# Cima: an interactive concept learning system for end-user applications

David Maulsby

Section on Medical Informatics
Stanford University
Room MSOB x215
Stanford, CA 94305-5479 USA

tel. +01 (415) 725–1672
fax. +01 (415) 725–7944
maulsby@smi.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

**Running head** Interactive concept learning

**Contact author** 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

Personalizable software agents will learn new tasks from their users. In many cases the most appropriate way for users to teach is to demonstrate examples. Learning complex concepts from examples alone is hard, but agents can exploit other forms of instruction that users might give, ranging from yes/no responses to ambiguous, incomplete hints. Agents can also exploit background knowledge customized for applications such as drawing, word processing and form-filling.

The Cima system learns generalized rules for classifying, generating, and modifying data, given examples, hints, and background knowledge. It copes with the ambiguity of user instructions by combining evidence from these sources. A dynamic bias manager generates candidate features (attribute values, functions or relations) from which the learning algorithm selects relevant ones and forms appropriate rules. When tested on dialogues observed in a prior user study on a simulated interface agent, the system achieved 95% of the learning efficiency observed in that study.

## 1 Introduction

A truly personalizable software agent is one that learns new tasks from the user. For the interaction to be comfortable and reliable, the agent must facilitate rich communication with both user and task environment. Its learning mechanism should minimize the number of examples needed—in particular negative examples, which users perceive as time-wasting errors. To take advantage of users' knowledge and keep them in control, the learner could accept additional instructions: partial specifications or "hints." It could also exploit application-specific domain knowledge. Moreover, it should learn not merely how to distinguish positive from negative examples, but to describe those aspects of data relevant to the actions it is taught. Finally, since the set of potentially relevant features is enormous, even in simple situations, the learner must be able to select *and change* its focus of attention. The user might direct the bias by suggesting relevant features of the data. In turn, the agent should assess the user's hints and select among alternative interpretations by ranking them according to their plausibility and statistical utility.

This paper describes Cima (pronounced Chee-ma), an interactive learning algorithm for modeling the data selected and modified by an agent as it performs a task. Part of a programming-by-demonstration system, which learns tasks by watching the user perform them, Cima is invoked when actions are matched, to find a common description of their "operands." The algorithm is based on a standard DNF concept learner, Prism (Cendrowska, 1987). Prism finds a set of rules covering all and only positive examples. Taking a set of classified examples as input, it forms rules by adding "features" (attribute-value predicates) until the rule covers only positive examples. In contrast, Cima takes examples, task knowledge and instructions as input, and adds features to a rule until it meets stated *utility* and *instructional* criteria. Utility criteria ensure that a rule includes features required for a given type of action (classify, find, generate or modify data). Instructional criteria ensure that a rule includes features that the user suggests and avoids ones that the user rejects. To select features, Cima augments Prism's probabilistic coverage measure with a set of "justification" heuristics, including beliefs based on user hints or prior task knowledge.

The feasibility of utilizing ambiguous hints was established in a "Wizard of Oz" user study in which a researcher simulated an instructible agent called Turvy (Maulsby *et al.,* 1993). The data gathered in this study influenced the choice and weighting of heuristics. It also affords an opportunity to assess Cima's performance on real user interactions even before the system is ready for field testing.

## 2 Data descriptions and utility 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. For instance, suppose the user wants an agent to store email messages from Pattie Maes in the folder "Mail from pattie," as shown at the top of Figure 1. 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.

Conventional machine learning algorithms learn to classify examples. But agents *do* things with data, and to be useful, descriptions may require features in addition to those needed for classification. This is one reason why classification learning methods are rarely used in interface agents. Cima embodies a model for testing whether a description determines the necessary action parameters, as well as correctly classifying examples (c.f. the "operationality" tests used in the pioneering Eager system for programming by demonstration; Cypher, 1993b). Figure 1 illustrates four types of action: classify (sort) data; find data; generate new data; and modify properties. Cima ensures that each rule meets the utility criteria for the given type of action, preferring features that contribute most toward satisfying them.

*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 Prism and nearly all other learning algorithms.

*Find* adds a second criterion: the description must delimit 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 parts or constraints are preferred. For instance, the pattern *follows the string "fax "* is incomplete;

Cima adds *matches Number–Number* (or, if it has the appropriate concept, *is a phoneNumber*) to specify where the string ends. It prefers these over *precedes a blank* because they specify both endpoints of the text.

*Generate* adds 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, the system strongly prefers value "generators"—constants, such as *"toDo"*, or functions, such as *Next(DayName)*.

*Modify* is a special case of Generate, requiring the description to discriminate between positive and negative examples, and to determine (as far as possible) the property's new value. Again, deterministic or strongly constraining values are preferred. The graphics example in Figure 1 shows a conjunction of features determining a property value: two relations of the form *touch(Circle.center, Line)* together determine the circle's new location. By itself, each one removes a degree of freedom on the circle's center. The utility criteria for *set-2D-object-part-location* state that a location has two potential degrees of freedom ($x$,$y$). Features that remove both degrees of freedom, e.g. *touch(Circle.center,Line.midpoint),* are most strongly preferred, and after them, features that remove one degree of freedom. Cima continues adding *touch(Circle.center,Line)* features until zero degrees 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 is dashed—to meet the classification criterion.

## 3 Interacting with the user

An interactive learner elicits instructions from the user, processes them, and provides feedback. The feedback should help the user understand the learner's state well enough to formulate appropriate further instructions. Although elicitation and feedback are vital to the success of interactive agents, they lie beyond the scope of this paper (see Maulsby, 1994). Here we focus on the instructions that the learner can interpret, and show how they are processed.

*Instruction types*

Cima supports three types of instruction:

classifyExample (Example, Class, Concept)
classifyRule (Rule, Class, Concept)
classifyFeature (Feature, Class, Concept, Subset)

In practice, users omit or underspecify some of the arguments to these instructions. The first instruction *classifies* an example as positive or negative with respect to some concept: this is the usual instruction given to inductive learning programs. The user implicitly classifies example data by selecting or inputting it while demonstrating a task.

The second states whether a given rule is valid: it has been studied in systems that learn from an informant (Angluin, 1988). 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 classifyFeature instructions to guide it in generating a new rule.

The third instruction, classifyFeature, is more unusual. Formally, an attribute (e.g. *color*) or value (e.g. *color(red)*) is classified as relevant or irrelevant to some subset of examples—or, equivalently, some rule in the concept description. Henceforth we call this rule the "disjunct" to which the classifyFeature instruction applies. An instruction given by the user may suggest either a feature type or a specific value, and it may include ambiguous language that suggests several types or values. Cima handles ambiguous, incomplete, informal instructions, which we call "hints," by generating multiple interpretations, that is, multiple classifyFeature instructions. Although Cima can assign confidence scores to interpretations and thus rank them, it relies more on using multiple sources of information to decide which interpretation is best (see Section 5).

*Interpreting hints*

Hints may be verbal or gestural (pointing at objects to indicate whether they are relevant). For example, in the *sort mail* task at the top of Figure 1, the user might point at the substring *pattie* in the *From* field and say "Look at this." Cima generates the following interpretations (assuming that this particular example is labeled eg03):

> classifyFeature (Begins(SenderID, "pattie"), relevant, "sort mail", eg03)
> classifyFeature (Begins(SenderID, LowercaseWord), relevant, "sort mail", eg03)

If the user says "Always look at this," the disjunct specifier is allExamples rather than eg03. Note however that being relevant to eg03 does not imply being irrelevant to other examples; rather, the instruction biases the learning system to use these features when describing a subset of examples that includes eg03. Cima generates these initial 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. For pointing gestures, as in this example, it finds features relating the selected data to the target example, and generates both specific and generalized values. In this example, Begins(SenderID, Lowercase-Word) is obviously irrelevant to the task. Cima relies on the learning algorithm to choose the best interpretation based on other criteria, such as statistical fit to examples.

ClassifyFeature instructions can emanate from another agent or from domain knowledge.

Cima records the instruction's source and uses credibility ratings to select among conflicting suggestions. 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 may advise her that this causes the description to be inconsistent or excessively complex.

*Related work*

Some other work on machine learning addresses the use of classifyFeature instructions. Clint-Cia (de Raedt and Bruynooghe, 1992), which builds Horn-clause descriptions of classification concepts, can incorporate feature values (that is, predicates) suggested by the user as relevant to the current example. It displays the conjunction of feature predicates it has chosen to form a rule, and invites the user to classify them. Thus it processes instructions of the form

classifyFeature (<predicate>, relevant, <current concept>, <current example>).

Julien and Fenves's (1992) "interactive inductive learning system" (IILS) processes several forms of classifyFeature instructions. In terms of the IILS user interface metaphor, the user sets "constraints" on attributes in the current bias (that is, the set of available attributes), which classify some or all values of a given attribute as relevant, irrelevant, or unclassified—in the last case the learning algorithm itself decides whether to use it. The system also provides a crude method of specifying the disjunct, since the instruction indicates whether the constraint applies to all, some, or the remaining, examples. In terms of our notation, IILS accepts instructions of the form

classifyFeature (<attribute> | <attribute-value>, relevant | irrelevant | unclassified, <current concept>, some | all | remaining).

In relating our research to the machine learning literature, two points stand out. First, previous systems handle only peculiar forms of classifyFeature. Second, they assume that the instruction specifies the attribute or attribute value without ambiguity. They do not handle "hints" of the sort we encountered in our user study and present in the scenarios below, where the learning algorithm must decide which attribute or value is intended.

## 4 Example scenarios

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 strip the local area code (617). Sample data appears in part i of Figure 2: the positive examples (local numbers) are shown in boldface type. The scenarios that follow illustrate teaching the concept "local phone number" by examples (part ii of the Figure) and by using hints along with some domain knowledge (parts iii, iv, and v). We assume that Cima has not yet been taught the concept of phone number. (The “◊” symbol is used throughout to represent the space character.)

*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. Cima records the example and its context, and proposes the rule (a) in Figure 2.ii. When given the second example, 220–7299, Cima generalizes the rule to (b). It predicts the third example, and then the fourth, 255–6191. Because this is preceded by a nonlocal area code, the user rejects it by selecting *Not this one* from the menu. At present Cima is focusing only on the pattern 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). Because it had to create new special-case rules, Cima shifts bias and examines the neighboring text for distinguishing features. As a result, it finds the single general rule shown in (d). This predicts the remaining positive examples, except for an anomalous one, 339–8184, that lacks an area code. When the user says *I want this*, Cima forms the disjunctive ruleset (e). To maximize the similarity between rules, it adopts a generalized pattern for this final phone number—even though it is the only example of the new disjunct.

*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 the text “(617)” and chooses *Look at this* from the popup menu when giving the first example. This causes Cima to examine the user’s selection and infer that this text pattern should precede examples of the concept. Thus it forms the rule shown in line (a) of Figure 2.iii. After the second positive example, Cima generalizes the phone number, as shown in (b). This predicts the remaining examples (other than the anomalous one).

Rather than point at “(617)”, the user could have given a verbal (typed or spoken) hint such as “it follows my area code” while selecting the first example. The phrase “it follows” corresponds to text before and after the example, with preference to the former; “area code” is unrecognized. Using built-in knowledge, Cima selects as salient feature values the parenthesis before the example and the blank space after it. A second verbal hint, “any numbers OK,” given while pointing at the phone number, causes Cima to generalize the example, focusing on tokens of type Number and ignoring other properties such as string value and length. It forms the rule shown in line (a) of Figure 2.iv. After the negative example it specializes the *text FOLLOWS* feature, obtaining the rule in line (b).

A programmer could specify the concept in greater detail by classifying features as follows:

classifyFeature (Token_Sequence (Target, [Number–Number]), relevant, “local phone”, allExamples)

classifyFeature (Ends (BeforeTarget, “(617)”), relevant, “local phone”, allExamples)

The specification is incomplete, but Cima will add the requisite features when given examples. After the first positive example, it forms the rule shown in entry (a) of Figure 2.v; it has added the *Search direction* feature required for utility when searching for data but omitted by the programmer. To cover the anomalous positive example, Cima forms a second rule, using the Token_Sequence suggested by the programmer and an alternative value of the suggested Ends(BeforeTarget) feature, as shown in entry (b) of the Figure.

These scenarios illustrate some important behaviors of the Cima learning system:

- adding and focusing on features suggested by the user;
- focusing on features suggested by background knowledge;
- using knowledge and statistics to choose the most justified interpretation of a hint;
- shifting bias to find simpler descriptions.

## 5 System overview

Cima is implemented in the Common Lisp Object System (CLOS). Figure 3 illustrates its components (except the interface to applications). The *discourse manager* processes instructions, decides when to update the concept or shift bias, and generates feedback to the user. The *dynamic bias manager* loads the current set of features, matches and generalizes their values on new examples, and updates beliefs about their relevance based on user hints or domain knowledge. The *DNF learning algorithm* forms rules by selecting among features proposed by the bias manager, evaluating them on heuristics also supplied by the bias manager.

### *Knowledge sources*

Cima uses three sources of built-in knowledge. *Discourse knowledge* defines the types and forms of instructions users may give, as well as the forms of feedback and elicitation the system can generate. Rules state the context in which feedback and elicitation are used—for instance, if a concept describes the alignment of graphics, the agent draws a guideline to illustrate. Elicitation knowledge includes rules for deciding whether to shift bias or ask the user for a hint when the current bias appears inadequate. Although several such rules have been discussed in the machine learning literature (Haussler, 1988) and implemented in Cima, much more research remains to be done in this area.

*Bias/focus knowledge* comprises the heuristics for ranking the relevance of features, and the methods for interpreting user hints. General knowledge about the domains of text, numbers and graphics is combined with knowledge customized for an application. As explained in Section 2, utility criteria associated with each type of action prescribe the features and restrictions on feature values required for a concept to be operational. To interpret hints, the bias manager uses a thesaurus to map phrases into defined attributes and values. Finally, to select features when forming rules, the learning algorithm uses “justification” heuristics, testing the rules' statistical fit to examples, satisfaction of utility criteria, salience based on domain knowledge, and (most important) beliefs based on user hints.

*Feature knowledge* defines the types and values of features, and their generalization hierarchies and operators. When adding a new type of feature, application programmers define a class and methods for matching two feature values and finding their minimal common generalization. The programmer may also define three optional methods. The first generalizes a feature value based on domain knowledge: for instance, automatically deriving Number–Number from 243–6166. The second finds a generalization that *mismatches* some other value: Cima uses this to specialize a feature when given negative examples. The third computes a default salience score for a given feature value. For instance, the salience method for text gives a high score to short contextual features containing delimiters, such as *text FOLLOWS )◊*, which scores higher than *text FOLLOWS Word◊Word◊Word◊*.

*Processing instructions*

Pseudocode for Cima's instruction handlers appears in Figure 4. The handler for classifyExample instructions gets the observable and computed features of the new example and then updates the concept description. The handler for classifyFeature first generates interpretations of the feature specification and then calls the magnifyBias routine, which adds features suggested by the interpretations to the set of features available for describing the concept. The classifyFeature handler then marks the features as relevant or irrelevant to the given example or subset, and updates the concept description. The handler for classifyRule either marks the rule as accepted by the user, or asks the user to indicate why the rule is incorrect, by classifying some its features as irrelevant. The current classifyRule handler does not attempt to guess which features are wrong, nor does it deal with classifications of "too general" or "too specific."

Figure 4 also describes the routines that support the instruction handlers. The computeFeatures routine extracts the features of an example and then updates the set of feature values available for describing the concept by calling magnifyBias. Features may be directly observable properties of the example, or computed relations between its properties and those of previous examples. A new example may introduce new types of feature, in particular new relations. For instance, suppose we have a sequence of numbers, 2, 4, 8, .... By the third example, it is interesting to consider the relation value(current) = 2×value(previous). The bias knowledge passed to computeFeatures includes feature detectors that check for such interesting relationships. The magnifyBias routine fires these detectors, adds any new features discovered, and then tests or computes their values on all positive and negative examples. Note that some features, such as the relation noted above, may be interesting only if they apply only to positive examples, or only to negative ones, and to examples in sequence. The feature detectors can enforce such conditions.

The updateConceptDescription method is called from classifyExample and classifyFeature. It invokes the learning algorithm (described in the next section) to compute a new set of rules that describe the concept. Sometimes the new ruleset covers some negative examples, or grows much more complex—perhaps because the learning algorithm has formed new special-case rules. Either of these is symptomatic of an inadequate feature repertoire. To remedy this, updateConceptDescription calls the magnifyBias routine, which adds features to the current repertoire in accordance with pre-defined, application-specific rules for widening the learning system's focus of attention.

Cima is a dynamically biased learning system in two respects: it can focus attention by classifying features as relevant or irrelevant before computing a concept description; and it can broaden its search by enlarging the feature set. Both of these operations can be driven by instructions from the user or from the agent's own background knowledge. The next section explains how Cima searches for a concept within a given feature set.

## 6 Learning algorithm

Cima's learning algorithm seeks a DNF ruleset that meets all the utility and instructional criteria, and minimizes the number of rules and features per rule. It uses a greedy subdivision strategy pioneered in ID3 (Quinlan, 1986) and adapted for rule-learning in Prism (Cendrowska, 1987), in which rules are progressively specialized by adding the feature that appears most "useful" or "justified" according to a heuristic, until the rules satisfy utility criteria (such as correct classification). Figure 5 summarizes Cima's methods for creating a ruleset, forming a single rule, and selecting the next most justified feature.

The most noteworthy aspects of the algorithm are that it tests the candidate rule on multiple utility and instructional criteria, and that it selects features according to several heuristics, including beliefs based on user hints. Because the rules it creates meet action-specific utility criteria wherever possible, the algorithm can be used in a variety of applications beyond classification. By considering only rules that contain all features suggested by the user, the algorithm ensures that the learning agent "obeys" the user. To generate an alternative description by "independent thought," the system can set the instructional criteria to nil. The use of several feature selection heuristics allows both suggestions from the user and background knowledge to be exploited, yet their merit is assessed on other criteria, such as how well they distinguish positive and negative examples.

### *Selecting the features*

Perhaps the most important distinction among greedy DNF learning algorithms is the measure that they evaluate to select the next term with which to specialize a rule. ID3 uses an information-based measure, the "information gain" of an attribute with respect to the current subset of examples. Induct (Gaines, 1989) uses a probabilistic measure, Pr[Class | Feature], and then prunes terms from rules on the basis of the probability that the rule would be bettered by a randomly-chosen one. Prism does not prune terms and only generates "exact" rules that perform perfectly on the training set. It uses the same probabilistic measure as Induct, but breaks ties according to the number of examples covered. Cima extends this approach by treating the feature preference heuristics as a parameter and allowing any number of them.

Figure 6 lists the heuristics used by Cima, in order of importance. The first is *suggested relevance*, in which a feature's score depends on whether the user (or an agent) has classified it as relevant or irrelevant, and also reflects the credibility of the source. Figure 7 shows the range and interpretation of scores. A score of 0 indicates that no suggestion has been made. When the user positively affirms a feature as relevant or irrelevant, perhaps by selecting it from a property sheet, it scores 1 (User affirms) or –1 (User repudiates). If the system has inferred that the feature is relevant from a pointing hint, the score is about 0.75 (User suggests).

A hint may refer to an entire collection of features (e.g. "text before selection"), and there may be several competing interpretations. Thus many feature values may score equally on suggested relevance. Filtering the winners through other heuristics amounts to gathering multiple sources of evidence for disambiguating the suggestion.

The second heuristic measures the ability of features to predict the current partitioning of positive and negative examples. Prism's measure, Pr[Class | Feature], tends to overfit the example set, because it prefers features that cover only positive examples regardless of how few they cover. Cima uses *category utility*, adapted from the Cobweb clustering algorithm (Fisher, 1987) and defined as Pr[Class | Feature] × Pr[Feature | Class]. For example, in Figure 8 feature A covers two positive and no negative examples, B covers four of each, and C covers three positives and one negative: C scores highest.

The third heuristic is *utility for action*, which prefers features that contribute most toward achieving the utility criteria specified for the current action. If the action is to classify examples, then the utility metric is category utility (already tested). If it is to find data, the heuristic prefers features that specify the most delimiters or search parameters. For actions that generate data, it prefers features that deterministically specify object attributes. For actions that modify object properties, it prefers features that determine or most strongly constrain a property's new value.

The fourth heuristic, *used in other rule*, dictates that features found relevant to one disjunct are likely to be relevant to another. Particular values are preferred over attributes, so that if color(red) is used in some rule but color(blue) is not, color(red) is preferred (assuming that both have equal category utility in the current subset). But if color(red) is not among the candidate features, the heuristic still prefers any value of color over other attributes such as size and shape.

The fifth and sixth heuristics rank features on application-defined *salience* measures. The seventh heuristic, *generality or specificity*, prefers either more general or more specific values of features, according to the current bias. In the unlikely event that several candidate features remain, Cima finally chooses one arbitrarily.

Machine learning theorists concerned with refining the statistical or informational metric used for feature selection may view the use of multiple heuristics with suspicion. We point out that programming by demonstration presents special demands and opportunities. Application developers and end-users offer a rich variety of knowledge that lacks theoretical rigor and may be expressed ambiguously. In many applications, users expect the system to achieve correct performance after very few examples, and they tolerate only "reasonable" errors (Maulsby *et al*., 1993). Exploiting domain knowledge through multiple heuristics reduces sample complexity and increases the justifiability of statistical inference.

## 7 Evaluation

Cima is designed to interact with users and exploit customized domain knowledge, hence it is not easy to evaluate its performance. Testing it on the large public datasets used in machine learning research would not exercise its most important capabilities, nor predict its interactions with real users. On the other hand, we have yet to implement the robust user interface needed for an effective user study. Our initial evaluation therefore utilizes data gathered in the Turvy user study mentioned in Section 1. Cima was run on transcripts of users' input to Turvy—the examples and hints they gave in the course of teaching Turvy about syntactic structures in a bibliography.

### *Experimental design*

Cima was taught eight textual search patterns from five tasks in the Turvy user study. The concepts are named with a letter identifying the task and a number, as follows: (A1) underlined text, (B1) start and (B2) end of paper title, (C1) primary author's surname, (C2) publication date, (D1) all surnames, and (F1) colons and semicolons to be cut or (F2) replaced with periods. Part i of Figure 9 shows some of the bibliographic data (there were nineteen entries in total), while part ii presents some of the data descriptions that Cima learned.

Cima was tested on four user traces from the Turvy experiment, and on learning from examples alone. Three of the traces were chosen at random from the seven available; the fourth was chosen because it contained hints fraught with errors.

In evaluating Cima the aim is not to replicate all of Turvy's behavior, since Cima is only one component of a task learning agent. But since users accepted Turvy, meeting or exceeding Turvy's concept learning performance is Cima's primary design objective. The experimental conditions biased results both for and against Cima: where possible, more heavily against it. Preparing the input for Cima required a manual step—transcribing users' speech to text and segmenting the utterances—but this gave Cima no special advantage since Turvy understood speech perfectly. Turvy could disjoin attribute values or rules; Cima, only rules. This favored Turvy, which always chose the appropriate tactic. When Cima predicted a different negative example than Turvy, the researcher would classify it correctly, since Turvy's users correctly classified all examples.

Cima was put at a disadvantage by its interpretation of hints. Although both Cima's and Turvy's vocabularies were restricted to terms describing syntactic structures and relations, Turvy understood phrases, such as "Look for *quotes before italic text.*" Cima, however, merely spotted keywords—"quotes", "before", "italic", "text". Thus Cima generated more, and more ambiguous, interpretations than Turvy: in this case it would focus on quotes before italic text, but also on quotes or italics before the example, if those happened to be present. Cima also used a simplistic rule for filling in the Subset argument in classifyFeature instructions: if a hint is given prior to any examples, or includes the keywords "any," "every" or "all," then it applies to allExamples, otherwise it applies to the most recent example. Some users gave hints before demonstrating any example. In this case, Turvy "cleverly" associated hints with the appropriate examples when they appeared. Cima, however, interpreted them as applying to all examples: thus whatever features it might suggest were deemed relevant to any example in which they appeared. Since keyword spotting inevitably led Cima to propose feature types as well as values, it was certain to find some interpretation that would apply to any example. In consequence, Cima tended to form overly complex, specialized rules whenever hints were given prior to any examples.

*Preliminary results*

Learning system performance is often measured by the number of positive and negative examples required to achieve some level of predictive accuracy (as in the PAC learning model, Valiant, 1984). For programming-by-demonstration systems and interface agents, predictive performance *in the course of learning* is more relevant. We used a predictive utility score defined as Pr[Predicted | +] × Pr[+ | Predicted]; that is, the proportion of positive examples the learner predicted, times the proportion of predicted examples that were positive. Analogous to the category utility measure used in the learning algorithm, it balances breadth of coverage (Pr[Predicted | +]) against accuracy (Pr[+ | Predicted]). Scores are normalized relative to Turvy's average: thus a score equal to Turvy's becomes 1.0.

Figure 10 shows Cima's score on the eight concepts. The graph on the left depicts the range of performance on all user traces; that on the right compares learning from examples only with learning from examples and hints. Overall, Cima averages 95% of Turvy's score; 94% when learning from examples alone, and 96% when given hints. Variability between tasks is partly attributable to the relatively small number of examples of each concept. Cima's domain knowledge and automatic generalization enabled efficient learning from examples alone, requiring on average 1 positive and 0.4 negative examples per disjunct. But as the graphs show, performance on some tasks varied widely due to the quality of hints and Cima's interpretation of them.

Although the evaluation is only preliminary because the protocols employed were also used as inspirational examples during system design, we conclude that Cima performs well enough to warrant further development, and that the user interface should be designed to ensure that the user refers to an example when giving verbal hints.

## 8 Conclusion

Cima is a concept learner for agents that acquire task knowledge by recording end-users' input to computer applications. It supports task learning by forming generalizations of individual actions and data. It comprises one half of the agent's learning system; the other half induces the sequential structure of tasks. We envisage an architecture with three main components: the learning system; a discourse manager, which handles instructions from and feedback to the user; and a connection to applications, which records events and generates actions.

In addition to Cima, we have already implemented an algorithm, called ActionStreams, which forms a hierarchical description of a task from an unsegmented stream of example actions, and have tested it in the context of programming by demonstration (Maulsby, in press). Integrating this algorithm with Cima requires a negotiation between the two, to decide whether actions should be generalized by Cima so that they can be matched by ActionStreams. This negotiation is non-trivial, since simplifying the description of a program by matching actions provides a justification for generalizing, but finding a plausible generalization likewise provides a justification for matching. Thus, both Cima and ActionStreams may influence one another's decision to match and generalize. We have also implemented preliminary versions of the discourse manager and the connection to applications. When all components have been integrated, we will conduct further testing on new user dialogs and tasks.

Programming by demonstration offers new opportunities and challenges for research on machine learning. When an agent learns directly from the user, it must utilize multiple sources of information—including interaction with the user. In our research, the machine learning goal of minimizing example complexity is taken to an extreme, and the criteria for learning include action-specific operational utility as well as correct classification. This paper has presented an interaction model comprising three types of instruction, classifyExample, classifyRule, and classifyFeature, and a novel methodology for interpreting ambiguous and incomplete hints in terms of classifyFeature instructions. The model has been implemented in the Cima learning algorithm and has been shown to perform well on dialogs collected from human interaction with a simulated interface agent. The work demonstrates that keeping the user in the loop makes it possible for machine learning to enhance human-computer interaction.

## References

Angluin, D. 1988. Queries and concept learning. *J. Machine Learning (2)*, pp. 319–342.

Cendrowska, J. 1987. PRISM: an algorithm for inducing modular rules. *Int. J Man-Machine Studies*, *27*(4), pp. 349–370.

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

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

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

Fisher, D. 1987. Knowledge acquisition via conceptual clustering. *J. Machine Learning (2)*, pp. 139–172.

Gaines, B.R. 1989. An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. In *Proc. ML'89, Sixth International Workshop on Machine Learning*, pp. 156–159. San Mateo, CA.

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

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

Julien, B. and Fenves, S. J. 1992. Constraining induction with expert knowledge. In *AAAI-92 workshop on constraining learning with prior knowledge*, pp. 27–32. San Jose, CA.

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

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

Maulsby, D. In press. Inductive task modeling for user interface customization. In *Proc. International Conference on Intelligent User Interfaces.* ACM Press.

Quinlan, J.R. 1986. Induction of decision trees. *J. Machine Learning (1)*, pp. 81–106.

Valiant, L.G. 1984. A theory of the learnable. *Communications of the ACM (27) 11*, pp. 1134–1142.

## Figure captions

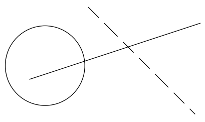
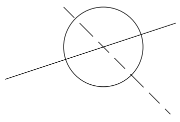
| | | | | |
|---|---|---|---|---|
| **Classify** | From pattie@media<br>To maulsby@media<br>Subject Tuesday meeting | ▶ | Mail from pattie | If the message is from “pattie”,<br>then put it in the folder “Mail from pattie”. |
| **Find** | tel 243–6166<br>fax 284–4707 | ▶ | tel 243–6166<br>fax $\boxed{284–4707}$ | Find the next telephone number preceded by the word “fax”. |
| **Generate** | Mon 21 toDo | ▶ | Mon 21 toDo<br>Tue 22 toDo | Insert a calendar template of the form: [Next(DayName) Tab Next(DayNumber) Tab toDo]. |
| **Modify** |  | ▶ |  | Move the circle to the point at which a dashed line intersects a plain line. |

Figure 1

i

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

ii

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 shift of bias

```
Searching forward,
    Selected text FOLLOWS  617)◊   and   MATCHES  Number(length 3)–Number(length 4)
```
*— Note: Cima proposes 617)◊ rather than 7)◊ because it tokenizes at the word level by default*

e. Ruleset after final positive example “339–8184”

```
Searching forward,
    Selected text FOLLOWS  617)◊   and   MATCHES  Number(length 3)–Number(length 4)
or  Selected text FOLLOWS  ;◊      and   MATCHES  Number(length 3)–Number(length 4)
```

iii

a. Rule formed after first example and pointing hint

Searching forward, Selected text FOLLOWS (617)◊ and MATCHES 243–6166

b. Rule generalized after second example

Searching forward, Selected text FOLLOWS (617)◊ MATCHES Number(length 3)–Number(length 4)

iv

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

v

a. Rule formed after first example and partial specification

Searching forward, Selected text FOLLOWS (617)◊ and MATCHES Number–Number

b. Ruleset after final positive example

```
Searching forward,
    Selected text FOLLOWS  (617)◊  and   MATCHES  Number–Number
or  Selected text FOLLOWS  ;◊      and   MATCHES  Number–Number
```

Figure 2

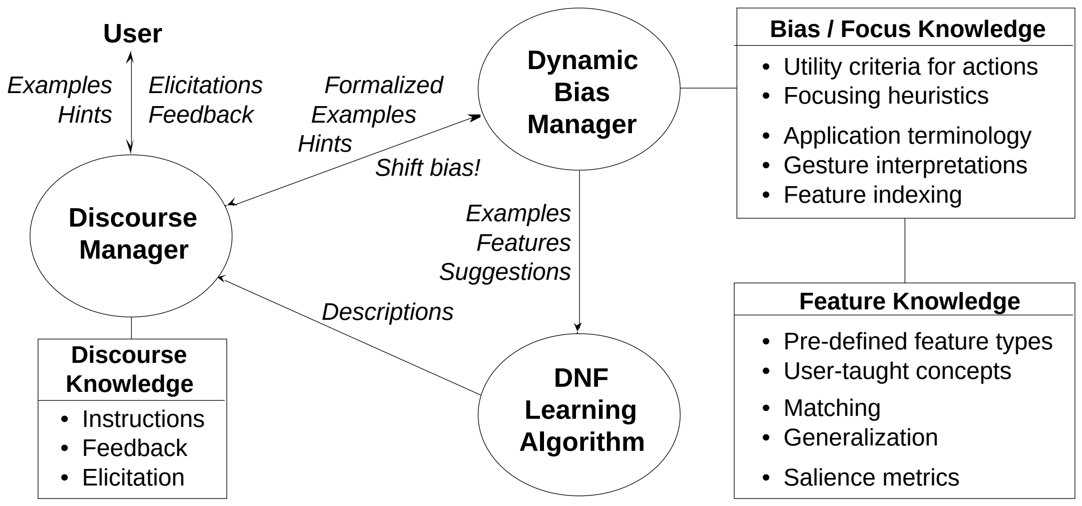


Figure 3

```
on classifyExample (Example, Class, Concept)
    computeFeatures (Example, Class, Concept, BiasRules for classifyExample)
    add Example to Examples of Concept
    updateConceptDescription (Concept, classifyExample)

on classifyFeature (FeatureSpec, Class, Concept, Subset)
    compute interpretations of FeatureSpec
    magnifyBias (FeatureSet for Concept, interpretations of FeatureSpec, Examples of Concept,
                 BiasRules for classifyFeature)
    repeat for each Feature in interpretations of FeatureSpec
        mark Feature as relevant or irrelevant (according to Class) to the given Subset of Concept
        note source of justification for Class [ie. suggested by user hint, background knowledge, etc.]
    updateConceptDescription (Concept, classifyFeature)

on classifyRule (Rule, Class, Concept)
    if Class is Correct, mark Rule as userAccepted
    else    mark Rule as UserRejected and
            ask user to classify some features of Rule

computeFeatures (Example, Class, Concept, BiasRules)
    repeat for Features of Concept
        if Feature is observable in Example, then get observed value of Feature
        else compute value of Feature in Example in relation to other Examples of Concept
        magnifyBias (Features of Concept, Features of Example, Examples of Concept,
            FeatureDetectors in BiasRules)

magnifyBias (FeatureSet, SuggestedFeatures, Examples, BiasRules)
    add SuggestedFeatures to FeatureSet
    fire BiasRules to add further features to FeatureSet
    repeat for each new Feature in FeatureSet
        observe or compute value of Feature in all Examples [if possible]

updateConceptDescription (Concept, Instruction)
    repeat until success or resource limits reached
        newRuleset = makeRules (Concept, FeatureSet for Concept, Examples of Concept,
            UtilityCriteria for Concept, PreferenceHeuristics for Concept)
        if newRuleset is inconsistent or has increased complexity then
            magnifyBias (FeatureSet for Concept, nil, Examples of Concept, BiasRules for Instruction)
        else signal success
```

Figure 4

```
makeRules (Concept, Features, Examples, Criteria, Heuristics)
    repeat until all positive examples are covered:
        add makeOneRule (Concept, Features, Examples, Criteria, Heuristics) to Concept’s definition
    return new Concept definition

makeOneRule (Concept, Features, Examples, Criteria, Heuristics)
    create new empty Rule
    repeat until Rule meets Utility Criteria and Instructional Criteria, or until all Features have been tried:
        add mostJustifiedFeature (Features, Heuristics, Concept, Examples) to Rule
        delete Examples no longer covered by Rule
        update bias, removing Criteria already satisfied and re-ordering preferences
    simplify (Rule, Concept, Features, Examples, Criteria, Heuristics)
    return Rule

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

Figure 5

1. suggested relevance
2. category utility
3. utility for action
4. used in other rule
5. feature value salience
6. feature type salience
7. generality or specificity
8. arbitrary choice

Figure 6

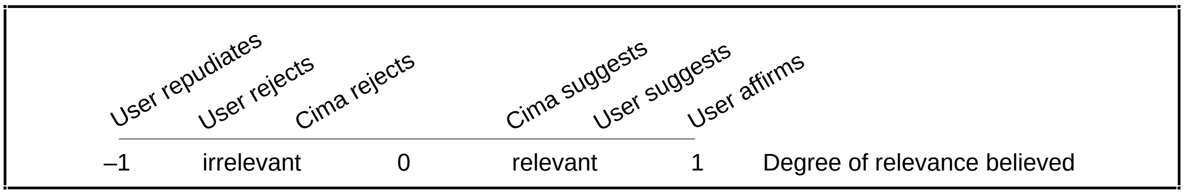
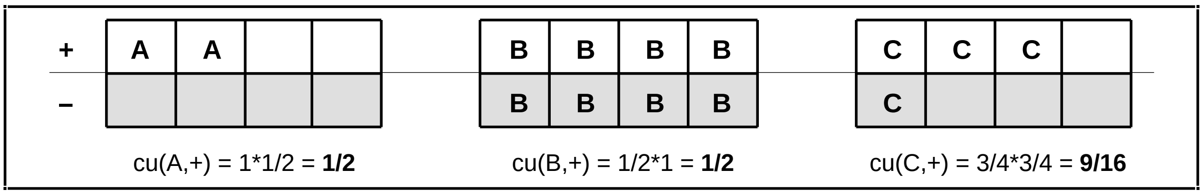


Figure 7



+
–
A
A
B
B
B
B
B
B
B
B
C
C
C
C
cu(A,+) = 1*1/2 = **1/2**
cu(B,+) = 1/2*1 = **1/2**
cu(C,+) = 3/4*3/4 = **9/16**

Figure 8

i

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

B1 Start of journal paper title

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

B2 End of journal paper title (actually, after colon at end of title)

Searching forward from previous example of B1, Insertion point PRECEDES ◊%

C1 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

D1 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 Selected text MATCHES Mitchell

Figure 9

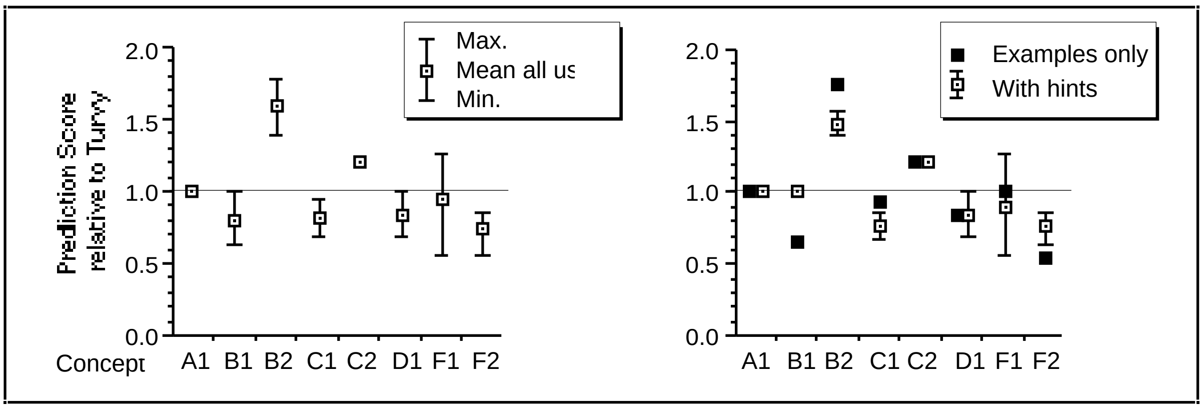


Figure 10