Mixed Initiative Interfaces for Learning Tasks: SMARTedit Talks Back*

Steven A. Wolfman Tessa Lau Pedro Domingos Daniel S. Weld
Department of Computer Science & Engineering
University of Washington, Box 352350
Seattle, Washington 98195-2350
{wolf, tlau, pedrod, weld}@cs.washington.edu

ABSTRACT

Applications of machine learning can be viewed as teacher-student interactions in which the teacher provides training examples and the student learns a generalization of the training examples. One such application of great interest to the IUI community is adaptive user interfaces. In the traditional learning interface, the scope of teacher-student interactions consists solely of the teacher/user providing some number of training examples to the student/learner and testing the learned model on new examples. Active learning approaches go one step beyond the traditional interaction model and allow the student to propose new training examples that are then solved by the teacher. In this paper, we propose that interfaces for machine learning should even more closely resemble human teacher-student relationships. A teacher's time and attention are precious resources. An intelligent student must proactively contribute to the learning process, by reasoning about the quality of its knowledge, collaborating with the teacher, and suggesting new examples for her to solve. The paper describes a variety of rich interaction modes that enhance the learning process and presents a decision-theoretic framework, called DIAManD, for choosing the best interaction. We apply the framework to the SMARTedit programming by demonstration system and describe experimental validation and preliminary user feedback.

Keywords

Mixed initiative, machine learning applications, programming by demonstration

1. INTRODUCTION

Machine learning (ML) is widely used in areas such as wrapper induction, credit approval, image analysis, data mining, and intelligent user interfaces. A learning module can endow a system with the ability to extract valuable information from substantial amounts of data or to adapt to new circumstances. However, many traditional applications of machine learning involve their users — the domain experts who provide and label training examples — in only the most primitive ways. The systems operate on batches of prelabelled examples and neither provide nor request feedback during the learning process. Indeed, the only "communication" between the user and the learning system besides the provision of this batch of examples is to test the system on yet another batch of examples.

Yet, experience and research shows that great benefits can be realized by machine learning applications with rich user interfaces. Work in active learning has shown that just allowing the system to pick which example to label next can greatly reduce the number of examples the user need consider [5]. Bauer *et al.* describe how the inclusion of a variety of interaction modes in a wrapper generation system can result in more robust wrappers with less annoyance to the user [3]. Our previous work on the SMARTedit text-editing system [10] led us to the same conclusion: machine learning tasks can benefit from careful management of interactions with the user. The result is a system that requires fewer examples to learn a concept, acquires domain knowledge with less user effort, and is more appropriate for users unfamiliar with machine learning.

1.1 Vision

We view the process of training a machine learning system as analogous to the interaction between a teacher and student. The teacher guides the student toward a concept by presenting and explaining examples. Although the teacher ultimately controls the interaction, the student also takes the initiative, directing the teacher's attention toward areas of knowledge that are unclear.

Interaction between a user and a machine learning system should mimic the interaction between a teacher and a student. The user/teacher demonstrates examples of a concept to the system/student. Traditional machine learning has filled the role of student by listening quietly. However, this strategy may waste valuable resources — the teacher's time and effort — by withholding information from her about the student's level of understanding. We propose that machine learning systems should instead follow the example of the proactive yet considerate student: asking questions, proposing examples and solutions, and relating its level of knowledge when appropriate to make the interaction more

*To appear IUI'01.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IUI'01, January 14-17, 2001, Santa Fe, New Mexico.
Copyright 2001 ACM 1-58113-325-1/01/0001 ..$5.00

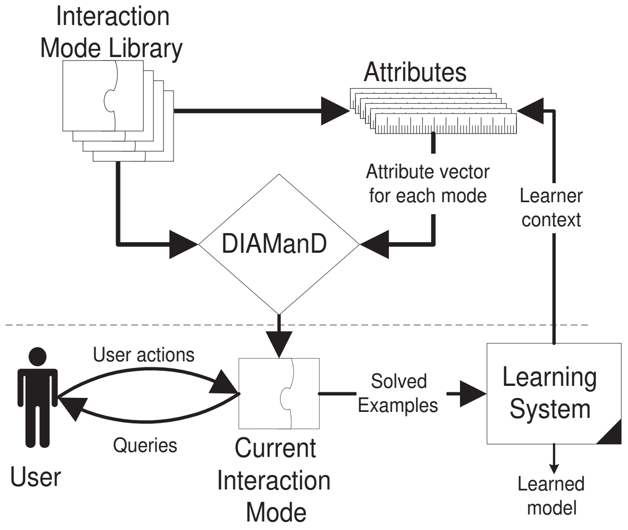

Figure 1: Diagram of the DIAManD interaction manager in use in a machine learning application. Traditional ML systems have only the bottom three components with a fixed interaction mode. DIAManD allows us to introduce a library of interaction modes and choose among them based on a set of attributes. The chosen interaction mode regulates the next stage of communication (*i.e.*, interaction) between the user and the learning system.

effective.

Such a mixed initiative interface is uniquely appropriate to machine learning systems because the user is directly interested in helping the underlying system. Indeed, the user's primary goal is for the system to refine its knowledge to the point that it has learned the correct concept and can aid her in her task. Thus, the user interface has a prerogative to act to maximize the value of the user's efforts to the learning system. The interface's guidance may take the form of informing the user of the learner's internal state in a comprehensible, domain-oriented manner, or of taking control of the discourse with the user to refocus her attention on new aspects of the learning process.

1.2 Overview

In the rest of this paper, we explore the issues that arise in constructing mixed initiative user interfaces for machine learning systems. We describe a general framework for interaction management in machine learning based on the common structure of most machine learning tasks: examples, labels or solutions, predictions, and data sets. In particular, we describe:

- a comprehensive list of classes of interaction modes for machine learning systems,
- a set of attributes for measuring aspects of the utility of these interaction modes,
- DIAManD: the **D**ecision-theoretic **I**nter**A**ction **Man**ager for **D**iscourse, a system for selecting among interaction modes using a multi-attribute utility function,
- a learning framework for adapting the weights of DIAManD's utility function to individual users, along with experiments validating the framework,
- a full implementation of DIAManD in the SMARTedit programming by demonstration system for text-editing,
- and preliminary user feedback on the SMARTedit implementation of DIAManD.

Figure 1 shows the high-level architecture of our system. DIAManD selects from among a set of interaction modes the one it judges most appropriate based on their attribute vectors. The best of these modes is presented to the user and controls the next stage of discourse, updating the state of the learner. The modes are then rescored based on the new state of the learner. Our implementation of DIAManD in SMARTedit currently includes six interaction modes, five attributes, the original SMARTedit learning system, and an adaptive utility function.

In the next section, we motivate the subject of user interfaces for machine learning with a brief case study of SMARTedit. We then present the top three blocks of Figure 1 in three sections: a list of classes of interaction modes, the DIAManD decision-theoretic framework for adaptively selecting among these modes, and an example set of attributes for use with DIAManD. The following section (Section 6) describes our implementation of the framework in the SMARTedit programming by demonstration system and discusses some preliminary user feedback. Finally, we conclude with our contributions and sketch out future lines of research building on this paper.

2. CASE STUDY: SMARTEDIT

One machine learning application that benefits from mixed initiative discourse is programming by demonstration, in which the user demonstrates to the system how to perform a task. The system treats this demonstration as a partial execution trace of the program that solves the task, and generalizes from the trace to the original program.

In prior work, we have formulated programming by demonstration (PBD) as an inductive learning problem [10, 11]. In our formulation, the learner is given the sequence of changes in the application state observed as the user performs some task on a concrete example. The learner then generalizes from these to a program that is capable of performing the task on new examples by learning the functions that transform from the initial application state to the desired final state. The actual learning mechanism involves maintaining a compact representation of a set of candidate hypotheses which is pared during learning to include only those hypotheses consistent with the observed task.

The user interface for PBD is similar to the regular macro recording interface: the user demonstrates the behavior of a program by recording how the program performs on a concrete example. In contrast to regular macros, however, the PBD system doesn't merely record a series of keystrokes. Instead, it *generalizes* from one or more demonstrations to a robust, executable program. After recording each demonstration, the user can ask the system to execute its learned program on the next example. In effect, the system has two interaction modes: recording the user's solution of the next example, and solving the next example using its learned program.

We have implemented this interface in the SMARTedit PBD system for text-editing. For example, a simple task for SMARTedit is to delete HTML comments from a text

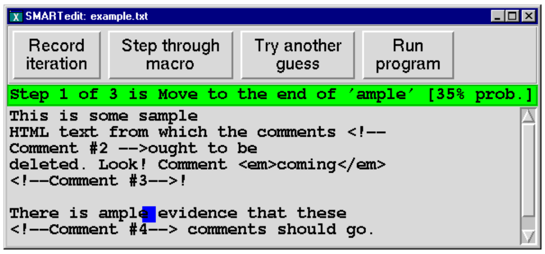

Figure 2: Screenshot of the original SMARTedit PBD system. The task is to delete all HTML comments. The first comment has already been deleted from just after the word "`sample`". The system now incorrectly predicts the user's next action is to move after the string "`ample`" rather than before "<!--".

file. A user demonstrates this task by starting the macro recorder, moving the cursor to the next comment, selecting the comment with the shift and cursor keys, and pressing the delete key to delete it (Figure 2). She then stops the macro recorder. After this demonstration, one of SMARTedit's candidate programs is a program consisting of three actions of the form "move to the next occurrence of <!--, select to the next occurrence of -->, and delete the selection."

2.1 Need for mixed-initiative interface

When we presented the interface to users, however, several problems were revealed. One user performed ten examples before asking the system to predict an action when, in fact, the system had learned the task correctly after the first two demonstrations. Another performed only one example before asking for the system's prediction. Because the system had not yet learned the right program, the user had to cycle through a large number of predicted actions before finding the correct one. A third user asked the system to begin predicting actions and then wished to return to demonstrating examples, but the system did not support returning to the recording mode.

These problems reflect on interactive machine learning applications as a whole. SMARTedit is an instance of a machine learning algorithm: it generalizes from solutions supplied by the user and learns a model explaining those solutions. The user interacts with the learning algorithm either by solving examples or by supervising the system's performance on an example. However, these are only two points in the space of possible interaction modes between the user and the learner. The problems revealed in our user tests suggest that other interaction modes could have benefits both in acquiring the concepts more quickly (e.g., through judicious choice of the example to classify, as in active learning) and in allowing the user more control over the learning process. The next section describes our space of interaction modes in more detail.

3. INTERACTION MODES FOR LEARNING

We have formed a comprehensive list of effective, general classes of interaction modes for machine learning. Each of these classes relies on only a few features common across learning domains and none assumes machine learning expertise on the part of the user. A supervised machine learning task includes distinct *examples* that the user *solves*, either by classifying them or by some more complex manipulation. Moreover, a *possibly-ordered set* of unsolved examples is either readily available or easily constructed. The learner searches through a *space* of *hypotheses* that explain the observed examples, and can *execute* one of these hypotheses on a new unsolved example to predict a *solution*.

Although users are generally familiar with the application domain (*e.g.*, text-editing), they may vary in their knowledge about the underlying machine learning system. While a machine learning expert might be able to peruse and correct the learner's individual hypotheses, we assume that users are not machine learning experts. This assumption is especially true in the case of adaptive user interfaces, where the users may not even be directly aware that they are using a machine learning system.

Our list of interaction mode classes includes:[1]

- *User solution*: In our simplest class, the standard for machine learning, the learning system observes an example and solution that the user provides and uses the result as input to the learning algorithm. SMARTedit uses an interaction mode in this class when the user provides solutions (or partial solutions) to examples by demonstrating her text-editing procedure on a section of text.
- *Collaborative solution*: The system executes a learned hypothesis on the next example and presents a solution to the user. The user either confirms the solution — in which case the example becomes training data for the system — or she asks for an alternate solution. The system can then propose an alternate solution to the user for acceptance or rejection, and so the process continues. For example, after a user demonstrates how to delete the first comment in Figure 2, she asks the system to predict the next action. Because the system incorrectly predicts a move to the word "`ample`", she might then choose to cycle to the next most likely guess. Members of these first two interaction classes were implemented in the original SMARTedit.
- *System solution*: In this class of interactions the learning system assumes slightly more control, executing its learned concept on the next example (or a part of it), demonstrating the solution to the user, and giving her a chance to reject that solution. If the user does not take this chance, the system assumes the solution was correct and carries on to the next example. The user need exert very little effort in this type of interaction as long as the system's learned concept is of high quality. Note that unlike in a collaborative solution mode, the user does not actually correct the system's predictions but rather indicates only *whether* the prediction is correct. Recalling the example of Figure 2, SMARTedit could run its learned program a step at a time; if the program were correct and the user did not interrupt, SMARTedit would proceed to delete all the remaining HTML comments in the file.
- *Performance*: In a performance interaction mode, the learning system takes full control and executes its concept autonomously on new examples. If implemented in SMARTedit, for example, performance mode might be useful once the system has learned a program to delete HTML comments. It can then proceed to delete all the

[1] As described in Section 6, we have implemented representatives from all these classes except the generation classes and user example selection.

comments in all of the HTML files in a directory. Moreover, this program might be applied to new files immediately or at a future date. Even in a performance mode, there is the potential that the system can make further progress in learning by studying the distribution of new examples; however, the system is not supervised by the user.

- *System example selection*: The system proposes to the user that she shift her attention to a particular example, rather than a user-selected example, a randomly-chosen example, or the next one in order. As related in the active learning literature, system-driven example selection can often provide greater learning benefit because the system can collect data on the most interesting examples first.
- *User example selection*: The system requests that the user select or provide a promising new example to investigate. Ideally, this request is accompanied with a high-level description of the kind of example that would be most valuable to operate on next. This class of interactions trades off increased control for the user with increased burden (the onus of selecting the example).
- *System example generation*: In this class the system generates a new example for the user to solve. In learning applications with relatively unstructured and unrestricted input, the system may be able to generate arbitrary examples. However, in more structured environments, modifying existing examples may be more effective than generating entirely novel examples.
- *User example generation*: Here, the system asks the user to create a new example which can then be solved. This type of interaction is valuable if the system can specify to the user some quality of the new example which would help to discriminate among contending hypotheses. For example, SMARTedit might request an example in which an open HTML comment is unmatched by a closing tag. If the user provides the example, the system can clarify its learned concepts. If, on the other hand, the user indicates that such examples are impossible or irrelevant, the system has learned information about the distribution of examples.

4. DECISION-THEORETIC FRAMEWORK

The interaction modes described in the previous section form the core primitives of a rich and comprehensible discourse with the user. However, we can't assume that the human user will effectively control the dialogue unassisted because she will likely find the complete set of options bewildering. This assumption would be especially hazardous since (1) we expect that many users will have no experience with machine learning algorithms, and (2) the choice of the most effective interaction depends crucially on the state of the ML system.

Instead, we adopt a mixed-initiative framework (DIAManD) in which the learner and human user are each participants in a dialogue aimed at improving the learner's hypothesis with minimal effort on the part of the user. The user and the learning system communicate through one of the various interaction modes. A set of attributes provides the means to discriminate among interaction modes; each attribute can provide a numerical evaluation in the range $[-1, 1]$ for a given interaction and state of the learning system. DIAManD scores the interactions using the attributes and, when appropriate, presents the best one to the user. The user may either proceed with this interaction or override the system to choose a different one.

DIAManD scores the interactions based on a decision-theoretic framework [17] (*i.e.*, using a utility model) which allows the learner to balance an interaction's expected burden to the user against its estimated value to the task and learner. However, we believe that each human user will perceive the utility of a given interaction in a given situation differently. Some users may prefer a learning system which is highly active, taking control of the discourse early at the possible expense of user corrections. Others would prefer a system that watches quietly until it is more certain that it understands the user's concept. To account for these differences and others, DIAManD adapts its utility function to the preferences of each individual user.

To render the problem of adapting DIAManD's utility function feasible in a reasonable number of interactions, we model the function by a linear combination of the weighted attributes. Thus, DIAManD models an individual user's preferences as a set of weights on the available attributes. Given these weights and an interaction, the interaction's score is the dot product of the weight vector and the vector of values of each attribute for the interaction.

To use the DIAManD framework (Figure 1), an application designer provides a *learning system*, a *set of modes* for interacting with this system, and a *set of attributes* which measure important aspects of these interaction modes. The core DIAManD component then analyzes the available interaction modes based on its utility function and chooses the mode with the highest utility to present to the user.

DIAManD also imposes two constraints on the user interface of a system it mediates. First, in order to support collaborative selection of the appropriate interaction, the user of the system should be able to override the system's choice of interaction mode and choose a mode that she prefers. In the SMARTedit DIAManD implementation, we accomplish this by presenting the available interactions to the user as a list of radio buttons (Figure 4). Second, to facilitate rapid learning, the interface should provide some mechanism for feedback to DIAManD on particularly poor interaction mode choices. This requirement is satisfied by the "Bad choice" button in Figure 4.

Given these interface capabilities, DIAManD must now codify the feedback from which it is to learn. We use a simple feedback model that would lend itself to a variety of actual learning algorithms. Each interaction receives a feedback value in the range $[-1, 1]$ that should represent an update to its correlation with the user's preferences: an undervalued "good" interaction would receive positive feedback whereas an overvalued interaction would receive negative feedback.

DIAManD gathers this feedback by two mechanisms based on utility values. (For the purposes of this discussion, let the utility of an interaction choice c be $U(c)$.) First, any time the user overrides its choice of interaction mode, DIAManD takes this as implicit negative feedback on its choice (c_s) and positive feedback on the user's choice (c_u). In response, it generates a positive feedback value for the user's choice equal to $U(c_u) - U(c_s)$. DIAManD also generates negative feedback on its own choice based on the interaction with the second highest utility ($c_{s'}$) equal to $U(c_{s'}) - U(c_s)$. Second, when the user rebukes a choice made by DIAManD, the system generates a strong negative feedback value for

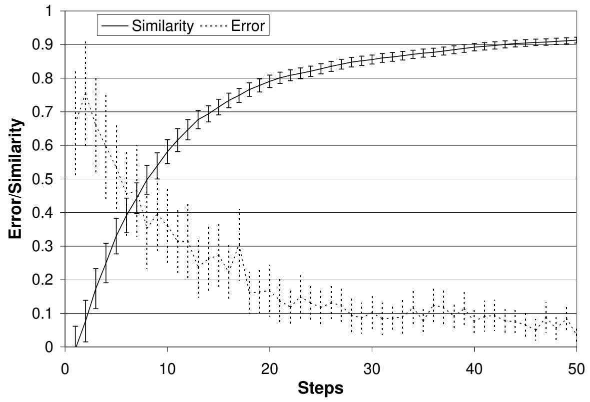

Figure 3: Error and similarity metrics averaged across the first 50 steps of 100 simulated DIAManD runs. There are fifteen attributes and eight choices in the domain. Confidence bars are set to 99% assuming a two-tailed Gaussian distribution of results.

that choice equal $U(c_{s'}) - U(c_s) - \rho$ where ρ is a parameter to the learning system.

Our learning mechanism for DIAManD responds to user feedback using an online reinforcement learning technique based loosely on the one described by Auer [2]. However, DIAManD performs no exploration — that is, it does not try apparently inferior interaction modes in order to solicit feedback on these choices.[2] Also, DIAManD allows feedback on more than one interaction at a time. The algorithm adjusts its weight vector by adding to it a scaled version of each interaction's attribute vectors. Each attribute vector is scaled by its interaction's feedback value and a uniform factor controlling the learning rate (as in gradient descent [15]).

Figure 3 shows the action of this learning system in an artificial domain. We have assumed that the domain has fifteen attributes and eight interaction modes (these numbers approximate those we hope to eventually achieve in SMARTedit). We model the user's preferences as a set of weights on the attributes; these are not revealed to DIAManD. We generate a random vector (with one entry for each attribute) by selecting values in the range $[-1, 1]$ for each element of the vector. For each run, a "user" is created by generating a random weight vector and scaling its length to 1.0. DIAManD's initial weight vector is also a random vector. At each step, a random attribute vector is constructed for each interaction. Based on their respective weight vectors, DIAManD and the user model each pick the best (*i.e.*, highest utility) interaction. If these choices differ, feedback is generated as if the user overrode DIAManD's choice in favor of her own. Moreover, if the user's utility for DIAManD's choice is less than 0, she issues a rebuke (*i.e.*, presses the "Bad choice" button). Finally, DIAManD updates its own model based on this feedback.

[2] An earlier version of DIAManD that did explore the space of interaction modes in this manner frustrated users by making the "wrong" choices for no apparent reason.

We measured two values (both averaged across 100 runs) at each step. The *error* at a step is the difference between the user's utility value for DIAManD's interaction choice and the user's utility for her own choice. The *similarity* between DIAManD's weight vector and the user's vector is the cosine of the angle between the vectors (the dot product of the two divided by their lengths). This value approaches 1.0 as the vectors become more and more similar.

Figure 3 shows that as long as the user's preferences really can be represented as a linear combination of the attributes, DIAManD will quickly learn the user's preferences. In this scenario, DIAManD's error is below 0.2 after only 18 interactions with the user — about one or two typical SMARTedit tasks.

We also tested variations on the number of attributes and interaction modes. The results indicate that more interactions make learning faster while more attributes slow learning; however, even with fifty attributes and only five choices, DIAManD's error level still drops well below 0.2 and its similarity level rises above 0.75 within 50 steps. Finally, starting DIAManD with a zero weight vector (rather than a random vector) makes little difference across all settings after only a few steps.

5. DIAManD ATTRIBUTE SET

The attribute set used in a DIAManD system defines the space of user adaptations available to that system. DIAManD can adapt to a user only if that particular user's preferences are expressible in the "basis set" defined by these attributes. Although the best attribute set for a specific DIAManD implementation will depend somewhat on both the learning system and interaction set used, the vital consideration that any attribute set must reflect is the balance between user effort and value to the task and system.[3].

For the SMARTedit implementation of DIAManD, we chose five appropriate but general attributes; each of these should be viable for most learning system and interaction library combinations. The first three (user input, level of continuity, and probability of correction) focus on user effort and represent physical and mental effort required from the user. The latter two (task progress and value to the system) focus on achievement of the user's objective. These measures reflect the typical objective of the user of a machine learning system: complete the task by refining the hypothesis of the learning system until it correctly describes the data. In the remainder of this section, we describe the five attributes used in our implementation.

- The *user input* attribute measures the difficulty of completing the interaction. For some interactions it may be possible to calculate the expected amount of input (*e.g.*, keystrokes and mouse clicks) necessary to complete the interaction. However, in the case of SMARTedit, we take a simple approach and hand-craft a function for each interaction which returns a rating between 0 and 1. The rating reflects the calculated possible courses of the interaction and our subjective estimate of the amount of input required for these potential courses. Note, however, that

[3] Ideally, attributes measuring user effort will draw on germane work in human-computer interaction such as task analysis [7] However, the initial work described here uses simple heuristics for these attributes.

any more complex or realistic scheme could be applied if its results could be mapped to the range $[-1, 1]$.

- The *level of continuity* between a new interaction and those preceding it measures the additional burden of some interactions on the user imposed by the cognitive context-switch required to change interaction modes. For example, if DIAManD moves the discourse to a performance mode directly from a user-solution mode, the sudden shift in initiative might confuse to the user. On the other hand, moving from collaborative-solution mode to performance mode will be less confusing because the change in initiative is not as great. In our SMARTedit DIAManD implementation we create a matrix of values between 0 and 1 that reflect our subjective estimation of the level of continuity between each pair of interactions. We track the most recently completed interaction mode and assign a value for this attribute based on the matrix entry for the last mode and the mode currently under consideration.

- The *probability of correction* is the probability that the user will be forced to correct an error by the system. This measure is used to reflect the amount of attention the user will be forced to pay to the system during an interaction. Typically, this attribute can be estimated using the learner's own confidence in its hypotheses. In the case of SMARTedit, we measure the value directly using its learning systems probability framework. Each interaction is simulated to determine the probability that the user will be forced to correct the system's prediction.

- The *task progress* achieved during an interaction is a measure of how much of the user's task is completed over the course of that interaction. In SMARTedit, we measure this value as the expected number of program actions that will be completed over the course of the interaction. This measure is mapped to the range $[-1, 1]$ by establishing an upper bound on the reported value and scaling.

- The *value to the system* of an interaction is the amount of improvement to or increase in confidence of the system's hypothesis space. To measure this change in quality of the hypothesis space, we note that a space with fewer distinct hypotheses is less "ambiguous" and more likely to favor the correct concept. In SMARTedit, we quantify this ambiguity by the entropy of hypothesis space: the amount of information the system would gain were it to ask the user: "Which of these hypotheses is the correct one?"[4]

  Note that each possible interaction mode implicitly asks a limited version of this broad question — by asking the user explicitly, displaying a proposed solution, or some other mechanism. So, each interaction divides the space of programs into disjoint subsets and discriminates among these subsets. For instance, recording an example divides the learning system's hypothesis space into the distinct subsets which predict each possible recording trace. Even before the demonstration, we can calculate these subsets. After the demonstration, we can point to the subset which explains the user's actions.

  Given this model, we can view the expected change in quality of the hypothesis space as the information gain of the interaction, I_G:

[4]For an introduction to entropy and information gain, see Chapter 3 of [15].

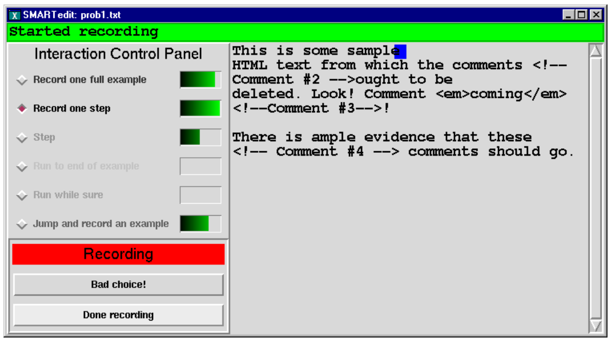

Figure 4: Screenshot of SMARTedit enhanced with DIAManD. The task, as in Figure 2, is to delete all HTML comments, and the first one has been deleted. The interactions are lined up on the left of the screen, and their scores are represented by the darkness of their fonts and the bar gauges to their right. The system has just selected the "Record one step" interaction, asking the user to demonstrate the next move. DIAManD selected this interaction over "Step" (the third down) because of the high probability of error in the "Step" interaction.

$$I_G = -\sum_{o \in O} (\sum_{c \in o} P(c)) \log(\sum_{c \in o} P(c))$$

Each o is a set of concepts (drawn from the universe O) which predict indistinguishable solutions on the current example, and $P(c)$ is the probability of concept c. As with the task progress, this measure is mapped to the range $[-1, 1]$ by truncating and scaling the range of entropy values.[5]

6. IMPLEMENTATION

We have implemented our framework in the context of the SMARTedit system. SMARTedit previously supported only two modes of interaction — one user solution and one collaborative solution — and had no intelligent interaction management. It was the user's responsibility to select the interaction mode and she made this choice without guidance. Moreover, it was impossible to switch back to the user solution mode from collaborative solution mode.

In order to introduce DIAManD's collaborative framework into SMARTedit, we followed the steps described in Section 4: provide a learning system, an interaction library, and an attribute set, and include in the interface the capacity to switch between interactions and rebuke a poor interaction choice.

Our learning system is the SMARTedit learning engine, based on Version Space Algebra [10]. Our five attributes are described in the Section 5. We also constructed a new library of six interaction modes.

To create the interaction library, we cleanly separated two interactions from SMARTedit's recording mode: one asks the user to record an entire example while the other asks

[5]This measurement is also used to compare the utility of demonstrating on different examples for the system selection interaction mode.

the user to record just a single action ("Record one full example" and "Record one step" in Figure 4). Each of these fits into the user solution class from Section 3. SMARTedit's ability to present a guess at the next action became "Step", a collaborative solution mode. We also introduced three entirely new interaction modes:

- "Run to end of example" successively presents the system's guesses for each step up to the end of the current example. After a brief pause to allow the user to interrupt, the system commits the guess. This interaction mode is from the system solution category.
- The "Run while sure" interaction immediately executes the learned program step by step until the system's confidence in the program at any step drops below a threshold (currently 99.9%). This interaction is a performance mode.
- Finally, we introduced system example selection in the form of the "Jump and record an example" mode. In this mode, the system repositions the cursor to just before an example that is particularly confusing to the system and asks the user to demonstrate that example.

SMARTedit's user interface was altered to display the interaction choices as a set of radio buttons. DIAManD's scores for the interactions are displayed by the contrast of the font and a horizontal gauge to the right of each interaction (as shown in Figure 4). Finally, the "Bad choice" button was added to allow users to rebuke DIAManD's choice.

The result of this implementation is a system capable of intelligently recommending which interaction to perform next. The user interface problems motivated in our case study are each addressed by the interaction manager. Users can tell when to stop recording examples because the system lowers the ranking of user solution mode interactions and instead recommends collaborative or system solution modes. Moreover, if a user does enter collaborative solution mode (or system solution mode) too early, she can now switch back to solving examples herself using the interaction manager.

We have not yet performed a formal user study focusing on DIAManD. However, we have received informal feedback on the SMARTedit implementation of DIAManD from collaborators and students in our department. Feedback from these users has motivated the current design of the user interface. For example, one user used SMARTedit to perform a task and felt he had trained the system correctly after a pair of examples. However, when he then had SMARTedit run in system solution mode[6] it made a mistake on one irregular example which went unnoticed. When he performed this task again with DIAManD, the interaction manager correctly recommended recording that irregular example, but the user failed to notice the change in recommendations and initiated the "Run" interaction anyway. To avoid this problem, weights are now represented in a format which admits easy comparison between options (the shaded "gauges" in Figure 4) and also by the attention-grabbing fading of fonts.

A similar process led to a new "User control" interaction in which the system declines to select an interaction automatically and asks the user to take the initiative. This interaction always receives a fixed utility rating. "User control" serves two purposes. First, it maintains the credibility of the system: DIAManD takes control and recommends an interaction only when it actually has a good one to propose. Second, it provides a "bar" toward which the user's choice is raised by the feedback mechanism. The effect of this extra positive feedback balances the highly negative feedback provided by the "Bad choice" button.

The qualitative feedback from users has so far been mixed. Some users are aggravated by the intrusion of DIAManD into control of the discourse. Others are happy to receive guidance and assistance. We believe that the existing DIAManD mechanism, with appropriate attributes measuring user tolerance of interruptions, can eventually accommodate both styles of users.

[6] The "Run" or system solution mode was so useful for DIAManD that we also introduced it into the plain SMARTedit system.

7. RELATED WORK

Several other lines of research have approached the problem of designing interfaces for machine learning in different ways — either by developing methodologies for learning interfaces or by creating interfaces for specific learning applications.

Work in active learning [1, 5] focuses on the "value to the system" attribute, ignoring the burden of interactions on the user. This focus requires the assumption that all interactions pose the same burden to the user, an assumption that may be reasonable in the face of a single mode of interaction. However, our work introduces numerous interaction modes, and distinguishing among these requires a richer set of attributes.

Boicu *et al.* [4] describe a framework for enabling domain experts to construct intelligent agents. Their framework selects among interactions to guide the user during exploration of a particular example. However, they do not address the problem of proactively selecting and moving among examples, nor are their interactions or interaction manager designed to generalize beyond their specific learning system.

Bauer and Dengler [3] describe a PBD system for wrapper induction. Their system performs heuristic search through the space of wrappers — programs that extract data from web pages. They describe a utility function specific to wrapper induction that can help users choose how to refine a wrapper. They introduce an attribute (the number of questions asked of the user) to help determine whether to bother the user with another interaction or use the current wrapper.

Previous work on adaptive user interfaces provided some mechanisms for collaborating to refine the learner's concept. Peridot [16] and Metamouse [14] are early PBD systems that request guidance from the user when generalizing actions. The "mail clerk" agent [13] learns by observing the user, from explicit feedback, and by being trained. In addition, it compares its confidence against two user-set thresholds ("tell-me" and "do-it") to decide whether to initiate action, make a suggestion, or remain quiet. None of these systems describes a general interface for machine learning, and none supports all the interaction modes presented herein.

Our work is partly inspired by recent work on mixed-initiative planning [8, 12]. For example, the TRAINS-95 system [9] describes a robust, multi-modal interface that treats the AI planner and the user as equal participants in the problem solving dialogue.

8. CONCLUSIONS AND FUTURE WORK

Machine learning applications are a fertile area for research into intelligent user interfaces. Users who are experts in their own domain are thrown together with complex systems whose internal functioning may be arcane to them, yet their goal in their interactions is to assist the system's learning process. Our paper makes the following contributions:

- We advocate a mixed-initiative interface in which the machine learner and human user more equally share responsibility for guiding the learning process.
- We define a comprehensive set of eight interaction modes which capture a wide variety of interaction types.
- We present a decision-theoretic framework in which the learner repeatedly chooses the interaction mode that maximizes expected utility.
- We define a multi-attribute utility function that balances direct user effort, cognitive load, and the cost of correcting computer-introduced errors, against the potential progress of the user and gain in the learner's "understanding."
- We describe our initial implementation of DIAManD in the context of the SMARTedit system for programming by demonstration.

There are a plethora of interesting directions for future work stemming from this paper. This interface model should be thoroughly tested through user studies. Applying the DIAManD framework to other domains will reveal new and challenging problems. Graphical programming by demonstration systems such as Eager [6] or Peridot [16] seem a natural next step, allowing us to explore collaborative example selection in a less constrained environment than the naturally sequential text domain. The DIAManD framework may also be applicable to choosing interaction modes in a general user interface context rather than solely for the interfaces of machine learning systems. Our current ongoing research involves introducing novel interaction modes (such as automatic example generation) and more realistic and effective attributes into DIAManD. Finally, different learning algorithms and more complex, non-linear utility functions might improve the efficacy of DIAManD's core learning component.

9. ACKNOWLEDGMENTS

We thank people who provided code, help, and discussion: Corin Anderson, Peter Auer, Jim Guerber, Geoff Hulten, Zachary Ives, Dutch Meyer, Denise Pinnel, Rachel Pottinger, and Kelly Shaw. This research was funded in part by the Office of Naval Research Grant N00014-98-1-0147, National Science Foundation Grants IRI-9303461 and IIS-9872128, the ARCS Foundation Barbara and Thomas Cable Fellowship, a National Science Foundation Graduate Fellowship, and a Microsoft Fellowship, and an NSF CAREER Award.

10. REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–42, 1987.

[2] Peter Auer. An improved on-line algorithm for learning linear evaluation functions. In *Proceedings of the Thirteenth Annual Conf. on Computational Learning Theory*. Morgan Kaufmann, June 2000.

[3] M. Bauer, D. Dengler, and G. Paul. Instructible information agents for web mining. In *Proceedings of the 2000 Conf. on Intelligent User Interfaces*, January 2000.

[4] M. Boicu, G. Tecuci, D. Marcu, M. Bowman, P. Shyr, F. Ciucu, and C. Levcovici. Disciple-coa: From agent programming to agent teaching. In *Proceedings of the Seventeenth Int'l. Conf. on Machine Learning*, June 2000.

[5] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[6] Allen Cypher. Eager: Programming repetitive tasks by demonstration. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, pages 205–217. MIT Press, Cambridge, MA, 1993.

[7] D. Diaper, editor. *Task Analysis for Human–Computer Interaction*. Ellis Horwood, 1989.

[8] G. Ferguson and J. Allen. Arguing about plans: Plan representation and reasoning in mixed-initiative planning. In *Proceedings of the Second Int'l. Conf. on Artificial Intelligence Planning Systems*, pages 43–48. Menlo Park, Calif.: AAAI Press, 1994.

[9] G. Ferguson, J. Allen, and B. Miller. TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Int'l. Conf. on Artificial Intelligence Planning Systems*, pages 70–77, Edinburgh, Scotland, May 1996. Menlo Park, Calif.: AAAI Press.

[10] Tessa Lau, Pedro Domingos, and Daniel S. Weld. Version space algebra and its application to programming by demonstration. In *Proceedings of the Seventeenth Int'l. Conf. on Machine Learning*, pages 527–534, June 2000.

[11] Tessa Lau and Daniel S. Weld. Programming by Demonstration: an Inductive Learning Formulation. In *Proceedings of the 1999 Int'l. Conf. on Intelligent User Interfaces*, pages 145–152, Redondo Beach, CA, USA, January 1999.

[12] N. Lesh, C. Rich, and C. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh Int'l. Conf. on User Modelling*, Banff, Canada, July 1999.

[13] Pattie Maes and Robyn Kozierok. Learning interface agents. In *Proceedings of the Fourteenth National Conf. on Artificial Intelligence*, pages 459–465, 1993.

[14] D. Maulsby and I. Witten. Metamouse: An Instructible Agent for Programming by Demonstration. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, pages 154–181. MIT Press, Cambridge, MA, 1993.

[15] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[16] Brad A. Myers. Peridot: Creating User Interfaces by Demonstration. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, pages 125–153. MIT Press, Cambridge, MA, 1993.

[17] Howard Raiffa. *Decision Analysis: Introductory Lectures on Choices Under Uncertainty*. Addison-Wesley, 1968.