The Knowledge Engineering Review, Vol. 19:1, 61–88. © 2004, Cambridge University Press
doi:10.1017/S0269888904000141 Printed in the United Kingdom

Can user models be learned at all? Inherent problems in machine learning for user modelling

MARTIN E. MÜLLER

Department of Computer Science, University of Augsburg, D-86159 Augsburg, Germany;
e-mail: martin.e.mueller@informatik.uni-augsburg.de

Abstract

Machine learning seems to offer the solution to many problems in user modelling. However, one tends to run into similar problems each time one tries to apply out-of-the-box solutions to machine learning.

This article closely relates the user modelling problem to the machine learning problem. It explicates some inherent dilemmas that are likely to be overlooked when applying machine learning algorithms in user modelling. Some examples illustrate how specific approaches deliver satisfying results and discuss underlying assumptions on the domain or how learned hypotheses relate to the requirements on the user model. Finally, some new or underestimated approaches offering promising perspectives in combined systems are discussed. The article concludes with a tentative ''checklist'' that one might like to consider when planning to apply machine learning to user modelling techniques.

1 Introduction

In Webb *et al.* (2001), the authors pointed out several problems that are likely to be overseen when trying to learn user models. The motivation was that:

> At first blush, user modelling appears to be a prime candidate for straightforward application of standard machine learning techniques.

This apparently obvious observation resulted in a boom in research on machine learning (ML) for user modelling (UM). ML and UM have also moved into the focus of interest of many industrial applications and, as a consequence, ML for UM plays an increasingly important role.

However, the application of a variety of ML algorithms implies certain restrictions on the domain, and issues of human–computer interaction (HCI) have certain implications on the data available for learning (some of which have been discussed in Webb *et al.* (2001)). In this article, we present a detailed description of what both approaches (ML and UM) require from and can deliver to each other. In this way, we explicate problems that are likely to occur. We present several working approaches to solving detailed problems and we close with a general discussion of the title question including a ''checklist'' of what requirements one needs to meet in order to circumvent hidden pitfalls.

1.1 Learning about the user

Adaptive user interfaces (AUIs) adapt themselves to the user by reasoning about the user and refining their internal model to the user's needs. The refinement of the user model consists of a

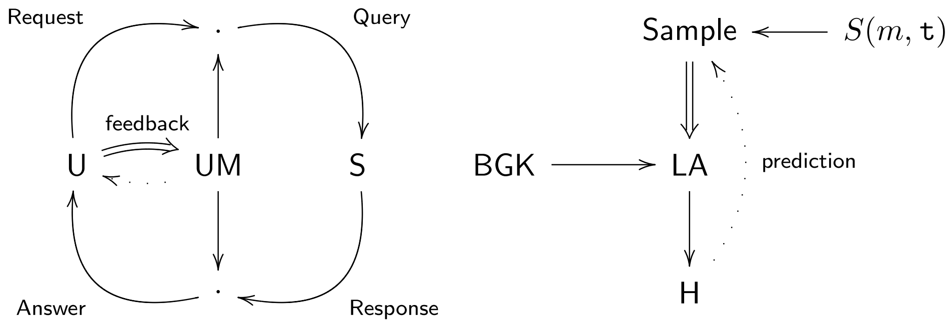

Figure 1 An abstract AUI and an abstract ML system

sequence of inferences based upon several observations of user interactions, commonly known as feedback (already introduced to the information retrieval community by van Rijsbergen (1979)). This yields an abstract architecture as shown in Figure 1. A user's information request is translated into a system query by taking into account knowledge about the domain (i.e. the system S) and knowledge about the user (taken from the internal user model UM). The system's response is transformed (by filtering, arrangement, aggregation, etc.) into an answer (again taking into account the user model UM) that is presented to the user. The user interactions (including explicit feedback) are used to refine the user model in order to be able to deliver more precise results next time.

In ML, artificial systems learn how to perform better through experience. By observing examples from a Sample, the learning algorithm LA tries to induce a hypothesis H that approximates a general, unknown law that explains the nature of all objects of the domain (both observed and unknown). This inductive inference is supported by a set of background knowledge BGK. As long as the hypothesis and according predictions disagree with observations, the hypothesis needs to be refined. This process is depicted on the right-hand side of Figure 1.

It seems obvious that ML can offer what is needed as a feedback loop in user adaptive systems.

1.2 The problem remains the same

However, when trying to adapt by learning one will sooner or later encounter one or more well-known problems (some of which have been discussed in Webb *et al.* (2001)).

1. There is only very little feedback available, so learning needs to yield results from very few examples only.
2. Quite often, there is no negative feedback so learning has to make do with positive evidence only.
3. Learning of content-based user models requires a sophisticated domain theory and, thus, a lot of background knowledge.
4. Learning in collaborative UM sometimes results in strange user models that cannot be explained in terms of domain knowledge.
5. Learning in the presence of very large datasets takes a reasonable amount of time, which is undesirable in many applications where a quick adaptation is needed.

Recently it seems that emphasis has mostly been put on the latter issue for two reasons. First, growing interest in industry for applicable methods that promise an advantage on the market requires quick adaptation and fast results. Second, growing computational power and standards for markup languages, which allow meta data to be represented and interchanged, might lead to the conclusion that many problems our now within reach of tractability.

On the other hand, it must be stressed, that HCI can be characterised by the following observations.

- Users are not generally willing to give feedback;
  $\rightarrow$ there are only few (positive only) examples.

- Users need to be observed unobtrusively, which requires a system to interpret interactive events as feedback;
  → samples are not reliable and noisy.
- Users need to feel in charge and understand the system;
  → hypotheses (that is, user models) need to be understandable and the system behaviour must be explained to the user.
- Users are only willing to spend extra effort on something if there is an immediate significant recognisable benefit;
  → learning must be fast and failsafe.

In many research projects, great results were achieved in solving one or more of these problems. The overall dilemma remains: there does not seem to be a system that learns quickly, is highly accurate, is nearly domain independent, does this from few examples with literally no bias, and delivers a user model that is understandable and contains breaking news about the user's characteristics. Each single problem favours a certain learning approach. A more generic approach to UM (although not primarily focusing on issues related to ML) was presented in Kobsa (2001).

This paper describes the problem of using ML methods in UM and presents a few promising approaches. The conclusion presents and discusses alternatives for future research directions in ML for UM that focus on the question of how UM can be interpreted in a way that allows for a better learnability of user models.

2 Understanding UM as a ML problem

It takes a reasonable amount of intelligence to learn about the environment. In general, learning is characterised as a generalisation process that enables the learner to induce intentional knowledge (such as rules or concepts) from examples. The newly acquired knowledge needs to explain observations from the past and will also help to recognise new observations. In other words, an initial model is refined by several observations such that the new theory is modified in a way that includes all known pieces of evidence, as well as many as possible unseen examples (and, at the same time, excluding as many as possible counterexamples).

2.1 What makes a learning algorithm?

Let $\mathfrak{U}$ denote the *universe*; that is, the set of all objects of the domain. Any subset of $\mathfrak{U}$ may form a *concept*. The goal of a learning algorithm $\mathbf{A}$ is to learn a hypothesis h that accurately describes a (unknown) *target concept* $c_{\mathbf{t}}$.

The target is characterised by a function $\mathbf{t}$, which (in the ideal case) delivers a noise-free binary classification on any object $x \in \mathfrak{U}$: $x \in c_{\mathbf{t}}$ if and only if $\mathbf{t}(x)=1$. A *sample* is a sequence of *examples* and an example is an object of our domain together with its label as assigned by $\mathbf{t}$. Accordingly, a sample consisting of m examples is a sequence delivered by a sampling function

$$S(m, \mathbf{t})=\mathbf{s}=[\langle x_1, \mathbf{t}(x_1)\rangle, \langle x_2, \mathbf{t}(x_3)\rangle, \ldots, \langle x_m, \mathbf{t}(x_m)\rangle]. \tag{1}$$

The problem is that the probability of the occurrence of objects of the domain is not uniformly distributed.

Example. Consider the problem of predicting keystrokes. Each letter has a different probability and, depending on the language, letter sequences also have different probabilities. Speaking in metaphors, we usually do not know the probability of a single letter or even the user's language. The task is to learn concepts for prediction of the upcoming character *independently* from any assumption about the probability distribution.

S_Δ can be regarded to as a biased choice function that draws objects x from $\mathfrak{U}$ depending on the (unknown) underlying probability distribution Δ. **A** is expected to induce a hypothesis that allows the next object to be predicted without knowing Δ. It is obvious that any knowledge or assumption about $\mathfrak{U}$ is desperately needed in order to be able to learn some h that is sufficiently accurate with as few examples as possible.

To give a measure of the quality of $h=\mathbf{A}(\mathbf{s})$, one can (given Δ) compute the *error* of h. Basically, the error is the Δ-weighted sum of all those $x \in \mathfrak{U}$ for which $h(x)\neq\mathbf{t}(x)$. If the error $e(h)$ does not exceed a boundary of ε, h is said to be ε-good.[1]

It is obvious that one wants to minimise $e(h)$. Since Δ is unknown, one needs to give upper bounds for expected errors. Accordingly, ''learnability'' means that there is some **A** that on a sample $S(m, \mathbf{t})$ induces some h such that $e(h)<\varepsilon$ can be guaranteed with a confidence of $1-\delta$. This may sound a bit difficult, but it is the strongest proposition one can show since Δ is unknown.

The learning model described so far is known as the *probably approximately correct* (PAC) learning model; see Valiant (1984), Kearns (1990), and Kearns & Vazirani (1994). Its disadvantage is that PAC learnability is a very pessimistic measure: problems that are known to be PAC-learnable are mostly rather uninteresting, while ''real-world problems'' (into which category ML in UM surely belongs) most likely will *not* be PAC.[2] Its advantage is that the formalisation can be closely related to the user-modelling processes. It remains to be shown whether our view on learning UM will also be pessimistic.

2.2 *What makes a program learn?*

It seems that most interesting problems are not efficiently tractable. Just as many other general problems in computer science are theoretically intractable, one should not just abandon all hope but try to find a subclass of problems that are solvable. The same holds for ML. As the reader will have realised, the trick is to provide the learning algorithm with extra knowledge and heuristics, which guide the search and restrict space. These techniques are known as *language* and *search bias*. Additionally, one might find a (good) estimate of the unknown distribution. Similarly, the domain sometimes allows for a simplified formalisation of the problem, which may dramatically decrease computational effort. Finally, the suitability of a hypothesis always depends on the desired quality of the sought solution. Quality measures that guide incremental learning and define stopping criterions are known as *validation bias*.

However, assumptions made with the application of different learning algorithms are known to the expert, but they are hidden from the public. Sometimes this does not matter, since algorithms perform well in practice. Apart from biases, a very popular technique is domain simplification. Trivial simplifications are domain restrictions; a very common method for further simplification is the assumption of conditional independence of events, which is implicitly made in some statistical approaches and information gain driven methods.

2.2.1 *Naive Bayesian classifiers*

Naive Bayesian classifiers (NBCs) (cf. Cowell *et al.* (1999)) are very popular in document classification. Given a set of document categories $C=\{c_1, \ldots, c_n\}$ and strings $S=\{s_1, \ldots, s_n\}$, a joint prior probability distribution on observables (that is, preclassified documents) can be approximated

[1] To be precise, ε-good and -bad hypotheses are determined by computing errors on positive examples and negative examples separately, where the error for each set must be less than ε.

[2] A description of PAC-learnable problem classes is far beyond the scope of this paper. It is, however, interesting that in his conclusions and open problems, Kearns (1990) names (amongst others) the following three challenges that, on an informal level, can be related to our problems in ML for UM: how to describe learning *with* background information (as opposed to learning from scratch); learning with few mistakes (as opposed to *expected*, ''bothering'' mistakes); and agnostic learning (with no proper learning target).

by the conditional probabilities of string frequencies.[3] A document d is assigned a class c', for which

$$c' = \max P(c|s_1, \ldots, s_n) \tag{2}$$

The above term is equal to

$$P(c|s_1, \ldots, s_n) = P(c) \prod_{i=1}^{n} \frac{P(s_i|c, s_1, \ldots, s_{i-1})}{P(s_i|s_1, \ldots, s_{i-1})}, \tag{3}$$

which is computationally very expensive due to the chain rule of factorisation, see Castillo *et al.* (1997). However, by simply *assuming all s_i to be independent of each other* and dropping the constant denominator, which does not affect the maximum, one yields a rather cheap function:

$$P(c|s_1, \ldots, s_n) = P(c) \prod_{i=1}^{n} P(s_i|c, s_1)., \tag{4}$$

This example shows how the simplification of the problem yields an efficiently solvable problem from a literally intractable problem. Furthermore, NBCs deliver fast and precise results that withstand most comparative evaluations. Finally, they are easy to implement as they do not need any background knowledge. Accordingly, they have become a very popular approach for quick document classification (Billsus & Pazzani, 1997; Craven *et al.*, 1998a).[4]

2.2.2 *Induction of decision trees*

Decisions trees are concept descriptions that classify objects of the domain by incremental partitions with respect to the value of most discriminative features. An exhaustive search for the smallest **s**-consistent decision tree is NP-complete as shown by Hyafil & Rivest (1976). Therefore, finding the most informative feature is carried out using an information gain method, which is based on information theoretic entropy measures (Shannon & Weaver, 1949).

We assume that all objects $x \in \mathfrak{U}$ are described by a set of features $\mathfrak{F}$. The entropy of a set $S \subseteq \mathfrak{U}$ with respect to a feature $f \in \mathfrak{F}$ is measured by

$$H(S) = -\sum_{v \in \mathrm{cod}(f)} p_i \log_2 p_i \tag{5}$$

with p_i being the probability (relative frequency) of $f(x)=v$ for $x \in S$. If, for example, $\mathrm{cod}(f) = \{a, b\}$ and $P(f(x)=a)=P(f(x)=b)=0.5$ for all $x \in \mathfrak{U}$, then the entropy of any concept with respect to f would be

$$H(S) = -\tfrac{1}{2}\log_2 \tfrac{1}{2} - \tfrac{1}{2}\log_2 \tfrac{1}{2} = 1 \tag{6}$$

In order to give an estimate of the quality of a decision node, we need to determine the entropy with respect to the learning target **t** (see Section 2.1). Accordingly, (5) can be rewritten as

[3] Strings can be words, phrases or n-grams.

[4] It is worth a note that the need for simplification is mentioned in Craven *et al.* (1998b), but not in Craven *et al.* (1998a).

$$H(S) = -\sum_{v \in \mathrm{cod(t)}} \frac{|\mathrm{S} \cap \{\mathrm{x} : \mathbf{t}(x) = v\}|}{|S|} \log_2 \frac{|S \cap \{x : \mathbf{t}(x) = v\}|}{|S|}. \tag{7}$$

If the entropy of S is considered to be to large for the current validation criterion, S needs to be further divided using another feature f'. The choice of f' is determined by the expected information gain: supposing that S will be further partitioned using f', the expected entropy will be

$$E(S)_{f'} = \sum_{v \in \mathrm{cod}(f')} \frac{|\{x \in S : f'(x) = v\}|}{|S|} H(\{x \in S : f'(x) = v\}), \tag{8}$$

which is the sum of all resulting entropies weighted by their relative size. This value is computed for all f' and, finally, f with maximum information gain is chosen:

$$f = \max(\{f' : f' = H(S) - E(S)_{f'}\}). \tag{9}$$

Basically, information gain based methods have the same domain requirements as naive Bayes.

First, unconditional probabilities of joint events presuppose a finite set of features, each with a finite set of feature values. While the first restriction is not of any effect in applications, the second might become important when dealing with numeric values. Following the last argument, in

$$H(X_1, \ldots, X_n) \leq H(X_1) + \ldots + H(X_n) \tag{10}$$

equality is only obtained in cases where X_i are pairwise conditionally independent (see, e.g., Welsh (1991) and Li & Vitanyi (1997)). Accordingly, information gain as defined above overestimates key features (i.e. injective functions). As a consequence, the gain functions have been refined and optimised. For example, the gain ratio takes into account entropy with respect to the chosen feature as an additional weight; see Quinlan (1993) and many others.

Even though entropy-based information measures are widely accepted and deliver good results, these approaches meet with criticism. While in probabilistic approaches prior probability distributions are unknown, the ''thermodynamic'' notion of entropy actually requires equally probable and independent states of a closed system. However, most systems that we work with are *not* closed – simply by the fact that we cannot account for all relevant information in our simulational or inductive processes.

2.3 What makes an interface adapt?

As already described in the introduction, an adaptive user interface tries to increase usability by learning about the user and adapting an internal user model. The internal user model determines the behaviour of the interface, and it is refined by reasoning about observable user interactions (Jameson, 2003).

Let us assume that the user model $\mathbf{M}_u$ for a user u predicts a user's behaviour as recorded by observing user interactions. Accordingly, a UM problem can be defined canonically. A learning algorithm $\mathbf{A}$ induces a user model $\mathbf{M}_u$ (that is, a hypothesis h) based on background knowledge Σ and feedback $\mathbf{f}$ (a sample $\mathbf{s}$). Feedback is recorded by collecting and *interpreting observable* interactions:

$$\mathbf{f} = F(m, \mathbf{o}) = [\langle x_1, \mathbf{o}(x_1)\rangle, \langle x_2, \mathbf{o}(x_2)\rangle, \ldots, \langle x_m, \mathbf{o}(x_m)\rangle]. \tag{11}$$

The user's interest $\mathbf{i}_u$ makes the user perform certain actions that can be observed, but the user's intention for performing this action remains unknown to the system. Accordingly, the usage of $\mathbf{o}$

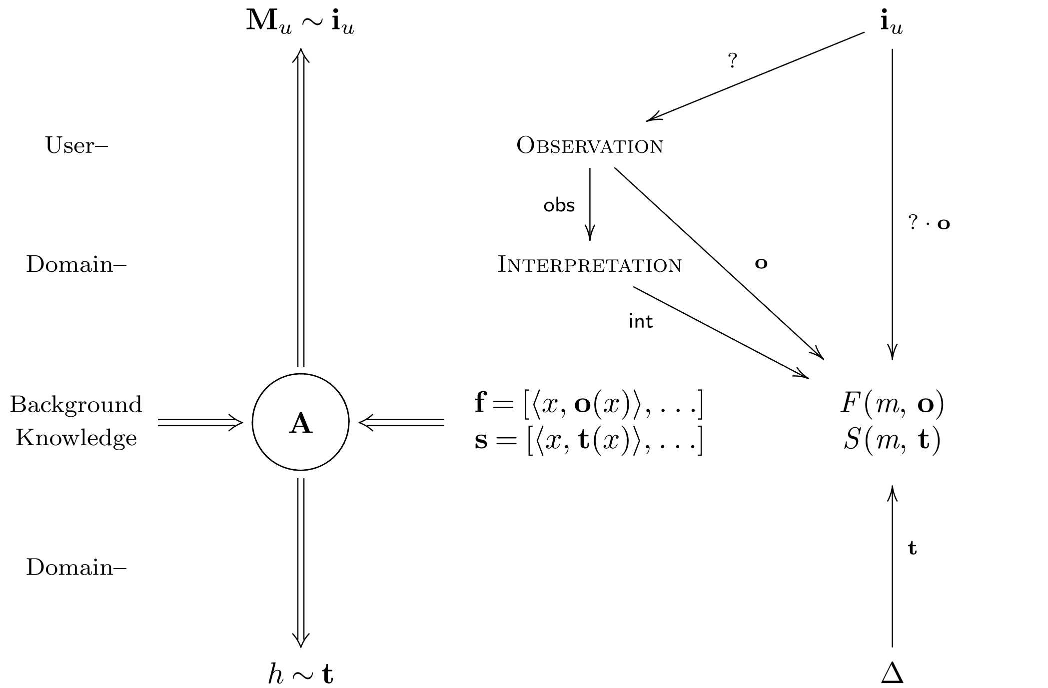

Figure 2 Parallels and differences in ML and UM

as a label in the sample means that the learning algorithm will not immediately try to predict $\mathbf{i}_u$ (which would correspond to $\mathbf{t}$), but it will try to predict what the next interaction will be. Figure 2 shows the parallels of ML (lower part) and UM (upper part).

At this point, it becomes clear that the behaviour of F is unclear at two different levels.

1. The intention behind performing $\mathbf{o}(x)$ is veiled by the fact that the user tries to realise his interest $\mathbf{i}_u$ based on his very own idea of the functionality of interactive elements as provided by the system. This is represented by the "?" arrows in Figure 2.
2. The label $\mathbf{o}(x)$ that is taken as a teacher signal for the learning task is based on the system's builtin interpretation of the user's interactions. As such, it is subject to noise for two reasons. First, the observation itself may be vague or noisy (`obs`). Second, the interpretation of the user's interactions is subject to domain simplification, but it is also prone to generating a noisy signal (`int`).

Concluding, it seems that a closer investigation of how interactions pertain to user interests and feedback labels elucidates the problems one encounters when trying to quickly induce user models from observation. Trying to summarise our considerations in one sentence, a pessimist might conclude that:

ML in UM tries to mimic a user's behaviour, but it does not model a user.

One good reason for this argument is that many successful approaches to ML in UM deliver hypotheses that are able to predict a user's behaviour, but cannot *explain* a user's behaviour. As an example, one might consider news filtering and forwarding to different display devices: a narrow bandwidth, small screen device (such as a cell phone or a palm-size computer) asks for more accurate results, while on a desktop computer a user is able to survey a larger collection of results. Feedback collected in this scenario may support this strategy, but the strategy fails as soon as we try to explain the reason behind the feedback by means of news content: the user may have different interests at home than he has in the office. Accordingly, the strategy delivers wrong predictions as soon as the user has a PC with a broad bandwidth connection to the Internet at home, but is not interested in job-related activities at home.

Another well-known example is the five-minute coffeebreak: time spent on a Web page can be interpreted as a supporting fact for interestingness, but the system is not usually able to disregard this observation in the case where the user has left his screen unattended.

As a consequence one might tend to ''abandon all hope'' when facing such a knotty problem. The other side of the coin is that in ML one is stuck with the problem of unknown distributions and the need for justifying domain simplifications, while in UM we might know a bit more about how to interpret interactions, how to avoid noise in observation and maybe even how to guide the user through planning and performing interaction sequences. Concerning the examples above, it means that knowledge about situative interest (job/private, work time/coffeebreak, etc.) might help to reduce noise or even gain reliable examples.

In the next section we see whether both ideas are just two sides of the same coin.

3 Finding out about the unknown: understanding *F*

As introduced in the last section, we investigate the nature of feedback $F(m, \mathbf{o})=\mathbf{f}$ by further decomposing F and $\mathbf{o}$. For the remainder of the article, $i'_u(x)$ denotes the *intended* (that is, the *planned*) action on x by u that is motivated by the user's interest $\mathbf{i}_u$ and his understanding of the system. $\texttt{obs}_u(x)$ is the observed interaction by the system that is interpreted by int. Accordingly, the following relations hold in general:

$$\mathbf{i}_u(x) \neq i'_u(x), \tag{12}$$

$$\mathbf{o}(x) = \texttt{int}(\texttt{obs}(x)), \tag{13}$$

$$\mathbf{o}(x) \neq \mathbf{i}_u(x). \tag{14}$$

A very well-known example is collaborative filtering, as performed by, e.g., Amazon. In collaborative filtering or UM, the ''user model'' consists of items or objects of the domain that are associated with the user. Similar sets of entities describe similar users. Accordingly, observed interactions ($\texttt{obs}(x)$) are purchases of items, but the reason behind a specific purchase is unknown. Here, the sample that is taken as input, $\mathbf{o}(x)$, is assumed to be a sign of interestingness to the user ($\texttt{int}(\texttt{obs}(x))$). This is why buying a bestseller novel as a birthday present for relatives pollutes one's own ''model of interest''. As a consequence, Amazon will never be able to learn an individual model of a users interest $\mathbf{i}_u(x)$.

There are, however, systems that try to model some $\mathbf{i}_u(x)$. Some try to reason about a user's plans and thus try to predict $i'_u(x)$. Other systems take into account cognitive states as a part of a user model.

We now investigate a few systems using this framework in order to obtain a clear picture on working systems that learn user models.[5]

3.1 Naive Bayes in UM applications

3.1.1 Manic: NBCs learn about learners

In relation to a student's knowledge level and preferences for presentation, the Manic system (Stern & Woolf, 2000) tries to learn whether a student would ''*want*'' to read a certain paragraph within an instructional text. Accordingly, the objects of U are information objects (including multimedia objects such as graphics or animations) that are linked by relations to form a semantic Web that describes dependencies between lecture topics.

[5] The list of discussed approaches is by no means complete or even remotely exhaustive. We simply chose several more or less well-known projects that allow this paper's issues to be discussed from a wide diversity of perspectives.

A lecture ''slide'' is generated by choosing a topic and then traversing the network, thereby collecting all information objects needed for the presentation. For each optional node, it has to be decided whether the student might want to see it (i.e. whether a switch for unfolding text should be displayed). The features used to describe these items are *media type*, *instructional type*, *abstractness*, place in *topic* and *concept* and, finally, the target feature *wanted*.

A NBC is used to learn and then predict whether a student might *want* to see a certain item. Observable actions are clicks on offered stretch-out icons: if the student unfolds an offered node, this is taken as positive feedback; not stretching a text of his current level of expertise is interpreted as negative feedback. The overall result of this approach is that a student will learn better because he will prefer the material that is presented as content and that presentation quickly adapts to his desires.

It is clear that in this case, $i'_u(x)$ and $\texttt{obs}(x)$ will match pretty well. However, the problem there remains the problem of interpreting ''not clicking'' as ''not wanting'': for an object x of matching difficulty, $\mathbf{o}(x)=0$ implies $\mathbf{i}_u(x)=0$, which is not necessarily true. Similarly, it has to be evaluated whether the features for describing objects of the domain are independent.

Nevertheless, a domain restriction, domain simplification, and few interactive elements allow for using the chosen ML method and receiving promising results.

3.1.2 Personalised information retrieval with NBCs

The Metiore system (Bueno & David, 2001) also uses NBCs for the recommendation of articles. In this paper, it is clearly stated that:

> [. . .] the result of the algorithm will be the degree of relevance of an object to the present objective for a user.

The interesting thing about it is that feedback is observed not only as a binary event, but recorded as *relevant*, *relevant but already known*, *indifferent* and *irrelevant*. Similar distinctions are made within NewsDude, where NBCs are also used for the recommendation of news items (Billsus & Pazzani, 1999). NewsDude particularly focuses on the difference between long-term and short-term interests: ''. . . *a user's information need changes as a direct result of interaction with information*''.[6] The system reads or displays news to the user and awaits explicit feedback, including two further types of positive feedback (''interesting, but I already know'', formerly often covered by negative feedback, and ''tell me more!''). Accordingly, here the same considerations apply as for the aforementioned Manic system.

Objections against NBC approaches in this domain have been discussed in Billsus & Pazzani (1997). Syskill & Webert is a meta search engine that offers the opportunity to give explicit feedback for each result. The feedback is used to build an individual user model that contains sets of Boolean key word vectors. Using trained Bayesian classifiers, Web documents (i.e. links on currently displayed pages) are recommended with respect to the user model. In their article, the authors show that the violation of the independence assumption does not affect the ability of NBCs to ''*perform well [. . .] in many domains that contain clear attribute dependence*''. However, the NBC approach is refined by the authors: since NBCs try to approximate joint probability distributions based on observations, the prior probabilities when learning ''from scratch'' may have a crucial effect on the overall accuracy (initial ''near-zero probabilities'' may lead to a myopic feature deletion). Accordingly, additional assumptions (ε-probabilities and Laplace distribution) on the prior distribution are made, and these assumptions are made with respect to the class that is to be described:

[6] It is worth noting that the NewsDude technology has now been further developed into a commercial product (Adaptive Information Server, cf. Billsus *et al.* (2002)) with many features similar to the academic InformationValet project (Macskassy *et al.*, 2000).

[. . .] The reason, why a strong estimation bias is beneficial in some domains, but hurts performance in others, seems to be linked to the ''quality'' of extracted features.

As a consequence, the author's algorithm chooses one of the aforementioned assumptions based on an information gain estimate of a feature's quality. Again, one might criticise the use of information gain methods – but again, results show that this method and its implicit requirements (although violated) work well on the domain.

3.1.3 Summary: NBCs in AUIs

One should keep in mind that the target learned by NBCs is $\mathbf{o}(x)$, and thus models the user behaviour as observed and interpreted by the system, but not the user's interest itself. Furthermore, the application of NBCs presupposes severe domain restrictions and biases based on assumptions that, in general, do not hold.

On the other hand, it has been shown that NBCs actually perform well, which is partially due to the ''simple'' task of predicting few observable interactions in restricted domains. Therefore, a larger set of observable user interactions (such as cod(`int`)={*interesting*, *interesting-but-already-known*, *uninteresting*, . . .}) and a pretty straightforward, and thus noise free, interpretation thereof should be seen as a major improvement when trying to achieve equality in (12) and (14).

Pretschner & Gauch (1999), Madrid & Gauch (2002) and Müller (2002) gave a more-detailed description of ongoing research in related fields.

3.2 Behind feedback: modelling a user's cognitive states

It is clear that knowing about a user's cognition and emotion helps to explain the difference between intention i'_u and interest $\mathbf{i}_u(x)$, and thus may help to explain observed data $\mathbf{o}$ as feedback $\mathbf{f}$. Furthermore, recent advances in multi-modal interfaces supply us with much more data of different qualities: pulse rate, breath rate, speech stress patterns, galvanic skin response. Such data can be used to build hypotheses about a user's emotional or cognitive state. When working with data from multiple resources, interpretation and integration is not trivial. Again, ML seems to offer a simple solution: feed data into the system and let the system learn to interpret and integrate data.[7] It is clear that if we do so, we lose the capability to explain observable interactions: they can be predicted, but they cannot be explained by being decomposed in intention and realisation. As a consequence, the user model may take into account further data, but it still does not explain a user's needs or cognitive or emotional needs. Therefore, more sophisticated models are needed.

3.2.1 Inside the user: personality and emotions

In general, a user's personality is said to be a set of user specific characteristics that persist through time while emotions are rather short termed. However, a user's personality can explain why different emotions shape up for equal contexts for different users: being impatient is a feature of a person's personality and will not change rapidly, but an impatient person will react angrily sooner than a patient person when repeatedly confronted with unwanted data.

Accordingly, Gmytrasiewicz & Lisetti (2001) described an agent's (and also: a user's) personality by a finite state machine (FSM). The FSM consists of nodes describing emotional states and links describing transitions between emotional states. The interesting part is that transitions are not performed by the agent itself, but are due to a probabilistic function that takes current emotional states (such as HAPPY or ANGRY) and environmental inputs (such as `satisfactory`) and delivers

[7] A prototypical example for this *modus operandi* was the hype followed by the resurrection of artificial neural networks: feed any data into the input layer of a multi-layer perception and let the hidden layers learn to map input patterns to output patterns. If it works, it is fine, even though nobody knows why – if it does not, the hidden layer is to small or the number of training cycles was too small.

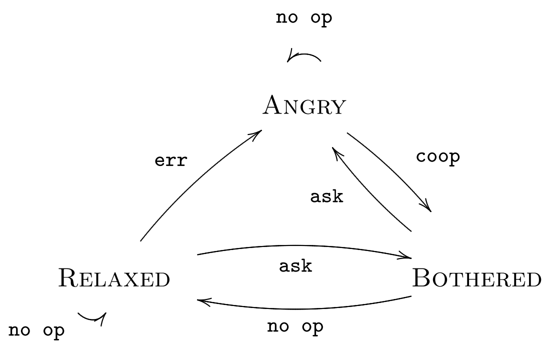

Figure 3 Personality as a FSM on emotions

the resulting emotional state. For example, if the user is Bothered, a cooperative system behaviour (here: remain silent by `no op`) makes the user feel Relaxed again, while an uncooperative system response results in an ANGRY user. Such a model of personality is shown in Figure 3, which is adapted from Gmytrasiewicz & Lisetti (2001).[8]

This approach might gain two important pieces of information.

1. Given knowledge about the current emotional state of a user and his personality, we are able to predict a user's emotional change when planning our (that is, the system's) action. For example, a patient and Relaxed user might accept one or two further bothering questions in order to obtain explicit feedback, while an impatient user will not. In this way, we can plan interactions in a way that satisfies an individual user's personality. One could also try to meta-communicate knowledge about the user. A system might say to an impatient user: ''*I know I am bothering you, but I need two more answers in order to satisfy your needs. I promise I will not bother you after that again*''.
2. Given an ambiguous interaction that the interface is not able to clearly identify as positive or negative feedback, our knowledge of the past emotional state and actions performed by the system can be used to interpret the user's interaction. An excellent well-known example here is that in such a case, cancelling a Web document download is not to be interpreted as negative feedback, but might be due to the impatience of the user.

On the one hand, this approach might help to explain the user's intention and thus de-noisify observed interactions **o**. On the other hand, this approach immediately gives rise to the question of where such models of personality come from and what kind of data should be used to detect such states. Here again, the personality model itself becomes subject to a UM process with all the difficulties that are mentioned in this article.

3.2.2 Probabilistic estimates of a user's cognitive states: Bayesian networks

A high cognitive load of a user can, for example, be detected by analysis of speech (silent and filled gaps, syllable frequency).[9] General preferences (and, for example, personality) and cognitive load have an impact on a user's behaviour.

Consider the following problem as shown in Figure 4: Observing a Cancel event, we want to know whether this is due to Disinterest in a displayed item or Impatience. Given prior probabilities, Bayes' theorem allows for causal relationships to be interpreted by a prior-to-post and post-to-prior decomposition. We represent the events by variables C, D and I and abbreviate $P(C{=}\texttt{yes})$ by $P(c)$ (and the reverse case by $P(\bar{c})$*. Supposing that the probabilities of a Web page being interesting is 50%, and the chance for a user being impatient is 0.2 (see prior probabilities in*

[8] A similar example is the hidden Markov model (HMM) with conditional probabilities for state transitions as shown in Figure 6.4 of Picard (1997) .

[9] There are also according ''symptoms'' in the context of interactions with graphical user interfaces: Lindmark (2000) gave a broad overview of observable events together with their interpretation as symptoms for resource limitations.

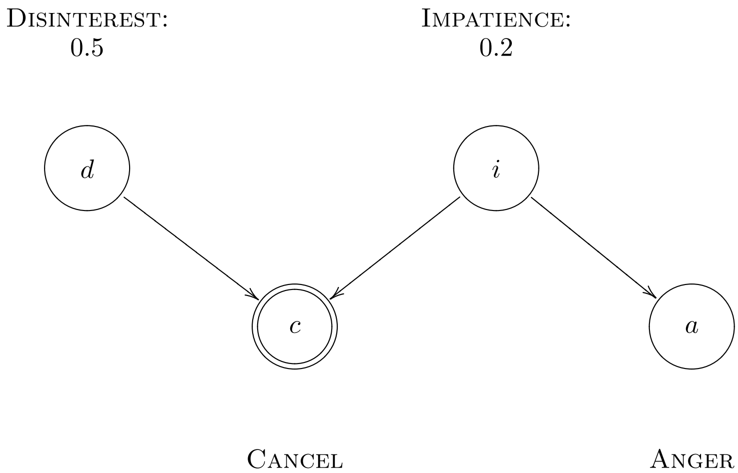

Figure 4 A very simple Bayesian network

Figure 4), one can derive a joint probability distribution. Both D and I are possible causes for c, and the likelihood of observing c given D and I can be inferred from previously observed events. We assume the following probabilities for $P(c|D, I)$:

$$P(c|D,I) = \begin{array}{cccc} d,i & d,\bar{i} & \bar{d},i & \bar{d},\bar{i} \\ 0.5 & 0.8 & 0.7 & 0.1 \end{array}.$$

By multiplying $P(I)$ and $P(D)$ one receives the joint probability distribution $P(I, D)$ (because I and D are assumed to be independent). Taking into account conditional probabilities from above, Bayes' law can be applied to calculate $P(I, D, C)$ and, finally, the posterior probabilities $P(D, I|c)$. In this example, the result is (assuming that we have just observed a cancel event c)

$$P(D, I|c) = \begin{array}{cccc} d,i & d,\bar{i} & \bar{d},i & \bar{d},\bar{i} \\ 0.33 & 0.33 & 0.28 & 0.07 \end{array} \tag{15}$$

This means that the probability of having an impatient but interested user is just 7%, while disinterest d will be the reason with a probability of 0.66 and i is the cause in 61% of all cancel events. However, this insight is not really helpful: both d and i are nearly equally good explanations for c. Also, they are not exclusive: 60% cannot be taken as evidence for ruling out the other possible reason.

Instead, the Bayesian network is extended by a further variable, which only depends on one of the two observable events. For this example, we assume that being angry A depends only on the personality of the user. Accordingly, we need a new local distribution that explains the probability of being angry given a certain personality. Here, we choose

$$P(a|D, C, i)=0.6 \text{ and } P(a|D, C, \bar{\imath})=0.1.$$

Now, the posterior distribution $P(D, I, c)$ is taken as new prior distribution to calculate $P(a, D, I|c)$ and finally $P(D, I|a,c)$:

$$P(D,I|a,c) = \begin{array}{cccc} di, & d,\bar{i} & \bar{d},i & \bar{d},\bar{i} \\ 0.49 & 0.07 & 0.41 & 0.02 \end{array} \tag{16}$$

.

The overall outcome of this calculation is that if we observe c and we know that the user is angry (a), then:

- disinterest d will be the reason in 56% of all cases; while
- the probability of having an impatient (i) user is 0.90.

In such a case, this outcome would be a strong argument for not labelling the document being presented to the user as uninteresting, because the negative feedback of the user is much more likely to be caused by his impatience.

Bayesian networks are a very popular approach in AUIs, because they inherently provide us with a very elaborate theory on how to deal with uncertainty. However, computational effort increases with growing networks. The most frequently uttered objections[10] against Bayesian networks are where do prior probabilities come from and where does the topology of the network come from? In the course of AUIs, substantial work has been carried out in course of the READY project. The READY system tries to recognise, reason about and adapt to a user's cognitive capacity which is, e.g., determined by time pressure, lack of knowledge or cognitive load. Two scenarios are a telediagnosis for a car breakdown, where the user is stressed by the problem, noise and time pressure and, a resource adaptive travel assistant for the Frankfurt airport (Jameson *et al*., 2000) (see also the footnote on page 11). In general, Bayesian networks for modelling causal relationships have become very popular. Accordingly, there are numerous projects in different application areas all of which use empirical and expert knowledge to elicit the required prior probabilities. Here, we only give two representative examples. Cockram *et al*. (2003) examined the software development process and management tasks related to it. They proposed a new model to increase efficiency of software engineering through optimised inspection modelled by a Bayesian network. Here, initial conditional probabilities were elicited from interviews with two independent groups of experienced experts. Kruegel *et al*. (2003) used a Bayesian network for intrusion detection on computer systems. Again, the authors make use of domain expert knowledge for initial conditional probability tables prior to the adaptation process. Of course, there are many more examples, especially in the technical and medical domain where prior observations, domain theories and experts are available.

As we have seen, the structure of an underlying Bayesian network can be justified by a causal network whose architecture and initial probabilities have been empirically validated. In this system, the user model $\mathbf{M}_u$ does not primarily describe a user's interest, but his cognitive state. As mentioned above, the interpretation of observed symptoms is empirically validated, such that the noise caused by `int(obs(`x`))` can be assumed to beminimal. Accordingly, this approach can be regarded to an approach where $\mathbf{f}$ delivers examples of very high quality. On the one hand, the Bayesian network can be used to predict user behaviour (lack of knowledge increases the probability of ''description errors'' and thus, ''click and re-click'' events), but also to explain observed events (''overscrolling'' is an observation that supports the symptom ''motor error'', which could be explained by time pressure).[11]

3.2.3 Affective computation

In affective computing, ''*one seeks to build systems with the ability to recognise, express and 'have' emotions*'' (Picard, 1997). Emotions have a direct influence on different levels of a human's cognitive system: a certain emotional state affects decision making and, thus, presupposing a rule-based model can be interpreted as modifiers of reasoning rules. On this level, a system is aware of its own emotional state and could be regarded to as a system which ''has'' emotions. Simultaneously,

[10] Apart from the opinion that Bayesian learning is not ML in the strong sense. This issue was briefly discussed at the 2001 ML4UM Workshop at the User Modeling Conference, but not accepted as a valid argument, since one would also rule out so many further learning approaches such as artificial neural networks and SVMs, etc.

[11] Examples are chosen in concordance with Figure 2 of Lindmark & Heckmann (2000). Similar examples with spoken language as observed interactions can be made up from the Bayesian network shown in Figure 3 of Müller *et al*. (2001). The latter article also describes how to design and validate the architecture of a Bayesian network based on an empirical experiment.

emotions affect behaviour on a level below (self-)consciousness: censoring affective signals from a dialog partner, an emotional system responds to those signals directly, without further reasoning. Both levels perfectly fit into the distinction between signal- and pattern recognition and inference systems. Pushing the analogy a bit further, one might conclude that this could be the ideal point where symbolic and subsymbolic learning need to collaborate to form a hybrid that is able to act as an affective system.

However, one should be *very* careful not to get lost in metaphors. Even though researchers agree that there are ''mixed emotions'', we cannot really define what even a single emotion is. The most popular approach to describing basic emotions and mixed emotions is the OCC model (Ortony *et al*., 1988). However, both number and type of basic emotions vary in literature. If, however, one agrees on basic emotions such as *fear*, *anger*, *sadness*, *happiness* and *disgust*, it seems a reasonable and solvable task to recognise those basic emotions from appropriate signal input.

Murray & Arnott (1993) described discriminative features in emotion recognition from spoken language. These are, among others, speech rate, pitch average, pitch range and intensity. However, if we now restrict ourselves to features that deliver rather reliable data, and if we further discretise the values of the features used to describe speech (as is already done in this approach), we end up with four features and three values each.[12] The interesting thing now is that using these attribute/value pairs, one can no longer distinguish (nor can one learn how to tell) between *fear* and *anger*. Much more alarming is the fact that *happiness* cannot be discriminated from *anger* either.

At this point it becomes clear that emotion recognition from speech is a very hard problem. The same cautious aloofness seems advisable in emotion recognition from facial expressions or bio-signals. In the former approach, one has to deal with problems connected with image processing, plus the fact that much data used as input are taken from actors instead of real users. The latter approach is concerned with all tripwires in signal processing and, again, the fact that some data recorded are taken from actors (even though there are much more real-world data available).

Again, having in mind a model of personality and some means for predicting likely changes to emotional states could help a lot. Accordingly, a collaboration of psychologists and experts on models of personality and emotion, together with UM and ML could help each of the involved disciplines. Finally, recent endeavours in recording and using bio-signals such as heart beat, pulse and respiration rate, galvanic skin response, electromyograms, etc., in ubiquitous computing show promising results and are very likely to become the topic of many research activities in the booming field of wearable and ubiquitous computing.

3.2.4 Summary: effectiveness by affectiveness?

As shown in the previous sections, taking into account information about the emotional state, a user's personality and, in general, rich context information about the affective component of a HCI may help in adapting to the user in several ways.

First, the user interface can interact more adequately since it is able to reason about the user's emotions. This knowledge can be used to react accordingly as well as to pro-actively trigger a certain emotional state. Of course, the latter (but also the former) behaviour can result in an interface which is better accepted than an interface which does not try to behave like a coequal (but still, subordinate) dialog partner. As a consequence, one might expect that a better accepted interface will also result in an interface that increases productivity: the interface tries to avoid distress and the user likes to work with it.

Second, knowing (or predicting) a certain emotional state can help to interpret data. This is crucial in the domain of ML for UM, where data is inherently noisy. Most of the time, we will not be able to extract reliable hard data out of noise by taking into account knowledge about personalities, but we might at least be able to detect noise – that is, given a sample we might be able

[12] These are the features mentioned above and values *average* and *below/above average* (and *non-average*). The results we refer to here are reprinted in Picard (1997, table 6.1).

to discard the noisy examples and, thus, increase chances for a good hypothesis given that the algorithm can cope with fewer examples.

However, an adequate, affective system will not always increase the effectiveness of interaction. The tradeoff between user satisfaction, usability and workflow has to be evaluated carefully in each domain. Whether an increase of effectivity (in terms of throughput) is actually what is wanted has to be considered thoroughly – quite often, increased turnover is accompanied by less quality, which is unacceptable in risky environments or applications. For example, it seems very unlikely that an adaptive, emotional pop-up agent will help in a data-typist's work. However, one might imagine that a system that is able to detect stress and thus postpone low-priority tasks can be of great help for an air-traffic controller's working environment.

3.3 *Understanding a user's plan*

An interaction sequence between a user and a computer usually consists of several steps, which finally should enable the user to achieve a certain goal.

''*Thus*'' writes Küpper (2002), ''*computer users may benefit from plans [. . .]''. Presupposing such a plan, an adaptive system should be able to recognise certain plans. If a plan cannot be recognised from a static plan library, then it seems appropriate to learn* different plans. However, ''*we should not expect users knowing plans for all their goals*'' (Küpper, 2002). In such a case, a user would behave irrationally if we presuppose a goal directed behaviour and need further knowledge about what the user thinks about the system's capabilities. We briefly discuss both issues in the following sections.

3.3.1 *Recognising goals and plans*

Planning and performing a plan is hidden from the system as it occurs prior to the observation obs of interaction events. An interaction sequence consists of a sequence of states and interactions as transitions between them:

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n.$$

Each action a_i is an instantiation of a *plan operator*. Its application is determined by so-called *preconditions* and it results in *postconditions* and an according state transition. For example, the conditiong of the goal to be achieved does not hold in states s_i where $i<n$. Pre- and postconditions often allow to plan operators to be rearranged such that the sequence of interactions is not fixed (even though pre and postconditions induce a partial ordering); similarly, several instantiations of different plan operators may yield different plans. Figure 5 gives an informal example showing that planning is usually much more complicated than suggested by the sequence that is displayed above. Recurring subsequences of interactions may form a parameterised, iterated loop. Here, one goal could be to learn how to automatically perform this loop in order to ease achievement of the user's goal (one example is automated macro construction, which will be discussed in Section 3.4.2). As already mentioned, one cannot be sure that the user knows a plan at all. If this is so, another problem occurs: the user plans an interaction not by knowing about each state or the effect of each operator, but rather by *what he thinks* a certain action will yield (see Section 3.3.2). This will most probably lead to an interaction sequence that does not correspond to what a machine planner would consider an optimal plan. Furthermore, single actions might simply lead to a undefined or undesired state, which seems to make a plan fail. Accordingly, plan recognition – and, thus, anticipation of the user's goal – is a hard problem.

For a more detailed and recent survey of the state of art in planning see Weld (1999).

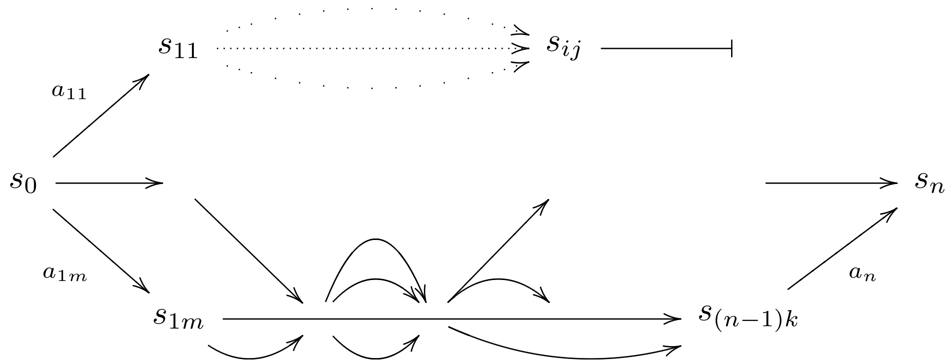

Figure 5 A more complex planning problem. In s_0, there are several operators applicable that, with different instantiations, may yield m different actions. There are three different subplans leading from s_{11} to s_{ij} , but this state marks a dead end. To reach $s_{(n-1)k}$, there is a complex operator starting at s_{1m} that can be decomposed into several other operators and actions

Recognising plans

Accordingly, recognising an unknown plan from observables is a major improvement in understanding the user and understanding the user's intention, which we marked as a ''?'' in Figure 2. However, recognising the goal of an interaction sequence is not trivial. By observing a_i , one can induce that the user wants to achieve some state $s \in S$, where S is the set of states accessible from s_i . If we have knowledge of more complex plans (which consist of sequences of actions) and we observed a subset of actions, it seems reasonable that the user wants to achieve the goal of the complex plan (or states that can be derived from the outcome of the complex plan). Many researchers worked on the problem of planning and plan recognition; the latter, for example, in the works of Allen, Litman, Sidner, etc.[13] However, as we are concerned with recognising the plans of a user interacting with a cooperative adaptive system, the planning paradigm based on Pollack (1990) as developed by Grosz & Sidner (1990) seems more appropriate. Accordingly, this approach was further developed to the COLLAGEN framework for collaborating agents in HCI (Rich & Sidner, 1998; Lesh *et al.*, 1999).

The latter article describes an approach to feasible plan recognition, as the general problem of plan recognition is intractable (assuming the current state after action a_c to be s_c), the closure $S=\{s|s_c \vdash s\}$ contains all possible succeeding states and the intention of the user can be an arbitrary subset of all postconditions of those states: $i' \in \mathfrak{P}(\{\mathtt{post}(s)|s \in S\})$. The trick is to only consider as search space those plans that require actions that have not been observed yet and that are reachable from a current *focus* action a_f.[14] The computational effort thus decreases dramatically if the number of observed actions is less than the longest possible plan and it further decreases the closer the focus is set to (observable) primitive actions. Multiple explanations (i.e. several possible plans) are disambiguated by taking into account the current focus; in cases where the user's goal cannot be identified, the focus is generalised. Finally, if no explanation could be found after a preset number of interactions, the interaction is interrupted by a clarification dialog. The individual behaviour of different user's personalities as expressed in more or less focused or unfocused behaviour can be described by different operations on a focus and plan stack (a focused user tends to a ''depth-first'' behaviour, which means that current tasks are completed before working on the next one). This approach has been discussed in Lesh *et al.* (2001).

It remains to be explained, *how* possible plans can be automatically acquired by the system.

[13] Perrault & Allen (1980) dealt with indirect speech acts (comparable to unknown goals of a plan) and pointed to Gricean maxims that apply to HCI in general; see Section 4.1.2.

[14] This idea roughly corresponds to the difference between an exhaustive search and a beam search *focused* through a_f.

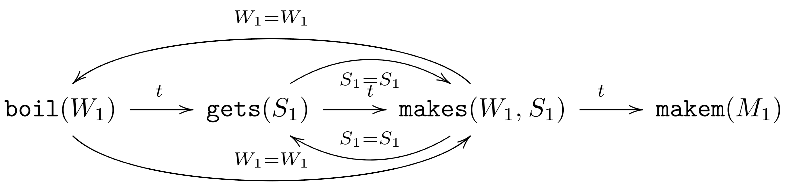

Figure 6 Observing a sequence of actions (adopted from Bauer (1998))

Learning plans

A user's interactive goal can only be recognised if the system has knowledge about possible goals and some theory about how goals can be reached by means of interaction sequences. However, as there are numerous ways to choose and prepare lunch or process email, it is impossible to extensionally define all possible interaction sequences for all possible goals.[15] It is even a hard task to define a sufficient plan library, by which a planner can generate a matching plan during a session. Accordingly, one wants to acquire a library of plans or plan operators and a set of possible goals, such that the system itself is able to generate plans that model a user's behaviour.[16]

Assuming that we have identified a set of possible goals on a given domain theory, we also need a sample of interaction sequences that lead to respective goals. This data can be assumed to be present from shell histories, log files, etc.[17]

Bauer (1999) described how plans for dinner cooking can be acquired. The ML method that is used here is a clustering approach on action graph joins; in Bauer (1996) plans were acquired quantitatively using a Dempster/Shafer approach.

The idea behind the acquisition of plans from observed action sequences can also be explained using a logic approach: structural and temporal dependencies of observed actions can be visualised using graphs as shown in Figure 6: First, a user boils some water, then gets spaghetti from the shelf, makes spaghetti using the boiling water and finally prepares a marinara sauce. Using a logical description, the same problem can be represented as:

prepare_pasta_menu() ←
  act(1, boil(W_1)),
  act(2, get_spaghetti(S_1)),
  act(3, make_spaghetti(W_1, S_1)),
  act(4, make_marinara(M_1)),

Note that temporal edges *t* from Figure 6 are represented by the ''<'' relation on the first argument of the predicate act. Similarly, argument agreement is implicitly achieved by unification of variables. Let us consider a second interaction sequence:

prepare_pasta_menu() ←
  act(1, go_kitchen(K_1)),
  act(2, boil(W_1)),
  act(3, make_pesto(P_1)),
  act(4, boil(W_2)),
  act(5, make_tagliatelle(W_2, T_1)),

[15] Processing email and cooking pasta dishes seem to be the most popular example domains in plan recognition for UM.

[16] At this stage, we do not take into account so-called mutual models, which may help to understand the user by making assumptions about the user's model of the system. This issue is discussed in Section 3.3.2.

[17] Explicit labelling, however, requires additional work, which is why Bauer (1998) assumed test subjects, or why most evaluations are carried out on the Unix domain set.

Here, the user boils water, prepares a sauce, boils water again and finally cooks pasta. Let us assume that we have domain knowledge of more abstract concepts, namely:

`make_pasta`(X, Y) ←`make_spaghetti`(X, Y).
`make_pasta`(X, Y) ←`make_tagliatelle`(X, Y).
`make_sauce`(X) ←`make_pesto`(X).
`make_sauce`(X) ←`make_marinara`(X).

Then, the following hypothesis can explain both observed interaction sequences:

`prepare_pasta_menu`() ←
`act`(N_1, `boil`(X_1)),
`act`(N_2, `make_pasta`(X_1, X_2)),
`act`(N_3, `make-sauce`(X_3)),
$N_1 < N$.

A ML approach that delivers logical formulas as generalisations from factual knowledge is known as inductive logic programming (ILP), which we discuss further in Section 3.4.

Knowledge about a user's plan can also be acquired by explicit instruction. Herein, a user identifies objects of interest and performs actions on them in order to teach the system to copy this behaviour in *similar* cases. Again, *programming by demonstration* (Lieberman, 2001) marks the borderline between learning a user model and mimicking a user's behaviour. If the demonstration consists of a stream of interactions only, the system will be able to recognise similar interaction sequences. If, on the other hand, the demonstration is performed on objects that have a deeper meaning in the discourse representation, the intention behind performing this action can be explicated. In this case, `int` can help to unveil information hidden in the unknown plan or realisation that was labelled ''?'' in Figure 2.

3.3.2 I think you know I think: a user's idea of a system's user model

For describing interaction scenarios such as dialogs, mutual models need to be taken into account. It is clear that when asking someone for a favour, our model of the dialog partner is crucial for planning our speech act in a way that determines whether to ask at all, and then how to ask. Accordingly, early work on UM was based on belief models in dialogs (Kobsa & Trost, 1984).

Morik (1986) showed that a dialog memory (which corresponds to a sequence of observable user interactions) is just a proper subset of a user model, as the dialog memory only contains dialog dependent knowledge.[18] Accordingly, different users behave (or communicate) in different ways in the same discourse situation. It is clear that a user has a model about the system he is working with. Adaptive systems might try to learn a model about the user.

It is quite likely that a user will recognise an adaptive behaviour and adapt his model of the system behaviour accordingly in a way so that his model of the machine includes a model of himself as he thinks the machine might have acquired. However, do adaptive systems model in the way that the user thinks they work? One might ask such a question. There are even considerations in mutual agent modelling where several individuals collaborate and try to achieve a common goal based on assumptions on each other's capabilities and needs (Suryadi & Gmytrasiewicz, 1999).

Knowing about a user's model of the system may help to understand the difference between $\mathbf{i}_u(x)$ (his needs) and $i'_u(x)$ (his plans by which he tries to achieve his goal). One the one hand, we want

[18] It is worth considering the question of whether a user's knowledge about a dialogue partner model (that is, a mutual model) belongs to a dialogue memory or not. In Kobsa & Wahlster (1988), Morik (1988) clarified the difference by the notion of a ''*user's wants*''. The wants (or needs) are parts of the user model that determine a user's specific behaviour in a dialogue situation that is described by the user model.

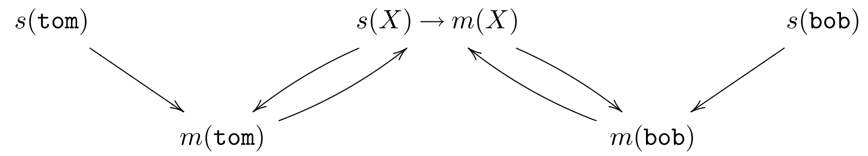

Figure 7 Resolution as deductive and inductive inference

to learn some model approximate $\mathbf{M}_u$, but, on the other hand, $\mathbf{f}$ is based on $\mathbf{o}(x)$, and thus on $i'_u(x)$ and not $\mathbf{i}_u(x)$.

A very primitive, implicit mutual model is used in systems where the user's acquaintance or skill level is described by attributes such as *beginner*, *intermediate* or *expert*. Assuming the user to be a rational agent and to be an *expert*, the user's model of the system is correct and the actions performed are exactly the actions that one should perform in order to achieve the goal. However, being *intermediate* or *beginner* means an increasing amount of noise in the interaction: the less accurate the user's model of the machine is, the more ''wrong'', unintentional actions will occur. If, for example, we as an adaptive system know that the user knows that we record time spent on a Web page to measure interestingness, then we can assume that session data is rather reliable. However, if we know that the user is not aware of our technique, then we know that we will acquire noisy samples.

If, as another example, we know that the user knows a certain command shortcut,[19] then we know that an alternative command or menu choice will most likely correspond to another plan than the expected one.

3.4 Further approaches of ML to UM

When trying to learn a user's plan, one needs to be able to describe a plan. Similarly, complex domain descriptions with lots of background knowledge involved need a richer description language. Sometimes, a user model consists of ''programs'' describing procedural behaviour. Finally, the desire for scrutable user models requires a system to describe its behaviour to the user.

Accordingly, it seems reasonable to employ ML techniques that deliver intentional knowledge instead of ''data extensions'', which can be read as programs or procedures and that are able to represent relations, reason with and even invent relations on observed data and background knowledge. One ML technique that is able to satisfy these needs is the underestimated approach to learning user models by relational learning or ILP. It is an interesting fact that ILP has not been considered as a promising technique in ML for UM in the past, especially since the first approaches to UM were logic based (and some still are, as we have seen in Section 3.3); see, e.g., Kobsa (1988).

However, there are few recent approaches in UM that make use of ILP because of its above mentioned advantages.

3.4.1 Relational learning

Relational learning tries to induce hypotheses that intentionally describe concepts by relational definitions. Instead of deductive inference by way of *modus ponens* it seems a promising approach to try inductive inference by reverting to the resolution that yields a more general rule describing a set of (ground) facts. This is shown in Figure 7: *modus ponens* infers that every student (*s*) is mortal (*m*), while induction infers from observations on the ground instances `tom` and `bob` that *every* (*X*) student is mortal. Approaches to efficient logic program induction range from least or relatively least generalisations and information gain heuristics through beam or A* search on rule schemes or sets of partial (so-called *saturants*) in inverted entailment.

Quinlan (1990) described FOIL, an efficient information gain guided algorithm for ''first-order induction of logic''. The basic idea is to describe a set of ground instances by generating a set (that

[19] And if we can also assume that he acts rationally; that is, he would choose it if needed.

is, a disjunction) of rules to cover increasingly many positive examples, and for each rule, to add (a conjunction of) literals to the premise of each rule to exclude increasingly many negative examples. Suppose we want to decide whether or not to offer a link to a new course item in an adaptive hypermedia tutoring system. Links should only be offered if the student has proven skills that are necessary to understand the linked concepts. The signature we use consists of the following predicates: *understands(X, Y)* means that a student X has proven skill concerning topic Y , *prerequisite(X, Y)* means that understanding X is required for understanding Y and *offer_topic(X, Y)* means that a link to Y should be presented to X. Now imagine we have the following observations:

```
understands(bob, terms). understands(tom, terms).
understands(bob, unification).
prerequisite(terms, unification).
prerequisite(unification, resolution).
offer_topic(bob, resolution).
```

The initial hypothesis is h=*offer_topic(X, Y)* ←., which is far too general as it also covers that tom should be presented a link to resolution. However, due to the CWA, we know that tom does not satisfy the requirement of knowing about unification. Therefore, the rule needs to be specialised by literal adding. The heuristic measure will choose to add *understands(X,Z)* to the hypothesis yielding h=*offer_topic(X, Y)* ←*understands(X,Z)*. Sadly, this clause still allows *offer_topic*(`tom`, `resolution`) to be derived, so we have to add a further literal. The new candidates are, among others,

```
prerequisite(Y,Z). prerequisite(Y,Z). prerequisite(W,Z).
prerequisite(Z,W). prerequisite(Y,W). prerequisite(Y,W).
```

The most promising candidates are *prerequisite(Z, Y)* and *prerequisite(W, Y)* (recall that Y is instantiated with `resolution`), but the former only uses known variables while the latter introduces a new variable W. Accordingly, the first candidate is chosen which finally yields a clause

offer_topic(X, Y) ←*understands(X,Z), prerequisite(Z, Y)*.

Now, the link to `resolution` is presented to `bob` but not to `tom`, as `bob` has understood `unification`, while `tom` has not.

3.4.2 Learning commando sequences

The LearningShell constructs user-dependent macros for a Unix command-line interface. The system takes a sequence of user interactions from a shell history and tries to extract frequent patterns from it. This technique is similar to unobtrusive, click-stream-based systems. The outcome, $\mathbf{M}_u$, is a macro (or mini shell script) that aggregates several commands and thus describes a user's interactive behaviour.

Jacobs & Blockeel (2001) described learning of an edit–compile–view sequence during the generation of a LaTeX document. By observing the user editing several files, the outcome is a macro which generalises of the arguments of the command:

```
#!/usr/bin/sh
vi $1.tex
latex $1.tex
xdvi $1.dvi
```

Note that the program learns to abstract from the filename, but takes into account the file extension. Such a macro can only be learned by means of relational learning based on the observation that a command (such as `vi`, `latex`, or `xdvi`) has been invoked on several attributes that consist of an `attributestem` and an `attributeextension`. If the command `vi` has been submitted with an attribute `file.tex`, then the succeeding commands should have the same attribute stem, but extensions `tex` (for `latex`) and `dvi` (for `xdvi`).

This approach can easily be extended by adding background knowledge about several editing commands such as `vim`, `emacs`, or `pico`.[20] Similarly, one could consider using `pdflatex` as an alternative to the `latex` command. Then, both the succeeding command needs to be changed (`acroread`) and the attribute extension, `pdf`. This can be accomplished by introducing similar commands as synonyms. Instead of learning from strings occurring in the shell history as in

```
command(1,1,vi). command(1,2,latex). command(1,3,xdvi) .
```

data would be enriched by background knowledge:[21]

```
command(1,1,X):-    command(1,2,X):-    command(1,3,X):-
   edit(X).           texcompile(X).          view(X) .

edit(vi). edit(emacs). edit(pico).
texcompile(latex). texcompile(pdflatex).
view(xdvi). view(acroread).
```

Note that determining the extension for a view command actually depends on the instantiation of X in the `command(1,2,X)` clause. Such strong hypotheses can be learned using ILP while other approaches must fail due to their restricted hypothesis language.

It is clear that such an approach allows for a much better scrutability of user models. When used as recommender system, for example, the system (using the example from above) would recommend ''*Don't you want to* `acroread file.pdf` *now?*'' instead of `xdvi file.dvi` after `pdflatex file.tex` was observed. Even better, the system can *explain* its action: ''*Both* `xdvi` *and* `acroread` *can be used to view camera-ready papers, but you have just generated a PDF-document, which can only be visualised using* `acroread`''.

Assuming that the user model that is to be learned contains descriptions of frequently reoccurring command sequences, then the observed interactions `obs`(x) (single command lines) can, in fact, be mapped onto a sample **s**. Furthermore, the only noise sources are frequent spelling mistakes or, for example, commands submitted in the wrong working directories.[22] Nevertheless, with only very few data (that is, sparse subsequences) the learning problem is very hard.

3.4.3 Learning to filter email

Kay & McCreath (2001) described an ILP approach to the problem of personalised mail filtering (MUMILP). Mail filtering is desperately needed in recent times and therefore has become a popular application area in UM. Furthermore, learning how to filter mail is supported by clearly labelled examples: moving mail into folders delivers reliable classification data as a teacher signal **t**.

[20] See also the example in Section 3.3.2.
[21] The predicate command takes three arguments: First, the string's position in the command line (`1` corresponds to `ARGV[0]`); second, the relative number of the string in a sequence; and third, the string itself.
[22] These errors, however, can also be learned, yielding a macro that contains erroneous commands together with their revocations. Having some background knowledge, this would further allow frequent mistakes to be learned as well as relearning the macro for optimisation (by deleting command–undo pairs such as ''`cd $1`'' followed by ''`cd ..`'').

In this domain, the classification tasks to be tackled are twofold: filtering spam is a binary decision, while pre-ordering of mails into existing folders is a more sophisticated learning task. MUMILP learns rules that preclassify incoming mails. In terms of ILP or, more generally, relational learning, the task is to learn a relation between messages and folders, which is performed by relating strings describing the folder's name to strings from the mail.

Here, the domain model requires emails to be represented in such a way that the ILP algorithm can construct hypotheses. Accordingly, the predicates used determine the sender, occurrence of a string, etc. Example hypotheses could be:[23]

```
filter(M, spam):-
  mail_subject(M, X), member(cash, X).

filter(M, friends):-
  sender_identity(M, X), member(X, [jack, jill]).

filter(M, mllist):-
  messages(X, mllist), most_similar(X,M).
```

The first tries to identify spam mail by the key word `cash` in the mail subject. The second one classifies mails from `jack` or `jill` as mails from friends. Finally, the third rule hands the problem of text classification over to some TFIDF classifier by comparing the incoming mail with all folders and succeeding if `mllist` is the folder that yields the best similarity value. Of course, all those literals could be combined to gain more complex rules.

In this approach, **f** can be assumed to be rather noise-free, as long as we want to discriminate spam from non-spam. Learning more sophisticated classifiers is more prone to noise as a single mail may belong to several classes. On the other hand, the fact that a certain mail belongs to folder `recreation` supports the proposition that it does not belong to the folder `conferences`. However, if we are concerned with folders for several conferences with overlapping committees this discrimination is not as easy as in the first case (this problem has been worked in the approach described in the next section). However, one can agree that a user files a mail with purpose (in order to retrieve it quickly) and, thus, the intention behind filing is the same to all users. Similarly, the events that are being observed are easy to interpret: filing a mail is a clear positive evidence for a classifier.

In this approach text clues are taken into account when trying to induce user models. Another idea is to discriminate text classification from user ML. This has been done in the project described in the next section.

3.4.4 Learning about interesting documents in Web search

Another project, which we want to describe only very briefly is OYSTER, where ILP was used to learn user models that contain logic programs describing a user's interest (Müller, 2001). Documents on the Web were classified into a taxonomy of content-specific concepts and document type specific information. The domain description language of concepts and types is used for both the documents and the user models. Upon submitting a search request to a meta search engine, OYSTER collects all documents that can be compared with the user model. The data taken as input consists of the documents together with a label that is derived from explicit feedback. The user model contains several aspects of a user's interest, where each aspect consists of a set of clauses representing a procedural description of ''interestingness''.[24]. One such clause could, for example, state that a document is interesting if it is a ''scientific publication', it deals with ''user modeling'',

[23] Slightly changed from Kay & McCreath (2001).

[24] This method can be vaguely related to the FOIL learning algorithm, where rules become more accurate by adding body literals as constraints and the whole program becomes more general by inventing more rules.

and it belongs to the concept ''machine learning'' with a confidence of at least, say, 75%. The use of different aspects allows for a more precise (that is, accurate) description of different topics of interest.

The primary motivation behind this approach was to overcome the lack of feedback: with only a very small sample **s**, a learning algorithm will produce a rather bad hypothesis (which will be either too general or too specific). By taking positive evidence for one aspect as negative evidence for another aspect, the sample could be enlarged such that a better user model could be learned. Of course, this method cannot work in scenarios where sufficient feedback for learning is already available: The more feedback we have, the bigger the danger of erroneously counting ''foreign'' evidence as positive or negative for interest, especially in the context of overlapping interest aspects or documents that may belong to several classes. Speaking of classes, another disadvantage of this approach becomes clear: one taxonomy for all presupposes an equal understanding for all users which certainly is not the case.

However, it was shown that for few observed events with a clear interpretation (obs records explicit feedback that describes interestingness of documents as `int`) accuracy could be significantly improved, while for a sample size $m>25$ the accuracy gain vanished; see Müller (2002). A further advantage of this approach is that the induced user model $\mathbf{M}_u$ is both a scrutable description of a user's interest as well as a procedure for user-specific recommendation.

As one can see, relational learning makes it possible to induce complex descriptions that go beyond what simple classifiers can deliver. It can deal with much more complex domains including background knowledge. Furthermore, it is worth a note that in early UM research, logic-based approaches were much more popular than they seem to be today. As an example, most articles in Kobsa & Wahlster (1988) present purely logic approaches to representing user models.

The disadvantage is, however, that without a decent domain theory or no background knowledge, one cannot expect impressive results and, thus, some effort of domain modeling is required. The argument of intractability, however, does not hold anymore, especially when taking into account that *any* ML for UM application needs to have appropriate bias in order to be able to deliver according hypotheses at all.

4 Trying to find an answer

It has been shown that many approaches allow a user's behaviour to be predicted. Some of these approaches are rather data driven; some are (much) more sophisticated and try to induce a real model that explains what is going on in a user's mind.

Therefore, it seems that – with reasonable biases and restrictions – user models can be learned. It has to be emphasised though that learning is not an all-purpose tool for adaptive systems: sometimes it takes too long, sometimes there are too few data, sometimes there is not enough knowledge available and, sometimes, the tool simply does not fit the problem (or the problem cannot be transformed into one that fits the algorithm). We need to emphasise this fact in order to prevent being accused of empty promises. Accordingly, the first part of our answer to the title question is

> ''Yes, user models can be learned, but . . .''.

However, HCI is inherently domain and application dependent. Accordingly, one needs to check whether adaptivity is *wanted* at all in the context under consideration.

4.1 Should a user model be learned at all?

Before answering this question, it seems appropriate to point out to some remarkable parallels in related areas in the past.

4.1.1 *Clear positions and wrong predictions*

There has been a discussion of the same quality with the advent of graphical user interfaces and, more recently, multimedia agent interfaces. Gladden (1986) said that

> ''GUIs are just a fad, when their novelty wears off, users will soon return to true command-line interfaces.''

Novelty has become a standard by now, and although many users who are *used* to using command-line interfaces prefer them for their better *usability* and better performance, most users have proven the above statement to be false.

Concerning (multimedia) agents, Shneiderman (1995) concluded his critical view on adaptive interfaces and agents with:

> ''But, possibly, [. . .] agents will merely become the Pet Rock of the 1990s – everyone knows they're just for fun.''

It has to be noted that the Pet Rock was not just a fun idea but a great commercial coup including profitable publications and even several copycats to satisfy the sudden buyer's needs.[25] Accordingly, in fierce e-business competition, a ''Me too'' agent is not just nice to have, but a necessity for surviving in the market. Furthermore, every simple extension to such an idea must be seen as an added value that increases the chance of survival. Although it may sound polemic, this argument is as valid as any other argument that demands solutions in ML and UM as required by industry.

On the other hand, it is argued that *being fun* is an argument good enough to justify building such agents: even if they do not increase performance or even make interaction easier, they might appeal to the user somehow and at least make interaction somehow bearable. An agent could at least try to cheer up the user or sympathise with him (see Preece (1994)). The problem is to understand how an agent could cheer up a person who disapproves of the abilities of the agent or a system the user identifies with such an agent. At this point, one might think of Lehrer (1965), who studied human–human interaction in the early 1960s:

> [. . .] one problem [. . .] is the inability of people to communicate [. . .]. And [people] spend hours bemoaning the fact that they can't communicate. I feel that if a person can't communicate, the very least he can do is to shut up!

For adaptive agents this means that if you have nothing to contribute, then get out of the way.

4.1.2 *Should we do it or not?*

As it is commonly known, opinions on this topic differ significantly. One answer to this question is that user models should be learned in order to make an intelligent, adaptive system that makes interaction easier and more comfortable and, thus, increases user satisfaction (and, in this way, increases productivity too). Another answer is that nothing can be learned such that it is sufficiently reliable. Even more, systems should not adapt at all (no matter whether this is by learning or any other method), as adaptation is, in general, unwanted by the user or imposes an additional cognitive effort on the user for constantly re-learning how to interact.

There is, however, a constructive and still very cautious approach to adaptive systems which has been presented in Miller (2000). It is argued that Gricean conversational maxims (Grice, 1975) also apply to the UM problem, which leads to the conclusion that not everything that can be done, should be done.

Any ''good'' AUI must provide the user with the right (''relevant'') information and it must provide the user with the right amount of information. While the concern of the first rule is similar

[25] Needless to say, the inventor, Gary Dahl, is an advertising expert.

to the question ''what to say'' (in the sense of content selection), the second rule deals with the problem of ''how to say it'' (in the sense of information presentation or choice of media). Both steps crucially depend on our definition of relevance.

A system that statically models the user's interest and the context would not help or guide, but rather force the user into a certain communication situation. This would inherently contradict the first aim of UM, namely to adapt to the user in order to make the interaction easier for him. Thus, as a rule of thumb, if a system should be adaptive, it at least needs to act as a subordinate communication partner.

It is essential for the user to have the feeling of understanding the system. Thus, the user needs to have an idea about what the system knows and what the system assumes about the user. In this way, the user is able to understand *why* the system performed a certain step towards adaptation. Therefore, a good AUI must be able to explain its actions to the user or at least be lucid and scrutable in its behaviour.

Furthermore, systems that promise real ''intelligent'' behaviour suggest functionality that the system may not offer – something that is likely to occur when designing ''cute agents''. It is argued that the reason for this behaviour is that humans build models of how such systems work internally. Acceptance then depends on whether the system behaves as the user expects on the basis of his model of the system.

As we try to decrease a user's cognitive effort, adaptive systems need to act in the background, but without feedback the system has no chance to determine whether it made a mistake, what the user is interested in or whether a context change demands another reaction. Obviously, user adaptation without feedback is impossible. This seems to conflict with another commandment to which all other HCI conversational maxims can be reduced:

> ''Never ever bother the user.''[26]

Thus, we need to get going with as little feedback as possible, asking for additional information as seldomly as possible and presenting as few questionnaires as possible. In consequence, an adaptive system should be able to interpret the natural user behaviour on all levels as user feedback, without the need for any further explicit feedback. The user model built upon this knowledge should be inspectable and easy to understand for the user, the adaptive process itself should be scrutable, self-explaining and, if necessary, the user should be able to override system actions.

4.2 PAC UM?

Throughout this article, the UM problem was related to the ML problem. Now let us relate it to the PAC-learning model of Section 2.2. Accordingly, one would have to determine whether some $\mathbf{M}_u$ is ''ε-good''. In cases where Δ is unknown, hypotheses h are evaluated against a test set $\mathbf{f}_{\text{test}}$, which is taken from the sample $\mathbf{f}$.[27] Assuming that the sample somehow represents Δ on $\mathfrak{U}$ (which is the underlying inductive hypothesis), the error set on $\mathbf{f}_{\text{test}}$ can be used to estimate ε. However, the method of testing with $\mathbf{f}_{\text{test}}$ cannot be compared with ML: the aim in ML is to satisfy $h \sim \mathbf{t}$ (which can be tested by the method described above), but in UM the sample $\mathbf{f}$ is used to learn some $\mathbf{M}_u \sim \mathbf{i}_u \neq \mathbf{o}$. Accordingly, the error measured when comparing $\mathbf{M}_u$ and $\mathbf{o}$ is the predictive error of $\mathbf{M}_u$ on observable user interactions, not the error in the user model itself.

To satisfy the requirements of HCI, one would need to test the agreement between$\mathbf{M}_u$ and $\mathbf{i}_u$. For this, one would need either an expert who evaluates $\mathbf{M}_u$ or an experiment where we would have to measure accuracy in interactions with real users.

Finally, determining a confidence in ε-goodness of hypotheses one needs to consider the probability of being able to learn a user model that has an error of at most ε. In the PAC model,

[26] A rule that immediately gives rise to the question, at which point does a ''funny'' cheering-up agent become a bothering one.

[27] As in ML, examples from $\mathbf{s}_{\text{test}}$ ($\mathbf{f}_{\text{test}}$) are not taken for learning.

this confidence value corresponds to δ. In practice, one would need to verify a user adaptive system in a test series with many participants.

The above considerations may seem to be pure sophistry, but as we should always be concerned with being clear about what is being modelled, what the domain restrictions look like and what biases are involved, we should also be clear about what should be evaluated. Accordingly, most evaluations carried out in the research described in this article focus on the ML part – here, $\mathbf{M}_u$ and $\mathbf{o}$ were compared as it is done with h and $\mathbf{t}$ (an exception is the READY project, see Section 3.2.2).

Real users for a valid evaluation are also required for determining ''user satisfaction''.[28] User satisfaction is what needs to be maximised in HCI – and sometimes it may turn out that the chosen modeling approach may help to derive a very accurate user model, but it does not make interaction better or easier at all.

4.3 User models can be learned, if . . .

The question we discussed in this article is whether user models can be learned at all. The answer is: ''*Yes, but . . .*'' as there are many, many requirements of which we always should be aware. In conclusion, models can be learnt if as many as possible of the following demands can be satisfied.

1. The content of the user model is self-contained and clearly restricted; and so is the domain.
2. The domain is sufficiently well described.
3. There is a legitimate tradeoff between scrutability, adaptivity and domain modeling effort.
4. We can live with assumptions on the domain and biases in learning.
5. There are enough noise-free data available.

There is, however, a chance to go a step further: if we are able to achieve more knowledge about the process that makes a user interact in a certain way, we might be able to de-noisify $\mathbf{f}$ by understanding the intended actions. This idea has already been partially worked on in plan recognition and affective interaction, but it seems to be a fruitful further research direction to integrate these approaches. As a consequence, an intelligent AUI will be able to ''understand'' observed interactions $\mathbf{o}(x)$ and thus yield a better approximation of the teaching signal $\mathbf{t}$.

This seems to motivate a joint research effort, where we combine different techniques to overcome drawbacks by another method's advantage. One should also try to build systems that are able to satisfy needs of other approaches or paradigms that currently do not seem to be in the focus of specialised research. This smells a bit like eclecticism, which is often a subject of criticism. However, in order to design increasingly intelligent AUIs one needs to unite several approaches in order to satisfy growing demands. Demands and research activity *are* growing and current issues of UM have overcome the early state of infancy when long-term UM (of personal information) and intention recognition were still the subject of harsh criticism (Kobsa, 1990).

Accordingly, this article is meant to serve as an argument for joint collaboration so we can answer ''*Yes, user models are learnable*!'' to the title question without the danger of wasting goodwill and/or reputation.

References

Bauer, M, 1996, A Dempster–Shafer approach to modeling agent preferences for plan recognition. *User Modeling and User-Adapted Interaction* **5**(3–4), 317–348.

Bauer, M, 1998, Towards the automatic acquisition of plan libraries. *Proceedings of ECAI-98*.

Bauer, M, 1999, From interaction data to plan libraries: a clustering approach. *Proceedings of IJCAI-99*.

Billsus, D, Brunk, CA, Evans, C, Gladish, B and Pazzani, M, 2002, Adaptive interfaces for ubiquitous Web access. *Communications of the ACM* **45**(5), 34–38.

[28] By satisfaction we subsume all requirements that are to be satisfied to guarantee high efficiency, low cognitive load, less stress, and high-quality throughput, etc.

Billsus, D and Pazzani, M, 1997, Learning probabilistic user models. In Jameson, A, Paris, C and Tasso, C (eds.), *User Modeling: 6th International Conference (UM97)*. New York: Springer.

Billsus, D and Pazzani, M, 1999, A hybrid user model for news story classification. In Kay, J (ed.), *User Modeling: Proceedings of the 7th International Conference (UM99)*, New York: Springer, pp. 99–108.

Bueno, D and David, AA, 2001, Metiore: a personalized information retrieval system. In Bauer, M, Gmytrasiewicz, P and Vassileva, J (eds.), *User Modeling: Proceedings of the 8th International Conference (UM2001)*. New York: Springer.

Castillo, E, Gutierrez, JM and Hadi, AS, 1997, *Expert Systems and Probabilistic Network Models*. New York: Springer.

Cockram, T, Hall, PA and Ince, DC, 2003, A model for inspection efficiency prediction. *Technical Report 2003/12*, Department of Computing, Open University, Milton Keynes, UK.

Cowell, RG, Dawid, AP, Lauritzen, SL and Spiegelhalter, DJ, 1999, *Probabilistic Networks and Expert Systems*. New York: Springer.

Craven, M, DiPasquo, D, Freitag, D, McCallum, A, Mitchell, T, Nigam, K and Slattery, S, 1998a, Learning to extract symbolic knowledge from the WorldWideWeb. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*. Menlo Park, CA: AAAI Press.

Craven, M, DiPasquo, D, Freitag, D, McCallum, A, Mitchell, T, Nigam, K and Slattery, S, 1998b, Learning to extract symbolic knowledge from the world wide web. *Technical Report CMU-CS-98–122*, School of Computer Science, Carnegie Mellon University.

Gladden, J, 1986.

Gmytrasiewicz, PJ and Lisetti, CL, 2001, Emotions and personality in agent design and modeling. In Bauer, M, Gmytrasiewicz, P and Vassilev, J (eds.), *User Modeling: Proceedings of the 8th International Conference (UM2001)*. New York: Springer.

Grice, HP, 1975, Logic and conversation. In Cole, P and Morgan, JL (eds.), *Syntax and Semantics, 3: Speech Acts*. New York: Academic Press.

Grosz, BJ and Sidner, CL, 1990, Plans for discourse. *Intentions and Plans in Communication and Discourse*. Cambridge, MA: MIT Press.

Hyafil and Rivest 1976.

Jacobs, N and Blockeel, H, 2001, The learning shell: automated macro construction. In Bauer, M, Gmytrasiewicz, P and Vassileva, J (eds.), *Proceedings of the 8th International Conference on User Modeling*. New York: Springer.

Jameson, A, 2003, Adaptive interfaces and agents. *Handbook of Human–Computer Interaction in Interactive Systems*, Erlbaum.

Jameson, A, Großmann-Hutter, B, March, L and Rummer, R, 2000, Creating an empirical basis for adaptation decisions. In Lieberman, H (ed.), *IUI2000: International Conference on Intelligent User Interfaces*. New York: ACM Press.

Kay, J and McCreath, E, 2001, Automatic induction of rules for e-mail classification. In Schäfer, R, Müller, ME and Macskassy, SA (eds.), *Proceedings of the UM2001 Workshop on Machine Learning for User Modeling*, pp. 59–66.

Kearns, MJ, 1990, *The Computational Complexity of Machine Learning*. Cambridge, MA: MIT Press.

Kearns, MJ and Vazirani, UV, 1994, *An Introduction into Computational Learning Theory*. Cambridge, MA: MIT Press.

Kobsa, A, 1988, User models in dialog systems. In Kobsa, A and Wahlster, W (eds.), *User Models in Dialog Systems*. New York: Springer.

Kobsa, A, 1990, User modeling in dialog systems: potentials and hazards. *AI and Society* **4** (3), 214–240.

Kobsa, A, 2001, Generic user modeling systems. *User Modeling and User–Adapted Interaction* **11,** 49–63.

Kobsa, A and Trost, H, 1984, Representing belief models in semantic networks. In Trappl, R (ed.), *Cybernetics and Systems Research II*. Amsterdam: North-Holland.

Kobsa, A and Wahlster, W (eds.), 1988, *User Models in Dialog Systems*. New York: Springer.

Kruegel, C, Mutz, D, Robertson, W and Valeur, F, 2003, Bayesian event classification for intrusion detection. *19th Annual Computer Security Application Conference*. Piscataway, NJ: IEEE Press.

Küpper, D, 2002, User modeling for user specific plan generation and presentation. *PhD Thesis*, University of Essen (in German).

Lehrer, T, 1965, *That Was The Year That Was*.

Lesh, N, Rich, C and Sidner, CL, 1999, Using plan recognition in human–computer collaboration. In Kay, J (ed.), *User Modeling: Proceedings of the 7th International Conference (UM99)*, New York: Springer, pp. 99–108.

Lesh, N, Rich, C and Sidner, CL, 2001, Collaborating with focused and unfocused users under imperfect communication. In Bauer, M, Gmytrasiewicz, P and Vassilev, J (eds.), *User Modeling: Proceedings of the 8th International Conference (UM2001)*. New York: Springer.

Li, M and Vitanyi, P, 1997, *An Introduction to Kolmogorov Complexity and its Applications*, 2nd edn. New York: Springer.

Lieberman, E (ed.), 2001, *Your Wish is my Command*. Morgan Kaufmann.

Lindmark, K, 2000, Interpreting symptoms of cognitive load and time pressure in manual input. *Master's Thesis*, Department of Computer Science, Saarland University, Germany.

Lindmark, K and Heckmann, D, 2000, Interpreting symptoms of cognitive load and time pressure in manual input. In Müller, ME (ed.), *Proceedings of the 8th GI Workshop ''Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen''*. Institut für Semantische Informationsverarbeitung, Universität Osnabrück.

Macskassy, S, Dayanik, AA and Hirsh, H, 2000, Information valets for intelligent information access. In Rogers, S and Iba, W (eds.), Adaptive user interfaces, *Technical Report SS–00–01*. Menlo Park, CA: AAAI Press, pp. 68–73.

Madrid, JM and Gauch, S, 2002, Incorporating conceptual matching in search. *11th Conference on Information and Knowledge Management (CIKM'02)*. Submitted.

Miller, CA, 2000, Rules of Etiquette – or How a Mannerly AUI should Comport Itself to Gain Social Acceptance and be Perceived as Gracious and Well-Behaved in Polite Society. *AAAI Spring Symposium on Adaptive User Interfaces*. Menlo Park, CA: AAAI Press, pp. 80–81.

Morik, K, 1986, Discourse models, dialog memories, and user models. *Computational Linguistics* **14**(3).

Morik, K, 1988, User models and conversational settings: modeling the user's wants. In Kobsa, A and Wahlster, W (eds.), *User Models in Dialog Systems*. New York: Springer.

Müller, C, Großmann-Hutter, B, Jameson, A, Rummer, R and Wittig, F, 2001, Recognizing time pressure and cognitive load on the basis of speech: an experimental study. In Bauer, M, Gmytrasiewicz, P and Vassileva, J (eds.), *User Modeling: Proceedings of the 8th International Conference (UM 2001)*. New York: Springer.

Müller, ME, 2001, Inducing content based user models with inductive logic programming techniques. In Schäfer, R, Müller, ME and Macskassy, SA (eds.), *Proceedings of the UM2001 Workshop on Machine Learning for User Modeling*, pp. 67–76.

Müller, ME, 2002, Inducing conceptual user models. *PhD Thesis*, Institute of Cognitive Science, University of Osnabrüuck. Electronic publication.

Murray, IR and Arnott, JL, 1993, Toward the simulation of emotion in synthetic speech: a review of literature on human vocal emotion. *Journal of Acoustical Society of America* **93**(2).

Ortony, A, Clore, GL and Collins, A, 1988, *The Cognitive Structure of Emotions*. Cambridge: Cambridge University Press.

Perrault, CR and Allen, JF, 1980, A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics* **6**(3–4).

Picard, RW, 1997, *Affective Computing*. Cambridge, MA: MIT Press.

Pollack, M, 1990, Plans as complex mental attitudes. In Cohen, PR, Morgan, J and Pollack, M (eds.), *Intentions in Communication*. Cambridge, MA: MIT Press, pp. 77–103.

Preece, J, 1994, *Human–Computer Interaction*. Reading, MA: Addison-Wesley.

Pretschner, A and Gauch, S, 1999 Personalization on the Web. *Technical Report ITTC-FY2000–TR-13591–01*, Department of Electrical Engineering and Computer Science, University of Kansas.

Quinlan, JR, 1990, Learning logical definitions from relations. *Machine Learning* **5**(3), 239–266.

Quinlan, JR, 1993, *C 4.5–Programs for Machine Learning*. Morgan Kaufmann.

Rich, C and Sidner, CL, 1998 Collagen: a collaboration manager for software interface agents, **8**(3–4), 315–350.

Shannon, C and Weaver, W, 1949, The mathematical theory of communication. *Technical Report*, University of Illinois, Urbana, ILL.

Shneiderman, B, 1995, Looking for the bright side of user interface agents. *ACM Interactions*, 13–15.

Stern, MK and Woolf, BP, 2000 Adaptive content in an online lecture system. In Brusilovsky, P, Stock, O and Strappavara, C (eds.), *Adaptive Hypermedia and Adaptive Web-Based Systems (AH-2000)* (*Lecture Notes in Computer Science*, vol. 1892). Berlin: Springer.

Suryadi, D and Gmytrasiewicz, P, 1999, Learning models of other agents using influence diagrams. In Kay, J (ed.), *User Modeling: Proceedings of the 7th International Conference (UM 99)*. New York: Springer.

Valiant, LG, 1984, A theory of the learnable. *Communications of the ACM* **27,** 1134–1142.

van Rijsbergen, CJ, 1979, *Information Retrieval*. Butterworths.

Webb, GI, Pazzani, MJ and Billsus, D, 2001, Machine learning for user modeling. *User Modeling and User–Adapted Interaction* (11), 19–29.

Weld, DS, 1999, Recent advances in AI planning. *AI Magazine* **20**(2).

Welsh, D, 1991, *Codes and Cryptography*. Oxford: Clarendon/Oxford University Press.