Better Personalization Using Learning Classifier Systems

Anil Shankar and Sushil J Louis

Department of Computer Science and Engineering
The University of Nevada
Reno, Nevada 89557
United States of America
anilk@cse.unr.edu, sushil@cse.unr.edu

Abstract. This paper describes an approach for better application personalization by learning user-context. We present Sycophant, a context-sensitive calendaring application capable of generating four different types of reminders. Sycophant uses a Genetics-Based Machine Learning technique, XCS, to learn the type of reminder preferred by a user and successfully performs this user-context learning for three different users. XCS performs as well as a decision tree learner in predicting whether or not to generate a reminder and outperforms the decision tree learner in learning to predict the correct reminder.

1 Introduction

Current computer applications provide little or no personalization to specific users. We attribute this lack of personalization to the meager use of context information related to a user. Modern operating systems use sparse user-context information gathered from an internal clock, keyboard and mouse to build a weak user-model and enhance personal productivity. As a result, application programs *behave* in the same way oblivious to the presence of a user, that is, application programs currently supported on any desktop computer lack *context-awareness*. Additional, potentially useful information is available in the form of speech and motion from the computer's immediate vicinity. We use this additional user-context information and harness the advances made in speech and natural language processing, computer vision and machine learning to address the issue of user-specific personalization of computer application programs. For this purpose, we design a context-sensitive calendaring application capable of generating four types of reminders and evaluate the personalization achieved by this program with respect to three different users.

We adhere to Dey's definition of *context* in our research. He defines context as any information which we can use to identify the situation of an entity; an entity can be a person, a place or an object [6]. Along with this working definition of context, we model a desktop PC as a stationary robot with sensors and actuators

that gather context from a user's environment. In this paper, we use a Genetics-Based Machine Learning (GBML) technique, XCS, for learning a mapping from sensor values to application actions. We hypothesize that we can learn to design better, more context-sensitive, user-interfaces (applications) using our stationary robot approach of a desktop PC to *sense* and respond to a user and increase the level of user-context awareness for applications.

For testing our hypothesis, we built *Sycophant*, our context-sensitive calendaring application which learns to predict the type of reminder preferred by a user for scheduled appointments and tasks [15]. Sycophant generates the following four types of reminders for an appointment 1) a visual reminder which displays the appointment text in a pop-up window 2) a voice reminder which speaks out the appointment text using the Festival Speech Synthesis system [3] 3) both visual and voice reminders 4) None (we suppress reminder generation until a more opportune moment).

In order to learn a model of useful user-behavior patterns and generate the correct user-preferred reminder, Sycophant continuously monitors and records information from a computer's external and internal environments using sensors. We store this data in a database and *mine* this user-context data to predict user preferences (behavior) and enhance user interaction. Sycophant uses XCS to mine user context data and constructs a mapping from user-related contextual features to user-preferred reminder types.

More generally, we want to investigate machine learning approaches for building a suite of improved human computer interfaces. In our earlier investigations, we limited the classifier population size of XCS to an estimate of the number of rules used by a decision-tree learner and were successfully able to learn the mapping for a single user (to appear). For the results presented here, we do not limit the classifier population, we operate XCS with and without *condensation* enabled and we evaluate our approach from data collected from three users. Condensation is a process of extracting a minimal subset of classifiers by reducing the size of the classifier population. This is done by running the genetic algorithm within XCS and setting cross-over and mutation probabilities to zero [20]. We validate the effectiveness of our approach by evaluating Sycophant's performance in learning this mapping task for all three users. We also compare our GBML approach for user-context learning with a well-established decision tree learner.

We break down our results presentation into two sets of supervised learning tasks; the first task is to decide whether or not to generate a reminder and the second task is to choose a reminder from amongst four reminder types. Our results on a data set that has user-context information from three users shows that XCS achieves a test-set accuracy of 88 percent for the task of predicting the type of reminder to use from among four reminder types. This test-set performance is significantly better than that of a decision-tree which has an accuracy of 82 percent on the same task. On the same data set, for the task of learning to decide whether or not to generate a reminder, we get an accuracy of 87 percent on the test-set using XCS and this accuracy is statistically the same as

that of a decision-tree learner. We obtain similar results with a reduced classifier population size after running XCS in the condensation mode. XCS's test set performance across different data sets validates our user-context data collection methodology for learning user-preferred reminder types.

Section 2 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 various reminder generation tasks. We discuss the lessons learned from our experiments in section 7. The final section contains directions for future work.

2 Related Work

Previous work in the area of context-aware applications have mainly addressed the issues of user-attention management and constructing models for user interruptibility. Horvitz and Apacible have built models for predicting the cost of interrupting a user and applied machine learning techniques to produce statistical models for inferring the state of interruptibility of a user [11]. Hudson, Fogarty, Atkeson et al. have conducted Wizard of Oz studies and collected data from a subgroup of users to build sensor-based predictions of interruptibility [7]. Bailey and Adamczyk have shown that a user's attention must be carefully managed among competing applications and that this management is necessary to mitigate any disruptive effects of necessarily interrupting a user [2]. *REBA* has demonstrated the need for systems to be context-aware for anticipating user actions [14].

John Holland introduced a domain-independent rule-based machine learning complex adaptive scheme, a Learning Classifiers System (LCS) and laid a comprehensive foundation for genetics-based machine learning techniques [8, 9]. Simplifying Holland's classifier system, Wilson introduced XCS, a learning classifier system derived from his own ZCS and Animat programs [18, 19, 20]. XCS has been successfully used for learning Boolean functions by Kovacs and Wilson in the area of data mining [20, 21]. Barry and Saxon further tested XCS on the Monk's problems [17]. Wilson evaluated XCS's performance on the Wisconsin Breast Cancer data set and Holmes did the same on epidemiological data in the domain of medicine [22, 10]. Decision-tree algorithms for example, *Weka*'s J48, an implementation of Ross Quinlan's C4.5 are also widely used in data-mining applications [16, 23].

Our approach leverages ideas from machine learning and context-aware systems. We use XCS to learn the mapping from user-context features to reminder types for constructing a user model and evaluate the feasibility of using XCS for this context learning task by directly comparing XCS's performance to that of J48. For a user, Sycophant learns whether or not to generate a reminder (two-class reminder problem) and also learns to predict the type of reminder preferred by her from a set of four reminder types (four-class reminder problem). This pre-

diction is based on the XCS-generated user model built by using context-data for that particular user.

3 Sycophant

A user's environment provides a rich source for internal and external contextual information. We designed Sycophant as a context-aware calendaring application which can generate four different types of reminder for scheduled appointments as described in section 1. In this section, we describe the way we structure Sycophant as context-aware application, the sensors which Sycophant uses for gathering user-context data, the methodology we use for creating context-data sets and, how Sycophant is used as a calendar.

3.1 Architecture

Sycophant's main architecture is comprised of sensors, the machine learning component and, the calendaring application. The calendaring application runs as a separate process while the sensors collect data from the computer (internal context) and from the computer's immediate vicinity (external context). All our sensors operate in a binary mode, that is, they only detect the presence or absence of a stimulus. For example, when the motion sensor detects motion it logs a value of 1 to the context data and logs a value of 0 otherwise. Figure 1 depicts Sycophant's architecture.

3.2 Sensor Details

We continuously collect binary activity data from the following sensors and store this information in a database along with the appointment data. Sycophant's sensors are:

1. Motion-sensor: We use the *motion* software package with a cheap USB Logitech web-camera to detect the presence of motion in the immediate vicinity of a user's environment [1].
2. Speech-sensor: We use the Sphinx Speech Recognition System for detecting the presence or absence of speech in the user's environment [5]. We do not attempt to recognize or parse the detected speech, therefore our process of detecting speech is fairly accurate.
3. Processes-sensor: The users under observation actively use the following five processes: `java`, `bash`, `gnome-terminal`, `mozilla` and `xscreensaver`. We keep track of each of these processes' state and consider tracking a process as monitoring one sensor, that is, we have five process sensors.
4. Mouse-sensor: We monitor the mouse activity with this sensor.
5. Keyboard-sensor: Similar to the mouse sensor, this sensor monitors the keyboard activity.

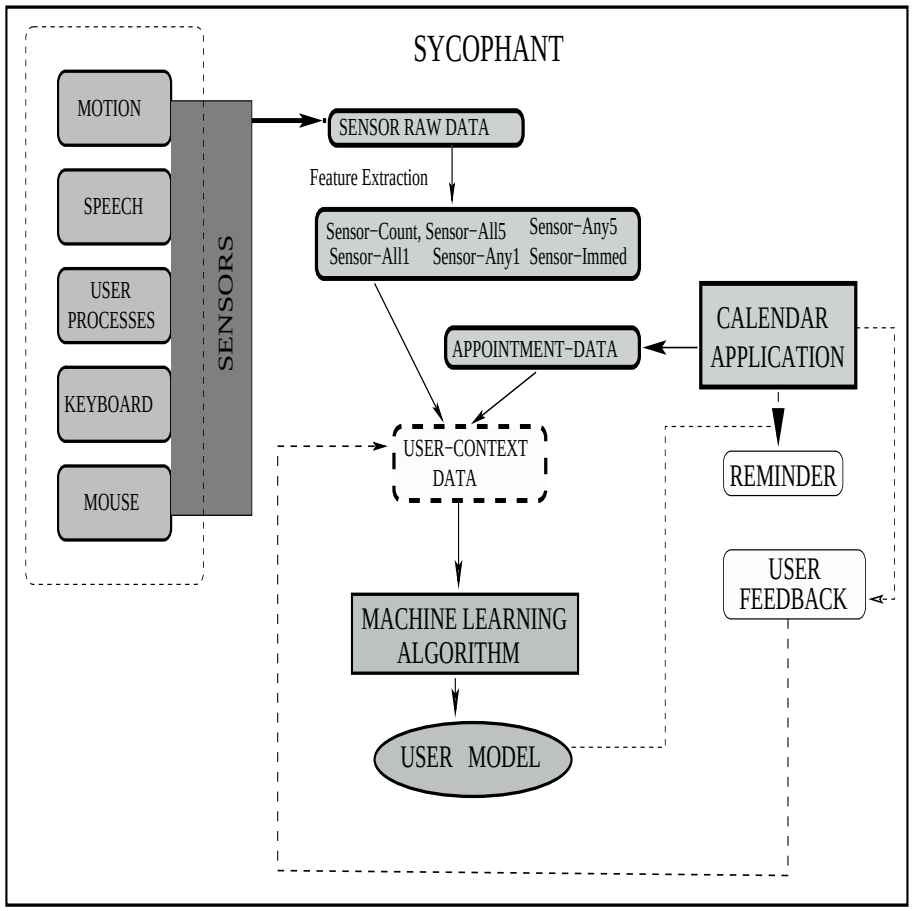

Fig. 1. Sycophant: Architecture

We have four sensors for motion, speech, keyboard, and mouse. An additional five sensors monitor active user processes; we therefore end up with a total of nine sensors for gathering user-context data.

At the scheduled time for an appointment, Sycophant generates a reminder for that particular task. We consider generating a reminder as an actuator action. With simple sensors providing richer user-context for Sycophant, even without knowing who is there or what is being said, user interaction can be improved. For example, Sycophant can learn the answers to the following questions if it was *serving* Jane:

- If there is someone else in the room other than Jane, should I remind her of a pending appointment?
- Will Jane prefer both a visual and voice reminder for her weekly meetings with her advisor?
- If Jane is not around should I pop up a scheduled reminder?
- Should I use a voice reminder for Jane if she is busy using a keyboard and mouse?

3.3 User-Context Data

We monitor our sensors for activity every fifteen seconds and store this activity data into a file. Next, we extract the following context features from the raw data for every sensor to create our user-context data sets [12]:

- All5: if the sensor was active during all the fifteen second intervals during the last five minutes.
- Any5: if the sensor was active during any of the fifteen second intervals during the last five minutes.
- All1: if the sensor was active during any of the fifteen second intervals during the last minute.
- Any1:if the sensor was active during any of the fifteen second intervals during the last minute.
- Immed: if the sensor was active during any of the last fifteen seconds.
- Count: the number of times the sensor was active during the last five minutes.

Each sensor therefore provides six features. The nine sensors as described in section 3.2 result in a total of 54 features. We also consider the appointment time and user-identifier as attributes. Therefore, we end up with a total of 56 features for our user-context data. Three users, *user-1*, *user-2* and *user-3* used Sycophant over a period of six to eight weeks. Our user-context data sets collected from these users had 347 exemplars from *user-1*, 354 exemplars from *user-2* and 352 exemplars from *user-3*. We therefore had a total of 1053 exemplars from all these three users. We label the data set from *user-1* as *ContextData-1*, *user-2*'s data as *ContextData-2*, *user-3*'s data as *ContextData-3*and the combined data from both users as *ContextData-all-users*.

Here is an exemplar from one of the data sets:

```
 user-3, 13.00, 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, 3
```

User-identifier is the first feature and the time of appointment is the second feature, The remaining set of features in groups of six represent the Sensor-Count, Sensor-All5, Sensor-Any5, Sensor-All1, Sensor-Any1 and Sensor-Immed. We order these sets of sensor features in the following sequence: motion, speech, process, keyboard and mouse. The last feature is type of reminder preferred by a user. We obtain this value through user feedback whenever Sycophant generates a reminder.

3.4 Using Sycophant

In our current research, users set their appointments mimicking their regular work-days for using Sycophant. These appointments included reminders for attending classes, meetings, conferences, showing up at colloquia dispensing free food and soda, taking regular coffee breaks and a few personal appointments. Sycophant decides on a reminder type (chosen from amongst the four reminder types) and generates a reminder for the task at the scheduled time and the user provides his feedback indicating the type of reminder he would have preferred. A hand-coded set of rules indicate the type of reminder for Sycophant to use at the initial phase when no user-context is available. Once sufficient user-context data is available, Sycophant generates a model for the preferred types of reminder for that particular user using a machine learning algorithm. Figure 2 gives a snapshot of the interface used for setting reminders.

4 A Brief Description of XCS

Wilson derived XCS from his Animat and ZCS programs [18, 19]. XCS differs from a traditional learning classifier system in that a classifier's fitness in XCS depends on the prediction of its expected payoff instead of the actual prediction itself. Subject to an accuracy criterion, XCS is shown to evolve minimal, maximally general and accurate model for a learning task [13]. We convert our user-context data into an appropriate representation which can be used by XCS. Next, we provide exemplars from this context-data set to XCS and use the classifier system for an independent single-step learning task where the system learns to decide on the type of reminder to be used for that particular exemplar. Figure 3 shows the architecture of XCS.

XCS consists of population of classifiers. Each classifier has the following important attributes:

1. Condition: defined over the ternary alphabet $\{0, 1, \#\}^L$, where L is the length of the bit-string representing a classifier. The meta-character # matches either a 0 or a 1. A classifier's condition is an input state sensed from the

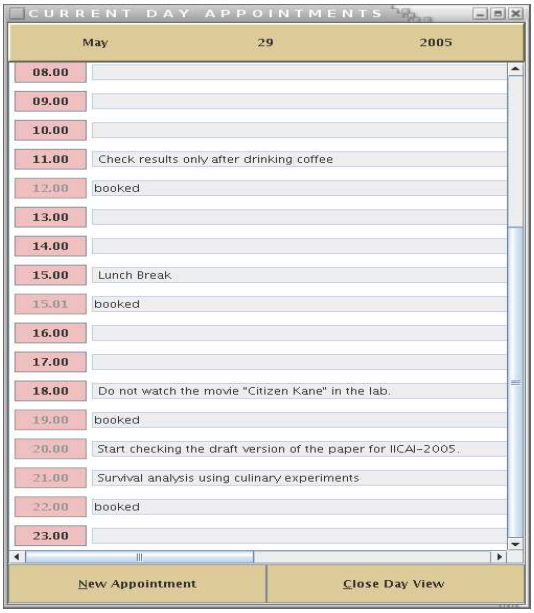

Fig. 2. Sycophant: Appointment Entry Interface

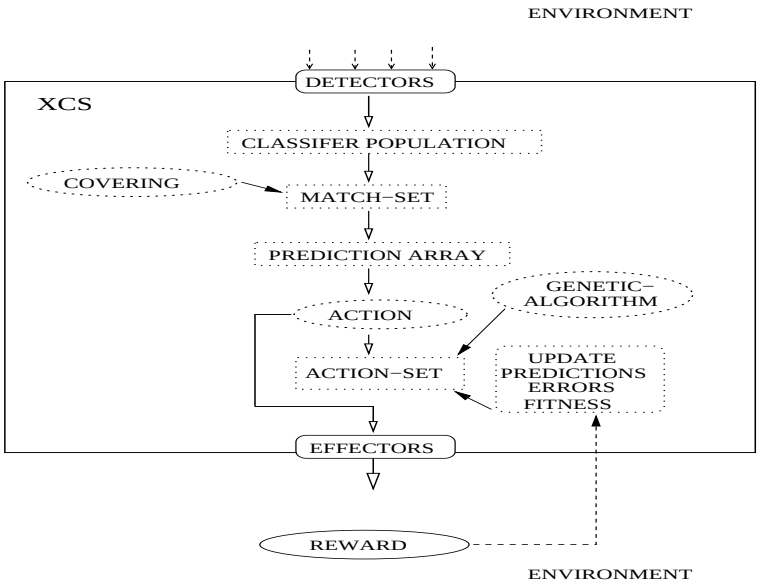

Fig. 3. XCS Architecture

environment it tries to match. In our case, a condition is an exemplar from the context-data set described in section 3.3.

2. Action: specifies an operation which a classifier can execute. For our user-context learning task, a classifier's action is the type of reminder. An action value is therefore chosen from $\{0, 1, 2, 3\}$.
3. Prediction Estimate: is an estimate of the expected pay-off if a classifier matches the environment's sensory input (a context-data exemplar) and its action is chosen by the system.
4. Prediction Error: is an estimate of the error made in predictions. This value is calculated after the reward from the environment is received.
5. Time-stamp: is a counter of the last occurrence of the genetic algorithm (GA) in an action-set to which this classifier belonged.
6. Experience: is the number of times which a classifier has belonged to the action-set since its initialization.
7. Action set size: is the average size of the action-sets to which this classifier belonged.
8. Fitness: represents a classifier's fitness. This is first calculated by re-estimating the attributes of the classifiers in the action-set according to the reward returned by the environment. This reward is a result of the system executing a particular action. Subsequently, predictions and errors are updated based on the reward and for each classifier, accuracy is estimated. Next, the accuracy estimate is converted into a relative accuracy value. This relative accuracy value in turn is used along with the learning rate (β) for updating the fitness.
9. Numerosity: is the number of micro-classifiers which this macro-classifier represents. N traditional classifiers having identical conditions and actions are represented by a single macro-classifier in XCS.

During system operation, XCS samples a user-context data exemplar as a message string from the external environment. This string is defined over $\{0, 1\}^L$ where L is the number of bits in each situation (classifier condition). Classifiers whose conditions match the environmental message become members of a set called the match-set (M). If none of the classifiers match the environmental message, the classifier system creates new classifiers with each of the possible actions and assigns these classifiers to M. This procedure is called *covering*. We initialize XCS with an empty population therefore covering occurs only at the beginning of a run in our system.

The system computes a fitness-weighted average of the predictions of each classifier in M for each of the actions present in M. During our preliminary investigation, we experimented with choosing the action non-deterministically. This method of action selection seemed to degrade the system performance. To remedy this problem we use the best-action selection mechanism for choosing an action from M and send it to the external environment as XCS's action for the externally sensed input string. A reward of 1000 is provided if the action taken by XCS is correct, 0 otherwise. Action-set (A) is created from the classifiers in M which proposed this particular action. XCS updates the different attributes of every classifier in A based on the actual reward returned by the environment.

Subsequently the fitness is also updated. The system has the option to execute a GA within A based on θ_{GA}, a threshold parameter . The populations of classifiers are thus evolved by a process of trial and error. Our GA parameter values are as follows: the probability of mutating one allele in an offspring classifier is 0.4, the probability of applying crossover in an offspring classifier is 0.67, the probability of using a don't care symbol in an allele ($P_{\#}$) is 0.33 and θ_{GA} is set to 25.

When the system senses an input, it creates a match-set and chooses a particular action to execute. After an appropriate reward is received from the environment, the attributes of the classifiers which resulted in this action being taken are updated. This procedure ensures that classifiers capable of accurately predicting the correct action to be taken tend to be reproduced more (due to their increased fitness for choosing the correct action) while inaccurate classifiers eventually get deleted. The cycle of sensing the input, choosing a particular action, running a GA after a certain threshold is done until some termination condition is met. We use the metric of randomly sampling 20000 problems from the context data training set or a training system performance of 1.0 as the termination condition in our experiment. The value of β, the learning rate is set to 0.2. Constrasted with the traditional model of a classifier system, in XCS, the GA is run on the action sets (instead of the entire population), and the classifier system has no message-list. [9, 8]. Details about the intricate process of formation of match-sets and action-sets, the execution of a GA within the system, widely used values for XCS parameters, and the methodology for updating classifier fitness and is found in Wilson [20].

5 Experimental Design

We design our experiment to answer the following questions:

1. How good is our method of collecting user-context data?,
2. How feasible is XCS for learning user-context ?
3. Is our context-learning approach generalizable across users?

As a base-rate accuracy for evaluating the performance of a learning scheme, we use the performance of Zero-R, a learning algorithm which predicts the majority class in the training set as the default class. We use a decision-tree learner on all our user-context data (*ContextData-1*, *ContextData-2*, *ContextData-3*, *ContextData-all-users*) for checking the feasibility of our user-context data collection methodology.

Next, we extend an implementation of XCS in Java to create a user-context related environment for XCS [4]. Section 3.3 describes the user-context data gathered. For each user-context data set, we transform all the contextual features except the reminder-type into an appropriate binary representation. After performing this transformation, we end up with a string like the example shown below:

```
0100010101010101010011100101100001001000010011011111010011011110 1
00111111100000000000001001111111000000000000000000000000:3
```

The value shown after the ":" is the correct action for the particular classifier condition. In the above example, the correct action is to choose a reminder of type 3. With user-context data thus transformed, we create ten stratified folds of all our user-context data sets for evaluating the training and testing performance of XCS. We also compare the performance of XCS with J48, a decision-tree learning algorithm. You should note that we evaluate both XCS and J48 on the same cross-validation training and test set pairs. By this process, we ensure that both the learning schemes are exposed to the same data samples. This design allows us to directly compare the performance of the two learning schemes.

We make XCS evolve a set of classifiers for a training fold and store these classifiers after our stopping criterion of repeatedly sampling 20000 random problems (exemplars) or when the training performance reaches 1.0. Thus XCS learns a classification model of the target concept (the type of reminder to generate) from the training data. Next, we test these evolved classifiers on the testing fold and record the performance of the system. Using the testing fold, we thus evaluate the predictive accuracy of the classification model learnt by XCS on unseen cases. Performance is defined as the percentage of correctly solved problems in the last 100 problems sampled. A problem is correctly solved if the classifier correctly predicts the type of reminder to be used for its condition component. To get a robust statistical estimate, we conduct ten runs for each pair of training-testing folds and record the performance. We repeat this procedure for all the ten folds of the cross-validation set to evaluate the system performance. The same procedure is repeated for the decision-tree algorithm.

We merge the data from all the three users in our study and evaluate XCS's performance on this data set. This metric helps us understand how XCS generalizes learning from user-context data from all the three users.

Earlier, we had investigated an approach of restricting the number of classifiers in XCS's population to an estimate of number of the rules (the number of leaves in the tree) used by a decision tree learner. 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. In this work, we therefore do not limit the classifier population size but use a maximum classifier population size of 20000 (N) for XCS. Further, we use condensation to extract minimal subset of classifiers which represent the final solution [20]. Condensation consists of running the system with crossover and mutation rates set to zero. This process suspends 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.

Currently, we avoid varying too many experimental parameters (XCS settings, data preprocessing options, learning schemes etcetera) to thoroughly explore the capabilities of our approach by simplifying the way we organize the presentation of our results. Sycophant has to learn the type of reminder preferred a user. We cast this problem into two sets of supervised learning tasks. In the first learning task set, we consider the task of reminder generation as question of whether or not to interrupt a user. We do this modification by assigning reminder type 0 (do not generate a reminder) to the first class and grouping reminder types 1, 2 and 3 to the second class (generate a reminder). For the second learning task, Sycophant has to learn to discriminate between the four types of reminder. We employ this procedure across all our user-context data sets and evaluate the training and testing performance of learning schemes.

6 Results

We use a two-sample t-test with a confidence interval of 95 percent ($\alpha = 0.05$) for comparing the mean test-set performance of XCS with that of J48. For the tables presented, we measure the training and testing performance on a scale of 0.0 to 1.0. We present the test-set performance of the learning schemes on the two-class and four-class reminder problems in Tables 1 and 2 respectively.

In both the tables, the first column represents the user-context data set on which the reminder generation task was learnt, the second column represents the performance of the Zero-R learning algorithm which we use as the base-rate for evaluating the performance of a learning scheme, J48's performance is shown in column three, XCS's performance is shown in column four, XCS's performance with condensation is shown in column five along with the number of classifiers present after condensation, the sixth column is the inference from a two-sample t-test comparing XCS's performance with that of J48 on the test-set and the seventh column is a similar inference comparing the performance of XCS with condensation to that of J48 on the test-set.

From table 1, we see that both J48 and XCS significantly outperform Zero-R except in case of *ContextData-3*. Closer analysis of *user-3*'s data revealed that this user preferred to be reminded with a single type of reminder most of the time. This explains the high accuracy achieved by Zero-R on this data set. Table 1 shows that J48 and XCS are capable of achieving a minimum accuracy of 74 percent across all the user-context data sets. This clearly validates our methodology for gathering user-context and creating context-data sets to learn user-preferred reminder types. Statistical tests provide the inference that XCS equals the performance of J48 across all the user-context data sets and also XCS does better than J48 on *Context-Data-2*. We notice that on *ContextData-all-users*, both the learning schemes achieve a base accuracy of 87 percent thereby showing that it is possible to learn preferences for reminder types across three different users. We plan to gather more context data from diverse population of users to strengthen our claim for generalization of our learning techniques across different users.

Table 2 shows that XCS with and without condensation enabled outperforms J48 across three user-context data sets, however the performance degrades on the *ContextData-user-3*. After condensation, *Context-Data-1* also seems to have more classifiers than what we expected. We strongly suspect that condensation phase could have been triggered prematurely for both these cases, thereby leaving a non-optimal subset in the classifier population. We intend to investigate this issue further.

Table 1. Test set performance of different learning schemes across various user-context data sets on the two-class reminder problem.

Learning Algorithm →	II Zero-R	III J48	IV XCS	V XCS$_c$, N	Inference	Inference$_c$
Data Set ↓						
ContextData-1	0.5677	0.7428	0.7460	0.7406, 5059	Equal	Equal
ContextData-2	0.5988	0.7885	0.9800	0.9300, 9678	Better	Better
ContextData-3	1.0000	1.0000	1.0000	1.0000, 4722	Equal	Equal
ContextData-all-users	0.6115	0.8685	0.8791	0.8788, 7699	Equal	Equal

Table 2. Test set performance of different learning schemes across various user-context data sets on the four-class reminder problem.

Learning Algorithm →	II Zero-R	III J48	IV XCS	V XCS$_c$, N	Inference	Inference$_c$
Data Set ↓						
ContextData-1	0.5677	0.7000	1.0000	1.0000, 16289	Better	Better
ContextData-2	0.5988	0.7267	1.0000	1.0000, 9433	Better	Better
ContextData-3	0.9913	1.0000	0.9118	0.9118, 5071	Worse	Worse
ContextData-all-users	0.3884	0.8236	0.8791	0.8236, 8493	Better	Better

We provide all the experiment related files at
http://www.cse.unr.edu/~anilk/IICAI_2005/experiment.html.

7 Conclusions

In this paper, we have outlined our methodology of using simple sensors to gather user-context information for better personalization of application programs. To show the feasibility of our approach, we presented Sycophant, our context-sensitive calendaring application which successfully performs user-context

learning (personalization) for three different users. We investigated and compared two learning schemes: XCS, a GBML technique and J48, a decision-tree learning algorithm to learn user-context and map the user-contextual features to reminder types. For learning this complex, non-linear classification function, we showed that XCS's performance (with and without condensation) equals that of J48 on the two-class reminder problem and that XCS outperforms J48 on the four-class reminder problem. XCS's predictive accuracy on unseen cases, that is, the test-set results across different users clearly indicates two things: the promise of employing genetics-based machine learning techniques for user-context learning and the potential for enhancing human-computer interaction by utilizing additional contextual information.

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. We investigated XCS's accuracy on *ContextData-user3* and found that the system error was higher when compared to the system error on other data sets on which XCS achieved high accuracy. This led us to suspect that we might have started condensation for this case prematurely. We would like to adopt a heuristic of using the system error for deciding when to start condensation and evaluate XCS's performance on the same data sets [13]. 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.

One of the limitations of using XCS is the time it takes to train the classifer system. XCS takes a few hours to generate classifiers while a decision tree takes a few minutes to generate a user model. To address this limitation we plan to train XCS off-line and store the classifiers. These classifiers can be used to decide on the reminder type when it is time to remind the user of an appointment. We are gathering data from more users to evaluate the performance of XCS and consider the scalability of our approach. In the future we want to combine expert-generated rules and machine-learned rules and use this to design better adaptive user interfaces.

We do not expect to capture all the relevant context and the model generated for a user is not going to be 100 percent accurate. There is always a scope for improving the design of Sycophant's user interface, refining our method of data collection and optimizing our system architecture to better model user-context.

Acknowledgments

The authors 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.

References

[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. V. Butz. XCSJava 1.0: An Implementation of the XCS classifier system in Java . Technical Report 2000027, Illinois Genetic Algorithms Laboratory, 2000.

[5] C.M.U. Sphinx: Open source speech recognition. http://www.speech.cs.cmu.edu/sphinx/.

[6] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware appplications. *Human Computer Interaction*, 16, 2001.

[7] 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.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[9] 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.

[10] J. H. Holmes. Learning classifier systems applied to knowledge discovery in clinical research databases. In *Learning Classifier Systems, From Foundations to Applications*, pages 243–262, London, UK, 2000. Springer-Verlag.

[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, 2001.

[23] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations.* Morgan Kaufmann, San Francisco, USA, 2000.