# Interactive concept learning for end-user applications

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

[1] Media Laboratory
Massachusetts Institute of Technology
Room E15–423
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

Personalizable software agents will learn new tasks from their users. This implies being able to learn from instructions users might give: examples, yes/no responses, and ambiguous, incomplete hints. Agents should also exploit background knowledge customized for applications such as drawing, word processing and form-filling. The task models that agents learn describe data, actions and their context. Learning about data from examples and hints is the subject of this paper.

The Cima learning system combines evidence from examples, task knowledge and user hints to form Disjunctive Normal Form (DNF) rules for classifying, generating or modifying data. Cima's dynamic bias manager generates candidate features (attribute values, functions or relations), from which its DNF learning algorithm selects relevant features and forms the rules. The algorithm is based on a classic greedy method, with two enhancements. First, the standard learning criterion, correct classification, is augmented with a set of utility and instructional criteria. Utility criteria ensure that descriptions are properly formed for use in actions, whether to classify, search for, generate or modify data. Instructional criteria ensure that descriptions include features that users suggest and avoid those that user reject. The second enhancement is to augment the usual statistical metric for selecting relevant attributes with a set of heuristics, including beliefs based on user suggestions and application-specific background knowledge. Using multiple heuristics increases the justification for selecting features; more important, it helps the learner choose among alternative interpretations of hints.

When tested on dialogues observed in a prior user study on a simulated interface agent, the learning algorithm achieves 95% of the learning efficiency standard established in that study.

## 1 Introduction

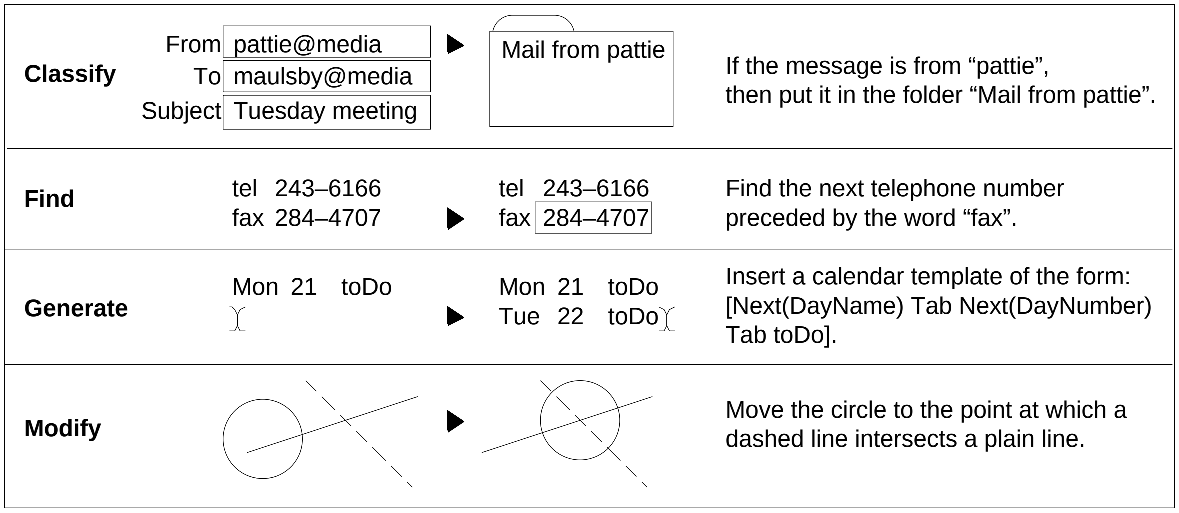
A truly personalizable software agent is one that learns new tasks from the user. For the interaction to be comfortable and reliable, the agent must facilitate rich communication with both user and task environment. Its learning mechanism should minimize the number of examples needed—in particular negative examples, which users perceive as time-wasting errors. To take advantage of users' knowledge and keep them in control, the learner should accept additional instructions: partial specifications or "hints." It should 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 biases and 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 on their plausibility and statistical utility.

This paper describes Cima (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 (PBD) 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 87]. Prism finds a set of rules covering all and only positive examples, and forms each rule 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 forms each rule by adding features 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 the user suggests and avoids ones 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 93]. 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. *Datadescriptions* [Halbert 93] 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



Classify
From pattie@media
To maulsby@media
Subject Tuesday meeting
Mail from pattie
If the message is from “pattie”, then put it in the folder “Mail from pattie”.
Find
tel 243–6166
fax 284–4707
tel 243–6166
fax 284–4707
Find the next telephone number preceded by the word “fax”.
Generate
Mon 21 toDo
Mon 21 toDo
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 — General types of action on data

data, and to be useful, 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. Cima embodies a model for testing whether a description determines the necessary action parameters (c.f. the “operationality” tests used in the pioneering Eager PBD system [Cypher 93b]). Figure 1 illustrates four types of action: classify (sort) data; find data; generate new data; and modify properties. Cima continues adding features to a rule until it 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 in 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* stipulates two criteria: the description should discriminate between positive and negative examples, and it should determine (as far as possible) the property’s new value. As when generating data, 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 $(x,y)$ 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 and the target is zero. Features that remove both degrees of freedom, e.g. *touch(Circle.center,Circle2.center)*, 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 be 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 94]). Here we focus on the instructions that the learner can interpret and show how they are processed.

Cima supports three types of instruction:

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

The first classifies an example as positive or negative with respect to some concept: this is the usual instruction supported by supervised learners [Michalski 83]. The second states whether a given rule is valid: it is widely adopted in systems that learn from an informant [Angluin 88].

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. Clint-Cia [de Raedt 92] supports this instruction in a restricted form: it displays the conjunction of feature predicates it has chosen to form a rule, and invites the user to classify them. Thus, each classifyFeature instruction given by the user specifies a particular attribute value, its class (relevant or irrelevant), the concept and disjunct (current rule). RAP lets a user classify words and phrases as relevant features of an email message [Bocionek 94]. Cima extends this approach by interpreting ambiguous, incomplete *hints*. A hint may map to several classifyFeature instructions, and it need not define all the arguments. The user may suggest either a feature type or a specific value, and may refer to a rule, a set of examples, or a particular example.

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. 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 test the initial interpretations on other criteria, such as statistical fit to examples, and choose the best one.

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.

## 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 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.

### *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, checking the context for distinguishing features. It now 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 consider the preceding context, focusing on the text the user suggested, and form 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).

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. Rule specialized after negative example

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

Figure 2 — i Sample phone numbers (positive examples shown in bold)
ii Series of data descriptions induced from examples
iii Data descriptions induced from examples and pointing hint
iv … from examples and verbal hints
v … from examples and partial specification

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 context before and after the example, with preference to the former; "area code" is unrecognized. Using default knowledge, Cima selects as salient feature values the parenthesis before the example and the blank space after it. The learning algorithm settles on *text FOLLOWS )◊* as the relevant feature, since that interpretation is preferred and no other evidence counts against 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. Thus, after the first positive example, it forms the rule shown in entry (a) of Figure 2.v; it has added the *Searchdirection* 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.

Cima uses three sources of built-in knowledge to interpret hints and find features relevant to a given action. *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 might draw 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.

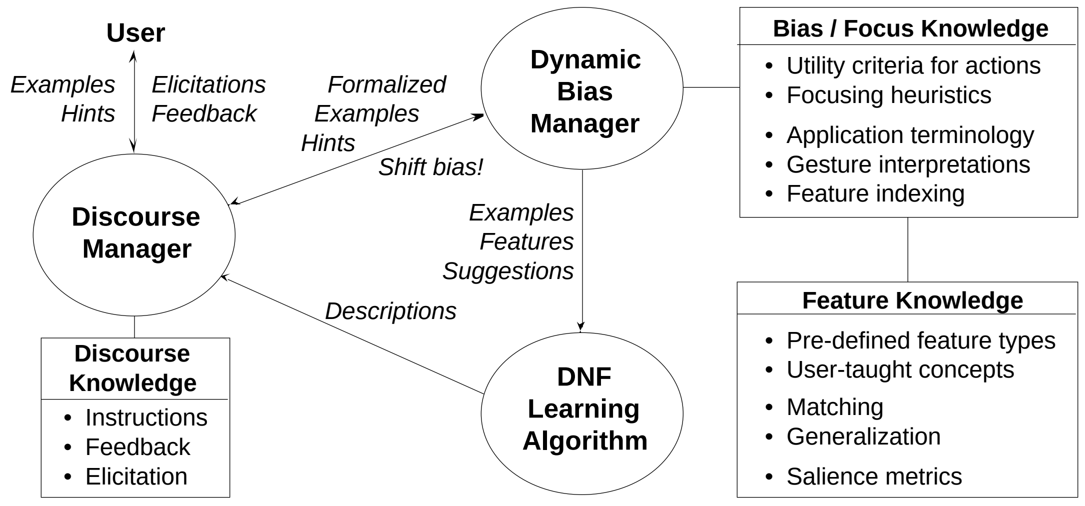


Figure 3 — Learning system components

*Bias/focus knowledge* comprises the criteria and heuristics to be applied when forming a concept, and the mappings from the content of user hints to inferences about the relevance of features. The knowledge is application-specific, although general knowledge about the domains of text, numbers and graphics can be combined with knowledge customized for an application. As explained in Section 2, utility criteria associated with each type of action define the features and restrictions on feature values required for a concept to be useful. To interpret hints, the bias manager uses knowledge about the meaning of words and gestures, encoded in a thesaurus which maps them to attributes and values. Justification heuristics, used by the learning algorithm to rank features, test their 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 CLOS 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◊*.

## 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 86], in which rules are progressively specialized by adding the feature

```
makeRules (Concept, Features, Examples, Criteria, Heuristics)
  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
  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 4 — Algorithm for composing DNF data description rules

that appears most “useful” or “justified” according to a heuristic, until the rules satisfy utility criteria (such as correct classification). Figure 4 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 89] 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 “justification heuristics” as a parameter and allowing any number of them.

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 5 — Heuristics used to select most justified feature

Figure 5 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 6 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). This enables Cima to acquire beliefs about feature relevance from multiple sources, and then focus on the most credible ones when selecting features.

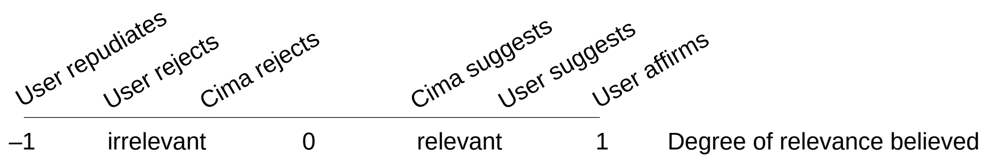


Figure 6 — Degree of the "suggested relevance" measure

A suggestion 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.

Prism 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 87] and defined as Pr[Class | Feature] × Pr[Feature | Class]. For example, in Figure 7 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.

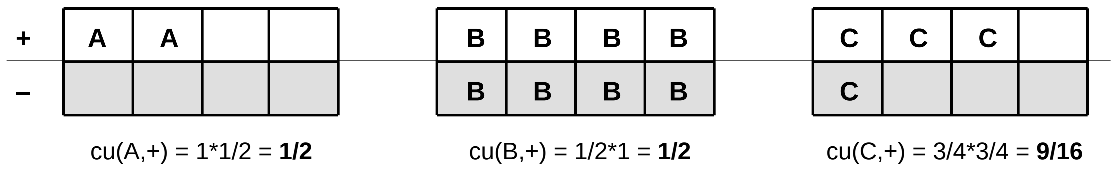

+
–
A
A
B
B
B
B
B
B
B
B
C
C
C
C
cu(A,+) = 1*1/2 = **1/2**
cu(B,+) = 1/2*1 = **1/2**
cu(C,+) = 3/4*3/4 = **9/16**

Figure 7 — Category utility of three features, A, B and C

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 another 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 prefers color(blue) (and other values of color) to size, shape, and so on.

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 preference setting. 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 PBD presents special demands and opportunities. Application developers and end-users offer a rich variety of knowledge, but it lacks theoretical rigor and may expressed ambiguously. In many applications, users expect the system to achieve correct performance after very few examples, and they tolerate only “reasonable” errors [Maulsby 93]. 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 8 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 its 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 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 8 — i Some of the bibliographic data used to evaluate Cima
(Legend: “%” = start or end italics; “@” = start or end boldface)
ii Data descriptions learned for some concepts from the user study

researcher would classify it correctly, since Turvy’s users correctly classified all examples. Some hints mentioned no example (disjunct) to which they should apply. Turvy rightly guessed whether to use the hint in some or all rules; Cima was told to use suggested features wherever possible, and to use some other value if the one suggested did not apply—a bias favoring overspecialization.

*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 84]). For PBD 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. Scores are normalized relative to Turvy’s average: thus a score equal to Turvy’s becomes 1.0. Figure 9 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. Turvy’s absolute score is overlaid on the

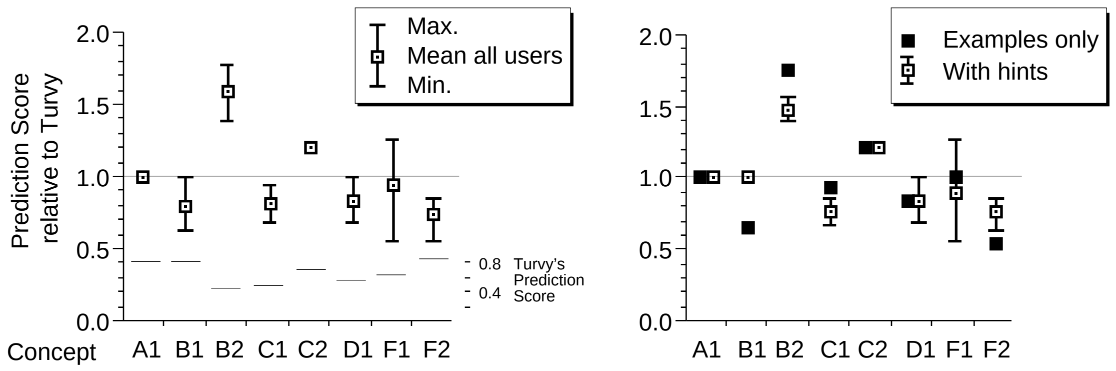


Figure 9 — Cima's performance relative to Turvy

left-hand graph. 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. But as the graphs show, Cima's performance on some tasks varied widely due to the quality of hints. On average, hints did not significantly improve learning, and Cima's ability to utilize hints fell short of Turvy's. Treating hints as applicable to all examples by default produced many extraneous disjuncts and failures to predict, accounting for most if not all of this shortfall. Moreover, 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. We conclude that Cima performs well enough to warrant further development, and that the user interface should be designed to ensure that hints are associated with examples.

## 8 Conclusion

When an interactive user agent learns, it must utilize multiple sources of information—including interaction with the user. The 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 operationalized as the Cima learning algorithm and has been shown to perform well on dialogues collected from human interaction with a simulated interface agent—although this evaluation is only preliminary because the protocols employed were also used as inspirational examples during system design.

Further work includes embedding the Cima algorithm into an interactive PBD system, and more extensive user testing on new dialogues and tasks. An important omission from Cima is the inference of the sequential structure of tasks. A software agent has already been described that completes sequential patterns [Schlimmer 93], but it works from several examples of each pattern, which have been delimited by the user. A more natural extension of Cima would be to synthesize a program from a single, continuous, unsegmented, example of a behavior stream. An algorithm has been developed for this [Nevill-Manning 94] but not yet placed into the context of PBD.

## References

[Angluin 88] Angluin, D. "Queries and concept learning." *J. Machine Learning (2),* pp. 319–342.

[Bocionek 94] Bocionek, S. "Software secretaries: learning and negotiating personal assistants for the daily office work," in *Proc. IEEE Int. Conference on Systems, Man and Cybernetics,* pp. 7–12. San Antonio TX.

[Cendrowska 87] Cendrowska, J. "PRISM: an algorithm for inducing modular rules." *Int. J Man-Machine Studies, 27*(4), pp. 349–370.

[Cypher 93a] Cypher, A. (ed.) *Watch what I do: programming by demonstration.* MIT Press. Cambridge MA. 1993.

[Cypher 93b] Cypher, A. "Eager: programming repetitive tasks by demonstration," in [Cypher 93a], pp. 205–217.

[de Raedt 92] de Raedt, L., and Bruynooghe, M. "Interactive concept-learning and constructive induction by analogy." *J. Machine Learning (8),* pp. 107–150.

[Fisher 87] Fisher, D. "Knowledge acquisition via conceptual clustering." *J. Machine Learning (2)* pp. 139–172.

[Gaines 89] Gaines, B.R. "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 93] Halbert, D.C. "SmallStar: programming by demonstration in the desktop metaphor," in [Cypher 93a], pp. 103–123.

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

[Maulsby 94] Maulsby, D. "Instructible agents," PhD thesis. Dept. of Computer Science, University of Calgary. 1994.

[Michalski 83] Michalski, R.S. "A theory and methodology of inductive learning," in *Machine Learning I,* pp. 83–134. Tioga. Palo Alto.

[Nevill-Manning 94] Nevill-Manning, C., Witten, I.H., and Maulsby, D.L. "Compression by induction of hierarchical grammars." Proc Data Compression Conference, edited by J.A Storer and M. Cohn. IEEE Press, 1994, pp. 244–253.

[Quinlan 86] Quinlan, J.R. "Induction of decision trees." *J. Machine Learning (1),* pp. 81–106.

[Schlimmer 93] Schlimmer, J.C. and Hermens, L.A. "Software agents: completing patterns and constructing user interfaces." *J AI Research, 1,* pp. 61–89.

[Valiant 84] Valiant, L.G. "A theory of the learnable." *Communications of the ACM (27) 11,* pp. 1134–1142.