THE UNIVERSITY OF NEVADA, RENO

Simple User-Context for Better Application Personalization

by

Anil K. Shankar

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science in Computer Science and Engineering

Reno, Nevada

May, 2006

UNIVERSITY
OF NEVADA
RENO

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

ANIL K. SHANKAR

entitled

Simple User-Context for Better Application Personalization

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Sushil J. Louis, Ph.D., Advisor

Monica Nicolescu, Ph.D., Committee Member

Hyunjeong Lee, Ph.D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

May, 2006

ABSTRACT

Simple User-Context for Better Application Personalization

by

Anil K. Shankar

Current computer applications and user interfaces rely on sparse contextual information to learn user preferences. This thesis describes an approach to better learn user preferences using additional contextual information from cheap motion and speech sensors. We present **Sycophant**, our user-context-sensitive calendaring application. Sycophant is capable of generating four different types of reminders and primarily uses machine learning techniques for predicting the type of reminder a user prefers. To learn user preferences Sycophant maps user-related contextual features to reminder actions. We show that XCS, a genetics-based machine learning technique outperforms a well established decision tree learning algorithm on this mapping task. Our research indicates that additional external contextual information using motion and speech sensors improves Sycophant's performance for learning user preferences. Our approach for learning user context successfully generalizes across three different users.

Acknowledgments

I am grateful to my thesis advisor and mentor Sushil Louis for providing me the resources and freedom along with guidance and support during this whole process. Thanks to my committee members for their patience and valuable feedback. I thank Dr.Linda Hayes and her research group at the Dept. of Psychology for helping me with the experimental design. I am grateful to the users who volunteered to provide data by regularly using Sycophant. I thank these users for their time and cooperation. I appreciate the great systems support provided by Michael Dahl and Colin King. I also thank fellow lab members past and present – Ryan Leigh, Chris Miles, Adam Olenderski, Juan Quiroz, and Kai Xu for providing me with enlightening and controversial discussions on a wide range of academic and non-academic topics which have helped me and Sycophant participate in a successful learning process. This work was supported in part by contract number N00014-0301-0104 from the Office of Naval Research. Last, but not the least, I am deeply indebted to my family, particularly my parents for all they have given me.

Contents

List of Figures

List of Tables

Chapter 1
Motivation

We feed, heal, wait and fight computers. They grunt and stare back stupidly. We serve them instead of them serving us.

— Michael Dertouzos, *The Unfinished Revolution*

1.1 Introduction

Rapid advances have been made in the areas of machine vision, speech and natural language processing, machine learning, and human computer interaction [1, 2, 3, 4, 5, 6, 7]. Despite this ability of computers to *see*, *hear*, and *learn*, we still find that computer applications do not integrate these modalities to better interact with a user. Computers today use an internal clock, keyboard and mouse to provide input or context for applications' information processing. Operating Systems support these devices and applications access the provided context through simple Application Programming Interfaces (APIs). Current application software uses this meager context to build user models in a bid to enhance personal productivity.

Consider this scenario – the phone in your office rings when you are listening to music on your computer. *You* pause the music player to answer your phone. In the same situation, *I* might decide to stop the music. This simple senario shows that the preferred actions for an application vary from user to user in the same circumstance.

On the other hand, if *you* are talking to someone in your office and the phone rings, *you* might stop the music; *I* might pause the music player. This situation shows

that the preferred actions for an application also vary for a single user in different situations.

Wouldn't it be nice if you had a context-sensitive music player that adapted its behavior to you and executed your preferred actions in different circumstances ? For example, if you have a tendency to pause the music for answering a phone in your office, a context-sensitive music player would learn to pause the music for you whenever it detected a phone ringing in your office.

Other commonly used applications – e-mail programs, calendaring applications, text editors – share the same shortcomings as the music player. Current computer applications pay insufficient attention to user preferences, use little or no context and make weak attempts to adapt their behavior to individual user needs. These applications are neither aware of the context in which they are used nor of their environment.

In this thesis, we describe an approach to learn user preferences based on contextual information from a user's environment. Our context learning calendaring application, **Sycophant**, learns a mapping from user related contextual features to user-preferred reminder actions. We provide more details about Sycophant's architecture in chapter 3. In the next section we describe the definition of *context* that we use in our research.

1.2 What is Context?

Context awareness is a widely researched topic in the area of ubiquitous computing [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. Researchers in this area have done a significant amount of work on standardizing a clear definition of *context*. Dey gives one widely accepted definition [19] :

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.*

Applying Dey's definition of context to a desktop PC, we further define **user-context** to be any information regarding a user's presence (or absence) in the vicinity of a computer. We differentiate between *external* and *internal* user-context in our research. External user-context is information that a computer senses from the external environment. This can include the movements of the user in the immediate vicinity of the computer and the presence or absence of speech. Internal user-context is any information that a computer senses from its internal environment. This contextual information generally relates to keyboard activity, mouse usage and the activity of different processes within a user's computer.

In our research we use Dey's definition of context to address the issue of learning user preferences. We believe that context-aware applications can better *personalize*

themselves to a user. An application personalizes itself to a user if it learns user-preferences and adapts its behavior accordingly. We advance to the central claim of this thesis in the next section.

1.3 The Main Claim of this Thesis

Various sophisticated sensors – gaze tracking devices, retinal scanners, finger print recognition devices – can gather information about a user. However, these sensors are expensive and are complicated to use. In our research, we deploy simple and cheap devices like a web-camera or a microphone to collect user-context. We probe a user's environment using simple sensors and gather user-context. We do motion detection instead of object recognition or face recognition; our approach detects speech instead of doing speech recognition; we monitor the activity of the keyboard, mouse and five user processes. We provide more details about these sensors in Section 3.3. Our approach of detecting motion and speech is more robust and computationally faster when compared to approaches that do face-recognition or speech-recognition.

Even with such simple sensors, we believe that applications can learn the type of action a user prefers in diverse scenarios. For example, if you were Jane's user-interface, learning user-context might help you to answer the following questions:

- If no one is in the room should I pop up a scheduled appointment?
- Should I remind Jane of a pending appointment if someone else is in the room

talking with her?

- Should I turn down Jane's music player when the telephone rings?
- Should I pause the current song when Jane leaves the room?

The central claim of this thesis is :

Simple user-context helps applications to better learn user preferences.

Our long-term goal is to enable applications to learn user preferences based on user-context. In this thesis, we describe *Sycophant*, our context learning calendaring application. Sycophant learns to choose a correct reminder type from a set of four reminder types.

1.4 Structure of this Thesis

Chapter 2 explores related research in Human Computer Interaction (HCI). In this chapter we develop a stationary robot model of a desktop Personal Computer (PC) by integrating different ideas in HCI research.

In the next chapter we introduce the Sycophant's architecture. We describe our combination of sensors for gathering user-context data. Details about the four types of reminders which Sycophant generates and the way in which we cast the reminder prediction task into two-class and four-class problems are given in this chapter.

We present two approaches for learning user preferences: Genetics Based Machine

Learning (GBML) and Decision Trees in chapter 4. We describe XCS, a GBML technique, and J48, a decision tree, in this chapter.

Chapter 5 covers the design of our experiments for evaluating user-context learning. We outline the methodology we use for comparing the learning performance of XCS and J48 to learn user preferences in this chapter.

We provide results illustrating Sycophant's performance in learning a single user's preferences for reminder types with different machine learning techniques in chapter 6. A part of these results was published in the proceedings of *2004 International Conference on Information Reuse and Integration* [20]. Our results show that removing external contextual information from a user's environment significantly degrades Sycophant's ability for learning to predict the reminder types preferred by a user.

Sycophant's improved its performance using XCS, a GBML approach, for predicting reminders. We present results in chapter 7 which show that XCS significantly outperforms J48 on the four-class reminder problem for a single user. These results were published in the proceedings of *2005 IEEE Congress of Evolutionary Computation* [21].

We give results which demostrate the generalizability of our user-context learning approach in chapter 8. In this chapter we show how Sycophant learns user preferences for three different users and compare the performance of XCS with a popular decision tree learning algorithm for learning users preferences. A portion of these results were

published in the proceedings of *2005 Indian International Conference on Artificial Intelligence* [22]. Chapter 9 summarizes the contributions of our research for learning user-context. We explore promising directions for future research in the last chapter.

Chapter 2
Related Work In Context Aware Systems

The real voyage of discovery consists not in seeking new landscapes but in having new eyes.

— Marcel Proust

2.1 Introduction

In the first chapter we hypothesized that applications can better learn user-preferences if they pay attention to user-context. We review previous attempts made in the area of Human-Computer Interaction (HCI) to address the need for applications and user environments to be context-aware in the next section.

Hudson, Fogarty, Atkeson et al. built predictive models to infer the state of interruptability of a user [23, 24], Bailey and Adamzyck quantitatively evaluated the effect of interruptions on the productivity of a user [25], and Kulkarni's *ReBa* used sensor devices to localize a user in context-aware environments [26]. Our research consolidates all these ideas in HCI. We aim to build an application which can execute the most appropriate action by anticipating user-preferences. For learning user preferences we design a sensor-based approach for gathering user-context. Finally, we integrate the relevant research ideas in HCI to model a desktop personal computer as a *stationary robot*.

2.2 Related Work

Much work had been done in Human-Computer Interaction (HCI) to create context-aware applications and programs. In this section we review the related research in pervasive computing, quantitative evaluation of computer generated interruptions, and sensors-based approach for user modeling.

Kulkarni's *ReBa* is a reactive system which generates context-aware actions for the Intelligent Room, a project in pervasive computing [26]. *ReBa* uses information from cameras, microphones, and other sensors for choosing appropriate actions. Kulkarni based *ReBa*'s design on the following principles put forth by Rodney Brooks: *situatedness* where a mobile robot uses the external world as its own best model, *embodiment* which physically grounds an agent in the real world to deal with imperfect sensor and actuator issues, using the dynamics of agent's interaction with its environment to determine the structure of *intelligence*, and *emergence*, where intelligent functionality emerges from the interaction of components within an agent [27]. These principles emphasize a scalable, robust, decentralized and situated approach for building intelligent agents. Built upon these principles *ReBa* showed the necessity for systems to be context-aware for anticipating user actions. Kulkarni modeled the Intelligent Room as a stationary robot. Similarly we model a desktop PC as a stationary robot in our research.

Bailey and Adamczyk have shown that computer generated interruptions which

require user input or feedback have a disruptive effect on the user's emotional state as well as the user's task performance [25]. Their study showed that at the point of interruption, the degree of disruption depends upon the user's mental load. The work highlighted the need to carefully manage a user's attention among competing applications and that this management is necessary to mitigate the disruptive effects of interrupting a user.

Hudson, Fogarty, Atkeson et al. have examined approaches for constructing robust sensor-based predictions of interruptability by conducting a Wizard of Oz study [23]. They considered which sensors might be useful and how they could be constructed. In their study they used experience sampling to collect self-reports of interruptability. Next, they built statistical models predicting human interruptability and achieved an overall accuracy of 78 percent using several models. The self-reports from their initial Wizard of Oz study, where a subject was asked to distinguish between different levels of interruptability, showed that it is possible for humans to be accurate to an extent of 76.9 percent and statistical models could achieve as much as 82.4 percent accuracy. In their subsequent study they used real sensors to construct models of human interruptability for three different groups of people which included interns, managers and researchers by [24]. Their predictive models achieved an accuracy of 87, 81 and 80 percent for managers, researchers, and interns, respectively.

Our research integrates and extends all these ideas in HCI. For learning user

preferences we design a sensor-based approach for gathering user-context to minimize the disruptive effects of interrupting a user. Like Fogarty, we use real sensors in our research to gather user-context. In addition to predicting the interruptibility of a user, we also predict the type of reminder to use for an individual user.

2.3 Desktop PC as a Stationary Robot

Kulkarni's *ReBa* showed that it would be desirable for applications to infer user needs by sensing and perceiving the environment. Accordingly, in our research we model a Desktop PC as a *stationary robot* with effectors and actuators. On this stationary robot, we define an *effector* as an application action; we define any module which enables the effector to execute a user-preferred action as an *actuator*. Based on this stationary robot model of a computer, in the next chapter we present Sycophant, our calendaring application. We describe Sycophant's architecture, the sensors it employs for gathering user-context and the type of reminders which it can generate to remind a user about pending appointments and tasks.

Chapter 3
Sycophant

What we crave, what we want to see in others eyes, is that servile expression, an unconcealed infatuation with our gestures

— Emile M. Cioran

3.1 Introduction

We present Sycophant, our calendaring application in this chapter. Sycophant's goal is to generate a user-preferred type of reminder for pending appointments and tasks. This application has to learn to *please* a user by learning her preferences and hence the name *Sycophant*.

A user's environment provides a rich source for internal and external contextual information. Sycophant uses this information to learn user preferences. We describe how we structure Sycophant as context-aware application in Section 3.2. In the next section we give details about Sycophant's sensors used for gathering user-context data. The methodology we use for creating context-data sets is given in Section 3.4. Section 3.5 describes the types of reminders which Sycophant generates. The last section gives information about how different users set their appointments using Sycophant.

3.2 Architecture

Figure 3.1 depicts Sycophant's architecture. Sycophant's main architecture consists of the 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. We merge the data from different sensors with the appointment data to create raw sensor data and extract different features in the data set to yield user-context data. A machine learning algorithm builds a user model based on user-context data to predict a user-preferred reminder type. Next, the user provides feedback indicating the type of reminder she would have preferred. We append this feedback information to the user-context data and a learning algorithm uses this information while building a user-model. We next describe Sycophant's sensors used to gather user-context information.

3.3 Sensor Details

Sycophant uses nine sensors for gathering user-context information. This application continuously collects binary activity data from the following sensors and stores this information in a database along with the appointment data.

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 [28].

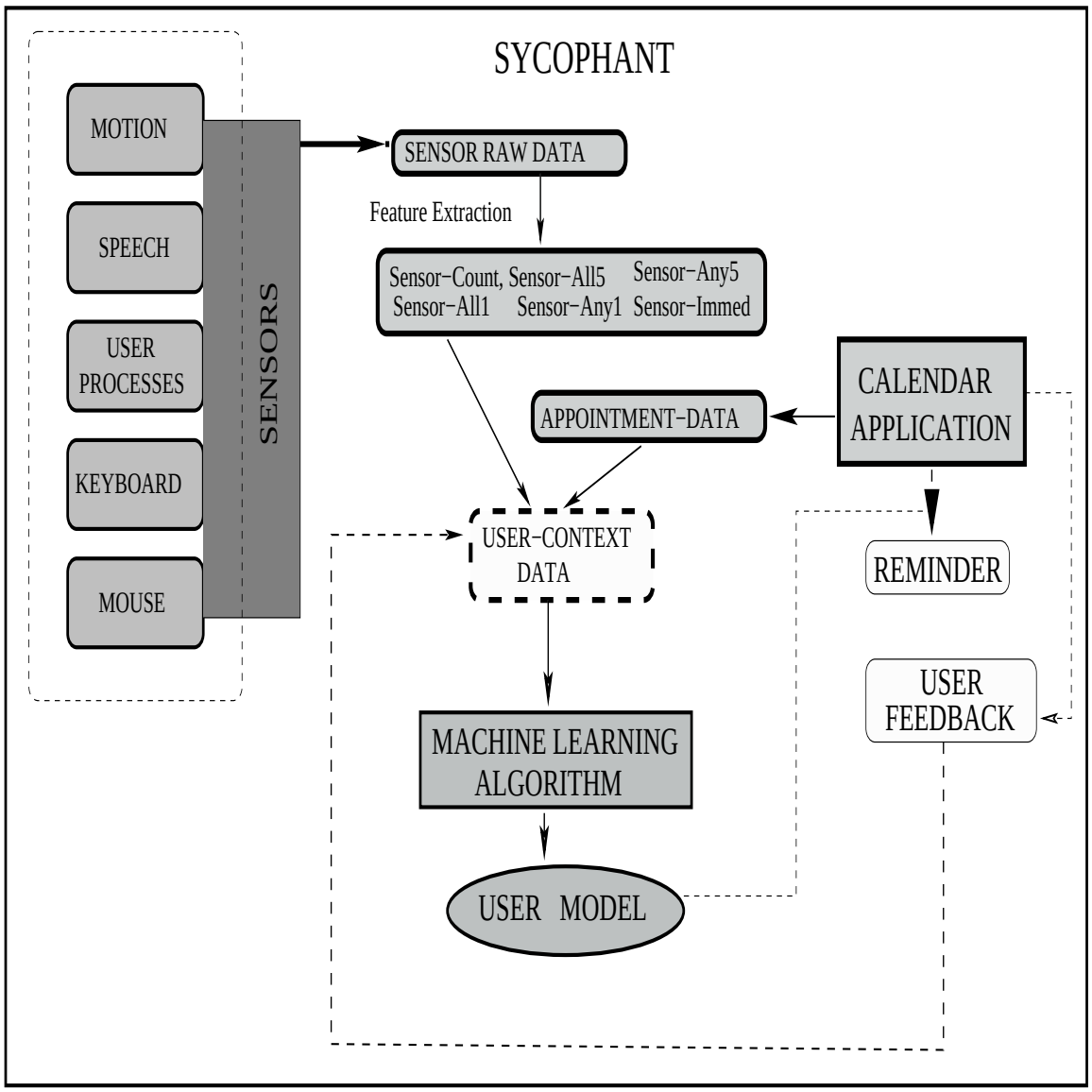

Figure 3.1 Sycophant: Architecture

Speech-sensor : We use the Sphinx Speech Recognition System for detecting the presence or absence of speech in the user's environment [29]. We do not attempt to recognize or parse the detected speech; therefore our process of detecting speech is fairly accurate.

Processes-sensor The users under observation actively use the following five processes: `java, bash, gnome-terminal, mozilla`, and `xscreensaver`. We label these processes as `process1, process2, process3, process4` and `process5` respectively in our context data set. 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.

Mouse-sensor : We monitor the mouse activity with this sensor.

Keyboard-sensor : This sensor monitors the keyboard activity.

We have four sensors for motion, speech, keyboard, mouse, and additional five sensors monitor active user processes. We therefore end up with a total of nine sensors for gathering user-context data. Based on our definition of internal and external user-context in Section 1.2, we consider the motion and talk sensors as sources of *external user-context* information. The five process sensors, keyboard sensor and mouse sensor serve as sources of *internal user-context* information.

3.4 Collecting User-Context Data

To collect 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 [23]:

All5 : the sensor was active during all the fifteen second intervals during the last five minutes.

Any5 : the sensor was active during any of the fifteen second intervals during the last five minutes.

All1 : the sensor was active during all of the fifteen second intervals during the last minute.

Any1 : the sensor was active during any of the fifteen second intervals during the last minute.

Immed : 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 3.3 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. During the process of data collection, we ensured that a user's identity remained anonymous and we stored the data in a secure place. The data from *user-1* initially consisted of 323 exemplars; in our preliminary investigation we used this data to consider feasibility of our approach for learning user preferences. We label *user-1*'s reduced data set as *ContextData-0*. We collected more data from this user at a later stage which resulted in *Context-Data-1*. Our user-context data sets collected from three 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 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 all 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
```

Table 3.1 explains the meaning of these individual features. 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

Table 3.1 Organization of user-context data

Feature	Value(s)
User-identifier	`user-3`
Appointment Time	`13.00`
Motion (Count, All5, Any5, All1, Any1, Immed)	`0, 0, 0, 0, 0, 0`
Speech (Count, All5, Any5, All1, Any1, Immed)	`7, 0, 1, 0, 0, 0`
Process-1 (Count, All5, Any5, All1, Any1, Immed)	`20, 1, 1, 1, 1, 1`
Process-2 (Count, All5, Any5, All1, Any1, Immed)	`20, 1, 1, 1, 1, 1`
Process-3 (Count, All5, Any5, All1, Any1, Immed)	`20, 1, 1, 1, 1, 1`
Process-4(Count, All5, Any5, All1, Any1, Immed)	`0, 0, 0, 0, 0, 0`
Process-5 (Count, All5, Any5, All1, Any1, Immed)	`20, 1, 1, 1, 1, 1`
Keyboard (Count, All5, Any5, All1, Any1, Immed)	`0, 0, 0, 0, 0, 0`
Mouse (Count, All5, Any5, All1, Any1, Immed)	`0, 0, 0, 0, 0, 0`

in the following sequence: motion, speech, process, keyboard, and mouse. Sycophant uses these contextual features for deciding which type of reminder to generate for a user.

3.5 Sycophant's Reminder Types

The goal of Sycophant, our context-sensitive calendaring application, is to predict the correct type of reminder a user prefers for her scheduled appointments and tasks. Sycophant generates the following four types of reminders for an appointment

Reminder-type 0 : Sycophant does not generate any reminder. That is, we suppress reminder generation until a more opportune moment.

Reminder-type 1 : A visual reminder displays the appointment text in a pop-up window.

Reminder-type 2 : A voice reminder speaks out the appointment text using the Festival Speech Synthesis system [30].

Reminder-type 3 : Sycophant generates both visual and voice reminders for an appointment.

We next describe how three different users set their appointments using Sycophant and provided user-context data for evaluating our user-context based approach to learn user preferences.

3.6 Using Sycophant

Figure 3.2 gives a snapshot of Sycophant's interface that the users in our study used for setting their appointments. These appointments included reminders for attending classes, meetings, conferences, showing up at colloquia dispensing free food and soda, taking regular coffee breaks, playing computer games in the lab, taking covert naps, and a few other 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 user can close the feedback window if he does not wish to provide feedback for his reminder preference. Sycophant discarded user-context data that did not have a user's preference for reminders when it generated a model for that particular user. A hand-coded set of rules indicate the type of reminder

Figure 3.2 Sycophant: Appointment Entry Interface

for Sycophant to use during the initial phase when no user-context is available. Once sufficient user-context data is available, Sycophant generates a model for a reminder-type preferred by a user using a machine learning algorithm. We rebuilt a user model and incoporated any additional user-context data whenever Sycophant decided to reminder a user.

3.7 Summary

In this chapter we described the two main components of Sycophant's architecture, the calendaring application and the sensors used for gathering context information. We also provided details about the types of reminders that Sycophant can generate and the way in which users in our study used Sycophant. This chapter also clarified the methodology we use for integrating the data from different sensors to create user-context data. The next chapter examines how we *mine* user-context data to learn user-preferences for reminder types.

Chapter 4
Learning User Preferences

"If you invent a breakthrough in artificial intelligence, so machines can learn," Mr.Bill Gates responded, "that is worth 10 MicroSofts."

—New York Times, Monday March 3, 2004

4.1 Introduction

We consider two approaches for learning user preferences. We explain what it means when we say a program *learns* in Section 4.2. Sections 4.3 and 4.4 briefly cover Genetic Algorithms (GAs) and Learning Classifier Systems (LCS) respectively. In the following section we describe XCS and examine the working of its components. We briefly describe how decision-tree learning algorithms work in Section 4.6. Sycophant primarily uses these two machine learning approaches, XCS and Decision-Trees, for learning user preferences.

4.2 Machine Learning

Sycophant has access to user-context data which contains the contextual information about a user (as described in Section 3.4) along with the type of reminder which he preferred. Generalizing from these examples, called *training exemplars*, Sycophant learns to predict the correct type of reminder to use for interrupting a user when it is presented with a new (unseen) set of contextual features which is called a *test exemplar*. This situation of having training exemplars and the need to induce a concept

from it has been researched extensively in the field of Machine Learning. In machine learning, we primarily use example data and past experience to program computers to optimize a performance criterion. One of the goals in machine learning is to build systems that do not need explicit hard coded instructions to deal with every special circumstance. These systems should be able to adapt to circumstances based on example data and past experience.

Tom Mitchell gives one widely accepted definition of learning [31]:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

In Sycophant's case, E is the set of training exemplars which contain the user-related contextual features along with the reminder type to use for that particular exemplar. T is the task of generating an appropriate (user-preferred) type of reminder. We measure P by checking Sycophant's reminder-type prediction accuracy on a test exemplar. In the next section we briefly explain a Genetic Algorithm (GA) which forms one of the key components in genetics-based machine learning.

4.3 Genetic Algorithms

John Holland along with his colleagues and students at The University of Michigan developed the concept of Genetic Algorithms (GA). The principles of natural

selection and natural genetics form the basis for search in a GA. A GA initializes its search from a randomly generated set of strings which constitute the initial population. Selection, crossover, and mutation are the operators used in guiding search within a GA. Evaluating, selecting, and recombining strings in the population is an iterative process. The genetic search process continues until some termination condition is met.

A structured, yet randomized, exchange is combined with survival of the fittest among string structures for guiding robust search in a GA. Bits and pieces of the fittest of the old strings are used in creating a new set of strings in the next generation [32]. For the evaluation of each string, a fitness function is defined which helps the GA to discriminate between two candidate strings. This fitness function generally depends on the problem which the GA is trying to solve. A very low probability operator that just flips a specific bit is used for mutation. Crossover is a high probability operator used to exchange parts of reproducing strings between points on the strings participating in reproduction. The string operations of crossover and mutation generate new solutions for evaluation. Strings (solutions) that perform poorly are filtered out probabilistically. Eventually, the population converges to highly fit solutions.

Algorithmically, if `Pop(i)` is the population of strings at generation `i`, a GA can be described as follows [33]:

i = 0

```
initialize Pop(i)
evaluate Pop(i)
while (termination condition is not met) do
    begin
        select Pop(i+1) from Pop(i)
        recombine Pop(i+1)
        evaluate Pop(i+1)
        i = i+1
    end
```

A GA forms a crucial component of a Learning Classifier System (LCS). We describe this genetics-based approach in the next section.

4.4 A Brief Overview of Learning Classifier Systems

John Holland was instrumental in laying the foundations for Learning Classifier Systems. Figure 4.1 shows a Learning Classifier System (LCS/CS) which is a machine learning system that learns syntactically simple string rules [34]. These string rules are called *classifiers*. An LCS consists of three main components:

1. **Rule and message system:** a production system which consists of rules of the form

   `if <condition> then <action>`

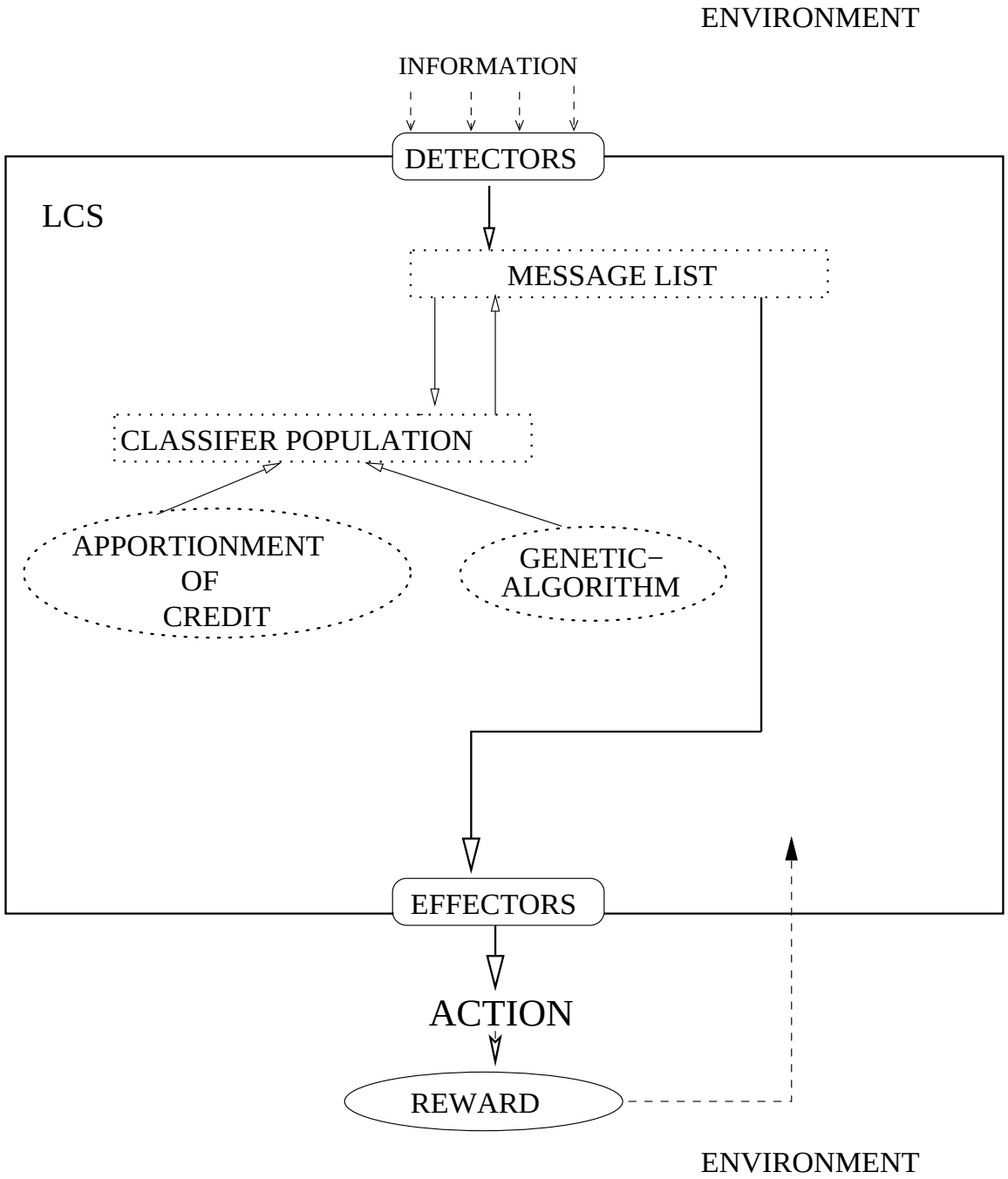

Figure 4.1 A Learning Classifier System (LCS) Architecture

The rule directs the CS to execute the `<action>` part when the `<condition>` component is satisfied.

A classifier is of the form :

`<classifier> :: <condition> : <message>`

External environmental information (messages) which flow into the CS through the detectors are posted to a finite-length message list. These messages may activate the classifiers. An activated classifier posts its message to the message list. This message in turn might invoke other classifiers and eventually may cause an action to be taken by the CS's effectors.

2. **Apportionment of credit system:** used to learn the relative value of different rules. A competition is held amongst classifiers where the right answer to relevant messages goes to the highest bidders, with subsequent payment of bids serving as a source of income to previously successful message senders. This leads to the *bucket brigade algorithm* where a chain of middlemen is formed from the manufacturer (the detectors) to the consumer (environmental action and payoff). The competitive nature of this economy ensures that good (profitable) rules survive and bad rules die off.

3. **Genetic Algorithm:** a tripartite process of reproduction, crossover, and mutation is used for injecting new, possibly better rules into the system. A GA can

be run periodically after a pre-determined threshold number of steps or depend on a metric based on the CS performance.

Fixed rule size allows genetic operators to be easily employed in an LCS. Also, parallel rule activation is possible in an LCS in contrast to the traditional model of an expert system since multiple classifiers (rules) can match the same environment message. Next we describe an implementation of LCS called XCS. Sycophant uses XCS as one of its approaches for learning user-context.

4.5 XCS

Wilson derived XCS from his Animat and ZCS programs [35, 36]. 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. 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. We describe XCS's architecture, its working, and *condensation* where XCS operates with cross-over and mutation turned off to reduce the number of classifiers (rules).

4.5.1 Architecture

Figure 4.2 shows the architecture of XCS, which consists of population of classifiers. Each classifier has the following important attributes:

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 environment it tries to match. In our case, a condition is an exemplar from the context-data set described in Section 3.4.

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 $\{0, 1, 2, 3\}$.

Prediction Estimate : an estimate of the expected payoff if a classifier matches the environment's sensory input (a context-data exemplar). Its action is chosen by the system.

Prediction Error : an estimate of the error made in predictions. This value is calculated after the reward from the environment is received.

Time-stamp : a counter of the last occurrence of the genetic algorithm (GA) in an action-set to which this classifier belonged.

Experience : the number of times which a classifier has belonged to the action-set since its initialization.

Action-Set Size : the average size of the action-sets to which this classifier belonged.

Fitness : 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.

Numerosity : 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.

4.5.2 Working

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

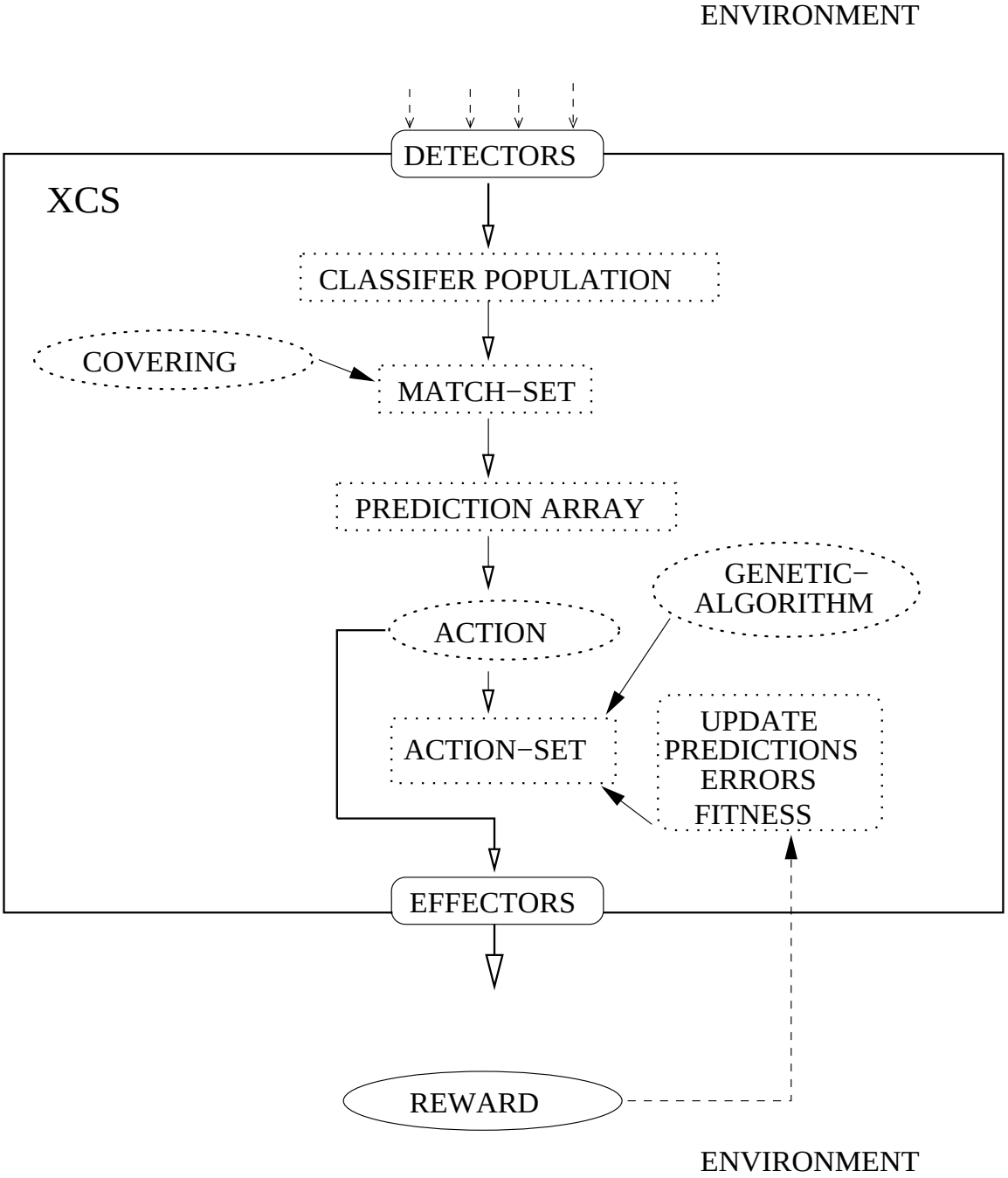

Figure 4.2 XCS Architecture

(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 ($P_{\#}$) in an allele is 0.33 and θ_{GA} is set to 25, that is, the GA is run within the action set A every 25 time steps.

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, and 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. Contrasted 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. [37, 32]. 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 are found in Wilson [38].

4.5.3 Condensation

Condensation is used to reduce the number of classifiers without trying to sacrifice the performance of 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. Therefore XCS needs a population, preferably a few thousand individuals to work with. Limiting the size of the population in XCS takes away the opportunity for the GA to experiment with recombination. We therefore do not limit the classifier population size but use a maximum classifier population size of 20000 (N) for XCS. We use condensation to extract minimal subset of classifiers which represent the final solution [38].

Condensation consists of running the classifier 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.

Apart from using XCS to learn user-context we also employ a widely accepted machine learning algorithm, a decision-tree for learning user-context. The next section gives a brief overview of this algorithm.

4.6 A Decision Tree Learner

A decision tree is a classifier which contains a structure that either indicates a *leaf* or a *decision node* [39]. A *leaf* indicates the class to which an exemplar belongs. An attribute's value gets tested at a *decision node*. A decision node has one branch and

subtree for each possible outcome of the test carried out on the attribute at that single node. For classifying a case, you start at the root of the decision tree and traverse it until a leaf (class of the case to be predicted) is encountered. At nodes which are not leaves you test a case's outcome and move down the subtree corresponding to the outcome of the test. Repeating this testing process for a case at non-leaf nodes eventually leads to a leaf node.

Figure 4.3 is a snapshot of a decision tree which Sycophant generates to predict the reminder-type for a hypothetical user. This tree is constructed after being trained

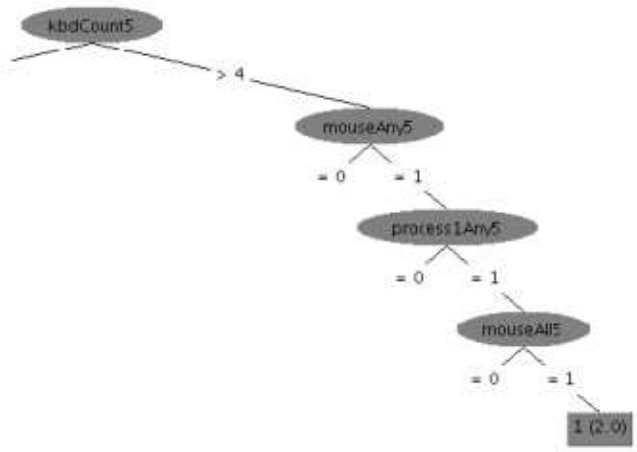

Figure 4.3 An example of a Decision Tree

on the user-context data with the nodes labeled according the convention described in Section 3.4. The root of this tree is an attribute which measures keyboard usage in

the last five minutes. If this value is greater than four, you traverse the tree to reach a node which detects if the mouse was active in the last five minutes. If the mouse was used in the last five minutes (branch value = 1), then you check if *process1* was active in the last five minutes. Once this condition is satisfied, you check if the mouse was active in all of the fifteen second intervals in the last five minutes. If this is the case, then Sycophant decides to generate a reminder of type 1 (visual reminder) for the decision tree learnt for this particular user.

Quinlan's widely used decision tree, C4.5, uses *gain ratio* measure for selecting the decision attribute at a node. *Weka*'s J48 decision tree algorithm is based on C4.5 [40]. J48 is a decision tree learner based on Quinlan's C4.5 and is a part of the *Weka* machine learning tool-kit. Sycophant uses J48 to learn the type of user-preferred reminder. We used a decision tree and interpreted the rules learnt by Sycophant for generating reminders. This was a crucial step in our research since the rules learnt reflect on the quality of the underlying data and hence the feasibility of the methodology we have set up for collecting user-context data. At this phase, our primary objective was to explore user-context learning from a descriptive data mining point of view. We describe in detail the trees and rules which Sycophant learns in section 6.5.

4.7 Summary

In this chapter, we have examined two machine learning approaches for learning user preferences – XCS, a GBML technique and J48, a decision-tree learning algorithm. We next describe how we set up experiments to evaluate our user-context learning approach.

Chapter 5
Experimental Design

It pays to be obvious, especially if you have a reputation for subtlety

— Issac Asimov, *The Foundation*

5.1 Introduction

We describe how we transform user-context data into an appropriate binary representation for XCS in Section 5.2. Section 5.3 explains our experimental set-up to evaluate Sycophant's performance for learning user-context. In this section, our experimental design answers the following questions:

- Can we use machine learning techniques to learn user preferences for reminder types?
- How can we compare XCS's performance with that of J48 for learning user preferences?
- How effective is user-context learning if we remove external contextual informaton?
- Does our context-learning approach generalize 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. For example, if you tossed a coin 10 times

and Heads showed up 8 times during these 10 tosses, Zero-R would predict Heads as the default class for this coin-toss experiment. We use a decision-tree learner on all our user-context data (*ContextData-0*, *ContextData-1*, *ContextData-2*, *ContextData-3*, *ContextData-all-users*) for checking the feasibility of our user-context data collection methodology.

5.2 Experimental Setup and Data Preprocessing

We extend an implementation of XCS in Java to create a user-context related environment for XCS [41]. Section 3.4 described our approach for gathering user-context data. In our experiment, XCS uses binary representation for the condition part of a classifier. Accordingly, 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:

```
0100010101010101010011100101100001001000010011011110100110111101
00111111100000000000010011111110000000000000000000000:3
```

The value shown after the ":" is the correct action for the particular classifier condition.

5.2.1 A Short Note on Supervised Learning

Learning to classify the reminder types into one of the four appropriate classes is a form of *supervised learning*. In supervised learning the actual outcome (correct action) for each of the training examples is provided by a teacher. That is, each exemplar is explicitly labeled with the correct class to which it belongs. For example, the correct action is to choose a reminder of type 3 for the context-data examplar given earlier in this section. This information, reminder-type 3, is obtained through user feedback and we use this feedback to label the exemplar accordingly. We provide training exemplars to a learning algorithm and generate a model which maps user-context features to reminder classes. Next, we use a test set (unseen exemplars) to check the predictive accuracy of the model generated by the learning algorithm. The task of learning a user's preference for reminder types based on his feedback is a supervised learning task.

5.2.2 Two-class and Four-class Reminder Problems

Sycophant's goal is to learn the type of reminder preferred by a user. In order to do this, we cast this problem of learning reminder types into two sets of supervised learning tasks.

1. **Two-class reminder problem**: In the first learning task we consider the task of reminder generation as a 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).

2. **Four-class reminder problem**: For the second learning task Sycophant has to learn to discriminate between the four types of reminder described in Section 3.5.

We employ this procedure across all our user-context data sets and evaluate the training and testing performance of learning schemes for these two reminder problems.

5.3 Experimental Design

In this section we explain how we set up our experiments to answer the questions we posed on experimental design in Section 5.1

5.3.1 Feasibility of learning User preferences for reminder types using machine learning techniques

We consider five machine learning algorithms and evaluate their performance on the reminder prediction tasks in the initial phase of our research. This exploratory evaluation of different machine learning algorithms helps us to select the best ones. We use the performance of this group of machine learning algorithms to evaluate the feasibility of learning user preferences. This answers the first question in Section 5.1 about our experimental design.

5.3.2 Comparing XCS's performance with that of J48

J48, our decision tree learner, uses the original representation of the user-context data set for building a model to predict the correct reminder type, while XCS uses a binary equivalent of the context data. We split the data into ten *folds* (sets) for training and testing respectively. In this process we also ensure that the distribution of classes is the same in each of the training and testing folds. Data folds created this way are called *stratified* folds. With user-context data thus transformed appropriately for these two algorithms, we create ten stratified folds of all our user-context data sets for evaluating the training and testing performance.

We make XCS evolve a set of classifiers for a training fold and store these classifiers after our stopping criterion of repeatedly sampling $20,000$ 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 system performance. 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. 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.

This evaluation on ten stratified data folds gives a measure of a learning algorithm's test set performance. We evaluate both XCS and J48 on the same training and test set pairs to compare the performance of XCS with J48. 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 and answers the second question about our experimental design in Section 5.1.

5.3.3 Evaluating the effect of external user-context on learning user preferences

We create data sets by removing external user-context (as defined in Section 1.2) to evaluate the effectiveness of using external contextual information for learning user preferences. Next, we compare the performance of J48 and XCS (operated in the condensation mode) on the two classes of reminder problems.

5.3.4 Validating the generalizability of our approach

We merge the data from all the three users and evaluate J48 and XCS's performance for learning user preferences for reminder types. This method gives us an idea about the generalizability of our approach for learning user preferences and answers the fourth question about our experimental design in Section 5.1.

5.4 Summary

In this section we delineated the methodology we use to evaluate our approach for learning user preferences for reminder types. In the following chapters, we showcase Sycophant's user-preference learning performance using different machine learning algorithms. Chapter 6 provides user preference learning results for a single user. We show that Sycophant, after being trained on user-context data, successfully learns to remind a user with his preferred reminder-type and that using external contextual information significantly improves Sycophant's performance on this reminder generation task. In chapter 7, based on user-context data collected from a single user, we present results which show that using XCS for learning to predict a reminder-type is better than using J48. Chapter 8 presents results which show that our approach for learning user preferences generalizes across three different users and that XCS significantly outperforms J48 on this task. Our results in this chapter also show that using external user-context helps Sycophant to better learn user preferences.

Chapter 6
Results From Learning User-Context for a Single User

You want a toe? I can get you a toe, believe me. There are ways, Dude. You don't wanna know about it, believe me.

— 'Walter Sobchak', *The Big Lebowski*

6.1 Introduction

In this chapter we provide results from a single user which show that user-context helps applications to better learn user-preferences. Sycophant's performance, using J48, to predict a user-preferred reminder on both the four-class and two-class reminder problems is given in the first section. Section 6.3 shows that many popular machine learning algorithms perform equally well on learning user preferences. We show that J48's performance for learning user preferences significantly degrades when we remove external contextual information in Section 6.4. We interpret a decision-tree generated by J48 for learning *user-1*'s preference for different reminder-types in Section 6.5. Finally we examine a few interesting rules which Sycophant uses to predict a reminder type for *user-1*.

6.2 Feasibility of Learning User Preferences

In data mining, the model generated directly reflects on the quality of the underlying structure of the data. That is, if you have good (bad) data then you can

generate a good (bad) model. Therefore, in the nascent phase of our research, we wanted to assay the quality of the user-context data we had collected from *user-1* (*ContextData-0*) to evaluate the feasibility of learning user-context.

Subsection 5.2.1 described the task of learning to classify a reminder type into one of its appropriate classes as a supervised classification task. In any supervised classification task, some of the features that describe the exemplars in the data may be irrelevant or redundant [42]. Such irrelevant or redundant features tend to have a negative effect on the predictive accuracy of the algorithm learning the supervised classification task. To avoid this negative effect on the learning algorithm's performance, we use *Wrapper Subset Evaluation* in *Weka* to select the most useful attributes for a decision tree and remove the redundant and irrelevant attributes [40]. The classification model generated on a data set with fewer features tends to have improved comprehensibility. Wrapper Subset Evaluation uses a classifier (decision tree) to evaluate attribute sets and employs cross-validation for estimating the accuracy of the learning algorithm for each subset of attributes.

We constructed data sets with a reduced number of features to identify the most useful features for learning user preferences after ranking the individual features based on *information gain ratio* criterion [39]. We gradually removed features from *Context-Data-0* until we detected a significant change in the performance of J48. For this investigation, we considered the top 29 features in one set, the top 25 in the next set,

and the top 20 in the last set. Next, we compared the performance of J48 on these ranked data sets as well as on the complete data set with all the features.

Figure 6.1 shows the performance of J48 on different data sets derived from *ContextData-0* with varying number of features for the four-class reminder problem. J48 correctly classified 64.4 percent of the instances and generated 35 rules for the complete data set with 55 features. On the reduced data set with 25 features J48 generated 36 rules and achieved 62.5 percent accuracy. We also see that the performance of J48 remained about the same on the complete data set with all the features and the reduced features data set with 25 features.

We found the following top 25 features:

> *Keybd-Count5,Keybd-Any5,Mouse-Count5,Mouse-Any5,Keybd-Any1,Mouse-Any1,Keybd-Immed,Mouse-Immed,MotionCount5,Motion-Any5,Motion-Any1, ApptTime,Mouse-All1,Keybd-All1,Motion-Immed,P3-Count5,P1-Count5,P2-Count5,P1-All5,P2-All5,P4-All5,P4-Count5,Talk-Count5,Motion-All1.*

Based on the division of user-context into external and internal user-contexts (as given in Sub-section 1.2), you should note here that the attributes related to external user-context (motion, talk) feature predominantly in the top 25 features. This suggests that external user-context might have a significant beneficial effect for learning user preferences.

Next, we constructed a confusion matrix to examine J48's ability to differentiate

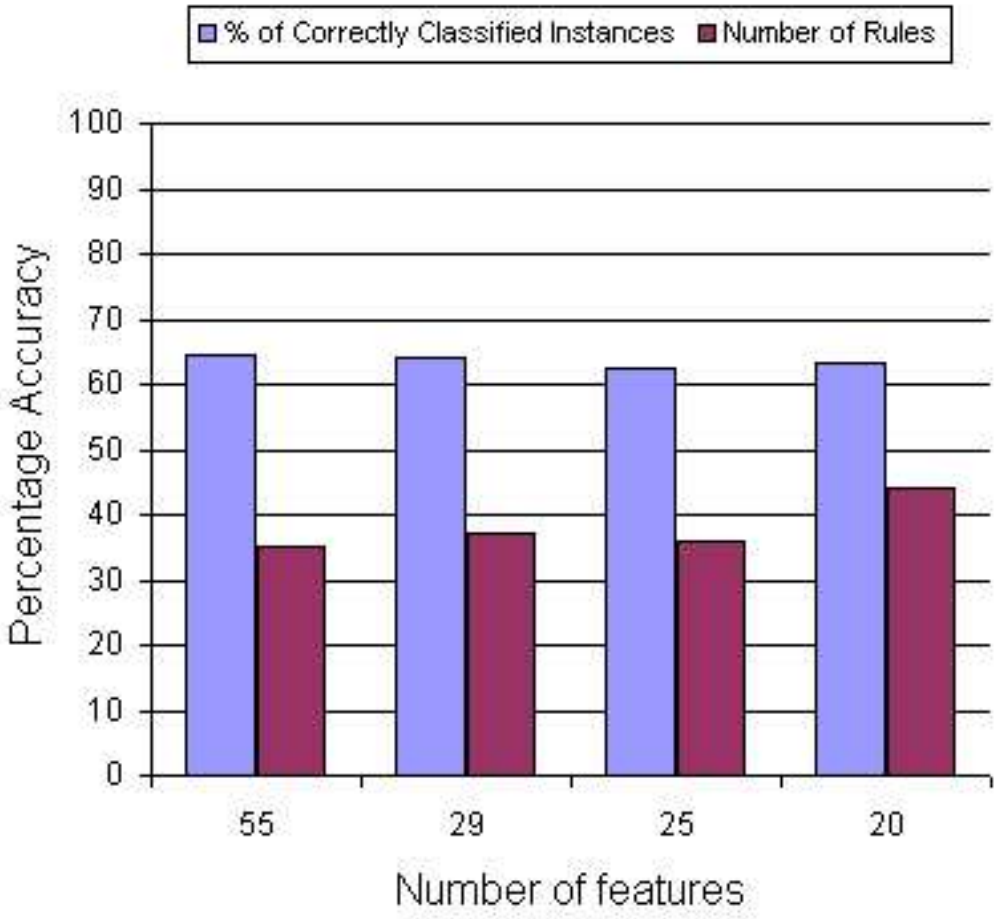

Figure 6.1 Performance of J-48 on ranked data sets with different number of features on the four-class reminder problem.

between each of the four reminder types. Table 6.1 shows the confusion matrix obtained for the complete data set with all the 56 features.

Table 6.1 Confusion Matrix for the four-classes of reminders for the full featured data set with 56 features on the four-class reminder problem.

Predicted ⇒ Actual ⇓	0	1	2	3
0	170	7	7	4
1	19	20	8	15
2	11	6	4	10
3	6	17	5	14

Table 6.2 Confusion Matrix for the four-classes of reminders for the reduced data set with 25 features on the four-class reminder problem.

Predicted ⇒ Actual ⇓	0	1	2	3
0	165	8	10	5
1	18	24	5	15
2	13	8	2	8
3	10	14	7	11

The elements along the principal diagonal $(170, 20, 4, 14)$ are the true-class values. The non-principal diagonal elements represent the misclassifications. From this confusion matrix, we see that J48 correctly classifies 64.40 percent of the instances. The first row shows that J48 can predict with a 90 percent accuracy (170/188), when to not interrupt a user with a reminder. The misclassifications in the confusion matrix show that J48 is not able to discern very accurately when to use reminder types $1, 2,$

and 3. From Table 6.2 we see that J48's performance is similar on the reduced set with 25 features. Here the overall accuracy of J48 is 62.5 percent with 87 percentof type-0 (165/188) being correctly classified.

We pooled reminder types 2 and 3 into reminder type 1. Now, Sycophant had to decide whether or not to generate a reminder. Table 6.3 and Table 6.4 show the confusion matrices for the two-class problem.

Table 6.3 Confusion Matrix for the two-class reminder for the full featured data set with 56 features on the two-class reminder problem.

Predicted ⇒ Actual ⇓	No-Reminder	Generate-Reminder
No-Reminder	162	26
Generate-Reminder	36	99

Table 6.4 Confusion Matrix for the two-class reminder for the reduced data set with 25 features on the two-class reminder problem.

Predicted ⇒ Actual ⇓	No-Reminder	Generate-Reminder
No-Reminder	164	24
Generate-Reminder	32	103

J48 improved its performance to 80.81 percent in case of the data set having all the features and to 82.66 percent on the reduced feature data set. J48's predictive accuracy on the two-class problem of deciding whether or not to generate a reminder (interrupt) for a user is comparable with Fogarty's results on predicting the state

of interruptability of a user [23, 24]. At this stage, Sycophant is more successful in solving the two-class problem than the four-class problem of choosing between the four types of reminders.

In the next section we compare the performance of different machine learning algorithms on the two-class and four-class reminder problems to decide on the most appropriate algorithm(s) to use for learning user-context. We consider an algorithm to be appropriate if it has high predictive accuracy and also helps us interpret how Sycophant learns user preferences.

6.3 Performance of Different Machine Learning Algorithms

Encouraged by J48's performance for learning user preferences, we next compared the performance of different machine learning algorithms on the same task. We evaluated the performance of the following machine learning algorithms on the original data set and on the one with the 25 features:

Zero-R : a primitive learning scheme which predicts the majority class in categorical data or average class if the class is numeric.

One-R : generates a one level decision tree which tests only one particular attribute and forms a set of rules based only on that attribute [43].

J48 : builds a C4.5 decision tree as described in Section 4.6.

Bagging : creates artificial data sets from the original data set and applies a decision tree inducer on each of them. The classifiers generated vote for the class to be predicted.

LogitBoost : uses a learning algorithm for numeric prediction and a combined model is formed which is then used for classification [44].

NaiveBayes : selects the most likely classification based on a set of attribute values using prior probabilities and conditional densities of the individual features.

We show the comparison of these algorithms on the four-class reminder problem in Figure 6.2. Figure 6.3 shows the performance of the same algorithms on the two-class reminder problem. We use Zero-R's performance as the base rate for evaluating the performance of the remaining four machine learning algorithms. All the learning algorithms outperform Zero-R and these algorithms have almost the same performance on both the four-class and two-class reminder problems. We also noted that *One-R* chose keyboard usage as the most useful feature. This result adequately characterizes the user under study who has significant keyboard usage during regular hours. The amount of keyboard usage can therefore be considered as an indicator for the level of interruptability of this user while trying to generate a reminder.

J48 can generate human-readable rules and its performance matches that of the other machine learning algorithms which we explored on the data from *user-1*. There-

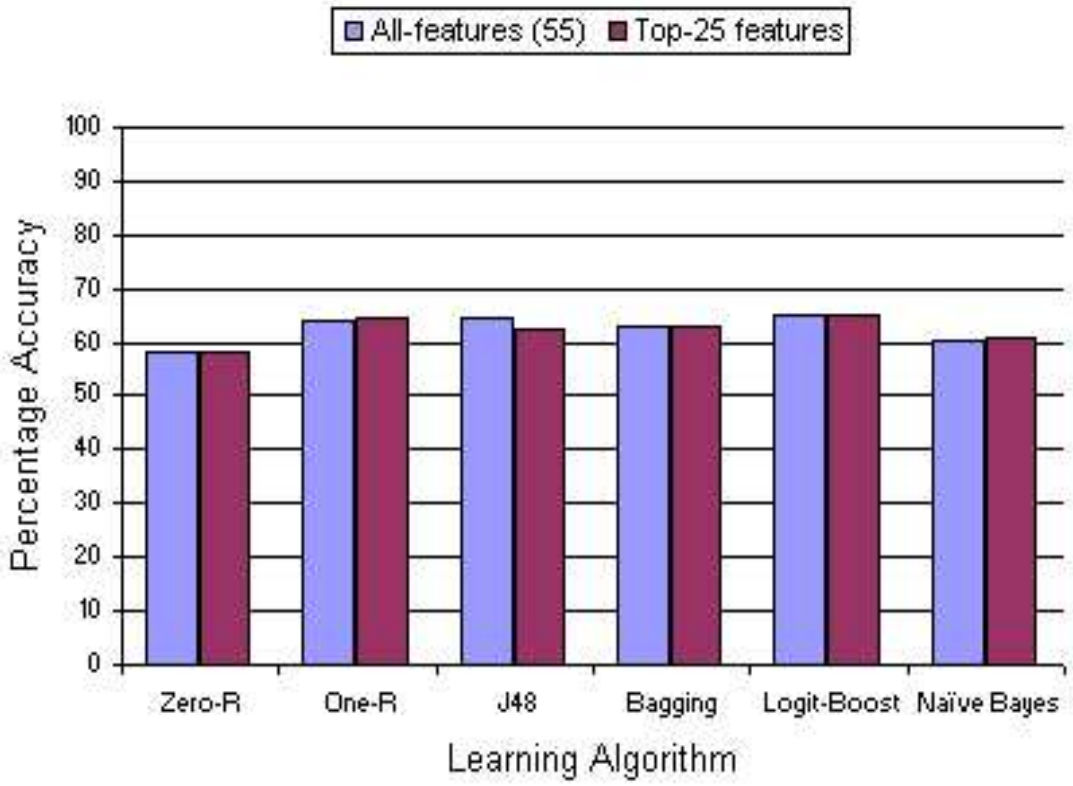

Figure 6.2 Comparison of different learning algorithms on the complete data set and reduced features data set on the four-class reminder problem.

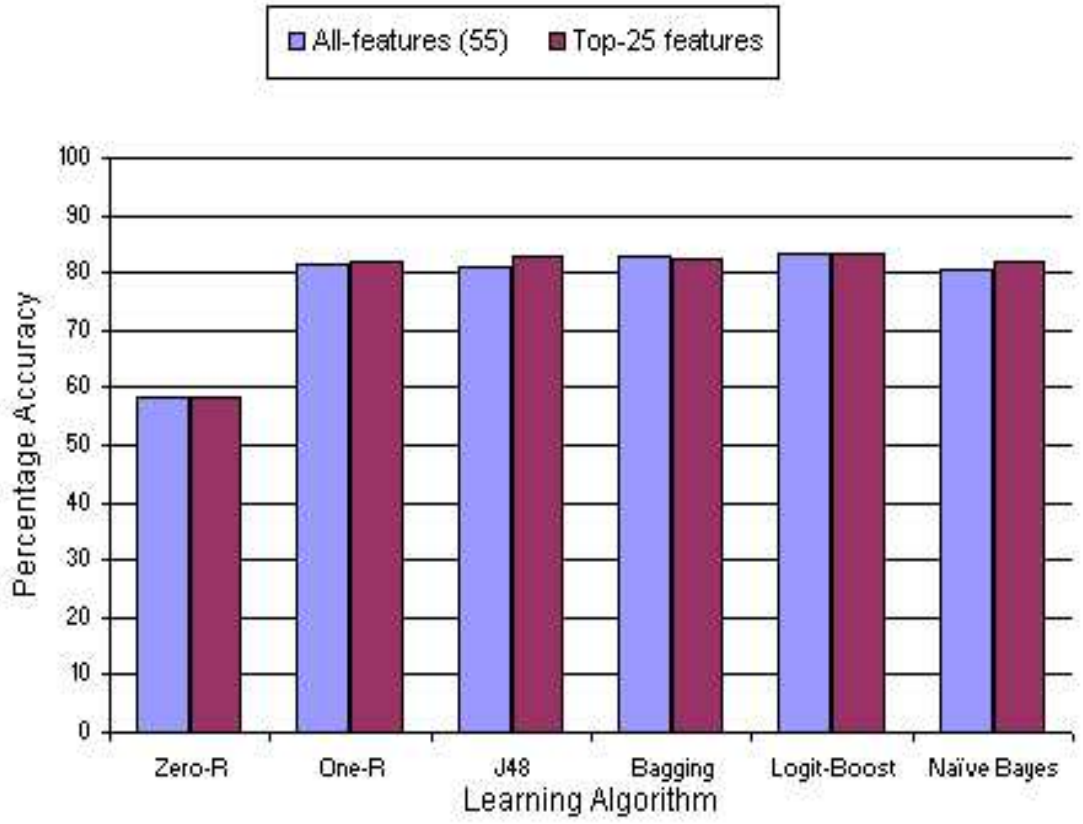

Figure 6.3 Comparison of different learning algorithms on the complete data set and reduced features data set on the two-class reminder problem.

fore, we regarded J48 as a desirable approach for learning user preferences. In the next section, we present results which show that removing external user-context significantly degrades the performance of J48 for learning reminder preferences.

6.4 Importance of External Context

Sycophant senses both the internal and external environments of a user to learn his preferences. In this section we evaluate the usefulness of external context for learning user preferences.

Table 6.5 shows the benefit of external contextual information for learning user-context on *Context-Data-1*. In this table, the first column represents the reminder

Table 6.5 J48's user-context learning performance for *user-1* showing the effect of external contextual information

Reminder Problem	Performance (percent) External Context		Distribution Of Reminders	Inference
	present	absent	Types [0, 1, 2, 3]	
Four-Class	0.6743	0.6599	197, 32, 111, 7	Worse
Two-Class	0.7431	0.7175	197, 150	Worse

problem on which we tested J48. The comparison of J48's performance for learning user-preferences for reminder types with external user-context is given in the second column. The third column shows the performance of J48 after external context is removed. We show the distribution of the four different types of reminder in *Context-Data-1* in the fourth column. The four numbers indicate the frequency

of reminder types 0, 1, 2, and 3 respectively. The last column is our statistical inference checking how J48 performed on the reminder-class problems after external user-context was removed from *Context-Data-1*. J48's performance degrades on both the two-class and four-class reminder problems when external user-context is removed. These results clearly suggest that external user-contextual information is favorable for learning user preferences for reminder types.

Examining the rules generated by J48 gives us an idea about the quality of our context data and thereby shows the potential of our approach for learning user preferences. We next examine a decision-tree generated for learning *user-1*'s reminder-type preferences and examine four rules generated for the same user.

6.5 Tree Generated by J48 for *user-1*

Figure 6.4 shows a partial snapshot of *user-1*'s decision tree on the four-class reminder problem which Sycophant learns from *Context-Data-1*. We show a similar snapshot of a different portion of the same tree learnt by Sycophant in Figure 6.5. The number of leaves is an estimate of the number of rules used by J48. The complete tree generated for *user-1* had 27 leaves. We use *Wrapper Subset Evaluation* in *Weka* to select the most useful attributes for a decision tree [40]. This attribute subset selection process picked the following attributes as the most useful attributes for J48:

```
appointmentTime, motionCount5, motionAny5, motionAll1, motionAny1,
process2Count5, process2Any5, process2All1, process2Any1, process2Immed,
kbdCount5, kbdAny5, kbdAny1, kbdImmed, mouseCount5, mouseAll5, mouseAny5,
mouseImmed.
```

Here, `process2` is the `bash` shell process on Linux. Notice that user-related attributes, the sensor features that indicate presence or absence of the user (motion, keyboard, and mouse features) and the applications which the user regularly uses (`bash` process) are selected as the most useful features for learning user preferences. This selection of user-related attributes supports our claim that user-context is necessary and beneficial for learning user-preferences. We interpret a few rules generated by the decision tree in the next section.

6.6 Rules Learned by Sycophant for *user-1*

Examining Figure 6.4 shows that one of the motion attributes (`motionAny5`) is selected as the root (most useful feature) for the decision tree. We present four interesting rules which Sycophant learns based on this decision tree. We interviewed *user-1* to interpret the rules which Sycophant generated for this user.

Rule-1 If the user is present (`motionAny5`) and the mouse is actively used in the last five minutes (`mouseAll5`), the decision is to use a voice reminder (reminder type 1) for interrupting the user. In this case, browsing the Internet made this user prefer a voice reminder.

Rule-2 If the user is present (`motionAny5`) and the user is not using the `bash` (process2All1 = 0) process but using the `java` process sparingly in the last minute (`process1All1, process1Count5`), then the decision is to use a voice reminder

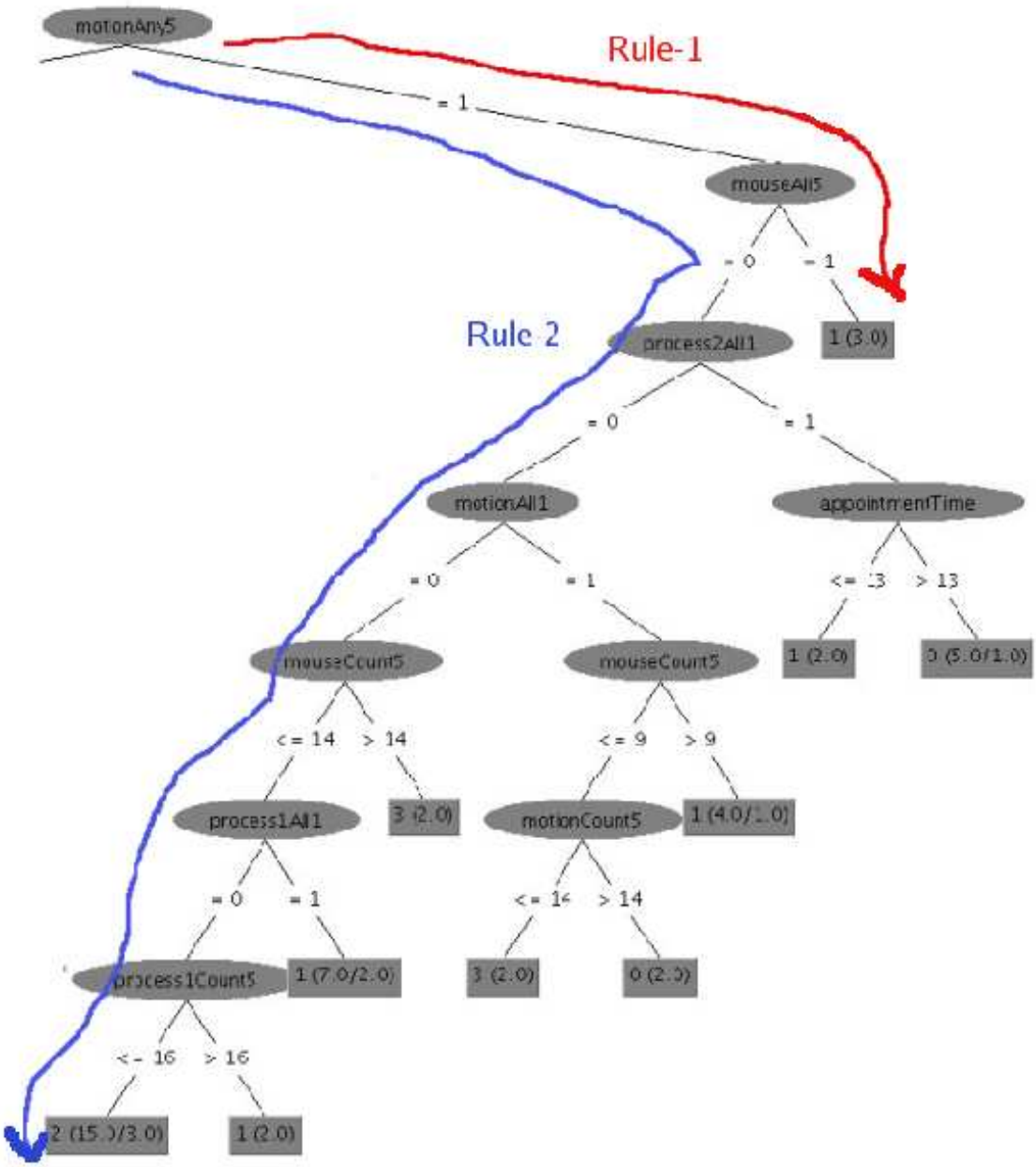

Figure 6.4 A partial snapshot of a complete Decision Tree learnt for *user-1*

(reminder type 1) for interrupting the user contingent upon the fact that the the mouse is also sparingly used (`mouseCount5` <= 14). In this case, actively writing code made this user prefer a voice reminder.

Similarly, examining Figure 6.5, which is a different snapshot of the same decision tree generated for *user-1*, shows the following interesting rules which Sycophant learns.

Rule-3 If the user is present and the appointment time is greater than 10:40 A.M. and the keyboard is sparingly used, then interrupt the user using both visual and voice reminders (reminder type 2). In this case, taking a coffee-break made the user prefer both types of reminders.

Rule-4 This rule is similar to the previous one in that the same set of user-context features are tested for generating a reminder, but this rule also checks to see if the keyboard has been used in the last fifteen seconds. Here, Sycophant uses a visual reminder to interrupt the user (reminder type 1). In this situation, not taking a coffee-break but doing work on his computer made the user prefer a visual reminder.

Interpreting these rules which Sycophant learns for *user-1* clearly indicates that user-context helps Sycophant to learn the appropriate reminder type.

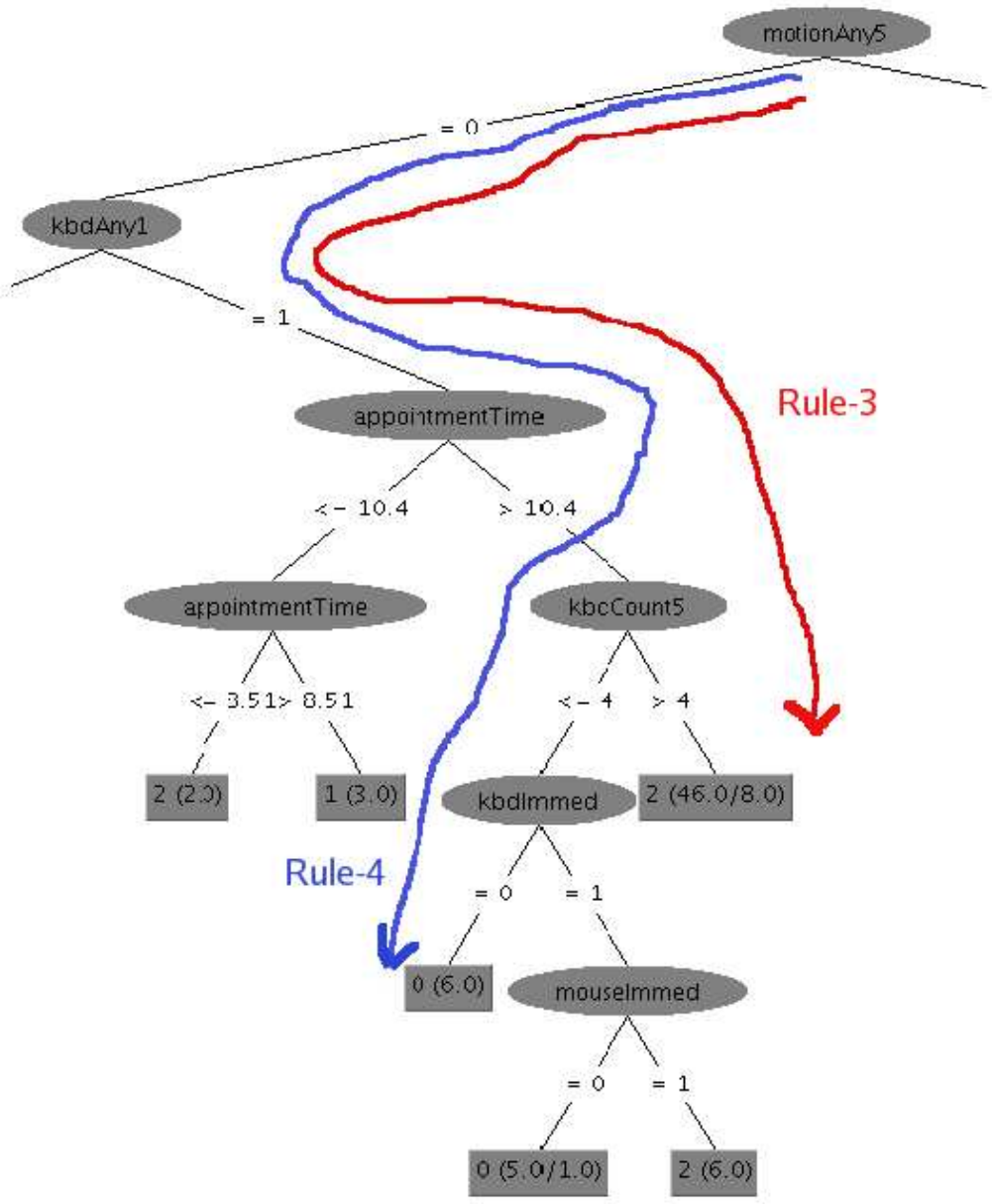

Figure 6.5 Another partial snapshot of a complete Decision Tree learnt for *user-1*

6.7 Summary

In this chapter, we showed that, for learning user preferences based on user-context, a number of machine learning algorithms achieve more or less the same level of accuracy as a decision-tree. This result validated our primary hypothesis that user-context helps Sycophant to learn a user's preference for reminder types. Next, we showed that removing external user-context degrades J48's performance on both the four-class and two-class reminder problems. This result emphasized that external user-context is beneficial to Sycophant on the reminder generation task. Our exploration of the decision tree generated for *user-1* again supported our primary hypothesis that user-context helps Sycophant to learn *user-1*'s preferences for reminder types. Our results in the next chapter show that XCS improves Sycophant's ability to learn user preferences.

Chapter 7
Results from using XCS to learn User-Context for a Single User

It is grindingly, creakingly, crashingly obvious that if Darwinism was really a theory of chance, it could not work.

— Richard Dawkins, *Climbing Mount Improbable*

7.1 Introduction

In the previous chapter we validated our approach for learning user-context by showing that Sycophant successfully learns to predict the type of reminder preferred by *user-1*. However, Sycophant's performance on the four-class reminder problem had a base accuracy of 67 percent. In this chapter we unleash XCS, a GBML technique, to improve Sycophant's performance on the four-class reminder problem.

7.2 GBML for a Single User

The initial promising results of our approach for learning user preferences encouraged us to improve Sycophant's performance on the four-class reminder problem using a Genetics Based Machine Learning (GBML) technique. Table 7.1 shows the performance of XCS when compared to J48 on *ContextData-1*. The first column is the data set used for evaluating the performance of a learning algorithm; the data set is either a training fold or a testing fold. We show the performance of J48 in column two. XCS's performance with and without condensation is shown in columns

Table 7.1 Comparison of the mean performance of XCS with J48 on the four-class reminder problem based on *ContextData-1*.

Data Set	Performance of J48	Performance of XCS	Performance of XCS_c	Inference	$Inference_c$
I	II	II	III	IV	V
Training-Set	0.812	0.940	0.937	Equal	Equal
Test-Set	0.700	0.940	1.000	Better	Better

three and four respectively. Our statistical inference comparing XCS's performance with that of 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 an average $19,386$ 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 $16,428$ when we operated XCS in the condensation mode.

We note from Table 7.1 that while XCS equals the performance of J48 on the training set, it significantly outperforms J48 on unseen examples in 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 user-related contextual features in condition space to reminder-types in the action space [45]. Sycophant's performance improved from 70 percent using J48 to 100 percent with XCS run in the condensation mode. XCS's increased performance is at the cost of

taking a longer time to *train* on the context data. XCS takes on an average 82 minutes to produce an optimal set of rules for *user-1* while J48 takes a minute to produce a decision tree for the same data set. XCS also has a few thousand rules for learning the mapping from contextual features to reminder types. This large number of classifiers suggests that we might have terminated condensation prematurely inside the classifier system. We plan to investigate this issue further.

7.3 Summary

In this chapter we improved Sycophant's test-set performance on the four-class problem from 67 percent (using J48) to 100 percent using XCS. Having successfully learnt reminder preferences for a single user, in the next chapter, we check if our user-context learning approach generalizes across three different users. Again we use J48 and XCS to learn user preferences and compare their performance on both the two-class and four-class problems.

Chapter 8
Results From Using XCS To Learn User Preferences For Multiple Users

8.1 Introduction

In this chapter we examine if our approach for learning user preferences generalizes across three different users. We find that XCS outperforms J48 and that external user-context significantly improves Sycophant's capability for learning user preferences. We discuss the generalizability of our approach in Section 8.2. Finally, we evaluate the effect of external context for learning reminder preferences.

8.2 Learning User Preferences for Three Different Users

Results from the previous chapter show that XCS helped Sycophant to achieve a better predictive accuracy on the four-class reminder problem in case of *ContextData-1*. In this section we evaluate the generalizability of our approach by measuring Sycophant's performance on user-context data collected from three different users.

We present the test-set performance of the learning schemes on the two-class and four-class reminder problems in Tables 8.1 and 8.2 respectively. In both tables, the first column represents the user-context data set, 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 8.1, we see that on the two-class reminder problem, both J48 and XCS significantly outperform Zero-R (our base-rate for learning reminder preferences) except in case of *ContextData-3*. Closer analysis of *user-3*'s data revealed that this

Table 8.1 Test set performance of different learning schemes across various user-context data sets on the two-class reminder problem.

Learning Algorithm →	Zero-R	J48	XCS	XCS$_c$, N	Inference	Inference$_c$
Data Set ↓						
I	II	III	IV	V	VI	VII
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

user *always* preferred to be reminded of appointments. This user also preferred a visual and voice reminder (reminder type 3) most of the time. In this case, user-context was not beneficial for capturing *user-3*'s preferences for reminder types. This explains the high accuracy achieved by Zero-R (which always predicts the majority

class) on this data set.

Table 8.1 shows that J48 and XCS are capable of achieving a minimum accuracy of 74 percent across all the user-context data sets. Statistical inference tests show that XCS equals the performance of J48 across all the user-context data sets; 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 with a high accuracy level across three different users.

Table 8.2 shows that, on the four-class reminder problem, XCS with and without condensation enabled outperforms J48 across three user-context data sets.

Table 8.2 Test set performance of different learning schemes across various user-context data sets on the four-class reminder problem.

Learning Algorithm →	Zero-R	J48	XCS	XCS_c, N	Inference	$Inference_c$
Data Set ↓						
I	II	III	IV	V	VI	VII
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

However, the performance degrades on the *ContextData-user-3*. We again attribute this drop in performance to *user-3*'s static preference for a single reminder

type. In the next section we evaluate Sycophant's performance in learning user preferences after removing external user-context from all the data sets.

8.3 Effect of External User-Context for Learning User Preferences for Multiple Users

Section 6.4 showed that external user-context helps Sycophant to better predict reminder type preferences for *user-1*. In this section we evalute the effect of external context on J48 and XCS for learning user-preferences across all three users in our study.

8.3.1 Effect of external user-context on J48's performance

Tables 8.3 and 8.4 show the effect of external contextual information on J48 for predicting the reminder types in case of the four-class and two-class reminder problems, respectively.

In both the tables, the first column is the name of user-context data set. The comparison of J48's performance for learning user-preferences for reminder types with external user-context is given in the second column. The third column shows the performance of J48 after external context is removed. We show the distribution of the four different types of reminder in *Context-Data-1* in the fourth column. The four numbers indicate the frequency of reminder types 0, 1, 2, and 3 respectively. The last column is our statistical inference checking how J48 performed on the reminder-class problems after external user-context was removed from a user-context data set.

Table 8.3 Effect of External Contextual Information on J48 for learning user-context on the Four-Class Reminder Problem.

Data Set	Performance with external user-context (percent)	Performance without external user-context (percent)	Distribution Of Reminders Types [0, 1, 2, 3]	Inference
I	II	III	IV	V
Context-Data-1	0.6743	0.6599	$197, 32, 111, 7$	Worse
Context-Data-2	0.7203	0.7118	$212, 45, 72, 25$	Worse
Context-Data-3	0.9914	0.9914	$0, 0, 3, 349$	Same
Context-Data All-Users	0.8081	0.7815	$409, 77, 186, 381$	Worse

Table 8.4 Effect of External Contextual Information on J48 for learning user-context on the Two-Class Reminder Problem.

Data Set	Performance with external user-context (percent)	Performance without external user-context (percent)	Distribution Of Reminders Types [0, 1]	Inference
I	II	III	IV	V
Context-Data-1	74.31	71.75	$197, 150$	Worse
Context-Data-2	81.07	77.68	$212, 142$	Worse
Context-Data-3	100.00	100.00	$0, 352$	Same
Context-Data All-Users	86.41	83.38	$409, 644$	Worse

In case of J48, Tables 8.3 and 8.4 show that external user-context is beneficial for learning user preferences. We see this effect of external user-context by the degradation in J48's performance on *Context-Data-1, Context-Data-2 and Context-Data-All-Users*. We find that external user-context does not seem to have any effect in case of *Context-Data-3*. After examining the distribution of reminders in *Context-Data-3*, we found that *user-3* had only one preference and opted to be reminded of an appointment in all except 3 out of the 352 context-data exemplars we had collected from this user. We believe that since *user-3* had only one static preference for a reminder type, contextual information was not very useful in learning his single preference.

8.3.2 Effect of external user-context on XCS's performance

Table 8.5 shows the effect of external contextual information on XCS (operated with condensation) for predicting the reminder types in case of the four-class and two-class reminder problems. The first column is the name of user-context data set. XCS's performance with external user-context on the four-class problem is shown in column two. XCS's performance on the same problem without external user-context is shown in column three. The fourth column is our statistical inference checking how XCS performed on this reminder-class problem after external user-context was removed from a user-context data set.

We show XCS's performance with external user-context on the two-class problem in column five. Column six shows XCS's performance on the same problem with-

Table 8.5 Effect of External Contextual Information for learning user-context on XCS's performance on both the four-class and two-class reminder problems.

	4-class problem			2-class problem		
Data Set	External User-Context			External User-Context		
	present	absent	Inference	present	absent	Inference
I	II	III	IV	V	VI	VII
Context-Data-1	1.0000	0.7745	Worse	0.7406	0.2254	Worse
Context-Data-2	1.0000	1.0000	Same	0.9300	0.8000	Worse
Context-Data-3	0.9118	0.9557	Better	1.0000	1.0000	Same
Context-Data All-Users	0.8236	0.5460	Worse	0.8788	0.4539	Worse

out external user-context. The seventh column is our statistical inference checking how XCS performed on this reminder-class problem after external user-context was removed from a user-context data set.

Removing external user-context significantly degrades XCS's performance on both the four-class and two-class reminder problems for *Context-Data-1*. XCS's performance degrades only on the two-class problem in case of *Context-Data-2*. On *Context-Data-3*, XCS's performance improves on the 4-class problem. External user-context improves XCS's performance on the two reminder class problems for *Context-Data-All-Users*.

From this table, we see that external user-context helps XCS to better learn user preferences in case of the two-class problem. On the four-class problem, external user-context is beneficial for XCS and XCS's improved performance on *Context-Data-All-Users* shows that external contextual information helps XCS to generalize learning

user preferences across all the three users.

User-3 had no varying preferences for reminder types, therefore contextual information was not very useful to XCS for learning his reminder preferences (Subsection 8.3.1). We also believe that this user's static preference for reminder types could be correlated with XCS's improved performance on the four-class problem after we removed external user-context.

8.4 Summary

The results in this chapter showed that our user-context based approach for learning user preferences generalizes across three different users in our study. XCS outperformed J48 on both the four-class and two-class reminder problems for these three users. Removing external user-context degraded J48's performance for learning user preferences for reminder types except in case of *user-3*. Our analysis of *user-3*'s data showed that this user had only one preference for reminder types. For *user-3*, external user-context had no beneficial effect for helping Sycophant learn to discriminate between different reminder types. Similar results in case of XCS showed the usefulness of external user-context in learning user-preferences except for *user-3*. In the next two chapters we summarize the main contributions of this thesis and explore promising directions for future research in the area of context-aware applications.

Chapter 9
Discussion and Conclusion

One can acquire certainty only by amputating inquiry.

— Marvin Minsky, *Society of Mind*

The primary objective of this thesis was to demonstrate that incorporating simple user-context helps software applications to better learn user-preferences. We summarize the main contributions of this thesis in the next few paragraphs.

We traced the lack of personalization of current applications to the meager use of contextual information and hypothesized that simple user-context can enable applications to better learn user preferences in chapter 1. To remedy this dependence of applications on inadequate context, we introduced a sensor-based technique for gathering simple user-contextual information from both the internal and external environments of a computer in chapter 3. In this chapter, we described an approach to gather user-context and create user-context data sets to learn user-preferences. Next, we designed *Sycophant*, our context-aware calendaring application capable of generating a user-preferred reminder to substantiate our hypothesis.

In chapter 4, we investigated machine learning techniques to enable Sycophant learn user preferences for reminder types. Sycophant's prediction technique involved mapping user-related contextual features to reminder types. We investigated two machine learning schemes: a Genetics Based Machine Learning technique, XCS, and

J48, a decision-tree learning algorithm in chapter 4 for learning this complex non-linear function.

We delineated our experimental methodology for gathering user-context data in chapter 5. Our results in chapter 6 showed for *user-1*, Sycophant successfully learnt to predict his preferred reminder-type. Sycophant had a predicitive accuracy of 67 percent on the *four-class problem* of choosing a reminder from a set of four reminder types; Sycophant achieved an accuracy of 82 percent on the *two-class problem* of deciding whether or not to generate a reminder for a user. Our results on the two-class problem were comparable to Fogarty's results on predicting the interruptability of a user [24, 20]. The degradation of J48's performance on the two-classes of reminder problems indicated that external user-context proved to be beneficial for learning *user-1*'s reminder preferences. We interpreted four rules generated by a decision-tree and showed that Sycophant learns human understandable rules for deciding the type of reminder to generate.

XCS, a GBML technique gave us the most promising results for learning *user-1*'s preferences for reminder types in chapter 7. Our results in this chapter showed that XCS significantly outperformed J48 in learning user preferences. Sycophant used XCS and improved its accuracy from 67 percent (using J48) to 100 (using XCS) percent on predicting the correct reminder-type to generate for *user-1* on the four-class reminder problem.

We evaluated the generalizability of our user-context based approach to learn user preferences for three users in chapter 8. Both XCS and J48 had a base accuracy of 74 percent on the two-class problem; J48 had a base accuracy of 70 percent, and XCS had a base accuracy of 82 percent on the four-class problem. Our results in this chapter showed that, in case of users who had different preferences for reminder types, removing external user-context significantly degraded J48's and XCS's performance on both the four-class and two-class reminder problems. This showed that for such users, external user-context is beneficial for an application to learn user preferences. These results indicated that an application has to sense both the external and internal user-related environments to better learn user preferences.

XCS and J48's predictive accuracy for learning user preferences on unseen cases, that is, these algorithms test-set performance across different users, indicates two promising avenues for user-context learning: the promise of employing machine learning techniques for learning user preferences and the potential for enhancing human-computer interaction by using simple contextual information.

Chapter 10
Future Work

Prediction is very difficult, especially of the future

— Neils Bohr

This thesis was mainly concerned in bridging the gap in interaction between computer applications and users by exploiting contextual information from a user's environment. Encouraged by Sycophant's promising results for learning user preferences using internal and external user-contextual information, we wish to enable other software applications to become more user-context aware by using our simple sensors-based context learning approach.

We have promising results for learning user preferences across three different users. For evaluating Sycophant's ability to learn user preferences and to test the generalizability of our approach, we plan to gather data from a diverse set of users. Currently, Sycophant uses machine generated rules for learning user preferences. In the future, we wish to combine expert-generated rules with machine-learnt rules and use this combined knowledge to design better adaptive user interfaces. There is always 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 preferences. This work will hopefully contribute to building applications that utilize user-context to behave more socially and interact more naturally with users.

References

1. David Beymer. Person counting using stereo. In *Workshop on Human Motion*, pages 127–, 2000.

2. Aphrodite Galata, Neil Johnson, and David Hogg. Learning variable length markov models of behaviour, 2001.

3. M. Witbrock and A. Hauptmann. Speech recognition and information retrieval, 1997.

4. Elizabeth D. Mynatt and Gerhard Weber. Nonvisual presentation of graphical user interfaces: Contrasting two approaches. pages 166–172.

5. Pat Langley. Machine learning for adaptive user interfaces. In *KI '97: Proceedings of the 21st Annual German Conference on Artificial Intelligence*, pages 53–62, London, UK, 1997. Springer-Verlag.

6. Cynthia A. Thompson, Mehmet H. Göker, and Pat Langley. A personalized system for conversational recommendations. *J. Artif. Intell. Res. (JAIR)*, 21:393–428, 2004.

7. Pat Langley. User modeling in adaptive interfaces. In *UM '99: Proceedings of the seventh international conference on User modeling*, pages 357–370, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.

8. P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, October 1997.

9. P. J. Brown. The stick-e document: a framework for creating context-aware applications. In *Proceedings of EP'96, Palo Alto*, pages 259–272. also published in it EP–odd, January 1996.

10. Anind K. Dey, Gregory D. Abowd, Mike Pinkerton, and Andrew Wood. Cyberdesk: A framework for providing self-integrating ubiquitous software services. In *ACM Symposium on User Interface Software and Technology*, pages 75–76, 1997.

11. D. Franklin and J. Flachsbart. All gadget and no representation makes jack a dull environment. In *Proceedings of AAAI 1998 Spring Symposium on Intelligent Environments*, 1998. AAAI TR SS-98-02.

12. Richard Hull, Philip Neaves, and James Bedford-Roberts. Towards situated computing. In *ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers*, page 146, Washington, DC, USA, 1997. IEEE Computer Society.

13. Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *ISWC*, pages 92–99, 1998.

14. Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

15. T. Rodden, K. Cheverst, N. Davies, and A. Dix. Exploiting context in hci design for mobile systems, 1998.

16. Jason Pascoe, Nick Ryan, and David Morse. Using while moving: Hci issues in fieldwork environments. *ACM Trans. Comput.-Hum. Interact.*, 7(3):417–437, 2000.

17. Bill Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.

18. A. Ward, A. Jones, and A. Hopper. A new location technique for the active office, 1997.

19. Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware appplications. *Human Computer Interaction*, 16, 2001.

20. Sushil J. Louis and Anil 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.

21. Anil Shankar and Sushil J. Louis. Learning classifier systems for user context learning. In *2005 IEEE Congress on Evolutionary Computation,September 2-5 2005, Edinburgh, UK*, 2005.

22. Anil Shankar and Sushil J. Louis. Better personalization using learning classifier systems. In *Proceedings of the 2005 Indian International Conference on Artificial Intelligence, December 20-22 2005, Poona, INDIA*, 2005.

23. 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.

24. James Fogarty, Scott E. Hudson, and Jennifer 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.

25. Piotr D. Adamczyk and Brian 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.

26. A. Kulkarni. A reactive behavioral system for the intelligent room. *Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2002.*, 2002.

27. Rodney A. Brooks. *Cambrian intelligence: the early history of the new AI.* MIT Press, Cambridge, MA, USA, 1999.

28. Motion. http://motion.sourceforge.net/.

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

30. Alan Black, Paul Taylor, and Richard Caley. The festival speech synthesis system. 1998.

31. Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

32. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

33. Sushil John Louis. *Genetic algorithms as a computational tool for design.* PhD thesis, Bloomington, IN, USA, 1993.

34. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning, Chapter 6, Introduction to Genetics-Based Machine Learning, pp 217-259.* Addison-Wesley, Reading, MA, 1989.

35. Stewart W. Wilson. Classifier systems and the animat problem. *Mach. Learn.*, 2(3):199–228, 1987.

36. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.

37. 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.

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

39. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

40. Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, USA, 2000.

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

42. Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

43. Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Mach. Learn.*, 11(1):63–90, 1993.

44. J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Technical report,Department of Statistics, Stanford University*, 1998.

45. Tim Kovacs. *XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions.*, pages 59–68. Springer-Verlag, August 1997.