ABSTRACT

We study two existing Learning Classifier Systems (LCSs): XCS, which has a complete map (which covers all actions in each state), and XCSAM, which has a best action map (which covers only the highest-return action in each state). This allows XCSAM to learn with a smaller population size limit (but larger population size) and to learn faster than XCS on well-behaved tasks. However, many tasks have difficulties like noise and class imbalances. XCS and XCSAM have not been compared on such problems before. This paper aims to discover which kind of map is more robust to these difficulties. We apply them to a classification problem (the multiplexer problem) with class imbalance, Gaussian noise or alternating noise (where we return the reward for a different action). We also compare them on real-world data from the UCI repository without adding noise. We analyze how XCSAM focuses on the best action map and introduce a novel deletion mechanism that helps to evolve classifiers towards a best action map. Results show the best action map is more robust (has higher accuracy and sometimes learns faster) in all cases except small amounts of alternating noise.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning—knowledge acquisition, concept learning

General Terms
Algorithms, Experimentation, Performance

Keywords
Learning Classifier System, XCS, XCSAM, complete action map, best action map, classification

1. INTRODUCTION

Learning Classifier Systems (LCSs) [7] are rule-based learning systems that evolve general rules (called classifiers) using Genetic Algorithms [6] to represent human-readable solutions. The accuracy-based LCSs [21] [1] are the most popular classifier system model so far and have been successfully applied to a wide range of problem domains [2] such as classification [8], regression [5], and sequential decision making problems [4]. Accuracy-based LCSs can be classified to two types according to a difference of learning mechanisms that they employ: Accuracy-based Reinforcement Learning Classifier Systems (ARLCS) which employ reinforcement learning (RL) [18], i.e., the XCS classifier system [21], and Accuracy-based Supervised Learning Classifier Systems (ASLCS) which employ supervised learning instead of RL, i.e., the UCS classifier system [1]. While ASLCS is a useful and robust model in supervised learning tasks [14], it is not designed to be suitable for RL tasks.

When we design ARLCS for given problems, we have to choose either one of two learning strategies: a complete action map which predicts the rewards of all actions in each state and a best action map which in contrast covers only the highest-return action in each state. Most real world applications have difficulties like noise and class imbalances in their state-space and reward functions. Accordingly, a key which can help to design LCS is to understand which strategy (complete or best-action map) is more robust to which kind of difficulties.

Orriols-Puig shows that UCS (with a best action map) can derive better performance than XCS (with a complete map) on a class-imbalanced classification problem [14]. Stone and Bull show the ZCS classifier system [20] (which is a strength-based LCS and employs a best action map) exceeds performance of XCS on noisy problems [17]. Also, Fani shows a revised ZCS for data mining can derive higher classification accuracy than XCS in a classification problem with real world data [19]. These results may suggest the best action map is a more robust strategy than the complete action map. However, ZCS and UCS are not ARLCS. Specifically, as noted previously, UCS can successfully identify a highest-return action (i.e., a correct class) using supervised data. ZCS, like ARLCS, employs reinforcement learning, but it may not correctly generalize classifiers because of its strength-based fitness. Strength-based fitness directly estimates reward (i.e.,
high fitness means high estimated reward). This is useful to identify the best action map but is less useful to find accurate classifiers. In contrast, the accuracy-based fitness of ARLCS is calculated from an error in estimated prediction; it is useful to accurately generalize classifiers but is very sensitive to noise (especially noise which is added to a reward). This means accuracy-based fitness mistakenly leads to incorrect generalization in noisy problems. Accordingly, best action maps may not be robust in class-imbalanced, noisy and real-world classification.

In summary, previous studies have found ZCS and UCS were more robust than XCS on class-imbalanced and noisy tasks, but it is not clear if this is due to their best-action maps, or due to their definition of fitness. This paper compares complete and best maps in systems with the same definition of fitness; specifically, we compare two ARLCSs; XCS and XCSAM [12] [11]. XCS evolves a complete action map while XCSAM is a variation of XCS which evolves best action maps. We apply them to versions of a classification problem (the multiplexer) with class imbalance, Gaussian noise or alternating noise. We also compare them on real world data. In section 5 we analyze the results of the XCSAM on problems with noise and class-imbalance, and real world data. In section 6 we show experimental results for XCSAM with the same action and the symbol ‘#’ is the don’t care symbol which matches all input values (i.e., 0 or 1). Classifiers consist of a condition, an action, and four main parameters [21, 3]; (i) the prediction $p$, which estimates the relative payoff that the system expects when the classifier is used; (ii) the prediction error $\epsilon$, which estimates the error of the prediction $p$; (iii) the fitness $F$, which estimates the accuracy of the payoff prediction given by $p$; and (iv) the numerosity $num$, which indicates how many copies of classifiers with the same condition and the same action are present in the population.

2.2 Mechanism

At time $t$, XCS builds a match set $[M]$ containing the classifiers in the population $[P]$ whose condition matches the current sensory input $s_t$, if $[M]$ does not contain all the feasible actions covering takes place and creates a set of classifiers that matches $s_t$ and cover all the missing actions. This process ensures that XCS can evolve a complete mapping so that in any state it can predict the effect of every possible action in terms of expected returns.\(^1\)

For each possible action $a$ in $[M]$, XCS computes the system prediction $P(s_t, a)$ which estimates the payoff that XCS expects if action $a$ is performed in $s_t$. The system prediction $P(s_t, a)$ is computed as the fitness weighted average of the predictions of classifiers in $[M]$ which advocate action $a$:

$$
P(s_t, a) = \sum_{cl_t \in [M]} p_k \times \frac{F_k}{\sum_{cl_t \in [M]} F_i}$$

where, $[M](a)$ represents the subset of classifiers of $[M]$ with action $a$, $p_k$ identifies the prediction of classifier $cl_k$, and $F_k$ identifies the fitness of classifier $cl_k$. Then XCS selects an action to perform; the classifiers in $[M]$ which advocate the selected action form the current action set $[A]$.

The selected action $a_t$ is performed, and a scalar reward $r_{t+1}$ is returned to XCS together with a new input $s_{t+1}$. When the reward $r_{t+1}$ is received, the estimated payoff $P(t)$ is computed as follows:

$$
P(t) = r_{t+1} + \gamma \max_{a \in [M]} P(s_{t+1}, a)$$

where $\gamma$ is the discount factor [18], which is 0 for single-step tasks such as those in this work. Next, the parameters of the classifiers in $[A]$ are updated in the following order [3]: prediction, prediction error, and finally fitness. Prediction $p$ is updated with learning rate $\beta$ ($0 \leq \beta \leq 1$):

$$p_k \leftarrow p_k + \beta(P(t) - p_k)$$

Then, the prediction error $\epsilon$ and classifier fitness are updated with the absolute accuracy $\kappa$:

$$\kappa_k \leftarrow \kappa_k + \beta(|P(t) - p_k| - \kappa_k)$$

$$\kappa_k = \begin{cases} 1 & \text{if } \kappa_k \leq \kappa_0, \\ \alpha(\kappa_k/\kappa_0)^{-\nu} & \text{otherwise.} \end{cases}$$

$$F_k \leftarrow F_k + \beta\left(\frac{\kappa_k \times \text{num}_k}{\sum_{cl_j \in [A]} \kappa_j \times \text{num}_j} - F_k\right)$$

The GA is applied to classifiers in $[A]$ [21]. It selects two classifiers, copies them, and with probability $\chi$ performs crossover on the copies; then it mutates each element in its condition with probability $\mu$.

3. XCSAM CLASSIFIER SYSTEM

XCSAM [21] evolves classifiers that represent complete action maps which are composed of all available actions in every possible situation, but the complete action map increases the population size required to evolve an optimal solution. UCS [1] deals with the best action map and is trained using supervised learning, unlike XCS which uses reinforcement learning. Therefore, UCS is limited in that it cannot be applied to reinforcement learning applications. XCS with Adaptive Action Mapping (or XCSAM), provides a trade-off between XCS and UCS. Like XCS, it can solve both reinforcement learning (multi-step) and supervised classification (one-step) problems; like UCS, XCSAM does not evolve a

\(^1\)In the algorithmic description [3], covering is activated when the match set contains less than $\theta_{\text{num}}$ actions; however, $\theta_{\text{num}}$ is always set to the number of available actions so that the match covers all the actions.
complete action map but tries to focus on the best actions only while learning proceeds. XCSAM extends the original XCS by (i) including a mechanism to adaptively identify the redundant actions from those that should be included in a best action map and by (ii) getting rid of redundant actions while still exploring the space of viable optimal solutions.

3.1 Identification of the Best Action Map

Firstly, XCSAM identifies the best action when the selected action is performed. In a typical reinforcement learning problem involving delayed rewards, the expected future reward at the current state \( \max P(s_t, a) \) (Equation 2), tends to be higher than the expected future reward at the previous state \( \max P(s_{t-1}, a) \) (because of the discount factor \( \gamma \)). Accordingly, since the action corresponding to a higher reward also tends to correspond to a shorter state sequence, the best actions will tend to have a value of \( \max P(s_t, a) \) larger than \( \max P(s_{t-1}, a) \). More precisely, \( \max P(s_{t-1}, a) \) converges to \( \gamma \max P(s_t, a) \) at the next state, while \( \max P(s_t, a) \) converges to the \( \max P(s_{t-1}, a)/\gamma \). Thus, when an executed action is a best action, in XCS the predictions of the accurate classifiers in \([\bar{A}]\) tend to converge to \( \max P(s_{t-1}, a)/\gamma \). For this reason, XCSAM can identify the actions that are likely to be part of the best action map by comparing \( \max P(s_t, a) \) against \( \zeta \times \max P(s_{t-1}, a)/\gamma \) (where \( \zeta \) is a learning rate added to guarantee convergence). If \( \max P(s_t, a) \) is greater than the threshold \( \zeta \times \max P(s_{t-1}, a)/\gamma \), then \( a \) is a good candidate for the best action map and should be maintained.

After having identified good action candidates for the best action map, XCSAM needs to adaptively identify classifiers that may be good candidate for the final best action map. Accordingly, XCSAM adds a parameter \( eam \) to classifiers, or effect of adaptive mapping, that for classifier \( cl \) is updated according to Equation 7, where \( nma \) represents the number of available actions. The value of \( eam \) of classifiers advocating the selected action converges to 1 if the classifier is a good candidate for the final best action map; otherwise, \( eam \) converges to \( nma \). Therefore, classifiers with a \( eam \) close to one are good candidates to represent the final best action map while classifiers with an \( eam \) close to \( nma \) are less likely to be maintained as they are probably advocating actions with lower expected return.

\[
em_a \leftarrow \begin{cases} 
 em_a + \beta (1 - em_a) & \text{if } \max P(s_t, a) \geq \zeta \times \max P(s_{t-1}, a)/\gamma \\
 em_a + \beta (nma - em_a) & \text{otherwise.} 
\end{cases}
\] (7)

3.2 Evolution on the Best Actions

To focus evolution on the best actions, XCSAM acts on the covering operator to prevent the generation of classifiers that are not likely to be included in the final solution. In particular, XCSAM tunes the covering threshold \( \theta_{nma} \) based on the actions’ predicted reward and the \( eam \) parameters. Initially, \( \theta_{nma} \) is set to the number of feasible actions (the same value as in XCS). When \([M]\) is generated, XCSAM computes the prediction array before covering is applied (whereas XCS computes it only after covering). Then, XCSAM computes the current \( \theta_{nma} \) as the average \( eam \) of the classifiers in \([M]\) weighted for the expected future return \( \max P(s_t, a) \). If the number of different actions in \([M]\) is smaller than the computed \( \theta_{nma} \), covering is called and the prediction array is computed again to update the prediction array.

If solving a single-step problem, as in the current work, XCSAM computes a selection array \( S(a) \) that associates a selection probability to each action as follows:

\[
S(a) = \frac{\sum_{k\in[M]} p_k \times F_k}{\sum_{k\in[M]} F_k} 
\] (8)

After action selection is performed, XCSAM generates both the action set \([A]\) (as XCS) and also the not action set \([\bar{A}]\) consisting of the classifiers in \([M]\) not advocating the selected action. The selection probability is calculated from the fitness \( F \) and parameter \( eam \) [13]. When the executed action is considered a candidate best action, the parents in the GA are selected from \([A]\) to promote the evolution of classifiers that are likely to be in the final best action map while the deleted classifiers are selected from \([\bar{A}]\) to get rid of classifiers that are not likely to be part of the final solution. Otherwise, if there is not enough information about the executed action, or \([\bar{A}]\) is empty, XCSAM applies deletion in \([P]\) as in XCS. When the executed action is not identified as a candidate best action, the parents are selected from \([\bar{A}]\) to explore the solution space even further and deletion is applied to the population as in XCS.

4. EXPERIMENT: XCS VS XCSAM

We apply XCS and XCSAM to the multiplexer and a classification problem with real world data. We add three kinds of problem difficulty to the multiplexer: class-imbalance, Gaussian noise or alternating noise. Class-imbalance can be observed in many real-world datasets such as oil spills in satellite images, failures in manufacturing process, and rare medical diagnosis [15]. Gaussian noise simulates real-world-like reward functions which sometimes return a different reward value when the same action is executed in the same state. Alternating noise simulates the dataset having several aliasing states which have the same input but different class. All three conditions can occur in real-world problems.

4.1 Experiments on Multiplexer Problems

4.1.1 Multiplexer problem and Experimental design

Multiplexer problem. The multiplexer (MP) [21] is defined over a binary string of \( k + 2^k \) bits; the first \( k \) bits represent an address pointing to the remaining \( 2^k \) bits. For instance, the 6-multiplexer function \( (k = 2) \) applied to the input string 110001 will return 1 as an answer (i.e., class), while when applied to 110110 it will return 0. We compare XCS and XCSAM using the 11MP \( (k = 3) \). Following XCS tradition [21] we generate inputs randomly, so the train and test sets are the same.

Experimental design. During learning problems, the system selects actions randomly from those represented in the match set. During evaluation problems, the system selects the action with highest expected return. When the system performs the correct action, it receives a 1000 reward, otherwise it receives 0. The genetic algorithm is enabled only during learning problems, and it is turned off during evaluation problems. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate. The performance is reported as the moving average over the last 50000 (the class imbalance) or 5000 (Gaussian noise and alternating noise) evaluation problems. All the plots are averages over 20 experiments.
in the class-imbalanced multiplexer problem, both systems have a tendency to overgeneralize classifiers for the majority class. Overgeneralized classifiers have relatively high fitness because inputs where they are wrong (minority-class inputs) happen with very low frequency. Since XCS deletes classifiers based on their fitness, XCS mistakenly keeps the overgeneralized classifiers as good classifiers. XCSAM also reaches optimality with \( \mu = 5 \) reaches 0.94 faster than XCS. With \( \mu = 6 \) and 7 XCSAM clearly has better performances than XCS. While in XCSAM overgeneralized classifiers are also generated, it can detect and delete them in contrast with XCS. This is because XCSAM deletes classifiers which are not candidates for the best action map when it identifies overgeneralized classifiers (which have incorrect best actions) using the reward signal. These results suggest that the best action map can enhance the performance of AR LCS in class-imbalanced problems, because the evolution of the best action map works to detect and quickly delete overgeneralized classifiers.

### 4.1.3 Gaussian noise

In our second set of experiments we add Gaussian noise to the reward on the (normal) multiplexer [21]. Gaussian noise is added with mean zero and standard deviation \( \sigma \) to the environment reward. The parameter settings are the same as the previous experiment. We set \( \sigma \) to different values: 100, 200, 300, 400 and 500. The maximum iteration is 500,000.

Figure 1-b) shows the performances of XCS and XCSAM on the 11-bit multiplexer problem with Gaussian noise. XCS reaches optimality with \( \sigma = 100 \) and 200. However, it suddenly fails to reach optimal performance with higher \( \sigma \). In contrast, XCSAM reaches optimality faster than XCS with \( \sigma = 200 \) and 300. Even with a large noise (\( \sigma = 400 \) and 500), XCSAM reaches about 0.96 and 0.85 respectively.
With Gaussian noise, the fitness of a classifier is much lower because the prediction error $\epsilon$ which is used to calculate the fitness becomes large. This means most classifiers are inaccurate, hence XCS cannot select a correct action using the inaccurate system predictions (which are calculated from the fitness). In XCSAM most classifiers are also inaccurate but XCSAM can identify classifiers which are candidates for the best action map. This is because the reward signal, even with added noise, is still useful to identify best action rules; a positive reward, which is returned when a best action is executed, is usually larger than a negative reward, which is returned when a not-best action is executed. That is, the reward signal still indicates the best action even with Gaussian noise. Hence, XCSAM can select the best action classifiers, which remain in the population, by deleting redundant classifiers which are not candidate for the best action map. These results suggest that the best action map contributes to stably derive high performance by deleting redundant classifiers in Gaussian noise where the positive reward is larger than the negative reward. In the next experiments we test the case of noise where the positive reward is smaller than a negative reward.

### 4.1.4 Alternating noise

In our third set of experiments we add alternating noise to the reward on the (normal) multiplexer problem. With alternating noise, the reward is swapped with a certain probability $P_{an}$. Specifically, the positive reward (1000) is replaced by the negative reward, which means this noise simulates a state space which includes aliasing states. For instance, an instance of 6MP “001000” has a best action “1”; however, when a learner executes an action “1”, 1-$P_{an}$% of all instances “001000” returns the positive reward 1000 but $P_{an}$% instance return the negative reward 0 even. The parameter settings are the same as the previous experiment. We use different $P_{an}$ to 0.10, 0.15, 0.20, 0.25 and 0.30. The maximum iteration is set to 500,000.

Figure 1-c) shows the performance of XCS and XCSAM on the 11-bit multiplexer problem with alternating noise. With $P_{an} = 0.10$ and 0.15, XCS reaches 0.98 and 0.95 respectively; it fails to reach optimal performance with $P_{an} = 0.20$, 0.25 and 0.30. The performance of XCSAM with $P_{an} = 0.10$ and 0.15 are worse than XCS, in fact they converge to 0.95 and 0.93 respectively. However, with $P_{an} = 0.20$, 0.25 and 0.30, XCSAM clearly improves on the performances of XCS. With alternating noise, XCSAM has a tendency to delete classifiers which are candidates for the best action map, because XCSAM mistakenly identifies them as redundant classifiers when the returned reward is 0 (i.e., reward 0 represents that executed action is not the best action). In contrast, XCS keeps them even with the added noise (XCS deletes classifiers based on their fitness), because they have relatively high fitness in the case of small noise. Hence, with small noise $P_{an} = 0.10$ and 0.15, the performance of XCSAM is lower than that of XCS. However, with large noises $P_{an} = 0.20$, 0.25 and 0.30, the fitness of classifiers is much lower, hence, XCS cannot select actions using the incorrect system prediction (which is the same reason of Gaussian noise).

### 4.2 Experiment on Real World Data

To evaluate performance of XCS and XCSAM on a real world application, we apply both systems to classification tasks with real world data from the UCI repository [16]. As shown in Table 1 we use 20 datasets. In the Table, #len denotes the length of input in each dataset and #cls denotes the number of classes in each dataset. As in previous work [1], if the attributes are binary, the data is codified in the ternary alphabet. If the features are real, the data is codified as an interval range. For simplicity, all real attributes are normalized to the range [0,1]. Nominal attributes are translated into numeric values, and so considered as real attributes. The datasets are run on a stratified ten-fold cross-validation test. To test the differences between the two systems, we use a paired t-test from the ten-fold cross-validation results. The parameter settings are the same as previously except the population size limit $N = 6400$. The maximum iteration is 100,000.

The classification accuracies of XCS and XCSAM are shown in Table 1. We note XCS is significantly better ($p<0.05$) than XCSAM on Balance S., Cmc and Hepatitis while XCSAM is significantly better Audiology and Soybean which have longer inputs and more classes than other datasets. Overall, we cannot find a clear advantage for either system.

### 5. ANALYSIS

To understand why XCSAM does not derive higher classification accuracies than XCS, we analyze how XCSAM focuses on the best action map. From the analysis, we introduce a deletion strategy for XCSAM which can help to evolve the best action map.

#### 5.1 Analysis

Specifically, we analyze the parameters $cam$ (effectiveness of adaptive action mapping) and $\epsilon$ (prediction error) of classifiers in the population when XCSAM is applied to the real dataset Vowel. The reason for focusing on Vowel is that XCSAM did not derive a better performance than XCS although Vowel has many classes like Audiology and Soybean.
where XCSAM clearly derived higher classification accuracy than XCS. In Vowel, the number of available actions \( n_{ma} \) is 11, which is the same as the number of classes of Vowel.

Figure 2 shows the classifiers in the population when XCSAM is applied to Vowel. The vertical and horizontal axes indicate the classifiers’ \( eam \) and \( \epsilon \) parameters. A classifier has the best action if its \( eam \) is close to 1, while a not-best action if its \( eam \) is close to 11 (\( n_{ma} \)). Similarly, a classifier is accurate if its \( \epsilon \) is close to 0, while inaccurate if its \( \epsilon \) is close to 500. For instance, classifiers which are placed at the left-top corner are accurate but not candidates for the best action map; classifiers which are placed at the left-bottom corner are accurate and candidates for the best action map. From figure a), at the 10,000th iteration, XCSAM generates redundant classifiers which are inaccurate (large \( \epsilon \)) or not candidates for the best action map (large \( eam \)). At the 50,000th iteration, XCSAM gradually generates classifiers which have large \( \epsilon \) but small \( eam \); on the other hand many redundant classifiers remain. At the 100,000th as the final iteration, XCSAM successfully learns several accurate classifiers (left-bottom corner) which have a best action (\( eam = 1 \)); however many inaccurate classifiers and redundant classifiers (left-bottom corner) still remain. This analysis suggests XCSAM can partially learn the best action map, but it has many redundant classifiers which are inaccurate or are not best action map candidates. Hence, XCSAM sometimes fails to efficiently evolve the best action map.

### 5.2 Deletion mechanism

From the analysis, to efficiently focus on the best action map, XCSAM needs to delete many redundant classifiers. However, using a steady-state GA, XCSAM deletes only two classifiers in each generation, hence many redundant classifiers remain in the population. Here, we introduce a new deletion mechanism that deletes more than two classifiers as necessary. Specifically, in XCSAM, \( \theta_{nma} \), which is calculated from the parameter \( eam \), is the number of actions which should be in the matchset \( [M] \). When \( \theta_{nma} = n_{ma} \) (number of action types), classifiers in \( [M] \) should not be removed because XCSAM does not detect the best action in its state. In contrast, when \( \theta_{nma} = 1 \) (which means XCSAM correctly identified the best action), classifiers will be redundant if \( [M] \) has more than one type of action. More generally, if the number of action types in \( [M] \) (\( n_{ma}([M]) \)) is larger than \( \theta_{nma} \), we can suspect there are some redundant classifiers in \( [M] \). For this reason, the new deletion mechanism deletes classifiers if \( n_{ma}([M]) > \theta_{nma} \). The new deletion mechanism is called when the executed action is identified as the best action. First, the average eam \( EAM(a) \) is computed from classifiers with action \( a \). Then XCSAM builds a maximum eam set \( [E] \) composed of classifiers in \( [M] \) which advocate the max\( EAM(a) \) (i.e., they are not even close to the best action map); and deletes all classifiers in \( [E] \) if \( n_{ma}([M]) > \theta_{nma} \). XCSAM repeats this process while \( n_{ma}([M]) > \theta_{nma} \).

### 6. EXPERIMENT: REVISED XCSAM

We apply the revised XCSAM with the new deletion mechanism to the multiplexer and real world data.

#### 6.1 Experiments on Multiplexer Problems

The experimental settings are the same as the previous experiments (Section 4.1). Figure 3 shows the performance of the revised XCSAM on the 11-bit multiplexer with class-imbalance, Gaussian noise or alternating noise. With class-imbalance (Figure 3 a)), the revised XCSAM reaches optimality with \( \sigma = 0 \), 3 and 4; with \( \sigma = 5 \) and 6 it reaches 0.99 and 0.96 respectively, which is higher accuracy than the standard XCSAM which reaches to 0.94 and 0.87 respectively (see Figure 1-a)). With Gaussian noise (Figure 3 b)), with \( \sigma = 100, 200, 300 \) and 400, the performance is as good as that of the standard XCSAM. With \( \sigma = 500 \), the revised XCSAM reaches 0.90 while standard XCSAM reaches 0.85 (see Figure 1-b)). In contrast, with alternating noise (Figure 3 b)), the performance of revised XCSAM is lower than the standard XCSAM. For instance, with \( \sigma_{an} = 0.10 \) and 0.15, the revised XCSAM reaches 0.89 and 0.84 respectively, while the standard XCSAM reaches 0.95 and 0.93. As noted in Section 4.1, with class-imbalance and Gaussian noise, since XCSAM can identify the best action map, the new deletion mechanism correctly deletes redundant classifiers. Hence, the revised XCSAM can successfully focus on the best action map and evolve them toward accurate classifiers. However, with alternating noise, XCSAM sometimes mistakenly identifies redundant classifier as the best action map due to noise. This causes the new deletion mechanism to mistakenly delete even classifiers which are candidates for the best action map. Hence, the revised XCSAM does not perform better than the standard XCSAM. This suggests that, if ARCS can evolve only best action maps and have no redundant classifiers in their population, the best action map is more robust than the complete action map for class-imbalance and Gaussian noise; in contrast, robustness with alternating noise is worse than the complete action map.
6.2 Experiment on Real World Data

We test the revised XCSAM on classifying the real world datasets from Section 4.2, using the same parameter settings. Table 2 shows the classification accuracy of XCS and revised XCSAM for the standard XCSAM and revised XCSAM. The revised XCSAM clearly improves on the classification accuracy of XCSAM (see Table 1). We also found seven positive significant differences of revised XCSAM from XCS. In particular, the revised XCSAM derives high classification accuracy on the datasets Audiology, Libras, Soybean and Vowel, where their input-lengths are relatively long and there are many possible classes. Additionally, since three positive significant differences of revised XCSAM from the standard XCSAM are noted, the revised XCSAM successfully improves on the performance of XCSAM on the real world data. From these results, the best action map is more effective on the classification of data where an input has many elements (i.e., it is high-dimensional) and many classes.

Finally, to understand how the revised XCSAM focuses on the best action map, we analyze the classifiers of revised XCSAM when it is applied to dataset Vowel; see figure 4. At the 10,000th iteration, like the standard XCSAM, revised XCSAM generates redundant classifiers which are inaccurate (large $\epsilon$) or not candidates for the best action map (large $eam$). At the 50,000th iteration, the revised XCSAM gradually learns only the classifiers which have small $eam$ parameters. Note in the standard XCSAM, many classifiers which have large $eam$ remain, but the revised XCSAM successfully deletes them. At the 100,000th iteration, many accurate classifiers (which have $\epsilon = 0$ and $eam = 1$) are generated and the classifiers which have large $eam$ do not remain. From these results, the revised XCSAM successfully deletes classifiers which are not best action map candidates, hence, the revised XCSAM can derive high performance.

7. SUMMARY AND CONCLUSION

This paper compares XCS [20] with XCSAM [11, 12] on multiplexer problems and classification of real world data. Specifically, firstly, we test both system on the multiplexer problem with the three problem difficulties: class-imbalance, Gaussian noise and alternating noise. Experimental results show that XCSAM can derive better performance (higher accuracy and sometimes faster learning) than XCS with class-imbalance and Gaussian noise, where XCSAM can identify classifiers which are best action map candidates. Hence, XCSAM can efficiently evolve classifiers toward a best action map. In contrast, XCSAM sometimes derives lower accuracy than XCS when the alternating noise is small. From analysing this we introduce a new deletion mechanism. While original XCSAM sometimes fails to focus on the best action map in real world data, we show our new deletion mechanism can focus on it with real world data. The revised XCSAM which employs the new deletion mechanism can derive higher accuracy than XCSAM on the class-imbalanced and Gaussian noise problems; however with alternating noise, its accuracy degrades more than XCSAM. On classification with real world data, the revised XCSAM can derive higher classification accuracy on data which have a high-dimensional input and many possible classes. These
result suggest that the best action map is more robust to the class-imbalance and Gaussian noise, since the evolution of best action maps works to detect and quickly delete over-generalized classifiers and classifiers which do not have a best action map. Note, despite these problem difficulties, the reward signal is still useful to guide the LCS to the best action. The alternating noise causes XCSAM to mistakenly identify a not-best action as best action. Accordingly, XCSAM deletes accurate classifiers which have a best action by wrongly identifying them as not-best actions. Hence, the best action map is less robust than the complete action map in noise where the best action is mistakenly evaluated as the not-best action, i.e., in aliased states.

Our conclusion is that the best action map is a powerful strategy in problems which distinctly return a positive reward for the best action with a negative reward for the not-best action (i.e., the positive reward is always larger than the negative reward). Additionally, if ARLCS can successfully focus on only the best action map, the best action map contributes to high performance on a complex problem which has a high-dimensional input and many possible classes. The complete action map is a better strategy than the best action map when given problems include the aliasing states. Results with RL problems could be different from classification problems but we leave this for future work.

8. REFERENCES


