PBD SYSTEMS: WHEN WILL THEY EVER LEARN?

Ian H. Witten*

*Computer Science, University of Waikato, Hamilton, New Zealand; ihw@waikato.ac.nz

Abstract. *We trace the tragicomic courtship between programming by demonstration and machine learning, two fields that seem made for each other but have never quite got together. A long-term historical perspective illuminates the twin roles of interaction and sequence learning in programming by demonstration, and reveals the parallel growth of machine learning as a research area in its own right. A view of the present shows a few—but only a few—intimations that these two fields are beginning to develop a meaningful relationship. The long-term prognosis is not so much wedded bliss as assimilation—ultimately PBD and ML shall, in the words of Genesis, cleave unto each other: and they shall be one flesh.*

Key Words. Programming by demonstration, Machine learning, Sequence learning, Instructability, MDL principle

The field of programming by demonstration (PBD) was born in a workshop held at Apple Computer in March 1992, and is effectively defined by the eighteen computer implementations documented in the book that resulted (Cypher, 1993). The workshop was actually on programming by *example*, whereas the book is titled programming by *demonstration*, and this difference highlights a basic tension. Are we talking about computer systems that are taught by giving them examples of what is to be done?—an example is a "fact or thing illustrating a general rule," and that seems to imply a strong reliance on *learning* from the examples that are given. Or are we talking about ones that accept a demonstration of how it should be done?—to demonstrate means to "describe and explain by help of specimens or experiments," which seems to place stress on *interaction* to communicate the steps that are to be undertaken.[1]

Of course, we want our systems to do both. We want them to acquire tasks from the user naturally and effortlessly. We want them to accept whatever the user chooses to provide: examples, demonstrations, advice, hints, instructions, constraints, assertions, specifications, execution traces, even programs. We want them to be resourceful: to employ domain knowledge whenever it is available, to solicit new information when appropriate, to interact with the user to clarify his or her intentions where necessary. The problem is not just to process each piece of information individually and operationalize it as a component of the task, but—far more difficult—to integrate different sources together, make sense of them, and work out when and how to clarify them by seeking more information, or more examples, or more direction. And, when the task is at least partly learned, to handle the transfer of control from user to machine, teacher to apprentice—and, when problems arise, back again—smoothly, elegantly, and naturally.

This takes us far beyond what is normally considered to be "machine learning." Fundamentally, PBD differs from ML in that it embodies a strong motivation to get the very most out of very small samples—it is just not acceptable to rely on statistical averaging and build systems that behave well only in the limit. If statistical methods border one side of ML, then PBD lies on the very opposite flank. PBD combats error by capitalizing on *interaction* with the user, not by processing large samples of data.

This article is about PBD and the role of machine learning. It provides a long-term historical perspective, beginning at the dawn of time and ending far in the future, with a brief tour of the present on the way. It's a personal view: I hereby apologize, and once only, for the emphasis on the work of my students and colleagues—it is only because that is what I know best.

1 The very beginning

It is easier for a tutor to command than to teach—John Locke, 1693

We trace the twin beginnings of PBD in interactive programming environments and sequential learning algorithms, two streams of activity that only recently began to meet. The renaissance of "conventional" ML had remarkably little effect on either: until recently, PBD seems to have been more concerned with commanding than with teaching.

Roots in interaction. PBD dates back to the early 1970s in what was then without doubt the Mecca of

[1] Definitions are from the *Concise Oxford Dictionary*.

highly-interactive computer systems, Xerox PARC. David Smith, working under the supervision of Alan Kay, who even at that time had the justly deserved reputation as the doyen—one might almost say the prophet—of interactive software, designed and implemented a "creative programming environment" that suggested a radical new approach to programming. Smith's 1975 thesis is a fascinating and literary work that spans the "two cultures" of art and science—it is a superlative model for a doctoral thesis. The first part, *Aspects of creative thinking*, comprises a wide-ranging discussion of thought, mental images, creativity, and computers; the second, *Pygmalion*, describes an "iconic programming system" which strove to do for the specification of procedures what graphical icons had done for the manipulation of objects. The fundamental problem was seen as one of *abstraction* and the issue of "learning" was not mentioned explicitly: PYGMALION's emphasis was "on *doing* rather than *telling*, ... [on] descriptions in terms of the *concrete* which PYGMALION turns into the *abstract*" (Smith, 1975).

Eventually the iconic approach to the human interface emerged in commercial form as the Xerox STAR computer of the early 1980s, the first incarnation of the windows-icons-mouse-pointer interface paradigm that is almost universal today (Smith *et al.*, 1982). Although Smith led the design team, PYGMALION had been left by the wayside: STAR lacked any facility for PBD. However, a 1981 project by Dan Halbert, Smith's student, added PBD to produce a prototype called SMALLSTAR, and Halbert's 1984 thesis, *Programming by example*, developed and amplified that work. SMALLSTAR followed the PYGMALION philosophy of employing interactive techniques to allow users to specify and manipulate programs in concrete terms. For example, generalizations were not induced from examples, but defined explicitly by having the user fill out "data description" property sheets.

Roots in sequence learning. At the same time as Smith was working on PYGMALION, John Andreae, who was something of a thorn in the flesh of artificial intelligence at the time, was working in New Zealand on interactive "learning machines," a topic that was distinctly unfashionable in AI circles (even ten years later it was derided by Charniak and McDermott in their *Introduction to Artificial Intelligence*). I visited Andreae in 1976 because his seminal research on what is now called reinforcement learning related to my just-completed thesis on networks of two-armed bandit controllers. He was finishing a book, *Thinking with the teachable machine*, that described PURR-PUSS, a program that was capable of being taught rudimentary operations such as counting and substitution, interactively through an ASCII terminal. As well as being teachable, PURR-PUSS was supposed to have a "mind of its own" and seek new experiences of its own volition: thus it was—intentionally!—not easy for teachers to impose their will on the direction of its learning.

Andreae's revolutionary book conveyed a radical new vision of artificial intelligence, replete with breathtaking anthropomorphic analogies, and met a predictably chilly reception from the AI community. Nevertheless it spawned a good deal of research on the identification of sequences, in other words, the inference of structural descriptions from a "behavior sequence" of discrete abstract tokens or "events." The question of inferring structure from behavior had been addressed, and solved, in the 1950s—but only for deterministic sequences. That is, given a sequence produced by a deterministic finite automaton, the structure of the automaton can be inferred (up to homomorphism). The drawback is the assumption of determinism, which is crucial; the slightest acausality in the sequence renders the inferred structure meaningless (Gaines, 1976a).

A decade earlier, Mark Gold (1967), in a paper that was later hailed as the first theoretical foundation of ML, had modeled the notion of language acquisition in terms of an informant who presented positive and, perhaps, negative examples of well-formed sentences to a learning machine. Using a simple enumerative argument, Gold showed that whereas the class of context-free languages is learnable if the informant presents negative examples, not even the regular languages are learnable from positive examples alone. A former colleague of Andreae's, Brian Gaines, implemented the enumeration as an "optimal causal modeler" that produced finite-state models of non-deterministic sequences by generating all models with a given number of states and choosing the best 1-state model, the best 2-state model, and so on, for the given sequence (Gaines, 1976b).[2] Clearly the procedure is exponential, but he carried it out typically to 10- or 12-state models for particular sequences, both natural and artificial (including one that represented observations of the grooming behavior of the house-fly). The results showed clearly and graphically the tradeoff between *model complexity*—the size of the model—and *goodness-of-fit*—the extent to which it corresponds to the observed data.

Enumeration is, of course, hopeless for large models. The alternative is to restrict the *context* of past events that are used for prediction. Witten (1979) investigated the transformation of fixed-context models into state models, while Andreae's student Cleary (1980) achieved a similar effect more simply with context-based models whose context length was allowed to vary up to a fixed maximum. These techniques turned out to be remarkably successful in predicting certain kinds of action sequence, and in the early 1980s they were employed for PBD in a predictive calculator and a predictive computer interface (Witten, 1981, 1982)—the latter is still used by disabled computer users (Darragh and Witten, 1992).

[2] Gaines called his system ATOM for "A tom-cat," a barbed reference to the feline PURR-PUSS.

From evaluation via compression to MDL. A problem underlying all the early work on behavior-sequence induction was one that is endemic to general-purpose learning systems: for any particular problem, a hand-crafted solution is bound to outperform anything that a "learning machine" might come up with. The virtue of learning is its generality, rather than its performance in any particular situation. But it was—and still is—very difficult to obtain suitable sources of data on which to evaluate alternative methods of sequence learning. Eventually we hit upon the idea of evaluating sequence induction methods by applying them to the compression of discrete sequences like text, which provided a huge source of test data—every file becomes an interesting challenge—and a clear, unambiguous performance metric. Once the technique of arithmetic coding became known in the early 1980s, it was a simple matter to transform a series of probabilistic predictions into a bitstream from which the original sequence could be regenerated. Surprisingly, although people had worked for years on compression algorithms, the variable-context method of sequence prediction turned into a compression scheme that far outperformed all others (Cleary and Witten, 1984)—and still does (Bell *et al.*, 1990; Cleary *et al.*, 1995).

The foray into compression was stimulated solely by a desire to evaluate sequence-induction methods quantitatively, and it proved very satisfying to work in an area which, unlike PBD, has an unequivocal, cut-and-dried performance metric. However, the linkage is much more fundamental. During the last decade it has emerged that compression provides a more satisfactory philosophical foundation for ML than does either Gold's (1967) model of inference or the more recent "probably approximately correct," or PAC-learning, criterion (Valiant, 1984). Both of these have proved relatively fruitless for practical PBD, although the former, with its "in the limit" orientation, is probably more apt than the latter, which assumes that input is characterized by probability distributions.

According to the *minimum description length principle* (MDL), of all "theories" (or "models," or "programs") that "explain" (or "account for," or "generate") a given set of observations, one should prefer the one that minimizes the sum of (a) the intrinsic complexity of the theory and (b) the amount of information needed to derive the original observations from it. The intrinsic complexity of a theory is, essentially, its size: in science one prefers small, parsimonious, theories. The information needed to derive the original observations is, essentially, the poorness-of-fit: we prefer accurate theories that account for as much as possible of the data presented. The tradeoff was noted above in reference to Gaines's enumerative inducer; the MDL twist is to simply add the two measures and use the sum as an overall figure of merit.

Machine learning. The renaissance of "conventional" machine learning within AI in the late 1970s and early 1980s had little effect on the development of PBD. Winston's landmark 1970 thesis on learning structural descriptions like that of an "arch" was clearly concerned with toy problems that could not possibly give any realistic help to a potential PBD user. Mitchell's formalization of the version space technique in his 1978 thesis seemed more promising—and, much later, its use was actually investigated in a PBD system (Mitrović, 1990). However, it did not seem to be applicable to problems on a realistic scale. In fact, although the version space method eliminates any dependence of what is learned on the order of presentation of examples, the time taken to assimilate each example is highly dependent on presentation order, and unfortunate ordering can effectively cause the internal search to blow up. Statistical techniques like Quinlan's ID3 (now C4) appeared in the early 1980s (Quinlan, 1983). These involved heuristic subdivision rather than search. However, they relied on far larger numbers of examples than was reasonable to assume in the PBD context.

The raw material for PBD is sequential—examples are sequences and inferences are programs—whereas ML invariably addressed non-sequential learning problems (and still does). The few exceptions, such as SPARC/E (Dietterich and Michalski, 1986) and TDAG (Laird and Saul, 1994), proved to be less than helpful to practical PBD. Just one ML project had a discernable influence on PBD. Andreae's son Peter performed doctoral research at MIT under Winston's direction, creating a bridge between the cultures of sequence learning and conventional AI. The result was NODDY, a robot programming system that acquired procedures from examples using an approach called "justified generalization" (Andreae, 1984) that eventually became the precursor of METAMOUSE (Maulsby 1988) and ETAR (Heise, 1989).

What about teaching? Right from the beginning, John Andreae's early work emphasized *teaching* just as much as *learning*. But it is hard to draw a firm line between teaching and programming. Here are some examples that illustrate the problem. First, it is normally viewed as the teacher's job to choose examples in an order that is helpful to the learner, and presentation order certainly has an effect on learnability. But this can be taken to extremes: Gold (1967) showed that descriptions unlearnable from positive and negative examples *can* be learned if the presentation sequence is sufficiently regular; moreover, positive examples alone suffice—yet the constraints on presentation order are so rigid that one would hardly call this "teaching." Second, Gaines (1976b) showed that the task of sorting can be "taught" by demonstrating the steps in an example sorting sequence to a learning automaton that is trivially simple to implement; yet the steps must be spelled out in such detail that no natural "teacher" would be capable of producing a correct demonstration. Third, Biermann (1972) demonstrated a "trainable Turing machine" in which the user supplied examples of the machine's input, output, and head movements during a computation, and the system would algorithmically create its own finite-state controller that handled a class of "similar" computations. Training this machine was tantamount to specifying a Turing machine for the desired task, something that most teachers would surely cavil at. Finally, PURR-PUSS itself can in principle be taught anything, for Andreae's student MacDonald (1984) demonstrated that it is possible to teach it to behave like a Turing machine, and thereafter "teach" it an arbitrary program.

The real distinction between teaching and programming is that a teacher does not use a formal model of the learner, whereas a programmer normally expects to know exactly how his instructions are going to be interpreted. In other words, a teacher adopts the *intentional stance* while a programmer adopts the *design stance* (Dennett, 1987).

The notion of a sympathetic teacher has been formalized in terms of "felicity conditions," which are constraints satisfied by a teacher that make learning better than from random examples (Van Lehn, 1983)—such as that examples should be classified correctly, the teacher should show all work, not use examples that contain misleading coincidences, introduce just one new feature per lesson, etc. While these conditions provide some design guidelines for PBD, they do not go very far. More recently, MacDonald (1991) defined a measure of instructability that expresses the *instructional complexity* of a task as a function of its inherent *task complexity.* The instructional complexity measures the difficulty of instruction, including any prerequisite training that is required. The more slowly it increases with task complexity, the more instructable the system.

2 The present state of the art

It is better to have wisdom without learning, than to have learning without wisdom—Charles Colton, 1825

We pause here to take stock of the present state of the art. As before, we commence with the twin strands of interaction and sequence learning, and then move on to the application of machine learning algorithms, the problem of feature selection, and the state of interactive learning in the ML field itself. We look at the question of embodying domain knowledge, and the state of the art in instructability. Finally we comment on implementation and explain why building operational PBD systems is *hard.*

Interaction. The highly-interactive nature of PBD that was pioneered at Xerox PARC in the 1970s has matured into an enduring emphasis on graphical interaction. Most PBD projects place great stress on graphics. As Cypher's (1993) book illustrates, the concept of PBD has come to embrace graphical user interface development environments as well as end-user applications, a tradition that began with Myers's (1988) PERIDOT. And of the end-user applications, the majority involve example-based graphical editing, programming through geometry, graphical histories, and so on. Potter (1993) even uses the pixels on the screen as a lowest-common-denominator graphical communication medium for all tasks, whether graphical or not.

Graphical interaction forms an appealing basis for PBD because it is direct, immediate, and supports reference to component parts by pointing. An even more tangible form of PBD is the "leading" that is used to program some industrial robots, for example, paint-sprayers, in which the trainer moves the robot's limbs through the desired motions and the robot repeats these motions later.

Sequence learning. A program is executed in a sequential series of steps, and part of PBD is the inference of program structure from observation of the steps. *Grammatical inference*, the construction of grammars from example sentences, is a standard—and difficult—problem in language acquisition. However, in PBD we have to deal with a single, continuous, unsegmented, behavior sequence rather than a set of sentences, and this renders standard grammatical inference techniques inapplicable. In fact, until recently the only practical way to infer a sequence's structure was to use one of the methods mentioned earlier: limited-context prediction on the one hand and the enumerative technique on the other.

However, two new developments are underway (Nevill-Manning, forthcoming). First, a method for inducing a deterministic context-free grammar for a sequence has been devised that is able to combine terminal symbols to produce higher-level symbols in a recursive manner, leading to a potentially complex, compact structural description of the sequence. Second, it has been shown that, given a sequence, a push-down finite-state automaton can be derived that is capable of recognizing branching and looping, as well as recursive and non-recursive procedure calls. The first technique provides a powerful way of compressing strings generated from formal grammars, and has been applied in computer graphics to natural-looking structures generated from recursive L-systems; it also forms the basis of a competitive text compression technique. However, it has the drawback that the slightest variation between two subsequences causes the algorithm to ignore their similarities. The second scheme, on the other hand, excels at recognizing branching, looping, and recursive structures, but is particularly sensitive to the level of abstraction of the symbols that it takes as input. A combination of the methods appears to be quite powerful, and is capable of finding structures which each method on its own overlooks.

Applying ML algorithms. Most current PBD projects do not employ techniques from machine learning, and those that do use *ad hoc*, domain-specific, generalization methods. A notable exception is Maulsby's CIMA (Maulsby, 1994), which bases its learning component on the PRISM algorithm (Cendrowska, 1987). PRISM constructs conjunctive rules, building up rule after rule until all the examples are covered. When a rule is being constructed, negative examples are excluded by conjoining new terms, one by one, until none are covered. The terms are chosen using a simple coverage heuristic—this differs from Quinlan's better-known C4, which uses an information-based metric to choose the best feature to add (Quinlan, 1993). However, PRISM is a relatively unsophisticated ML algorithm. Unlike most other similarity-based learning methods, it does not cope with "noise": noisy examples will cause uncontrolled overfitting to occur, resulting in a ruleset that is excessively detailed and tailored to the specific examples that have been seen. (There are extensions of PRISM that avoid this problem.)

In many PBD situations, the problem of noise is less prominent than it is for general ML. Noise sometimes stems from genuine non-determinism in the data—and this is a particular issue in PBD for robot programming. In other contexts, it may be caused by an inadequate set of attributes. Another source is teacher error: teachers must be able to recover from mistakes easily. Rather than treating noise as inherent and coping with it using probabilistic techniques, it may be worth seeking, and correcting, inadequacies in the description language used. Although a model that makes non-deterministic predictions is fine for some purposes (such as compression), it is less useful for the kind of prediction that a PBD system must undertake. Sometimes it is more productive to strive to eliminate noise by adding new attributes than to put up with it as a necessary evil.

The traditional ML technique for dealing with inadequate description languages is *constructive induction*, where new attributes are created as functions of old ones. The aim is to transform the original representation into a space in which the examples exhibit more regularity. Typical operators include quantizing numerical attributes, conjoining nominal ones, tallying how many of a given set of boolean attributes are true, and comparing ($=$, $\geq$, etc) different attributes.

Even without constructing new attributes, there is invariably a host of features that may be relevant to any particular decision. This leads to a problem of feature selection or, to use the accepted ML term, "bias": how can one determine a set of likely candidate features and expand that set when it proves inadequate—that is, when non-determinism appears? Although there have been some studies of how to detect changes of bias, these do not seem to have led to useful general results or techniques. However, in PBD, with its emphasis on interactive learning, the selection of an appropriate bias becomes an ideal point at which to involve the teacher.

Interactive ML techniques. Little research in ML has addressed interactive learning, which is surprising because learning systems have a great deal to gain by showing more initiative—in particular, by actively posing test examples rather than passively waiting for more examples to appear. An early system, MARVIN, learns concepts by asking questions (Sammut and Banerji, 1986). The teacher begins by presenting an example of the desired concept, whereupon Marvin begins to ask questions to eliminate possible hypotheses. While learning a concept, it modifies its current hypothesis by *generalization* and *specialization* transforms until it converges on the target concept. While this is an appealing model, in practice MARVIN asks a huge number of apparently extremely trivial questions that would certainly not be tolerated by any serious user. Moreover, it presupposes a hierarchically-structured network of concepts and breaks down in more general partially-ordered domains. This can be ameliorated by more sophisticated techniques, and the number of unproductive or inappropriate questions can be reduced (Krawchuk and Witten, 1988).

De Raedt's CLINT is a more recent interactive learning system which speeds up the questioning process by permitting the teacher to classify features as relevant or irrelevant (de Raedt, 1991). Since it has no means of deciding which features to ask about, it invites the teacher to examine the current concept description and classify features it has proposed.

CLINT gives the teacher formal control over the learner's bias. CIMA provides a more informal mechanism by allowing teachers to give linguistic "hints" that are interpreted in terms of changes of bias (Maulsby, 1994). These hints are not processed by a comprehensive natural-language understanding system; to do so would be an interesting and potentially productive exercise. Instead, the user's utterances are merely scanned for keywords that appear in a lexicon along with the relevant feature that each keyword relates to. The appearance of a keyword in a hint is interpreted as a request to augment the learner's bias by including the related feature. This nicely circumvents the chief difficulty of natural language processing, that it is easy to completely misinterpret an utterance (for example, by ignoring a word like "not" that inverts its meaning). The inclusion of a feature in the learner's bias does not imply any commitment as to *how* it will be used: that will be determined by the examples which are processed by the learner. For example, if a feature is used in a negative sense, in that the required condition obtains only when it is *not* present, that will be determined quite naturally by the induction component from the examples given. It does not matter if the keyword spotter fails to notice the sense in which the term is used—perhaps, for example, the word "absent" is missing from its lexicon. The use of verbal hints to focus learning from examples is a powerful general idea that is likely to see widespread applicability in the future.

Embodying domain knowledge. Wherever possible, PBD must take advantage of domain knowledge to increase the leverage of the examples supplied, and to ensure that the concepts learned are *operational* in the sense that they can be used to make specific, useable, predictions. But domain knowledge is unlikely to take the form of a "deep" theory—say a scientific or mathematical one—of the kind used in classical explanation-based learning. Rather, it will be expressed as some sort of classification scheme that supports appropriate actions and inferences. We call such schemes "microtheories": they tend to be heuristic in nature and are difficult to defend in any way other than the extent to which they expedite learning.

One example is the theory of constraints that underlies the METAMOUSE system, a demonstrational interface for graphical editing tasks within a drawing program (Maulsby, 1988). Let us examine this more closely, not to extol its virtues as a constraint classification scheme, for it is *ad hoc* and hard to justify from any theoretical perspective, but rather to expose the nature of this microtheory from which METAMOUSE gains much of its power. In METAMOUSE, the term *constraint* is used to mean "a spatial relation of special interest." For instance, if a horizontal construction line is moved upward to coincide with the bottom of a box on the screen, a variety of touch relations are observed: the line touches the box's lower-left corner (line-to-point), perhaps its center coincides with the lower-right corner (point-on-point), and maybe the line intersects another line elsewhere on the screen (line-to-line). Each of these corresponds to a constraint, and is assigned a level of significance. A *determining* constraint is one that leaves no free variables (e.g. point-on-point); *weak-1* and *weak-2* constraints leave one (e.g. line-to-point) and two (e.g. line-to-line) degrees of freedom respectively; while *trivial* constraints are ones that hold necessarily. For METAMOUSE, constraint-solving is the process by which predicted program steps are realized as actions with specific values for object variables, and so constraint classification forms the key to generalizing actions.

The taxonomy of actions in CIMA is another microtheory (Maulsby and Witten, 1995). There are four types: *classify* actions, which (merely) discriminate between positive and negative examples; *find* actions, for which the descriptions must delimit objects and the direction of search; *generate* actions, which specify all features of a new object; and *modify* actions, which not only discriminate between positive and negative, but also determine the property's new value. Again, the point is that this microtheory forms the basis of operationality testing in CIMA.

Instructability. The most ambitious vision of an instructable system that appears to be implementable with today's technology is Maulsby's TURVY, and its partial incarnation in the CIMA learning algorithm. This is based on a simple model—or microtheory—of instruction that postulates three entities that can be communicated between teacher and system: *examples*, *rules* and *features*. Each of these can be classified by the teacher as positive or negative. Examples are classified with respect to some concept: this is the usual "membership query" instruction supported by supervised learners. Rules are judged to be valid or not, an operation that is widely adopted in systems that learn from an informant.

Features are more unusual, for they affect the learner's bias rather than judging a particular example or rule. Formally, an attribute (e.g. *color*) or value (e.g. *color(red)*) is classified as relevant or irrelevant to some subset of examples. A complete classify-feature instruction would specify a particular attribute value, the concept and disjunct (current rule), and whether it is relevant or irrelevant. But the system is capable of interpreting ambiguous, incomplete "hints." A hint may map to several classify-feature 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, the latter involving pointing at objects to indicate whether they are relevant. As explained earlier, verbal hints are interpreted by simple keyword-spotting.

Implementation. In our experience, the implementation of a highly reactive system that employs search-based AI techniques and works robustly with casual users is a very demanding undertaking. For example, METAMOUSE was originally constructed in Lisp in the form of an experimental prototype, then rewritten in C++ and Motif. Reimplementation proved to be a far more difficult undertaking than originally anticipated. The combination of interactive graphics, constraint manipulation, and inductive inference made METAMOUSE an intricate and delicate program to work with. We observed with regret that our test users encountered many new bugs in the system, which supposedly had been thoroughly exercised.

METAMOUSE's tool-based metaphor intentionally encourages users to behave creatively by discovering novel ways to teach and novel constructions to "explain" aspects of what is taught. The system applies inferential methods to make, and execute, generalizations about the behavior. The searching that is involved means that when unanticipated situations arise they often lead to crashes. Constructing and debugging systems of this nature is intrinsically difficult. The conclusion of the METAMOUSE experience was that simulated systems should be tested extensively before a full implementation is attempted; this philosophy has been followed in TURVY and related projects (Maulsby, 1994).

Recent trends in software technology, such as AppleEvents and Microsoft's WordBasic, permit increased program access to the internal operations of application programs using scripting and recording techniques. There is no question that METAMOUSE would be far easier to implement today if it were based on an existing graphics editor that was AppleEvent-aware. Nonetheless, many of the problems we encountered were attributable to the amalgamation of AI-style searching with creative interactive operation, and these would remain even with the new technology.

3 And the future . . .

Experience is the only teacher—Emerson, 1845

Eventually, machine learning will disappear—not in the sense of being wiped out, or forgotten, but in the sense of becoming invisible, not identifiable as a "discipline." Techniques of ML will be absorbed into application areas, just as techniques of AI—list processing from the 60s, searching from the early 70s, rule-based programming from the late 70s, object-orientation from the early 80s—have become absorbed into mainstream computer science. Already, some ML is disappearing into database mining; rule induction from examples has become commonplace; and ML toolkits are emphasizing integrated environments for data exploration rather than individual techniques.

Let me speculate (yes, self-indulgently) about four areas of future concern to PBD.

Communication. PBD will go multimodal. Different modes of expression—verbal, visual, gestural or deictic, even tactile—will help to widen the channel of communication through which computers are instructed. Already there are PBD systems that use these modes (including tactile communication for robot "leading," e.g. Heise, 1989). Research in combining modes will burgeon.

More distantly, PBD researchers will look at theories of conversation, such as Austin's view of language as action, Searle's speech act theory, and Grice's conversational postulates, to help model communication. They will rediscover plan inference from AI and implement agents that infer their user's plans. They will begin to wonder whether agents are conscious—or should be.

Instructability. Far richer models of interaction will be developed. Microtheories of instruction, such as CIMA's, will be extended to deal with a wider variety of information exchanged between user and system, and a more comprehensive account of how ambiguous "hints" relating to this information can be interpreted and acted on.

New models of interaction will classify different types of question, such as ones designed to debug concepts, ones that challenge the teacher's accuracy, and ones that test a hypothesis that the learner has formed. Even theoretical ML has considered different query types—not just membership but also equivalence, existential, subset, superset, disjointness, exhaustiveness—but these do not seem particularly germane to PBD.

PBD researchers will address issues such as when systems should ask questions, how questions should be formulated, how the most appropriate question can be chosen. They will ponder the automation of pedagogy.

Interactive software engineering. One class of ML applications is to the automated production of software; from similarity-based induction of rules for expert systems, through inductive generation of logic programs, to the derivation of operational models ("skills") from deep models ("understanding") using explanation-based learning.

If ML strives to automate the production of software, PBD goes beyond by attempting to automate the *interactive* production of software—and far beyond by employing, as teachers, end-users with no programming knowledge. This will be seen as one solution to the problem of custom software—too many applications needed and not enough specialists to write them.

The question of bugs in programs generated by demonstration will arise. While academics contemplate issues of reliability of artificial agents, perhaps using theoretical models of trust (Luhmann, 1979), end-users will simply cope with bugs on a pragmatic basis, just as secretaries cope with bugs in word-processors at present. Of course, damage-control mechanisms will be put in place to limit the effect of bugs in PBD-generated software.

PBD systems will perform automatic sanity checks on the programs they generate, and these will become ever more sophisticated and comprehensive as the effective bandwidth between user and system increases—as they come to understand each other better. Eventually, interactive proof techniques for software quality assurance will be integrated with interactive programming environments, so that as a program is generated by demonstration, it is accompanied by a proof of correctness.

Theoretical foundations. The MDL principle will emerge as the underlying theoretical foundation of PBD. This is because, unlike statistical (e.g. t-tests) and probabilistic (e.g. PAC) measures, it is applicable even when the number of examples is very small. However, MDL tacitly assumes that the effect of the particular language in which the theory is couched can be eliminated by compressing all information by an entropy coder before it is measured—providing links to Kolmogorov complexity on the one hand and Bayesian inference on the other. Unfortunately, this tactic works only in the limit for large theories. For small theories, the language of expression constitutes an unavoidable biasing factor and care must be taken to ensure that the language is appropriate.

MDL applies to situations where a theory is generated from the examples prior to measuring its complexity and goodness-of-fit. This "off-line" orientation does not reflect for the on-line, continually adaptive, realities of PBD. Adaptive forms of MDL will be devised to overcome this deficiency. Eventually, the dynamic, adaptive, realm will become the norm for theoretical development learning, and theories that address static snapshots will seem anachronistic.

4 When will they ever learn?

A little learning is a dangerous thing—Alexander Pope, 1711

A teacher is one who enables learning: by creating a suitable environment, by demonstrating, by directing attention, by supplying motivation, by inspiring. ML is in danger of becoming bogged down in an insipid kind of learning. We train our ML systems like we train chickens, or plants, by accustomization: endless drill and practice. Teaching, in contrast, is interactive and relies on creative interplay between the participants. PBD is about interaction through multiple channels of communication, about showing and leading, language, looking and listening. And a little learning.

Acknowledgments

In preparing this paper, I benefited enormously from the work of all my graduate students, in particular Dave Maulsby and Craig Nevill-Manning. The Natural Sciences and Engineering Research Council of Canada and the New Zealand Foundation for Research in Science and Technology have both supported my work at times over the past several years.

References

Andreae, J.H. (1977) *Thinking with the teachable machine*. Academic Press, London, England.

Andreae, P.M. (1984) *Justified generalization: acquiring procedures from examples.* PhD thesis, MIT.

Bell, T.C., Cleary, J.G. and Witten, I.H. (1990) *Text compression.* Prentice Hall, Englewood Cliffs, NJ.

Biermann, A.W. (1972) "On the inference of Turing machines from sample computations," *Artificial Intelligence 3*: 181–198.

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

Cleary, J.G. (1980) *An associative and impressible computer.* PhD thesis, University of Canterbury, Christchurch, New Zealand.

Cleary, J.G. and Witten, I.H. (1984) "Data compression using adaptive coding and partial string matching," *IEEE Trans Communications COM-32*(4): 396-402.

Cleary, J.G., Teahan, W.J. and Witten, I.H. (1995) "Unbounded length contexts for PPM," *Proc Data Compression Conference*, Snowbird, Utah.

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

Darragh, J.J. and Witten, I.H. (1992) *The reactive keyboard*. Cambridge University Press, Cambridge, England.

de Raedt, L. (1991) *Interactive concept-learning.* PhD thesis, Katholieke Universiteit Leuven, Belgium.

Dennett, D.C. (1987) *The intentional stance*. MIT Press, Cambridge, MA.

Dietterich, T.G. and Michalski, R.S. (1986) "Learning to predict sequences." In *Machine learning: an artificial intelligence approach II*, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, 63–106. Morgan Kaufmann, Los Altos, CA.

Gaines, B.R. (1976a) "On a danger in the assumption of causality," *IEEE Trans Systems, Man and Cybernetics SMC-6*: 56–59.

Gaines, B.R. (1976b) "Behaviour/structure transformations under uncertainty," *Int J Man-Machine Studies 8*: 337–365.

Gold, E.M. (1967) "Language identification in the limit," *Information and Control 10*: 447–474.

Halbert, D.C. (1981) *An example of programming by example*. Technical Report, Xerox PARC, Palo Alto, CA.

Halbert, D.C. (1984) *Programming by example*. Technical Report, Xerox PARC, Palo Alto, CA.

Heise, R. (1989) *Demonstration instead of programming: focusing attention in robot task acquisition.* MSc thesis, University of Calgary, Canada.

Krawchuk, B.J. and Witten, I.H. (1988) "On asking the right questions." *Proc Fifth International Conference on Machine Learning*: 15–21. Ann Arbor, Michigan.

Laird, P. and Saul, R. (1994) "Discrete sequence prediction and its applications," *Machine Learning 51*(1): 43–68; April.

Luhmann, N. (1979) *Trust and power.* Wiley and Sons, Chichester, England.

MacDonald, B.A. (1984) *Designing teachable robots.* PhD thesis, University of Canterbury, Christchurch, New Zealand.

MacDonald, B.A. (1991) "Instructable systems," *Knowledge acquisition 3*: 381-420; December.

Maulsby, D. (1988) *Inducing procedures interactively.* MSc thesis, University of Calgary, Canada.

Maulsby, D. (1994) *Instructible agents.* PhD thesis, University of Calgary, Canada.

Maulsby, D. and Witten, I.H. (1995) *Interactive concept learning for end-user applications.* Working Paper 95/4, Department of Computer Science, University of Waikato, New Zealand.

Mitchell, T.M. (1978) *Version spaces: an approach to concept learning.* PhD thesis, Stanford University.

Mitrović, A. (1990) *Interaktivno indukovanje procedura primenon tehnika mašinskog učenja na osnove sličnosti u primerima.* MSc thesis, University of Niš, Yugoslavia.

Myers, B.A. (1988) *Creating user interfaces by demonstration.* Academic Press, Boston.

Nevill-Manning, C.G. (forthcoming) *Description, prediction, production and compression of sequences.* DPhil thesis, University of Waikato, New Zealand.

Potter, R. (1993) "Triggers: guiding automation with pixels to achieve data access," in Cypher (1993), 361–380.

Quinlan, J.R. (1983) "Learning efficient classification procedures and their application to chess end games." In *Machine learning:*

an artificial intelligence approach, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, 463–482. Tioga, Palo Alto, CA.

Quinlan. J.R. (1993) *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, CA.

Sammut, C. and Banerji, R. (1986) "Learning concepts by asking questions." In *Machine learning: an artificial intelligence approach II*, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, 167–191. Morgan Kaufmann, Los Altos, CA.

Smith, D.C. (1975) *Pygmalion: A creative programming environment.* PhD thesis, Stanford University.

Smith, D.C., Irby, C., Kimball, R., Verplank, B. and Harslem, E. (1982) "Designing the Star user interface," *Byte 7*(4): 242–282.

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

Van Lehn, K. (1983) *Felicity conditions for human skill acquisition: validating an AI-based theory*. Research Report CIS-21, Xerox PARC, Palo Alto, CA.

Winston, P.H.. (1970) *Learning structural descriptions from examples.* PhD thesis, MIT.

Witten, I.H. (1979) "Approximate, non-deterministic modelling of behaviour sequences," *Int J General Systems 5*: 1–12.

Witten, I.H. (1981) "Programming by example for the casual user: a case study," *Proc Canadian Man-Computer Communication Conference*: 105–113. Waterloo, Ontario.

Witten, I.H. (1982) "An interactive computer terminal interface which predicts user entries," *Proc IEE Conference on Man-Machine Interaction*: 1–5. Manchester, England.