Machine Learning in Programming by Demonstration: Lessons learned from CIMA

David Maulsby[1] and Ian H. Witten[2]

[1] Media Laboratory
Massachusetts Institute of Technology
Room E15–311
Cambridge MA 02139–4307 USA

tel. +01 (617) 253–9832
maulsby@media.mit.edu

[2] Department of Computer Science
University of Waikato
Private Bag 3105
Hamilton, New Zealand

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

Abstract

An agent that learns tasks from the user faces several problems: learning executable procedures; identifying relevant features of data in complex environments; filtering out irrelevant user actions; keeping the user in control without miring the user in the details of programming; and utilizing all forms of instruction the user might give including examples, ambiguous hints and partial specifications. This paper discusses the design and preliminary implementation of a system that aims toward these goals. Several lessons for system designers arise from this experience: the quality of user interaction is more important than the power of inference, though plausible inferences surely improve user interaction; involving end-users in all phases of design is critical to justifying design decisions; micro-theories of users, actions and data can take the place of domain theory in domains that have no formal theory; and, finally, a conventional concept (classification) learner can be adapted to utilize micro-theories to learn deterministic, operational action descriptions from examples, hints and partial specifications.

1. Introduction

Programming-by-demonstration (PBD) systems learn tasks by watching the user perform them. CIMA is an interactive learning system for modeling actions performed by the user of a computer system. CIMA is invoked when user actions are matched to find a common description of their pre- and postconditions. Although the system's interfaces to users and applications are still too rough-hewn to permit field trials, its performance on recorded dialogs between users and a simulated agent meets the design goals established prior to its implementation.

The contributions of this work lie in three areas:

- a design methodology for PBD systems;
- a framework for user actions in PBD;
- novel methods of interaction with the user.

These are discussed in separate sections below. First, however, it is necessary to convey the flavor of what it is like to use the CIMA system, and this is done in the following section.

2. Working with Cima

In its present embodiment, CIMA is connected to a text editor within the Macintosh Common Lisp environment. It can learn to search for textual patterns based on the user's selections. The system prints rules and feature descriptions in a listener window; the user can select them to classify them. The user can also type verbal hints into the listener. The learning system communicates with the text editor through an application interface.

Suppose that the user has a text file of addresses and is creating an agent to retrieve and dial phone numbers. She wants to teach the agent to identify the phone numbers that have the local area code (617), and strip it off. Sample data appears in part i of Figure 1, and the positive examples are listed in part ii. The scenarios that follow illustrate teaching the concept "local phone number" by examples and by using hints along with some domain knowledge. We assume that CIMA has not yet been taught the more general concept of phone number.

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 rule (a) in Figure 2.i is proposed. When the user gives a second example, 220–7299, the rule is generalized to (b). CIMA 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 focusing only on

```
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
```

i Sample data for teaching

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

ii The positive examples

Figure 1 — The "local phone number" task

a Rule formed after first example:
Searching forward,
Selected text MATCHES "243–6166"

b Rule generalized after second example:
Searching forward, Selected text MATCHES
Number(length 3)–Number(length 4)

c Ruleset formed after negative example 255–6191:
Searching forward,
Selected text MATCHES "243–6166"
or Selected text MATCHES "220–7299"
or Selected text MATCHES "284–4707"

d Rule formed after change of focus:
Searching forward, Selected text FOLLOWS "617)◊"
and MATCHES Number(length 3)–Number(length 4)

e Ruleset after final positive example 339–8184:
Searching forward, Selected text FOLLOWS "617)◊"
and MATCHES Number(length 3)–Number(length 4)
or
Searching forward, Selected text FOLLOWS ";◊"
and MATCHES Number(length 3)–Number(length 4)

i Series of data descriptions induced from examples

a Rule formed after first example and pointing at (617):
Searching forward, Selected text FOLLOWS "(617)◊"
and MATCHES "243–6166"

b Rule generalized after second example:
Searching forward, Selected text FOLLOWS "(617)◊"
and MATCHES Number(length 3)–Number(length 4)

ii Data descriptions induced from examples and pointing

a Rule formed after first example and verbal hints:
Searching forward, Selected text FOLLOWS ")◊"
and MATCHES Number–Number

b Rule specialized after negative example:
Searching forward, Selected text FOLLOWS "617)◊"
and MATCHES Number–Number

iii Data descriptions induced from examples and hints

a Rule formed after first example and partial specification:
Searching forward, Selected text FOLLOWS "(617)◊"
and MATCHES Number–Number

b Rule specialized after negative example:
Searching forward, Selected text FOLLOWS "(617)"
and MATCHES Number–Number
or
Selected text FOLLOWS ";◊"
and MATCHES Number–Number

iv Data descriptions induced from examples and partial specification

Figure 2 — Sample data and four scenarios for teaching "local phone number"

features of the selected text: since no generalization covers all three positive examples yet excludes the negative, it forms three special-case rules, 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 the 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 token level, tokens being words, punctuation and special characters. However, 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.

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 line (a) of Figure 2.ii. After the second positive example, the phone number is generalized as shown in (b). 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 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. Prior knowledge about the salience of features is encoded in two forms: tables that pair feature types or values with preference scores; and procedures that analyze the syntax of text to compute a score. In this case, Cima analyzes the syntax of the text to discover salient delimiter characters near the bounds of the user's selection. Thus 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 rule shown in line (a) of Figure 2.iii is formed. 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 (b).

A programmer could partially specify the concept of "local phone number" by stating that the following features are *relevant* to the learning process:

- MATCHES (Target, [Number–Number])

• FOLLOWS (Target, "(617)")

This specification is incomplete, because it ignores the direction of search. However, the system will add an appropriate *Search direction* qualifier when it forms rules from examples. Thus, after the first positive example the rule shown in entry (a) of Figure 2.iv is obtained. To cover the anomalous positive example, 339–8148, which has no area code, a second rule, shown in (b), is formed later, using the *MATCHES* feature value suggested by the programmer and an alternative value of the suggested *FOLLOWS* feature.

The application interface

In our experience, the most difficult aspect of implementing a PBD system is not the learning algorithm but rather the interface between it and the application program. The PBD system must be able to access relevant features of data, and compute other features that the application does not represent directly (in particular, relations among example feature values). More problematic is the recording of user actions. As Kosbie (1994) points out, the development of PBD tools has been hampered because most operating systems enable the recording of only low-level events such as mouse clicks and key presses. He suggests that representing events in hierarchies will enable PBD systems to analyze actions at an appropriate level of abstraction. We have recently implemented a preliminary version of the CIMACONNECTION, an application interface that defines protocols for recording actions and data.

At its simplest, the CIMACONNECTION defines the syntax for describing example actions and data. Features are transmitted as tuples of the form (featureName {:of objectsList} {:type featureType} {:aspect aspect} := value). For example, (location :of target :type graphicLocation :aspect result := (27 184)), means that, as a result of the action, the target object's location is now at coordinates (27, 184). In most cases, the arguments :of, :type and :aspect may be omitted, but they are provided to improve the expressiveness of what is essentially an attribute-value notation: ":of" permits the description of relations among objects; ":type" tells CIMA which functions to use to interpret the value; and ":aspect" allows a given feature to appear more than once, in different aspects of an example — for instance, an object's location can be specified both before and after a "drag-graphic" action, by putting a "location" feature under the "prior" and "result" aspects of that action.

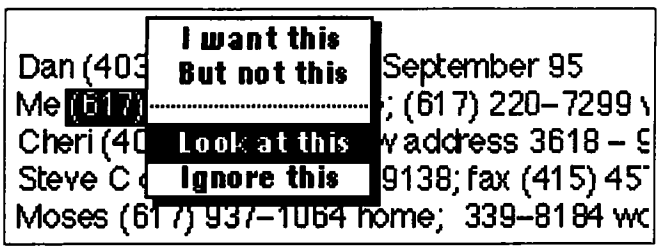
Beyond specifying the syntax for describing recorded actions, the CIMACONNECTION defines a protocol, specific to a given application program, for which features to transmit automatically, and which to transmit only when CIMA requests them: for instance, a drawing program might automatically send all properties of the current target object, but transmit information about neighboring objects only when CIMA requests it. The feature recording protocol eliminates features that are bound to be irrelevant to a given action (e.g. an object's color is irrelevant to the result of a "drag-graphic" action) and focuses attention on salient features, without eliminating the possibility of inspecting other features and objects. To support this protocol, the CIMACONNECTION provides "callbacks" by which CIMA can query an application for the features of a given object, or for all objects satisfying a given predicate (for instance, all objects currently on a drawing program's display list). But most applications are unlikely to be able to evaluate such predicates themselves, so implementing the CIMACONNECTION for a given application may require writing code to access the application's data and evaluate the predicates.

Still more problematic is controlling the application to execute predicted actions. Operating-system level support for this is still only rudimentary: on the Apple Macintosh, for instance, most applications support only the minimal AppleEvents required to execute commands from AppleScript. But even if systems support full scripting, it is all too easy to generalize actions in such a way that they could not be executed by a given application: for instance, CIMA could form a syntactic pattern that a word processor's search and replace facility cannot recognize. Since the ability of an agent to generalize data adds considerable value to an application, it should be preserved and therefore the PBD system must provide the extra functionality, in the form of pattern matchers and constraint solvers. The CIMACONNECTION defines a callback protocol so that applications can ask the PBD system to compute action parameters. Another problem, brought up by Cypher (1993) is forming "operational" descriptions. A generalization that is valid for recognizing examples of an action is not necessarily valid for executing them. For instance, "drag graphic such that its x location = previous x + 10" expresses the generalization that objects are being moved to the right by 10 pixels, but it cannot be executed by the application unless a y location is also specified. Section 4 of this paper explains how CIMA addresses this problem.

The user interface

In other research (see Maulsby 1992 and Section 3 below) we have built and experimented with the individual components of the user interface for an instructible task agent. The CIMA learning algorithm was designed to work with such interfaces, since it can process examples, verbal and graphical hints, and partial specifications given via menus or a formal language. Although the exact form of the user interface depends on the application, there is a small common set of commands to classify examples, features and rules (see Section 5) and to direct the agent to execute its predictions.

At present, CIMA has two application and user interfaces. One is a generic recorder of actions described in an English-like dialect of Lisp. The other, illustrated in Section 2, is a generalized search and replace facility in an emacs-like text editor. To classify examples and features of items that the user wants to find, the user pops up the menu shown in Figure 3.i. The first two menu choices classify an example as positive or negative, the latter two classify a feature as relevant or irrelevant. The user can select a range of text and classify it either as a whole example or as a feature (a part of an example). In the figure, the user is classifying the feature "(617)" as relevant to an example phone number. To communicate verbally, the user types the hint or speaks it into a microphone; examples are shown in Figure 3.ii. The input is free text. No attempt is made to formally parse it: instead, keywords are identified and processed as described in



i Popup menu for classifying examples and features

"it follows my area code" "any numbers OK"

ii Verbal hints, typed or spoken

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

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

iii Formal (partial) specification of a concept

Figure 3 — Interaction techniques

Section 5. Finally, formal concept specifications are communicated by typing them in the form of predicates to the listener window, as illustrated in Figure 3.iii. These two examples show features (*MATCHES* and *FOLLOWS* features, respectively) being classified as relevant to a concept called "local phone number."

3. Design methodology

The first contribution of the CIMA project concerns the way in which the system was conceived. The feasibility of the interaction protocol was established in a "Wizard of Oz" user study in which a researcher simulated an instructible agent called TURVY (Maulsby *et al.*, 1993). This exercise established the feasibility of the general approach and the necessity of utilizing ambiguous hints. The data gathered influenced the choice and weighting of heuristics. This methodology also affords an opportunity to assess the CIMA's performance on real user interactions even before it is ready for field testing.

The TURVY study

TURVY is an instructible agent that behaves in some ways like a human apprentice, yet has primitive background knowledge and limited language understanding. In fact, TURVY was a researcher hidden behind a screen. Our users rapidly found simple effective ways of teaching through demonstration and verbal hints, while TURVY found ways to elicit them.

Figure 4 shows an sample task: given a bibliography file, make a heading with the author's name and date. This task is one of the most difficult, since it involves parsing lists of people's names, which may include initials, baronial prefixes like "van", and special annotations like "(ed.)". TURVY must learn to find the primary author's surname—the word before the first comma or colon. It may include a lower case word (like "van"). The date is the last two digits before the period at the end of the paragraph. In some cases the final period is missing.

TURVY extends the notion of programming by demonstration. It watches demonstrations, but also invites the user to point at relevant objects and give verbal hints. TURVY also 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. Although we did not work out all the details of the learning system

i Original input

John H. Andreae, Bruce A. MacDonald: Expert control for a robot body: *Journal IEEE Systems, Man & Cybernetics:* July 1990.

Ray Bareiss: **Exemplar-based knowledge acquisition:** Academic Press: San Diego CA:1989

D. Angluin, C. H. Smith: Inductive inference: theory and methods: *Computing Surveys 3 (15),* pp. 237-269: September 1983.

Michalski R. S., J. G. Carbonell, T. M. Mitchell (eds): **Machine Learning II:** Tioga. Palo Alto CA. 1986

Kurt van Lehn: "Discovering problem solving strategies: Proc. Machine Learning 7th Int'l Workshop, pp. 215–217: 1989.

ii Reformatted version

[Andreae 77]
John H. Andreae, Bruce A. MacDonald: Expert control for a robot body: *Journal IEEE Systems, Man & Cybernetics:* July 1990.

[Bareiss 89]
Ray Bareiss: **Exemplar-based knowledge acquisition:** Academic Press: San Diego CA:1989

[Angluin 83]
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

[van Lehn]
Kurt van Lehn: "Discovering problem solving strategies: Proc. Machine Learning 7th Int'l Workshop, pp. 215–217: 1989.

Figure 4 — A TURVY task: Make a heading with author's name and date

```
makeRules (Concept, Features, Egs, Criteria, Heuristics)
  repeat until all positive examples in Egs are covered:
    add makeARule (Concept, Features, Egs, Criteria,
  Heuristics) to Concept's definition
  return new Concept definition

makeARule (Concept, Features, Egs, Criteria,
  Heuristics)
  create new empty Rule
  repeat until Rule meets Utility Criteria and Instructional
    Criteria, or until all Features have been tried:
    add featureWithHighestExpectedUtility (Features,
      Heuristics, Concept, Egs) to Rule
    delete examples in Egs no longer covered by Rule
    remove Criteria already satisfied, re-order preferences
  simplify (Rule, Concept, Features, Egs, Criteria,
    Heuristics), and return simplified Rule

featureWithHighestExpectedUtility (Features,
  Heuristics, Concept, Egs)
  set Candidates to Features
  repeat for each SelectionHeuristic in Heuristics
    until only one Candidate remains:
    set Candidates to FeaturesScoringHighest
      (SelectionHeuristic, Features, Concept, Egs)
  return first feature in Candidates
```

Figure 5 — Algorithm for composing DNF data description

before testing TURVY, most of its components had already appeared in the machine learning literature, so we were confident that it could be implemented.

Observations of users in the TURVY validated our design, and led to further refinements. Later on, data from the experiment were used as a benchmark in testing CIMA.

4. Framework for user actions

The second contribution relates to the conceptual level at which machine learning is applied. The standard "classification" paradigm used in machine learning is too low a level for PBD, not so much because classifications are inappropriate—there are indeed many positive and many negative examples generated during the course of a typical interaction with a PBD system—but because viewing input as a stream of examples to be classified is a very low-level way of looking at the learning problem. CIMA's learning algorithm, shown in Figure 5, extends a greedy DNF concept learner, Prism (Cendrowska 1987), by requiring that the learned description not only classify correctly but also specify all features of data required for a given type of action (the "Utility Criteria"), and include all features suggested by the user (the "Instructional Criteria"). The next two sections explain how the system makes use of these utility and instructional criteria.

To model tasks, an agent needs to learn about data, actions, and when to act. *Data descriptions* (Halbert, 1993) specify criteria for selecting objects, and the results of actions. Conventional machine learning algorithms learn to classify examples. But agents *do* things with data, and to be useful, data descriptions may require features in addition to those needed for classification. This is one reason why rule-learning algorithms are rarely found in interface agents.

We propose a set of utility criteria that parameterize a concept learner according to the types of action and data presented to it. Figure 6 illustrates four types of action: classify data; find data; generate new data; and modify properties. Utility criteria ensure that a data description determines the necessary action parameters. The learner should also prefer features with high utility. Together, utility criteria and preferences comprise general domain knowledge that can greatly improve learning efficiency.

Classify actions have a single utility criterion: to discriminate between positive and negative examples. Features with the most discriminating power are therefore strongly preferred. This is the criterion tested by CIMA's ancestor Prism and nearly all other concept learning algorithms. For instance, suppose the user wants an agent to store email messages from someone in the folder "Mail from pattie," as shown at the top of Figure 6. The data description *sender's id begins "pattie"* tells it which messages to select — those from Pattie, regardless of her current workstation. The data description *folder named "Mail from <first word of sender's id>"* tells it where to put them. These two data descriptions will be learnable from the user's "classify" actions on the *sender's id* and *folder name* respectively.

Find adds a second criterion: the description must delimit

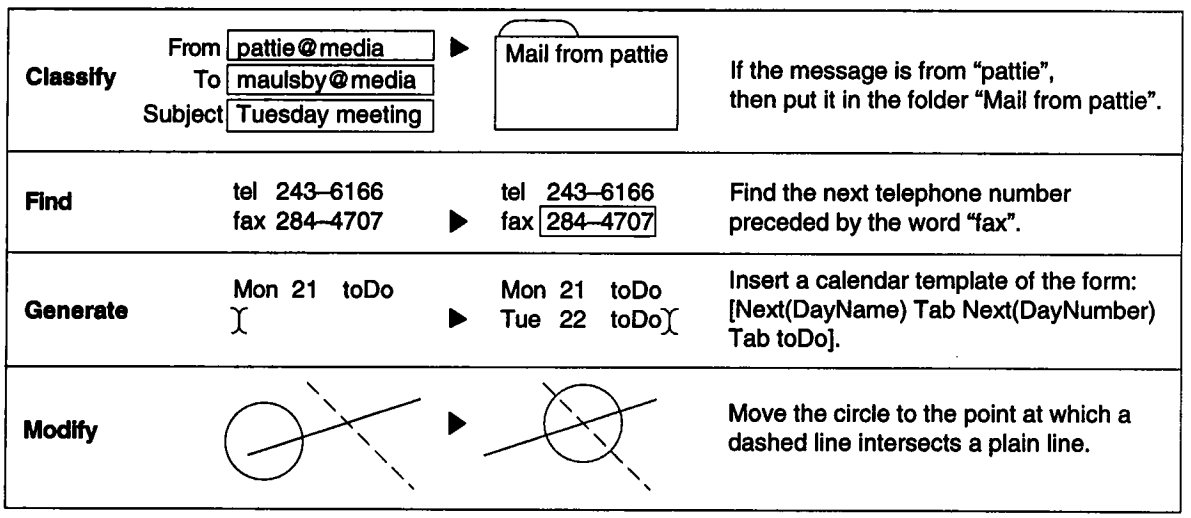


Figure 6 — General types of action on data

objects, and in some domains state the direction of search. Thus a text search pattern specifies where the string begins and ends, and whether to scan forward or backward. Features that describe more delimiters or constraints are preferred. Although the rule *FOLLOWS the string "fax "* might distinguish fax numbers from others, it only partially specifies a search pattern, since it indicates where the fax number begins but not where it ends; CIMA adds *MATCHES [Number–Number]* to complete the pattern.

Generate introduces a third criterion: the description should specify all features of a new object. If generating a graphic, the description must specify size, shape, color, etc.; for text, it must specify the actual string. Though *user input* is a valid feature value (i.e., the data may need to be input by the user), the system strongly prefers automatic value "generators"—constants, such as *"toDo"*, or functions, such as *Next(DayName)*, which return a specific value of the feature for each example object the system creates when performing the task.

Modify stipulates two criteria: the description should discriminate between positive and negative examples, and it should generate the property's new value. Features that determine the new value are preferred to those that merely constrain it. The graphics example in Figure 6 shows a conjunction of features that together determine a property value: two relations, *touch(Circle.center, Line1)* and *touch(Circle.center, Line2)*, establish the circle's new (x,y) location. By itself, each intersection leaves one degree of freedom on the circle's location. The utility criteria for setting an object's location assume that the goal is to remove all degrees of freedom if possible. Hence, features that remove both degrees of freedom, e.g. *touch(Circle.center, Line1.midpoint)*, are preferred over features that remove one degree of freedom. CIMA continues adding *touch(Circle.center, Line)* features until zero degrees of freedom remain. If the user rejects an example in which the circle touches two solid lines, CIMA adds a third feature—that one of the lines be dashed—to meet the classification criterion. Note that while the learner uses some knowledge about spatial relations—their number of degrees of freedom—it does not explicitly reason about geometry to derive a minimal set of constraints.

5. Interaction with the user

The third, and perhaps the major, contribution is in novel methods of interacting with users. CIMA accepts demonstrations from the user and treats them as "examples" of what is to be done. Moreover, it provides further means for the user to interact and control the learning process. A few learning systems, such as Clint-Cia (de Raedt 1992), can suggest features for the user to classify. CIMA allows the user to suggest features. 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 formed from interpreting them more loosely or even ignoring them.

Users can give three types of instruction:

- classifyExample (Example, Class, Concept)
- classifyFeature (Feature, Class, Concept, Disjunct)
- classifyRule (Rule, Class, Concept)

The first classifies an example as positive or negative with respect to some concept: this is the usual instruction given to supervised concept learners. For example, in Figure 2.i, all instructions are of this type. The *I want this* and *But not this* menu items in Figure 3.i communicate information of this kind.

The second instruction, classifyFeature, is known to speed learning (Haussler, 1988), but has received less attention in machine learning research, perhaps because it sidesteps the fundamental learning problem, which is to find relevant features of examples. Formally, the classifyFeature 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 kind of information is supplied by interaction (a) in Figure 2.ii and 2.iii. In the first case it is provided by a deictic hint; in the second by a verbal hint.

The Class argument for all classifyFeature instructions is either *relevant* or *irrelevant*. The Concept argument identifies the concept that is currently being taught (e.g. "local phone number"). In CIMA, concepts are represented as sets of disjuncts, and the role of the fourth argument is to distinguish which disjunct is involved. This information may be conveyed by identifying a rule, a set of examples, or a particular example.

Hints may be input through menus, speech or pointing. A hint may map to several classifyFeature instructions, and need not define all the arguments. For instance, suppose the user in the scenario in Section 2 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:

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

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

classifyFeature (FOLLOWS(Target, (617)◊), relevant, currentTask, thisExample)

classifyFeature (FOLLOWS(Target, (Number)◊), relevant, currentTask, thisExample)

CIMA generates these interpretations by applying domain knowledge to the data involved in the user's action. For verbal hints, it extracts key phrases and searches a thesaurus for corresponding attributes and values, generating one interpretation for each meaning (as in "around"). For pointing gestures, it finds features relating the selected data to the target example, and generates both specific and generalized values. CIMA relies on the learning algorithm to test these initial interpretations on other criteria, such as statistical fit to examples, to choose the best one. Thus, given only a single example phone number, CIMA prefers the more spe-

cific interpretation FOLLOWS(Target, (617)◊).

ClassifyFeature instructions can originate with other agents or modules of domain knowledge. 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 in a hint (the suggested value) and by ignoring it altogether. If the resulting ruleset is simpler or more accurate, CIMA advises the user of this alternative.

The third type of instruction states whether a given rule is valid: it has been studied in systems that learn from an informant (which may take the form of a human teacher). This was not illustrated in the example scenario. A rule may be incorrect in several ways (cf. Rajamoney & DeJong, 1987); CIMA may form rules that are too general, too specific, or which overlap only part of the positive and negative example sets. In any case, an incorrect rule contains some features that are irrelevant or have the wrong value. When the user states that a rule is incorrect, CIMA asks the user whether s/he can tell it which features are irrelevant or incorrect, in effect generating classifyFeature instructions. The system then creates a new rule. When the user states that a rule is correct, CIMA retains it without modification as further examples arrive, though it does notify the user if the rule covers negative examples or if it finds another more general rule that makes this one unnecessary. The user may reclassify a rule at any time, or modify it by stating that features are relevant or irrelevant to it.

Conclusion

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

Developers of PBD systems have encountered two serious limitations in standard machine learning algorithms; they do not generate operational descriptions, and they do not interact well with users. CIMA addresses the first problem by enforcing operationality criteria. It supports interactive teaching by incorporating the user's hints as additional knowledge sources. CIMA does not solve the interaction problem, however, because it does not specify the rules of discourse between an agent and its users.

Designers of intelligent agents have tended to focus on technology, assuming that any intelligent agent will be easy for humans to deal with. This goes against common sense. Indeed, 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 violates 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 so that computer-based work is more productive and enjoyable.

References

J. Cendrowska (1987) "PRISM: an algorithm for inducing modular rules." *International Journal of Man-Machine Studies 27*, pp. 349–370.

A. Cypher (ed.). (1993) *Watch what I do: programming by demonstration.* MIT Press. Cambridge MA.

A. Cypher (1993) "Eager: programming repetitive tasks by demonstration," in Cypher (ed., 1993), pp. 205–217. MIT Press. Cambridge MA.

L. de Raedt, M. Bruynooghe (1992) "Interactive concept-learning and constructive induction by analogy." *Machine Learning (8) 2*, pp. 107–150.

D. C. Halbert (1993) "SmallStar: programming by demonstration in the desktop metaphor," in Cypher (ed., 1993), pp. 103–123.

D. Haussler (1988) "Quantifying inductive bias: AI learning algorithms and Valiant's learning framework." *Artificial Intelligence 36*, pp. 177–221. 1988.

D. S. Kosbie, B. A. Myers (1994) "Extending programming by demonstration with hierarchical event histories," in *Proc. 1994 East-West International Conference on Human-Computer Interaction*, pp. St. Petersburg, Russia.

D. Maulsby (1992) "Prototyping an instructible interface: Moctec," in *Proc. ACM SIGCHI'92*, pp. 153–154. Monterey CA.

D. Maulsby, S. Greenberg, R. Mander (1993) "Prototyping an intelligent agent through Wizard of Oz," in *Proc. InterCHI'93*, pp. 277–285. Amsterdam.

S. Rajamoney, G.F. DeJong (1987) "The Classification, Detection and Handling of Imperfect Theory Problems," Proc. 10th International Joint Conference on Artificial Intelligence (IJCAI-87), pp. 205–207. Morgan Kaufmann, Los Altos, CA.