Learning agents: from user study to implementation

David Maulsby
Medicine Wheel Netware Design
430, 820 – 89 Av SW
Calgary T2V 4N9 Canada

tel. +01 (403) 252–2139
fax. +01 (403) 255–0274
maulsby@medicine-wheel.com

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

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

Abstract
This paper describes the design, implementation and evaluation of an agent that learns to automate repetitive tasks within the human-computer interface. In contrast to most Artificial Intelligence projects, the design centers on a user study, with a human-simulated agent used to discover the interactions that people find natural. This study shows the ways in which users instinctively communicate via "hints," or partially-specified, ambiguous, instructions. Hints may be communicated using speech, by pointing, or by selecting from menus. We develop a taxonomy of such instructions. The implementation demonstrates that computers can learn from examples and ambiguous hints.

Keywords
Intelligent agents, programming by demonstration, machine learning.

Learning agents: from user study to implementation

1 INTRODUCTION

Graphical, direct-manipulation user interfaces have evolved a world of ready information, in which computer use centers on viewing and editing, rather than on programming. Yet the need for end-user programming is more and more apparent. Professionals find themselves performing repetitive tasks they once delegated to clerks and secretaries. People with repetitive strain injuries need to reduce input through keyboard and mouse. Many users want to customize applications. Developers have responded with a barrage of new features, whose number and complexity create a usability problem.

End-user programming raises a hard question—how will users program? The "modern" user interface eschews formal notation and encourages concrete expressions of ideas.[10] Perhaps the best representation of a program, therefore, is an annotated example. Experiments have shown that users can teach a computer by demonstrating the desired behavior.[4] Teaching demands less planning, analysis and understanding of the computer's internal representation than programming, but it raises new problems of how the system, as a learning machine, will correlate, generalize and disambiguate instructions. Human pupils interact with their teacher to solve these problems; a computer needs assistance tailored to its more limited abilities, and will fail in its mission if it confuses or frustrates the user.[9]

With the aid of end-users, we have developed and tested interaction methods for teaching a computer agent. We conducted an experiment in which people train an agent, TURVY, to edit a bibliography.[7] TURVY is simulated by a researcher hidden behind a screen, as shown in Figure 1—interface designers call this a "Wizard of Oz" scenario. Users demonstrated tasks that range from trivial (replace underlining with italics) to taxing (put all authors' given names and initials after their surnames—difficult because the user must select many short strings of text, keep track of errant whitespace, and handle special case names like those containing "van" or "de"). Users invented their own teaching methods and spoken commands. To ground the experiment in reality, we set limits on TURVY's abilities, in particular on the types of instruction it understands, the features of text it observes, and the generalizations it can learn.

Armed with this experience, we then implemented an interactive machine learning system, CIMA. By keeping users in the loop and allowing them to influence the course of learning, CIMA acquires

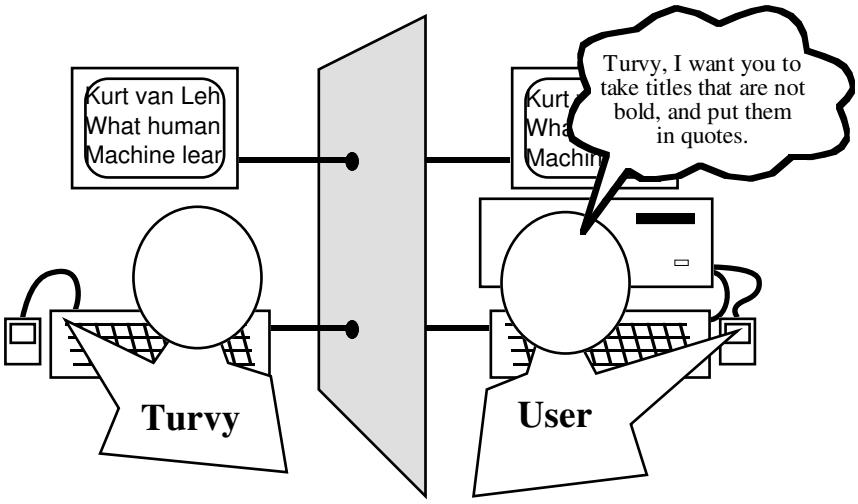

Figure 1 The experimental setup

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

concepts in a rich domain from few examples. We tested it on transcripts from the TURVY experiment and found that it learns at roughly the same level as TURVY. We also conducted experiments with using TURVY and CIMA to learn graphical layouts and manage files.

This paper describes both the user study and the implementation. We begin with an example of TURVY in action, describe the different styles that users adopt when interacting with it, and then draw out the lessons learned for the design of agents. Next we turn to CIMA, giving an example of various ways a task can be taught, depending on the user's teaching style. We discuss how the system manages interaction with the user. Finally, we summarize the initial evaluation study.

2 USER STUDY

The best way to understand what TURVY does is to see it in action. In Figure 2, the task is to make a heading for each bibliography item, showing the primary author's surname and the date of publication. With no background knowledge of bibliographies, names or dates, TURVY must learn these constructs in terms of text formatting and syntax. The relevant surname is normally the word before the first comma or colon, but sometimes includes a lower case "van", or precedes an "(ed)". 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, though sometimes the final period is missing. TURVY must learn special-case rules to handle these exceptions.

A transcript from the videotaped record of the experiment appears in Figure 3. Pam, the subject, has already completed two tasks. Prior to the experiment, a facilitator told her that TURVY is simulated by a human being, 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 predictions with verbal feedback, encouraging the user to annotate her examples with hints. When testing a particular hypothesis for the first time, TURVY offers a detailed description; as it gains confidence, it reduces feedback and increases execution speed.

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 3 Partial trace of Turvy learning task 3 from Pam

There are really three actors in this dialog: the subject, TURVY, and the researcher portraying TURVY. This affects the dialog in subtle ways. Although told 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 influence the amount and complexity of instructions users give. As an experimental "instrument," TURVY probes for more detailed instructions than might be appropriate in a real system. To elicit as much experimental data as possible, while keeping the session on schedule, the researcher may use his discretion to make TURVY more passive or proactive.

2.1 Observations and results

Four users participated in a pilot study, and seven in the formal experiment. The subjects included secretaries, an office manager, a multimedia artist, and psychology and computer science students. Their experience with computers ranged from occasional to professional usage. Each subject worked with Turvy for approximately one hour. We videotaped them and conducted interviews afterward. Although the video transcripts were analyzed quantitatively, we focused more on users' qualitative accounts of their experience.

Dialog styles. Users were evenly split between two dialog styles—talkative and quiet—we speculate that these styles match with extroverted and introverted personalities. Dialog style did not correlate with gender. 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 tells it what to do. TURVY says "Show me what you want," and the user performs the correction, repeating the verbal hint. TURVY might ask for features that distinguish this solution

from the predicted one. Sometimes the user is puzzled, so TURVY proposes a guess, which the user either accepts or corrects with another hint.

Quiet users work through the first example without giving hints 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. When TURVY makes a mistake, the user cries "Stop," but then *demonstrates* a correction, and is reluctant to answer TURVY's questions about distinguishing features. A quiet user will likely tell TURVY to skip a troublesome case rather than explain it.

Command set. By and large, all users "discovered" the same set of 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 fewer forms of instruction for focusing attention than expected. They almost never volunteered vague hints like "I'm repeating actions," or "look here," but instead mentioned specific features, as in "look for the colon before italics."

TURVYTalk. We found that users learn to describe concepts like titles and authors' names in terms of syntactic features—but only after hearing TURVY do so. 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]

Tasks and teaching difficulty. Programming by demonstration should make simple tasks easy to teach and complex tasks teachable. In the post-session interviews, all subjects reported that they found TURVY easy to teach, especially once they realized that it learns incrementally, 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 had more confidence in TURVY's ability to understand speech because they knew it was human. All experimental subjects said they would use TURVY if it were real. All were concerned about completeness, correctness and autonomy. They believed it would be foolhardy to leave TURVY unsupervised, but they concluded, based on their experience, that using TURVY would save them time and effort, freeing them to concentrate on more important aspects of writing.

One user refused to work with TURVY. This person rejected the very idea of an instructible agent, believing that he would have to anticipate all special cases in advance, as in writing a program.

2.2 Lessons for intelligent agents

A wide range of programming by demonstration techniques have been implemented in research projects,[4] ranging from the automatic generalization of user actions, through systems that allow users to control generalization explicitly. None, however, provide the flexibility of interaction and learning strategy that TURVY exhibits. TURVY can use background knowledge to learn from one example, or find similarities and differences over multiple examples. It watches demonstrations, but also invites the user to point at relevant objects and give verbal hints.

From this study, we learned several things that apply to intelligent agents in general, and which existing research systems do little to address.

- 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: concise verbal feedback, given while predicting actions, maintains users' confidence.
- By using its input language in feedback, an agent helps users learn how to teach.
- To elicit a hint, it is better to propose a guess than ask the user "what's relevant here?".

- 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.

3 IMPLEMENTATION

Connected to a text editor within the Macintosh Common Lisp environment, CIMA learns concepts that characterize textual structures. It learns from positive and negative examples: positive examples are text selections that the user actually edits, or predicted selections that the user accepts, and negative examples are predictions the user rejects. To give a "pointing hint," the user selects text and chooses "Look here" from a popup menu. Verbal hints are typed in a separate window. Our experience with TURVY demonstrated that hints are bound to be ambiguous, incompletely specified, and sometimes even misleading. Therefore CIMA interprets them in light of (a) domain knowledge about text editing, and (b) the examples.

The system learns classification rules whose antecedents are a set of alternative descriptions, each a conjunctions of terms, where each term is an attribute-value test. (This is a "disjunctive normal form" (DNF) description of the concept.) Rules are constructed by a modified version of the greedy covering algorithm[3] shown in Figure 4.

```
repeat until all positive examples are covered by some rule
    make a new empty rule R
    inner loop:  repeat until rule R covers no negative examples
        choose an attribute value A that
            maximizes coverage of positive examples not already covered by some rule,
            and minimizes coverage of negative examples
        add (conjoin) A to R
    end inner loop
    prune attributes from R that were rendered unnecessary by those added later
    add (disjoin) R to the set of rules
```

Figure 4 Greedy covering algorithm for inducing concept descriptions

This standard elementary machine learning technique is modified in two important ways. First, the inner loop continues adding features until a rule not only excludes all negative examples (as in the standard algorithm) but also meets "operationality criteria," ensuring that the rule will be useful for carrying out an action—be it to search for, generate or modify data. Second, features are selected not only for their statistical utility, but also for a combination of other criteria: whether the user has suggested them, whether they are used in other rules, whether they contribute to operationality, and their *a priori* salience based on domain knowledge. A sample of the system's background knowledge for matching, generalizing and assessing the relevance of features appears in Figure 5. The knowledge is encoded in Lisp as data (association lists), predicates and procedures, and includes facts about data types, generalization hierarchies, methods for matching and generalizing examples, default rankings for the salience (that is, "interestingness") of example attributes, and directed graphs that encode suggested changes in focus of attention. Details of these mechanisms, which are complex and not particularly elegant, can be found elsewhere.[8]

Built-in concepts (patterns)

Pattern-matching knowledge is declarative or procedural:

```
(declare_class 'EnclosureChar `('(' ')' '[' ']' '{' '}' '\" '\'"))

(defmethod isa ((theToken TextToken) (eql theClass :EnclosureChar))
        (find (text theToken) EnclosureChar))
```

Generalization hierarchies are used for matching or finding a generalization:

```
(declare_hierarchy 'TokenTypes
        (:Character (:Alphanumeric (:Alphabetic (:Lowercase :Capital)))
                (:NonAlphanumeric (:EnclosureChar :Punctuation ... etc.))))
```

Finding the common generalization of two text tokens by generalizing their attributes:

```
(defmethod generalize ((token1 TextToken) (token2 TextToken))
        (newToken       :tokenType (generalize (tokenType token1) (tokenType token2))
                        :text (generalize (text token1) (text token2)) ... etc.))
```

Types of features (attributes or relations)

A feature is defined by its type and methods for matching and generalizing examples:

```
(declare_feature 'TextMatches :printname "Matches" :type TextTokenSequence)
(defmethod match ((f1 TextMatches) (f2 TextMatches))
        (match (tokenSequence f1) (tokenSequence f2))
(defmethod generalize ((f1 TextMatches) (f2 TextMatches))
        (generalize (tokenSequence f1) (tokenSequence f2)))

(declare_feature 'TextBegins :printname "Begins" :type TextTokenSequence)
(defmethod match ((f1 TextBegins) (f2 TextBegins))
        (match (prefix (tokenSequence f1)) (prefix (tokenSequence f2)))
(defmethod generalize ((f1 TextBegins) (f2 TextBegins))
        (generalize (prefix (tokenSequence f1)) (prefix (tokenSequence f2)))
```

Salience of features relative to actions

The salience of a feature value (or pattern) is defined relative to actions and features in which it occurs; salience of a type of feature is defined relative to actions:

```
(declare_salience EnclosureChar :action 'SelectText
        :features `(TextMatches) :score Medium)

(declare_salience EnclosureChar :action 'SelectText
        :features `(TextBegins TextEnds TextFollows TextPrecedes) :score High)

(declare_salience TextMatches :action 'SelectText :score High)
```

Widening the focus of attention when the current feature set proves inadequate

The knowledge engineer can specify that features be added or removed when the learner fails to find a consistent or reasonably compact description:

```
(declare_focus 'add_features :when `(special_case_disjunct_created)
        :given_feature TextMatches :add_features `(TextBegins TextEnds)))

(declare_focus 'add_features :when `(special_case_disjunct_created)
        :given_feature TextBegins :add_features `(TextFollows)))

(declare_focus 'add_data :when `(repeated_bias_failure 3)
        :given_data TextToken :add_data `(Character))
```

Figure 5 Elements of CIMA's background knowledge as encoded in Lisp

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 training an agent to dial phone numbers, so that when she clicks with the mouse anywhere inside a phone number and presses the Command key, the agent copies it to a telephone dialer widget. Sample addresses appear in Figure 6. In a typical demonstration, the user tells the agent to start watching, then, while holding down the Command key, clicks between the 6 and 1 in 243–6166, selects the string 243–6166, copies it, pastes it into the dialer widget, and tells the agent she is done. The challenge for CIMA is to learn how to select these phone numbers, which it does not yet know how to parse. Moreover, it must learn to distinguish local numbers, so that it can strip off the area code (617).

Two different scenarios for teaching the "phone number" concept follow. The first uses examples only, the second uses hints. These correspond with the talkative and quiet user dialog styles.

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, (403) 255–6191, (415) 457–9138,
(415) 457–8099, 937–1064, 339–8184

Figure 6 The "phone number" task

3.1 Learning from examples

To give the first example phone number, the user selects 243–6166 with the mouse. When recording this action, CIMA notes the selection and its surrounding text. An operational description of this action must specify where the selection starts and ends, and so CIMA forms a text-selection rule, item (a) in Table 1. 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, a nonlocal number, the user corrects it by selecting (403) 255–6191, implicitly classifying 255–6191 as a negative example. Since no generalization covers all four positive examples yet excludes the negative, CIMA forms a rule with four special-cases, shown in (c). Being forced to create new special-case rules is symptomatic of an inadequate attribute language; CIMA therefore widens its focus of attention to include features of the surrounding text. Three of the positive examples follow "617)_"; the negative example does not. Using this information, it forms the two rules shown in (d). The string "617)_" is proposed rather than merely "7)_", which would discriminate equally well, because CIMA is matching text at the word level. It matches at the character level when its focus of attention is directed there.

When the user selects the next positive example, (415) 457–9138, the second rule in (d) is generalized to MATCHES (Number(length 3)) Number(length 3)–Number(length 4). The rules predict the remaining positive examples, except for an anomalous one, 339–8184, which lacks an area code. When the user selects this, the set of rules (e) is formed. CIMA is configured to maximize the similarity between rules by re-using features, hence a generalized pattern is adopted for this final phone number—even though it is the only example of the new rule.

	Example	Class	Rule: "Search backward from click location for 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	–	
	(403) 255–6191	+	MATCHES "243–6166" or "220–7299" or "284–4707" or "(403) 255–6191"
	After change of focus to include features of neighboring text:		
(d)			FOLLOWS "617)_" and MATCHES Number(length 3)–Number(length 4) or text that MATCHES "(403) 255–6191"
	After final positive example:		
(e)	339–8184	+	FOLLOWS "617)_" and MATCHES Number(length 3)–Number(length 4) or text that MATCHES (Number(length 3)) 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

Table 1 Three scenarios for teaching "phone number"

3.2 Suggestions from the user

Now consider the same concept taught by examples and hints. To point out the local area code, the user selects "(617)" before the first example, and chooses *Look at this* from a popup menu. This directs CIMA to focus on text preceding the selected phone number, and to construct a rule incorporating it, shown in item (f) of Table 1. After the second positive example, the rule is generalized as shown in (g). The other two rules are learned as before.

Rather than point at "(617)" while selecting the first example, the user could have given a verbal hint, *it follows my area code*. Looking in its thesaurus of feature words, CIMA finds the keyword *follows*, which suggests two features, FOLLOWS and PRECEDES, with preference given to the former. The system parses two lines of the address file around the example and picks out several features: the literal text, its tokenization, and the string ")_" just before the example. Built-in knowledge, that punctuation and parentheses are salient features, biases the learning algorithm to choose FOLLOWS ")_" as the relevant attribute. A second verbal hint, *any numbers*, which the user gives while selecting the phone number, causes CIMA to generalize MATCHES, 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 (h) in Table 1. But this rule predicts a negative example, since the FOLLOWS pattern is too general. To eliminate the negative example, CIMA specializes the FOLLOWS attribute value to "617)_" forming rule (i).

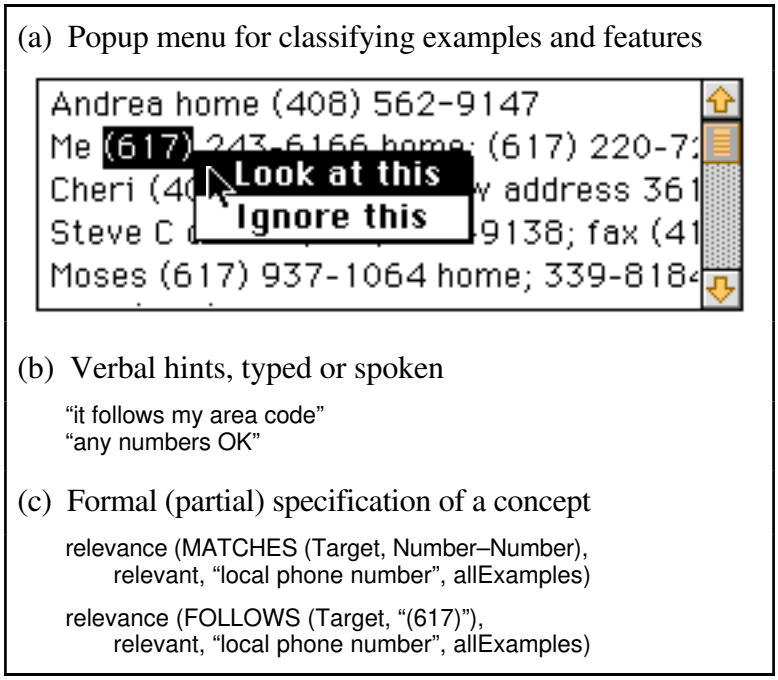

Figure 7 Three separate interaction techniques for specifying relevance

3.3 A taxonomy of instructions

The various forms of examples and hints can be abstracted to three types of instruction:

Classify (Example, {+ve, –ve}, Concept, Subset)
Relevancy (Feature, {relevant, irrelevant}, Concept, Subset)
Consistency (Rule, {correct, overgeneral, overspecific, incorrect}, Concept)

In practice users omit or underspecify some of the arguments to these instructions. 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. The user implicitly classifies example data by selecting or inputting it while demonstrating a task.

The *Relevancy* instruction states that an attribute (e.g. FOLLOWS) or value (e.g. FOLLOWS "(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. Figure 7 shows different ways of inputting relevancy instructions to CIMA: a popup menu for "pointing," verbal hints, and partial specifications. Typically, a hint describes the attribute or value ambiguously; the learner must therefore explore several interpretations.

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 one form of this instruction—namely, when the user classifies the rule as correct or incorrect through a menu command. If incorrect, the rule must contain incorrect or irrelevant features, and so CIMA asks the user for Relevancy instructions to guide it in generating a new rule.

4 Evaluating the implementation

CIMA has been tested on learning the text structures—surnames, publication dates, etc.—encountered in the TURVY tasks. The examples and instructions were taken directly from

transcripts of user interactions with TURVY. Pointing gestures were coded as selections of text, spoken hints as text strings. CIMA and TURVY use the same attribute-value language, but CIMA learns DNF rules only, whereas TURVY can disjoin attribute values and thus learn simpler, more general descriptions. For instance, it represents "primary author's surname" as follows:

Searching forward from start of paragraph (this feature used in both rules),
1) Selected text MATCHES CapitalWord or LowercaseWord "_" CapitalWord and PRECEDES ":" or ","
or 2) Selected text MATCHES "Michalski"

In contrast, CIMA learned this somewhat more complex-looking description:

Searching forward from start of paragraph (this feature used in all four rules),
1) Selected text MATCHES CapitalWord and PRECEDES ":"
or 2) Selected text MATCHES CapitalWord and PRECEDES ","
or 3) Selected text MATCHES "Michalski"
or 4) Selected text MATCHES LowercaseWord "_" CapitalWord

Despite such differences, CIMA achieved 95% of TURVY's predictive accuracy, measured over the course of both learning and performing tasks. (In normal use, learning and performance are inextricably interleaved.) CIMA's performance on some tasks fell short of TURVY's, due to its primitive interpretation of hints (CIMA merely spots keywords; TURVY parsed sentences). 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 always associate hints with an example, to reduce the set of plausible interpretations.

5 CONCLUSIONS

One of the key problems in transferring task knowledge from users to agents is to capture users' intentions. Often, the intent of an action is to select data which match a certain pattern, or modify data to match a pattern. In a "Wizard of Oz" user study, we discovered the forms of instruction that people naturally adopt when communicating such intentions to a computer agent of limited intelligence. We established the utility of ambiguous verbal and gestural "hints." Although one may claim in retrospect that this is obvious, machine learning systems to date have not exploited such hints. We showed the feasibility of teaching users how to teach by giving feedback that mirrors the desired instructional input. An unexpected result was the sharp division of users into two distinct dialog styles, one relying almost exclusively on examples, the other volunteering hints.

We defined a simple taxonomy of instruction types to account for the various observed forms of examples and hints. To demonstrate that hints can expedite learning from examples, and to establish that both dialog styles could be accommodated in a unified interface, we developed the CIMA concept learning program. CIMA uses three sources of information—examples, user hints and domain knowledge—to generate rule-like descriptions for classifying, finding, modifying or generating data. A preliminary evaluation indicates that CIMA performs up to a level that would provide useful assistance in practice. Our experience with both TURVY and CIMA demonstrates that even ambiguous hints improve learning from examples, provided the learning system uses other sources of knowledge to interpret them and measure their credibility.

The potential applications of agents that learn from their users are legion. Most interactive tasks involve a degree of repetition that can become intensely annoying in practice. Programming by demonstration offers a general solution to reduce boring, repetitive, work. We have explored one domain, repetitive text editing, and believe that the ideas apply to other applications such as spreadsheets, scheduling programs, and personal digital assistants.

Designers of intelligent agents have tended to focus on technology, assuming that any intelligent agent will be easy for humans to deal with. Perhaps the most important lesson we have learned is the value of involving users in design. By testing and critiquing our design ideas, they keep us focused on our objective: agents that make computer-based work more productive and enjoyable.

ACKNOWLEDGMENTS

This research has been supported by the Natural Sciences and Engineering Research Council of Canada and Apple Computer Inc. Many thanks to Craig Nevill-Manning and the anonymous reviewers, whose suggestions helped us improve the presentation of this work.

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. URL http://smi.stanford.edu/pub/Maulsby/Phd_Thesis/
9. B. Shneiderman, "Beyond intelligent machines: Just Do It!," *IEEE Software 10,* 1 (January 1993), 100–103.
10. D. C. Smith (1977) *Pygmalion: a creative programming environment*. Birkhäuser Verlag, Basel and Stuttgart.

AUTHOR BIOGRAPHIES

David Maulsby received his PhD in Computer Science at the University of Calgary, where his supervisor was co-author Witten. The two themes joined in his research are programming for non-programmers and understanding through examples. In prior work, Maulsby created Metamouse, one of the early efforts in programming by demonstration. During his doctoral studies, he conducted research at Apple Computer, where he worked with Allen Cypher and David C. Smith on a programming system they had designed for schoolchildren. He also assisted Cypher as a co-editor of the book "Watch What I Do: Programming by Demonstration" (MIT Press, 1993). After his PhD, he completed two years of postdoctoral studies, working with Henry Lieberman and Pattie Maes at the MIT Media Laboratory on programmable agents, and with Angel Puerta and Mark Musen in the Section on Medical Informatics at Stanford University, on the use of task models to guide the design and implementation of user interfaces. At present, he is engaged, both independently and as a consultant, in the design of products that will move his research on

programming by example out of the laboratory and into commercial applications for personal finance and computer-based learning.

Ian H. Witten received degrees in Mathematics from Cambridge University, Computer Science from the University of Calgary, and Electrical Engineering from Essex University, England. A Lecturer at Essex from 1970, he returned to Calgary in 1980 and in 1992 moved to New Zealand to take up a position as Professor of Computer Science at Waikato University. The underlying theme of his research is the exploitation of information about the past to expedite interaction in the future. He has worked in machine learning, which seeks ways to summarize, restructure, and generalize past experience; adaptive text compression, which uses information about past text to encode upcoming characters; and user modeling, or the general area of characterizing user behavior. He is director of two large research projects at Waikato: one on machine learning, the another in digital libraries with activities in the area of document compression, indexing, and retrieval. He has published around 200 refereed papers on machine learning, speech synthesis and signal processing, text compression, hypertext, and computer typography. He has written six books, the latest being "Managing Gigabytes: Compressing and Indexing Documents and Images" (Van Nostrand Reinhold, 1994, co-authored with A. Moffat and T. Bell).