework with which practitioners could understand the generalisation capacity and learning of such algorithm and also configure it to maximise the output according to the particular problem to solve. Butz (2004) demonstrated that, effectively, XCS was capable of solving problems accurately and efficiently and also derived bounds on important parameters that practitioners had to take into account. Later on and using Butz's framework as a basis, Orriols-Puig (2008) derived a formal model in which was demonstrated that XCS is capable of solving problems with moderate imbalance ratios (i.e., classification problems in which one of the classes, the so called nourished class, has a much larger frequency of appearance than the others, the so called starved classes) if configured properly. These models, however, were derived for the ternary representation of the algorithm. Recently, Butz (2004) introduced the models for the real-coded version of XCS and later Stalph and Butz (2010) formalised these ideas for XCSF.

In this chapter we perform a further look at Fuzzy-CSar by introducing a formal framework towards a theory of generalisation and learning departing from the formalisms done in XCS and the related techniques. These formalisms result in several configuration recommendations for properly configuring the system.

This chapter provides the following contributions:

- It gives the first steps towards a formal framework for Fuzzy-CSar.
- It gives guidelines for configuring Fuzzy-CSar according to the problem to face.

In the following we introduce the necessary background.

7.2 Framework

Following the work of Butz et al. (2004), we start describing the necessary conditions for which XCS works. It is worth noting that these conditions are pretended for the ternary representation of XCS, although some posterior research (Orriols-Puig et al., 2010, Stalph and Butz, 2010, Stalph et al., 2012) deduced that with minor modifications the same formalisms apply to other representations (e.g., the real-coded one). Afterwards, for each one of these conditions we extrapolate them onto the Fuzzy-CSar domain.

7.2.1 Learning Challenges, a Brief Tour

Fuzzy-CSar, being an implementation of the Michigan-style LCS open architecture, has to solve the same challenges than XCS in order to learn successfully. In this regard, Butz (2006) identified a series of challenges that hamper the learning stage of XCS. Despite the use of the original ternary representation to define these challenges, later on Stalph and Butz (2010) showed that these challenges appear as well using the real-coded representation. These challenges are (1) the covering challenge, (2) the schema and reproductive opportunity challenge, (3) the learning time challenge, and (4) the solution sustenance challenge. These are more elaborated in what follows.

- **The covering challenge**. The initial population in the algorithm should be able to cover the full problem space (hence the name of this challenge). If this condition is not met, the algorithm can get stuck in an infinite covering-deletion loop.
- **The schema and reproductive opportunity challenge**. This challenge is focused in the difficulty the genetic algorithm has at discovering and reproducing better solutions, or in other words: in the lack of fitness signal guidance of the evolutionary component.
- **The learning time challenge**. It is concerned with the time until maximally general and accurate solutions evolve. For instance, small populations may delay the learning time because there is a higher possibility of deleting good individuals due to the lack of space in the population.
- **Solution sustenance challenge**. Good solutions have to last in the population whereas bad ones have to perish. This challenge is focused in the deletion mechanism of the system avoiding the obliteration of maximally-general and accurate individuals.

The study of this challenges allow the derivation of critical bounds in the population size and learning time, which are of the utmost usefulness for practitioners in order to properly configure the system (Stalph et al., 2012). It is worth noting that each challenge assumes that the rest behave in an ideal manner.

7.2.2 The Covering Challenge

As pointed out elsewhere (Stalph et al., 2012), before deriving the bounds for overcoming the covering challenge, it is important to highlight that the original bounds derived for XCS are based on the *specificity* of particular individuals; that is, the average proportion of *non don't care symbols* of individuals in the population. The specificity is directly related to the probability of matching of individuals. Later on Butz (2006) related the notion of specificity with the hyper-volumes in real-valued input spaces using a real-coded version of XCS. In our particular case, we have related the matching degree of the variables used by individuals with the probability of matching.

We consider the probability that one instance is covered by, at least, one individual $P(\text{cover})$. According to Stalph et al. (2012), this probability given a randomly initialised population of size N is:

$$P(\text{cover}) = 1 - [1 - P(\text{match})]^N . \tag{7.1}$$

The main difficulty on performing this analysis lies in the determination of $P(\text{match})$ under the fuzzy logic domain. For solving that issue, we related the matching degree of the variables in individuals with the probability that one individual matches an input instance $P(\text{match})$, that is:

$$P(\text{match}) = \frac{\mu_{\widetilde{A}}(x)^{\ell-1} \cdot \mu_{\widetilde{C}}(x)}{\lambda^{\ell}}, \tag{7.2}$$

where $\mu_{\widetilde{A}}(x)$ is the matching degree of the antecedent part of the rule, ℓ is the number of variables of the rule, $\mu_{\widetilde{C}}(x)$ is the matching degree of the consequent part of the rule, and λ is the number of fuzzy labels used during the fuzzication process.

We simplify Equation 7.2 by handling all the variables in the same form via the matching degree of the input variables $\mu_{\widetilde{F}}(x)$, that is:

$$P(\text{match}) \approx \left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}. \tag{7.3}$$

Then, $P(\text{cover})$ related to the matching degree of each variable in the rule in the following way:

$$P(\text{cover}) = 1 - \left[1 - \left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}\right]^{N}. \tag{7.4}$$

Recognizing that $(1 - r/n)^n \approx e^{-r}$ we can manipulate the aforementioned probability obtaining:

$$P(\text{cover}) \approx 1 - e^{-N\cdot\left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}}, \tag{7.5}$$

which increases exponentially with the population size and the matching degree of all the variables used by the rule.

Following the reasoning of Stalph et al. (2012), we can derive a population bound size out of Equation 7.5 when requiring a sufficiently high probability of covering all input instances, that is $P(\text{cover}) \geq 1 - 10^{-c}$, where $c \geq 1$ is the *certainty* (the greater c, the more higher $P(\text{cover})$) as shown:

$$\begin{aligned}
&P(\text{cover}) \geq 1 - 10^{-c};\\
\Longrightarrow\;& 1 - e^{-N\cdot\left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}} \geq 1 - 10^{-c};\\
\Longrightarrow\;& e^{-N\cdot\left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}} \geq 10^{-c};\\
\Longrightarrow\;& N \cdot \left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell} \geq c \cdot \ln 10;\\
\Longrightarrow\;& N \geq \frac{\lambda^{\ell} \cdot c \cdot \ln 10}{\mu_{\widetilde{F}}(x)^{\ell}}.
\end{aligned} \tag{7.6}$$

This bound shows that the population size grows exponentially on the number of labels, linear with the certainty exponent and inversely exponentially with the matching degree of the variables of the individual. Recall that Fuzzy-CSar is designed to generate individuals with maximal matching degree during the covering stage (see chapter 6). Notice that this bound is useful to prevent a covering-deleting cycle and also gives guidelines in order to properly configure Fuzzy-CSar.

7.2.3 The Schema and Reproductive Opportunity Challenge

As stated by Stalph et al. (2012), guarantying a full coverage of the problem space does not assure that the learning is successful. The discovery component (the GA) has to have a fitness signal guidance towards better individuals. This issue is critical in order to allow any Michigan-style LCS to obtain maximal general and accurate solutions. Notwithstanding, analysing this issue under the fuzzy logic domain is quite a challenge due to the fact that the notion of building block (BB) is not clear.

Assuming a random deletion (a typical assumption of design decomposition for simplification (Goldberg, 2002)), the probability of erasing an individual out of the population $P(\text{deletion})$ is defined by (Butz, 2006):

$$P(\text{deletion}) = \frac{2}{N}, \tag{7.7}$$

since each time the GA is triggered two offspring are generated. Following that, we derive the probability that one individual is reproduced $P(\text{reproduction})$, assuming the extreme case that every possible combination of variables is present and that all have the same probability (i.e., a pessimistic bound):

$$P(\text{reproduction}) = \frac{\mu_{\widetilde{F}}(x)^{\ell}}{\sum_{i=2}^{\ell} i \cdot \binom{\ell}{i}}. \tag{7.8}$$

Following the reasoning of Butz (2006), we can determine the probability that neither reproduction nor deletion occurs by combining Equation 7.7 with Equation 7.8:

$$\begin{aligned} P(\text{no del., no rep.}) &= (1 - P(\text{deletion})) \cdot (1 - P(\text{reproduction})) = \\ &= \left(1 - \tfrac{2}{N}\right) \cdot \left(1 - \frac{\mu_{\widetilde{F}}(x)^{\ell}}{\sum_{i=2}^{\ell} i \cdot \binom{\ell}{i}}\right) = \\ &= 1 - \tfrac{2}{N} - \frac{\mu_{\widetilde{F}}(x)^{\ell}}{\sum_{i=2}^{\ell} i \cdot \binom{\ell}{i}} \cdot \left(1 - \tfrac{2}{N}\right). \end{aligned} \tag{7.9}$$

Now we are in position to derive the probability that a certain individual is part of an association set—hence reproduced—before it is removed from the population, again following Butz (2006):

$$\begin{aligned}
P(\text{rep. before del.}) &= P(\text{reproduction}) \cdot (1 - P(\text{deletion})) \cdot \tfrac{1}{1-P(\text{no del., no rep.})} = \\
&= \tfrac{\mu_{\widetilde{F}}(x)^\ell}{\sum_{i=2}^{\ell} i \cdot \binom{\ell}{i}} \cdot \left(1 - \tfrac{2}{N}\right) \cdot \tfrac{1}{1-\left\{1-\frac{2}{N}-\frac{\mu_{\widetilde{F}}(x)^\ell}{\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}\cdot\left(1-\frac{2}{N}\right)\right\}} = \\
&= \frac{\frac{\mu_{\widetilde{F}}(x)^\ell}{\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}\cdot\left(1-\frac{2}{N}\right)}{\frac{2}{N}+\frac{\mu_{\widetilde{F}}(x)^\ell}{\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}\cdot\left(1-\frac{2}{N}\right)} = \\
&= \frac{\frac{\mu_{\widetilde{F}}(x)^\ell}{\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}}{\frac{2}{N-2}+\frac{\mu_{\widetilde{F}}(x)^\ell}{\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}} = \\
&= \frac{(N-2)\cdot\mu_{\widetilde{F}}(x)^\ell}{(N-2)\cdot\mu_{\widetilde{F}}(x)^\ell+2\cdot\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}.
\end{aligned} \tag{7.10}$$

Furthermore, we can derive the complementary probability $1 - P(\text{rep. before del.})$, as follows:

$$1 - P(\text{rep. before del.}) = \frac{\sum_{i=2}^{\ell} i \cdot \binom{\ell}{i}}{(N-2)\cdot\mu_{\widetilde{F}}(x)^\ell + 2\cdot\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}. \tag{7.11}$$

Finally, recognising that a certain minimum reproduction before the deletion is required, that is: $P(\text{rep. before del.}) > 1 - 10^{-c}$, approximating $1 - 10^{-c} \approx 1$ for even small values of c (e.g., $c = 10$ will fit), and solving for the population size N, we derive the following bound:

$$\begin{aligned}
&P(\text{rep. before del.}) > 1 - 10^{-c}; \\
&\Longrightarrow \tfrac{(N-2)\cdot\mu_{\widetilde{F}}(x)^\ell}{(N-2)\cdot\mu_{\widetilde{F}}(x)^\ell+2\cdot\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}} > 1 - 10^{-c}; \\
&\Longrightarrow (N-2)\cdot\mu_{\widetilde{F}}(x)^\ell > (N-2)\cdot\mu_{\widetilde{F}}(x)^\ell + 2\cdot\textstyle\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}; \\
&\Longrightarrow N > \tfrac{2\cdot\mu_{\widetilde{F}}(x)^{-\ell}\sum_{i=2}^{\ell} i\cdot\binom{\ell}{i}}{1-P(\text{rep. before del.})}.
\end{aligned} \tag{7.12}$$

It is worth mentioning that this population bound has been derived from a pessimistic assumption and that, as experiments in the previous chapter shown, Fuzzy-CSar is able to evolve a solution without such a large bound.

7.2.4 The Learning Time Challenge

According to Butz (2006) and considering the effects of mutation in isolation, the learning time depends on the time on the number of mutations from initial completely-general individuals to maximally-general and accurate ones. In this sense, specialisation is randomly performed via the mutation operator, provided that the best individual is not lost and it is selected as offspring when it participates in the association set. Assuming the effects of removing a variable in the antecedent of a rule and ignoring the rest we can estimate the probability that mutation correctly specifies the next feature $P(\text{correct mut.})$ as:

$$P(\text{correct mut.}) = P(\text{remove var. antecedent}) \cdot (1 - P(\text{remove var. antecedent}))^k, \tag{7.13}$$

where k is the number of variables to leave *untouched* by the mutation operator and $P(\text{remove var. antecedent})$ is the probability of removing a variable in the antecedent via the mutation operator, that is:

$$P(\text{remove var. antecedent}) = \frac{P_C \cdot (\aleph - 1)}{2 \cdot (\ell - 2)}, \tag{7.14}$$

where P_C is the user-defined parameter that controls mutation and $\aleph$ is the number of variables in the antecedent part of the rule of the current individual.

We can easily derive probability of reproduction $P(\text{reproduction})$ given an individual that specifies k variables:

$$P(\text{reproduction}) = \frac{\mu_{\widetilde{F}}(x)^k}{\sum_{i=2}^{k} i \cdot \binom{k}{i}}. \tag{7.15}$$

Then, combining Equation 7.13 with Equation 7.15 the probability of generating a better offspring than the current best $P(\text{gen. better ind.})$ is determined by:

$$\begin{aligned} P(\text{gen. better ind.}) &= P(\text{reproduction}) \cdot P(\text{correct mut.}) = \\ &= \frac{\mu_{\widetilde{F}}(x)^k}{\sum_{i=2}^{k} i \cdot \binom{k}{i}} \cdot \left(\frac{P_C \cdot (\aleph-1)}{2 \cdot (\ell-2)}\right) \cdot \left(1 - \frac{P_C \cdot (\aleph-1)}{2 \cdot (\ell-2)}\right)^k. \end{aligned} \tag{7.16}$$

We can simplify Equation 7.16 recognizing that $(1 - r/n)^n \approx e^{-r}$ as follows:

$$P(\text{gen. better ind.}) \approx \frac{\mu_{\widetilde{F}}(x)^k}{\sum_{i=2}^{k} i \cdot \binom{k}{i}} \cdot \left(\frac{P_C \cdot (\aleph - 1)}{2 \cdot (\ell - 2)}\right) \cdot e^{\frac{-k \cdot P_C \cdot (\aleph-1)}{2(\ell-2)}}. \tag{7.17}$$

We are now in position to derive the time required to generate the next best individual as:

$$\begin{aligned} t(\textit{gen. better ind.}) &= \frac{1}{P(\text{gen. better ind.})} = \\ &= \frac{1}{\frac{\mu_{\widetilde{F}}(x)^k}{\sum_{i=2}^{k} i \cdot \binom{k}{i}} \cdot \left(\frac{P_C \cdot (\aleph-1)}{2 \cdot (\ell-2)}\right) \cdot e^{\frac{-k \cdot P_C \cdot (\aleph-1)}{2(\ell-2)}}} = \\ &= \frac{2 \cdot (\ell-2) \cdot \sum_{i=2}^{k} i \cdot \binom{k}{i}}{\mu_{\widetilde{F}}(x)^k \cdot P_C \cdot (\aleph-1) \cdot e^{\frac{-k \cdot P_C \cdot (\aleph-1)}{2(\ell-2)}}}. \end{aligned} \tag{7.18}$$

Finally, given a problem in which k_v variables need to be specified we can derive the time until the generation of the global best individual as:

$$\begin{aligned} t(\textit{gen. best ind.}) &= \sum_{k=1}^{k_v} \frac{2 \cdot (\ell-2) \cdot \sum_{i=2}^{k} i \cdot \binom{k}{i}}{\mu_{\widetilde{F}}(x)^k \cdot P_C \cdot (\aleph-1) \cdot e^{\frac{-k \cdot P_C \cdot (\aleph-1)}{2(\ell-2)}}} = \\ &= \frac{2 \cdot (\ell-2)}{P_C \cdot (\aleph-1)} \cdot \sum_{k=1}^{k_v} \frac{\sum_{i=2}^{k} i \cdot \binom{k}{i}}{\mu_{\widetilde{F}}(x)^k \cdot e^{\frac{-k \cdot P_C \cdot (\aleph-1)}{2(\ell-2)}}}. \end{aligned} \tag{7.19}$$

7.2.5 The Solution Sustenance Challenge

Assuming that all the other challenges are met, Fuzzy-CSar guarantees that it evolves a solution to the problem faced. However, this solution has to last in the population. As pointed out by Stalph et al. (2012), this challenge is focused with the deletion probability, where the goal of preventing the obliteration of a niche in the solution representation. Moreover, given a particular probability of matching, it is possible to determine the population size bound that is necessary to assure matching and thus reproduction before complete deletion of representatives. Butz et al. (2007) demonstrated that the distribution of niche representatives yields a binomial distribution. The probability that a particular subspace is represented by j individuals $P(\text{Subspace represented by } j)$ is thus derived as:

$$P(\text{Subspace represented by } j) = \binom{N}{j} \cdot P(\text{match})^j \cdot (1 - P(\text{match}))^{N-j}. \tag{7.20}$$

Following Stalph et al. (2012) reasoning, the population size can be bounded to ensure that no representatives are removed. A niche is lost when there are no representatives—i.e., $P(\text{Subspace represented by } 0) = (1 - P(\text{match}))^N$—thus, the probability of losing a niche decreases exponentially with the population size. Using the same procedure as in the case of Equation 7.6 we obtain the following:

$$\begin{aligned}
&1 - P(\text{Subspace represented by } 0) \geq 1 - 10^{-c};\\
&\implies 1 - (1 - P(\text{match}))^N \geq 1 - 10^{-c};\\
&\implies 1 - e^{-N\cdot\left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}} \geq 1 - 10^{-c};\\
&\implies e^{-N\cdot\left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell}} \geq 10^{-c};\\
&\implies N \cdot \left(\frac{\mu_{\widetilde{F}}(x)}{\lambda}\right)^{\ell} \geq c \cdot \ln 10;\\
&\implies N \geq \frac{\lambda^{\ell} \cdot c \cdot \ln 10}{\mu_{\widetilde{F}}(x)^{\ell}},
\end{aligned} \tag{7.21}$$

which is the exact population size bound as derived for the covering challenge. This bound shows that to sustain maximally-general and accurate solutions the population size has to grow exponentially on the number of labels, linear with the certainty exponent and inversely exponentially with the matching degree of the variables of the individual.

7.3 Parameter Setting Guidelines

As in common with many competitive members of the Michigan-style LCS family, Fuzzy-CSar has several parameters that have to be properly configured in order to obtain high quality solutions. Despite the experiments in the last chapter show the robustness of Fuzzy-CSar to configuration parameters, these will affect the behaviour of the algorithm it is important to know how to configure them correctly for particular problems. In the following we give guidelines for parameter configuration.

- **Population size** N. Probably the most fundamental of them all, the population size specifies the available workspace for the evolutionary search. As reflected in section 7.2, it is critical to set this parameter high enough to prevent the deletion of accurate individuals.

- **Generalization in initialization** $P\#$. This parameter controls the degree of generalisation of Fuzzy-CSar, thus it important to set it large enough in order to let the algorithm to generalise. However, it should be set small enough to avoid the generation of over general individuals.

- **Fitness pressure** ν. In XCS and derivates, the fitness pressure parameter drives the genetic search towards a highly general an accurate set of individuals avoiding over generals (Orriols-Puig et al., 2009a), hence being a good practice setting it to a high value (i.e., 10). However, in the particular case of Fuzzy-CSar, this parameter had little effects as experiments in the last chapter support.

- **GA frequency threshold** θ_{GA}. As it happens with other Michigan-style LCS algorithms, this thresholds controls the application rate of the evolutionary discovery component. Ideally, it has to be set small enough to allow the discovery of new and interesting solutions but large enough to prevent forgetting.

- **Crossover probability** P_χ. The probability crossover controls the application rate of application of the crossover operator. Since all GBML learning techniques work in roughly the same way, that is: decomposition and reassembly, having this parameter set to a high value is usually a good practice.

- **Mutation rates** $\{P_C, P_{I/R}, P_\mu\}$. Typically in GBML algorithms, the mutation rate is set to a low value (e.g., $1/\ell$) to, as shown in section 7.2, minimise the deletion of good solutions. Other studies related to GBML (Goldberg, 2002) agree with the notion that mutation should be keep to small values.

- **Other thresholds** $\{\theta_{exp}, \theta_{del}, \theta_{sub}, \theta_{mna}, \delta\}$ These parameters define the behaviour of Fuzzy-CSar in certain situations such as the minimum experience required for an individual in order to subsume, the minimum experience required for being deleted from the population the number of covered classifiers or the ratio of the average population fitness used in the deletion mechanisms. Studies in Fuzzy-UCS (Orriols-Puig et al., 2009b) suggest that these parameters have little effects on the behaviour of the algorithm.

7.4 Summary and Conclusions

In this chapter we extrapolated the theory on XCS, the most studied Michigan-style LCS, to Fuzzy-CSar. Because both algorithms share the system architecture, they both share the same challenges that have to be overcome. These challenges are the following:

The covering challenge, which is based on the observation that the initial population in the algorithm should be able to cover the full problem space. If this condition is not met, the algorithm can get stuck in an infinite covering-deletion loop.

The schema and reproductive opportunity challenge which is focused in the difficulty the evolutionary component has at discovering and reproducing better solutions (i.e., lack of fitness signal guidance).

The learning time challenge that is concerned in the time until maximally general and accurate solutions evolve.

Solution sustenance challenge which is centred in the survival of good solutions.

These must be satisfied in order to guarantee that the algorithm is able to learn a problem. The study of this challenges has permitted the derivation of critical bounds to ensure that the algorithm work, and also give configuration recommendations to practitioners.

8 Summary, Conclusions and Future Work Lines

Mining large amounts of data in the form of continuous, high-speed, noisy and time-changing streams of information is a trending topic in the machine learning community since many industrial and scientific applications generate data in such way. This thesis has investigated the online learning nature of Michigan-style LCSs for mining data streams. More precisely, we addressed the challenges of learning from both supervised and unsupervised domains when tackling data stream problems based on the guidance of the Michigan-style LCS open framework. Regarding the supervised learning domain we designed and implemented SNCS, a neural-constructivist LCS with a fast reaction capacity to concept drift and robust to noisy inputs. In the case of the unsupervised learning paradigm we addressed two distinct fields: clustering and association streams. In the one hand we extended and improved XCSc (now referred to as XCScds) for clustering data streams in the Smart Grid scenario, outperforming previous proposals while fitting with the Smart Grid premises. In the other hand we presented association stream mining, a novel field that regards on modelling dynamically complex domains via association rules. For that purpose we detailed Fuzzy-CSar, a Michigan-style LCS for mining in such environments.

In this chapter, we summarise and conclude the results and observations provided along the consecution of each challenge. Finally, we discuss the forthcoming research that will be made mostly as a consequence of the insights and results provided by this thesis.

8.1 Summary and Concluding Remarks

In recent times, the machine learning community, strongly inspired by artificial intelligence and statistics, has developed computer programs that learn from the experience to solve real-world problems. Machine learning techniques are adequate to solve problems that are way too complex to manually design and code, when the problem requires high degree of adaptation to changing environments, and when there is a need to process large amounts of data to extract useful information from these data.

Today's data of most industrial and scientific applications are generated online and collected continuously and massively (Gama, 2010). Consequently, the need of systems that are able to monitor, track and control these massive amounts of information has increased; that is, systems that are able to extract useful information from data streams are a trending topic. Smart Grids and stock markets are sound examples of the primary targets of this kind of knowledge extraction. Differently from traditional warehousing systems, these continuous flows of data poses the following main characteristics: (1) data are potentially unbounded in size and consequently practitioners dispose from a limited usage of memory, (2) data can only be handled once due to its size and fast arrival rate, and (3) a fixed data distribution cannot be assumed since concept drifts may happen. This dynamic nature is one of the distinctive traits of data streams, thence hampering the learning process and making it mostly unpractical for traditional data mining systems. Also, as these data come mostly from sensors, large and varying amounts of noise are present.

This thesis started with the description of the challenges that data streams poses to learning systems and the identification of Michigan-style LCSs as a mature framework that fulfils these requisites since they are online learners that foster crossbreeding between different learning paradigms and are characterised by: (1) a knowledge representation—typically based on a set of independent classifiers—that enables the system to map sensorial status with actions, (2) an apportionment of credit mechanism based on reinforcement learning which evaluates the classifiers, and (3) a knowledge discovery mechanism—typically a GA. The general Michigan-style LCS open framework has been detailed in this thesis by concisely describing XCS, by far the most well-known and influential Michigan-style LCS (Orriols-Puig, 2008). We proposed to explore the online learning nature of this architecture when facing data streams. More specifically:

1. Revise and improve the characteristics of Michigan-style LCSs for supervised learning in data stream classification tasks.

2. Revise, extend and improve the characteristics of Michigan-style LCSs for unsupervised learning in clustering data streams.

3. Explore and enrich the characteristics of Michigan-style LCSs for unsupervised learning by introducing association streams.

In our proposal we did not limit ourselves to a single branch of machine learning, but we covered distinct issues of data streams from the point of view of both supervised and unsupervised learning paradigms. As proceeds, we summarise our contributions and provide the key conclusions extracted from each point.

Revise and improve the characteristics of Michigan-style LCSs for supervised learning in data stream classification tasks. Data streams are a tough environment due to its dynamic characteristics and poses the following challenges to be faced by learning algorithms: fast reaction to concept changes, robustness against noise, and obtaining accurate results under high dimensional spaces. Due to the online nature of Michigan-style LCSs, XCS is capable of handling dynamic data stream environments. However, as shown in chapter 4, it requires a huge population to obtain accurate results and its reaction capacity is not as competitive as desirable under high dimensional problems. This issue is mainly due to its rule-based representation and its reinforcement learning behaviour. With the aim of solving this issue we adapted the Michigan-style LCS open architecture by integrating a more flexible representation resulting in a brand new Michigan-style neural-learning classifier system, SNCS, which has been designed to deal with data streams. We identified the common difficulties of learning form supervised data streams as abrupt concept changes, varying concept drift speeds, varying noise levels, virtual drifts, spurious relationships among padding variables under high dimensional spaces and non-linear boundaries. We selected a set of widely-used benchmark problems that allowed us to test those complexities—we also proposed a newone—and we analysed in detail the results obtained by SNCS. We also included other well-known data stream mining algorithms for comparisons as well as XCS and UCS, the most well-known relatives to SNCS. Experiments performed support that SNCS has a high degree of robustness against the identified data stream mining complexities. That robustness is specially noticeable on the SEA problem and variants, where SNCS showed a remarkable reaction capacity to concept changes and noisy inputs, even with padding variables in high dimensional spaces, obtaining better overall results than CVFDT, a competitive data stream miner. We extended the analysis of the competitiveness of SNCS by experimenting with a set of real-world classification problems, all containing stationary concepts, by comparing the accuracy results of SNCS with the ones of the most significant machine learning methods. SNCS has shown to be competitive with stationary classification tasks as well, resulting in models that were not significantly different from that of the ones created by some of the most successful machine learning techniques. In summary, the Michigan-style LCS framework has shown empirically that it is very adequate for supervised data stream tasks, being at the same level as the top-rated algorithms of the field.

Revise, extend and improve the characteristics of Michigan-style LCSs for unsupervised learning in clustering data streams. Supervised learning aims at making accurate predictions after deeming an underlying structure in data, which requires the presence of a *teacher* during the learning phase. In many real-world problems this paradigm may be ill-suited due to the dearth of training examples and the costs of labelling the required information to train the system. Building useful model representations from these unlabelled data is a task that requires the use of unsupervised learning because this paradigm does not assume any *a priori* structure in the data. A cornerstone of unsupervised learning under dynamic environments is found in clustering data streams. Despite the fact that the Michigan-style LCS framework has been successfully used for clustering tasks, these studies focused exclusively on pure offline approaches. Therefore, we revised and upgraded the learning architecture of XCSc to foster the usage of this algorithm in online domains. Further, we implemented partitioning policies of a Smart Grid data storage system with XCScds, where evolving behaviour capacities were required. Conducted experiments show the competitive behaviour of the presented approach by conducting a series of experiments on a classic data stream synthetic environment, an extended data stream synthetic environment with evolving components, and a realistic scenario using the standard benchmarks proposed by the *Yahoo!* Cloud Serving Benchmark. It is worth mentioning that in this last environment XCScds outperforms previous proposals and truly fits with the Smart Grid premises.

Explore and enrich the characteristics of Michigan-style LCSs for unsupervised learning by introducing association streams. The field of association streams regards on modelling dynamically complex domains via association rules without assuming any a priori structure. Its main characteristics are that data comes in continuous and massive flows that have to be processed in a single pass having into account that there are memory constraints and that the concept to be learnt may drift over time while providing a high degree of interpretability. Also, it is not possible to assume any underlying structure nor fixed data distribution. The challenges of association streams have been addressed by introducing Fuzzy-CSar, a Michigan-style learning fuzzy-classifier system for extracting association rules. Differently from the classic association mining strategies, the proposed technique performs a single pass over data to obtain the output rules by defining a maximum number of rules to undercover. Our proposal crossbreeds ideas from LCSs and fuzzy logic in order to handle imprecision and approximate reasoning and provide the creation of highly legible models from continuous flows of both qualitative and quantitative unlabelled data. An extensive analysis has been performed in order to check the behaviour of the proposed algorithm under association stream problems, first by making use of synthetic environments, each one with a different complexity and then with a real data stream problem with unknown dynamics. The experimentation has been extended by carefully analysing the complexity and scalability of Fuzzy-CSar and the quality of the models evolved by the algorithm. Finally, the analysis of the quality of the models under real-world data sets with static concepts has shown that Fuzzy-CSar is very competitive in terms of both support and confidence. Moreover, because it specialises the architecture of XCS for handling association streams, Fuzzy-CSar shares the same learning challenges. The study of this challenges has permitted the derivation of critical bounds to ensure that the algorithm works, and also it gives configuration recommendations to practitioners. The framework provided by Michigan-style LCS is very attractive for handling association streams due to its inherent online learning architecture that is able to adapt quickly and reliably to new situations.

As concluding remarks, the contributions of this thesis show that the Michigan-style LCSs present a general-purpose, competitive, flexible and mature framework for managing data streams whether the environment to handle requires the use of supervised or unsupervised learning paradigms. Consequently, LCSs are an appealing machine learning framework for facing any kind of problem. In the next section we discuss the forthcoming research that will be made as a consequence of the insights and results provided by this thesis.

8.2 Future Work Lines

This thesis has revised the Michigan-style LCS as a competitive framework capable of handling data streams in both supervised ad unsupervised learning tasks. However, there are still some open issues left for further investigations. In this section we discuss the important research lines that will be followed and that come naturally from the work done in this thesis. These are the following:

1. study data streams under imbalanced domains,
2. crossbreeding between supervised and unsupervised techniques,
3. explore graphical models for interpretability enhancement, and
4. exploit the online capacities of Michigan-style LCSs for Big Data.

While the first results in an improvement over the presented algorithms—and specially in the case of SNCS—, the second research line results in the mixing of the two distinct learning paradigms to overcome their individual limitations, the third explores graphical representations for further increase in readability of the knowledge obtained by the learners, and finally the fourth research line focusses on exploiting the online capacities of Michigan-style LCSs framework for extracting new and useful information out of today's most trendy topic: Big Data. As proceeds, we detail the aforementioned research lines in the following.

Study data streams under imbalanced domains. Currently, most of the environments and data sets used to test whether a classifier competently works when addressing data streams are all balanced—i.e., data has approximately the same number of instances for each one of the classes of the problem. This characteristic is due to the inherent difficulties that data streams poses to learning algorithms. However, it has been identified elsewhere (Orriols-Puig, 2008) that the key knowledge of problems that elude solution is hidden in examples that are *rare* in nature—i.e., instances of the minority class when the relation between the majority and the minority classes is high. In this regard, we envisage the generation of a set of standardised problems containing configurable class imbalances ratios to encourage practitioners to test their algorithms for solving the data stream challenges.

Crossbreeding between supervised and unsupervised techniques. Data mining techniques are traditionally classified into two distinct disciplines: supervised and unsupervised learning paradigms. The former aims to make accurate predictions after assuming an underlying structure in data, which requires the presence of a teacher during the training stage of the system to obtain a reliable knowledge model. On the contrary, the latter aims to discover regular-occurring patterns beneath the data without making any *a priori* assumptions concerning their underlying structure. Nevertheless, some modern problems in data mining have failed to fully fit into one of these paradigms. In fact, constructing a predictive model from a pure supervised way in real-world domains is often unfeasible due to (1) the dearth of training examples and (2) the costs of labelling the required information to train the system. In addition, the unsupervised paradigm does not take into account the particular characteristics of the problem domain, thus it cannot exploit the search guidance that uses the supervised approach. This issue makes pure unsupervised learners prone to fail at recognising the interesting patterns—i.e., those that are uncommon and valuable—from the uninteresting ones. This situation has driven practitioners to explore semi-supervised learning, which consists on combining both approaches to overcome their individual limitations. Semi-supervised learning exploits the unsupervised strategy to obtain accurate predictive models from a reduced set of previously labeled (i.e., supervised) instances, which minimises the costs associated to obtaining a reliable and fully mapped training set from real-world domains. In this sense, the algorithm first trains the system with a reduced set of labeled examples to obtain a preliminary *protomodel*, which will be used to label the vast amount of remaining data—this strategy is referred to as *self-training* (Huynh and Hua, 2012). Then, the final model obtained by the learner is used for future predictions. So far, this strategy has been successfully applied in a variety of challenging domains such as artificial olfaction (De Vito et al., 2012), gene classification (Huang and Feng, 2012), protein prediction (Wang et al., 2012), and non-invasive diagnosis of Scoliosis (Seoud et al., 2010), which supports its effectiveness. An appealing framework for semi-supervised learning lies in the Michigan-style LCS approach, which in turn may take benefit from its inherent online architecture. Recently, the first steps towards this direction have been taken in (Navarro et al., 2013a).

Explore graphical models for interpretability enhancement. A requisite of association streams lies in the readability of the results obtained by the learner: in this area readable models are preferred over complex and more accurate ones. Due to this issue we designed an algorithm that uses fuzzy logic; a knowledge representation that is close to human reasoning (Cordón et al., 2001) and that allows a linguistic representation for the creation of highly legible models from both qualitative and quantitative data. However, we do not always succeed: in certain problems we may have a large number of overlapping rules that hinder the readability of the output. This is a major issue in some critical areas such as decision support systems for medicine, where the readability is of the utmost importance. For this reason Pancho et al. (2013) proposed a visual representation of fuzzy rules based on graphical models that enhances the comprehensibility of the knowledge mined. We want to integrate these graphical models into our framework, adapting them to the problem of association streams when concept drift occurs.

Exploit the online capacities of Michigan-style LCSs for Big Data. Nowadays we live in the era of Big Data: private and public sector domains increase year after year the volume of data to process, search and store (Joseph and Johnson, 2013). Ambitious projects such as the Human Genome Project or the Large Hadron Collider generate terabytes of information that must be handled and stored. It is in these environments where the challenge to find novel and useful information is accentuated. As a practical example it is considered the *Flow Cytometry* data repository (Aghaeepour et al., 2013), which has the following characteristics: (1) these data contain a huge quantity of instances, (2) these data have a low-moderate dimensionality and (3) these data are not labeled. Interestingly, these data are used by biologists and health staff making clusters by grouping features *visually* using their expertise in the field—the expert decides what values of the features should be in the same set—, with the cost it entails. In general, the challenge of mining Big Data require of learning algorithms that meet the two following characteristics (Wu et al., 2014): (1) efficiency at processing large quantities of instances and (2) parallelisation. Is in that scenario that Michigan-style LCSs can be exploited since this family meet the aforesaid requirements. Recently, a pioneering study showed the feasibility of applying LCSs (and other GBML techniques) for mining large-scale problems (Bacardit and Llorà, 2013). Moreover, the problem of mining Big Data is strongly related to the one of mining data streams in the sense that in both problems it is mandatory to process large volumes of data. In this regard, a future work line will be to integrate the Michigan-style LCS architecture with the existing Big Data technologies—e.g., *Hadoop*, *MapReduce*—taking advantage of the competitive algorithms studied in this thesis.

A

Statistical Comparisons of Learning Algorithms

Statistical validation has become an important process in the machine learning community to compare the results of distinct learning algorithms at a particular task. Several methodologies can be applied in order to statistically verify the hypothesis of improved performance. These are classified in parametric and non-parametric tests. The difference is that while the first group assumes a fixed distribution the second does not. In this thesis we used non-parametric tests following the recommendations of Demšar (2006) and García and Herrera (2008) for both pairwise and multiple comparisons of distinct algorithms. However, we did not give the details of such hypothesis testing. In this appendix we describe the procedures used for comparing distinct algorithms. First, in section A.1 we introduce the fundamental concepts of the statistical analysis performed in the machine learning community for hypothesis testing. Afterwards, in section A.2 we detail the methodology for comparing pairs of learning algorithms. Finally, in section A.3 we describe the methodology for comparing multiple techniques at once.

A.1 Essential Concepts

The problem of comparing two (or more) learning algorithms on a set of problems is not trivial. The first thing to do is to select a reliable estimate of how well the algorithm performs. There are many measures for quality estimation; for example in this thesis we used the 0/1 test accuracy, the Kohen's Kappa statistic, and the F-measure for classification tasks and support and confidence for mining association rules. Each measure of the performance is obtained for each data set and for each algorithm to be compared, thus yielding large tables of numbers. These are the tables we use to compare how well the distinct learning algorithms perform and to statistically verify the hypothesis that our algorithm performs better on average. More precisely, we are interested in the following hypothesis (Orriols-Puig, 2008):

- On average, algorithm A_1 outperforms algorithm A_2—for pairwise comparisons.
- On average, algorithm A_1 is the best of the comparisons, surpassing the results of algorithms A_2, A_3, A_4 and A_5—for multiple comparisons.

For this purpose, some essential concepts are detailed in the following.

- **The null hypothesis (H_0).** The hypothesis we want to prove false using, in our case, a statistical test; typically that all the learning algorithms perform the same on average.
- **The alternative hypothesis (H_A).** The opposite hypothesis to the null hypothesis. In our particular case that not all the learning algorithms perform the same on average.
- **Type I error or a false positive.** Rejecting the null hypothesis (incorrectly) when is actually true.
- **Type II error or a false negative.** Conversely to type I error, not rejecting the null hypothesis when is actually false.
- **The level of significance (α).** It tells us whether we found a true pattern in data or not (i.e., that it was just chance). For example that algorithm A_1 is better, on average, than algorithm A_2 for the given set of problems, with a certain probability of avoiding both type I error and type II error. This probability is coined as α. It identifies the level of significance. The larger this value, the more chance of committing type II error, and also we have less statistical power. Conversely, the lower this value, the more chance of committing type I error. Typical values are 0.05 and 0.1.
- **The computed p-value.** The p-value is the smallest level of significance that results in the rejection of the null hypothesis. This is a key concept, because if a test of significance gives a computed p-value lower than or equal to the significance level α, the null hypothesis is rejected.

Table A.1 summarises the relations between truth/falseness of H_0 and the outcomes of the test. In this table, columns represent the reality and rows the hypothesis.

	H_0 is true	**H_A is true**
Reject H_A	Accurate	**Type II Error**
Reject H_0	**Type I Error**	Accurate

Table A.1: Relations between truth/falseness of H_0 and the outcomes of the test.

In the forthcoming sections we detail the methodology used to compare pairs of algorithms and multiple algorithms. For all the statistical test, we provide a general description of which type of H_0 the statistical tests checks.

A.2 Pairwise Comparisons: The Wilcoxon Signed-Ranks Test

The Wilcoxon signed-ranks test (Wilcoxon, 1945) is a non-parametric test used for comparing two learning algorithms. According to Demšar (2006), it ranks the differences in performances of two learning algorithms for each data set, ignoring the signs, and compares the ranks for the positive and the negative differences.

Data Set	$\mathbf{A}_1$	$\mathbf{A}_2$	δ	**Rank**
app	0.910	0.787	-0.123	15
bpa	0.919	0.748	-0.171	18
fam	0.914	0.872	-0.042	5
gls	0.914	0.865	-0.049	6
h-s	0.896	0.871	-0.025	3
irs	0.906	0.729	-0.177	19
mag	0.913	0.796	-0.117	12
μca	0.923	0.930	0.007	2
pho	0.895	0.693	-0.202	20
pim	0.915	0.807	-0.108	9
rng	0.920	0.848	-0.072	7
seg	0.913	0.911	-0.002	1
son	0.898	0.779	-0.119	13
thy	0.935	0.826	-0.109	10
veh	0.914	0.799	-0.115	11
wav	0.891	0.761	-0.130	16
wdbc	0.929	0.823	-0.106	8
wne	0.896	0.728	-0.168	17
wpbc	0.917	0.796	-0.121	14
yst	0.926	0.897	-0.029	4

Table A.2: Comparison of the performance of the learning algorithms A_1 and A_2 over 20 data sets. For each data set, δ is the difference between A_2 and A_1.

The first step of this procedure is to compute the differences of the performance measures obtained by each algorithm in the N distinct data sets used. Next, the differences are ranked according to their absolute values, considering that the smallest difference holds the first position of the ranking and the largest difference gets the last position of the ranking. In case of ties, the average ranks are assigned. Afterwards, the procedure computes the sum of ranks for the data sets on which A_2 outperformed A_1 (referred to as R^+, and its opposite: the sum of ranks for the data sets on which A_1 outperformed A_2 (referred to as R^-). Ranks for which the difference is zero are split evenly among R^+ and R^-; if there is an odd number of them, one is ignored.

$$R^+ = \sum_{\delta_i > 0} \text{rank}(\delta_i) + \frac{1}{2}\sum_{\delta_i = 0} \text{rank}(\delta_i) \quad R^- = \sum_{\delta_i < 0} \text{rank}(\delta_i) + \frac{1}{2}\sum_{\delta_i = 0} \text{rank}(\delta_i), \tag{A.1}$$

where δ_i is the difference in performance obtained by each algorithm in the i-th data set. Finally, the smaller value between R^+ and R^- is set as T, that is:

$$T = \min(R^+, R^-). \tag{A.2}$$

To obtain the critical value which may reject H_0 we have to look for a table of critical T values for Wilcoxon's signed-ranks test—e.g., Table A5 in (Sheskin, 2004). However, we can approximate a normal distribution by computing the z statistic as follows:

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}}. \tag{A.3}$$

For a significance value of $\alpha = 0.05$, the null hypothesis can be rejected if z is smaller than -1.96 (see Table A1 in (Sheskin, 2004) for obtaining this value). With $\alpha = 0.1$, H_0 can be rejected if z is smaller than -1.65. The computed p-value can be extracted from the table of the normal distribution.

In the following we exemplify the procedure by applying the statistical test on the results of Table A.2, in which the performance of two algorithms A_1 and A_2 are compared on a collection of 20 data sets. First, we compute the differences, which are in the fourth column of Table A.2. Next, we compute R^+ and R^-. In particular we found that $R^+ = 2$ and $R^- = 208$. Then, we select T as the minimum value between $R^+ = 2$ and $R^- = 208$; that is $T = 2$. Finally, replacing $T = 2$ and $N = 20$ into Equation A.3 we obtain the statistic $z = -3.845$. According to the table of exact critical values for the normal distribution, for a confidence level of $\alpha = 0.05$ $z < -1.96$. As we obtained $z = -3.845$ we therefore reject H_0; that is, we reject the hypothesis that both algorithms perform the same on average, with $\alpha = 0.05$. Consulting the table of the normal distribution, we can compute the exact p-value, which results in p-value $= 1.204 \cdot 10^{-4}$.

A.3 Multiple Comparisons

Most often we are interested in comparing the performances of multiple learning algorithms. This can be done in two possible ways: (1) by means of multiple pairwise comparisons or (2) by using a multiple comparison procedure. According to Demšar (2006), the first methodology has the drawback that when so many tests are made, a certain proportion of the H_0 is rejected due to random chance. In this regard, it is preferable to first apply a multiple comparison test to analyse whether all the algorithms performed the same on average (Orriols-Puig, 2008). If this is the case, no further actions can be taken. Otherwise, different post-hoc tests can be applied. In the following we first describe the Friedman test and then the post-hoc Nemenyi and Holm procedures.

A.3.1 The Friedman Test

The Friedman test (Friedman, 1937) is a non-parametric test used for comparing multiple learning algorithms. It does so by checking the H_0 of whether all algorithms perform the same on average (Orriols-Puig, 2008). This procedure ranks the algorithms for each data set separately, the best performing algorithm getting the rank of 1, the second best rank 2, an so on. In case of ties, average ranks are assigned. Then, the procedure computes the average rank R_i of each algorithm i as follows, where N is the total number of data sets:

$$R_i = \frac{1}{N}\sum_j \text{rank}_j^i. \tag{A.4}$$

Data Set	**A**$_1$	**A**$_2$	**A**$_3$	**A**$_4$	**A**$_5$	**A**$_6$	**A**$_7$
ann	0.9850 (3.0)	0.9725 (4.0)	0.8840 (7.0)	0.9710 (5.0)	0.8940 (6.0)	0.9875 (2.0)	0.9941 (1.0)
aut	0.8405 (1.0)	0.6695 (5.0)	0.5545 (6.0)	0.6895 (4.0)	0.3260 (7.0)	0.7830 (2.0)	0.7285 (3.0)
ban	0.6915 (2.0)	0.6185 (5.0)	0.6140 (6.0)	0.6275 (4.0)	0.4240 (7.0)	0.6845 (3.0)	0.6970 (1.0)
bpa	0.6585 (3.0)	0.6250 (4.0)	0.5425 (5.0)	0.4285 (6.0)	0.4250 (7.0)	0.6940 (2.0)	0.7608 (1.0)
col	0.8500 (1.0)	0.8165 (4.0)	0.7885 (6.0)	0.8305 (3.0)	0.8355 (2.0)	0.8105 (5.0)	0.7711 (7.0)
gls	0.6645 (4.0)	0.6940 (1.0)	0.4555 (6.0)	0.5245 (5.0)	0.1860 (7.0)	0.6680 (3.0)	0.6829 (2.0)
h-c	0.7405 (7.0)	0.8185 (5.0)	0.8315 (1.0)	0.8290 (2.0)	0.8235 (4.0)	0.8255 (3.0)	0.8068 (6.0)
h-s	0.7880 (6.0)	0.7835 (7.0)	0.8385 (1.0)	0.8345 (2.0)	0.8280 (3.0)	0.8035 (4.0)	0.7890 (5.0)
hov	0.9655 (1.0)	0.9245 (6.0)	0.9030 (7.0)	0.9590 (2.0)	0.9485 (4.0)	0.9425 (5.0)	0.9547 (3.0)
ion	0.9105 (2.5)	0.8490 (5.0)	0.8260 (6.0)	0.8835 (4.0)	0.7175 (7.0)	0.9105 (2.5)	0.9183 (1.0)
irs	0.9470 (6.0)	0.9530 (3.5)	0.9530 (3.5)	0.9670 (2.0)	0.9265 (7.0)	0.9735 (1.0)	0.9529 (5.0)
k-p	0.9945 (1.0)	0.9670 (4.0)	0.8780 (7.0)	0.9575 (5.0)	0.9140 (6.0)	0.9930 (2.0)	0.9820 (3.0)
lab	0.7505 (6.0)	0.9475 (1.0)	0.9135 (3.0)	0.9470 (2.0)	0.5110 (7.0)	0.8600 (4.0)	0.8219 (5.0)
lym	0.7795 (7.0)	0.8110 (5.0)	0.8365 (3.0)	0.8705 (1.0)	0.7920 (6.0)	0.8300 (4.0)	0.8521 (2.0)
mam	0.8340 (1.0)	0.7670 (7.0)	0.8205 (2.0)	0.7950 (6.0)	0.7960 (5.0)	0.8085 (3.0)	0.8008 (4.0)
μca	0.6185 (6.0)	0.6560 (1.0)	0.6485 (4.0)	0.6540 (3.0)	0.5495 (7.0)	0.6480 (5.0)	0.6547 (2.0)
mnk	0.8840 (2.0)	0.8030 (3.0)	0.5385 (7.0)	0.5390 (5.5)	0.5390 (5.5)	0.7955 (4.0)	0.9198 (1.0)
msm	1.0000 (2.5)	1.0000 (2.5)	0.9570 (7.0)	1.0000 (2.5)	0.9990 (6.0)	1.0000 (2.5)	0.9997 (5.0)
pim	0.7325 (5.0)	0.7355 (4.0)	0.7580 (1.5)	0.7580 (1.5)	0.5130 (7.0)	0.7515 (3.0)	0.7079 (6.0)
pri	0.3745 (6.0)	0.4030 (4.0)	0.4630 (2.0)	0.4230 (3.0)	0.0980 (7.0)	0.3835 (5.0)	0.5544 (1.0)
son	0.7190 (5.0)	0.8305 (2.0)	0.6740 (6.0)	0.7670 (4.0)	0.6395 (7.0)	0.8295 (3.0)	0.8422 (1.0)
tao	0.9545 (2.0)	0.9640 (1.0)	0.8080 (7.0)	0.8395 (5.0)	0.8365 (6.0)	0.8615 (3.0)	0.8506 (4.0)
thy	0.9255 (5.0)	0.9500 (4.0)	0.9720 (1.0)	0.8900 (6.0)	0.5730 (7.0)	0.9630 (2.0)	0.9547 (3.0)
veh	0.7150 (4.0)	0.7035 (5.0)	0.4285 (6.0)	0.7290 (3.0)	0.3280 (7.0)	0.8120 (2.0)	0.8318 (1.0)
vot	0.9655 (1.0)	0.9305 (6.0)	0.9030 (7.0)	0.9600 (2.0)	0.9470 (3.0)	0.9415 (4.0)	0.9349 (5.0)
vow	0.7975 (4.0)	0.9675 (1.0)	0.6400 (6.0)	0.7100 (5.0)	0.3230 (7.0)	0.9290 (2.0)	0.9049 (3.0)
wav	0.7540 (7.0)	0.7775 (6.0)	0.7875 (5.0)	0.8645 (1.0)	0.8530 (3.0)	0.8300 (4.0)	0.8599 (2.0)
wbcd	0.9535 (6.0)	0.9680 (1.5)	0.9610 (3.0)	0.9680 (1.5)	0.9605 (4.0)	0.9500 (7.0)	0.9586 (5.0)
wne	0.9380 (6.0)	0.9600 (5.0)	0.9835 (2.0)	0.9915 (1.0)	0.2565 (7.0)	0.9770 (3.0)	0.9753 (4.0)
zoo	0.9290 (5.0)	0.9285 (6.0)	0.9470 (4.0)	0.9580 (1.5)	0.6415 (7.0)	0.9580 (1.5)	0.9509 (3.0)
Avg	3.933	3.950	4.600	3.283	5.850	3.217	3.167
Pos	4	5	6	3	7	2	1

Table A.3: Comparison of the performance of algorithms A_1, A_3, A_4, A_5, A_6, and A_7 over 30 data sets. For each algorithm and data set the average rank is supplied in parentheses. The lasts two rows show (1) the average rank of each learning algorithm and (2) the Friedman ranking.

Afterwards, the Friedman statistic is computed as:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_i R_i^2 - \frac{1}{4}k(k+1)^2\right], \tag{A.5}$$

where k is the number of algorithms in the comparison. The Friedman statistic is distributed according to χ_F^2 with $k-1$ degrees of freedom when N and k are big enough (typically $N > 10$

and $k > 5$), although for a smaller number of algorithms and data sets, exact critical values have been computed (Demšar, 2006).

Due to the fact that the Friedman statistic is conservative, Iman and Davenport (1980) derived a better statistic:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2}, \tag{A.6}$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom (Demšar, 2006).

In the following we exemplify the procedure by applying the statistical test on the results of Table A.3, in which the performance of the learning algorithms A_1, A_3, A_4, A_5, A_6, and A_7 are compared over a collection of 30 data sets. Table A.3 shows the rank of each learning algorithm for every data set—supplied in parentheses—and the average ranks and the final Friedman ranking for the distinct algorithms. If we substitute $N = 30$ and $k = 7$ in Equation A.5 we obtain the Friedman statistic considering the reduction in performance (distributed according to the χ^2-distribution with six degrees of freedom): $\chi_F^2 = 36.071$.

If we plug $\chi_F^2 = 36.071$, $N = 30$ and $k = 7$ in Equation A.6 we obtain the Iman and Davenport statistic considering the reduction in performance (distributed according to the F-distribution with six and 174 degrees of freedom): $F_F = 7.268$.

For an $\alpha = 0.05$ the computed p-value is $6.1097 \cdot 10^{-7}$. Because the p-value is much less than α we therefore reject H_0; that is, there are significative differences between the learning algorithms. Recall that the computed p-value is extracted from the F-distribution table.

When the Friedman procedure rejects H_0, a post-hoc test is applied to detect further differences. There are many distinct post-hoc tests and all of them are applied to detect furthers differences among algorithms. In the following we detail the post-hoc Nemenyi and the Holm-Shaffer procedures.

A.3.2 Post-hoc Nemenyi Test

The post-hoc Nemenyi test (Nemenyi, 1964) is used when all learning algorithms are compared to each other. The performance of two algorithms is significantly different if the corresponding average ranks differ by at least the critical difference CD, computed as:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{A.7}$$

where q_α is the critical value based on the Studentized range statistic (Orriols-Puig, 2008)—Table A.4 shows the critical values for $\alpha = 0.05$ and for $\alpha = 0.1$ and from $k = 2$ to $k = 10$—, N is the number of data sets and k is the number of algorithms.

k	2	3	4	5	6	7	8	9	10
$\mathbf{q_{0.05}}$	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164
$\mathbf{q_{0.10}}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

TABLE A.4: Critical values for the two-tailed Nemenyi test for $\alpha = 0.05$ and for $\alpha = 0.1$. k is the number of learning algorithms in the comparison. These values have been taken from (Demšar, 2006).

In the following we exemplify the procedure by applying the statistical test on the results of the aforementioned Friendman test example (see Table A.3), in which the performance of the learning algorithms A_1, A_3, A_4, A_5, A_6, and A_7 are compared over a collection of 30 data sets. Friendman test rejected H_0 that the algorithms perform the same on average, therefore we check whether there are statistically significant differences, on average, among learning algorithms by computing CD for $\alpha = 0.05$: $CD_{\alpha=0.05} = 1.645$. Next, we obtain the maximum and the minimum ranks from the Friedman ranking (Table A.3)—that is: $maxRank = 5.58$ and $minRank = 3.167$—and we subtract them. If this value is greater or equal to CD_α, there are significative differences and we can proceed with the rest of the procedure. In our example we have $maxRank - minRank = 2.413 > CD_{\alpha=0.05} = 1.645$, therefore we proceed to compute the pairwise differences between algorithms to identify the differences. These are shown in Table A.5.

	$\mathbf{A_1}$	$\mathbf{A_2}$	$\mathbf{A_3}$	$\mathbf{A_4}$	$\mathbf{A_5}$	$\mathbf{A_6}$	$\mathbf{A_7}$
$\mathbf{A_1}$	-	1.25	1.90	1.92	2.57	2.63	2.68
$\mathbf{A_2}$	1.25	-	0.65	0.67	1.32	1.38	1.43
$\mathbf{A_3}$	1.90	0.65	-	0.02	0.67	0.73	0.78
$\mathbf{A_4}$	1.92	0.67	0.02	-	0.65	0.72	0.77
$\mathbf{A_5}$	2.57	1.32	0.67	0.65	-	0.07	0.12
$\mathbf{A_6}$	2.63	1.38	0.73	0.72	0.07	-	0.05
$\mathbf{A_7}$	2.68	1.43	0.78	0.77	0.12	0.05	-

Table A.5: Differences between the average Friedman rankings of the learning algorithms.

Those pair of algorithms whose difference is greater than CD_α have a significative difference. Figure A.1 show each learning algorithm according to its rank. Groups of algorithms that are not significantly different at $\alpha = 0.05$ are connected.

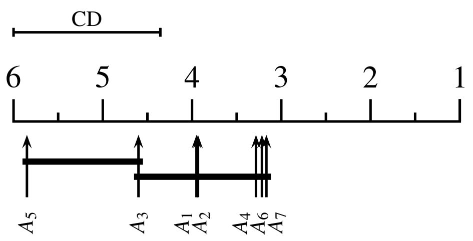


Figure A.1: Comparison of the performance of all learning algorithms against each other with the Nemenyi test. Groups of algorithms that are not significantly different at $\alpha = 0.05$ are connected.

A.3.3 Holm's Procedure

As we saw with the Nemenyi test, once Friedman's test rejects H_0, we can proceed with a post-hoc test in order to find the concrete pairwise comparisons which produce differences (García and Herrera, 2008). However, other post-hoc more powerful and sophisticated procedures exist. These are focused on controlling the family-wise error when comparing with a control classifier.

i	Algorithms	z	p	p-Holm	p-Shaffer
21	$\mathbf{A_5}$ **vs.** $\mathbf{A_7}$	4.8107	$1.5033 \cdot 10^{-6}$	0.0024	0.0024
20	$\mathbf{A_5}$ **vs.** $\mathbf{A_6}$	4.7211	$2.3451 \cdot 10^{-6}$	0.0025	0.0033
19	$\mathbf{A_4}$ **vs.** $\mathbf{A_5}$	4.6016	$4.1920 \cdot 10^{-6}$	0.0026	0.0033
18	$\mathbf{A_1}$ **vs.** $\mathbf{A_5}$	3.4363	$5.8976 \cdot 10^{-4}$	0.0028	0.0033
17	$\mathbf{A_2}$ **vs.** $\mathbf{A_5}$	3.4060	$6.5825 \cdot 10^{-4}$	0.0029	0.0033
16	A_3 vs. A_7	2.5697	0.0102	0.0031	0.0033
15	A_3 vs. A_6	2.4801	0.0131	0.0033	0.0033
14	A_3 vs. A_4	2.3606	0.0182	0.0036	0.0036
13	A_3 vs. A_5	2.2410	0.0250	0.0038	0.0038
12	A_2 vs. A_7	1.4044	0.1602	0.0042	0.0042
11	A_1 vs. A_7	1.3745	0.1693	0.0045	0.0045
10	A_2 vs. A_6	1.3147	0.1886	0.0050	0.0050
9	A_1 vs. A_6	1.2849	0.1988	0.0055	0.0055
8	A_1 vs. A_3	1.1952	0.2320	0.0062	0.0062
7	A_2 vs. A_4	1.1953	0.2320	0.0071	0.0071
6	A_2 vs. A_3	1.1653	0.2439	0.0083	0.0083
5	A_1 vs. A_4	1.1653	0.2439	0.0100	0.0100
4	A_4 vs. A_7	0.2092	0.8343	0.0125	0.0125
3	A_4 vs. A_6	0.1195	0.9049	0.0167	0.0167
2	A_6 vs. A_7	0.0896	0.9286	0.0250	0.0250
1	A_1 vs. A_2	0.0299	0.9762	0.0500	0.0500

TABLE A.6: Holm / Shaffer Table for $\alpha = 0.05$. Algorithms that perform significantly different according to both Holm's and Shaffer's procedures are marked in bold.

This is the case of the Holm's procedure (Holm, 1979), and for this purpose the z test statistic for comparing the i-th and j-th learning algorithm have to be computed first as follows:

$$z = \frac{R_i - Rj}{\sqrt{\frac{k(k+1)}{6N}}}, \tag{A.8}$$

where R_i is the average Friedman rank for the i-th learning algorithm. It is worth noting that z approximates the normal distribution.

The Holm's procedure adjusts the value of α in a step down method; that is, if we have a set of ordered p-values $p_1 \leq p_2 \leq \cdots \leq p_{k-1}$ and the corresponding hypothesis $H_1, H_2, \ldots, H_{k-1}$ we can sequentially test the hypotheses ordered by their significance. Starting with the most significant p-value, Holm's procedure **rejects** H_1 to H_{i-1} if i is the smallest integer such that $p_i > \alpha/(k - i + 1)$ (i.e., the Holm's p_i-value). As soon as a certain null hypothesis cannot be rejected, all the remaining hypotheses are retained as well (Demšar, 2006). The computed p-values are extracted from the table of the normal distribution.

Table A.6 exemplifies the Holm's procedure. Algorithms that perform significantly different according to the Holm's procedure are marked in bold. We need to check the better ranked technique in the Friedman ranking in order to know the algorithm that outperforms in each pairwise comparison—e.g, in i 21: A_5 vs. A_7 we observe a significant difference. Then, according tho the Friedman ranking (Table A.3) se see that algorithm A_5 has ranking 7 and that algorithm A_7 has ranking 1, thus we can conclude that A_7 performs significantly better than A_5. Notice that Table A.6 shows the same conclusions than the post-hoc Nemenyi test (Figure A.1).

Also, in the aforementioned analysis (see table A.6) we included the Shaffer procedure (Shaffer, 1986), which works in a similar way than Holm's.

B
Index Terms

Michigan-style LCSs are complex systems that have several internal variables that allow the system to solve machine learning problems. The purpose of this appendix is to list and describe the distinct internal variables for each one of the algorithms described in this thesis. First, SNCS internal variables are detailed. Following that, XCS$_{cds}$ internal variables are described. Finally, Fuzzy-CSar internal variables are shown.

B.1 SNCS Internal Variables

Short Name	**Full Name**	**Description**
$[P]$	Population	The stored population of solutions.
$[M]$	Match set	Those individuals that match with the incoming example.
$[S]_i$	Species set	The set of individuals that predict the same outcome i.
F	Fitness	The guidance signal used by SNCS for problem solving.
exp	Experience	The number of times an individual has been triggered.
t	Time stamp	The time stamp of the last occurrence of a genetic event.
w_{ij}	Weight$_{ij}$	A particular weight for an input signal.
acc	Accuracy	The average number of instances correctly classified.
k	Relative Accuracy	The relative accuracy of an individual.
n_{hid}	Neurons in the hidden layer	The number of neurons in the hidden layer.

Table B.1: The distinct internal variables used by SNCS.

B.2 XCS$_{cds}$ Internal Variables

Short Name	Full Name	Description
p_i	Lower allele	The lower allele of an interval using the unordered-bound representation.
q_i	Upper allele	The upper allele of an interval using the unordered-bound representation.
c_k	Cluster k	The cluster k predicted by a rule.
$[P]$	Population	The stored population of solutions.
$[M]$	Match set	Those individuals that match with the incoming example.
F	Fitness	The guidance signal used by XCS$_{cds}$ for problem solving.
ϵ	Error	Error of an individual.
num	Numerosity	The number of copies of an individual in the population.
σ	Match set size	The average size of the match sets in which the individual has participated.
exp	Experience	The number of times an individual has been triggered.
t	Time stamp	The time stamp of the last occurrence of a genetic event.
k	Relative Accuracy	The relative accuracy of an individual.
d	Deletion vote	The vote of an individual for deletion.

Table B.2: The distinct internal variables used by XCS$_{cds}$.

B.3 Fuzzy-CSar Internal Variables

Short Name	Full Name	Description
$[P]$	Population	The stored population of solutions.
$[M]$	Match set	Those individuals that match with the incoming example.
$[A]_i$	Association set	Those individuals that share the antecedent part.
sup	Support	The support of a rule.
con	Confidence	The confidence of a rule.
lif	Lift	The lift of a rule.
acc	Accuracy	The matching degree between the rule and the data.
F	Fitness	The guidance signal used by Fuzzy-CSar for problem solving.
num	Numerosity	The number of copies of an individual in the population.
as	Association set size	The average size of the association sets in which the individual has participated.
exp	Experience	The number of times an individual has been triggered.
$timeOfInd$	Time stamp	The time stamp of the last occurrence of a genetic event.
p_{sel}	Selection probability	The selection probability of an association set.
imp	Implication	The fuzzy implication of the rule.
ant_mat	Antecedent match	The estimated degree of matching of the antecedent part of a rule.
con_mat	Consequent match	The estimated degree of matching of the consequent part of a rule.
d	Deletion vote	The vote of an individual for deletion.

Table B.3: The distinct internal variables used by Fuzzy-CSar.

References

Hussein Abbass, Jaume Bacardit, Martin Butz, and Xavier Llorà. Online adaptation in learning classifier systems: Stream data mining. Technical report, Illinois Genetic Algorithms Laboratory, 104 S. Mathews Avenue, Urbana, IL, june 2004. 4, 35

Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. StreamKM++: A clustering algorithm for data streams. *Journal Experimental Algorithmics*, 17:2.4:2.1–2.4:2.30, may 2012. ISSN 1084-6654. 82

Charu C. Aggarwal, editor. *Data Streams - Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer, 2007. ISBN 978-0-387-28759-1. 3, 36

Charu C. Aggarwal and Philip S. Yu. A new approach to online generation of association rules. *Knowledge and Data Engineering, IEEE Transactions on*, 13(4):527–540, july 2001. ISSN 1041-4347. 98

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In Morgan Kaufmann, editor, *Proceedings of the international conference on very large data bases*, VLDB '03, pages 81–92, 2003. ISBN 0-12-722442-4. 82

Nima Aghaeepour, Greg Finak, Holger Hoos, Tim R. Mosmann, Ryan Brinkman, Raphael Gottardo, and Richard H. Scheuermann. Critical assessment of automated flow cytometry data analysis techniques. *Nature Methods*, 10(3):228–238, Jan 2013. ISSN 1548-7091. 146

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, BLDB*, Santiago, Chile, September 1994. 99, 101

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22, pages 207–216, Washington D. C., USA, June 1993. ACM. 99, 100

Jesús S. Aguilar-Ruiz, José C. Riquelme Santos, and Miguel Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 33(2):324–331, 2003. 21

David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991. ISSN 0885-6125. 37, 48, 66

Jesús Alcalá-Fdez, Rafael Alcalá, María José Gacto, and Francisco Herrera. Learning the membership function contexts for mining fuzzy association rules by using genetic algorithms. *Fuzzy Sets and Systems*, 160(7):905–921, 2008. ISSN 0165-0114. 99

Jesus Alcalá-Fdez, Luciano Sánchez, Salvador García, Manuel del Jesus, Sebastián Ventura, Josep M. Garrell, José Otero, Cristóbal Romero, Jaume Bacardit, Víctor M. Rivas, Juan C. Fernández, and Francisco Herrera. KEEL: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13:307–318, 2009. ISSN 1432-7643. 65

Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, and Salvador García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011. 124

Plamen Angelov. *Autonomous learning systems: From data streams to knowledge in real-time.* Wiley, first edition, 2012. ISBN 978-1-119-95152-0. 3, 12, 98

Plamen Angelov and Zhou Xiaowei. Evolving fuzzy-rule-based classifiers from data streams. *Fuzzy Systems, IEEE Transactions on*, 16(6):1462–1475, december 2008. ISSN 1063-6706. 38, 82, 105

Jaume Bacardit. *Pittsburgh genetic-based machine learning in the data mining era: Representations, generalisation, and run-time.* PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, October 2004. 2, 13, 15, 18, 20

Jaume Bacardit and Martin V. Butz. Data mining in learning classifier systems: comparing XCS with GAssist. In *Proceedings of the 2003-2005 international conference on Learning classifier systems*, IWLCS'03-05, pages 282–290, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-71230-5. 13

Jaume Bacardit and Xavier Llorà. Large-scale data mining using genetics-based machine learning. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 3(1):37–61, 2013. 19, 147

Stephen K. Bache and Moshe Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. xix, 11, 65

Rashmi Baruah and Plamen Angelov. Evolving local means method for clustering of streaming data. In *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, pages 1–8, june 2012a. 38, 105

Rashmi D. Baruah and Plamen Angelov. Evolving local means method for clustering of streaming data. In *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, june 2012b. 82, 83

Ester Bernadó-Mansilla and Josep M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, september 2003. ISSN 1063-6560. 4, 13, 31

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. MOA: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research (JMLR) Workshop and Conference Proceedings, Volume 11: Workshop on Applications of Pattern Analysis*, pages 44–50. Journal of Machine Learning Research, 2010. 4, 82, 89, 105

Abdelhamid Bouchachia, Edwin Lughofer, and Daniel Sanchez. Editorial of the special issue: Online fuzzy machine learning and data mining. *Information Sciences*, 220:1–4, january 2013. 105

Erick A. Brewer. Pushing the CAP: Strategies for consistency and availability. *IEEE Computer*, 45(2), 2012. 78, 79, 80, 93

Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: generalizing association rules to correlations. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, SIGMOD '97, pages 265–276, New York, USA, 1997. ACM. ISBN 0-89791-911-4. 101

Larry Bull. On using constructivism in neural classifier systems. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, PPSN VII, pages 558–567, London, UK, 2002. Springer-Verlag. ISBN 3-540-44139-5. 36, 37

Larry Bull and Toby O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911, New York, USA, 2002. Morgan Kaufmann. ISBN 1-55860-878-8. 27, 37, 43

Martin V. Butz. *Rule-based evolutionary online learning systems: Learning bounds, classification,and prediction*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 2004. 2, 131, 132

Martin V. Butz. *Rule-Based Evolutionary Online Learning Systems - A Principled Approach to LCS Analysis and Design*, volume 191 of *Studies in Fuzziness and Soft Computing*. Springer, 2006. ISBN 978-3-540-25379-2. xv, 1, 19, 23, 26, 28, 29, 30, 31, 38, 70, 88, 113, 131, 132, 133, 135, 136

Martin V. Butz, Tim Kovacs, Pier L. Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8 (1):28–46, 2004. xv, 29, 30, 31, 87, 132

Martin V. Butz, Kumara Sastry, and David E. Goldberg. Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, 6(1):53–77, march 2005. ISSN 1389-2576. 27, 43, 112

Martin V. Butz, David E. Goldberg, Pier L. Lanzi, and Kumara Sastry. Problem solution sustenance in XCS: Markov chain analysis of niche support distributions and the impact on computational complexity. *Genetic Programming and Evolvable Machines*, 8(1):5–37, 2007. ISSN 1389-2576. doi: 10.1007/s10710-006-9012-8. 138

Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *In 2006 SIAM conference on data mining*, pages 328–339, 2006. 82

Jorge Casillas and Francisco J. Martínez-López. Mining uncertain data with multiobjective genetic fuzzy systems to be applied in consumer behaviour modelling. *Expert Systems with Applications*, 36(2, Part 1):1645–1659, 2009. ISSN 0957-4174. 21, 105

B. Chandra and Shalini Bhaskar. A novel approach of finding frequent itemsets in high speed data streams. In *Eighth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD*, volume 1, pages 40–44, july 2011. 38, 99, 105

James Cheng, Yiping Ke, and Wilfred Ng. Maintaining frequent closed itemsets over a sliding window. *Journal of Intelligent Information Systems*, 31(3):191–215, December 2008. ISSN 0925-9902. 99

Yun Chi, Haixun Wang, P.S. Yu, and Richard R. Muntz. Moment: maintaining closed frequent itemsets over a stream sliding window. In *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, pages 59–66, 2004. 98

Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz, and Lukasz A. Kurgan. *Data Mining: a knowledge discovery approach.* Springer US, 1 edition, 2007. ISBN 978-0-387-33333-5. 99

Oscar Cordón, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. *Genetic fuzzy systems: Evolutionary tuning and learning of fuzzy knowledge bases*, volume 19 of *Advances in Fuzzy Systems–Applications and Theory.* World Scientific Publishing Co. Pte. Ltd., first edition, 2001. ISBN 981-02-4016-3. 102, 146

Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, february 2010. ISSN 1066-8888. 99

Guiomar Corral, Alvaro García-Piquer, Albert Orriols-Puig, Albert Fornells, and Elisabet Golobardes. Analysis of vulnerability assessment results based on caos. *Applied Soft Computing*, 11(7):4321–4331, 2011. 98

Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism: A workload-driven approach to database replication and partitioning. *PVLDB*, 3(1), 2010. 79, 80

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989. ISSN 0932-4194. 39

Charles Darwin. *On the Origin of species by means of natural selection, or the preservation of favored races in the struggle for life.* John Murray, London, 1859. 1, 14

Sudipto Das, Shoji Nishimura, Divyakant Agrawal, and Amr E Abbadi. Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. *PVLDB*, 4(8), 2011. 81, 94

Saverio De Vito, Grazia Fattoruso, Matteo Pardo, Francesco Tortorella, and Girolamo Di Francia. Semi-supervised learning techniques in artificial olfaction: A novel approach to classification problems and drift counteraction. *Sensors Journal, IEEE*, 12(11):3215–3224, 2012. ISSN 1530-437X. 146

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. xxi, 49, 127, 149, 151, 152, 154, 156

Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011. 49

Mahmood Deypir and Mohammad Hadi Sadreddini. A dynamic layout of sliding window for frequent itemset mining over data streams. *Journal of Systems and Software*, 85(3): 746–759, 2012. 99

Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998. 68

Didier Dubois, Eyke Hüllermeier, and Henri Prade. A systematic approach to the assessment of fuzzy association rules. *Data Mining and Knowledge Discovery*, 13(2):167–192, october 2006. ISSN 1573-756X. 99, 103

Wei Fan. Systematic data selection to mine concept-drifting data streams. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 128–137, New York, USA, 2004. ACM. ISBN 1-58113-888-1. 46, 114

Wei Fan, Toyohide Watanabe, and Koichi Asakura. Ratio rules mining in concept drifting data streams. In *Proceedings of the World Congress on Engineering and Computer Science 2009 Vol II, WCECS 2009*, San Francisco, USA, october 2009. ISBN 978-988-18210-2-7. 99, 107

Zahra Farzanyar, Mohammadreza Kangavari, and Nick Cercone. Max-fism: Mining (recently) maximal frequent itemsets over data streams using the sliding window model. *Computers & Mathematics with Applications*, 64(6):1706–1718, 2012. 99

Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, september 1987. ISSN 0885-6125. 82

Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley & Sons, New York, USA, 1966. 14

Albert Fornells. *Marc integrador de les capacitats de soft-computing i de knowledge discovery dels mapes autoorganitzatius en el raonament basat en casos.* PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, february 2006. 86

María A. Franco, Natalio Krasnogor, and Jaume Bacardit. GAssist vs. BioHEL: Critical assessment of two paradigms of genetics-based machine learning. *Soft Computing*, 17(6): 953–981, 2013. 21

Alex A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. ISBN 3540433317. 14, 15

Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. ISSN 01621459. 48, 49, 152

Douglas J. Futuyma. *Evolution.* Sunderland, Massachusetts: Sinauer Associates, Inc., 2005. ISBN 0-87893-187-2. 14

João Gama. A survey on learning from data streams: Current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, 2012. ISSN 2192-6352. 4, 12, 36, 82, 106

João Gama, editor. *Knowledge Discovery from Data Streams.* Advances in Database Systems. Chapman and Hall/CRC, first edition, may 2010. ISBN 978-1439826119. 3, 12, 13, 36, 98, 105, 142

João Gama and Mohamed M. Gaber, editors. *Learning from data streams: processing techniques in sensor networks.* Springer, first edition, December 2007. ISBN 3540736786. 12, 36

João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence*, Lecture Notes in Computer Science, pages 286–295. Springer Verlag, September 2004. ISBN 3-540-23237-0. 122

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013. 36

Salvador García and Francisco Herrera. An extension on "Statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, December 2008. 49, 149, 155

Álvaro García-Piquer. *Facing-up challenges of multiobjective clustering based on evolutionary algorithms: Representations, scalability and retrieval solutions.* PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, december 2012. 79

Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveis*, 38(3), september 2006. ISSN 0360-0300. 101

David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley Longman, Inc., Alabama, USA, 1989. ISBN 0-201-15767-5. 14, 15, 18, 19, 99, 110

David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 1402070985. 15, 16, 17, 18, 37, 87, 99, 131, 135, 139

David Perry Greene and Stephen F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13(2-3):229–257, November 1993. ISSN 0885-6125. 20, 21, 105

Vincenzo Gulisano, Ricardo Jiménez-Peris, Marta Patiño-Martínez, Claudio Soriente, and Patrick Valduriez. Streamcloud: An elastic and scalable data streaming system. *IEEE Transactions on Parallel Distributed Systems*, 23(12), 2012. 80

Vehbi C. Gungor, Dilan Sahin, Taskin Kocak, Salih Ergüt, Concettina Buccella, Carlo Cecati, and Gerhard P. Hancke. Smart grid technologies: Communication technologies and standards. *IEEE Transactions on Industrial Informatics*, 7(4), 2011. 78

Vehbi Cagri Gungor, Dilan Sahin, Taskin Kocak, Salih Ergüt, Concettina Buccella, Carlo Cecati, and Gerhard P. Hancke. A survey on smart grid potential applications and communication requirements. *IEEE Transactions on Industrial Informatics*, 9(1), 2013. 78, 80, 81

Brian K. Hall and Benedikt Hallgrímason. *Strickberger's Evolution*. Jones & Bartlett Learning, 4 edition, 2008. ISBN 0763700665. 14

Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, january 2004. ISSN 1384-5810. 104

Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2 edition, January 2006. ISBN 1558609016. 99

Michael Bonnell Harries, Claude Sammut, and Kim Horn. Extracting hidden context. *Machine Learning*, 32(2):101–126, august 1998. ISSN 0885-6125. 122

Francisco Herrera, Manuel Lozano, and José L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4): 265–319, 1998. 15

Chandima HewaNadungodage, Yuni Xia, John Jaehwan Lee, and Yi Cheng Tu. Hyper-structure mining of frequent patterns in uncertain data streams. *Knowledge and Information Systems*, 37(1):219–244, 2013. 107

John H. Holland. Processing and processors for schemata. In E. L. Jacks, editor, *Associative Information Techniques*, pages 127–146, New York, 1971. American Elsevier. 13, 14

John H. Holland. Adaptation. *Progress in Theoretical Biology*, 4:263–293, 1976. 2, 13, 14, 15, 16, 19

John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, second edition, 1992. ISBN 9780262581110. 1, 13, 36, 37, 99

John H. Holland and Judith S. Reitman. Cognitive systems based on adaptive algorithms. *SIGART Bull.*, 1(63):49–49, june 1977. ISSN 0163-5719. 2, 13, 20

John H. Holland, Lashon B. Booker, Marco Colombetti, Marco Dorigo, David E. Goldberg, Stephanie Forrest, Rick L. Riolo, Robert E. Smith, Pier L. Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. What is a learning classifier system? In Pier L. Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems*, volume 1813 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 1999. ISBN 3-540-67729-1. 13, 19

Soren Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979. 49, 156

Tzung-Pei Hong, Chan-Sheng Kuo, and Sheng-Chai Chi. Trade-off between computation time and number of rules for fuzzy mining from quantitative data. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(5):587–604, 2001. 99, 100, 104, 125

Tzung-Pei Hong, Chun-Hao Chen, Yeong-Chyi Lee, and Yu-Lung Wu. Genetic-fuzzy data mining with divide-and-conquer strategy. *Evolutionary Computation, IEEE Transactions on*, 12(2):252–265, april 2008. ISSN 1089-778X. 99

Gerard Howard, Larry Bull, and Pier L. Lanzi. Towards continuous actions in continuous space and time using self-adaptive constructivism in neural XCSF. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1219–1226, New York, USA, 2009. ACM. ISBN 978-1-60558-325-9. 37, 42

Gerard Howard, Larry Bull, and Pier L. Lanzi. A spiking neural representation for XCSF. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, july 2010. 37

Hong Huang and Hailiang Feng. Gene classification using parameter-free semi-supervised manifold learning. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 9(3):818–827, 2012. ISSN 1545-5963. 146

Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams, 2003. URL `http://www.cs.washington.edu/dm/vfml/`. 48

Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *2001 ACM SIGKDD international conference on knowledge discovery and data mining*, pages 97–106, 2001. 37, 45, 46, 48, 115

Dat T. Huynh and Wen Hua. Self-supervised learning approach for extracting citation information on the web. In Quan Z. Sheng, Guoren Wang, ChristianS. Jensen, and Guandong Xu, editors, *Web Technologies and Applications*, volume 7235 of *Lecture Notes in Computer Science*, pages 719–726. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29252-1. 146

Ronald L. Iman and James M. Davenport. Approximations of the critical region of the friedman statistic. *Communications in Statistics*, 9(6):571–595, 1980. 154

Hisao Ishibuchi, Yutaka Kaisho, and Yusuke Nojima. Complexity, interpretability and explanation capability of fuzzy rule-based classifiers. In *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on*, pages 1730–1735, 2009. 108

George John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995. 37, 48, 66

Rhoda. C. Joseph and Norman A. Johnson. Big Data and transformational government. *IT Professional*, 15(6):43–48, Nov 2013. ISSN 1520-9202. 146

Latifur Khan. Data stream mining: Challenges and techniques. In *22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, page 295, oct. 2010. 36

Shinji Kikuchi and Yasuhide Matsumoto. Impact of live migration on multi-tier application performance in clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012. 81, 94

Tim kovacs. XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. Technical report, School of computer science and cognitive science research centre, The University of Birmingham, Birmingham B15 2TT, UK, 1997. 29

Juraj Koščak, Rudolf Jakša, and Peter Sinčák. Stochastic weight update in the backpropagation algorithm on feed-forward neural networks. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–4, july 2010. 42

John R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, Massachusetts, USA, 1992. ISBN 0-262-11170-5. 14

Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, 29(2): 249–272, 2011. ISSN 0219-1377. 82

Chan Man Kuok, Ada Fu, and Man Hon Wong. Mining fuzzy association rules in databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27, pages 41–46, New York, NY, USA, march 1998. ACM. 99

Prasana K. Lakshmi and Chandan R. K. Reddy. A survey on different trends in data streams. In *Networking and Information Technology (ICNIT), 2010 International Conference on*, pages 451–455, june 2010. 36

Carson K.-S. Leung and Hao Boyu. Mining of frequent itemsets from streams of uncertain data. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*, pages 1663–1670, april 2009. 99

Hua-Fu Li and Suh-Yin Lee. Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 36(2):1466–1477, march 2009. ISSN 0957-4174. 99

Hua-Fu Li, Man-Lwan Shan, and Suh-Yin Lee. Dsm-fi: an efficient algorithm for mining frequent itemsets in data streams. *Knowledge and Information Systems*, 17:79–97, october 2008. ISSN 0219-1377. 99

Hua-Fu Li, Chin-Chuan Ho, and Suh-Yin Lee. Incremental updates of closed frequent itemsets over continuous data streams. *Expert Sysems Applied*, 36(2):2451–2458, march 2009. ISSN 0957-4174. 99

Hongyan Liu, Yuan Lin, and Jiawei Han. Methods for mining frequent items in data streams: an overview. *Knowledge and Information Systems*, 26(1):1–30, 2011. ISSN 0219-1377. 106

Xavier Llorà. *Aprenentatge artificial evolutiu emprant paral·lelisme de gra fi en el marc de la mineria de dades*. PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, february 2002. 20

Mariela J. Louis-Rodríguez, Joan Navarro, Itziar Arrieta-Salinas, Ainhoa Azqueta-Alzúaz, Andreu Sancho-Asensio, and José E. Armendáriz-Iñigo. Workload management for dynamic partitioning schemes in replicated databases. In *the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013)*, volume In Press, may 2013. 93, 94

Jun Li Lu, Li Zhen Wang, Jun Jia Lu, and Qiu Yue Sun. Research and application on KNN method based on cluster before classification. In *International conference on machine learning and cybernetics*, volume 1, pages 307–313, July 2008. 82

Edwin Lughofer and Plamen Angelov. Handling drifts and shifts in on-line data streams with evolving fuzzy systems. *Applied Soft Computing*, 11(2):2057–2068, March 2011. ISSN 1568-4946. 12, 36, 38, 105

John MacQueen. Some methods for classification and analysis of multivariate observations. In *Procredings of the 5th Berkeley Symp. on Mathematics Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967. 82

Marcus A. Maloof and Ryszard S. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154(1-2):95–126, 2004. ISSN 0004-3702. 36

María Martínez-Ballesteros, Francisco Martínez-Álvarez, Alicia Troncoso, and José C. Riquelme. An evolutionary algorithm to discover quantitative association rules in multidimensional time series. *Soft Computing*, 15:2065–2084, 2011a. ISSN 1432-7643. 21, 99, 104, 112

María Martínez-Ballesteros, Sancho Salcedo-Sanz, Joseé C. Riquelme, Carlos Casanova-Mateo, and Joseé L. Camacho. Evolutionary association rules for total ozone content modeling from satellite observations. *Chemometrics and Intelligent Laboratory Systems*, 109(2):217–227, 2011b. ISSN 0169-7439. 21

Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M. Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *Knowledge and Data Engineering, IEEE Transactions on*, 23(6):859–874, 2011. 36

Jacinto Mata and José Cristóbal Riquelme. Mining numeric association rules with genetic algorithms. In *Proceedings of the international conference on adaptive and natural computing algorithms*, pages 264–267, 2001. 101

Jacinto Mata, José Luis Álvarez Macías, and José Cristóbal Riquelme Santos. Discovering numeric association rules via evolutionary algorithm. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD, pages 40–51, London, UK, 2002. Springer-Verlag. ISBN 3-540-43704-5. 99

Gregor Mendel. Experiments in Plant Hybridization. In *Natural History Society of Brunn in Bohemia*, 1865. 14

Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs*. Artificial intelligence. Springer, third edition, 1996. ISBN 9783540606765. 18

Renée J. Miller and Yuping Yang. Association rules over interval data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 452–461, 1997. 99

Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, third edition, 1998. ISBN 0262631857. 13, 18

Tom Mitchell. *Machine Learning*. Prentice Hall, Pittsburgh, 1997. ISBN 0070428077. 1, 9, 42, 45

Tom Mitchell. The discipline of machine learning, 2006. 9

Antonello Monti and Fernando Ponci. Power grids of the future: Why smart means complex. In *Complexity in Engineering, 2010*, COMPENG '10, pages 7–11, february 2010. ISBN 978-1-4244-5982-7. 78, 83

Amir Motamedi, Hamidreza Zareipour, and William D. Rosehart. Electricity price and demand forecasting in smart grids. *Smart Grid, IEEE Transactions on*, 3(2):664–674, june 2012. ISSN 1949-3053. 98

Niloofar Mozafari, Sattar Hashemi, and Ali Hamzeh. A precise statistical approach for concept change detection in unlabeled data streams. *Computers & Mathematics with Applications*, 62(4):1655–1669, 2011. 38, 105

Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evolutive Computation*, 1(1):25–49, march 1993. ISSN 1063-6560. 15

Joan Navarro, José E. Armendáriz-Iñigo, and August Climent. An adaptive and scalable replication protocol on power smart grids. *Scalable Computing: Practice and Experience*, 12(3), 2011. 78, 79, 80, 89, 92, 93

Joan Navarro, Andreu Sancho-Asensio, Carles Garriga, Jordi Albo-Canals, Julio Ortiz-Villajos Maroto, Cristobal Raya, Cecilio Angulo, and David Miralles. A cloud robotics architecture to foster individual child partnership in medical facilities. In *Proceeding of the 26th IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS'13. IEEE, november 2013a. 146

Joan Navarro, Agustín Zaballos, Andreu Sancho-Asensio, Guillermo Ravera, and José E. Armendáriz-Iñigo. The information system of INTEGRIS: INTelligent Electrical GRId Sensor communications. *Industrial Informatics, IEEE Transactions on*, 9(3):1548–1560, 2013b. ISSN 1551-3203. 78, 80, 81, 82, 89, 92

Paul Nemenyi. *Distribution-free multiple comparisons.* PhD thesis, Princeton University, New Jersey, USA, 1964. 49, 154

Marlon Núñez, Raúl Fidalgo, and Rafael Morales. Learning in environments with unknown dynamics: towards more robust concept learners. *Journal of Machine Learning Research*, 8:2595–2628, 2007. ISSN 1532-4435. 12, 36, 45, 46, 47, 105, 122

Albert Orriols-Puig. *New challenges in learning classifier systems: mining rarities and evolving fuzzy models.* PhD thesis, Arquitectura i Enginyeria La Salle, Universitat Ramon Llull, Passeig de la Bonanova 8, 08022 - Barcelona, November 2008. 1, 2, 3, 10, 13, 14, 15, 16, 18, 20, 21, 24, 29, 30, 31, 32, 33, 34, 48, 84, 89, 102, 115, 132, 142, 145, 149, 152, 154

Albert Orriols-Puig and Ester Bernadó-Mansilla. Revisiting UCS: Description, fitness sharing, and comparison with XCS. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin V. Butz, Tim Kovacs, Xavier Llorà, and Keiki Takadama, editors, *IWLCS*, volume 4998 of *Lecture Notes in Computer Science*, pages 96–116. Springer, 2008. ISBN 978-3-540-88137-7. 4, 31

Albert Orriols-Puig and Jorge Casillas. Evolution of interesting association rules online with learning classifier systems. In Jaume Bacardit, Will Browne, Jan Drugowitsch, Ester Bernadó-Mansilla, and Martin V. Butz, editors, *Learning Classifier Systems*, volume 6471 of *Lecture Notes in Computer Science*, pages 21–37. Springer Berlin Heidelberg, 2010a. ISBN 978-3-642-17507-7. 4, 13, 99, 104

Albert Orriols-Puig and Jorge Casillas. Fuzzy knowledge representation study for incremental learning in data streams and classification problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, pages 1–26, 2010b. ISSN 1432-7643. 1, 3, 4, 13, 36, 38, 45, 47, 75, 100, 105, 114

Albert Orriols-Puig, Jorge Casillas, and Ester Bernadó-Mansilla. First approach toward online evolution of association rules with learning classifier systems. In *Proceedings of the 2008 GECCO conference companion on Genetic and Evolutionary Computation*, GECCO, pages 2031–2038, New York, USA, 2008a. ACM. ISBN 978-1-60558-131-6. 99, 132

Albert Orriols-Puig, Jorge Casillas, and Ester Bernadó-Mansilla. Genetic-based machine learning systems are competitive for pattern recognition. *Evolutionary Intelligence*, 1(3): 209–232, 2008b. ISSN 1864-5909. 132

Albert Orriols-Puig, Ester Bernado-Mansilla, David E. Goldberg, Kumara Sastry, and Pier L. Lanzi. Facetwise analysis of XCS for problems with class imbalances. *Evolutionary Computation, IEEE Transactions on*, 13(5):1093–1119, 2009a. 23, 139

Albert Orriols-Puig, Jorge Casillas, and Ester Bernado-Mansilla. Fuzzy-UCS: A michigan-style learning fuzzy-classifier system for supervised learning. *Evolutionary Computation, IEEE Transactions on*, 13(2):260–283, april 2009b. ISSN 1089-778X. 75, 88, 99, 113, 139

Albert Orriols-Puig, Xavier Llorà, and David E. Goldberg. How XCS deals with rarities in domains with continuous attributes. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1023–1030, New York, USA, 2010. ACM. ISBN 978-1-4503-0072-8. 23, 29, 132

Albert Orriols-Puig, Francisco J. Martínez-López, Jorge Casillas, and Nick Lee. A soft-computing-based method for the automatic discovery of fuzzy rules in databases: Uses for academic research and management support in marketing. *Journal of Business Research*, In Press:–, 2012. ISSN 0148-2963. 4, 79, 99

David P. Pancho, Jose M. Alonso, Oscar Cordón, Arnaud Quirin, and Luis Magdalena. Fingrams: Visual representations of fuzzy rule-based inference for expert analysis of comprehensibility. *Fuzzy Systems, IEEE Transactions on*, 21(6):1133–1149, Dec 2013. ISSN 1063-6706. 146

Chen Peng, Su Hongye, Guo Lichao, and Qu Yu. Mining fuzzy association rules in data streams. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 4, pages 153–158, april 2010. 99, 105

John Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998. 37, 66

Steven R. Quartz and Terrence J. Sejnowski. The neural basis of cognitive development: A constructivist manifesto, 1999. 37

John R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann Publishers, 1993. ISBN 1-55860-238-0. 37, 66

Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Problemata, 15. Frommann-Holzboog, 1973. ISBN 9783772803734. 14

David E. Rumelhart, George E Hinton, and James L. McClelland, editors. *Parallel distributed processing: explorations in the microstructure of cognition, volume 1: foundations.* MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. 37, 66

Sebnem Rusitschka, Kolja Eger, and Christoph Gerdes. Smart grid data cloud: A model for utilizing cloud computing in the smart grid domain. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, 2010. 78, 80

Hans P. Schwefel. *Artificial intelligence through simulated evolution.* John Wiley & Sons, New York, USA, 1981. ISBN 0471099880. 14

Josep M. Selga, Agustín Zaballos, and Joan Navarro. Solutions to the computer networking challenges of the distribution smart grid. *IEEE Communication Letters*, 18(3):588–591, 2013. 78, 80, 81

Lama Seoud, MathiasM. Adankon, Hubert Labelle, Jean Dansereau, and Farida Cheriet. Towards non invasive diagnosis of scoliosis using semi-supervised learning approach. In Aurlio Campilho and Mohamed Kamel, editors, *Image Analysis and Recognition*, volume 6112 of *Lecture Notes in Computer Science*, pages 10–19. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13774-7. 146

Juliet P. Shaffer. Modified sequentially rejective multiple test procedures. *Journal of the American Statistical Association*, 81:826–831, 1986. 49, 157

David J. Sheskin. *Handbook of parametric and nonparametric statistical procedures.* Chapman & Hall, third edition, 2004. ISBN 1-58488-440-1. 152

Liangdong Shi, Yinghuan Shi, Yang Gao, Lin Shang, and Yubin Yang. XCSc: A novel approach to clustering with extended classifier system. *Internotional Journal on Neural Systems*, 1(21), 2011. 4, 77, 82, 83, 88

Xiaoxiao Shi, Wei Fan, and Philip S. Yu. Efficient semi-supervised spectral co-clustering with constraints. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 1043–1048, 2010. 75, 98

Ajay Singh, Mukesh Chaudhary, Ajay Rana, and Ghanshyam Dubey. Online mining of data to generate association rule mining in large databases. In *2011 International Conference on Recent Trends in Information Systems, ReTIS*, december 2011. 99

Stephen F. Smith. *A learning system based on genetic adaptive algorithms.* PhD thesis, University of Pittsburgh, Pittsburgh, USA, 1980. 2, 20

Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, volume 25, pages 1–12, Montreal, Quebec, Canada, June 1996. ACM. 99, 101

Patrick O. Stalph and Martin V. Butz. Current XCSF capabilities and challenges. In *IWLCS*, volume 6471 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 2010. 132, 133

Patrick O. Stalph, Xavier Llorà, David E. Goldberg, and Martin V. Butz. Resource management and scalability of the XCSF learning classifier system. *Theoretical Computer Science*, 425:126–141, 2012. 89, 131, 132, 133, 134, 135, 137, 138

Christopher Stone and Larry Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):298–336, 2003. 25, 84

Michael Stonebraker and Jason Hong. Researchers' big data crisis; understanding design and functionality. *Commun. ACM*, 55(2), February 2012. 78, 80

W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '01*, pages 377–382, New York, USA, 2001. ACM. ISBN 1-58113-391-X. 45, 46, 114

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, an Introduction.* The MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-19398-1. 12

Kreangsak Tamee, Larry Bull, and Ouen Pinngern. A learning classifier system approach to clustering. In *Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, ISDA, pages 621–626. IEEE Computer Society, 2006. ISBN 0-7695-2528-8. 82, 88, 132

Kreangsak Tamee, Larry Bull, and Ouen Pinngern. Towards clustering with XCS. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1854–1860, New York, USA, 2007a. ACM. ISBN 978-1-59593-697-4. 4, 13, 77, 82, 83, 84, 86, 88

Kreangsak Tamee, Larry Bull, and Ouen Pinngern. YCSc: A modified clustering technique based on lcs. *Journal of Digital Information Management*, 5(3):160–166, 2007b. 82, 88

Dayrelis Mena Torres, Jesús S. Aguilar-Ruiz, and Yanet Rodríguez Sarabia. An instance based learning model for classification in data streams with concept change. In *11th Mexican International Conference on Artificial Intelligence, MICAI 2012, San Luis Potos, Mexico, October 27 - November 4, 2012, Special Session Proceedings*, pages 58–62, 2012. 36

Cheng-Jung Tsai, Chien-I. Lee, and Wei-Pang Yang. Mining decision rules on data streams in the presence of concept drifts. *Expert Systems and Applications*, 36(2):1164–1178, march 2009. ISSN 0957-4174. 99

Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag, New York, USA, 1995. 37, 66

Gilles Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. In Pavel Brazdil, editor, *Machine Learning: ECML-93, European Conference on Machine Learning*, volume 667 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 1993. ISBN 3-540-56602-3. 20, 104

Gilles Venturini. *Apprentissage adaptatif et apprentissage supervisé par algorithme génétique.* PhD thesis, Université de Paris-Sud, Paris, France, 1994. 27

Periasamy Vivekanandan and Raju Nedunchezhian. Mining data streams with concept drifts using genetic algorithm. *Artificial Intelligence Review*, 36(3):163–178, 2011. 36

Ching-yao Wang, Shian-shyong Tseng, Tzung-pei Hong, and Yian-shu Chu. Online generation of association rules under multidimensional consideration based on negative-border. *Journal of Information Science and Engineering*, 23:233–242, 2004. 99

En T. Wang and Arbee L.P. Chen. Mining frequent itemsets over distributed data streams by continuously maintaining a global synopsis. *Data Mining and Knowledge Discovery*, 23 (2):252–299, 2011. ISSN 1384-5810. 99

Jingyan Wang, Yongping Li, Ying Zhang, and Jianhua He. Semi-supervised protein function prediction via sequential linear neighborhood propagation. In De-Shuang Huang, Yong Gan, Prashan Premaratne, and Kyungsook Han, editors, *Bio-Inspired Computing and Applications*, volume 6840 of *Lecture Notes in Computer Science*, pages 435–441. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-24552-7. 146

Ke Wang, Soon Hock, William Tay, and Bing Liu. Interestingness-based interval merger for numeric association rules. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD*, pages 121–127. AAAI Press, 1998. 99

Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct 1990. ISSN 0018-9219. 42

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, april 1996. ISSN 0885-6125. 36

Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, september 1990. ISSN 0018-9219. 26, 37, 40, 66, 86

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 6(1): 80–83, december 1945. 127, 151

Stewart W. Wilson. ZCS: a zeroth level classifier system. Technical report, The Rowland Institute for Science, 100 Edwin H. Land Blvd. Cambridge, MA 02142, 1994. 19, 37

Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutive Computation*, 3:149–175, June 1995. ISSN 1063-6560. 2, 19, 20, 23, 24, 27, 29, 37, 43, 86, 99, 109

Stewart W. Wilson. Generalization in the XCS classifier system. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann. ISBN 1-55860-548-7. 20, 23, 24, 27, 99, 110

Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems, From Foundations to Applications, LNAI-1813*, pages 209–219. Springer-Verlag, 2000. 25

Stewart W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1:211–234, June 2001. ISSN 1567-7818. 37

Stewart W. Wilson. Classifier conditions using gene expression programming. In Jaume Bacardit, Ester Bernadó-Mansilla, Martin Butz, Tim Kovacs, Xavier Llorà , and Keiki Takadama, editors, *Learning Classifier Systems*, volume 4998 of *Lecture Notes in Computer Science*, pages 206–217. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-88137-7. 37

Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufmann, third edition, January 2011. ISBN 978-0-12-374856-0. 10, 48, 66

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey McLachlan, Angus Ng, Bing Liu, Philip Yu, Zhi-Hua Zhou, Michael Steinbach, David Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, January 2007. ISSN 0219-1377. 66

Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with Big Data. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):97–107, Jan 2014. ISSN 1041-4347. 147

Xiaowei Yan, Chengqi Zhang, and Shichao Zhang. Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert Systems with Applications*, 36(2, Part 2):3066–3076, 2009. ISSN 0957-4174. 99, 104

Ye Yan, Yi Qian, Hamid Sharif, and David Tipper. A survey on smart grid communication infrastructures: Motivations, requirements and challenges. *IEEE Commun. Surveys& Tutorials*, 15(1), 2013. 78, 80

Lotfi A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965. 99

Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association. 82

Guoqiang P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, November 2000. ISSN 10946977. 37

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 103–114, New York, USA, 1996. ACM. ISBN 0-89791-794-4. 82

Shi Zhong. Efficient online spherical K-Means clustering. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 5, pages 3180–3185, 2005. 82

Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi. Active learning from stream data using optimal weight classifier ensemble. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 40(6):1607–1621, december 2010. ISSN 1083-4419. 36