LEARNING TO DESCRIBE DATA IN ACTIONS

DAVID MAULSBY* AND IAN H. WITTEN**

**Media Laboratory, Massachusetts Institute of Technology, Room E15–423*
Cambridge MA 02139–4307 USA; tel. +01 (617) 253–9832; email: maulsby@media.mit.edu

***Department of Computer Science, University of Waikato, Private Bag 3105*
Hamilton, New Zealand; tel. +64 (7) 838–4246; email: ihw@cs.waikato.ac.nz

Abstract. *Traditional machine learning algorithms have failed to serve the needs of systems for Programming by Demonstration (PBD), which require interaction with a user (a teacher) and a task environment. We argue that traditional learning algorithms fail for two reasons: they do not cope with the ambiguous instructions that users provide in addition to examples; and their learning criterion requires only that concepts classify examples to some degree of accuracy, ignoring the other ways in which an active agent might use concepts. We show how a classic concept learning algorithm can be adapted for use in PBD by replacing the learning criterion with a set of instructional and utility criteria, and by replacing a statistical preference bias with a set of heuristics that exploit user hints and background knowledge to focus attention.*

Key Words. Machine Learning, Programming by Demonstration, Concept Learning

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. A promising approach is Programming by Demonstration (PBD), in which the user teaches by performing and describing a task in the actual environment (or a simulation of it). PBD combines machine learning with user interaction, but existing PBD systems eschew classical machine learning methods in favor of *ad hoc* application-specific inferences. This is because formal learning methods have failed to meet important design criteria for PBD systems. First, the user should be able to control what the system learns, by guiding its inferences with hints or partial specifications. To facilitate this, the system should give the user comprehensible feedback. Second, the system should exploit application-specific domain knowledge to improve the speed and quality of learning. It should minimize the number of examples needed—in particular negative examples, which users perceive as time-wasting errors—and in some cases generalize from even a single example. Third, the system should learn not merely how to distinguish positive from negative examples, but to describe all aspects of data relevant to actions it is taught. For instance, when the task is to generate data, the system should learn how to set all parameters, and which settings require user input or confirmation.

In this paper we show how a classical concept learning algorithm, Prism [Cendrowska 87], can be adapted for use in a PBD system. The concept learner is only part of the system, but a vital part, since defining criteria for selecting and modifying data is central to modeling the individual actions that comprise a procedure. The algorithm may be used in a system for learning condition-action rules, rewrite rules, or sequences of commands applied to data.

In its original form, 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. The enhanced version, called Cima (pronounced *Chee-mah*) 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 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 importance of using 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.

The next two sections describe the utility and instructional criteria. Section 4 presents worked examples, and Section 5 describes the algorithm. Section 6 briefly summarizes the results of a preliminary evaluation of Cima.

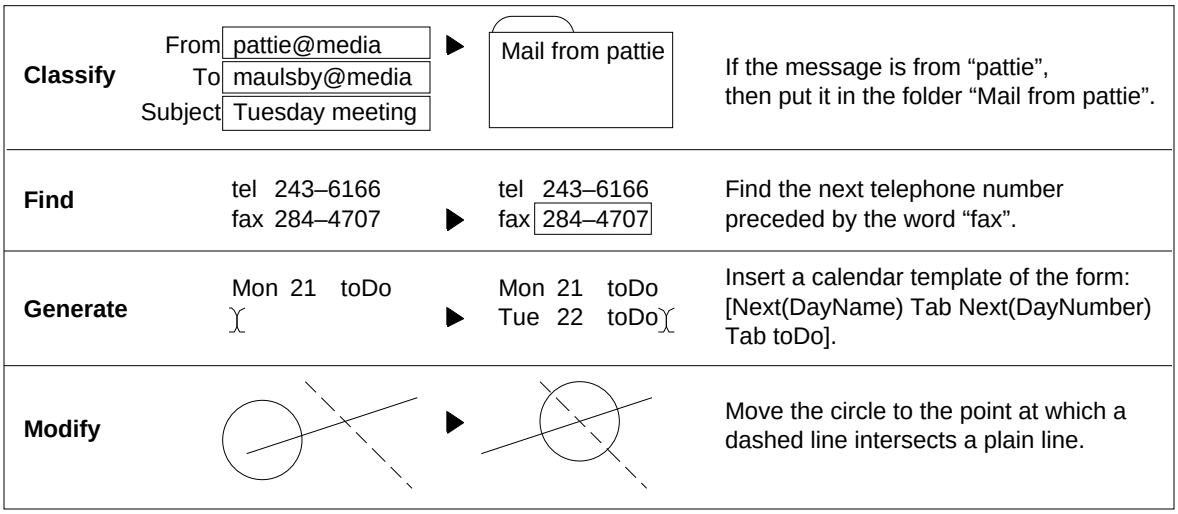

Figure 1 *General types of action on data*

2 UTILITY CRITERIA

To model tasks, an agent needs to learn about data, actions, and when to act. *Data descriptions* [Halbert 93] specify criteria for selecting objects, as well as 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, data descriptions may require features in addition to those needed for classification. This is one reason why rule-learning algorithms are rarely found in interface agents. Explanation-Based Generalization [Mitchell 86] augments correct classification with an operationality criterion, a theoretical bias on the form of descriptions to be learned, which requires that features in the concept description be directly observable in examples. For PBD, we generalize the notion of operationality, proposing a set of utility criteria selected according to the type of action to be performed. Figure 1 illustrates four types of action: classify data; find data; generate new data; and modify properties (these types may be specialized for a particular application). Utility criteria ensure that a data description determines the necessary action parameters (c.f. the “operationality” tests used in the pioneering Eager PBD system [Cypher 93b]). They also impose a preference bias toward features with high utility.

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 concept 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 delimiters or constraints are preferred. For instance, the rule *follows the string “fax ”* is incomplete; Cima adds *matches Number–Number* to specify where the string ends.

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, *touch(Circle.center, Line1)* and *touch(Circle.center, Line2)*, determine the circle’s new (x,y) location. By itself, each leaves one degree of freedom on the circle’s

location. The utility criteria for setting an object's location assume that the goal is to remove all degrees of freedom if possible. Hence, features that remove both degrees of freedom, e.g. *touch(Circle.center, 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 of freedom remain. If the user rejects an example in which the circle touches two solid lines, Cima adds a third feature—that one of the lines be dashed—to meet the classification criterion.

Note that utility criteria and preferences do not by themselves solve the problem of generating features—they merely select among features proposed by generalization operators. To model the richness of actions—in particular those that generate or modify data—a PBD system relies on generalization operators to discover constants, variables and functions.

3 INSTRUCTIONAL CRITERIA

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. Restricted forms of this instruction appear in Inductive Logic Programming systems. For instance, Clint-Cia [de Raedt 92] displays the conjunction of features it has chosen to form a rule, and invites the user to classify them as correct, incorrect or irrelevant. RAP [Bocionek 94] lets a user classify words and phrases as relevant features of an email message. 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 the Disjunct argument 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 internally labeled eg03):

classifyFeature (Begins(SenderID, "pattie"), relevant, "sort mail", eg03)
classifyFeature (Begins(SenderID, LowercaseWord), relevant, "sort mail", 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. Cima relies on the learning algorithm to test these initial interpretations on other criteria, such as statistical fit to examples, to choose the best one.

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) from local phone numbers. Sample data appears in part i of Figure 2. The scenarios that follow illustrate teaching the concept "local phone number" by examples and by using hints along with some domain knowledge. We assume that Cima has not yet been taught the concept of phone number.

4.1 LEARNING FROM EXAMPLES

To give the first example, the user selects 243–6166 with the mouse and chooses *I want this* from a menu.

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 at (617)

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 *Sample data and four scenarios for teaching "local phone number"*

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

Cima records the example and its context, proposing 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 thereof 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.

4.2 SUGGESTIONS FROM THE USER

Now consider the same task taught by examples and hints. Realizing that the distinguishing feature of a local phone number is its area code, the user selects the text "(617)" and chooses *Look at this* from the popup menu when giving the first example. This causes Cima to shift bias and examine the text preceding the example, focusing on the text suggested by the user; Cima forms the rule shown in line (a) of Figure 2.iii. After the second positive example, it 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" suggests text either before or 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. Thus, after one example and two hints, Cima forms the rule shown in line (a) of Figure 2.iv. After a negative example it specializes the *text FOLLOWS* feature, obtaining rule (b).

A programmer could partially specify the concept 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 *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 a user;
- focusing on features suggested by task knowledge;
- using knowledge and statistics to choose the most justified interpretation of a hint;
- shifting bias to find simpler descriptions.

5 LEARNING SYSTEM

Cima is implemented in the Common Lisp Object System (CLOS). Figure 3 illustrates its components (except the interface to applications). The *interaction manager* processes instructions, decides when to update the concept or shift bias, and generates feedback to the user. The *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 *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. *Interaction 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. Rules decide whether to shift bias or ask the user for a hint when the current bias appears inadequate.

Bias/focus knowledge comprises the criteria and heuristics to be applied when forming a concept, and

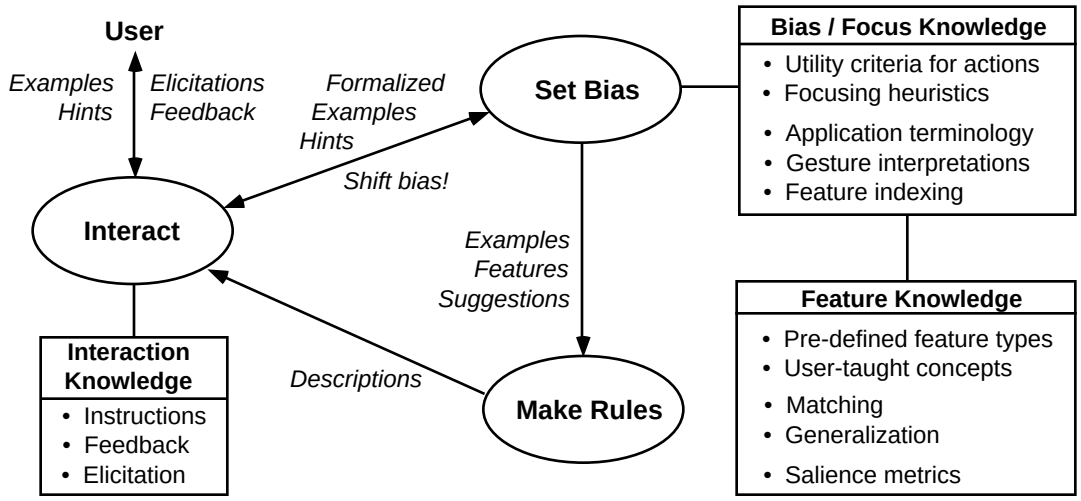

Figure 3 *Concept learning system components*

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, an application programmer defines a CLOS class and two required methods: one for matching a pair of feature values and finding their minimal common generalization.

```
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 FeatureWithHighestExpectedUtility (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

FeatureWithHighestExpectedUtility (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*

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*

The programmer may also define optional methods. One method generalizes a feature value based on domain knowledge: for instance, automatically deriving Number–Number from 243–6166. Another finds generalizations that cover a set of examples (this is used for instance to find recurrence relations and transformation functions). The third method finds a generalization that *mismatches* some other value: Cima uses this to specialize a feature when given negative examples. The fourth 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◊*

5.1 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 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 aspect of the algorithm for making a rule is that it tests the candidate rule on utility and instructional criteria, which are parameterized according to the current action as described in Sections 2 and 3. On each iteration, it adds the feature that is expected to contribute most toward satisfying these criteria, as measured by a suite of heuristic tests which assign numeric scores to feature values. 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 requiring that rules contain any 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 instructional criteria to nil. By combining several feature selection heuristics, the algorithm deals with conflicting or ambiguous suggestions from the user and background knowledge, assessing their relative merit on other criteria, such as how well they predict positive and negative examples.

5.2 HEURISTICS

Perhaps the most important distinction among greedy DNF learning algorithms is the choice of heuristic for selecting 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], to select terms during rule formation. Once a rule is complete, it prunes terms by calculating the probability that a randomly-chosen rule would perform better than the original rule. If this probability decreases when a term is removed from the rule, it removes the term. 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. Candidate features are filtered through the suite of heuristics until only one candidate remains.

Figure 5 lists the heuristics used by Cima in order of importance (the order in which they are tested). 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 featurerelevance from multiple sources, and then focus on the most credible ones when selecting features.

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*

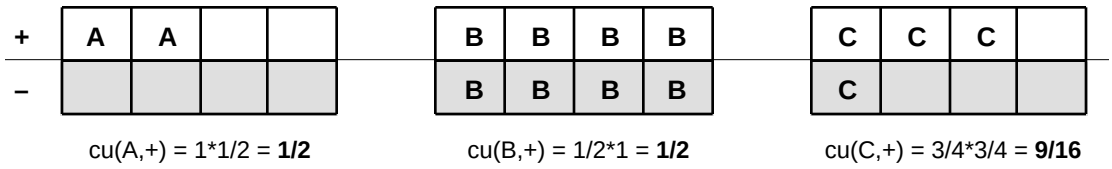

Figure 7 *Category utility of three features, A, B and C*

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.

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.

6 EVALUATION

Since Cima is intended to learn about data as they are manipulated during a task, and to learn through interaction with users, the most appropriate form of evaluation is a user study. But since our task-learning system is still under development, we cannot yet perform a live user study. Instead, we tested Cima on traces of the instructions that users gave to the simulated agent Turvy, described in Section 1. Space permits only a brief summary of the results; the reader is referred to [Maulsby 94] for details. Since Cima does not learn action sequences, task traces were reduced to sequences of example data, and each class of data was taught separately. Users' verbal and gestural hints were included in the traces; the former coded as text strings, the latter as selections of text combined with a "look here" command. Cima and Turvy were compared on their ability to predict positive examples (and avoid predicting negative ones). Under these conditions, Cima achieved 95% of Turvy's predictive accuracy, indicating that the system design specifications derived from the Turvy experiment are practically sufficient to

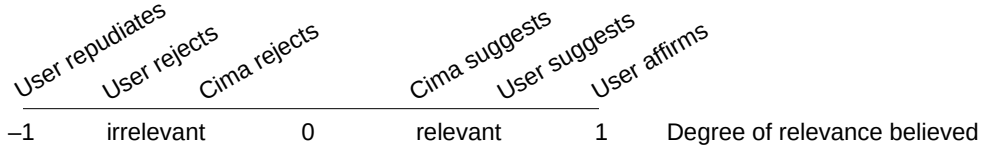

Figure 6 *Degree of "suggested relevance" measure*

replicate Turvy's behavior, at least in respect to learning data descriptions.

On comparing Cima's ability to learn from examples with hints versus learning from examples alone, we found that hints did *not* significantly improve learning on average. This is because some hints, given without reference to particular examples, were interpreted too broadly and thus applied to inappropriate examples, causing the formation of excessively specialized rules. Apparently, the Wizard of Oz unconsciously adopted a narrower interpretation. Cima has been modified to emulate this behavior, by not requiring the adoption of features suggested by such disconnected hints, though nonetheless preferring them.

7 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 instructions, classifyExample, classifyRule, and classifyFeature, and a novel methodology for interpreting ambiguous and incomplete hints in terms of classifyFeature instructions. The model has been used to adapt a traditional concept learning algorithm so that it can learn data descriptions for finding, generating and modifying data as well as classifying them.

Further work includes embedding the Cima algorithm into a task learner, where it may also form concepts to describe actions and their context. The complete task learner can then undergo more extensive user testing. 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 application of Cima would be in synthesizing a program from a single, continuous, unsegmented, example behavior stream. An algorithm has been developed for this [Nevill-Manning 94]. Another interesting direction is to augment other machine learning algorithms with utility and instructional criteria, so that they can support the interactive learning of task-situated concepts and thereby meet the needs of PBD systems.

ACKNOWLEDGMENTS

This work has been funded by the Natural Sciences and Engineering Research Council of Canada, by Apple Computer Inc., and by the Media Laboratory at the Massachusetts Institute of Technology.

REFERENCES

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

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

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

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

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

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

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

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

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

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

Mitchell, T.M., R.M. Keller, and S.T. Kedar-Cabelli. "Explanation-based generalization: a unifying view." *Machine Learning 1*, pp. 47–80.

Nevill-Manning, C., I.H. Witten, and D. Maulsby. "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, J.R. "Induction of decision trees." *J. Machine Learning (1)*, pp. 81–106.

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

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