A Modified XCS Classifier System for Sequence Labeling

Masaya Nakata
The University of Electro-Communications
Tokyo, Japan
m.nakata@cas.hc.uec.ac.jp

Tim Kovacs
University of Bristol
Bristol, UK
kovacs@cs.bris.ac.uk

Keiki Takadama
The University of Electro-Communications
Tokyo, Japan
keiki@inf.uec.ac.jp

ABSTRACT

This paper introduces XCS-SL, an extension of XCS for sequence labeling, a form of time-series classification where every input has a class label. In sequence labeling the correct class of an input may depend on data received on previous time stamps, so a learner may need to refer to data at previous time stamps. That is, some classification rules (called "classifiers" here) must include conditions on previous inputs (a kind of memory). We assume the agent does not know how many conditions on previous inputs are needed to classify the current input, and the number of conditions/memories needed may be different for each input. Hence, using a fixed number of conditions is not a good solution. A novel idea we introduce is classifiers that have a variable-length condition to refer back to data at previous times. The condition can grow and shrink to find a suitable memory size. On a benchmark problem XCS-SL can learn optimal classifiers, and on a real-world sequence labeling task, it derived high classification accuracy and discovered interesting knowledge that shows dependencies between inputs at different times.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*knowledge acquisition,concept learning*

General Terms

Algorithms, Experimentation, Performance

Keywords

Learning Classifier System, XCS, generalization, time series data, sequence labeling, classification

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2576768.2598352.

1. INTRODUCTION

Many real world applications, such as stock exchange, inventory control, and the observation of natural phenomena, generate time-series data, in which data points have a natural sequence [6]. Classification of time-series data has attracted great interest in the machine learning field. The goals of classification are to find a model or a function that classifies data to predefined classes [6] and to discover important patterns as knowledge that help to understand why data is classified to the class. For the latter case, while the model or function can be variously represented by *IF-THEN rules*, *decision trees*, *mathematical formulae* and *neural networks* [6], to be human-readable the representation should generate simple, compact, and understandable patterns. IF-THEN rules are a human-readable representation and they can be extracted directly from data using, for example, a Learning Classifier System (LCS) [7]. An LCS is an IF-THEN rule-based learning system based on a Genetic Algorithm [5], which has been successful as a classification tool [17] [18]. LCS can discover human-readable patterns by the generalization of rules (called "classifiers") using a simple symbol (i.e., don't care symbol #) to represent features which are not relevant to the classifiers. We study LCS for learning human-readable rules from time-series data.

In normal (i.e., not time-series) classification, each data point in a given dataset is an independent observation for an object, hence the dataset has no sequential or time-oriented relationships between data. Even if these are the relationships, i.e., the order of data in dataset is sequential or time-series, the conventional classification tools do not consider them to find patterns. Typical time-series data is a sequence of values or observed events [6] such as EEG (electroencephalography) data and spoken sound data as in [4] [9]. In typical time-series classification a whole sequence of data is given one class label, e.g., a whole series of EEG readings is given one label "normal" or "abnormal". Preen has proposed an extension of XCS for financial time-series data [13]. In contrast, this paper considers a kind of time-series classification called *sequence labeling* [10], in which every input in the sequence is given a class, e.g., in part of speech tagging each word in a sentence is classified as a noun, verb etc. As the field of time-series data mining matures, a wider range of real world sequence labeling applications has been studied, such as the observation of natural phenomena [22], part of speech tagging [15] and care support for human daily activity [8]. To classify the input to the correct class, a learner may need to refer back to inputs at previous time stamps to disambiguate the current input. However, in most real

world applications, the learner does not know where and how many previous inputs are needed to classify the current input. In addition, each input may need a different number of previous inputs.

This paper introduces an extension of the XCS classifier system [20] for sequence labeling called XCS-SL. A novel idea is a classifier that has a *variable-length* condition which is composed of sub-conditions as a kind of memory to refer back to previous inputs. The classifier condition can *grow* and *shrink* by evolution of classifiers to explore the suitable memory size for classifying the current input to the correct class. Accordingly, we also address the questions of how XCS-SL should evolve classifiers to efficiently find out the suitable memory size for each input and what are the advantages of variable-length conditions over fixed-length conditions, in which the memory size is a fixed value.

The remainder of this paper is organized as follows: In section 2 we explain the sequence labeling task in detail. In section 3 we present the related works of LCSs that use memory in brief and introduce the mechanism of XCS-SL. In section 4 we test XCS-SL on a novel benchmark problem (the Layered Multiplexer problem) and compare two evolutionary algorithms in XCS-SL. In section 5 we apply XCS-SL to Activity of Daily Living (ADL) recognition [12] as a real world application. Finally, in section 6, we conclude this paper by summarizing the contributions and discussing the possible future directions.

2. SEQUENCE LABELING

In normal (non-time-series) classification, the data consists of *input/class* pairs: $\langle \boldsymbol{x} : y \rangle$, where $\boldsymbol{x}$ is an input vector and y is a class of the data. For example, if we are predicting hobbies, the data could be in this format: $\langle$*male*, *twenties* : *likes-football*$\rangle$. Alternatively, if we are predicting engine failure from diagnostic tests the data could be in this format: $\langle 0.6, 0.5, 0.3 :$ *defect*$\rangle$. Typically, each element $x_i \in \boldsymbol{x}$ of the input is a different observed event from other elements such as $x_1 =$ *male* and $x_2 =$ *twenties*. The class of an input is not dependent on any other input since they are independently observed events. Hence the dataset has no sequential or time-oriented relationships between data.

In contrast, while data for time-series classification has input/class pairs, the input is a *sequence* of values from one event: $\langle (t_0, \boldsymbol{x}_0), (t_1, \boldsymbol{x}_1), \cdots, (t_n, \boldsymbol{x}_n) : y \rangle$ where t denotes a time stamp. Since the time-series data is the sequence of one event, all elements of the input are values of the same observed event at different time stamps e.g., $\langle$(t_0,*breakfast*), (t_1, *exercise*), (t_2, *shower*):*good-sleep*$\rangle$ or e.g. $\langle (t_0, 0.3), (t_1, 0.5), (t_2, 0.2)$:*defect*$\rangle$. Note that there is only one class label y which classifies the whole sequence.

Sequence labeling is a kind of time-series classification in which there is a class for *every* time stamp: $\langle (t_0, \boldsymbol{x}_0):y_0 \rangle$, $\langle (t_1, \boldsymbol{x}_1):y_1 \rangle$, $\cdots$, $\langle (t_n, \boldsymbol{x}_n):y_n \rangle$. In sequence labeling, the data is similar to both normal classification, since every input has a class, and to time-series classification, since the inputs represent a sequence. For example, the data can be: $\langle (t_0,$ *bathroom*$)$:*shower*$\rangle$, $\langle (t_1,$ *kitchen*$)$:*breakfast*$\rangle$, or e.g. $\langle (t_1, 0.4, 0.2)$:*buy*$\rangle$, $\langle (t_2, 0.6, 0.2)$:*sell*$\rangle$.

In predicting the class of an input, it may in general be useful to consider data from past or future events. For example, in part of speech tagging, to classify a word it may be useful to know the class of words that come both before and after the current word. We are interested in a specific

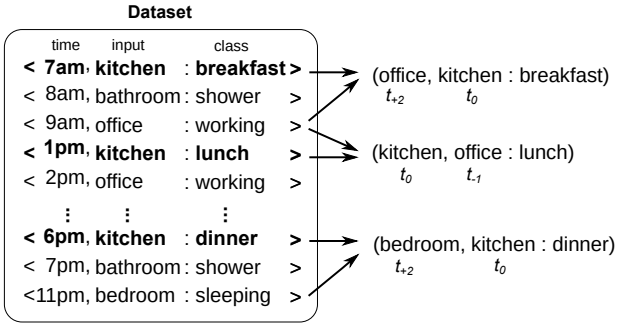


Figure 1: Example of data for sequence labeling.

sequence labeling application called Activity of Daily Living (see section 5). In this problem, like many problems, we have a training set of labeled data, but we also want to be able to classify new unlabeled data. For this reason we will learn patterns in the inputs but not in the actions of the learner, because on unlabeled data we do not know if the actions were correct. (In future work we will evaluate action patterns too.) In this work we do not use the time stamps except to order the inputs. We do not pass the time stamp as a feature to XCS-SL, so it cannot learn patterns such as "If time is between 2 and 4pm THEN...".

Figure 1 gives an example of how sequence information can help classify inputs. The left half of the figure shows part of an activity recognition dataset which records where a person was at a certain time and what activity they were doing. In this data the input "*kitchen*" is placed at different time stamps "*7am*", "*1pm*" and "*6pm*" but it has the different classes "*breakfast*", "*lunch*", or "*dinner*" respectively. Hence, the input "*kitchen*" does not unambiguously identify the current class. To borrow a term from reinforcement learning, we could say there are a set of activity classes ("*breakfast*", "*lunch*", and "*dinner*") which are *perceptually aliased* and which appear the same (as "*kitchen*") to the learner. In non-time-series classification, a learner considers only the current input, hence he probably fails to classify "*kitchen*" to the correct class. However, the learner can successfully classify it when considers current, previous and future inputs. That is, he can disambiguate the aliased input by considering inputs at other times. The right half of figure 1 and the arrows in the middle of the figure show combination of inputs which can identify the current activity (class) correctly. For example, if the person is in the kitchen at the current timet_0 and they are in the office at time t_{+1} then we predict they are having breakfast at t_0. While many patterns can be accurate, optimal patterns should be composed of minimum elements. For instance, the pattern for the class "*lunch*" may be $\{(t_{+1},$ *office*$)$, $(t_0,$ *kitchen*$)$, $(t_{-1},$ *office*$)$, $(t_{-2},$ *bathroom*$)\}$ but this is not the minimal representations. A minimal one is $\{(t_0,$ *kitchen*$)$, $(t_{-1},$ *office*$)\}$.

A difficulty of sequence labeling is that the leaner does not know where and how many previous and future inputs are needed to classify the current input. The learner explores many possible patterns to find optimal patterns, hence it may need many memories to refer back or forward to data at different time stamps. In many tasks, such as online learning, we want to predict the current or future class strictly from the past, and so we consider only past data.

3. XCS FOR SEQUENCE LABELING

Classifiers require memory to refer back to the previous data to find patterns in sequence labeling tasks. Specifically, the additional memories can be added to the "IF" part (i.e., condition C). Some LCS papers e.g. [11] [19] have added memory to classifiers, but to our knowledge they were all used to solve POMDP (Partially Observable Markov Decision Process) or Semi-MDP problems, which are sequential decision making tasks, unlike the classification task in this paper. These works used small numbers of memories, or short memory lengths (a few time steps), but we are interested in more difficult problems which need a lot of memories, some of which must be quite old. Also, most previous methods used fixed-length memories, which, as we will see, as not as suitable as variable-length.

One possible way to represent the condition structure is, like the previous works [11] [19], with a *fixed-length condition* where the memory size has a fixed value. In our system each memory is sub-condition C_n for the corresponding previous time t_{-n} from the current time t_0. Accordingly, classifiers can be represented as $\{C_0, C_{-1}, \cdots, C_{-M}$:A$\}$, where A is a class ("THEN" part) and M is the maximum memory size. For example, some classifiers cl with M=2:

$$cl_{11} = \{kitchen,\ office,\ bathroom{:}lunch\}$$
$$cl_{12} = \{kitchen,\ office,\ \ \#\ \ :lunch\}$$

While the classifier cl_{11} is not the minimum representation for the class "*lunch*" (see Figure 1), by employing the *don't care symbol* # which matches any input, we can represent the more general and optimal classifier cl_{12}. The limitation of a fixed-length condition is that it is requires to set a fixed value although the LCS does not know how many memories are needed to find disambiguating patterns. If we underestimate the memory required then classification accuracy will suffer. If we overestimate it then the LCS needs a huge population size to explore many possible patterns, which needs a long run-time and a lot of training.

The novel idea introduced here is a *variable-length* condition so the memory size can be changed adaptively to fit the memories that are needed for each input. As a similar idea, a *chain* of classifiers which is a kind of variable length-condition is introduced by DACS [3] and CCS [16]. Classifiers can be connected with other classifiers as a chain. Our idea also requires the maximum memory size as a fixed value, but the memory size of classifier can reduce in during learning. For example new classifiers can be:

$$cl_{31} = \{kitchen,\ office,\ bathroom{:}lunch\}$$
$$cl_{32} = \{kitchen,\ office{:}lunch\}$$

Variable-length conditions can be not only the same the fixed-length conditions (classifier cl_{31}) but also more compact than fixed-length conditions (classifier cl_{32}). Accordingly, the classifier is required to explore the suitable memory size for each input. Classifiers with fewer conditions are more general and we want to find compact conditions that have as few memories as possible.

This paper proposes an extension of the XCS classifier system [20] for sequence labeling (XCS-SL), which employs classifiers that have variable-length conditions. In XCS-SL, classifiers can *grow* and *shrink* by evolution to explore a suitable memory size for data that are matched to its classifier condition. Hence, the evolution mechanism of XCS-SL is important in learning optimal memory patterns. XCS-SL

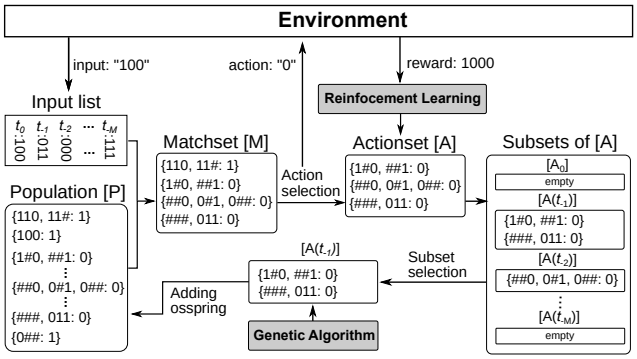


Figure 2: Overview of XCS-SL

almost works the same as standard XCS [2] but some mechanisms in the performance and the discovery components [2] are modified. In the remainder of this section we describe classifiers and the extended components. We also explain *subsumption* [21] and the *shrinker* mechanism we introduce, which can help to find compact conditions.

3.1 XCS-SL Classifiers

Like standard XCS classifiers [2], classifiers in XCS-SL consist of a condition C, an action A, and four main parameters [2]. Additionally, in XCS-SL, classifiers have a new parameter which is the memory size m to determine the condition length; the condition is composed of sub-conditions $C_0, C_{-1}, \cdots, C_{-m}$. Each sub-condition C_{-n} corresponds to the input at the time stamp t_{-n}. The memory size is determined and fixed when the classifier is generated but the maximum memory size for all classifiers M is set to a fixed value.

3.2 XCS-SL Mechanisms

This section details how the mechanisms of XCS-SL are different from those of XCS.

3.2.1 Performance component

As shown in Figure 2, at the current time t_0, XCS-SL stacks the input to the *input list*. When the number of inputs in the list is larger than the maximum memory size M, XCS-SL deletes the input at the oldest time stamp t_{-M}. Next, XCS-SL builds a *match set* [M] containing the classifiers in the population [P] whose sub-conditions each match C_{-n} the stacked input at the corresponding time t_{-n}. If [M] does not contain all the possible actions *covering* [2] takes place and generates classifiers; their memory size m is set uniform randomly but the maximum value is the number of inputs in the input list. Each sub-condition C_{-n} is copied from the corresponding input at the time stamp t_{-n} but each element of the sub-condition is replaced by the don't care symbol # with a probability $P_{\#}$. From here, XCS-SL works the same as XCS in the performance component. Specifically, for each possible action in [M], XCS-SL computes the *system prediction*, which estimates the payoff that XCS-SL expects if action a is performed on the current input (see [2] for more detail). Then, XCS-SL selects an action to perform; the classifiers in [M] which advocate the selected action form the current *action set* [A]. Finally, the selected action is performed, and a scalar reward is returned to XCS-SL together with an input at a next time stamp at t_{+1}. After

the performance component, the reinforcement component [2] using Reinforcement Learning is performed the same way as in XCS to update the classifier parameters.

3.2.2 Discovery component

After the reinforcement component is performed, XCS-SL evolves classifiers using a Genetic Algorithm (GA) [5]. In the sequence labeling, each input can have its own suitable memory size (i.e., each input may need a different number of previous inputs). Hence, XCS-SL is required to evolve classifiers which have suitable memory size. Accordingly, XCS-SL builds *subsets* $[\mathrm{A}(t_{-n})]$ of the action set which each consist of classifiers in [A] whose memory size m is equal to n. Then XCS-SL selects one subset from among the subsets $[\mathrm{A}(t_0)] \cdots [\mathrm{A}(t_{-M})]$ to perform the GA on. After selection, the GA is applied to classifiers in the selected subset. Two offspring are generated as copies of two selected parents and the crossover and mutation operators are applied to the offspring with probability χ and μ respectively. In crossover, each sub-condition is recombined with the corresponding sub-condition of the other offspring. Mutation changes elements in each sub-condition. After that, it also changes the memory size m of a classifier to a random value with probability μ. If the memory size is reduced, the extra sub-conditions C_{-n} $(n > m)$ are removed. If the memory size grows, new sub-conditions C_{-n} $(n > m)$ are added which are copies of the corresponding input at the time stamp t_{-n} in the input list.

The subset-selection method is important in finding suitable memory sizes. We have compared two selection methods: random selection and fitness-based selection. In fitness-based selection, the selection probability of subset $[\mathrm{A}(t_{-n})]$ is the average of the fitness of the classifiers in it. We hypothesise that subsets with necessary memory will have higher average fitness than subsets with more memory than they need, which should make fitness-based selection successful. We hypothesise this because if a subset has a classifier that is accurate but has more memory than it needs, this classifier should be subsumed by an accurate classifier with less memory (see Section 3.2.3). Hence, the only classifiers with more memory than they need will tend to be inaccurate and have low fitness. We compare fitness-based selection and random selection in section 4. If fitness-based selection is superior that suggests our hypothesis is correct.

3.2.3 Subsumption

Subsumption is an generalization operator that helps to decrease the population size by subsuming a classifier to a more general classifier. Subsumption comes in two forms: *Action set subsumption* and *GA subsumption* [21] [2]. In XCS-SL, subsumption applies to classifiers which have different condition lengths from each other. To compare the generality of these classifiers, we assume the shorter classifier has extra virtual maximally general sub-conditions (that have only #) to fit the condition length of the longer classifier. For instance, as shown in Figure 3, to compare the generalities of the classifiers cl_a and cl_b, we consider that cl_a has one maximally general sub-condition "####" added. Accordingly, the sub-conditions C_0 and C_{-1} of cl_a are more general than the corresponding sub-condition of cl_b, hence, cl_a is more general than cl_b. In the other case, cl_c is not more general than cl_d because the sub-condition C_0 of cl_c is not more general than the C_0 of cl_d.

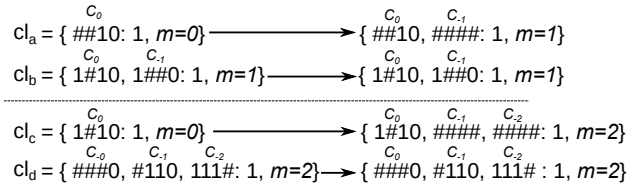


Figure 3: Example of subsumption

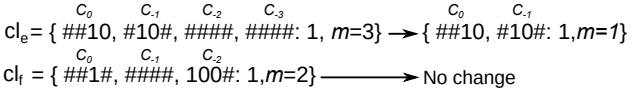


Figure 4: Example of shrinker

3.2.4 Shrinker

The shrinker is a compaction operator that helps to find compact conditions; it decreases the memory size of classifiers whose sub-conditions are maximally general. Specifically, if the sub-condition C_{-m} for the oldest time stamp is coded by only #, then its sub-condition is removed and the memory size m is decreased by 1. This process is repeated recursively. Note that the shrinker is not applied to classifiers whose memory size is 0 (i.e., the condition is composed only one sub-condition C_0). For instance, as shown in Figure 4, the sub-condition C_{-3} of classifier cl_e is removed, since C_{-3} is the maximally general condition "####", and the memory size of cl_e is reduced to 2. Iterating, the sub-condition C_{-2} is then removed. In the other case, the sub-condition C_{-2} of classifier cl_f is not the maximally general condition, hence C_{-2} is not removed. The shrinker is applied to classifiers generating by covering and the Genetic Algorithm.

4. TEST ON BENCHMARK PROBLEM

We test XCS-SL on the layered multiplexer problem as a benchmark sequence labeling task. From the test, we derive answers for two key questions on XCS-SL; (i) how XCS-SL should evolve classifiers to efficiently find a suitable memory size for each input? That is, which is the better evolutional algorithm: random selection or fitness-based selection? (ii) what are the advantages of the variable-length condition over the fixed-length condition?

4.1 Layered Multiplexer Problem

4.1.1 Problem Definition

This paper introduces a new kind of multiplexer problem [20], which we call n-Layered l-bit Multiplexer Problem (n-l LMP). The length l of input is determined from the length of the *address bits* k as $l = k + 2^k$. In the LMP, the class of an input at time t may depend on inputs at previous time stamps. The LMP defines a Boolean function on a sequence of l-bit strings. We make a dataset of D random binary strings. We send them as inputs to the LCS in sequence from t_0 to t_D and then repeat. The class of the current input is decided by referring to a previous input at a *reference time* rt; the reference time is calculated from the current input. XCS-SL stores M inputs to the input list before the start of the experiment, so there is a history for the first classifiers if they need it. Also, we consider the dataset to wrap

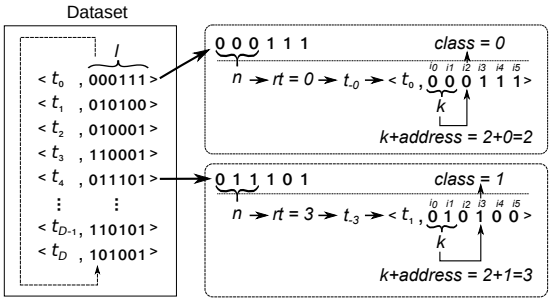


Figure 5: Example of 3-Layered 6-Multiplexer problem (3-6 LMP)

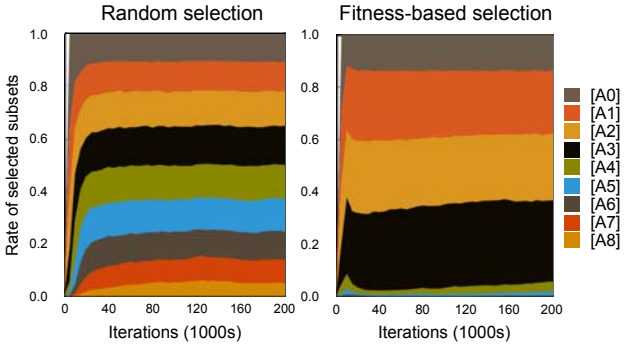


Figure 8: Rates of selected subsets for GA

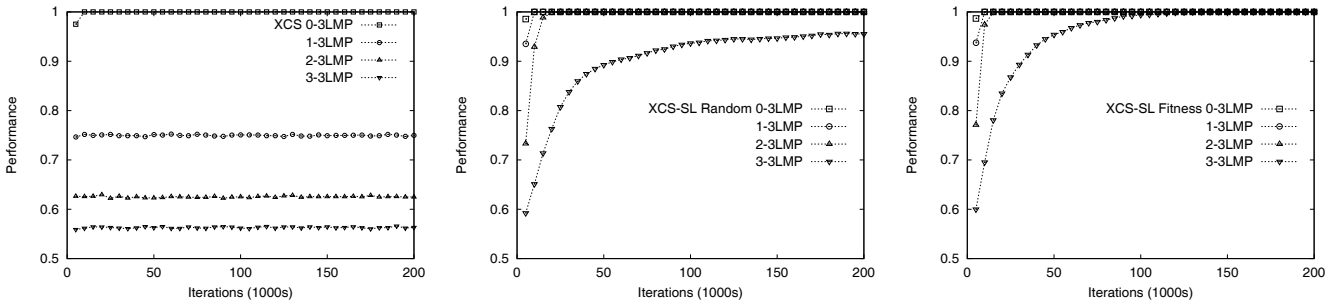
around, so if the reference time would be negative then we start counting backwards from the end of the dataset to find the reference input i.e., if $t_{-rt} < t_0$ then we wrap around to the end of the dataset and use t_{D-rt} as the reference input. Specifically, the first n bits of the current input is converted to a decimal number as the reference time. Next, the class of the current input is set to an *answer* bit of the previous input at t_{-rt} which is computed as in the normal l-bit multiplexer problem [20]. In detail, the first k bits of input t_{-rt} represent an address bit pointing to the remaining 2^k bits. The answer is the bit at $k + address$ where $address$ is a decimal number which is converted from the address bits. Figure 5 shows an example of the 3-Layered 6-bit ($k = 2$) Multiplexer Problem (which we denote 3-6LMP). In the input at time t_0, the reference time rt is equal to 0 (which is computed from the first n=3 bits "000"). The class of the current input is determined by the previous input at t_{-0} i.e., in this case the current input determines its own class. The class of the current input is found by applying the normal multiplexer function to the input at the reference time: since k=2 we compute the $address$ 0 from the first two bits ("00"). The data bits are 0111, of which the 0th bit has value 0. Hence the class of the current input is 0. As another example, the class of input at time t_4 is 1, because the rt is 3 (which is computed from "011") and the answer of the previous input at t_{-3}=t_1 is the bit at 3 (=2+1). The LMP sends a reward of 1000 when the LCS performs a correct action, otherwise 0. We use $D = 50000$ in this paper, and employ the set of {0, 1, 2, 3}-Layered 3-bit ($k = 1$) Multiplexer Problems ({0, 1, 2, 3}-3 LMP). Note 0-3 LMP is the normal 3-bit multiplexer.

4.2 Random vs. Fitness-based Set Selection

We test XCS-SL and XCS on the LMP from section 4.1, and compare the two strategies for subset selection in XCS-SL: random selection and average fitness-based selection. Each experiment consists of a number of problems that the system must solve. In each problem as one iteration, the LCS alternatively solves a *learning* problem and an *evaluation* problem (see [20]). During *learning* problems, the system selects an action randomly from those represented in the match set. During *evaluation* problems, to evaluate the system performance, the system selects the action with highest expected return but does not apply the reinforcement and discovery components. We use the standard parameter settings [2]: $N = 3000$, $\epsilon_0 = 1$, $\mu = 0.04$, $P_\# = 0.33$, $\chi = 0.8$, $\beta = 0.2$, $\alpha = 0.1$, $\delta = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $\theta_{del} = 20$, $\theta_{sub} = 20$ and Action set subsumption and GA subsumption are turned on. For XCS-SL, we set the maximum memory size $M = 8$. The maximum iteration is 200,000. The *performance*, which is the rate of correct actions the system executed, and the *population size*, which is the number of (macro) classifiers [20] in the population, are reported as the moving average over the last 5000 evaluation problems. All the plots are averages over 20 experiments.

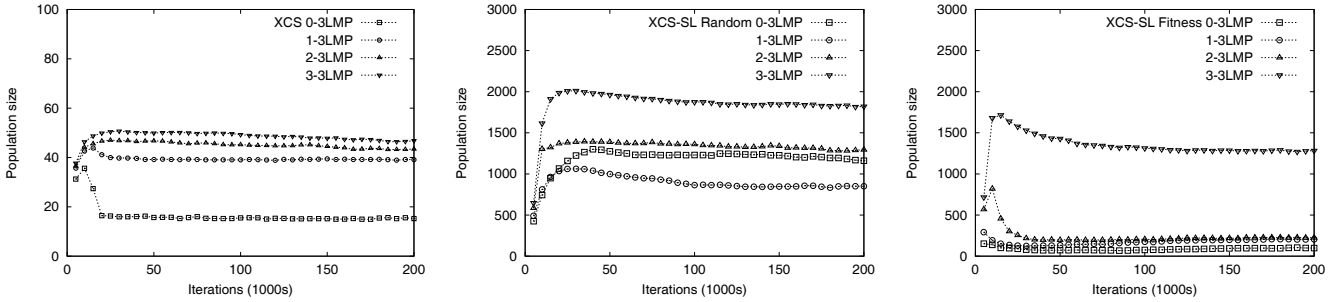
Figures 6 and 7 compare the performance and population size of XCS and XCS-SL with random and fitness-based selection on {0, 1, 2, 3}-3 LMP. Note the scale of the vertical axis for XCS is different from that of XCS-SL. XCS reaches optimal performance on 0-3 LMP and successfully decreases the population size to about 15 classifiers. As can be expected, XCS fails to reach optimal performance on other LMP problems, since it has no memory. XCS-SL with random selection reaches optimal performance on all LMP problems except for the 3-3 LMP. However, the population size is larger than with fitness-based selection (and XCS's population), even on 0-3 LMP where it is about 1160 classifiers. In contrast, XCS-SL with average-fitness selection reaches optimal performance on all LMP problems successfully decreases the population size. In particular, the population size 0-3 LMP is about 98 classifiers, although that is larger than XCS's. From these results, by employing fitness selection in XCS-SL, it can derive the better performance than the random selection. Also, XCS-SL successfully learns classifiers for inputs that are depend on the previous inputs.

We now analyze why fitness-based selection works to reduce the population size compared to random selection. Figure 8 shows the rates of selected subsets by random selection and fitness-based selection on 2-3 LMP. Note the maximum memory size M in XCS-SL is 8 while in 2-3 LMP the maximum necessary memory size is 3 (since maximum reference time is $3 = 2^2 - 1$). Hence, classifiers in $[\mathrm{A}(t_{-4})] \cdots [\mathrm{A}(t_{-8})]$ are redundant and the rates of selected subsets $[\mathrm{A}(t_{-4})] \cdots [\mathrm{A}(t_{-8})]$ should preferably converge to 0. From Figure 8, the random selection evenly selects all subsets. In contrast, the fitness-based selection intensively selects the subsets of $[\mathrm{A}(t_0)] \cdots [\mathrm{A}(t_{-3})]$, while the rates of $[\mathrm{A}(t_{-4})] \cdots [\mathrm{A}(t_{-8})]$ are clearly reduced. From these results, random selection generates many redundant classifiers whose memory size is larger than the necessary memory size. Fitness-based selection successfully focuses on the classifiers which have a suitable memory size. In the remainder of this paper we employ fitness-based selection on XCS-SL.



a) XCS b) XCS-SL Random selection c) XCS-SL Fitness-based selection

Figure 6: Performance of XCS and XCS-SL with the random selection and the fitness-based selection



a) XCS b) XCS-SL Random selection c) XCS-SL Fitness-based selection

Figure 7: Population size of XCS and XCS-SL with the random selection and the fitness-based selection

4.3 Fixed length vs. Variable length

Here we test the standard XCS-SL (using the variable-length condition) and Fix-XCS-SL (using a fixed-length condition). Specifically, we modify XCS-SL to Fix-XCS-SL where the memory size m of all classifiers is set to the maximum memory size M, i.e., the condition length is fixed. The experimental settings are the same as the previous test. We apply both systems to 3-3 LMP, and we set M = 8, 9, 10, 11, 20, 30. Both systems store M inputs in the input list before the start of experiment.

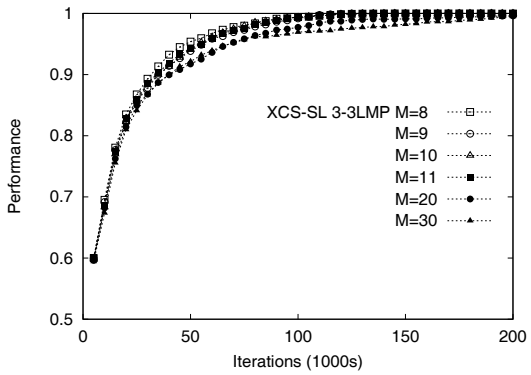
As shown in Figure 9, XCS-SL stably reaches optimal performance with different maximum memory size M. This suggests that, even with a much larger maximum memory than necessary, XCS-SL can successfully find out the suitable memory size, and it can adaptively tune condition length to fit the memory size the problem requires. Fix-XCS-SL reaches optimal performance faster than XCS-SL on M=8, 9. This is because Fix-XCS-SL starts with the right amount of memory but XCS-SL has to find the right amount. However, Fix-XCS-SL fails to reach optimality on M=10, 11, and it does not show any sign of adapting on M=20, 30. With the larger memory sizes, the search space is too large for Fix-XCS-SL to explore effectively. These results suggest that the fixed-length condition is useful in problems where we can estimate how many and where previous inputs are needed to classify data to a class. However, in most real applications we do not know how many and where, hence the maximum memory size must be set to a "large enough" value. Accordingly, the variable-length condition is more useful in most problems since it works well even when the maximum memory length is much bigger than is needed. In other words, the overhead for overestimating the maximum memory size is small when we use variable-length conditions.

5. APPLICATION TO ADL RECOGNITION

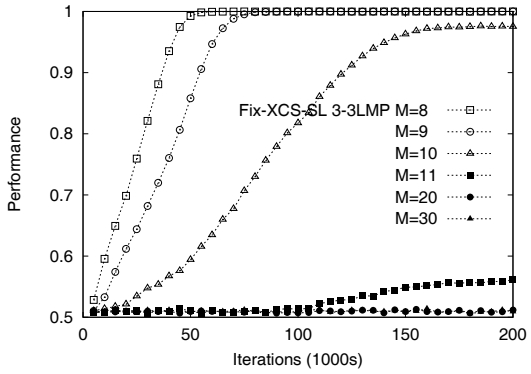
Activity of Daily Living (ADL) recognition [12] is a classification task to recognize the human activity from binary sensors. ADL data is available in UCI repository [14] as a classification problem. We modify the data (OrdonezA) to be a sequence labeling task. As shown in Figure 10, the format of each data point is a timestamp/input/class; an input in the form of binary sensor data consists of 3 elements (*sensor*, *sensor type* and *room*); a class indicates a human activity. Specifically, the *sensor* input can be one of 12 sensors: *Shower*, *Basin*, *Cooktop*, *Maindoor*, *Fridge*, *Cabinet*, *Cupboard*, *Toilet*, *Seat*, *Bed*, *Microwave* and *Toaster*; the sensor type can be one of 5 types: *PIR*, *Magnetic*, *Flush*, *Pressure* and *Electric*; the room can be one of 5 rooms: *Bedroom*, *Bathroom*, *Living*, *Entrance* and *Kitchen*; the class can be one of 10 human activities: *Sleeping*, *Showering*, *Breakfast*, *Lunch*, *Dinner*, *Snack*, *Grooming*, *Leaving*, *Toileting* and *Spare_Time/TV*. The dataset has 397 data points and contains some aliasing data that has the same input but a different class. For example, some aliasing data is:

$\langle\ t_a,$ *Toaster*, *Electronic*, *Kitchen*:*Breakfast* $\rangle$
$\langle\ t_b,$ *Toaster*, *Electronic*, *Kitchen*:*Lunch* $\rangle$
$\langle\ t_c,$ *Basin*, *PIR*, *Bathroom*:*Toileting* $\rangle$
$\langle\ t_d,$ *Basin*, *PIR*, *Bathroom*:*Grooming* $\rangle$

We set the first 70% of the data to training data and the last 30% of the data to test data. Each experiment consists of a *training* phase and *test* phase. The *training* phase is composed of *learning* problems and *evaluation* problems the same as previous tests in Section 4, and the system uses



a) XCS-SL



b) Fix-XCS-SL

Figure 9: Results of XCS-SL and Fix-XCS-SL on 3-3 LMP

	input			class
timestamp	sensor	sensor type	room	activity
< 2011-11-28 02:27:59 - 10:18:11	Bed	Pressure	Bedroom	Sleeping >
< 2011-11-28 10:21:24 - 10:21:31	Cabinet	Magnetic	Bathroom	Toileting >
< 2011-11-28 10:21:44 - 10:23:31	Basin	PIR	Bathroom	Toileting >
< 2011-11-28 10:23:02 - 10:23:36	Toilet	Flush	Bathroom	Toileting >
< 2011-11-28 10:25:44 - 10:32:06	Shower	PIR	Bathroom	Showering >
< 2011-11-28 10:34:23 - 10:34:41	Fridge	Magnetic	Kitchen	Breakfast >
⋮	⋮	⋮	⋮	⋮
< 2011-12-11 15:29:03 - 15:30:14	Toilet	Flush	Bathroom	Toileting >
< 2011-12-11 15:41:34 - 15:43:30	Basin,	PIR	Bathroom	Grooming >

Figure 10: Dataset of ADL

the training data. The test phase happens after the training phase. During the *test* phase, the system must solve the test data and, as in the evaluation phase, does not apply the reinforcement and discovery components. We compare XCS, XCS-SL and Fix-XCS-SL on the ADL recognition task, and we employ the same parameter settings of the previous tests except for $N = 5000$ and Action Set subsumption was turned off to avoid overly strong generalization pressure which has been noted in other papers e.g., [1]. Also we use a different maximum memory size M=8 and 12. The maximum iteration is 200,000. The *performance on training data* which is the classification accuracy during the training phase is reported as the moving average over the last 5000 evaluation problems. We also calculate the *classification accuracy* during the test phase. All the plots are averages over 20

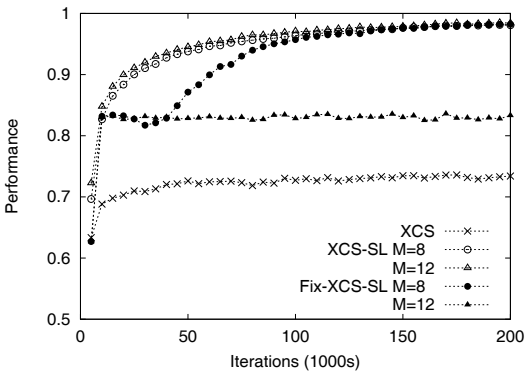


Figure 11: Performance on training data in ADL recognition

experiments. XCS-SL and Fix-XCS-SL store M inputs to the input list before the start of experiment.

Figure 11 shows the performance on training data of XCS, XCS-SL, and Fix-XCS-SL. As can be expected, XCS cannot reach the optimal performance because of the aliasing data. XCS-SL reaches about 97% performance faster than Fix-XCS-SL, and with different memory sizes M. While Fix-XCS-SL slowly reaches to 97% with M=8, it cannot reach close to optimal performance and converges to 83% with M=12. Tables 1 and 2 shows the classification accuracies on test data and p values of XCS, XCS-SL and Fix-XCS-SL. The classification accuracies of XCS-SL (M=8, 12) are better than XCS, the positive significant differences ($p < 0.001$) between XCS and XCS-SL (M=8, 12) are noted, and no significant difference between M=8 and 12 on XCS-SL was found. This result shows that XCS-SL stably derives high classification accuracy even with large memory size. Similarly, the classification accuracy of Fix-XCS-SL (M=8) is also better than XCS, the positive significant difference ($p < 0.001$) between XCS and Fix-XCS-SL (M=8) is noted. However, the classification accuracy of Fix-XCS-SL (M=12) is much lower than XCS and XCS-SL. This is because Fix-XCS-SL fails to learn correct classifiers due to the large memory size.

Figure 12 shows examples of interesting classifiers that XCS-SL (M=8) learned on the ADL task. The classifier cl_a represents "the subject cooks breakfast after the shower". Notice that the condition for the current time is fully general: no matter what the current input is, the classifier predicts breakfast. The current condition effectively aliases many possible states (sensor inputs), but the memory of the shower disambiguates the current state and predicts the subject is having breakfast. This is an example of the kind of human-readable pattern mentioned in the introduction.

Classifier cl_b, "subject cooks lunch after Spare_time/TV" also uses memory to disambiguate the current state. The current condition matches when the cooktop is used, but breakfast, lunch and dinner all use the cooktop, so they are aliased. This classifier uses memory of the *seat* sensor, which indicates a unique human activity *Spare_time/TV*, to disambiguate the current state.

Classifiers cl_c and cl_d use the *Entrance* room category, which represents the human activity *Leaving* (the house). There is no sensor to represent returning to the house, but any sensor after *Leaving* indicates the subject has returned.

Table 1: Classification accuracies of XCS, XCS-SL and Fix-XCS-SL

XCS		0.750
XCS-SL	M=8	0.853
	M=12	0.853
Fix-XCS-SL	M=8	0.850
	M=12	0.187

Table 2: p values of XCS, XCS-SL and Fix-XCS-SL (Bold text indicates p<0.001)

		XCS-SL		Fix-XCS-SL	
		M=8	M=12	M=8	M=12
XCS		**8.05E-05**	**8.93E-05**	**4.14E-04**	**2.33E-16**
XCS-SL	M=8	-	9.50E-01	7.53E-01	**1.87E-26**
	M=12	-	-	7.87E-01	**4.13E-24**
Fix-XCS-SL	M=8	-	-	-	**6.61E-22**

cl_a = { (#, #, #), (#, #, #), (Shower, #, #): Breakfast}
cl_b = { (Cooktop, #, #), (#, #, #), (Seat, #, #):Lunch}
cl_c = { (Fridge, #, #), (#, #, Living), (#, #, Entrance):Snack}
cl_d = { (Basin, PIR, Bathroom), (Maindoor, #, Entrance):Grooming}

Figure 12: Examples classifiers on ADL recognition

Classifiers cl_c represents "the subject eats a snack after he/she came back home". Classifier cl_d represents "the subject grooms after he/she came back home".

6. CONCLUSION

This paper introduced XCS for sequence labeling (XCS-SL). We introduced the Layered Multiplexer problem as a benchmark for sequence labeling. From the test on the LMP we showed XCS-SL successfully learns correct classifiers which have suitable memory size for given problems. Specifically, we showed that the fitness-based selection as an evolutionary algorithm can enhance XCS-SL to efficiently find out the suitable memory size. We also showed that the fixed-length condition is useful for problems where we know the suitable memory size; the variable-length condition, in contrast, is useful in most problems since it can handle a larger memory size than the fixed-length condition. In future work we will consider messy-coding, learning patterns on both actions and inputs, comparisons with other memory-using LCS.

7. REFERENCES

[1] E. Bernadó-Mansilla & J. M. Garrell-Guiu. Accuracy-based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11:209–238, 2003.

[2] M. V. Butz & S. W. Wilson. An algorithmic description of XCS. *Journal of Soft Computing*, 6(3–4):144–153, 2002.

[3] B. Carse & T.C. Fogarty. A Delayed-Action Classifier System for Learning in Temporal Environments. In *Evolutionary Computation, 1994. IEEE World Congress on Comp. Int.*, p. 670–673, 1994.

[4] W. A. Chaovalitwongse, O. A. Prokopyev, & P. M. Pardalos. Electroencephalogram (EEG) time series classification: Applications in epilepsy. *Annals of Operations Research*, 148(1):227–250, 2006.

[5] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison Wesley*, 1989.

[6] J. Han. *Data Mining:Concepts and Techniques*. Morgan Kaufmann, 2005.

[7] J. H. Holland. Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-based system. *Machine Learning*, 2:593–623, 1986.

[8] B. Kaluža, V. Mirchevska, E. Dovgan, M. Luštrek, & M. Gams. An agent-based approach to care in independent living. In *Ambient Intelligence*, *LNCS* 6439, p. 177–186. Springer, 2010.

[9] M. Kudo, J. Toyama, & M. Shimbo. Multidimensional curve classification using passing-through regions. *Pattern Recogn. Lett.*, 20(11-13):1103–1111, 1999.

[10] J. D. Lafferty, A. McCallum, & F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML2001*, pages 282–289, 2001.

[11] P. L. Lanzi & S. W. Wilson. Toward Optimal Classifier System Performance in Non-Markov Environments. *Evolutionary Computation*, 8(4):393–418, 2000.

[12] F. J. Ordóñez, P. de Toledo, & A. Sanchis. Activity Recognition Using Hybrid Generative/Discriminative Models on Home Environments Using Binary Sensors. *Sensors*, 13(5):5460–5477, 2013.

[13] R. Preen. An XCS Approach to Forecasting Financial Time Series. In *GECCO2009*, p. 2625–2632. ACM, 2009.

[14] UCIrvine Machine Learning Repository. http://archive.ics.uci.edu/ml/.

[15] H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Inter. Conference on New Methods in Language Processing*, pages 44–49, 1994.

[16] Andy Tomlinson & Larry Bull. An accuracy based corporate classifier system. *Soft Computing*, 6(3-4):200–215, 2002.

[17] Fani A. Tzima & Pericles A. Mitkas. ZCS Revisited: Zeroth-level Classifier Systems for Data Mining. *ICDM Workshops*, pages 700–709, 2008.

[18] Fani A. Tzima & Pericles A. Mitkas. Strength-based learning classifier systems revisited: Effective rule evolution in supervised classification tasks. *Eng. Appl. Artif. Intell.*, 26(2):818–832, February 2013.

[19] A. Webb, E. Hart, P. Ross, & A. Lawson. Controlling a Simulated Khepera with an XCS Classifier System with Memory. In *ECAL2003*, pages 885–892, 2003.

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

[21] S. W. Wilson. Generalization in the XCS classifier system. In *In Proceedings of 3rd the Third Annual Genetic Programming Conf.*, pages 665–674, 1998.

[22] K. Zhang & W. Fan. Forecasting skewed biased stochastic ozone days: Analyses, solutions and beyond. *Knowl. Inf. Syst.*, 14(3):299–326, 2008.