

# Learning Classifier Systems for User Context Learning

Anil Shankar

Evolutionary Computing Systems Laboratory (ECSL)  
Dept. of Computer Science and Engineering  
University of Nevada, Reno  
anilk@cse.unr.edu

Sushil Louis

Evolutionary Computing Systems Laboratory (ECSL)  
Dept. of Computer Science and Engineering  
University of Nevada, Reno  
sushil@cse.unr.edu

## Abstract-

**Current computer applications and user interfaces lack user context and are not successful in learning user preferences to improve user interaction. We present *Sycophant*, a context learning calendaring application program which is designed to learn a mapping from user-related contextual features to reminder actions. In this paper, we consider the feasibility of using a genetics-based machine learning technique, XCS, for the purpose of learning this mapping from a set of context features to reminder actions as a predictive data-mining task. We compare XCS's performance with a decision tree algorithm on this learning task and show that XCS outperforms the decision tree learner.**

## 1 Introduction

Current operating systems support devices like an internal clock, keyboard and mouse for providing input or context. Computer applications use this paltry context through simple Application Programming Interfaces (APIs) to enhance personal productivity by building user models. Application programs built this way can only make weak attempts to adapt to individual user needs. There is no personalization through learning, no long term memory, and advances in vision, speech, text analysis, and the availability of cheap computing power have not been fully utilized for improving user interaction. The result is that many current computer applications lack *context awareness*.

We consider *context* as any information that is pertinent to the interaction between an application and a user, which can be used to characterize the situation of entities (a person, a place or an object) [7].

We propose to use simple sensors to continuously gather data on a computer system's internal and external environment, store this data in a data warehouse, and *mine* this data for useful user-behavior patterns in order to better predict user preferences (behavior) and improve user interaction. Applications can then use this learned model of user preferences to better interact with the user. More generally, we want to apply machine learning techniques to data gathered from simple context sensors to build improved human computer interfaces.

In our current work, we use a Genetics Based Machine Learning (GBML) technique to learn the mapping from user-related contextual features to application actions for a context-sensitive user interactive application. Specifically, we investigate the effectiveness of **Sycophant**, a

simple calendaring application in learning a mapping from context-features to reminder type as predictive data-mining task [15]. We also compare the performance of our GBML technique with a decision-tree learning algorithm. Our results for predicting which of the four different types of reminders to generate using Sycophant with external (motion, speech) and internal (keyboard, mouse activity) sensors show that both the training-set and test-set performance of XCS is better than that of a decision tree learner.

The next section presents related work in the area of context-aware applications and genetics-based machine learning. We introduce Sycophant's architecture in section 3. Section 4 gives a brief introduction to XCS. Details about our experimental design and data pre-processing are given in section 5. Section 6 presents results comparing our GBML technique with a decision-tree learner on the reminder generation task. We discuss the lessons learned from our experiments in section 7. Section 8 explores directions for future work.

## 2 Background and Related Work

To improve user interaction for application programs, we can gather external and internal contextual information from the user's environment. We can sense the internal context of a computer with respect to a user by monitoring different active processes on a user's machine. Here are few ways of how we can gather simple external context from a user's environment: sense the presence or absence of motion by using a motion-sensor (web-camera), detect the presence or absence of speech by using a microphone, monitor keyboard-usage etcetera. Even without knowing who is there or what is being said, such simple sensors can be used to improve user interaction. For example, if you were Jane's user-interface you could learn answers to the the following questions.

- Should I pause the current song when Jane leaves the room?
- If there is no one in the room should I pop up a scheduled appointment?
- Should I ask Jane if she wants me to cancel a scheduled meeting (that was supposed to start five minutes ago) ?
- If there is someone else in the room at the time should I remind Jane?

In our research, we view a computer as a stationary robot

with simple sensors for sensing the external and internal environments of a computer system and a simple actuator which is the response of an application program on the same computer [14]. On this stationary-robot model of a desktop PC, we built Sycophant, a simple calendaring application program that stores appointments and reminds the user using different types of reminders. Our system continuously gathers binary activity data from the keyboard, mouse, a motion detector, and a speech sensor. We also monitor the activity of five processes on the computer. Whenever Sycophant generates a reminder, it expects the user to indicate whether Sycophant used the correct reminder type. A reminder can be visual (a pop-up window), speech (using a text-to-speech system), both, or neither.

In the area of context-aware applications and environments, *Reba* demonstrated the necessity for systems to be context-aware for anticipating user actions and simplify user interaction, Bailey and Adamczyk's work showed that a user's attention must be carefully managed among competing applications and that this management is necessary to mitigate the disruptive effects of necessarily interrupting a user, Horvitz and Apacible have built models for predicting the cost of interrupting users and used machine learning techniques to generate statistical models to infer the state of interruptibility of users and, Hudson, Fogarty, Atkeson et al. have constructed robust sensor-based predictions of interruptibility [2, 14, 11, 8].

In the area of data-mining, XCS has been used for learning Boolean functions by Wilson and Kovacs, on The Monk's problems by Saxon and Barry and on the well-known Wisconsin Breast Cancer Data Set by Wilson [20, 21, 13, 22, 17].

Our work integrates the above described approaches in context-aware systems, genetics-based machine learning and data mining. Sycophant learns whether or not to interrupt the user as well as how to interrupt the user. Like Fogarty, we use real sensors but in addition to learning whether to interrupt the user, Sycophant learns which one of the four different types of reminders to use in interrupting the user [8]. We use a GBML technique, XCS, which consists of a set of rules and actions to learn the mapping from context-features (gathered by our sensors) to reminder types as predictive data-mining task. We also compare the performance of this GBML technique with a decision-tree algorithm for learning this mapping from sensors to reminder types.

### 3 SYCOPHANT

Sycophant can generate four different types of reminders: a simple pop-up window containing the appointment text, a voice reminder where the appointment text is spoken using the Festival Speech Synthesis System [3], a pop-up window and a voice reminder, and no pop-up or voice reminder. The appointments for Sycophant were set up to mimic a user's regular work-day. Figure 1 shows a screen-shot of Sycophant's appointment entry interface.

Figure 2 depicts Sycophant's architecture. The calendaring application runs as separate process and five sensors

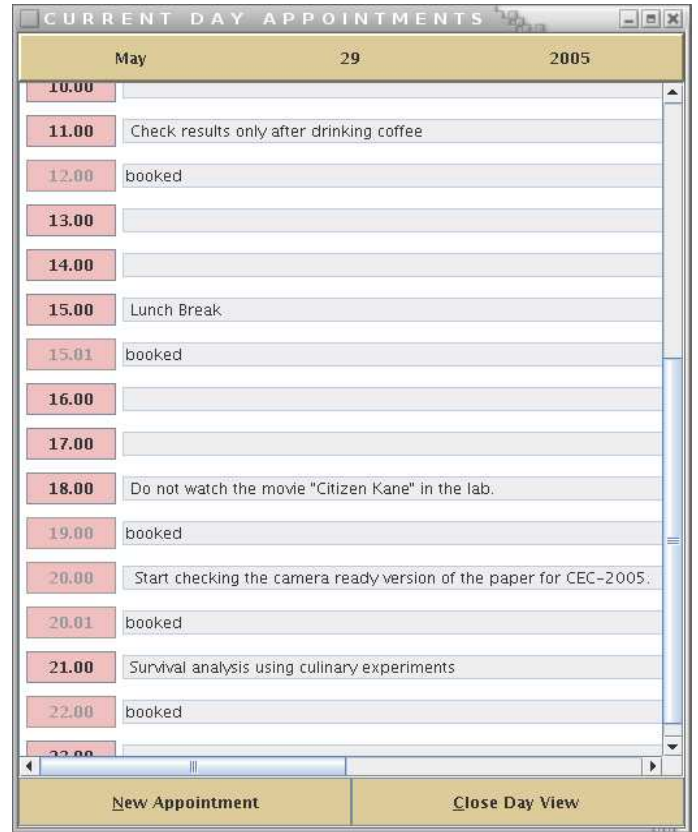


Figure 1: Screen Shot of Sycophant's Appointment Entry Interface

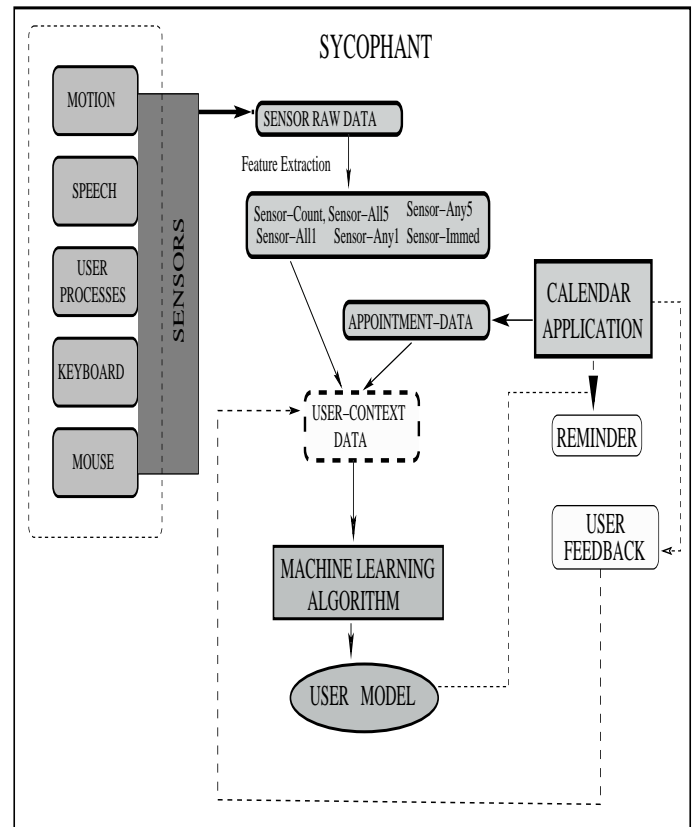


Figure 2: Sycophant Architecture

collect data on the computer and immediate vicinity. These sensors are binary, for example, when the motion sensor detects motion it reports a value of 1, otherwise 0. Our sensors are:

- **Motion:** We use a cheap Logitech USB web-cam with the motion package [1].
- **Talk:** We use the Sphinx Speech Recognition System to simply detect the presence or absence of speech. This is fairly accurate and fast as long as we do not actually attempt to recognize the detected speech [6].
- **Processes:** The user under observation actively uses the following five processes: `java`, `bash`, `gnome-terminal`, `xscreensaver`, `mozilla` and we kept track of these processes' state.
- **Keyboard:** The keyboard sensor monitors keyboard activity.
- **Mouse:** Like the keyboard sensor, the mouse sensor monitors mouse activity.

For this study, we collected 387 user-context data exemplars from a single user over a period of eight to ten weeks. Every fifteen seconds, we checked all sensors for activity and stored these values to a file. Next, we extracted the following six features from the raw data [12]: *Any5*, if the sensor is active during any of the fifteen second intervals during the last five minutes. *All5*, if the sensor is active during all of the fifteen second intervals during the last five minutes. *Any1*, if the sensor is active during any of the fifteen second intervals during the last minute. *All1*, if the sensor is active during all of the fifteen second intervals during the last minute. *Immed*, if the sensor is active during the last fifteen second interval. *Count*, the number of intervals during which the sensor is active during the last five minutes. Therefore every sensor provides six features. We consider each of the five user processes as a separate sensor so the number of sensors grows to nine and We therefore ended up with a total of 54 features. Finally, We also include a user identifier and the next appointment time.

Here is an exemplar from our context data-set:

```
User1, 05.00, 0, 0, 0, 0, 0, 0, 0, 7,
0, 1, 0, 0, 0, 20, 1, 1, 1, 1, 1, 20,
1, 1, 1, 1, 1, 20, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 20, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

where the first two features correspond to User-Id and Time of Appointment. The remaining set of features in groups of six represent Sensor-Count, Sensor-All5, Sensor-Any5, Sensor-All1, Sensor-Any1 and Sensor-Immed. The sensors are ordered as follows: Motion, Talk, Process, Keyboard and Mouse. Each of these six features is derived for each sensor mode. The last value from the above data row corresponds to the type of reminder actually preferred by a user, which is obtained through user feedback. During the initial phases of training when insufficient user-context data is available, Sycophant uses a hand-coded set of rules for generating a reminder. After collecting an adequate number

of exemplars, Sycophant uses a machine learning algorithm to build a user-model and uses the reminder generated from this model.

## 4 Brief Introduction to XCS

XCS is derived from Wilson's Animat and ZCS [18, 19]. Figure 3 shows the architecture of XCS. The system consists of a population of classifiers where each classifier consists of the following components:

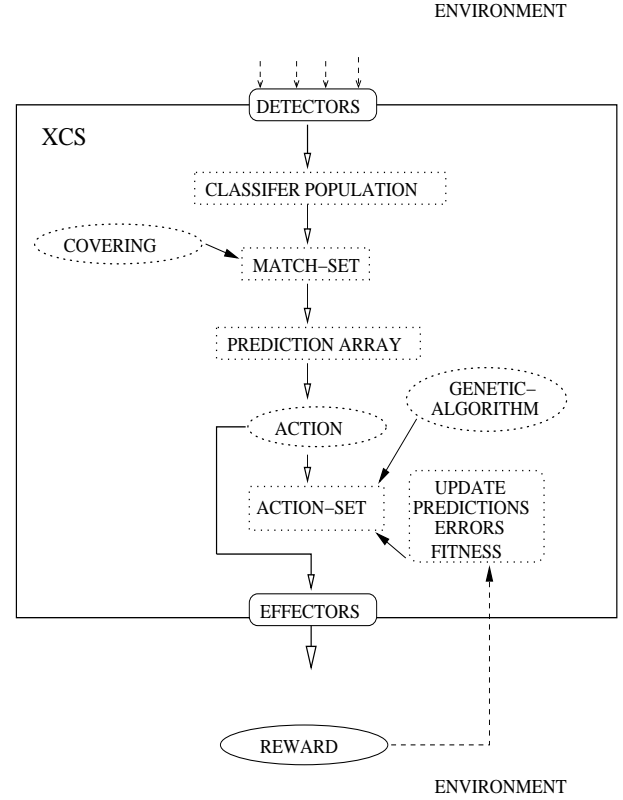


Figure 3: XCS Architecture

- **Condition:** defined over the ternary alphabet  $\{0, 1, \#\}^L$ , where  $L$  is the length of the bit-string. It specifies an input state sensed from the environment which a classifier tries to match.  $\#$  is a meta-character which can match either a 0 or a 1. In our case, a condition is an exemplar from the context-data set as described in section ??.
- **Action:** specifies the action which a classifier can take. The possible values of an action in our study is the classification of a reminder-type, that is, an action can take the values from the  $\{0, 1, 2, 3\}$ .
- **Prediction Estimate:** the expected pay-off if a classifier matches the environment's sensory input and its action is chosen by the system.
- **Prediction Error:** estimate of the error made in predictions. This is calculated after the reward from the environment is received.

- Fitness: denotes the classifier's fitness.
- Experience: the number of times which the classifier belonged to a set of actions since its creation.
- Time-stamp: a counter or time-step of the last occurrence of the genetic algorithm (GA) in an action set to which this classifier belonged.
- Action set size: average size of the action sets to which this classifier belonged
- Numerosity: the number of micro-classifiers which this *macro-classifier* represents. A macro-classifier represents  $n$  traditional classifiers having identical conditions and actions.

The external environment provides a message string, an exemplar from the context data-set which is a string belonging to  $\{0, 1\}^L$ , where  $L$  is the number of bits in each situation, to XCS. Next, XCS forms a match-set ( $M$ ) which is a set of classifiers from the population of classifiers whose conditions match the environmental message. If none of the classifiers in the population match the environmental message, *covering* is done by creating matching classifiers with each of the possible actions and placing them in  $M$ . In the implementation of XCS we are using, the initial population is empty and covering occurs only at the beginning of a run. Next, for each of the actions presented in  $M$ , the system computes a fitness-weighted average of the predictions of each classifier in  $M$  having that particular action. At the next step, XCS chooses an action from those represented in  $M$  and sends it to the environment. Each action results in a reward whose value is 1000 if the action is correct and 0 otherwise. Our initial attempts to choose the action in a non-deterministic way seemed to degrade the system performance, therefore we chose to pick the best action and this approach of choosing the action improved the system performance. Classifiers in  $M$  which proposed this particular action are added to a set which is called the action-set ( $A$ ).

XCS next updates the classifiers by re-evaluating the classifiers in the action-set based on the actual reward ( $R$ ) returned by the environment when the system chose to execute its best chosen action. First, the predictions are updated and next, the errors. For each classifier, accuracy and its relative accuracy are also computed. Finally, the fitness of each classifier is updated based on  $\beta$ , the learning rate for updating fitness, prediction, prediction error, and action set size estimate for the classifiers in XCS. Based on  $\theta_{GA}$ , the threshold for the GA application in an action set, the system can also execute a GA within the action-set. Classifiers are thus evolved by a process of trial and error in which, given an input and a match-set, a particular action is tried and an appropriate reward is provided from the environment and the parameters of the associated classifiers are also updated. Classifiers which are able to accurately predict the correct action tend to be reproduced, while inaccurate classifiers get deleted eventually. This process of sensing the input, choosing a particular action and running the GA on an action-set is done until some termination condition is met, for example when 20,000 problems are sampled randomly or when the performance of the system reaches 100 percent.

More details concerning the commonly used values for parameters, formation of match-set and action-set, updating of classifier parameters, and operation of the GA within the system is found in Wilson [20]. A detailed algorithmic description is found in Wilson and Butz [4].

XCS digresses from the traditional model of a learning classifier systems in the following ways: the fitness of a classifier is based on the accuracy of its payoff prediction instead of the actual prediction itself, the GA is run in the action set instead of the entire population and the classifier system has no message-list [10, 9].

In our research, we use XCS for an independent single-step task in which an input, in the form of context-data features (binary representation), is presented to the system and the system makes a decision (the type of reminder to be chosen). The environment provides a reward of 1000 if the system takes the correct action (chooses the correct reminder-type) and 0 otherwise.

## 5 Experiment Details

### 5.1 Data Preprocessing

Our approach extends an implementation of XCS in Java, *XCSJava 1.0*, and modifies this implementation to work with our context data-set collected using Sycophant [5]. We transformed our contextual data set described in section 3 into a binary representation after removing the User-Id attribute since our current data is collected from a single user. We converted the appointment time into seconds and consider it as an integer value and we also considered *Count* as integer value. Next, we converted these two integer valued attributes into their binary equivalents and retained the other binary valued features of our data-set. After this transformation, we end up with a string like the example shown below :

```
10001010101010101001110010110000
10010000100110111101001101111010
01111111000000000000100111111100
00000000000000000000:3
```

The value after the ":" is the correct action for the set of features. In the above case, the correct action is to choose a reminder of type 3. You should note that this string represents a classifier where the first part represents the condition (of the sensors) and the second part after the ":" represents the action (reminder-type).

### 5.2 Design

We wanted to directly compare the performance of XCS with an implementation of C4.5, a decision tree algorithm called *J48* from the Weka machine learning tool-kit [16, 23]. We therefore created ten stratified folds of the context data-set for evaluating the training and testing performance and saved each of these tens folds of training sets and corresponding testing sets. Next, we transformed these training

and testing sets into their binary equivalents are mentioned in subsection 5.1 to be used by XCS. We thereby ensured that both J48 and XCS were exposed to the same sets of training and testing data folds. For both J48 and XCS we use the training fold and its corresponding testing fold and record the training and testing performances. We repeat the same process for all of the ten folds.

We make XCS evolve a set of classifiers for the first training fold and store these classifiers after our stopping criterion which is defined as repeatedly sampling 20,000 random problems (context data-set exemplars) from the training fold or when the system performance during training reaches 1.0. Next, we use this evolved set of classifiers on the testing fold and record our performance metrics of system error, performance, and number of classifiers. The system error is the difference between the actual reward received and the system prediction for the chosen action. Performance is the percentage of correctly solved problems in the last  $M$  problems sampled, in our experiment, the value of  $M$  is 100. A problem or exemplar is correctly solved if the classifier is able to correctly predict the type of reminder to be generated. The number of classifiers is the size of the population in terms of macro-classifiers at a certain time step. More details about the meaning of these parameters are given in [4]. For each training and testing fold, we conduct ten runs to get an average of the training and testing performance. We repeat the same process for all the ten folds of our cross-validation set. For both XCS and J48, we average the results from the ten folds of their corresponding data-sets to get a measure of the final performance for comparison purposes.

### 5.3 XCS Parameter Settings

We set the number of trials, that is, the number of exemplars to be randomly sampled from a training-fold to 20000. The learning rate,  $\beta$ , for updating fitness, prediction, prediction error, and action set size estimate an XCS's classifiers is tuned to 0.2. Finally,  $\theta_{GA}$ , the threshold for the GA application in an action set is tuned to 25. We set the different parameters of the GA as follows: the probability of applying crossover in an offspring classifier is 0.67, the probability of mutating one allele and the action in an offspring classifier is 0.4, and, the probability of using a don't care symbol in an allele ( $P_{\#}$ ) is 0.33.

Earlier, we had investigated an approach of restricting the number of classifiers to an estimate of the number of rules (the number of leaves) used by a decision-tree learner. This approach restricted XCS's performance by limiting the GA within the classifier system. A GA in XCS provides a niching facility for allowing cooperative rule-sets to coexist within a population while allowing competing rule-sets to converge on optimum rule attributes within a niche. Limiting the size of the population in XCS takes away the opportunity for the GA to experiment with recombination. We therefore explored using XCS in two modes. In the first setting, we do not limit the classifier population size but use a maximum classifier population size of 20000 ( $N$ ) for XCS. Our second setting is the same as the first one but

here we use condensation to extract minimal subset of classifiers which represent the final solution [20]. Condensation consists of running the classifier system with crossover and mutation rates set to zero. This process freezes genetic search as no new classifier conditions can now be generated. However, the classifier selection and deletion processes still continue to operate whenever the GA is triggered. This results in a tendency for less fit and less general classifiers to be weeded out of the population. We enable condensation after our stopping criterion during the training fold evaluation. Condensation stops after an additional 20000 problems have been sampled from the training fold. We examine XCS's performance on the training set after randomly sampling 100 problems from a training fold and stop when the system has sampled 20000 problems or when condensation is suspended. We set all the other parameters to their default values. A complete description the default parameter settings for XCS is given by Butz and Wilson [4].

## 6 Results

Our training and testing performance is measured on a scale of 0.0 to 1.0. We use a two-sample t-test for comparing the mean values of the performance of XCS and J48. We set up our statistical inference test to have 95 percent confidence interval for a one-sided comparison, that is we check if the value of a result from one algorithm is greater (or lesser) than the corresponding result from the second algorithm.

Table 1 shows the performance of XCS when compared to J48 on our user-context data set. The first column is the data set used for evaluating the performance of a learning algorithm; the data set is a training fold or a testing fold. We show the performance J48 in column two. XCS's performance with and without condensation is show in columns three and four respectively. Our statistical inference regarding how XCS fared when compared with J48 is given in columns four and five. Column four is the inference when XCS is operated without condensation. Inference when XCS is operated with condensation is given in column five.

XCS without condensation had on an average 19386 classifiers in the population at the end of the training phase. We were able to reduce the number of classifiers to an average value of 16428 when we operated XCS in the condensation mode. We note from table 1 that while XCS equals the performance of J48 on the training set, it significantly outperforms J48 on the test set. We attribute this improved performance on the test-set to the tendency of XCS to form accurate generalizations and its ability to form a better mapping from the condition space to the action space (reminder-types) [13]. We provide our context-data set, the cross-validation data-sets used by J48 and XCS, and result files containing the classifiers evolved by XCS at [http://www.cse.unr.edu/~anilk/CEC2005/after\\_comments/](http://www.cse.unr.edu/~anilk/CEC2005/after_comments/).

## 7 Conclusions

In this paper, we have considered the feasibility of using a Genetics Based Machine Learning technique, XCS, to learn the mapping from a set of context-features to reminder type

Table 1: Comparison of the mean performance of XCS with J48

I	II	II	III	IV	V
Data Set	Performance of J48	Performance of XCS	Performance of XCS <sub>c</sub>	Inference	Inference <sub>c</sub>
Training-Set	0.812	0.940	0.937	Equal	Equal
Test-Set	0.700	0.940	1.000	Better	Better

as a predictive data-mining task in the area of context learning for improving user interaction. We have shown that XCS is capable of learning this complex, non-linear classification function and that it can be used to accurately predict unseen cases. We have also directly compared the performance of XCS with a popular decision-tree machine learning algorithm and have shown that XCS significantly performs better than a decision-tree on unseen examples in the test-set while matching the decision-tree’s performance on the training sets.

## 8 Future Work

We use condensation to obtain a minimal number of non-overlapping optimal set of rules to describe the problem space [20]. Currently we trigger condensation whenever XCS’s performance reaches 1.0 and remains close to this value for a certain number of consecutive time steps. In this way we have been able to reduce the number of classifiers and improve XCS’s performance. However, we end up with a large number of rules (classifiers) when compared with a rule-estimate number of a decision-tree learner. This led us to suspect that we might have terminated condensation prematurely. We would like to adopt Kovac’s heuristic of using the system error for deciding when to start and terminate condensation to better evaluate XCS’s performance [13]. Ideally, we would like to have minimal and optimal set of rules for learning the mapping from user-context features to reminder types. Our intention is to end up with classifiers which are capable of describing the patterns in the user-context data which are human readable. These rules can then be compared with the rules produced by a decision tree learner. Thus, we would like to venture into the area of descriptive data-mining where we want to analyze the classifiers evolved by XCS and compare the rules generated by XCS with that of a decision tree algorithm. Ideally, we seek to automate the process of translating the classifiers evolved into human-understandable rules and examine their relevance for learning user preferences for reminder type. We wish to combine expert-generated rules with machine-learned rules and use this combined knowledge to design better adaptive user interfaces.

The type of a reminder can have a significant impact on user effectiveness and thereby her preferences, for instance, not generating a reminder (type-0) would be ideal in some cases and would be a costly error in other cases. We intend to consider weighting the rewards for different reminder types. For example, not generating a reminder

could be given a higher or lower reward depending upon user-context. We would like to evaluate the XCS classifier system under these conditions. We also want to test the performance of XCS on the task of whether or not to generate a reminder.

In this paper, we directly convert our context data-set into a binary representation to be used by XCS, preliminary investigation into attribute subset selection and discretization of our context data-set has given us promising results for improving the accuracy of reminder type prediction by a decision tree. We would like to consider this transformed context-data set where attribute subset selection and discretization have been performed and use it with XCS and again evaluate XCS’s performance on this transformed data set.

Further, we plan to collect data from different users to scale up our research in the area of context learning applications and consider the possibility for personalization to individual users. Our results in this paper clearly indicate that classifier systems are a viable method for this task.

## Acknowledgments

The authors would like to thank the reviewers for helping us improve our paper.

This work was supported in part by contract number N00014-0301-0104 from the Office of Naval Research.

## Bibliography

- [1] Motion. <http://motion.sourceforge.net/>.
- [2] P. D. Adamczyk and B. P. Bailey. If not now, when?: the effects of interruption at different moments within task execution. *Proceedings of the 2004 conference on Human factors in computing systems*, pages 271–278, 2004.
- [3] A. Black, P. Taylor, and R. Caley. The festival speech synthesis system. 1998.
- [4] M. Butz and S. W. Wilson. An algorithmic description of xcs. *Soft Comput.*, 6(3-4):144–153, 2002.
- [5] M. V. Butz. XCSJava 1.0: An Implementation of the XCS classifier system in Java. Technical Report 2000027, Illinois Genetic Algorithms Laboratory, 2000.
- [6] C.M.U. Sphinx: Open source speech recognition. <http://www.speech.cs.cmu.edu/sphinx/>.
- [7] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction*, 16, 2001.
- [8] J. Fogarty, S. E. Hudson, and J. Lai. Examining the robustness of sensor-based statistical models of human interruptibility. *Proceedings of the 2004 conference on Human factors in computing systems*, pages 207–214, 2004.
- [9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [10] J. H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 593–623. Kaufmann, Los Altos, CA, 1986.
- [11] E. Horvitz and J. Apacible. Learning and reasoning about interruption. *Proceedings of the 5th international conference on Multimodal interfaces*, pages 20–27, 2003.
- [12] S. Hudson, J. Fogarty, C. Atkeson, J. Forlizzi, S. Kiesler, J. Lee, and J. Yang. Predicting human interruptibility with sensors: A wizard of oz feasibility study. *Proceedings of CHI 2003, ACM Press*, 2003.
- [13] T. Kovacs. *XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions.*, pages 59–68. Springer-Verlag, August 1997.
- [14] A. Kulkarni. A reactive behavioral system for the intelligent room. *Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2002.*, 2002.
- [15] S. J. Louis and A. Shankar. Context learning can improve user interaction. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IRI - 2004, November 8-10, 2004, las Vegas Hilton, Las Vegas, NV USA*, pages 115–120, 2004.
- [16] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.
- [17] S. Saxon and A. Barry. XCS and the monk's problems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, page 809, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.
- [18] S. W. Wilson. Classifier systems and the animat problem. *Mach. Learn.*, 2(3):199–228, 1987.
- [19] S. W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.
- [20] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
- [21] S. W. Wilson. Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.
- [22] S. W. Wilson. Mining oblique data with XCS. *Lecture Notes in Computer Science*, 1996:158–??, 2001.
- [23] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, USA, 2000.