CSSE Technical Report UWA-CSSE-2010-001
March 2010
Learning Classifier Systems for Fraud Detection

Mohammad Behdad Tim French Luigi Barone
Mohammed Bennamoun
School of Computer Science & Software Engineering
The University of Western Australia
Crawley WA 6009 Australia
email: {behdad,tim,luigi,bennamou}@csse.uwa.edu.au
tel: +61 8 6488 1945
fax: +61 8 6488 1089

March 8, 2010

Abstract

Fraud is a serious problem that costs the worldwide economy trillions of dollars annually. However, fraud detection is difficult as perpetrators actively attempt to masquerade their actions, among typically overwhelming large volumes of, legitimate activity. In this paper, we investigate the fraud detection problem and examine how learning classifier systems can be applied to it. We describe the common properties of fraud, introducing an abstract problem which can be tuned to exhibit those characteristics. We report experiments on this abstract problem with a popular real-time learning classifier system algorithm; results from our experiments demonstrating that this approach can overcome the difficulties inherent to the fraud detection problem. Finally we apply the algorithm to a real-world problem (KDD Cup 1999 network intrusion detection) and show that it can achieve very good performance in this domain.

Keywords: Learning Classifier Systems, XCSR, Fraud Detection, Network Intrusion Detection

1 Introduction

Fraud is the general term used to describe actions undertaken by one party to obtain an unjust advantage over another. Typically, the perpetrators, using false or misleading representations, attempt to disguise their activities in an effort to maximise the effects of their fraudulent behaviour. Fraud is a serious

problem that costs the worldwide economy trillions of dollars annually. Indeed, it was estimated that the cost of personal fraud for Australians in 2007 was $980 million [1]. This research deals with the growing problem of electronic fraud (e-fraud) — fraud perpetrated using electronic technologies; the most common forms including network intrusion, email spam, online credit card fraud, and telecommunications fraud .

Fraud detection is the task of attempting to detect and recognise fraudulent activities as they occur, typically by flagging suspicious behaviour for review and further action by the appropriate authorities. However, fraud detection is difficult to approach using automated methods since most perpetrators actively try to undermine any attempt to classify their activities as fraud. Fraud detection hence necessarily corresponds to an adversarial environment.

Previous work on automated fraud detection has focused on specific domains such as spam detection [13], network intrusion [19] and phishing [8]. In this paper, we identify some general characteristics of electronic fraud detection and examine the performance of learning classifier systems (LCSs) on abstract problems that exhibit these characteristics. We emulate simple fraud detection problems using a parameterizable random Boolean function; this abstract test problem retaining the same underlying features of real-world fraud detection problems while offering a simple domain to understand and analyse. We also evaluate the performance of LCSs on a real-world problem (KDD Cup 1999 intrusion detection); results confirming the efficiency of this approach.

In the next section, we examine the abstract properties of fraudulent behaviour, defining characteristic properties we typically expect to observe in such domains. In Section 3 we present background information about LCSs and discuss how they might be used for fraud detection in dynamic environments. In Section 4 we present experiments that explore the performance of XCSR, a real-time LCS variant, on a representative problem that captures the defining characteristic properties introduced in Section 2 and on a real-world problem. Finally, Section 5 concludes this work.

2 Properties of Fraudulent Activity

Fraud detection is an extremely difficult task. As is evidenced by the ongoing large financial losses of organisations and individuals due to fraudulent behaviour [1], no general off-the-shelf solution exists for this problem. The reason for this failure is the fact that fraud detection must deal with some uniquely challenging properties. These properties are described below:

1. **Biased Classes**: Perhaps the most defining and important characteristic of fraudulent activity is the bias in positive and negative experiences presented during learning; i.e. the proportion of fraudulent records to non-fraudulent records is vastly biased. This "needle-in-a-haystack" like effect typically leads to unacceptably high false positive rates in the data [15]. Indeed, these "rare" instances can have significant negative impacts on

performance in data mining problems [27] and hence any fraud detection system must learn to overcome the problems associated with the rarity of positive examples in its learning process.

2. **Online Learning**: Since complete data is not available from the onset (it becomes available gradually over time), we typically have only partial information about the problem at hand [26]. This complicates the learning process as there is a lack of sufficient data for training purposes. Indeed, the detection system may need to approximate or generalise from the limited experiences it has been presented, but more importantly must also be able to quickly adapt as new (potentially conflicting) experiences are observed.

3. **Adaptive Adversaries**: Another difficult aspect of fraud detection is the perpetual "arms race" that occurs when dealing with intelligent adaptive adversaries who vary their techniques over time [7]. As the detection techniques mature, fraud perpetrators also become increasingly sophisticated in both their methods of attack and detection evasion. This means that classifiers may be undermined by their own success; over-specialisation is a risk.

4. **Concept Drift**: In machine learning, "concept drift" means that the statistical properties of the target variable that the model is trying to predict change over time in unforeseen ways. This makes the predictions less accurate as time passes. Fraud detection suffers from a continuously changing concept definition [18]. Therefore, the approaches used for this purpose should be able to adapt themselves in the presence of drifting concepts over time.

5. **Misclassification Costs**: In fraud detection, misclassification costs (false positive and false negative error costs) are unequal, uncertain, can differ from example to example, and can change over time. In some domains, a false negative error is more costly than a false positive (e.g. intrusion detection), whereas in other domains (e.g. spam detection) the situation is reversed [15]. This adds to the complexity of the learning problem. Another consequence of this property is the absolute need for high accuracy.

6. **Noise**: Noise is the random perturbations present in data that lead to variation in observations from the true data signal. The presence of noise in data is almost unavoidable, potentially introduced during any number of stages including data collection, transmission, or processing. Alas, the data-sets analysed and monitored in fraud detection, partly due to their magnitude, have a considerable amount of noise. Unfortunately, the presence of noise makes learning more difficult, typically slowing the rate of learning as the system needs to compensate for false experiences or inaccurate rewards. One of the challenges of fraud detection is distinguishing between inconsequential random noise and deliberate fraudulent behaviour — fraudulent activity potentially resembling noise due to its

atypical appearance [3]. Even a small amount of noise is harmful when high accuracy is required.

7. **Fast Processing over Large Data Sizes**: In some applications such as network intrusion detection and email spam detection, fraud detection must be done in near real-time. In these cases, the speed of processing is critical, thus not affording the fraud detection software the luxury of extended processing times [7].

3 Learning Classifier Systems

Many researchers have explored different machine learning and computational intelligence techniques for specific applications of fraud detection. For example, email spam detection has been examined using naïve Bayes [17], support vector machines [16], visualization [30], artificial immune systems [13], evolutionary algorithms [6], and artificial neural networks [31]. Similar examples exist for other domains like network intrusion detection.

However, precious few examples exist in applying a learning classifier system (LCS) to fraud detection; the most notable being the recent works of Shafi et al. [19] and Tamee et al. [23] on using an LCS for network intrusion detection. The former analyse two LCS variants on a subset of a publicly available benchmark intrusion detection data-set, suggesting some improvements for these two algorithms. They also compare the performance of these methods with other machine learning algorithms and conclude that LCSs are a competitive approach to network intrusion detection. The work by Tamee et al. proposes using Self-Organizing Maps (SOMs) with LCS for network intrusion detection. They analyse the performance of their method and show that the proposed system is able to perform significantly better than twelve machine learning algorithms.

This apparent under-utilisation of LCSs for fraud detection problem is somewhat surprising, as LCSs have been successfully used in a number of related machine learning and data mining tasks [4] and, due to their online learning capabilities in dynamic environments [2] and straightforward knowledge representation, seem readily applicable to this kind of domain.

3.1 Learning Classifier System

First proposed by John Holland [10] in 1976, a learning classifier system (LCS) is a machine learning technique that combines reinforcement learning, evolutionary computing, and other heuristics to produce an adaptive system capable of learning in dynamic environments. It uses a population of condition-action-prediction rules called classifiers to encode appropriate actions for different input states; rules are of the If-Then type — if a condition is true, then the corresponding action will be performed. The decision of which rule to use is based on the reward (the prediction in the classifier triple) the system expects to receive from the environment as a result of performing the chosen action. In essence, an LCS

attempts to enhance its understanding of the environment through improving its classifiers over time.

The three major components of an LCS are rule discovery, credit assignment, and the production system. Rule discovery is the task of creating new rules and is generally performed using a genetic algorithm (GA). The GA operates on the population of classifiers, evolving them through recombination and selection. The fitness function used depends on the type of LCS; a strength-based LCS uses the predicted reward of a rule as its fitness, whereas an accuracy-based LCS uses the inverse of the prediction error as the fitness value of a rule. Other evolutionary algorithm variants have also been used in place of the GA.

As an LCS should adapt to changes over time, it must therefore learn from its environment and improve its rules over time. Credit assignment is the process of updating the reward prediction value of the rules based on the feedback received from the environment after performing an action.

When deciding on which rule to use in a particular input state, sometimes more than one rule whose condition evaluates to true are found. In this situation, it is the responsibility of the production system to determine which rule to use. The production system usually uses the strength (reward prediction) of the matching rules in its decision making process. However, other mechanisms such as roulette wheel selection also accompany this process to create a balance between exploitation and exploration.

3.2 Accuracy-based Learning Classifier System

For many years, LCS algorithms used the classifier strength parameter (the estimated reward) both as a predictor of reward and as the classifier fitness in the GA. However, in 1995 Stewart Wilson investigated this practice, deciding instead to use two separate parameters for the two different processes [28]. Creating a new LCS variant called XCS (which has now become the most applied and studied LCS over the last several years [11]), the new parameter he introduced was accuracy. Under his scheme, it is not the classifiers which produce the greatest rewards that have the best chance of propagating their genes into subsequent generations, but rather the classifiers which predict the reward more accurately independent of the magnitude of the reward. As a result, XCS is said to cover the state space more efficiently [20].

When a state is perceived by an LCS, the rule-base is scanned and any rule whose condition matches the current state is added as a member of the current "match set" M. In XCS, once M has been formed, a rule is chosen based on its accuracy. All members of M that propose the same action as the chosen rule then form the "action set" A. The accuracy of all members of the action set are then updated when the reward from the environment is determined.

Another significant difference of XCS over its predecessors is that the GA is applied to A (or niches) rather than the whole classifier population. This narrows the competition to occur between classifiers which can be applied in similar scenarios rather than all the classifiers in the LCS [20].

The other important concept in XCS is that of the macro-classifier. During rule discovery, it is possible for classifiers identical to those already in existence to be generated. Instead of keeping several identical classifiers in the population, XCS associates with each classifier a numerosity corresponding to the number of examples in the population, thus allowing the LCS to explore other potential rules while maintaining recognition of the currently dominant classifiers [20].

3.3 XCSR

Traditionally XCS has been used for problems with binary strings as input. However, many real-world problems have continuous values and can not be represented by the ternary representation used by XCS. A continuous version of XCS called XCSR was introduced by Wilson [29] which handles this kind of problems using an interval-based representation. There have been many different suggestions for how to implement an LCS using this representation. For this work, we have used the Unordered Bound Representation [22].

4 Experiments

In this section, we present experiments using an abstract problem (random Boolean function) that exhibit the characteristic properties of fraudulent activity defined in Section 2 and a real-world problem (KDD Cup 1999 intrusion detection data-set) that exhibits many of these features in one composite problem. We analyse the performance of XCSR on five facet-wise experiments, namely in the presence of skewed input, biased input, biased output, incomplete information, and noise. Since we are dealing with fraud detection where data is heavily biased, accuracy — the percentage of the correct predictions made by the system — is not a suitable performance measure, as it may produce misleading conclusions. Instead, we use a performance metric based on the product of the true positive rate ($TPR = TP/(TP + FN)$) and the true negative rate ($TNR = TN/(TN + FP)$) [25].

4.1 Experimental Framework

4.1.1 Random Boolean Function

A Boolean function of n Boolean variables, $p_1, p_2, ..., p_n$, is a function $f : B^n \rightarrow B$ such that $f(p_1, p_2, ..., p_n)$ is a Boolean expression [9]. If $n = 6$, a random Boolean function (RBF) is created by generating randomly a $2^n = 64$ bit binary string called the *action string* which represents the correct action for every possible state (6-bit binary string). By calculating the decimal value of a 6-bit binary string, the index of the correct action in the action string is found. For example, if the first 10 bits of the action string are 0110010110, then the correct action for state 000101 is the value of the 5^{th} bit (counting from 0) in the action string — a 1 in this example. In order to ensure a fair comparison between the different algorithms, 20 different random Boolean functions (20 action strings)

are generated at the start of an experiment; experimental results report the performance of each algorithm averaged across all 20 functions.

As we are dealing with real valued inputs, a real version of the RBF problem is defined as follows: for an input vector $x = (x_0, ..., x_n)$, every x_i is a real value in the range $0 \leq x_i < 1$. Every real-valued input vector is mapped to a binary string input, using a threshold ($0 \leq \theta < 1$): if $x_i < \theta$, x_i is interpreted as 0, otherwise it is interpreted as 1.

4.1.2 XCSR

The underlying accuracy-based LCS (XCS) is based on Butz's implementation of XCS [5]. For representation of intervals, we use the Unordered Bound Representation (UBR) without limiting the ranges to be between 0 and 1.

4.1.3 KDD Cup 1999

We also evaluate the performance of XCSR on the 1999 KDD Cup data-set — a benchmark data-set for the evaluation of network intrusion detection systems.

4.2 Experiment 1: Input Skewness

In our continuous version of the RBF problem, all bits are set to a value between 0 and 1, and if a bit is less than some threshold (which normally is 0.5), it is interpreted as a 0, otherwise as a 1. By shifting the threshold towards 0 or 1, we are able to change the skewness of the input. It should be noted that the concept of "input skewness" is not meaningful in discrete environments and is only observed in continuous environments.

Fig. 1 illustrates the average performance (product of TPR by TNR) of XCSR on a 4-bit RBF in the presence of different levels of input skewness in three time windows: during the early stages of learning (between 5,000 and 6,000 trials), in the middle of the run (between 15,000 and 16,000 trials), and finally after the performance has stabilized (between 35,000 and 36,000 trials). This experiment was repeated 20 times and the results averaged.

As Fig. 1 shows, changes in input skewness have no effect on the performance of XCSR except in extremely skewed cases — in other words, when the threshold is very close to the predefined boundaries (0 or 1). The reason is that the mutation step and other parameters of the algorithm are not small enough for such situations. However, as the figure shows, the LCS is able to obtain a very high level performance (to learn the correct threshold), but only after a significantly higher number of learning experiences than for lower input skewness values. Also, by tuning the parameters for the environment at the onset, or removing the predefined boundaries, this problem can be overcome.

4.3 Experiment 2: Input Bias

In this experiment, we start with a 4-bit RBF environment in which every state has the same number of bits belonging to class 0 as the bits belonging to class

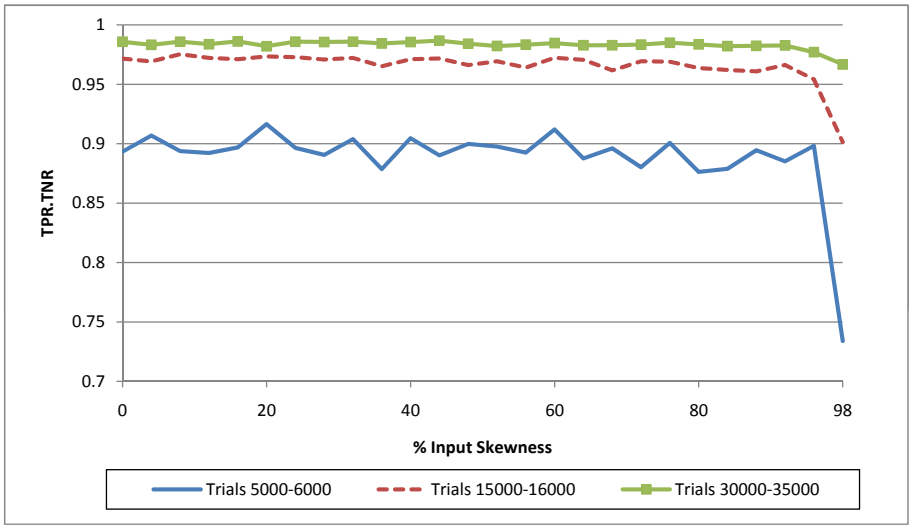

Figure 1: The effect of skewed input for the RBF-4 problem

1. We then modify this balance and gradually introduce bias to the input by making 1 bits rare. This means, for a 0% input bias a bit has an equal chance of being a 0 or a 1, and for 98%, a bit has a 98% chance of being a 0 and 2% being a 1. This experiment was repeated 20 times and the results averaged.

Fig. 2 illustrates the average performance (product of TPR by TNR) of XCSR in three different time windows: during early stages of learning (between 5,000 and 6,000 trials), in the middle way (between 15,000 and 16,000 trials), and finally after the performance has stabilized (between 35,000 and 36,000 trials).

As the figure shows, input bias does affect the performance of XCSR, especially during the early trials of a run. Interestingly, during the early phases of learning, the performance in environments with a higher input bias appears to be better than environments with a lower input bias. But, as the algorithm continues learning, performance on environments with lower input bias catches up and the algorithm ends up performing just as well as on the highly-input-biased ones. The reason is that with a high bias in input strings, most of the states belong to a much smaller state space. Learning and modelling this space is easier and requires fewer training trials than lowly biased environments.

4.4 Experiment 3: Output Bias

This experiment analyses the performance of XCSR for different levels of output bias. For this experiment, the 6-bit RBF problem (RBF-6) is used; the experiment varies the proportion of 1s and 0s in the action string of the random

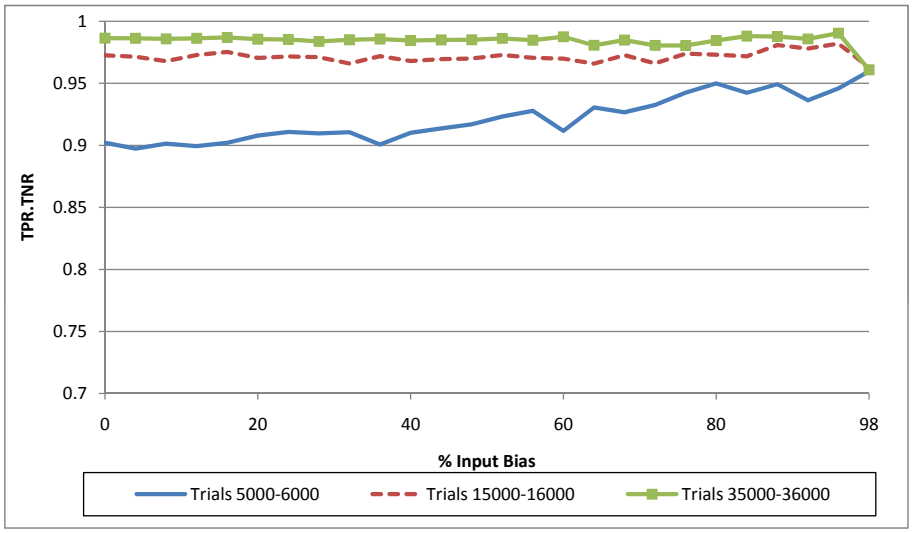

Figure 2: The effect of input bias for the RBF-4 problem

Boolean function providing a means of varying the output bias in experiences offered to the LCS. A 0% biased environment is a completely balanced environment, while 100% represents a completely biased environment. It should be noted that this form of bias is what all other papers working on "class imbalance" use.

In order to investigate the effects of output bias, the output bias was systematically varied between 0% and 96% and the performance of different LCS variants determined for each output bias value. Fig. 3 summarises the results, reporting the performance for 1,000 trials after trial 50,000. Fig. 4 reports the same, but for much longer number of trials for problems with higher output bias. Reported results are the average of 10 independent runs.

As we can see from Fig. 3, in the unbiased RBF-6 environment, all XCSR variants obtain an acceptable performance (TPR.TNR) of 93% after approximately 50,000 trials. The red broken line in Figs. 3 and 4 plot the average performance of the baseline XCSR. Note that the performance generally decreases as the output bias is varied from 0% to 96%. Surprised by the relative poor performance of XCSR on highly biased environments, we varied the algorithm's control parameters in order to "tune" the algorithm for the problem at hand. We found that by raising the threshold before subsumption can occur (θ_{Sub}) and making it more difficult for a classifier to be judged as perfectly accurate (ϵ_0), the performance of XCSR was significantly improved. We also observed that disabling (both forms of) subsumption improves the performance substantially in the biased environments. The black solid line decorated with triangles in Figs. 3 and Fig. 4 plots the performance of this variant. We ob-

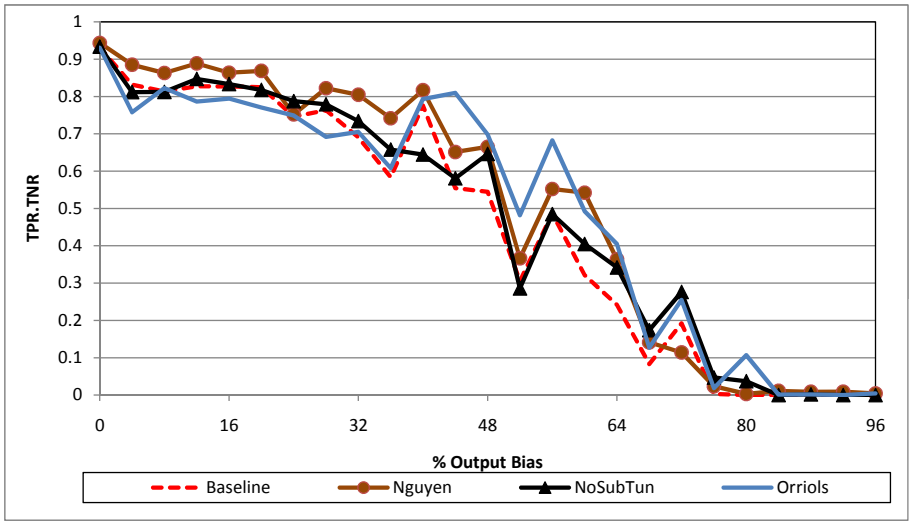

Figure 3: The effect of output bias for the RBF-6 problem in trials 50,000 to 51,000

serve from Fig. 4 that by tuning and disabling subsumption, the performance of XCSR is substantially surpassing the performance of baseline XCSR, obtaining an acceptable performance for output-biased environments. By removing subsumption, the number of specialised classifiers in the population increases, hence allowing a relatively larger knowledge base for decision making, giving the LCS the power to discriminate between states with a finer resolution. However, this increase in the number of specialised classifiers decreases the generalization capability of the algorithm and is ultimately limited by the maximum size of the classifier set in the LCS. For many problems, this fine-scale specialisation may simply not be feasible.

Also plotted on Figs. 3 and 4 are two variants of XCSR specifically modified to handle output-biased environments. The first, by Nguyen et al. [12], modifies three baseline XCSR control parameters: two of which are set to different constant values and the third — the minimum error (ϵ_0) — is calculated based on the inverse of the output bias level. Its performance is plotted using a brown solid line decorated with circles. The second one is an XCSR variant based on the recommendations of Orriols-Puig and Bernadó-Mansilla [14] — the performance of this variant plotted using a blue solid line. This XCSR variant uses tournament selection in the discovery component of the LCS and GA subsumption (but no action set subsumption) with the minimum GA experience (θ_{GA}) set to be inversely proportional to the output bias of the environment. In essence, this variant "protects" classifiers from being subsumed or replaced without adequate experience in biased environments, ensuring GA occurs less

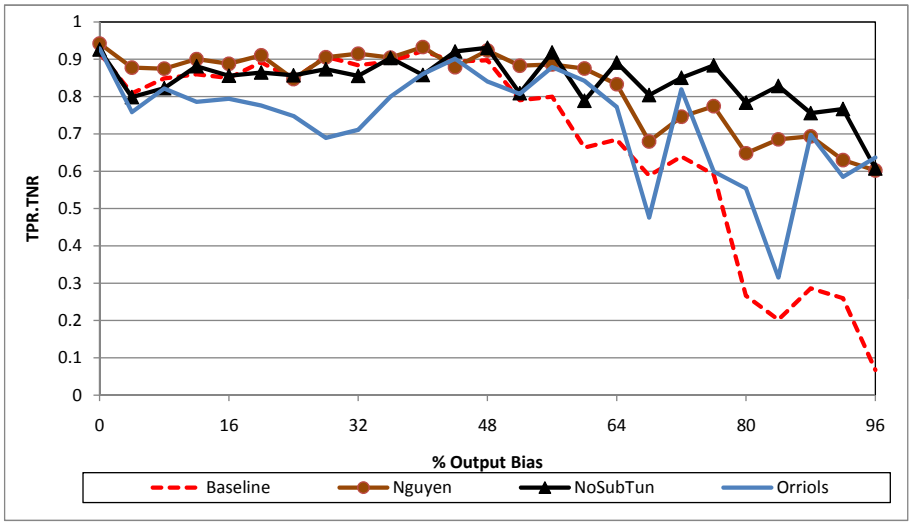

Figure 4: The effect of output bias for the RBF-6 problem in the last 1000 trials suggested by Orriols method

frequently in these situations.

The recommendations of Orriols-Puig and Bernadó- Mansilla also include running the XCSR algorithm for a significantly longer number of trials than those reported in Fig. 3 — the number of trials increasing proportionally with the output bias of the environment. Using their recommendations, Fig. 4 plots the performance of each algorithm variant after that many trials.

In comparison with Fig. 3, we see in Fig. 4 that the performance of all XCSR variants — especially in highly biased environments — significantly improves with the increased numbers of learning trials. However, the number of trials suggested by Orriols-Puig and Bernadó-Mansilla is very large, with over 2.2 million trials needed at the 96% output bias value. Recalling the desire for a fraud detection system to respond to adaptive adversaries in a timely manner, such large training times are likely to be infeasible, thus limiting the applicability of this approach.

Fig. 5 plots the average performance of the four LCS variants when we continue learning up to the maximum number suggested by Orriols — $2,205,000$ for 96% output skewness — reporting performance during the last $1,000$ trials of the run. As the figure shows, most of the algorithms learn a perfect answer to the problem, except when the output bias is very high, but performance is still acceptable in these cases. This is not surprising given large number of training trials.

Overall, the experiments reported in this section show that the baseline XCSR algorithm does not perform well on environments with highly biased

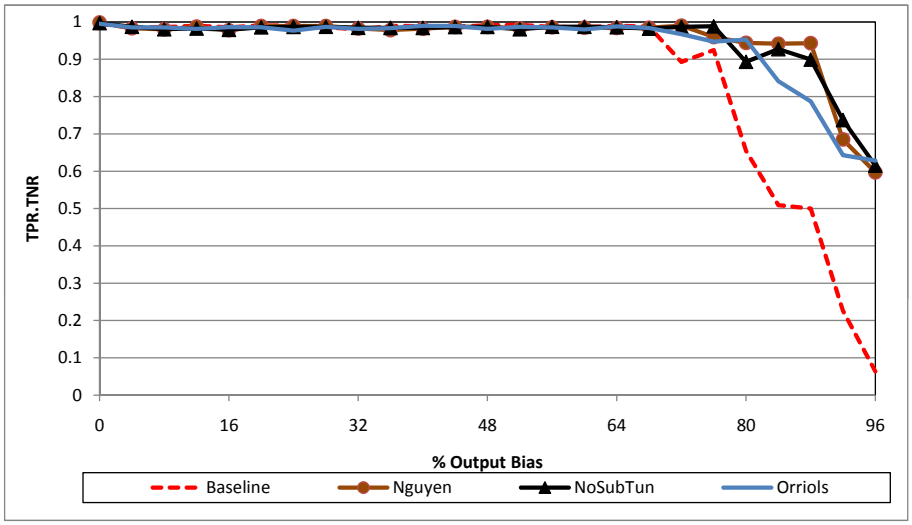

Figure 5: The effect of output bias for the RBF-6 problem in trials 2,204,000 to 2,205,000

outputs. In these cases, the algorithm needs to be tuned to the problem at hand and be "trained" for much longer periods of time.

4.5 Experiment 4: Adaptability in Incomplete Domains

This experiment evaluates the performance of XCSR on a 4-bit RBF problem in the presence of incomplete information, simulating the requirement for online learning in a fraud detection system. To achieve this, a part of the environment (a percentage of the input states) is kept hidden and only after some predetermined time (the midway point of the experiment) are these hidden states made visible; the hidden states being representative of some fraudulent activity occurring within the environment that is not present at the beginning of the "monitoring". That is, the algorithm is not exposed to these hidden states (and hence can not directly learn from these experiences) until the midway point of the experiment.

In this experiment, we hide different percentages of possible states of the 4-bit RBF problem from the learning algorithm for $25,000$ trials, reveal them at the $25,000$ point (allow the system to use them), and then the experiment continues for another 25,000 trials. Since we are dealing with continuous-valued inputs, the hiding is applied to the ranges of each component of the state. For instance, when the problem is the 4-bit version of the RBF, each of the four components of every state is a real value between 0 and 1. For an incompleteness level of 10%, ten percent of every component, whose range is determined at the

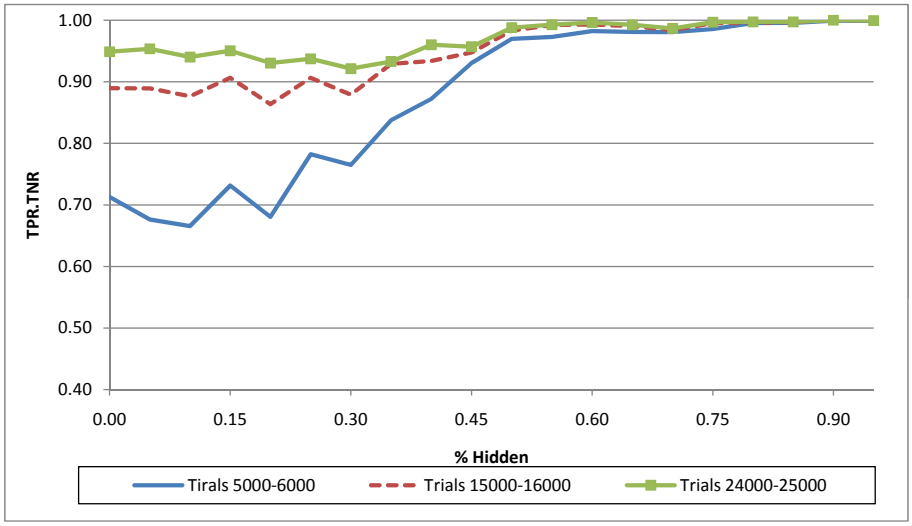

Figure 6: The effect of hidden states for the RBF-4 problem before revelation of hidden states

beginning of each experiment randomly, is hidden.

Fig. 6 reports the results of this experiment at three different time frames before revealing the hidden states. Fig. 7 similarly reports the results in three time frames after the revelation of the hidden states. These experiments were repeated 20 times and the results averaged.

Fig. 6 shows that before the revelation of the hidden states, XCSR is able to obtain near perfect performance on highly incomplete environments due to the small size of the learning space. Fig. 7 shows that after the revelation of the hidden states, this performance sharply declines shortly after their inclusion. However, by continuing the learning, the LCS can improve its understanding of the environment. Another observation evident from Fig. 7 is that up to 30% incompleteness does not have a significant adverse effect on the performance after revelation as long as the algorithm has enough time to learn (about $40,000$ trials in the case of RBF-4). These results confirm that XCSR does not have a problem with over-fitting and is indeed capable of adaptively responding to changes in the environment.

4.6 Experiment 5: Noise

This experiment tests the effect of noise on performance of XCSR by introducing different levels of noise from 0% to 50% to the 4-bit RBF problem. Here, noise corresponds to the environment erroneously inverting the reward for a chosen action; i.e. returning 0 for 1000 (maximum payoff) and 1000 for 0.

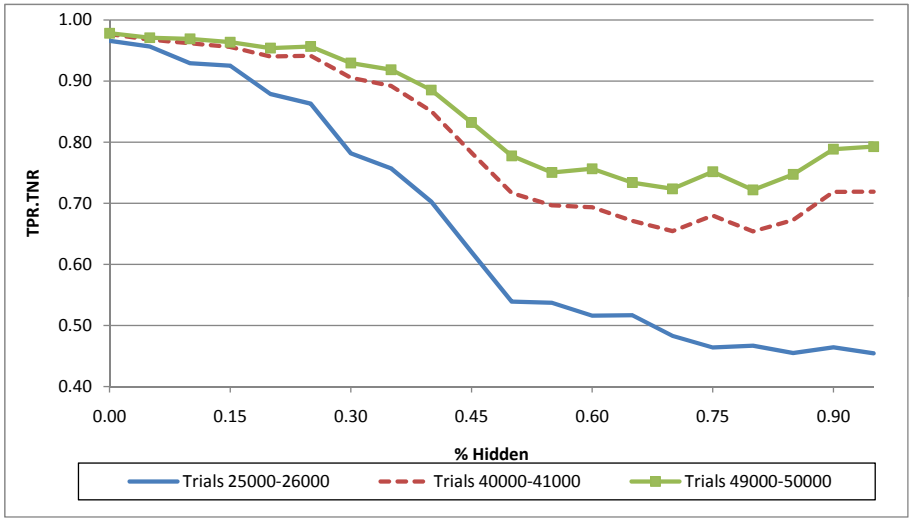

Figure 7: The effect of hidden states for the RBF-4 problem after revelation of hidden states

Fig. 8 shows the performance of XCSR in presence of different levels of noise in three periods: between 5,000 and 6,000 trials, between 15,000 and 16,000 trials, and between 35,000 and 36,000 trials. The results show that XCSR is reasonably tolerant to noise; performance only starting to significantly degrade for noise levels of 25% and higher.

4.7 Experiment 6: Real-World Data

In this experiment, we test our implementation on the KDD Cup 1999 intrusion detection data-set [21]. This data-set contains 4898430 experiences (connections) in the training file and 311029 in the test file. A smaller version of the training file, which contains a 10% subset of the training data at the same distribution as the complete file, was used in this work to reduce overall running times. For each file, every "connection" has a vector of 41 features and a label. The label determines if the connection is normal or an attack. There are 22 types of attack in the training file and 39 in the testing file. Every attack type belongs to one of four attack categories: DOS, R2L, U2R, and probing. Performance is measured using a cost per example value which is based on a cost matrix (note however, as per the contest rules, the cost matrix is hidden from the learning system during training). Although it is possible (and probably beneficial) to train XCSR using multiple passes of the training data, we only use one pass in order to evaluate the algorithm as an online learner.

Looking closely at the training and testing files, we noticed that connec-

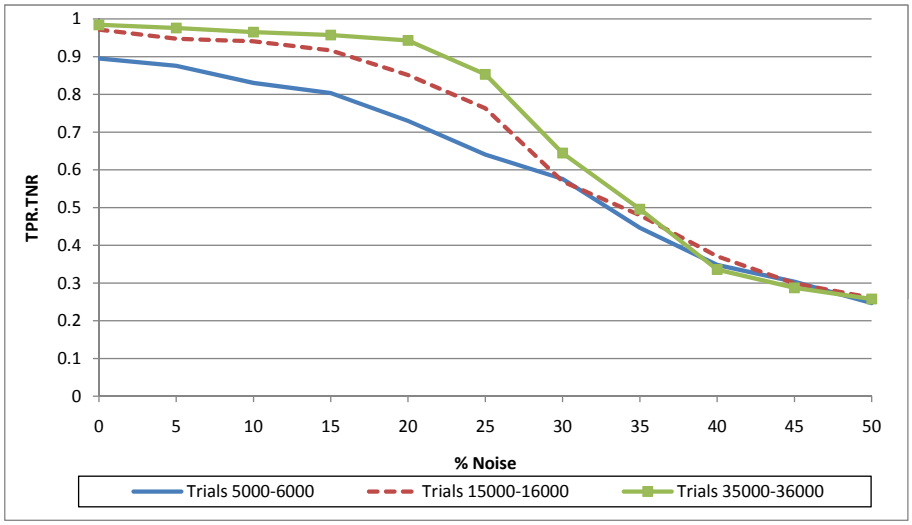

Figure 8: The effect of noise for the RBF-4 problem in three different stages of learning

tions come in "chunks"; that is, thousands of normal connections, followed by thousands of one type of attack with little interleaving of attacks and normal connections. This gives a temporal aspect to the problem which acts as an advantage for an LCS. When observing a series of connections belonging to the same class, the LCS quickly develops very good classifiers for that niche and hence, will classify the next block of connections correctly. In order to avoid this bias, we also ran the experiments on randomized versions of the training and testing files (where in each file experiences are randomly interleaved).

In our implementation, after receiving a connection, the system normalizes the values between -1 and 1. During the training phase, explore is used in which there is a 50% chance of selecting the class randomly (for exploration purposes). During the test phase, explore and exploit are interleaved, but during explore phases, all classes are selected deterministically (and not randomly). Also, we ran experiments in which during the test phase, learning is disabled (and only exploit is used). As expected, this decreases overall performance. Table 1 summarizes the results of these experiments.

As Table 1 shows, XCSR performs best when the learning is allowed during the test phase (feedback is provided) and test file is not randomized. However, it still performs better than the KDD cup 1999 winner if the test file is randomized provided the system is able to learn, via online learning, during the test phase. When learning is not allowed during testing, the performance of XCSR is still reasonable, but not as good as the KDD cup 1999 winner. Note that while XCSR has seemingly not learned as accurate a model as the KDD cup 1999 winner,

Table 1: Cup winner and XCSR performances on KDD Cup 1999 data-set

Algorithm	CPE	Accuracy
KDD Cup 1999 Winner	0.2331	92.71%
LCS, original test file, no online learning	0.7254	67.65%
LCS, randomized test file, no online learning	0.3505	84.00%
LCS, original test file, online learning	0.0917	96.48%
LCS, randomized test file, online learning	0.1613	95.19%

XCSR was not tuned for this problem at all. We plan to explore techniques for improving the underlying model in future work.

Two metrics are used for evaluation of performance. The first one is *accuracy*, which is the percentage of the correct predictions made by the system and is calculated using true positive, true negative, false positive and false negative values: $Accuracy = (TP + TN)/(TP + FN + FP + TN)$. The second metric is *Cost Per Example* (CPE) and is calculated using the following formula [24]:

$$CPE = \frac{1}{N} \sum_{i=1}^{m} \sum_{j=1}^{m} CM(i,j) * C(i,j) \quad (1)$$

where CM and C are the Confusion Matrix and the Cost Matrix respectively, N represents the total number of test instances and m is the number of the classes. In a confusion matrix, columns correspond to the predicted classes and rows correspond to the actual classes. $CM(i,j)$ represents the number of instances that belong to class i and are classified as a member of class j. Hence, an ideal classifier will have a confusion matrix with zero values for all entries except the primary diagonal. The Cost Matrix has a similar definition and represents the cost penalty for misclassifying examples [21].

Tables 2 and 3 show the detailed results of the winner of the KDD Cup 1999 and XCSR respectively, in the form of confusion matrices. As these tables show, XCSR performs better on most of the classes. A very important observation is the significantly better performance of XCSR over cup winner on classes 3 and 4 which are the rare classes here (in the test file, class 3 has 228 instances, class 4 has 16,189 instances, versus 229,853 instances of class 2). This is particularly valuable in fraud detection in which fraudulent instances often are rare and most methods have difficulty detecting them.

It should be noted that this real-world problem has most of the common properties of fraud discussed in Section 2:

1. Input bias. Most fields of the corpus have rare inputs. For example, the *su_attempted* field in the test file contains 311024 connections with a value of 0, only 1 connection with a value of 3, and 2 connections with a value of 2. The training file has a similar distribution.

2. Output bias. For instance, in the training file, 3884410 connections belong to category 2 and only 52 connections belong to category 3.

Table 2: KDD Cup 1999 Winner Performance. CPE: 0.2331, Accuracy: 92.71%

		predicted					
		0	1	2	3	4	
actual	0	60262	243	78	4	6	99.50%
	1	511	3471	184	0	0	83.30%
	2	5299	1328	223226	0	0	97.10%
	3	168	20	0	30	10	13.20%
	4	14527	294	0	8	1360	8.40%
		74.60%	64.80%	99.90%	71.40%	98.80%	

Table 3: XCSR (with on-line learning) Performance. CPE: 0.0917, Accuracy: 96.48%

		predicted					
		0	1	2	3	4	
actual	0	56451	141	67	49	3885	93.20%
	1	1018	3125	11	10	2	75.00%
	2	797	395	228608	24	29	99.50%
	3	105	27	14	81	1	35.50%
	4	4029	165	108	74	11813	73.00%
		90.50%	81.10%	99.90%	34.00%	75.10%	

3. Adaptive adversaries and online learning. There are 14 new attack types in the test file which do not occur in the training file.

4. Concept drift. One attack type in the training file changes category in the test file.

5. Misclassification costs. In the cost matrix, the cost of misclassifying some attack classes is higher than others.

In conclusion, XCSR is an effective learning methodology for complex real world data, and is especially very effective at online learning.

5 Conclusions and Future Work

Fraud detection is a challenging problem due to the adversarial nature of perpetrators and the rarity of fraudulent events among the magnitude of legitimate activity. In this research, we detailed seven general characteristics of fraud and motivated the use of LCSs for learning in such environments. Through the use of abstract problems which enable systematic analysis of these characteristics, we examined the behaviour of XCSR in the presence of these challenges. Our experiments suggest an LCS is capable of overcoming them. Also, a real-world data-set was used to evaluate the performance of XCSR in realistic situations. These experiments showed that XCSR is an effective learning methodology for

Table 4: Experiments Summary

Properties	Exp	LCS Performance
Biased Classes	1,2,3,6	Requires tuning and more trials, but in general, overcomes these challenges
Online Learning	4,6	Very responsive and quick to adapt
Adaptive Adversaries	6	Inherently an online-learner
Concept Drift	6	Inherently an online-learner
Misclassification Costs	6	Requires incorporating cost into rule updating procedure
Noise	5	Sufficiently robust
Fast Processing over Large Data Sizes	6	Responsive because of online learning (although larger number of dimensions require larger population sizes)

complex real-world data, and is especially very effective at online learning. Table 4 shows a summary of the common properties of fraud and the matching experiments in this work.

To be an effective method for automated fraud detection, LCSs must be accurate in highly biased-output environments and adapt to change in a timely manner; although we achieved an acceptable performance in such environments, this typically comes at the cost of large numbers of training trials which may not be feasible in real-world problems. Future work will involve investigating methods to improve performance in this setting.

For the real-world data-set, the randomization of the test data significantly decreased the performance for the learning algorithm. This indicates that there is some temporal aspect to the data that could be exploited. Other future work will include: examining the complexity of functions as well as their bias and skewness; investigating methods to automatically tune learning parameters on the fly; and examining the connection between dimensionality and numerosity.

Acknowledgments

The first author would like to acknowledge the financial support provided by the Robert and Maude Gledden Scholarship.

References

[1] Personal fraud, 2007. Technical Report 4528.0, Australian Bureau of Statistics, 2008.

[2] Jaume Bacardit, Bernadó-Mansilla Ester, and Martin V. Butz. Learning classifier systems: looking back and glimpsing ahead. In *10th International Workshop on Learning Classifier Systems*, pages 1–21. Springer, 2008.

[3] Angelo Baglioni and Umberto Cherubini. Accounting fraud and the pricing of corporate liabilities: Structural models with garbling. *SSRN eLibrary*, 2007.

[4] Larry Bull. *Applications of Learning Classifier Systems*. Springer, 2004.

[5] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. Technical Report 2000017, Illinois Genetic Algorithms Laboratory, 2000.

[6] James Dudley, Luigi Barone, and Lyndon While. Multi-objective spam filtering using an evolutionary algorithm. In *Congress on Evolutionary Computation*, pages 123–130, 2008.

[7] T. Fawcett, I. Haimowitz, F. Provost, and S. Stolfo. Ai approaches to fraud detection and risk management. *AI Magazine*, 19(2):107–108, 1998.

[8] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *WWW2007*, pages 649–656. ACM Publishers, 2007.

[9] Rod Haggarty. *Discrete Mathematics for Computing*. Addison Wesley, 2001.

[10] John H. Holland. Adaptation. *Progress in Theoretical Biology*, 4:263–293, 1976.

[11] Tim Kovacs. Learning classifier systems resources. *Journal of Soft Computing*, 6(3–4):240–243, 2002.

[12] Thach Huy Nguyen, Sombut Foitong, Phaitoon Srinil, and Ouen Pinngern. Towards adapting xcs for imbalance problems. In *PRICAI '08*, pages 1028–1033. Springer, 2008.

[13] Terri Oda and Tony White. Immunity from spam: an analysis of an artificial immune system. In *4th International Conference on Artificial Immune Systems*, pages 276–289. Springer, 2005.

[14] Albert Orriols-Puig and Ester Bernadó-Mansilla. Evolutionary rule-based systems for imbalanced data sets. *Soft Computing*, 13(3):213–225, 2008.

[15] Clifton Phua, Vincent Lee, Kate Smith-Miles, and Ross Gayler. A comprehensive survey of data mining-based fraud detection research. Technical report, Monash University, 2005.

[16] D. Sculley and Gabriel M. Wachman. Relaxed online SVMs for spam filtering. In *30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 415–422. ACM Publishers, 2007.

[17] Alexander K. Seewald. An evaluation of naïve Bayes variants in content-based learning for spam filtering. *Intelligent Data Analysis*, 11(5):497–524, 2007.

[18] Kamran Shafi and Hussein A. Abbass. Biologically-inspired complex adaptive systems approaches to network intrusion detection. *Information Security Technical Report*, 12(4):209–217, 2007.

[19] Kamran Shafi, Tim Kovacs, Hussein A. Abbass, and Weiping Zhu. Intrusion detection with evolutionary learning classifier systems. *Natural Computing*, 8(1):3–27, 2009.

[20] Olivier Sigaud and Stewart W. Wilson. Learning classifier systems: a survey. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 11(11):1065–1078, 2007.

[21] SJ Stolfo et al. KDD cup 1999 dataset. UCI KDD repository. http://kdd.ics. uci. edu, 1999.

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

[23] Kreangsak Tamee, Pornthep Rojanavasu, Sonchai Udomthanapong, and Ouen Pinngern. Using self-organizing maps with learning classifier system for intrusion detection. In *PRICAI '08*, pages 1071–1076. Springer, 2008.

[24] Adel Nadjaran Toosi and Mohsen Kahani. A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Computer Communications*, 30(10):2201–2212, 2007.

[25] S. Vanderlooy, I.G. Sprinkhuizen-Kuyper, and E.N. Smirnov. Reliable classifiers in ROC space. In *Proceedings of the 15th BENELEARN Machine Learning Conference*, pages 27–36. Citeseer, 2006.

[26] V. Vatsa, S. Sural, and AK Majumdar. A game-theoretic approach to credit card fraud detection. *Lecture notes in computer science*, 3803:263–276, 2005.

[27] Gary M. Weiss. Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7–19, 2004.

[28] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[29] Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems, From Foundations to Applications*, pages 209–222. Springer, 2000.

[30] Ching-Tung Wu, Kwang-Ting Cheng, Qiang Zhu, and Yi-Leh Wu. Using visual features for anti-spam filtering. In *International Conference on Image Processing*, volume 3, pages 509–512. IEEE Publishers, 2005.

[31] Yue Yang and Sherif A. Elfayoumy. Anti-spam filtering using neural networks and Baysian classifiers. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 272–278. IEEE Publishers, 2007.