

Incremental learning of user models - an experimental testbed

T. P. Martin

University of Bristol

Advanced Computing Research Centre / Department of Engineering Mathematics

Phone/Fax +44 117 928 8200

Trevor.Martin@bristol.ac.uk

Abstract

The next generation of consumer goods, including computers, will be much more sophisticated in order to cope with a less technologically literate user base. A user model is an essential component for “user friendliness”, enabling the behaviour of a system to be tailored to the needs of a particular user. In this paper, we briefly outline the problems of user modelling, especially uncertainty, understandability, adaptability and the difficulty of obtaining data. A new user modelling approach (FILUM) is described, using inheritance from prototypes which are implemented as support logic programs. An experimental testbed based on the iterated prisoner’s dilemma, which allows the generation of unlimited data for learning or testing, is proposed. The system is implemented using Fril++, an object-based logic programming language with uncertainty.

Keywords: user modelling, iterated prisoner’s dilemma, Fril, logic programming, support logic, uncertainty.

1 Introduction

The next generation of consumer goods, including computers, will be more sophisticated in terms of user-modelling and intelligent help systems, to cope with a less technologically literate user base. For example, an integrated home information / entertainment system (computer, VCR, TV, hi-fi, etc) should be able to suggest TV/video choices

based on past preferences, and automatically record programmes judged to be interesting to a user. With the increased access to information arising from the web and integration of digital TV and computer networking, this area of intelligent consumer goods is an extremely important next step.

We define user modelling to be the provision of a software sub-system able to observe and predict the actions of a user (from a limited set of possibilities), with the aim of improving the overall interaction between user and system. This is a relatively “soft” definition, as the quality of interaction is almost always a subjective judgement, and it is therefore difficult to discuss the success (or otherwise) of user modelling. In this paper, we outline a new approach to user modelling and an experimental testbed based on the iterated prisoner’s dilemma, which allows the generation of unlimited data.

2 User Modelling

Although some products already include rudimentary user-modelling capabilities, there is a long way to go before “intelligent” user modelling becomes a reality. There are two developments driving the need for user modelling. Software is becoming vastly more capable and complicated, but most people use only a small subset of those capabilities. This leads to a need for the software to “customise” itself in order to help the user work (or play) more effectively. The Microsoft Office assistant is an example of this. The need for automatic customisation is also relevant for many “appliances” - from consumer electronics and household appliances through to cars.

The second driver for user modelling arises from the problems of information overload and the rapidly

increasing amount of data available electronically (for example via the internet). Getting through to the important pieces of information requires some kind of automated filtering which in turn means that the software needs information on a user's interests - i.e. it needs some kind of user model. Pointcast news delivery systems and personalised TV channels are examples of the first steps towards this goal [18,19,21].

There is a clear role for the methods of computational intelligence in user modelling. It is difficult for a programmer to write a user model; the user model should also adapt and evolve to reflect the user's current activities and interests, i.e. it needs to be able to learn. This puts the user modelling problem into the domain of machine learning.

User modelling needs to be relatively transparent to a user. We can think of email filters and kill files as an elementary form of user modelling - the user is in complete control and there is no automation. Taking a simple step forward, we could allow the system to consider the set of emails which are deleted without being read. If the system could find a common feature in those emails - in other words, a rule which picks out those emails - then it could offer the user the choice of using that rule in future to kill emails without further user intervention. A set of examples such as

title	sender	action
make money!!!	abc@hotmail.com	delete
re: Project Meeting	boss@acme.com	read and save
free \$\$\$	freemoney@aol.com	delete
Fuzzy Conference	fuzzy@MailList.net	read and delete
... etc		

might lead to rules such as

```

if title includes $ or money
then action = delete

if sender = boss
then action = 'read and file'

if sender = mailing list
then action = 'read and delete'
```

This requires methods which can generate rules that the user can understand and change if necessary.

The example above is a conventional propositional learning task, and a number of algorithms exist to create rules or decision trees on the basis of data such as this [4,7,16,17]. Typically, the problem must be expressed in an attribute-value format, as above; some feature engineering may be necessary to enable efficient rules to be induced. Rule-based knowledge representation is better than (say) neural nets due to better understandability of the rules produced - the system should propose rules which the user can inspect and alter if necessary.

Bayesian nets have also been used successfully in user modelling based on attribute-value data [11].

3 Problems in User Modelling

User modelling is inherently uncertain – as Horvitz [11] observes,

“Uncertainty is ubiquitous in attempts to recognise and agent's goals from observations of behaviour”, and even strongly logic-based methods [15] acknowledge the need for “graduated assumptions”. There may be uncertainty over the feature definitions (for example

```

if sender is a close colleague
then action = read very soon,
```

where *close colleague* and *very soon* are fuzzily defined terms) or uncertainty over the applicability of rules (for example

```

if user has selected several options
from a menu and undone each action,
then it is very likely that the user
requires help on that menu
```

where the conclusion is not always guaranteed to follow from the conditions).

It is an easy matter to say that uncertainty can be dealt with by means of a fuzzy approach, but less easy to implement the system in a way that satisfies the need for understandability. The major problem with many uses of fuzziness is that they rely on “intuitive semantics”, which a sceptic might translate as “no semantics at all”. It is clear from the fuzzy control literature that the major development effort goes into adjusting membership functions to tune the controller. Bezdek [9] suggests that membership functions should be “adjusted for maximum utility in a given situation”. However, this leaves membership functions with no objective

meaning - they are simply parameters to make the software function correctly. For a fuzzy knowledge based system to be meaningful to a human, the membership functions should have an interpretation which is independent of the machine operation - i.e. one which does not require the software to be executed in order to determine its "meaning". Probabilistic representations of uncertain data have a strictly defined interpretation, and the approach adopted here uses Baldwin's mass assignment theory and voting model semantics for fuzzy sets [2,8].

One problem with propositional learning approaches is that it is difficult to extract relational knowledge - for example,

```
if several identical emails arrive
consecutively from a list server,
then delete all but one of them.
```

Also, it can be difficult to express relevant background knowledge such as

```
if person has an email address at
acme.com
then person is a work colleague.
```

These problems can be avoided by moving to relational learning, such as inductive logic programming [14], although this is not without drawbacks as the learning process becomes a considerable search task.

Additional problems arise from the need to update the user model. Most machine learning methods are based on a relatively large, static set of training examples, followed by a testing phase on new or previously unseen data. The incorporation of new training examples can normally be addressed only by restarting the learning process with a new, expanded, training set. As the learning process is typically quite slow, this is clearly undesirable. Additionally in user modelling it is relatively expensive to gather training data - explicit feedback is required from the user, causing inconvenience. The available data is therefore more limited than is typical for machine learning. The final problem is one of evaluation - user modelling is intended to enhance the efficiency of the whole system, human

and computer, and this is almost inevitably a subjective judgement. Repeatable and reliable trials are very difficult to organise.

To summarise, as distributed computing and information appliances become more widespread, the need for user modelling will increase. To be successful, the user model must

- improve the user-machine interaction
- be gathered unobtrusively, by observation or with minimal effort from the user
- be updateable - adapting to changes in user preferences and work patterns
- be understandable and changeable by the user - both in terms of the knowledge held about the user and in the inferences made from that knowledge
- be correct in actions taken as well as in deciding when to act

In short, user modelling has to be transparent, learnt without interfering with the user's work, and done well or not at all.

4 The FILUM approach

Flexible Incremental Learning of User Models (FILUM) is described elsewhere [12]. Instead of representing a user as a set of attribute-value pairs, we define a set of prototypical behaviours implemented as Fril programs. Each of these programs defines a class, which can use as much (or as little) local storage and processing (facts and rules) as necessary. A user model is treated as an instance which has a probability of belonging to each class according to how well the class behaviour matches the observed behaviour of the user. The probabilities are expressed as support pairs, and updated each time a new observation of the user's behaviour is made.

We start with a set of classes

$$C = \{c_1, c_2, \dots, c_k\},$$

a set of possible output values

$$B = \{b_1, b_2, \dots, b_n\}$$

emphasised that the game itself is of no interest here; it is used purely as a platform for generating data to test user modelling approaches.

The Prisoner’s Dilemma is a well-known example of a non-zero sum game in game theory. As an illustration, consider two prisoners who are jointly charged with a crime for which the standard

sentence is five years. They are separately offered a deal whereby they can stay silent (co-operate with their partner) or defect, providing evidence to implicate their partner. Each can choose to co-operate or defect, and the reduction in their sentences according to their joint behaviour is given in the following table:

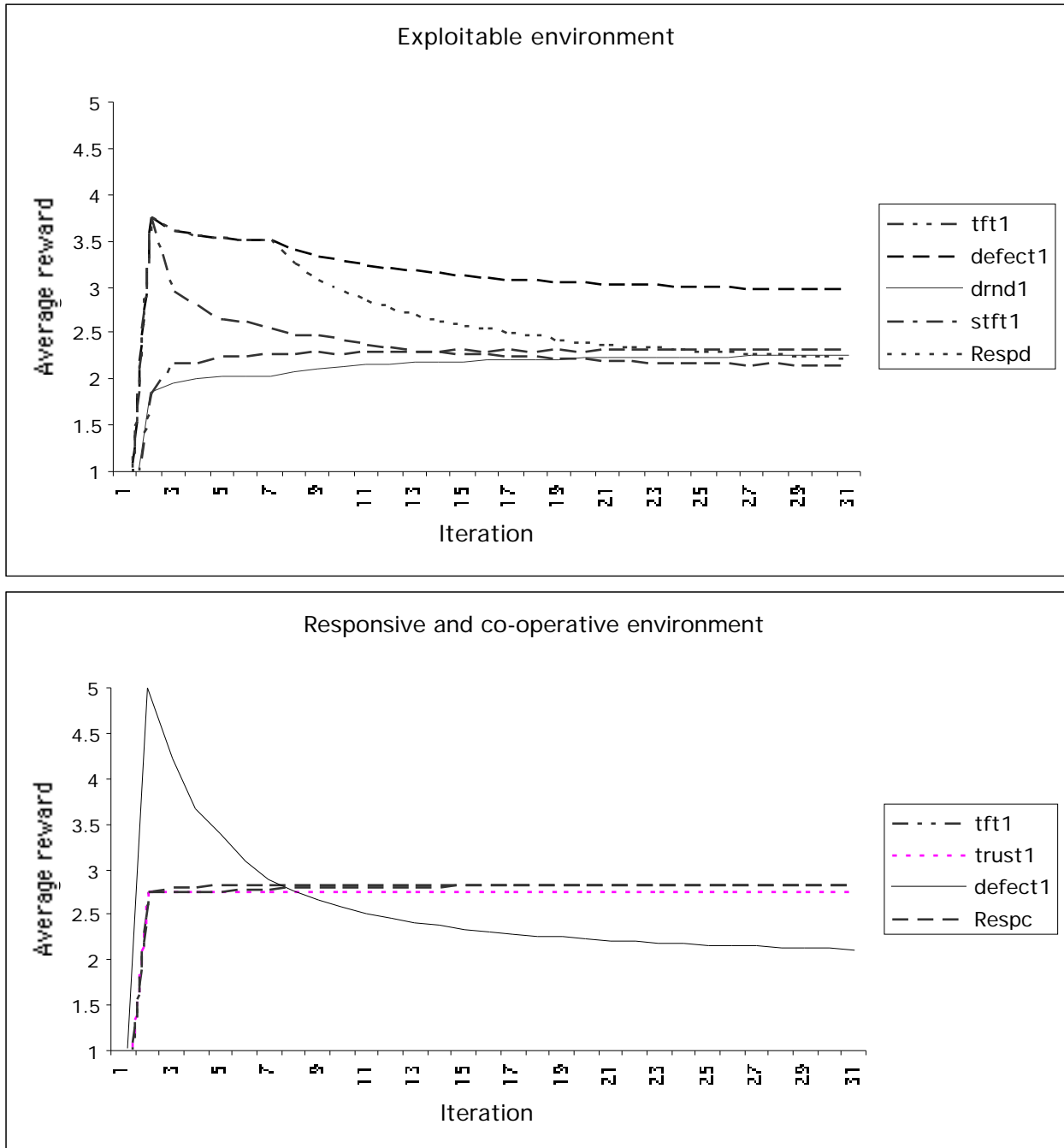


Figure 1: Average reward per interaction for various players in two n-IPD tournaments (a) exploitable environment, where the top line represents a player defecting on every move, and (b) responsive environment where the defecting strategy scores highly at first but then performs poorly. Twenty strategies were represented in each tournament, drawn from those listed in Section 7. It is possible for the same strategy to be included several times as different “players” in the tournament.

player1 / player2	co-operate	defect
co-operate	3 / 3	0 / 5
defect	5 / 0	1 / 1

From each individual's point of view, the rational strategy is to defect; collectively the best solution is for both to co-operate.

The iterated version extends the game to a sequence of interactions, where each player has access to the history of interactions. The n-player version considers more than two individuals. In each round, each pair of players participates in a pairwise interaction as above.

In all cases, each player aims to maximise their own score. There is an incentive to co-operate (e.g. the payoff from three co-operative interactions, c-c, c-c, c-c will be 9 whereas the payoff from one "exploitative" interaction and two mutual defections (d-c, d-d, d-d) will only be 7).

The iterated prisoner's dilemma [1] has been used to explain arms race escalation, the formation of economic cartels, and evolutionary biology, as well as acting as a test bed for multi-agent systems [10] and evolutionary programming. It can be shown that there is no optimum strategy, as much is dependent on the environment. For example if all other players co-operate irrespective of one's own actions, the optimum strategy is to defect. If all players respond by echoing one's last move (the "tit-for-tat" strategy) then the optimum strategy is to co-operate. The game was widely publicised in computer tournaments e.g. [13,20] in which co-operative strategies tended to do best. In particular successful strategies are

- "nice" in that they do not defect without provocation
- "responsive" in that they punish provocation (i.e. defection by the other player)
- "forgiving" in that they will attempt to co-operate again after punishing an opponent

6 n-player Iterated Prisoner's Dilemma as a Test Bed for User Modelling

The n-IPD is a good test bed for user modelling as it is possible to generate as much data as necessary and the true behaviour can be obtained in each case, so that it is possible to get an objective evaluation of the predictive accuracy of the user model.

The aim is to replace a selected player in an n-player tournament by a user model, which behaves in the same way as the player without knowledge of or access to the internal structure of the player, i.e. without knowing the code governing the player's behaviour. The user model has access to the interaction history of the selected player. There is a close analogy to the situation in user modelling applications, where the previous behaviour of a user is known without any detailed understanding of the "algorithm" that led to the decision.

There is no intention to judge whether a given strategy is successful or to optimise a strategy. At each iteration, the user model of a player P simply makes a prediction of P's behaviour in an interaction with each other player in the tournament.

7 Experiments

A Fril++[3,5,6] -based system was developed to run n-IPD tournaments, allowing a choice of strategies to be included in the environment. The number of players using each strategy could also be specified. Examples of strategies are:

- trust - always co-operates
- defect - always defects
- tft (tit-for-tat) initially co-operates, subsequently echoes whatever opponent did last time
- rand - random (50-50) choice
- crand - random (75-25) choice biased towards co-operation
- drand - random (25-75) choice biased against co-operation
- tfft - (tit-for-two-tats) co-operates unless two consecutive defections from opponent
- stft - sneaky tit-for-tat initially defects then echoes last opponent response
- massret (massive retaliation) - co-operates until a defection, then defects against that opponent forever
- other responsive strategies - a method must be supplied to determine initial behaviour, subsequently co-operate unless the number of defections by opponent in a given number of previous interactions exceeds some threshold, in which case defect (tit for tat, tit for two tats, massret are all subtypes of this strategy). Examples included in the tournaments are

- *respc* - co-operate unless all of the opponent's last 6 responses were d
- *respd* - defect unless all of the opponent's last 6 responses were c

Figure 1 shows the results from two sample tournaments, plotting the average reward per interaction for selected player in the population. If all interactions are co-operative, then the average will be 3; similarly if a player manages to exploit every other player then its average reward would be 5. In Figure 1 (a) there are too few responsive players, and too many that co-operate unconditionally. The “defect every time” strategy is able to exploit the other players and maintain a clearly better overall average.

In Figure 1 (b) the tournament contained a similar number of co-operative players but more were responsive, and withdrew co-operation from the the “defect every time” strategy. The latter does worse than the others, despite its initial success.

The precise details are not important, other than to note that the average score is higher in more co-operative environments, and that the best strategy in one environment is not necessarily the same as the best strategy in another. Such observations are in line with other n-IPD tournaments and help to verify that the software performs correctly.

From the point of view of user modelling, we aim to reproduce the behaviour of a player by means of some simple prototypes. The properties of successful players observed in tournaments are niceness, responsiveness, and forgiveness. In order to roughly model these characteristic behaviours, the four prototypes exhibit the following patterns :

- trusting - always co-operates
- defective - always defects
- responsive - identified by echoing the opponents last move
- provocative - identified by defecting when the opponent co-operated last time

(this can be viewed as the negation of nice)

Note that there is no prototype to explicitly identify “forgiving” behaviour, although the “responsive” prototype will effectively detect it.

Thus if we regard the interaction between two specified players P and Q as a sequence of pairs of actions (pi, qi), we are looking for instances of :

- (c, _) to support P belonging to the co-operative prototype
- (d, _) to support P belonging to the unco-operative prototype
- (c, d), (d, _) or (d, c), (c, _) to support P belonging to the responsive prototype
- (c, c), (d, _) to support P belonging to the provocative prototype

With a history of interactions generated by an n-IPD tournament, we can use these prototypes to model different players. Table 2 shows selected user models after 12 iterations of a 20 player tournament. The user models have converged after 12 iterations, i.e. the supports for the models belonging to each class do not change significantly after this iteration.

Table 2 : Prototype supports in user models derived from 12 iterations of an n-IPD tournament. Column headings refer to prototypes Trusting, Responsive, Provocative and Defective as defined above

Player	T	R	P	D
co-op	[0.9 1]	[0, 1]	[0. 0.09]	[0, 0.09]
uncoop	[0, 0.09]	[0, 1]	[0. 0.09]	[0.9 1]
tit-for-tat	[0.6,0.7]	[0.3,1]	[0,0.5]	[0.3,0.4]
random	[0.4,0.5]	[0.3,0.6]	[0.2,0.7]	[0.5,0.6]
respd	[0.2,0.3]	[0.1,0.8]	[0.1,0.9]	[0.7,0.8]

Prediction success rates vary between 40-60% for the random strategies, and 80-95% for the others.

The support for class membership is determined purely by the success of the class rule in predicting the behaviour of the user. Thus a player showing a strategy of “always defect” appears as highly unco-operative but has [0, 1] support for membership in the “provocative” class, as the player never exhibits the provocative behaviour pattern of mutual co-operation followed by a defection.

The models for the selected player are derived from all of the player's interactions; within each model, there are models for the other players derived from their interactions with the selected player.

Overall predictive success rates are good although the random strategies are difficult to predict, as would be expected. As a rule of thumb, if a user model is giving a success rate of less than 60% then a new prototype is required, either by a human

expert or using an inductive logic-type approach to generate prototypes which predict the observed behaviour more accurately. Note that these new prototypes need only explain the behaviour in a subset of cases; they can give uncertain support for all outcomes when the prototype is not applicable.

The success rate of a user model can easily be calculated by comparing the predicted and observed behaviour of the user. Clearly the user model changes with each new observation, and there is very little overhead in updating the user model. This approach depends on having a “good” set of prototypes, which are able to give a reasonable coverage of possible user behaviour. It is assumed that a human expert is able to provide such a set; however, it is possible that new prototype behaviours could be generated by techniques such as inductive logic programming. This is an interesting avenue for future research.

Acknowledgements

This work was partially supported by BT under the Short Term Research Fellowship Scheme.

References

1. Axelrod, R., *The Evolution of Cooperation*. 1984, New York: Basic Books.
2. Baldwin, J.F., *The Management of Fuzzy and Probabilistic Uncertainties for Knowledge Based Systems*, in *Encyclopedia of AI*, S.A. Shapiro, Editor. 1992, John Wiley. p. 528-537.
3. Baldwin, J.F., *et al. Implementing Fril++ for Uncertain Object-Oriented Logic Programming*, in *IPMU 2000*. 2000. Madrid.
4. Baldwin, J.F., J. Lawry, and T.P. Martin, *A mass assignment based ID3 algorithm for decision tree induction*. *International Journal of Intelligent Systems*, 1997. **12**(7): p. 523-552.
5. Baldwin, J.F. and T.P. Martin. *Fuzzy Classes in Object-Oriented Logic Programming*. in *FUZZ-IEEE-96*. 1996. New Orleans, USA.
6. Baldwin, J.F. and T.P. Martin. *Fuzzy Objects and Multiple Inheritance in Fril++*. in *EUFIT-96*. 1996. Aachen, Germany.
7. Baldwin, J.F. and T.P. Martin, *Basic Concepts of a Fuzzy Logic Data Browser with Applications*, in *Software Agents and Soft Computing: Concepts and Applications*, H.S. Nwana and N. Azarmi, Editors. 1997, Springer (LNAI 1198). p. 211-241.
8. Baldwin, J.F., T.P. Martin, and B.W. Pilsworth, *FRIL - Fuzzy and Evidential Reasoning in AI*. 1995, U.K.: Research Studies Press (John Wiley). 391.
9. Bezdek, J.C., *Fuzzy Models*. *IEEE Trans. Fuzzy Systems*, 1993. **1**(1): p. 1-5.
10. Carmel, D. and S. Markovitch, *Model-based learning of interaction strategies in multi-agent systems*. *Journal of Experimental & Theoretical Artificial Intelligence*, 1998. **10**(3): p. 309-332.
11. Horvitz, E., *et al. The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users*. in *Fourteenth Conference on Uncertainty in Artificial Intelligence*. 1998. Madison, WI.
12. Martin, T.P., *Flexible Incremental Learning of User Models - the FILUM Approach*. To be published, 2000.
13. Mowbray, M., *Evolutionarily stable strategy distributions for the repeated prisoner's dilemma*. *J.Theoretical Biology*, 1997. **187**(2): p. 223-229.
14. Muggleton, S., *Inductive Logic Programming*. 1992: Academic Press. 565.
15. Pohl, W., *Logic-based representation and reasoning for user modeling shell systems*. *User Modeling and User-Adapted Interaction*, 1999. **9**(3): p. 217-282.
16. Quinlan, J.R., *Induction of Decision Trees*. *Machine Learning*, 1986. **1**: p. 81-106.
17. Quinlan, J.R., *C4.5: Programs for Machine Learning*. 1993: Morgan Kaufmann.
18. ReplayTV, <http://www.replaytv.com/>, . 1999.
19. TiVo, <http://www.tivo.com/>, . 1999.
20. Yao, X., *Evolutionary stability in the n-person iterated prisoner's dilemma*. *Biosystems*, 1996. **37**(3): p. 189-197.
21. Yiming, Z. *Fuzzy User Profiling for Broadcasters and Service Providers*. in *Computational Intelligence for User Modelling*. 1999. Bristol, UK. (see <http://www.fen.bris.ac.uk/engmaths/research/aigroup/martin/CI4UMProceedings.html>)