J. Expt. Theor. Artif. Intell.
Vol. 16, No. 3, July–September, 2004, 161–188

Taylor & Francis
Taylor & Francis Group

Applying machine learning to programming by demonstration

GORDON W. PAYNTER and IAN H. WITTEN

University of California, Riverside and University of Waikato, New Zealand
e-mail: gordon.paynter@library.ucr.edu; ihw@cs.waikato.ac.nz

Abstract. 'Familiar' is a tool that helps end-users automate iterative tasks in their applications by showing examples of what they want to do. It observes the user's actions, predicts what they will do next, and then offers to complete their task. Familiar learns in two ways. First, it creates a model, based on data gathered from training tasks, that selects the best prediction from among several candidates. Experiments show that decision trees outperform heuristic methods, and can be further improved by incrementally updating the classifier at task time. Second, it uses decision stumps inferred from analogous examples in the event trace to predict the parameters of conditional rules. Because data is sparse—for most users balk at giving more than a few training examples—permutation tests are used to calculate the statistical significance of each stump, successfully eliminating bias towards attributes with many different values.

Keywords: programming by demonstration, adaptive user interfaces, human–computer interaction, machine learning, iterative tasks

Received October 2002; accepted July 2004

1. Introduction

Many repetitive tasks in direct-manipulation interfaces cannot be automated with standard application tools, forcing users to perform the same interface actions again and again. Experienced programmers might write a script to carry out such tasks on their behalf, but this lies beyond the ability of most end-users. Programming by demonstration (PBD) can solve these problems by letting users automate repetitive tasks without requiring programming knowledge: the user need only know how to perform the task in the usual way to be able to communicate it to the computer (Cypher 1993a).

'Familiar' is a PBD system that helps users automate iterative tasks within existing applications on a popular computing platform. It is able to work with completely new applications never before encountered, achieving a far greater degree of application independence than other systems. Moreover, it operates at a high level, gleaning abstract knowledge from each application and exploiting it to make predictions. The inference engine tolerates noise, incorporates feedback and generates explanations of its predictions.

Journal of Experimental & Theoretical Artificial Intelligence ISSN 0952–813X print/ISSN 1362–3079 online

DOI: 10.1080/09528130412331290520

The system is an 'adaptive interface': it learns specialized tasks for individual users. It is also adaptive in a deeper sense: it can adapt to an individual user's style. Suppose two people are teaching a task and happen to give identical demonstrations. The vast majority of demonstrational interfaces would necessarily infer identical programs—they learn from each user without adapting their behaviour to that user. Some, however, including learning apprentices (Mitchell *et al*. 1994) and Familiar, can adapt their learning to a specific user based on that user's interaction history.

Familiar employs standard machine learning (ML) techniques to infer the user's intent—unlike most PBD tools, which rely on *ad hoc* heuristics that incorporate prior knowledge of applications. ML has two roles. First, rules are used to evaluate predictions so that the 'best' one can be presented to the user at each point. Second, conditional rules are inferred so that users can teach classification tasks like sorting files based on their type and assigning labels based on their size.

This paper describes how Familiar uses ML methods to learn from, and adapt to, the user. Familiar works on the Macintosh and is based on Apple's scripting language (whether these are good choices in practice is outside the scope of this paper). Its only knowledge of each application is gleaned from the application itself, using the scripting language. The success of the ML methods is evaluated by analysing their effect on actual traces of users performing test tasks. This off-line evaluation methodology was chosen because it provides a more controlled environment for experimentation and refinement than on-line operation, in which users may vary their actions as a result of Familiar's suggestions. On-line operation raises many interesting questions about user interaction with learning systems, but they lie beyond the scope of this paper.

Our work highlights both the advantages and the difficulties of using ML in adaptive user interfaces. We first establish that PBD can be provided in an application-independent manner for widely-used commercial application programs. We show that replacing *ad hoc* heuristics with ML techniques not only leads to more rational designs, but also improves end-user performance. We show that incremental learning allows a PBD agent to adapt to individual users and further improve performance. Difficulties arise because existing application programs provide limited access to application and user data, and because interactive systems are most realistically evaluated through costly on-line usability experiments. The application of ML in adaptive interfaces is complicated by the difficulty of iterative system development: a change in one prediction may alter the user's strategy.

The structure of the paper is as follows. Section 2 describes the user interface by giving examples of various tasks. Section 3 explains the underlying architecture. Sections 4 and 5 describe how ML algorithms guide and make predictions. Section 6 evaluates learning performance, while section 7 discusses the lessons learned from this experience, and also includes a brief review of related research. A full account of Familiar appears in Paynter (2000).

2. The Familiar experience

Familiar's design follows four broad guidelines: the use of existing applications, simplicity, minimizing user effort, and educating the user. It *uses existing applications* because these are what end-users know. Many PBD systems are

built into prototype applications, and force users—who are often novices—to learn the application at the same time as the new programming paradigm. Familiar is application-independent in an immediate and practical sense: it works with new, unseen applications as soon as they are installed on the computer. The interface is also characterized by its *simplicity*. Natural-language descriptions are generated to explain reasoning and intentions (though the evaluation of these lies beyond this paper's scope). The agent metaphor is eschewed (though Familiar has agent functionality; see Erickson 1997). Familiar attempts to *minimize user effort* by inferring as much of the task as it can; consequently, it requires robust, accurate inferencing. Every Familiar instruction presented to the user is a command in an established scripting language, unobtrusively *educating the user* in the syntax.

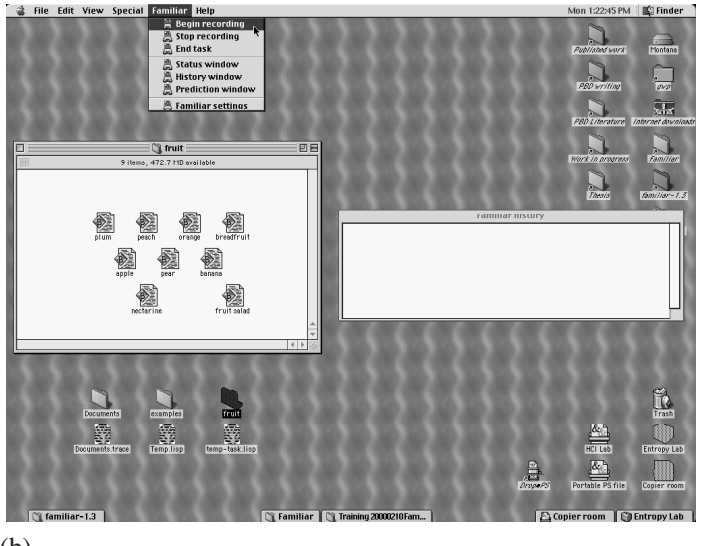
This section illustrates Familiar's use through a selection of iterative tasks. The first demonstrates the ability to iterate over sets and extrapolate sequences. In a variation of this task, the user asks Familiar to explain predictions and gives feedback about their accuracy. In the second example Familiar must infer an implicitly defined sort criterion. The third demonstrates the ability to work across multiple domains and infer long cycles from noisy demonstrations. These tasks are chosen for pedagogical purposes to help explain the system's capabilities: the actual tasks used for evaluation are described in the Appendix.

In each case the user first asks the agent to start observing their actions by selecting *Begin Recording* from the *Familiar* menu (figure 1a), which is available in every application, and proceeds to demonstrate the task by performing it.

2.1. *Arranging files*

In this task, files are to be arranged into a horizontal row in their folder window. Figure 1b shows the screen when *Begin recording* is selected. The files are in the active window, the *fruit* folder. The user begins by moving a convenient one, *plum*, to the top left of this folder (figure 2a). These actions are recorded and displayed in the

(a)



(b)

Figure 1. (a) The Familiar menu; (b) the screen before the *arranging files* task.

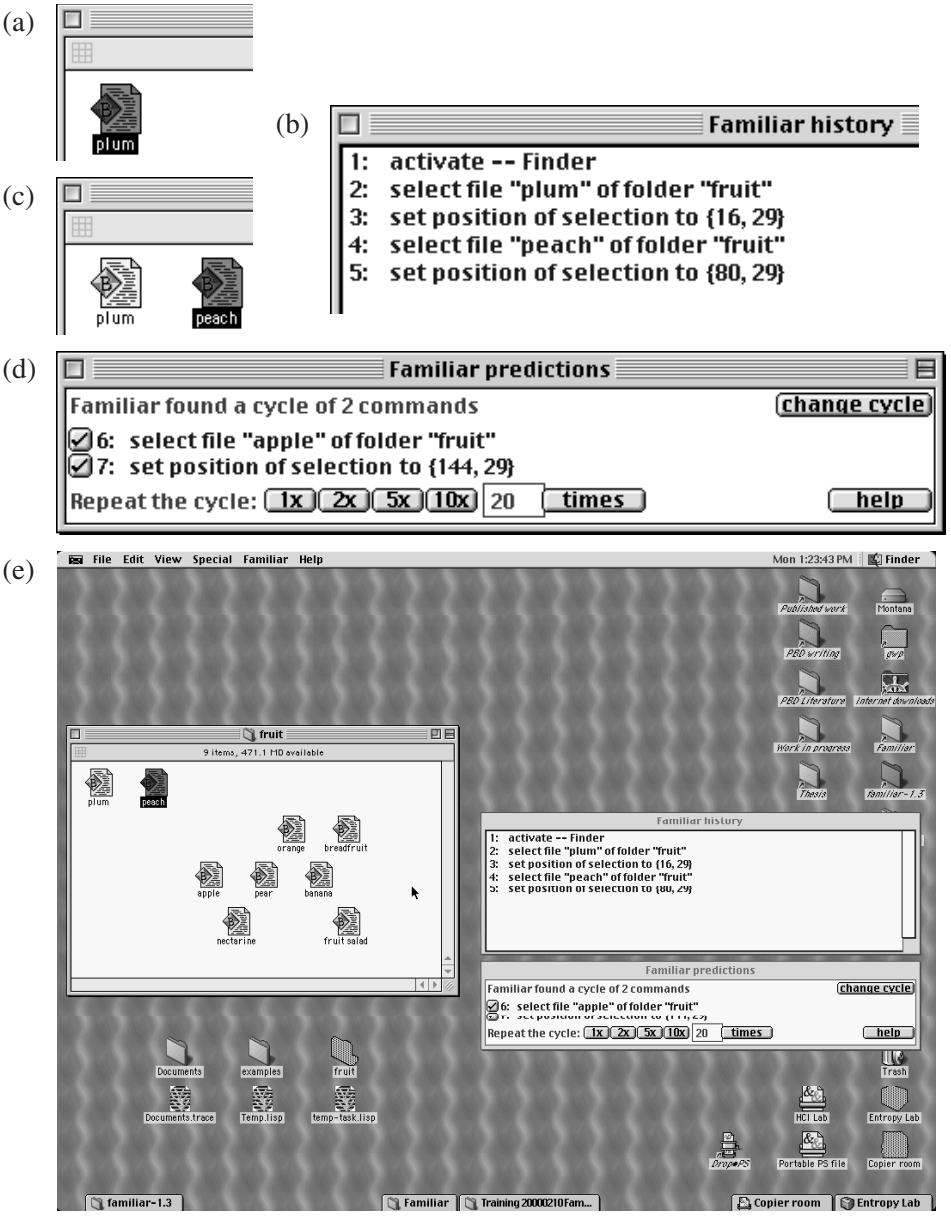


Figure 2. The *arranging files* task. The two demonstrations (a, c) are recorded (b); Familiar correctly predicts the next iteration (d); screen at this point (e).

history window (figure 2b). The *activate* command (event 1) indicates that the user is working in the Finder (the Macintosh operating system). The *select* and *set* commands (events 2 and 3) describe the positioning of the first file. The user continues by moving file *peach* (figure 2c); again the actions are displayed (figure 2b, events 4–5).

Each time it records an action, Familiar attempts to generalize the event trace and infer the user's intent. After event 5 it detects a cycle and suggests, in the *predictions*

window (figure 2d), that the next actions will *select file* '*apple*' (event 6) and *set* its position (event 7). Figure 2e shows the screen at this point.

From the *predictions* window, the user can press the *one time* (1×), *two times*, *five times* or *ten times* buttons to execute complete iterations of the cycle. In this example, they are satisfied with the predictions, and press the *one time* button (figure 3a). Familiar responds by sending the commands to the Finder, which selects and positions *apple*. As each command is executed it is added to the *history* window and its colour is changed in the *prediction* window. Afterwards, the prediction for the next iteration is displayed (figure 3b). Knowing how many files are left, the user types 6 (replacing the default of 20 visible in figure 3a), presses *times*, and watches as each file is moved into place (figure 3c).

2.2. *Dealing with errors*

The simplest way to correct a mistake is to demonstrate another example using the standard application interface. The new demonstration—and the fact that the old predictions were incorrect—will be incorporated into subsequent predictions. Alternatively, the *change cycle* button rejects the current iterative pattern and displays another. Suppose that after the first two examples of the task (figure 2a–c) Familiar incorrectly predicted a cycle of three events, *activate*, *set*, and *select* (figure 4a). *Change cycle* replaces this by another prediction (figure 4b).

The *help* button explains how predictions are made (figure 5). In figure 5a the *select* command's parameter (event 6) is incorrect: Familiar predicts *peach* but this file has already been moved. Its assumption is that the same file is selected in each iteration (figure 5b). Option-clicking the prediction replaces *peach* by *apple* (figure 5c), on the assumption that the iteration is over all *file* objects in folder *fruit*.

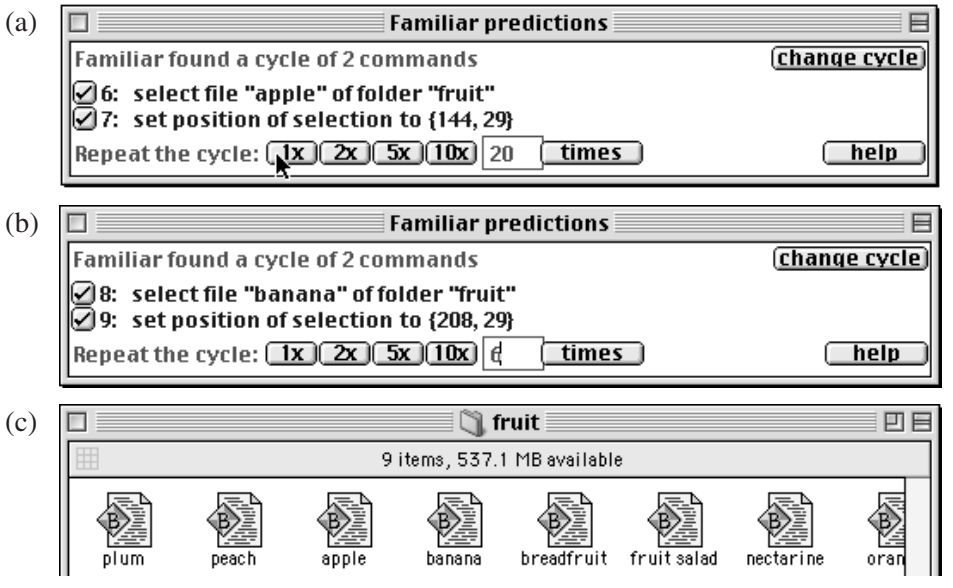


Figure 3. Completing the *arranging files* task. The user presses the *one time* button (a) then requests six more iterations (b); screen at this point (c).

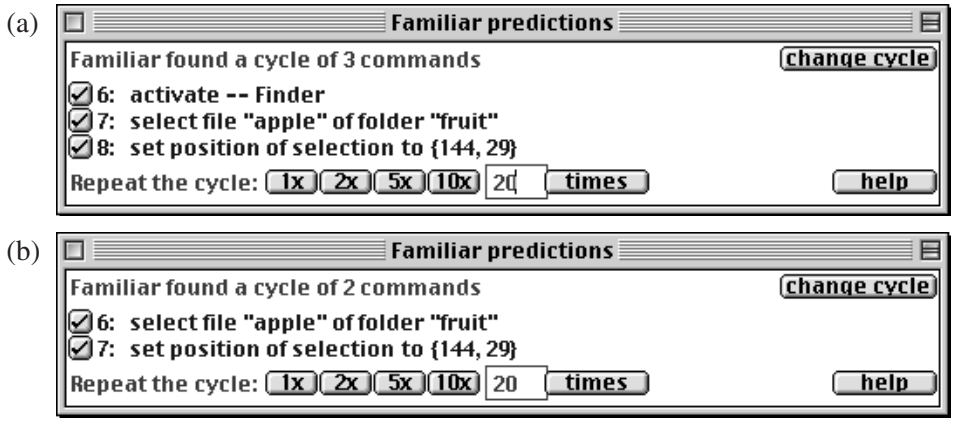


Figure 4. Changing an incorrect cycle. Incorrectly predicted cycle (a); result of pressing *change cycle* (b).

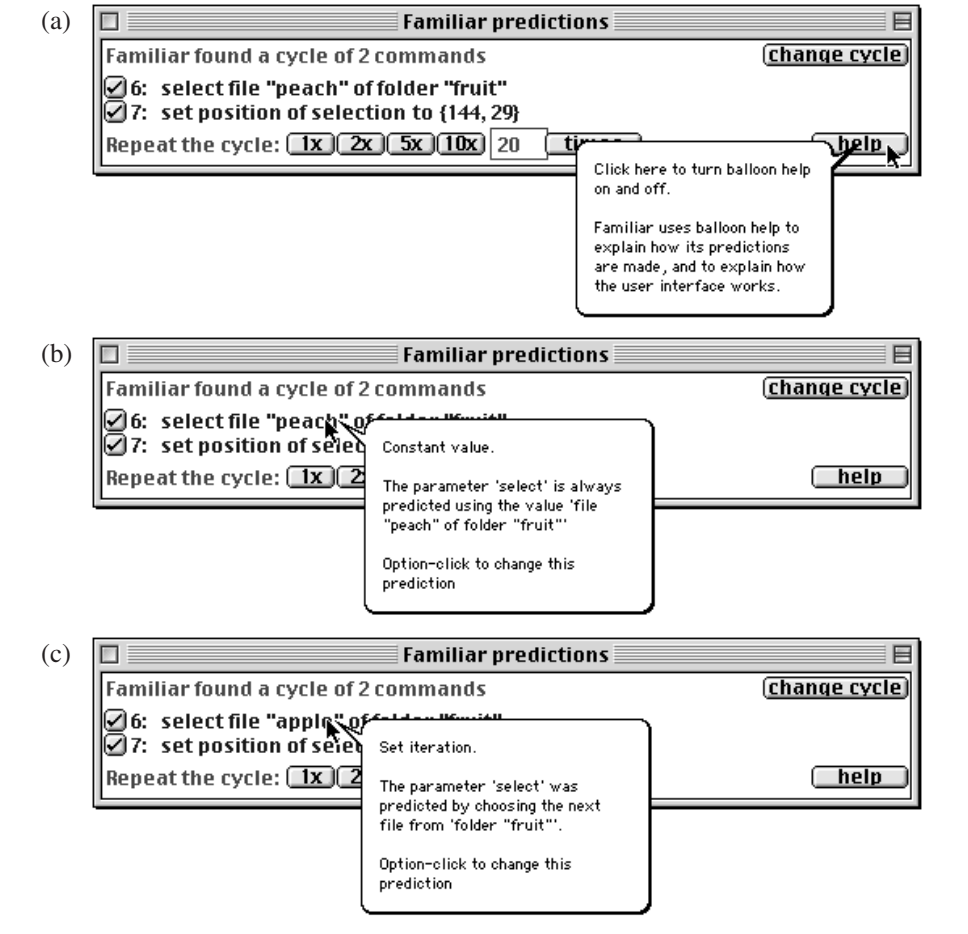


Figure 5. Examining and changing an incorrect prediction. Turning on help (a); explaining a particular command parameter (b); result of option-clicking that prediction (c).

2.3. *Sorting files*

The second task sorts files into folders for word-processor and spreadsheet documents. The user starts by creating a new folder and naming it *word processor*. Then *ACC01.doc* (figure 6a, events 5–6) and *ACC99.doc* (figure 6b, events 7–8), both word-processing documents, are moved into it. The third file, *Balance sheet*, is a spreadsheet and the user creates a folder (figure 6c, events 9–11) and moves it (figure 6c, events 12–13). So far over half the recorded events are initialization commands that do not contribute to the iteration.

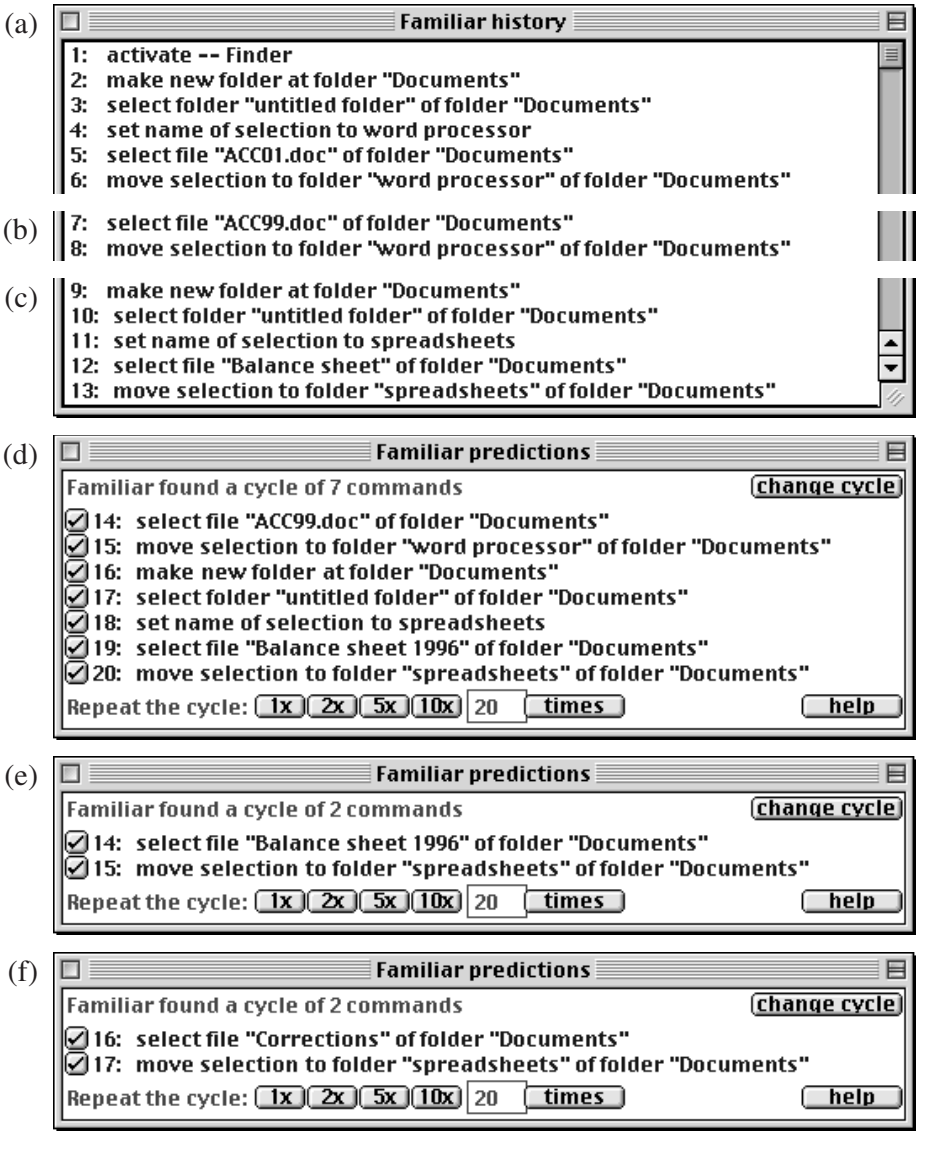


Figure 6. Changing an incorrect cycle. Events in the history window after processing two word-processor files (a, b) and a spreadsheet file (c); incorrect prediction at this point (d); correct prediction after pressing *change cycle* (e); still-correct prediction after pressing *one time* (f).

Familiar detects a pattern of seven events and makes a corresponding prediction (figure 6d). Unfortunately it is completely wrong. When the user presses *change cycle*, Familiar suggests a two-step pattern that works for the next file (figure 6e); the user presses *one time* and watches Familiar executing the commands to move file *Balance Sheet 1996* to the *spreadsheets* folder.

For the next iteration (figure 6f) Familiar proposes to select *Corrections*, a word-processor file, and moves it to the wrong folder, *spreadsheets* (figure 6f, event 17). Noticing the error, the user clicks *help*, moves the mouse over the incorrect prediction and learns (figure 7a) that the prediction is the constant *spreadsheets* (the value in the last two iterations), which the user can change by giving a new example. Familiar has no other suggestions to make: three examples are insufficient to teach this classification task.

Returning to the Finder, the user moves *Corrections* into the *word processor* folder. These actions are recorded (figure 7b) and used to predict the next file (figure 7c). Unfortunately, the prediction is incomplete: the agent correctly anticipates that the user will select *expenses for 1996*, but fails to predict the destination folder. The explanation (figure 7c) is that a relationship has been found between the *to* parameter and the *kind* attribute of the previous event, and Familiar can only predict event 19 after seeing event 18.[1] To proceed, the user clicks the *tick* button beside event 18: Familiar executes that event (figure 7d), adds it to the history window, and displays its prediction of the next two events (figure 7e), which are now correct.

Confident that Familiar has grasped the idea, the user types 1000 into the number of iterations field and presses *times* (figure 7e). After 135 iterations no files are left in the folder. Since Familiar can neither predict nor select the next file, it stops performing the task.

2.4. *Converting images*

Complex tasks may involve multiple domains, longer demonstrations, and noise. Figure 8 shows a user changing image files from PICT format to JPEG with an image conversion program. The task requires two applications and has a noisy event trace. Familiar's history window is shown after the first (figure 8a) and second (figure 8b) iterations have been demonstrated. The first three events initialize the environment. Event 15 is a singular noise event generated when the user shifts a window to get a better view. In figure 8c, Familiar has correctly identified a cycle of six significant events and predicted the next full iteration.

3. The Familiar architecture

Familiar learns tasks that change from one iteration to the next by generalizing and instantiating data descriptions. Its inferencing works on two levels. First, iterative patterns are sought in the *types* of events that have been demonstrated. Two sequence recognition schemes search for candidate patterns and make suggestions. The 'best' pattern is selected: this produces a predicted cycle of AppleScript commands. Second, *parameters* are determined for each of these command types. For every parameter appearing in the pattern, five pattern analysis schemes are asked to make candidate predictions, and the 'best' prediction is selected. The resulting cycle of parameterized commands is presented in the predictions window (figures 3d, 5a–b, 6, 7, 8d–f, 9, 10c).

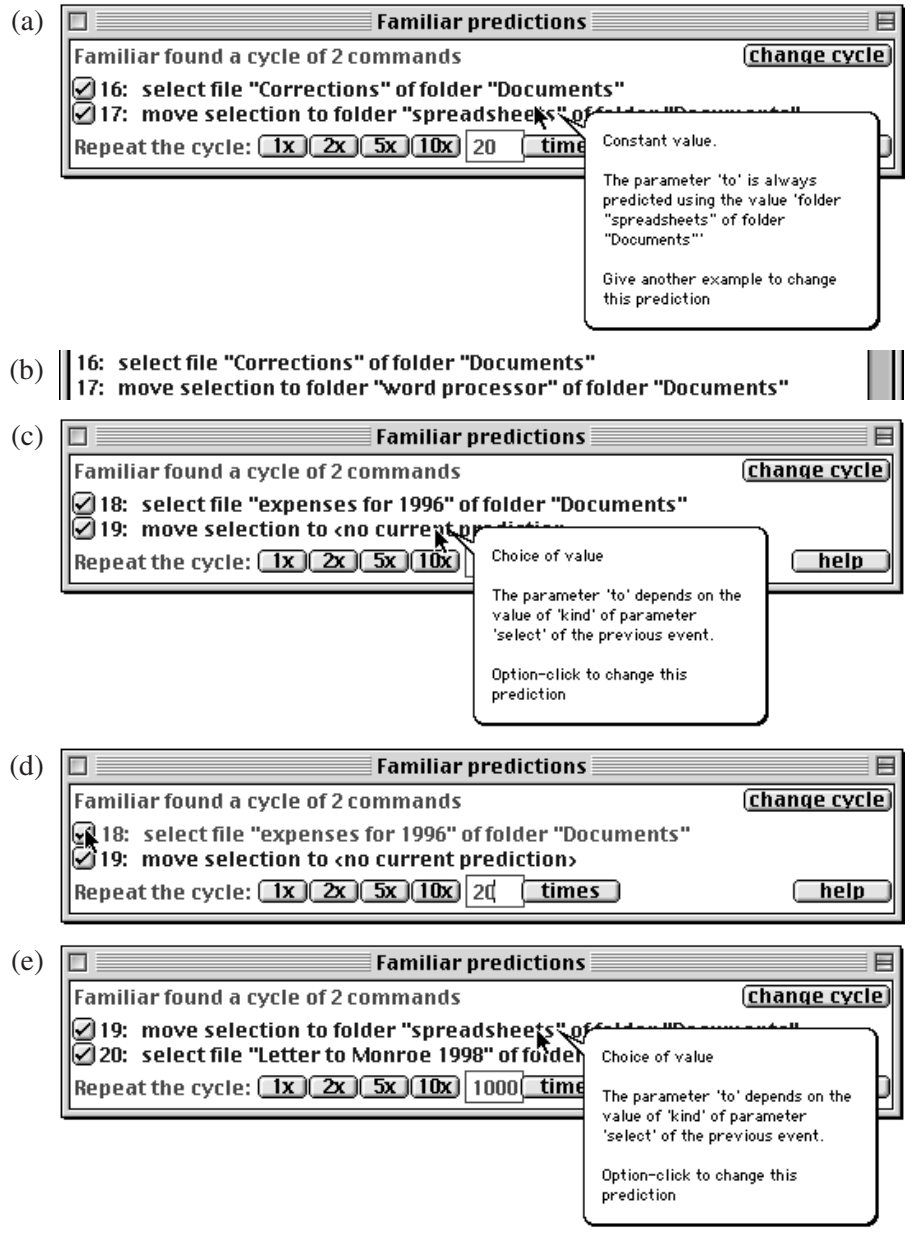


Figure 7. Changing an incorrect parameter. Familiar's explanation of its parameter choice (a); new choice, also incorrect, after a second demonstration (b); executing event 18 alone by clicking it (d); correct prediction at last (e).

In each case, competing algorithms make predictions. ML techniques are used to determine the 'best' sequence of command types (section 5.1) and the 'best' parameters for each (section 5.2). These decisions are crucial because any agent is useful only if it can be trusted to work autonomously. Correct predictions build a program that can immediately complete the user's task. Incorrect ones at best cost

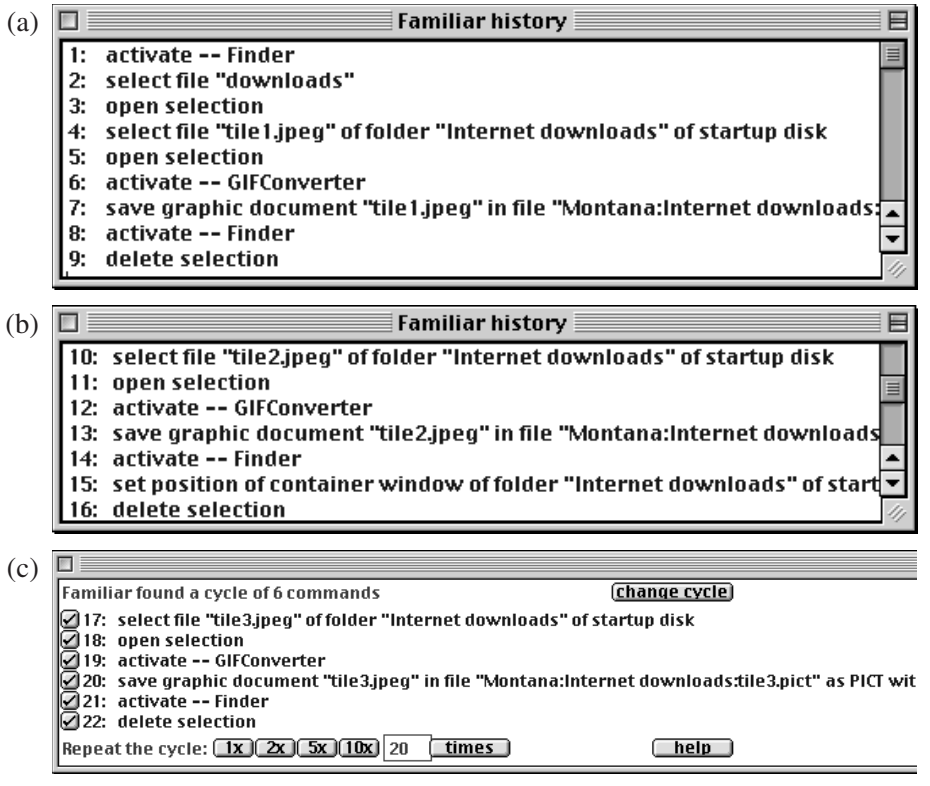


Figure 8. Converting image files from PICT format to JPEG. History window after demonstrating the first (a) and second (b) iterations; correct cycle of six events, with correct prediction (c).

the user time, and at worst issue the wrong commands, damaging the user's data and their confidence in the agent.

Figure 9 depicts Familiar's architecture in terms of major system components (circles), user interface components (rectangles) and the external environment (topmost oval). Application actions performed by the user are intercepted by the event recorder, causing data to flow along the solid grey arrows to the prediction window, which in turn sends new commands to this or other applications. The event recorder monitors the user (section 3.1): it parses each action and notifies the sequence recognition manager whenever a new event occurs. The sequence recognition manager detects patterns in the event trace (section 3.2). It uses two sequence recognition schemes to identify cycles, chooses the best and sends it on as a sequence of predicted command types. The pattern analysis manager binds these commands' parameters to generate a specific prediction (section 3.3).

The prediction window displays the predicted command cycle and lets the user send commands to application programs. Commands that are executed are also received by the event recorder, which sends an event notification to the sequence recognition manager, which updates the current pattern and sends it to the pattern analysis manager, which recalculated the parameters and passes the cycle to the prediction window, where the display is updated appropriately.

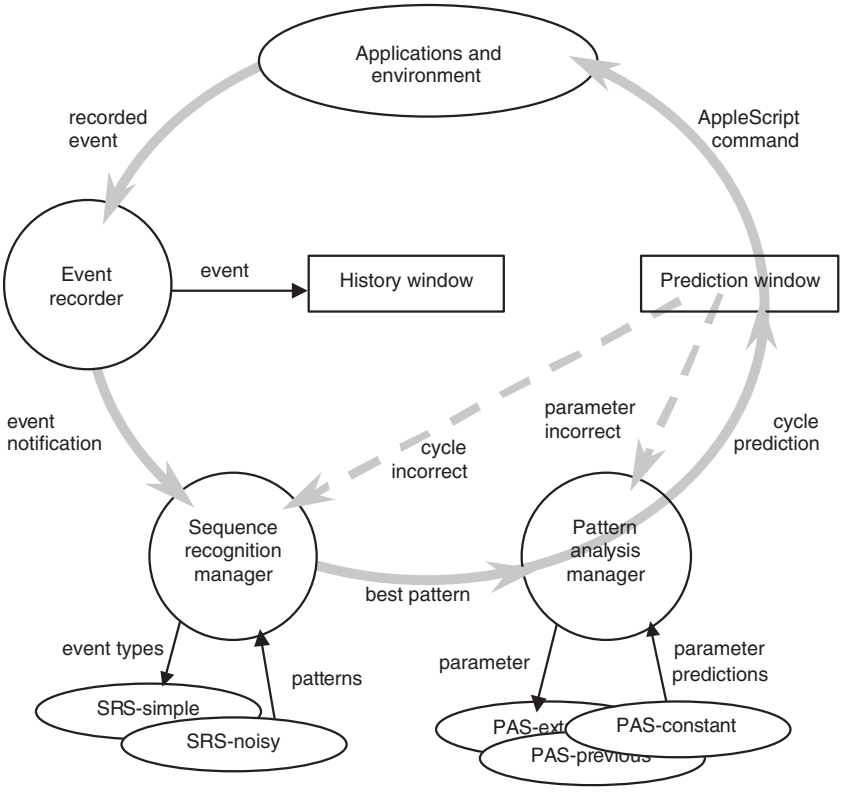


Figure 9. The Familiar architecture.

The user can give explicit feedback through the prediction window, or implicit feedback by ignoring it. In the latter case, new commands are recorded by the event recorder and processed as described above. The user can explicitly reject parameter and sequence predictions (represented by dashed grey lines in figure 9). Incorrect parameter predictions reinvoke the pattern analysis manager, which finds the next best prediction and passes it to the prediction window. Incorrect cycle predictions reinvoke the sequence recognition manager, which finds the next best pattern and sends it to the pattern analysis manager to have its parameters bound.

3.1. *The event recorder*

The event recorder, activated when the user selects *Begin Recording*, monitors the user and manages the information required to make predictions. Knowledge of AppleScript is intrinsic to both functions.

3.1.1. *AppleScript*

Macintosh applications are called *scriptable* if scripts and other programs using AppleScript commands can invoke them. Some (but not all) scriptable applications are also *recordable*: they can report the user's actions as they are made. When *Begin recording* is selected (figure 1), Familiar activates recording and all subsequent user actions are reported.

1	Tell application 'Microsoft Excel'
2	Activate
3	Select Range 'R1C2'
4	Set FormulaR1C1 of ActiveCell to 'July'
5	Select Range 'R1C3'
6	Set FormulaR1C1 of Active Cell to 'August'
7	Select Range 'R1C4'

Figure 10. AppleScript recorded in Microsoft Excel.

Figure 10 shows the commands recorded in a particular demonstration. The *tell* command (line 1) identifies the application that has reported the command; in this case it is named 'Microsoft Excel'. Familiar does not display this command to the user; instead, the application name is appended to the command that follows—invariably *activate* (line 2)—resulting in expressions like *activate - - Microsoft Excel* in the *history* window. In AppleScript syntax, any text following the token '- -' is comment, so *activate - - Microsoft Excel* and *activate* are equivalent.

Familiar requires knowledge of AppleScript to parse the event trace. The language has a core functionality (including the *tell* command) that Familiar is given as background knowledge, but every scriptable application extends it by adding publicly accessible representations of its own objects and commands, which appear frequently in the event trace. Familiar maintains an internal model of each application's commands and objects. The *tell* in figure 10, for example, is Familiar's first encounter with Excel. It immediately constructs a model of the application known as 'Microsoft Excel', which is used to parse the actions reported by the application.

3.1.2. *Application knowledge*

Familiar's application knowledge falls into two categories: roughly speaking, *class* and *instance* information. The former describes the application's capabilities, including the commands it supports and the classes it uses, and persists from one invocation of the program to the next. The latter describes the application's current state, including the event trace and the data being worked on, and changes constantly in response to user actions.

Familiar's source of class information is the 'application dictionary', a list provided by the application that describes all the commands, objects and enumerations it uses. The Finder dictionary, for example, contains 26 commands (including *select*, *open* and *delete*), 41 classes (*file*, *folder* and *disk*) and seven enumerations (used internally). Commands have typed parameters, and classes have containee classes and properties (i.e. attributes). Familiar gathers additional class information by inference from the event trace. For example, it displays the order of command parameters in the same order they are used in recorded events on the grounds that the designer has chosen this as the best order for communicating with the user.

AppleScript recording is the source of instance information. As interaction proceeds, Familiar builds an event trace of the commands and objects encountered, augmented by contextual information. It sends AppleScript commands to applications requesting three types of context: the properties of objects, the evaluation of properties, and the contents of container objects. These requests, constructed

from command templates, can be sent to any scriptable application, thus preserving application independence.

Familiar maintains knowledge of each application independently of the others, and its inferencing is based only on this information and the background information. Unlike other PBD systems, it has no prior knowledge of applications; it simply rebuilds its models as each application name is observed in the event trace. This level of application independence has an immediate, practical benefit: if the user installs a completely new scriptable application that Familiar has never before encountered, issues the *Begin Recording* command, and starts interacting with the application, PBD will be available right away.

3.2. *Sequence recognition*

The sequence recognition manager detects iterative patterns of command types in the event trace, without reference to their parameter values.

3.2.1. *Detecting patterns*

After each user action, Familiar searches the event trace for iterative patterns. In figure 2b, for example, the sequence of observed event types is *activate*, *select*, *set*, *select*, *set*. The predictions in figure 2d show that Familiar elicited a simple cycle of two steps, *select* and *set*, from this sequence.

Two *sequence recognition schemes* (SRS) are used to search for patterns: *SRS-simple* and *SRS-noisy*. The SRS are executed in parallel: their input is the sequence of event types, and their output a set of *candidate patterns* detected in this sequence. After the fifth event of figure 2, three candidates emerge. *SRS-simple* reports a pattern of two steps, *select* and *set*, described as (*select*, *set*)*, which has been observed for two iterations (events 2–3 and 4–5 in figure 2b) and is predicted in figure 2d. *SRS-noisy* also reports this pattern, along with the sequence (*activate*, *select*, *set*)* which is observed in events 1–3 and *nil*, 4–5 and predicted in figure 4a. The *nil* value indicates that no observed event corresponds to the first command.

The sequence recognition manager uses the rules in figure 11a, which were derived from training tasks (section 6.1), to select a single pattern to underpin the cycle presented to the user. In figure 2d, Familiar chose the pattern predicted by *SRS-simple*, which is estimated to be correct with 97% probability (figure 11a, line 7), as opposed to 69% for *SRS-noisy*'s two-event candidate (line 5) and 11.1% for its three-event candidate (line 4).

3.2.2. *Sequence recognition schemes*

Although Familiar implements only two sequence recognition schemes, it is easy to add new ones. Each scheme must be able to detect, report, and update patterns in a given event trace, and provide internal estimates of the predicted pattern's likelihood (used in section 5.1).

SRS-simple detects repeated cycles, and often reports more than one at a time. Given the sequence *abababab*, it will find *abab* repeated twice and *ab* four times. Using it alone, Familiar can detect the same patterns as Eager (Cypher 1993b). *SRS-noisy* makes allowances for mistakes in the user's demonstration. It reports the correct pattern (amongst others) if the demonstration contains at least one correct iteration, or if every iteration contains all the important steps in the pattern. Briefly, it notes the most recent event's type, locates all events of that type

(a)

```
1   if kind-of-SRS = SRS-noisy then
2       if times-confirmed-correct-passively > 1 predict 97.1%
3       else if times-confirmed-correct-passively <= 1 then
4           if number-of-complete-cycles <= 1 predict 11.1%
5           else if number-of-complete-cycles > 1 predict 69.0%
6   else if kind-of-SRS = SRS-simple then
7       if number-of-commands <= 40 predict 97.0%
8       else if number-of-commands > 40 then
9           if proportion-of-agreeing-patterns > 0.285714 predict 100%
10          else if proportion-of-agreeing-patterns <= 0.285714 then
11              if number-of-commands <= 41 predict 70.6%
12              else if number-of-commands > 41 predict 0%
```

(b)

```
1   if PAS-kind = PAS-constant then
2       if prop-agreeing-predictions <= 0.666667 then
3           if PAS-specific-2 <= 3 predict 0%
4           else if PAS-specific-2 > 3 then
5               if number-of-iterations > 5 predict 1.7%
6               else if number-of-iterations <= 5 then
7                   if agreeing-predictions > 0 predict 100%
8                   else if agreeing-predictions <= 0 then
9                       if PAS-specific-1 > 1 predict 100%
10                      else if PAS-specific-1 <= 1 then
11                          if number-of-iterations > 2 predict 0%
12                          else if number-of-iterations <= 2 then
13                              if other-predictions <= 0 predict 75.0%
14                              else if other-predictions > 0 then
15                              if PAS-specific-2 <= 5 predict 75.0%
16                              else if PAS-specific-2 > 5 predict 0%
17          else if prop-agreeing-predictions > 0.666667 then
18              if PAS-specific-1 > 3 predict 99.7%
19              else if PAS-specific-1 <= 3 then
20                  if number-pp <= 0 predict 97.1%
21                  else if number-pp > 0 then
22                      if other-predictions <= 1 predict 14.3%
23                      else if other-predictions > 1 predict 100%
    …
43  else if PAS-kind = PAS-set then
44      if proportion-true-pp > 0 predict 100%
45      else if proportion-true-pp <= 0 then
46          if agreeing-predictions <= 0 predict 7.3%
47          else if agreeing-predictions > 0 predict 100%
```

Figure 11. Estimating the probability that (a) a pattern, (b) a parameter, is correct.

in the trace, assumes that these are the final events in each iteration and creates a pattern from the event types that are common to every iteration. In figure 8, for example, events 1 through 16 can be represented as *abcbcdeafbcdeagf* (where $a = activate\ Finder$, $b = select$, $c = open$, $d = activate\ GIFConverter$, $e = save$, $f = delete$, and $g = set$). The most recent event has type f, so *SRS-noisy* splits the

sequence into two iterations ending with event *f*, with corresponding subsequences *abcbcdeaf* and *bcdeagf*, and collates the events that occur in both: *bcdeaf*. It is the last of these patterns that is selected by the sequence recognition manager (figure 8c).

3.3. *Pattern analysis*

The predicted pattern is sent to the pattern analysis manager, which breaks it into constituent commands, identifies the commands' parameters and predicts them by invoking pattern analysis schemes. The sequence is recombined and displayed in the prediction window. Consider the predicted cycle (*select*, *set*)* in figure 2. The first command has one parameter, whose value is predicted to be *file* '*apple*' *of folder* '*fruit*', leading to event 6. The second has two parameters, whose predictions are *position of selection* and {144,29} respectively, leading to event 7. The two commands make up the predicted iteration in figure 2d.

3.3.1. *Evaluating competing predictions*

Each parameter is predicted by competing pattern analysis schemes (PAS). Each is given a sequence of parameter values as input, and makes zero or one predictions, optionally using contextual information from the event trace and Familiar's other knowledge sources. Consider the direct parameter of the *select* command in the example above. According to the pattern, the previous values of the parameter were *file* '*plum*' and *file* '*peach*' (figure 2b, events 2 and 4). Two pattern analysis schemes predict the value at event 6: *PAS-constant* predicts *file* '*peach*' (the previous value) and *PAS-set* predicts *file* '*apple*'.

Just one prediction must be chosen for each parameter, and the pattern analysis manager uses rules derived from training tasks using ML techniques (section 5.2) to estimate the probability that each prediction is correct, then chooses the most likely (in the event of a tie, it chooses randomly between the most likely). Figure 11b shows the rules relating to *PAS-constant* and *PAS-set*. In this example, the rules estimate *PAS-set*'s prediction's probability as 7.3% (line 46), and *PAS-constant*'s as 0% (line 3). Familiar selects the former prediction and suggests *file* '*apple*' (figure 2d).

3.3.2. *Pattern analysis schemes*

Familiar incorporates five pattern analysis schemes, summarized in table 1. The architecture is flexible: more can be added. Given a parameter's past values, each scheme must be able to predict the next value; report, explain, and update its predictions; and (optionally) provide an accuracy estimate.

PAS-constant predicts that a parameter's next value will remain unchanged. Extrapolating from the trace in figure 10 it correctly predicts that event 7's *set* parameter is *FormulaR1C1 of ActiveCell*, and incorrectly predicts the *to* parameter as '*August*'. *PAS-constant* always makes a prediction; thus Familiar always has at least one prediction to display. Figures 5b and 7a show examples of the explanations it generates. The accuracy estimate is the number of times the parameter value has remained constant.

PAS-extrapolation finds textual patterns in parameter values. Given the trace in figure 10, it extrapolates the three *Select* parameters to predict a next value of *Range* '*R1C5*', and recognizes that *July* and *August* are elements of an enumeration whose next element is *September*. Its accuracy estimate counts how often the parameter value has been consistent with the pattern it is extrapolating. The explanation is

Table 1. Summary of the pattern analysis schemes.

(a)	Name	PAS-constant
	Summary	predict the last value of the parameter
	Accuracy estimate	number of times the parameter value has been constant
	Explanation title	Constant value
	Explanation text	The parameter < parameter name > is always predicted using the value < prediction >
(b)	Name	PAS-extrapolation
	Summary	find alphabetic, numeric, and enumeration sequences
	Accuracy estimate	number of times the sequence is consistent
	Explanation title	Sequence of values
	Explanation text	The parameter < parameter name > is predicted in < number of tokens > parts. Part 1 is found by < part 1 explanation >; Part 2 is found by < part 2 explanation >, etc.
(c)	Name	PAS-previous
	Summary	find identical parameter values
	Accuracy estimate	none
	Explanation title	Matches previous value
	Explanation text	The parameter < parameter name > is the same as the < parameter name > parameter of event < event number >
(d)	Name	PAS-set
	Summary	iterate over containee objects
	Accuracy estimate	none
	Explanation title	Set iteration
	Explanation text	The parameter < parameter name > is predicted by choosing the next < containee file type > from < container >
(e)	Name	*PAS-ML*
	Summary	create conditional rules based on contextual information
	Accuracy estimate	statistical significance of rule
	Explanation title	Choice of value
	Explanation text	The parameter < parameter name > depends on the value of < contextual attribute description >

constructed from the parameter name and descriptions of each token in the predicted string.

PAS-previous searches the event trace for parameters that consistently have the same value as the target parameter. This is useful when multiple operations are performed on the same object and in tasks that cannot be fully automated, because an object used early in a cycle is often reused later. The explanation identifies the parameter name and event number that supports the prediction.

PAS-set detects iteration over sets of objects that have the same container object. In figure 2, for example, the user iterates over all *files* in a *folder*. When the same container appears in consecutive iterations, *PAS-set* asks the application for all containee objects. In events 2 and 4 of figure 2b, the container is *folder* '*fruit*', and two containees are observed: *file* '*plum*' (line 2) and *file* '*peach*' (line 4). *PAS-set* assumes that the user is working through all *file* objects contained by *folder* '*fruit*', and fetches the remaining files with the AppleScript command *tell application* '*Finder*' *to get every file of folder* '*fruit*'. It then predicts the first containee not already seen in the trace (figure 2d, event 6). Explanations are formulated by identifying the containee type and the name of the container, as illustrated in figure 5c.

1	if *creator type of selection* = *MSWD* then
2	predict folder 'word processor' of folder 'Documents'
3	else if *creator type of selection* = *XCEL* then
4	predict folder 'spreadsheets' of folder 'Documents'
5	otherwise make no prediction

Figure 12. Predicting the *to* parameter based on file type.

PAS-ML predicts parameters using conditional rules based on contextual information gathered by the event recorder. Section 2.3 described a simple classification task: the user sorts files into folders for word-processor and spreadsheet documents. The difficulty is learning the value of the *move* command's *to* parameter. There is no obvious pattern in the destination folders—the previous values are *word processor*, *word processor*, *spreadsheet*, *spreadsheet* and *word processor*—so the event trace does not provide enough information for prediction. *PAS-ML* seeks contextual information that may influence the prediction, and constructs the conditional rule in figure 12 to explain the user's actions. After event 18, the *file type of selection* at event 19 is *XCEL* (figure 7d), so *PAS-ML* predicts the *to* parameter's next value to be *folder* '*spreadsheets*' *of folder* '*Documents*' (using figure 12, line 4) and suggests this in figure 7e.

4. Learning conditional rules

The previous two sections described the Familiar architecture, which comprises an event recorder, a sequence recognition manager and two sequence recognition schemes, and a pattern analysis manager along with five pattern analysis schemes and a way of evaluating competing predictions. One of these pattern analysis schemes, *PAS-ML*, uses simple ML to infer conditional rules. Although the example in section 3.3.2 shows how these rules are formed and applied, it glosses over two significant problems. First, hundreds or thousands of contextual parameters can participate in rules. Second, rules are based on very few examples. This section describes how *PAS-ML* uses permutation tests, a statistical technique well suited to sparse data, to solve both these problems.

4.1. *Finding attributes in instance information*

Conditional rules are learned whose 'class' is the next value of the parameter being predicted and whose attributes are drawn from Familiar's instance information (section 3.1.2). The rules comprise features appearing directly in the event trace and contextual information gathered from the application.

Figure 13a shows a new task, in which files are assigned labels based on their size.[2] The recorded sequence of *to* parameter values exhibits no apparent pattern; so Familiar requires further information to learn this task. Figure 13b lists the parameter values, along with potentially important contextual information—the *current selection* at the time of the *set* command, the selection's *file type* and its *size*. Contextual information is also shown for the upcoming event 19.

It is difficult to choose good contextual information. Typically there are very few training instances and an unbounded number of potential attributes. If too little context is considered, a crucial attribute may be overlooked. With too much, the

(a)

1	Activate -- Finder
2	select file ‘a’ of folder ‘letters’ of folder ‘tasks’
3	set label index of selection to 4
4	select file ‘b’ of folder ‘letters’ of folder ‘tasks’
5	set label index of selection to 2
6	select file ‘c’ of folder ‘letters’ of folder ‘tasks’
7	set label index of selection to 1
8	select file ‘d’ of folder ‘letters’ of folder ‘tasks’
9	set label index of selection to 4
10	select file ‘e’ of folder ‘letters’ of folder ‘tasks’
11	set label index of selection to 3
12	select file ‘f’ of folder ‘letters’ of folder ‘tasks’
13	set label index of selection to 5
14	select file ‘g’ of folder ‘letters’ of folder ‘tasks’
15	set label index of selection to 2
16	select file ‘h’ of folder ‘letters’ of folder ‘tasks’
17	set label index of selection to 3
18	select file ‘i’ of folder ‘letters’ of folder ‘tasks’

(b)

event	**parameter**	**selection**	**file type**	**file size**
3	4	file ‘a’	TEXT	64
5	2	file ‘b’	GIF	128
7	1	file ‘c’	GIF	32
9	4	file ‘d’	TEXT	64
11	3	file ‘e’	JPEG	96
13	5	file ‘f’	APPL	256
15	2	file ‘g’	APPL	128
17	3	file ‘h’	JPEG	96
19	unknown	file ‘i’	JPEG	128

(c)

if *current selection = file ‘a’* then predict 4
else if *current selection = file ‘b’* then predict 2
else if *current selection = file ‘c’* then predict 1
else if *current selection = file ‘d’* then predict 4
else if *current selection = file ‘e’* then predict 3
else if *current selection = file ‘f’* then predict 5
else if *current selection = file ‘g’* then predict 2
else if *current selection = file ‘h’* then predict 3
otherwise make no prediction

(d)

if *file type of selection = TEXT* then predict 4
else if *file type of selection = GIF* then predict 2
else if *file type of selection = JPEG* then predict 3
else if *file type of selection = APPL* then predict 5
otherwise make no prediction

(e)

if *size of selection* = 32 then predict 1
else if *size of selection* = 64 then predict 4
else if *size of selection* = 96 then predict 3
else if *size of selection* = 128 then predict 2
else if *size of selection* = 256 then predict 5
otherwise make no prediction

Figure 13. Predicting the *to* parameter based on file type; (a) an event trace; (b) contextual data for *to* (c) the current selection, (d) its file type; (e) its size.

learning scheme may overfit, and will take too long for interactive use. Section 3.1.2 described how Familiar gathers potentially useful information about each object it encounters. The amount is overwhelming. Consequently the contextual information is restricted by adding only the first few attributes to the initial training dataset, in the order in which the data were gathered. A further few attributes are added whenever an incorrect prediction occurs. In total, 71 attributes of the type shown in figure 13b are added to the training dataset in this example.

4.2. *Learning conditional rules*

Figures 13c–e shows some rules that could be used to classify the *to* parameter based on events 3–17 of figure 13b. The first tests the *current selection* attribute, but cannot predict event 19 because there is no branch for *file* '*i*'. The second tests the *file type of selection* attribute, which is *JPEG* at event 19, and (incorrectly) predicts the value 3. The third tests the *size of selection* attribute, currently 128, and (correctly) predicts 2.

Rules are built from a single attribute. A rules is created for each attribute, and the most accurate one is selected. This is the 1R scheme (Holte 1993); Familiar's implementation is essentially the 1Rw variant but makes no prediction in the default case.

4.3. *Permutation tests*

Choosing the rule with greatest accuracy on the training set, as 1R does, biases towards attributes with many distinct values. In figure 13d, *file type of selection* classifies 75% of the training instances correctly, while both *current selection* and *size of selection* are 100% accurate. 1R chooses one of these two at random, even though *current selection* is unlikely to be useful for predicting future values because it has different values for every instance.

To solve this problem, *PAS-ML* replaces the accuracy measure with a permutation test (Good 1994), which measures the probability that by chance alone an attribute will classify as accurately as it does in actuality. The measure is not estimated but calculated by considering all the possible permutations of class and attribute values (Frank and Witten 1998).

In figure 13b the class is the *to* parameter. The class values 2, 3 and 4 appear twice, while 1 and 5 appear once, so the distribution is 2, 2, 2, 1, 1 for classes 2, 3, 4, 1, 5 respectively. Given a particular attribute and its distribution, permutation tests calculate the probability p that the observed level of prediction accuracy occurs by chance alone.

The *current selection* attribute predicts the training data perfectly using the rule in figure 13c. However, its distribution is 1, 1, 1, 1, 1, 1, 1, 1, because every attribute value (*file* '*a*', *file* '*b*'...) appears exactly once, and any such rule will classify the training data correctly with probability $p = 1$, irrespective of the class values.

Size of selection predicts the class perfectly using the rule in figure 13e, and has distribution 2, 2, 2, 1, 1 because three values appear twice (64, 96, 128) and two appear once (32, 256). This distribution yields 100% accuracy by chance with probability $p = 0.014$, obtained by taking the (inverse) ratio of permutations of class and attribute values consistent with the distribution (861 in this case) and those that are 100% accurate (12). Permutation tests handle less obvious cases. *File type of selection*, with accuracy 75%, has distribution 2, 2, 2, 2 because every attribute value appears twice in the training data. The probability of this distribution yielding 75% accuracy by chance is $p = 0.20$.

In summary, *PAS-ML* chooses the rule that is least likely to yield the observed accuracy by chance alone—in this case, that based on *size of selection*. It begins making predictions after observing three iterations of a task. It subsequently makes predictions whenever possible, though early predictions have low significance because there are few examples and several attributes, and with so little data it is impossible to have much confidence in the result. The scheme provides the pattern analysis manager with an estimate of $1-p$ for the chosen rule's accuracy.

5. Guiding prediction

Familiar uses ML algorithms trained on historical data to guide prediction. Both the sequence recognition manager (section 0) and the pattern analysis manager (section 0) invoke competing schemes, then evaluate the candidates by using rules that estimate the probability that they are correct (figure 11). In each case, Familiar uses the candidate that it estimates is most likely correct. Both rulesets are obtained using the C4.5 decision-tree learner (Quinlan 1993).

5.1. *Sequence recognition*

Data was gathered from seven iterative tasks (see Appendix), performed by a single experienced user. Familiar detected many patterns, storing appropriate attributes for each. Later, the pattern was judged correct if it was consistent with the commands subsequently executed by the user.

Each instance consists of a class variable—*correct* or *incorrect*—and twelve attributes, chosen because they are easily calculated and featured in earlier attempts to define heuristics for choosing patterns. The attributes are described in figure 14. The first three are SRS-specific and include the scheme that detected the pattern and the internal accuracy estimate mentioned in section 3.2.2. The remainder describe the scheme's performance on previous iterations of the pattern.

The training data comprises 1466 instances, each representing a candidate pattern. There were many more correct (1380) than incorrect (86) patterns because noise events are almost never generated after the user starts interacting with the prediction window, so patterns were easily detected and consistently correct.

A classifier was built using C4.5, modified to split on the *kind-of-SRS* attribute at the root node. This restriction ensures that branches on the internal accuracy

	Attribute name	Description
1	kind-of-SRS	kind of sequence recognition scheme (SRS)
2	accuracy	accuracy estimate calculated by SRS
3	SRS-specific	auxiliary SRS-specific attribute
4	cycle-length	length of the pattern cycle in commands
5	number-of-cycles	number of partial cycles in pattern history
6	number-of-complete-cycles	number of complete cycles in pattern history
7	number-of-commands	number of commands in pattern history
8	number-of-agreeing-patterns	number of agreeing pattern cycles
9	proportion-of-agreeing-patterns	proportion of agreeing pattern cycles
10	times-confirmed-correct-passively	times pattern confirmed correct passively
11	times-confirmed-correct-actively	times pattern confirmed correct actively
12	times-confirmed-incorrect	times pattern confirmed incorrect

Figure 14. Attributes of candidate patterns.

estimates occur after the *kind-of-PAS* attribute, which reflects the semantics of the situation—accuracy estimates are only relevant in the context of the sequence recognition scheme they describe.

Figure 11a shows rules derived from the decision tree. Leaves are augmented with probability estimates derived from C4.5's accuracy figures. The *SRS-noisy* branch (lines 2–5) uses the number of times the pattern was confirmed correct by user actions in the application interface and the number of complete cycles observed. The most prominent attribute in the *SRS-simple* branch (lines 7–12) is the number of observed user actions consistent with the pattern, which branches both at 40 and 41, suggesting possible overfitting.

5.2. *Pattern analysis*

Training data for the pattern analysis classifier was gathered at the same time as that described above for sequence recognition. As before, each instance consists of a class variable—*correct* or *incorrect*—and (in this case) 17 attributes, chosen in an analogous manner—for example, the first attribute identifies the pattern analysis scheme making the prediction and the next is its internal accuracy estimate (section 3.3.1). The dataset contains 1875 instances representing parameter predictions, 1459 of which are *correct*.

Again the classifier was built off-line using C4.5, modified to split on *kind-of-PAS* at the root, to produce the rules in figure 11b. Interestingly, the first test in three of the five main branches involves the number of predictions that agree with the current one. The 'majority vote' heuristic that this suggests performs poorly in practice because there is seldom a clear majority. The estimated correctness probabilities in figure 11b handle this problem gracefully.

6. Evaluation

A user evaluation of Familiar's interface provided an opportunity to assess Familiar's inferencing (Paynter 2000). Each participant was asked to perform several iterative tasks. Because Familiar works with existing applications rather than being built into a particular prototype application as most PBD systems are, the evaluation tasks are quite realistic. As they worked, the predictions made for each parameter were recorded. This data was used to test the accuracy of the rules for selecting parameter predictions by comparing them to four other selection criteria: three heuristics and a simulated incremental learning scheme. The version of Familiar used in the evaluation let users select the best pattern themselves, so we restrict attention to parameter predictions.

6.1. *Training and test data*

Training data was gathered from the same training tasks as in section 5, while test data was gathered from users who participated in the evaluation of Familiar's interface (see Appendix). Table 2 gives the size of the datasets.[3] There were a variety of training tasks, but they were performed only once by a single user; whereas both test tasks were performed by ten users, sometimes more than once (with variations). The test tasks were different from the training tasks, and were performed by ten university students with varying academic backgrounds and levels of computer experience (Paynter 2000).

Table 2. Evaluation dataset statistics.

	Tasks	Users	Times performed	Instances in dataset
Training dataset	7	1	1	4280
Testing dataset	2	10	1–3	21 300

6.2. *Measuring performance*

The obvious way to evaluate a PBD tool is to make it available to users and measure the resulting savings in terms of time taken or actions performed. Such measures are questionable, however, because they can be manipulated at will simply by adjusting the number of iterations of the task that the user is asked to perform. A tool might increase the effort required to perform a task that is repeated twice, break even at five repetitions, provide meaningful savings at 10 and unprecedented efficiencies at 1000. Instead, we measure Familiar's performance by examining the accuracy of its predictions.

Familiar correctly classifies 97.9% of training instances and 96.8% of test instances. However, these accuracy figures are not very informative. Instead, we measure the number of times that a correct choice is made from the competing predictions of the same parameter. For example, in the scenario of section 0 two pattern analysis schemes predict the direct parameter of a *select* event, one correctly and the other incorrectly. Each of these predictions can be represented by an instance in the dataset: one with class *correct*, the other with class *incorrect*. For evaluation purposes, Familiar is used to choose between these two instances and present the corresponding prediction to the user; it is assessed by counting the number of times it chooses an instance with class *correct*. The purpose is to identify selection criteria that maximize the number of correct predictions that are chosen from those that are available. There are 7737 sets of predictions in the test data. Of these, 7455 include a true prediction (sometimes more than one), and 4381 include only true predictions. These figures represent bounds on the number of correct predictions that can be obtained from this dataset: no method can choose fewer than 4381 or more than 7455 correct predictions. These limits are shown in rows 1 and 7 of table 3.

6.3. *Comparing performance*

Three heuristics for selecting predictions were used in earlier versions of Familiar, but were later superseded by C4.5 decision rules. The *random choice* method simply

Table 3. Comparison of techniques for choosing the best prediction.

	Selection criteria	Correct predictions chosen	Incorrect predictions chosen
1	Lower limit	4381	3356
2	Random choice	5936	1801
3	Complexity	6792	945
4	Consecutive true	7249	488
5	C4.5 rules	7400	337
6	Adaptive rules	7430	307
7	Upper limit	7455	282

chooses a prediction randomly. The *complexity* method chooses the most complex prediction using a fixed (and rather arbitrary) ranking of pattern analysis scheme complexity (reflecting the order in which the schemes were implemented, namely *PAS-constant*, *PAS-extrapolation*, *PAS-previous*, *PAS-ML* and *PAS-set*.) The *consecutive true* method chooses the prediction that has been correct for the greatest number of iterations in a row. These methods proved increasingly accurate for selecting correct pattern analysis schemes. Their performance on the test data appears in rows 2–4 of table 3.

Row 5 shows the number of correct predictions chosen by the C4.5 rules. The improvement is substantial: 151 more correct predictions than the best of the three fixed heuristics, falling short of the upper limit by only 55 instances. This shows that replacing *ad hoc* heuristics with a standard ML technique improves end-user performance substantially.

6.4. *Adapting to the user*

Familiar learns the C4.5 model off-line from the training data (which, of course, is different from the test data). An alternative strategy is to learn incrementally, updating the classifier dynamically when the accuracy of each new prediction is confirmed. This is a more realistic scenario that reflects how Familiar would be used in practice. Over time, the classifier may adapt to the user's individual style and to their specific tasks.

A second experiment simulated incremental learning. Initial decision rules, built from the training data, are used to make the first prediction. After each prediction, the classifier is retrained to incorporate the new examples, thereby modelling a dynamic learner that updates itself after every user action. The experiment was repeated for each user who participated in the evaluation (starting from the original training data each time).

The combined results, in row 6 of table 3, show a further increase in performance: over half the remaining incorrect selections with correct alternatives (30 of 55) have been replaced by correct selections. Only 25 incorrect actions were chosen when correct alternatives were available.

Most of the improvement occurs in situations where the static classifier is unsuited to a particular user's style of demonstration. In the user evaluation, we observed that the static decision rules sometimes chose the same incorrect prediction for two or more consecutive iterations, despite new demonstrations from the user showing that the prediction was incorrect. The adaptive version immediately incorporates corrections into its model, so repeated mistakes occur less often.

7. Discussion

With few exceptions (see Section 7.3), PBD relies on *ad hoc* algorithms and specialized heuristics, Familiar demonstrates two situations where standard ML techniques are applicable—and demonstrably superior. In the first, decision stumps are learned from very few examples, a common scenario in PBD. In the second, a trained classifier guides prediction; and we further improve prediction by adapting to specific users through incremental learning.

7.1. *Learning conditional rules*

Familiar uses 1R (Holte 1993) to find relationships between a parameter to be predicted and data in the event trace. Bias towards attributes with many values is resolved by replacing 1R's simple accuracy measure with a permutation test (Good 1994).

Permutation tests are well suited to PBD because they work with very small datasets. Whereas ML researchers might consider a 100-instance dataset small, the problem in figure 13 has only eight instances. Moreover, this is large by PBD standards! Most users will bridle at the prospect of demonstrating a task eight times.

The plethora of potential attributes presents a further challenge: small datasets are overwhelmingly likely to contain attributes that are apparently highly predictive but nevertheless irrelevant. Permutation tests calculate the probability that each rule will yield its observed accuracy level by chance alone. With little data, confidence in a rule is necessarily low, but different rules are at least compared on an equal footing. With copious data, permutation tests become computationally demanding. Though this impacts Familiar's current implementation, efficient algorithms for large datasets are known (Frank 2000). Also, further work is required to rationalize Familiar's *ad hoc* policy of adding new attributes, a few at a time, whenever incorrect predictions occur. This is essentially a feature selection problem in a setting where features are not listed in advance but generated dynamically.

We focus on decision stumps because few example tasks require rules based on more than one attribute (Paynter 2000). If necessary, Familiar could easily be supplied with more complex predictors in the form of new pattern analysis schemes. Of course, the risk of overfitting increases as complexity grows—but fortunately permutation tests address this problem too.

7.2. *Guiding prediction*

Familiar uses ML techniques to choose between competing predictions, and evaluation shows that this outperforms three simple heuristics used originally. Moreover, incremental algorithms improve performance still further. Though the evaluation suggests the real-world utility of the ML approach, it has some shortcomings.

First, only two tasks were performed by only ten users: more comprehensive evaluation is desirable. Second, Familiar's performance will depend on the composition and execution of the training tasks. Although as many tasks were demonstrated as time and resources allowed, ideally more would be used. The seven tasks involve a range of applications, employ all pattern analysis schemes, and differ from the test tasks. Nevertheless, the rules in figure 11a probably overfit the training data.

Another shortcoming is the *ad hoc* way in which attributes made available to C4.5 were selected—although the evaluation suggests that the current ones are adequate, at least for the test tasks. It is interesting to observe the attributes that C4.5 chose to use. The sequence recognition tree uses only 5 of 12 available attributes, including the number of commands in the event trace that are consistent with the proposed pattern, how often the pattern has been confirmed correct, and the number of agreeing patterns (figure 11a). The pattern analysis tree uses 10 of 17 available attributes: the number and proportion of agreeing predictions

appear frequently, and the attributes not used are predominantly measures of past performance. In both cases, there is insufficient evidence to identify outstandingly good attributes, and the most prominent ones may change with further training data, or different training tasks, or even if the training tasks were executed by another user or with different mistakes.

Despite these caveats, adapting the model to the user's behaviour is beneficial. However, some issues require further study. Long-term test data may indicate that recent behaviour should be weighted more heavily (as Mitchell *et al.* (1994) found with the Calendar Learning Apprentice). Also, the simulated incremental learning strategy is computationally expensive. Because PBD systems must offer instant feedback, a true incremental learner is necessary in practice.

7.3. *Related research*

A large number of systems have been proposed for PBD (Cypher 1993a, Lieberman 2001); we mention only those directly related to the present paper. Familiar's nearest ancestor is Eager (Cypher 1993b), which laid the groundwork for our focus on iterative tasks with interaction with the application through AppleScript. Eager also pioneered the two-level architecture that underlies Familiar: first detect cycles, then fill in parameter predictions. Familiar extends Eager by eliminating the dependence on modified applications and platforms, and by replacing hand-crafted prediction heuristics with the noise-tolerant and application-independent rule inference system described in this paper. Other PBD projects like AgentScript and Tatlin utilize the AppleScript platform, but tackle quite different problems (Lieberman 1998).

Early PBD systems tended to rely on domain-specific heuristics to learn tasks (Cypher 1993a), but more recent projects have attempted to exploit machine learning schemes. For example, CIMA uses an adaptation of the PRISM concept learner (Maulsby 1994), the Calendar apprentice (Mitchell *et al.* 1994) and Gamut (McDaniel and Myers 1998) use the ID3 decision tree learner, and SmartEdit (Lau *et al.* 2001) uses the version space algorithm. These applications require more complex rules than Familiar because they work in specific domains and have detailed domain knowledge.

The present paper focuses on rules based on a single attribute because few of the tasks that Familiar was designed to solve required more (Paynter 2000). More complex algorithms could be slotted in alongside the prediction schemes currently in use. A practical difficulty is that when datasets have few instances and many attributes, and rules are formed from combinations of attributes, it is highly likely that good predictors will occur purely by chance. Permutation tests ameliorate this problem, as they do in the single-attribute case applied in this paper. The Calendar apprentice (Mitchell *et al.* 1994) does not encounter this problem because it is applied to a single repetitive task with a fixed set of attributes and training data based on long-term use, while Gamut (McDaniel and Myers 1998) narrows the number of attributes through heuristic search and explicit user interaction. SmartEdit (Lau *et al.* 2001) uses version space algebra to quickly home in on the appropriate set of attributes to use.

8. Conclusion

The work described in this paper highlights the advantages and the difficulties of using ML in adaptive user interfaces. Familiar uses ML techniques to replace

ad hoc heuristics, and reaps great benefit from doing so—benefit in terms of both rational design and end-user performance. Not only is Familiar 'adaptive' in that it learns specialized tasks for individual users, it 'adapts' in a deeper sense to each individual user's style—and we have demonstrated that this improves performance.

However, the iterative, improvement-driven, development style that has been so successful in other ML applications cannot be applied in this setting. Not only is no standard corpus of data available for evaluating PBD systems, but it is impossible *in principle* to create a fixed training/testing dataset for any given PBD problem. Each change to Familiar alters the test data by affecting how users react to it. It is not possible simply to rerun experiments to assess the effect of a change: re-evaluation requires new user tests, which are exceedingly time-consuming. Moreover, variations between users and between sessions make any evaluation results far less reliable than when a fixed body of training and test data is available. While this frustrates the development process, we believe that our results demonstrate that ML can fruitfully be applied to PBD.

Notes

1. An alternative approach would be to assume that the 18th action was performed as predicted when predicting the 19th action.
2. In the Macintosh OS, files can be assigned one of eight labels and are subsequently displayed in that label's colour. In AppleScript, a file's label is accessed through its *label index* property.
3. The data described in section 6.2 relate to a later performance of the same tasks, so the number of instances differ.

References

Cypher, A., (ed.), 1993a, *Watch what I do: Programming by Demonstration* (Cambridge, MA: MIT Press).

Cypher, A., 1993b, Eager: programming repetitive tasks by demonstration. In A. Cypher (ed.), *Watch what I do: Programming by Demonstration* (Cambridge, MA: MIT Press), pp. 205–217.

Erickson, T., 1997, Designing agents as if people mattered. In J. M. Bradshaw (ed) *Software Agents* (Menlo Park, CA: AAAI Press; Cambridge, MA: MIT Press), pp. 79–96.

Frank, E., 2000, Pruning decision trees and lists. PhD thesis, Department of Computer Science, University of Waikato, New Zealand.

Frank, E., and Witten, I. H., 1998, Using a permutation test for attribute selection in decision trees. In *Proceedings of International Conference on Machine Learning*, San Francisco, CA (Madison, WI: Morgan Kaufmann), pp. 152–160.

Good, P., 1994, *Permutation tests: A practical guide to resampling methods for testing hypotheses* (New York: Springer-Verlag).

Hendry, D. G., and Green, T. R. G., 1994, Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. *International Journal of Human-Computer Studies*, **40**: 1033–1065.

Holte, R. C., 1993, Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, **11**(1): 63–91.

Lau, T. A., Wolfman, S. A., Domingos, P., and Weld, D. S., 2001, Learning repetitive text-editing procedures with SMARTedit. In H. Lieberman (eds) *Your wish is my command* (San Francisco, CA: Morgan Kaufmann), pp. 209–226.

Lieberman, H., 1998, Integrating user interface agents with conventional applications. *Knowledge-Based Systems*, **11**(1): 15–24.

Leberman, H., (ed.), 2001, *Your wish is my command* (San Francisco, CA: Morgan Kaufmann).

Maulsby, D., 1994, Instructible agents. PhD thesis, Department of Computer Science, University of Calgary, Canada.

McDaniel, R. G., and Myers, B. A., 1998, Getting more out of programming by demonstration. In *Proceedings of Human Factors in Computing Systems (CHI)*, New York (Pittsburgh, PA: ACM Press). pp. 442–449.

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., and Zabowski, D., 1994, Experience with a Learning Personal Assistant. *Communications of the ACM*, **37**(7): 81–91.

Paynter, G. W., 2000, Automating iterative tasks with programming by demonstration. PhD thesis, Department of Computer Science, University of Waikato, New Zealand.

Quinlan, J. R., 1993, *C4.5: Programs for machine learning* (San Francisco, CA: Morgan Kaufmann).

A1. Appendix: training and test tasks

Here we describe the set of iterative tasks used to train and test the classifiers used in Familiar. Seven iterative tasks were used to train the classifiers, chosen to cover a range of applications and test all the pattern analysis schemes. Different tasks were chosen to generate the dataset for testing the classifier.

A1.1. *Training tasks*

A1.1.1. *Label by size*

The *letters* folder contains 26 files named for the letters of the alphabet. The task is to set the label of each file according to its size. Files 4 Kb or less will be labelled orange, 8 Kb or less green, 96 Kb or less yellow, 200 Kb or less blue, and the remainder red.

A1.1.2. *Label by kind*

The *extra files* folder contains 100 irregularly named files. There are six different kinds of file: Alias, BBEdit text file, Internet Explorer document, MCL document, Microsoft Word document and Text document. The task is to label files with a different label for each kind of file.

A1.1.3. *Spreadsheet computation*

A spreadsheet document contains a list of 500 records. The records are stored as numbers in column A and B. The task is to compute the average of column B for each of the ten blocks in the first ten cells of column C. (This task is drawn from Hendry and Green 1994).

A1.1.4. *Extended spreadsheet computation*

The spreadsheet file in the previous task now contains a list of 3000 records in blocks of 50. The task is to compute the average of column B for each of the 60 blocks in the first 60 cells of column C.

A1.1.5. *Resize tiles*

The folder *tiles to resize* contains a list of image files. These are all JPEG files, and all the pictures are approximately 60 pixels square. The task is to open each file, use GIFConverter to resize the image to 100 pixels square and save the modified version over the old version.

A1.1.6. *Move and rename files*

A folder contains a list of image files that are named for the days on which they were created, but not in a regular manner. The folder *new files numbered* is empty. The

Table 4. Test task 1: the calendar.

	July	August	September	October
1	Wed	Sat	Tue	Thu
2	Thu	Sun	Wed	Fri
3	Fri	Mon	Thu	Sat
...	...	...	...	...

task is to copy each file into the *new files numbered* folder and rename it. The files should be named *tile63.jpeg*, *tile64.jpeg*, *tile65.jpeg* and so on.

A1.1.7. *Position files*
The folder *tutorial example* contains a group of files named for the letters of the alphabet. They are not sorted in any particular order. The task is to arrange the files into a line across the window in name order. This task is similar to the example in section 2.1.

A1.2. *Test tasks*

A1.2.1 *Calendar*
The subject is asked to duplicate a printed calendar as a Microsoft Excel spreadsheet. The calendar comprised 645 characters in 221 cells; part of it is shown in table 4. (Note: column 1 and row 1 are left justified and the interior cells are right justified.) In its simplest form, the calendar task involved iterations of two high-level events, repeated on 225 (or 221) cells.

A1.2.2. *Image conversion*
This task spans three programs. The subject is asked to use a graphical FTP client to download image files from an FTP site onto the hard drive; to use an image manipulation program to convert them from PICT to JPEG format; to use the FTP client to return the modified files to the FTP site; and to use the operating system to clean up any copies left on the local machine. In total, 63 images were used, and approximately 15 high-level events must be performed to convert each of the images.