Working Paper Series
ISSN 1170-487X

Learning agents: from user study to implementation

by David Maulsby, Ian H Witten

Working Paper 96/8

April 1996

Learning agents: from user study to implementation

David Maulsby
CAMIS Medical Informatics Section
Stanford University
Room MSOB x215
Stanford, CA 94305-5479 USA

tel. +01 (415) 725–1672
fax. +01 (415) 725–7944
maulsby@camis.stanford.edu

Ian H. Witten
Department of Computer Science
University of Waikato
Private Bag 3105
Hamilton, New Zealand

tel. +64 (7) 838–4246
fax. +64 (7) 838–4155
ihw@cs.waikato.ac.nz

Abstract
Learning agents acquire procedures by being taught rather than programmed. To teach effectively, users prefer communicating in richer and more flexible ways than traditional computer dialogs allow. This paper describes the design, implementation and evaluation of a learning agent. In contrast to most Artificial Intelligence projects, the design centers on a user study, with a human-simulated agent to discover the interactions that people find natural. Our work shows that users instinctively communicate via "hints," or partially-specified, ambiguous, instructions. Hints may be input verbally, or by pointing, or by selecting from menus. They may be unsolicited, or arise in response to a query from the agent. We develop a theory of instruction types for an agent to interpret them. The implementation demonstrates that computers can learn from examples and ambiguous hints. Finally, an evaluation reveals the extent to which our system meets the original design requirements.

Keywords
Intelligent agents, instructibility, programming by demonstration, machine learning.

Learning agents: from user study to implementation

1 INTRODUCTION

Graphical, direct-manipulation user interfaces have brought a world of information to multitudes of people who have no inclination or ability to program a computer. But the need for some form of end-user programming is increasingly apparent. Professional people find themselves performing repetitive tasks once delegated to clerks and secretaries. Users with repetitive strain injuries want to minimize input through keyboard and mouse. And nearly all users want to customize interface behaviors to their personal needs. Application developers have responded with a barrage of new features, which are becoming a problem in themselves as users become lost in the interface.

Although end-user programming promises to help solve these problems, the question remains — how will users program? Most people have no acumen for formal notation, and it is well-known that people cope best with concrete, metaphorical expressions of ideas.[9] Perhaps the best representation, with programming in mind, is an annotated example. End users can teach the computer by demonstrating desired behavior within the actual user interface/application environment.[4] Demonstration can be augmented with instructions to suggest which features of data determine the results of an action; this speeds learning and effectively increases the variety of tasks which may be taught. Teaching demands less planning, analysis or understanding of the computer's internal representation than programming, but it raises new problems of how to correlate, generalize and disambiguate instructions. Human pupils interact with their teacher to solve these problems; a computer, with its more limited (and rather different) abilities, will need assistance that is specific to its "cognitive style." An agent that learns tasks from the user needs more (and more flexible) interaction than one that is programmed explicitly, and will fail in its mission if it confuses or frustrates the user.

With the aid of end-users, we have developed and tested interaction methods for teaching an agent. First, we conducted an experiment in which people train an agent, TURVY, how to edit a bibliography.[7] TURVY is simulated by a researcher hidden behind a screen, as shown in Figure 1. The tasks range from trivial (replace underlining with italics) to taxing (put all authors' given names and initials after their surnames). Users invented their own teaching methods and spoken commands. To ground the experiment in reality, limits were set on the agent's inference and language abilities, using a partially formalized model of instruction. This exercise established the

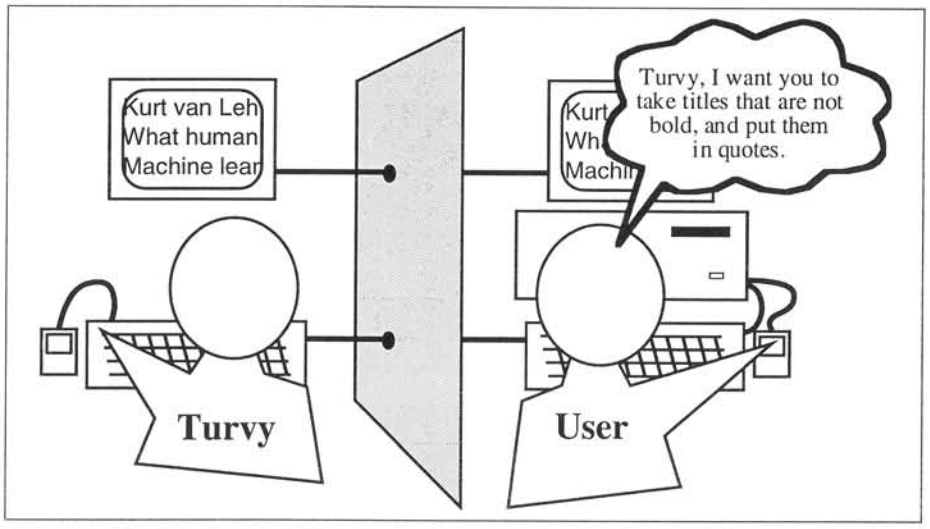

Kurt van Leh
What human
Machine lea
Kurt
Wh
Machi
Turvy, I want you to take titles that are not bold, and put them in quotes.
Turvy
User

Figure 1 The experimental setup

Before editing…	*After putting author and date into heading…*
	[Agre 88]
Philip E. Agre: The dynamic structure of everyday life: PhD thesis: MIT: 1988.	Philip E. Agre: The dynamic structure of everyday life: PhD thesis: MIT: 1988.
	[Angluin 83]
D. Angluin, C. H. Smith: "Inductive inference: theory and methods." *Computing Surveys 3 (15)*, pp. 237-269: September 1983.	D. Angluin, C. H. Smith: "Inductive inference: theory and methods." *Computing Surveys 3 (15)*, pp. 237-269: September 1983.
	[Michalski 86]
Michalski R. S., J. G. Carbonell, T. M. Mitchell (eds): *Machine Learning II:* Tioga: Palo Alto CA: 1986	Michalski R. S., J. G. Carbonell, T. M. Mitchell (eds): *Machine Learning II:* Tioga: Palo Alto CA: 1986
	[van Lehn 90]
Kurt van Lehn: *Mind bugs: the origins of procedural misconceptions:* MIT Press: 1990.	Kurt van Lehn: Mind bugs: the origins of procedural misconceptions: MIT Press: 1990.

Figure 2 A TURVY task: Make a heading with author's name and date

For each entry in the bibliography (i.e. each "bibliography" style paragraph), do this.

1. Find the first author's surname
   (it precedes the first comma or colon in the paragraph, and it comprises a single capitalized word, with perhaps a lowercase word preceding it);
2. Copy the surname, then go to the beginning of the paragraph and type "[", paste the surname and type "◊]↵"
   (the symbol ◊ means "blank" and ↵ means "paragraph break");
3. Set the style of the new paragraph to "citation;"
4. Find the last two digits of the publication date
   (they precede the period at the end of the paragraph, which may be omitted);
5. Copy the digits and paste them before the "]" typed in previously.

Figure 3 Programmatic specification of the task in Figure 2

feasibility of the general approach and established design specifications for the agent's learning system. In particular, it taught us the importance of being able to work with multiple sources of information, including ambiguous hints from the user. We then implemented an interactive machine learning system, CIMA, to meet these requirements. Although the system's interfaces to users and applications are still too rough-hewn to permit field trials, we tested it on transcripts from the TURVY experiment and found that it learns at roughly TURVY'S level of efficiency, exploiting examples, user hints and simple background knowledge.

This paper describes both the user study and the implementation. We begin with an example of Turvy in action, describe the different styles that are adopted by users when they interact with it, and then draw out the lessons learned for the design of agents. Next we turn to CIMA, giving an example of various different ways that the same task can be taught, depending on the user's teaching style. We discuss how the system manages interaction with the user, and how it interprets "hints," or partially specified instructions. Finally we briefly describe the initial evaluation study.

Pam: Take the last name, before the colon. [copies "Agre", pastes it to a new paragraph, types brackets, sets style]
This is a list of publications. Now we'll add the last two digits of the date. [types "88"]
Do you want another demo?

Turvy interprets Pam's question to mean the example is completed. Turvy has formed an initial generalization and now offers to edit the second entry (author "Angluin" in Figure 2a).

Turvy: I can try if you want.
Pam: OK.
Turvy: [picks "Smith"] Looking for word before first colon in paragraph, OK?
Pam: No—stop! You want the first author.
Turvy: Show me please.
Pam: [picks "Angluin"]
Turvy: You picked word before comma. You want word before first comma or colon?
Pam: Yes! The first comma or colon.
Turvy: [while doing actions] Copy, go to start of paragraph, OK?
Pam: OK.
Turvy: Insert new paragraph, type open bracket, paste, close bracket, change style to "citation", OK?
Pam: OK.

Now Turvy makes a mistake with the date. When Pam typed "88," Turvy assumed the number was either a constant or input.

Turvy: Type in blank space "88", OK?
Pam: No, the last two digits at the end.
Turvy: Show me please.
Pam: [picks "83"]
Turvy: Two digits before period at end of paragraph?
Pam: Yes.

... Later, Turvy encounters author "van Lehn", mistakenly selecting "Lehn". When the user corrects it, Turvy asks for an explanation.

Turvy: What's different about this case?
Pam: Hmm.
Turvy: You selected these two words, why?
Pam: Because these two words make up his last name.
Turvy: Is there some feature of this word [highlights "van"] I should look for? Lowercase word?
Pam: [hesitant] In this case, yes.

Turvy does the rest of the task without error.

Figure 4 Partial trace of Turvy learning task 3 from Pam

2 USER STUDY

The best way to understand what TURVY does is to see it in action. In Figure 2, the task is to teach it to make a heading for each bibliography entry, showing the primary author's surname and the date of publication. With no background knowledge of bibliographies, TURVY learns these concepts in terms of textual formatting and syntactic patterns. The relevant surname is normally the word before the first comma or colon, but sometimes it includes a lower case word like "van", or precedes an annotation like "(eds)". In some entries the author's name and initials are reversed (e.g. "Michalski R. S."). The date is the last two digits before the period at the paragraph's end, but sometimes the final period is missing. TURVY must ask about all these exceptions and learn special-case rules to handle them. Figure 3 gives a programmatic specification for the task.

A sample transcript from the videotaped record of the experiment appears in Figure 4. Pam, the subject, has already completed two tasks. Prior to the experiment, a facilitator told her that TURVY is simulated by a human being, and that it watches her actions and understands a bit of English. The facilitator gave no other suggestions on how to teach.

TURVY has a conversational interface,[2] in which either party may take the lead and both are learning about one another's goals. The learning system forms an initial hypothesis from the user's first example, and thereafter mixes prediction with further learning. It "annotates" its predictions with verbal feedback, encouraging the user to annotate her examples. When testing a particular hypothesis for the first time, it offers a detailed description; when confident, it reduces feedback and increases execution speed.

There are really three actors in this dialog: the subject, TURVY, and the researcher behind the curtain. This affects the dialog in subtle ways. Although instructed to teach TURVY, subjects are likely to have other goals—to finish the session as quickly as possible, to make a good impression on the researcher, or maybe to compete with him. These motivations make users more or less patient than they would be with a real agent and influence the amount and complexity of instructions they give. TURVY's goals are ostensibly to learn a program for the task and teach the user how to teach better, but as an experimental "instrument" designed to test people's ability to give hints about data descriptions, it probes for more details and is less forthcoming with its own hypotheses than might be appropriate in a real system. The researcher also wants to elicit as much experimental data as possible, while keeping the session on schedule. Thus he may use his discretion to make TURVY play dumb, or interpret a long silence from the user to mean that TURVY should propose a guess to get things moving.

2.1 Observations and results

Four users participated in a pilot study, and seven in the formal experiment. Although we did content analyses of user instructions and responses, we focused more on qualitative observations. When users encountered difficulty, we tried to find out why during a debriefing after the session.

Dialog styles. Users were evenly split between two quite different dialog styles — talkative and quiet — which seem to match with extroverted and introverted personalities. A typical talkative user (like Pam) gives a detailed verbal description even before starting a task, to which TURVY replies "Show me what you want." The user performs a single example and asks TURVY to try the next. If TURVY makes a mistake, the user cries "Stop" and then tells it what to do. TURVY listens to the hint but says "Show me what you want," and the user performs the correction, repeating the verbal hint. If the hint does not help it form a consistent hypothesis, TURVY asks for features that distinguish this case from the predicted one. Initially it asks vaguely, "What's different about this example?" Sometimes the user is puzzled, so TURVY proposes a guess, which the user either accepts or corrects with another hint.

Quiet users work silently through the first example and go on to the next without signaling that it is done or inviting TURVY to take over. When TURVY detects repetition, it interrupts, "I've seen you do this before, can I try?" After some hesitation, the user consents, often sounding skeptical. When TURVY makes a mistake, the user cries "Stop," just like a talkative user, but then *demonstrates* a correction, and is reluctant to answer TURVY's questions about differences between this example and others. A quiet user will likely tell TURVY to skip a troublesome case rather than try explaining it.

Command set. Prior to the experiment, we predicted a set of 30 instructions that users might employ. By and large, all users "discovered" the same small subset of these commands. To control learning, they would tell TURVY to "watch what I do" or "ignore this." To control prediction, they would say "do the next one" or "do the rest." The actual wording varied little, and all users adopted standard terminology once they heard TURVY use it, as when it asks "Do the next one?" Subjects used far fewer forms of instruction for focusing attention than expected. They almost never volunteered vague hints like "I'm repeating actions," "this is similar to one before," or even "look here." On the other hand, they annotated examples in detail (at least, the talkative ones did) or else offered hints in response to TURVY's questions or guesses. For instance, in one task users demonstrated selecting a conference paper title, without mentioning that it ends before italic text. But when TURVY predicted that the next title "Inductive inference: theory and methods" ends at the colon before "theory", most users said something like "look for the colon before the italics" while pointing at the right colon.

TURVYTalk. We found that users learn to describe user-culture concepts like titles and author's names in terms of low-level, syntactic features—but only after hearing TURVY adopt such terminology. In a pilot experiment, run prior to the videotaped sessions, TURVY did not verbalize its actions, and without prior exposure to its language, users had no idea how to answer questions like "what's different about this case?" Apparently, users learn how to talk to TURVY the way they learn how to talk to other people, by mirroring its speech.[5] During debriefing in the main experiment, we asked users to characterize the language that TURVY understands. Although they had trouble answering so vague a question, when given specific examples nearly all showed they clearly understood that TURVY looks for and understands low-level syntactic features only.

Tasks and teaching difficulty. Programming by demonstration should make simple tasks easy to teach and complex tasks teachable. The range of difficulty in the experiment, from changing text formats to parsing and re-ordering lists of names, is considerable. The easy tasks were trivially taught by giving a demonstration. The hard tasks involved far more verbal description, and the level of user effort observed was higher. The single most difficult problem users had was explaining to TURVY that the name "Michalski R. S." was already in the desired format, *almost* — a comma is missing after the surname. In the post-session interviews, all subjects reported that they found TURVY easy to teach, especially once they realized that it learns incrementally and continuously, so that they need not anticipate all special cases. In fact, this was TURVY's most popular feature.

General observations. After their sessions, most subjects commented that they would have used less speech had TURVY been in another room linked by phone, rather than behind a screen in the same room. They said they had more confidence in TURVY's ability to understand speech because they knew it was human. On the other hand, some users seemed embarrassed and unsure what to say when they first spoke to TURVY, and spoke curtly as if they really believed it was a computer. Also, most referred to TURVY as "it" when talking to the facilitator during the sessions.

All experimental subjects said they would use TURVY if it were real. They liked the way it learns incrementally and they did not mind answering a few questions. They believed they would have difficulty describing the syntax of text entirely on their own. They liked the way TURVY describes actions while doing them the first time; they did not notice that it works silently on subsequent iterations. All were concerned about completeness, correctness and autonomy; they believed it would be foolhardy to leave TURVY unsupervised.

One pilot user rejected the very idea of an instructible agent and refused to work with TURVY. This person believed that a user would have to anticipate all cases of a pattern in advance, as when writing a program.

2.2 Lessons for intelligent agents

TURVY extends the notion of programming by demonstration that has been developed in other systems.[4] It watches demonstrations, but also invites the user to point at relevant objects and give verbal hints. It adopts a more general-purpose learning method than other systems, using domain knowledge to learn from one example, but finding similarities and differences over multiple examples and matching the user's hints with observed features to zero in on a generalization.

From this study, several points emerged that apply to intelligent agents in general.

- Users appreciate and exploit incremental learning; they are content to teach special cases as they arise, rather than anticipate them.
- Many users want to augment demonstrations with verbal hints, and find it easy to do so.
- A static description of what the agent has learned is not required: by giving concise verbal feedback when predicting actions, an agent gives users sufficient knowledge of its state to maintain their confidence.
- By using its input language in feedback, an agent helps users learn how to teach.
- A reliable speech-based interface may be better than menus for teaching an agent, since it reduces distraction from actions on data.
- To elicit a hint, it is better to propose a guess than ask the user for a suggestion.
- Both agent and user must be able to refer to past examples and instructions.

The manifest advantages of this style of interaction, and the general success of the TURVY study, led us to develop CIMA, an implementation of TURVY's learning mechanism. Most of its components, including a concept learner,[3] have appeared in the machine learning and artificial intelligence literature but have not previously been brought together in one system.

3 Implementation

CIMA is the learning component of an intelligent agent that interacts with users and application programs. Connected to a text editor within the Macintosh Common Lisp environment, it learns to search for textual patterns based on the user's selections. It accepts demonstrations from the user and treats them as "examples" of what is to be done. It also provides means for the user to interact and control the learning process. For example, CIMA allows the user to suggest features that might be germane to classification. Our experience with TURVY demonstrated that such "hints" are bound to ambiguous, incompletely specified, and sometimes even misleading. Therefore the system interprets them in light of (a) any domain knowledge that is available, and (b) the examples. To combat misleading hints, the system can compare descriptions generated by interpreting them more loosely or even ignoring them. Details of the learning algorithm are presented elsewhere.[8]

The best way to understand what CIMA does is to see it in action. Although the system was evaluated on the TURVY tasks described above, for variety's sake we use a different example here. Suppose the user has a text file of addresses and is creating an agent to retrieve and dial phone numbers. She wants to teach it to identify the phone numbers that have the local area code (617), and strip it off. Sample data appears in Figure 5.

The task is to teach the "local phone number" concept, whose positive examples are listed in Figure 5. Two different scenarios for teaching this concept follow (both assume that CIMA has not yet been taught the more general concept of "phone number"). The first uses examples only, whereas the second uses hints along with some domain knowledge.

3.1 Learning from examples

To give the first example, the user selects 243–6166 with the mouse and picks *I want this* from a popup menu. The example and its surrounding text are recorded, and the first rule, item (a) in Table 1, is proposed. When the user gives a second example, 220–7299, the rule is generalized to (b). CIMA correctly predicts the third example, 284–4707. When it predicts 255–6191, which is preceded by a nonlocal area code, the user rejects it by selecting *But not this* from the menu. CIMA now attempts to specialize the description to exclude this negative example. At present it is considering only features of the selected text: since no generalization covers all three positive examples yet excludes the negative one, it forms a rule with three special-cases, shown in (c). When forced to create new special-case rules, CIMA surmises that its current attribute language may be inadequate, and therefore widens its focus of attention. In this case, it checks the surrounding text for distinguishing features. The positive examples follow "617) "; the negative example does not. Using this information, it forms the single general rule shown in (d). The reason why the string "617) " is proposed rather than merely "7) ", which would discriminate equally well, is that by default, text is matched at the word level, where words are delimited by

Sample data for teaching

```
Me (617) 243–6166 home; (617) 220–7299 work; (617) 284–4707 fax
Cheri (403) 255–6191 new address 3618 – 9 St SW
Steve C office (415) 457–9138; fax (415) 457–8099
Moses (617) 937–1064 home; 339–8184 work
```

The positive examples

```
243–6166, 220–7229, 284–4707, 937–1064, 339–8184
```

Figure 5 The "local phone number" task

	Example	Class	Rule: "Searching forward, select text that..."
	Learning from examples only		
(a)	243–6166	+	MATCHES "243–6166"
(b)	220–7229	+	MATCHES Number(length 3)–Number(length 4)
(c)	255–6191	–	MATCHES "243–6166" or "220–7299" or "284–4707"
	… after change of focus to include features of neighboring text		
(d)			FOLLOWS "617)◊" and MATCHES Number(length 3)–Number(length 4)
	… after final positive example		
(e)	339–8184	+	FOLLOWS "617)◊" and MATCHES Number(length 3)–Number(length 4) or text that FOLLOWS ";◊" and MATCHES Number(length 3)–Number(length 4)
	Learning from examples and pointing at "(617)"		
(f)	243–6166	+	FOLLOWS "(617)◊" and MATCHES "243–6166"
(g)	220–7229	+	FOLLOWS "(617)◊" and MATCHES Number(length 3)–Number(length 4)
	Learning from examples and verbal hints		
(h)	243–6166	+	FOLLOWS ")◊" and MATCHES Number–Number
(i)	255–6191	–	FOLLOWS "617)◊" and MATCHES Number–Number
	Learning from examples and partial specification		
(j)	243–6166	+	FOLLOWS "(617)◊" and MATCHES Number–Number
	… after final "anomalous" positive example		
(k)	339–8184	+	FOLLOWS "(617)" and MATCHES Number–Number or text that FOLLOWS ";◊" and MATCHES Number–Number

Table 1 Four scenarios for teaching "local phone number"

white space, punctuation and special characters like "(". Characters within words can be matched if CIMA's focus of attention is directed toward them.

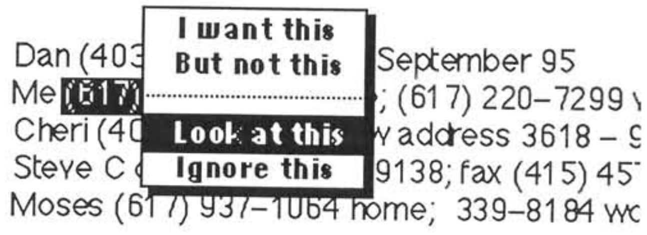
The new rule predicts the remaining positive examples, except for an anomalous one, 339–8184, which lacks an area code. When the user says *I want this*, the set of rules (e) is formed. To maximize the similarity between rules, a generalized pattern is adopted for this final phone number—even though it is the only example of the new rule.

3.2 Suggestions from the user

Now consider the same task taught by examples and hints. Realizing that the distinguishing feature of a local phone number is its area code, the user selects "(617)" and chooses *Look at this* from a popup menu when giving the first example. This directs CIMA to examine the text immediately preceding the example, focusing in particular on the string suggested by the user. Using this feature, CIMA forms the rule shown in item (f) of Table 1. After the second positive example, the phone number is generalized as shown in (g). This rule predicts the remaining examples other than the final, anomalous one, which is treated as before.

Rather than point at "(617)" while selecting the first example, the user could have given a verbal hint, such as *it follows my area code*. The phrase *it follows* suggests text either before or after the

(a) Popup menu for classifying examples and features



(b) Verbal hints, typed or spoken

"it follows my area code"
"any numbers OK

(c) Formal (partial) specification of a concept

```
relevance (MATCHES (Target, Number–Number),
     relevant, "local phone number", allExamples)

relevance (FOLLOWS (Target, "(617)"),
     relevant, "local phone number", allExamples)
```

Figure 6 Three separate interaction techniques for specifying relevance

example, with preference to the former; the phrase *area code* is not recognized. Using built-in knowledge that text delimiters such as punctuation and parentheses are salient, the system focuses on the parenthesis before the example. The learning algorithm settles on *text FOLLOWS)◊* as the relevant feature, since no other evidence counts against it. A second verbal hint, *any numbers OK*, which the user gives while selecting the phone number, causes CIMA to generalize the *MATCHES* feature, focusing on tokens of type Number and ignoring other properties such as string value and length. Thus, after one example and two hints, the system forms the rule shown in item (h) of Table 1. But this rule predicts a negative example, since the *FOLLOWS* pattern is too general. To eliminate the negative example, the *text FOLLOWS* feature is specialized to "617) " which is used in rule (i).

A programmer could partially specify the concept of "local phone number" by explicitly identifying the following features as being *relevant* to the learning process:

- MATCHES (Target, Number–Number)
- FOLLOWS (Target, "(617)")

This specification may appear complete, but it's not: because the system is learning to search for text, the concept to be learned must specify where the text begins and ends, as well as the direction in which to search. This latter feature is missing; but the system will add an appropriate *Search direction* qualifier ("Searching forward") when it forms rules from examples, as in Table 1 item (j). To cover the anomalous positive example, 339–8148, which has no area code, the system adds a second rule, (k), using the *MATCHES* feature suggested by the programmer and an alternative value of the suggested *FOLLOWS* feature.

3.3 A taxonomy of instructions

The various forms of instructions illustrated above for giving examples and hints can be abstracted to three general types. These instructions comprise the information needed to form a concept. If all their arguments were fully specified, the learning system would need no inferencing; in practice, however, user instructions omit or underspecify some arguments.

Classify (Example, {+ve, –ve}, Concept, Subset)

Relevancy (Feature, {relevant, irrelevant}, Concept, Subset)
Consistency (Rule, {correct, overgeneral, overspecific, incorrect}, Concept)

The first form *classifies* an example as positive or negative with respect to some concept, and may indicate that it belongs with some subset of its examples: this is the usual instruction (Subset omitted) given to inductive learning programs, and is illustrated in Figure 6: the menu commands *I want this* and *But not this,* shown in Figure 6a, classify the current example.

The *Relevancy* instruction states that an attribute (e.g. text before target) or value (e.g. text before target = "(617) ") is relevant or irrelevant to some subset of examples. This type of instruction is known to accelerate learning[6] because it reduces the dimensions of the search space for a concept (a combination of feature values). User hints and partial specifications, as in Figure 6b-c, fall under this category. Typically, a hint ambiguously describes the attribute or value; the learner must explore several interpretations, using other evidence such as statistical fit to examples to choose among them.

The *Consistency* instruction states whether a given rule is valid: it has been studied in systems that learn from an informant[1]. Although not illustrated in the example scenario, CIMA interprets two forms of this instruction — namely, when the user classifies the rule as correct or incorrect through a menu command popped up on the rules window (not illustrated). If correct, the system retains the rule, though it will warn the user if the rule covers a negative example. If incorrect, it experiments with classifying some of the rule's features as irrelevant (first asking the user to classify them), so that it can generate alternative descriptions.

3.4 Processing hints

Hints may be input through menus, speech or pointing. A single hint may generate several Relevancy instructions, and need not define all the arguments. For instance, suppose the user in the scenario above suggested *look at what's around this number*. The keyword *around* suggests the text before and after the target, without indicating a specific value. The keyword *number* suggests an attribute of the currently selected text—in this case, the target. On examining the actual text, CIMA forms the following interpretations:

Relevancy (FOLLOWS(Target), relevant, currentTask, thisExample)
Relevancy (PRECEDES(Target), relevant, currentTask, thisExample)
Relevancy (MATCHES(Target, Number–Number), relevant, currentTask, thisExample).

A hint in which the user points at "(617)" suggests two specific alternative feature values:

Relevancy (FOLLOWS(Target, (617)◊), relevant, currentTask, thisExample)
Relevancy (FOLLOWS(Target, (Number)◊), relevant, currentTask, thisExample).

CIMA generates these interpretations by retrieving domain knowledge indexed by the feature values (such as the fact that "617" is a number, and the general belief that parentheses are interesting). The learning algorithm ranks these initial interpretations against criteria such as statistical fit to examples and the utility of features in actions (e.g. a search action requires text delimiters and a scanning direction). Moreover, generalizations must be justified by the evidence. Given only a single example phone number, CIMA prefers the more specific interpretation FOLLOWS(Target, (617)◊).

Relevancy instructions can originate with other agents or domain knowledge modules. CIMA records the instruction's source and uses credibility ratings to select among conflicting interpretations. As a matter of courtesy, the user's suggestions are given priority, and the system always tries to use features that the user suggests and avoid ones that she rejects, though it advises the user when this causes the description to become inconsistent. As noted above, it also tries to generate alternative descriptions by discarding some information (it ignores the suggested feature value, while still considering that type of feature) or by ignoring the hint altogether. If the resulting ruleset is simpler or more accurate, CIMA advises the user of this alternative.

4 Evaluating the implementation

In a preliminary evaluation, CIMA was tested on its ability to learn search patterns for the classes of data—author surnames, publication dates, etc.—found in the TURVY tasks. The examples and instructions were taken directly from transcripts of user interactions. Users' verbal hints were coded as text strings, from which CIMA extracted key words and phrases. For instance, while teaching the concept of a primary author's surname, one user gave the hint *take the first word after the first full stop in each paragraph*. This was translated to four key phrases, *first in paragraph*, *target matches word*, *target follows period* and *all examples*. Pointing gestures were coded as selections of text combined with a *Look at this* command. When running CIMA on the traces, some manual intervention was required: in particular, CIMA sometimes predicted a different negative example than TURVY, in which case the researcher had to classify it himself. (He always classified it correctly as positive or negative, as did TURVY's users.)

Table 2 shows some search patterns that CIMA learned, based on the bibliographic data shown in Figure 6a, for the tasks of selecting (a) the start and (b) the end of paper titles, (c) the primary author's surname, and (d) all surnames. Overall, CIMA achieved 95% of TURVY's predictive accuracy, measured over the course of learning and performing each task. (Measuring performance on a test data set only ignores the importance of predicting as few negative examples as possible while learning from the user, since these errors can cause users considerable inconvenience and distress.) Its performance on some tasks varied due to the interpretation of hints. On occasion, users appeared to present hints with the wrong example, causing overspecialization of rules (TURVY rather too cleverly repaired this error). We concluded that CIMA satisfies our design requirements for the concept learning part of the system, and that the agent's user interface should be designed to ensure that users associate their hints with the appropriate examples.

5 CONCLUSION

One of the key problems in transferring task knowledge from users to agents is capturing users'

(a) Start of journal paper title:

Searching forward from start of paragraph,
Insertion point FOLLOWS ":◊" and PRECEDES CapitalWord

(b) End of journal paper title:

Searching forward from previous example, Insertion point PRECEDES "◊" ItalicText

(c) Primary author's surname:

Searching forward from start of paragraph (Note: this feature used in all four rules),
Selected text MATCHES CapitalWord and PRECEDES ":"
or Selected text MATCHES CapitalWord and PRECEDES ","
or Selected text MATCHES "Michalski"
or Selected text MATCHES LowercaseWord "◊" CapitalWord

(d) Any surname:

Searching forward (Note: this feature used in all seven rules),
Selected text MATCHES CapitalWord and PRECEDES ":◊" NonAlphanumericCharacter
or Selected text MATCHES CapitalWord and FOLLOWS CapitalWord ".◊" and PRECEDES ","
or Selected text MATCHES LowercaseWord "◊" CapitalWord and PRECEDES ":"
or Selected text MATCHES LowercaseWord "◊" CapitalWord and FOLLOWS CapitalWord ".◊"
or Selected text MATCHES CapitalWord(length 9) and FOLLOWS Linebreak
or Selected text MATCHES "Quinlan" or "Mitchell

Table 2 Data descriptions learned for some concepts from the user study

intentions from example actions. Often, the intent of an action resides in the choice of data on which to act, or in the results. By learning rules for selecting and modifying data, CIMA partially models the intent of user actions. Our experience with both TURVY and CIMA demonstrates that user suggestions—even ambiguous ones—improve learning from examples, provided that the learning system uses other sources of knowledge to generate interpretations of them and measure their credibility.

Designers of intelligent agents have tended to focus on technology, assuming that any intelligent agent will be easy for humans to deal with. This violates common sense—for example, in some phases of the TURVY study, we observed the distress that users experience when an agent gives too much feedback, or too little, or transgresses rules of cooperative discourse. Perhaps the most important lesson we have learned is the value of involving users in design. By testing and critiquing our design ideas, end users keep us focused on our objective: agents that learn how to help users in ways that make computer-based work more productive and enjoyable.

REFERENCES

1. D. Angluin (1988) "Queries and concept learning," in *Machine Learning* (2), pp. 319–342.
2. S. E. Brennan (1990) "Conversation as direct manipulation: an iconoclastic view," in B. Laurel (ed.) *The art of human-computer interface design*, pp. 383–404. Addison-Wesley. Menlo Park CA.
3. J. Cendrowska (1987) "PRISM: an algorithm for inducing modular rules," in *International Journal of Man-Machine Studies* (27), pp. 349–370.
4. A. Cypher (ed.) (1993) *Watch what I do: programming by demonstration.* MIT Press. Cambridge MA.
5. R. G. Leiser (1989) "Exploiting Convergence to Improve Natural Language Understanding," in *Interacting with Computers* 1 (3), pp. 284–298.
6. D. Haussler (1988) "Quantifying inductive bias: AI learning algorithms and Valiant's learning framework." *Artificial Intelligence 36*, pp. 177–221.
7. D. Maulsby, S. Greenberg, R. Mander. (1993) "Prototyping an intelligent agent through Wizard of Oz," in *Proc. InterCHI'93*, pp. 277–285. Amsterdam.
8. D. Maulsby (1994) "Instructible agents," PhD thesis. Department of Computer Science, University of Calgary.
9. D. C. Smith (1975) "Pygmalion: a creative programming environment," PhD thesis. Stanford University.

ACKNOWLEDGMENTS

This work has been supported by the Natural Sciences and Engineering Research Council of Canada and Apple Computer Inc. Many thanks to Craig Neville-Manning for his editing hints.