Personalisation for User Agents

J.J. Cole, M.J. Gray, J.W. Lloyd, and K.S. Ng
Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University
{Joshua.Cole, Matt.Gray, John.Lloyd, Kee.Ng}@anu.edu.au

ABSTRACT

This paper is concerned with personalisation of user agents by symbolic, on-line machine learning techniques. The application of these ideas to an infotainment agent is discussed in detail. Also experimental results, which indicate that a high level of personalisation can be achieved by this approach, are presented.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*induction, knowledge acquisition*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*intelligent agents*

General Terms

Algorithms, Human Factors, Design

Keywords

User agents, symbolic learning, personalisation

1. INTRODUCTION

The goal of the research described in this paper is to apply machine learning techniques to building user agents that facilitate interaction between a user and the Internet. This paper concentrates on the topic of personalisation in which the agent adapts its behaviour according to the interests and preferences of the user. There are many practical applications of personalisation that could exploit the technology presented here.

The research is set in the context of an infotainment agent, which is a multi-agent system that contains a number of agents with functionalities for recommending movies, TV programs, music and the like, as well as information agents with functionalities for searching for information on the Internet. This paper concentrates on the TV recommender as a typical such agent and shows how a high degree of personalisation can be achieved by symbolic, on-line machine learning techniques. The techniques can be ported comparatively easily to other agents of the system, thus providing a fully personalised infotainment system.

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

There is an extensive literature on recommender systems; typical recent work is in [1]. Differently to that work and other work on recommender systems that we are aware of is the emphasis here on *symbolic* machine learning techniques to discover the user's interests and preferences. Instead of more typical machine learning techniques such as naive Bayes classifiers that are used in recommender systems, here we employ a decision-list learner that produces *comprehensible* hypotheses. This has two advantages: the agent can explain to the user why it has made a recommendation and users can use the hypothesis language to give explicit rules about what they want to be recommended. This approach comes from the field of inductive logic programming ([5], [8]) in which (first-order) logic is used as a knowledge representation language and in which there is a strong emphasis on learning comprehensible hypotheses. Apparently the only area of learning in agents where symbolic techniques have been employed so far is in relational reinforcement learning [3], [4]; thus this paper serves to introduce symbolic learning techniques to another problem of interest, that of personalisation of user agents.

Section 2 gives an outline of the infotainment agent, concentrating on its TV recommender. Section 3 briefly describes the approach to adaptation taken here. Section 4 describes how the TV recommender is personalised to the user. Section 5 presents results of experiments that show the level of personalisation achieved. Section 6 contains some conclusions and future directions for research.

2. AN INFOTAINMENT AGENT

In this section, we describe the infotainment agent being developed, concentrating on the TV recommender.

The infotainment agent is a multi-agent system that combines a number of related functionalities concerning information search and entertainment. The agents that comprise the system and that are at least partly implemented include a TV recommender, a movie recommender, a music recommender, a news agent, a search agent, and a diary agent. In addition, there is a coordinator agent that has the responsibility of handling interactions between the user and the various agents in the system. The TV recommender has potential as a personalisation component in a system such as TIVO, for example, which supports sophisticated search functions to find TV programs of interest to a user but never has any actual knowledge of the interests or preferences of the user. Also the music recommender has potential as a personalisation component for ITUNES and similar software.

A detailed description of the architecture of the TV recommender is now given. The architecture of the other agents in the infotainment agent is similar. What functionality do we want the TV recommender to have? When the user first begins to use the TV recommender it clearly has no knowledge of the interests or preferences

of the user. The aim is to design an adaptive architecture for the TV recommender so that within a comparatively short time, perhaps several weeks, it is able to make helpful recommendations to the user. Furthermore, it should improve its performance over longer periods and accurately track changing user interests and preferences. To get started, the agent presents a short questionnaire to the user the first time it is used. The purpose of the questionnaire is to acquire, with as little effort as possible on the part of the user, some initial idea of the user's interests and preferences. After that, the agent collects training examples by observing the user's activities. Over time, the agent is expected to be able to make recommendations for programs in specified time periods (say, 'next week' or 'tonight') that the user finds helpful.

A detailed description of the most pertinent aspects of the design of the TV recommender is now given. The approach to knowledge representation taken here follows that in [2] and [7], to which the reader is referred for the details. We will need several standard types: Ω (the type of the booleans), Nat (the type of natural numbers), Int (the type of integers), and $String$ (the type of strings). The intended meaning of the constant $\top$ of type Ω is true and that of $\bot$ is false. Also $List$ denotes the (unary) list type constructor. Thus, if α is a type, then $List\ \alpha$ is the type of lists whose elements have type α.

Three domain-specific types, $Channel$, $Genre$, and $Classification$, will also be needed. Here are the data constructors for these types.

$$
\begin{aligned}
&ABC, Adventure\_1, Animal\_Planet, Arena,\\
&Biography, BBC\_World, Cartoon\_Network,\\
&\qquad\vdots\\
&Sky\_News, TCM, Tech\_TV, Travel,\\
&TV1, UK\_TV, W, World\_Movies : Channel\\
&Action, ActionAdventureGroup, Adult, Animals,\\
&Animated, Art, ArtsMusicLiving, Auto,\\
&\qquad\vdots\\
&Volleyball, War, Watersports, Weather,\\
&Western, WesternGroup, Wrestling : Genre\\
&Y7, Y, G, MA, M14, M, NA : Classification.
\end{aligned}
$$

There are 49 channels, 115 genres and 7 classifications.

We introduce the following type synonyms.

$$
\begin{aligned}
&Date = Day \times Month \times Year\\
&Time = Hour \times Minute\\
&Title = String\\
&Subtitle = String\\
&Duration = Minute\\
&Synopsis = String\\
&Program = Title \times Subtitle \times Duration \times\\
&\qquad (List\ Genre) \times Classification \times Synopsis\\
&Year = Nat\\
&Month = Nat\\
&Day = Nat\\
&Hour = Nat\\
&Minute = Nat\\
&Text = List\ String.
\end{aligned}
$$

The agent has access via the Internet to a TV guide (for the next week or so) for all channels. This database is represented by a function tv_guide having signature

$$tv\_guide : Date \times Time \times Channel \rightarrow Program.$$

Here the date, time and channel information uniquely identifies the program and the value of the function is (information about) the program itself. The TV guide consists of (thousands of) facts like the following one.

$$
\begin{aligned}
&((tv\_guide\ ((20,7,2004),(20,30),ABC)) =\\
&\qquad (\text{“}The\ Bill\text{”}, \text{“”}, 50, [Drama], M,\\
&\qquad\ \text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking\ the\\
&\qquad\quad people\ smuggling\ operation\text{”})).
\end{aligned}
$$

This fact states that the program on 20 July 2004 at 8.30pm on channel ABC has title "The Bill", no subtitle, a duration of 50 minutes, genre drama, a classification for mature audiences, and synopsis "Sun Hill continues to work at breaking the people smuggling operation".

3. ADAPTATION

The adaptation approach of this paper uses on-line learning, which works as follows. Suppose we want to learn a classification function f. For this, several ingredients are needed. First, we need an initial definition of f to get started. Second, we need an hypothesis language which is the space of all possible definitions that f could have. Finding a suitable hypothesis language is one of the key design decisions that has to be made. Finally, we need a sequence of training examples that the learning algorithm can use to move from one definition of f to another. (See Figure 1.) The intuitive idea is that the learning algorithm chooses the definition that most closely agrees with the current set of training examples. For many agent applications, certainly the ones considered in this paper, it is important that the hypotheses be comprehensible to the user of the agent. This is achieved here by the use of logic as the language in which the hypotheses are expressed. Comprehensibility ensures that the initial function can be written directly and that the definition can be inspected at any later stage to help understand (and perhaps modify) the behaviour of the agent.

A training example is simply a pair consisting of a particular individual and the class for that individual. Hypothesis languages are expressed by predicate rewrite systems as discussed in [7]. Essentially, a predicate rewrite system is a grammar for constructing predicates from more basic ingredients. A predicate rewrite system for the TV recommender is given below.

The learning system we use is called ALKEMY ([2], [7]) and is a decision-list learner [9] that works as follows. (See Figure 2.) Starting from a set of training examples, the learner looks for a predicate such that the subset of examples whose individuals satisfy that predicate all have the same class (that is, the subset of examples is pure). The learner constructs a left child containing those examples. It then continues at the right child with the new set of examples obtained by removing the examples in the left child from the original set. The learner terminates when it reaches a right child with a pure set of examples. (It will also terminate if it is unable to find a predicate that splits off a pure, non-empty left node.) Leaf nodes in the list are labelled by the (majority) class of the examples at that node. However, when classifying new individuals, those that reach the last node (that is, the bottom right-hand leaf node) are not given a classification because we do not have enough confidence in

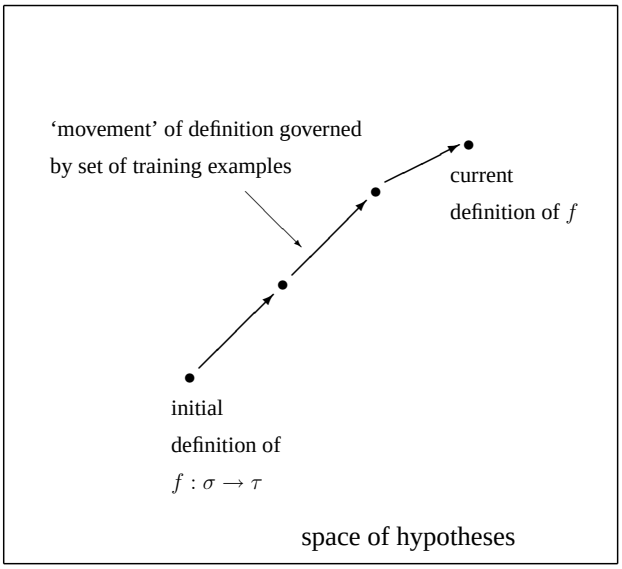

Figure 1: Adaptation in the space of hypotheses

the prediction. This means that the coverage of the learner is not 100%. If several predicates all produce a pure left node, the learner selects a predicate that produces the largest number of examples in that node. As shown in [9], this algorithm behaves well when there is little noise in the data.

In its on-line version (in contrast to the batch version), a decision-list learner receives a sequence of training examples over time and at any time may be asked to predict the class of some unseen individual based on its current decision list. Thus it is preferable that the learner be able to build decision lists incrementally, that is, it should be able to slightly modify the current decision list based on the next training example rather than have to rebuild the decision list from scratch each time. In its deployment in the infotainment agent, it is the on-line version of ALKEMY that we use. Furthermore, since a user's interests and preferences may change over time, it is likely that there are times when the current set of training examples is inconsistent, in the sense that the same individual may have two or more distinct classes in the training examples. This means that considerable care needs to be taken in deciding what should be the current set of training examples. For example, it is common to insist on some maximum size for the training set and to drop the oldest training examples as new ones are received to keep to this limit. We prefer the approach of returning the training set to consistency, even to the point of asking the user to resolve conflicts, if necessary. In this approach a training example could stay in the training set for a very long time and would only drop out if it contradicted another training example that was somehow confidently known to be correct. The current implementation uses a simple algorithm to check for consistency. More sophisticated schemes are being investigated.

As for all learning tasks, the main problem we face here is to decide which predicates should appear in decision lists, that is, what should be the hypothesis language. This problem is discussed for the TV recommender below.

4. PERSONALISATION OF THE TV RECOMMENDER

Now we turn to the personalisation aspects of the TV recommender. The key function that needs to be learned is the function $user\_likes\_tv\_program$ which takes a TV program as input and returns true if the agent considers the program to be worth recommending to the user; otherwise, it returns false. Thus the belief base of the TV agent contains the function $user\_likes\_tv\_program$ that has signature

$$user\_likes\_tv\_program : Program \to \Omega$$

and a definition that is a decision list of the form

$$\begin{array}{l}(user\_likes\_tv\_program\ x) = \\ \quad if\ (p_1\ x)\ then\ \top \\ \quad else\ if\ (p_2\ x)\ then\ \bot \\ \quad\quad \vdots \\ \quad else\ if\ (p_n\ x)\ then\ \top \\ \quad else\ \bot,\end{array}$$

where $p_1, \ldots, p_n$ are predicates on programs.

We now discuss the hypothesis language used by ALKEMY that contains these predicates $p_1, \ldots, p_n$ and is used to learn the definition of $user\_likes\_tv\_program$. The approach to constructing hypothesis languages is by means of predicate rewrite systems. The basic idea is to construct predicates by composing more basic ingredients. Thus we need the composition function

$$\circ : (a \to b) \to (b \to c) \to (a \to c)$$

defined by

$$((f \circ g)\ x) = (g\ (f\ x)).$$

The basic ingredients that are composed are made out of transformations [7]. The collection of transformations used in the application is now presented.

function $Learn(\mathcal{E}, \rightarrowtail)$ **returns** a decision list;
inputs: $\mathcal{E}$, a set of examples;
$\rightarrowtail$, a predicate rewrite system;

L := single node with examples $\mathcal{E}$;
S := the set of predicates defined by $\rightarrowtail$;
move to root node of L;
while set of examples $\mathcal{F}$ at node is not pure **do**
  foreach $p \in S$ **do**
    $\mathcal{F}_+ := \{(t, v) \in \mathcal{F} \mid (p\ t)\}$;
    $\mathcal{F}_- := \mathcal{F} \setminus \mathcal{F}_+$;
    if $\mathcal{F}_+$ is pure and non-empty **then**
      create left child with examples $\mathcal{F}_+$;
      create right child with examples $\mathcal{F}_-$;
      move to right child;
      break;
  if no split was found **then break**;
label each leaf node of L by the (majority) class of its examples;
return L;

Figure 2: Decision-list learning algorithm

We begin with two transformations whose definitions come from user interests and preferences, and so provide a way of personalising the hypothesis language used to learn the definition of $user\_likes\_tv\_program$. One of these is the function $genre$. To define this, we introduce the type synonym

$$Preference = Int,$$

where it is understood that only numbers in $\{-2, -1, 0, 1, 2\}$ are to be used as constants of type $Preference$. Then

$$genre : Genre \rightarrow Preference$$

is the function that maps each genre into an integer in the range -2 to 2, depending on how strong a preference the user has for that particular genre. Here is a typical definition of $genre$.

$$\begin{array}{l}(genre\ x) = \\ \quad if\ ((= Animals)\ x)\ then\ 1 \\ \quad else\ if\ ((= Animated)\ x)\ then\ -2 \\ \qquad\qquad \vdots \\ \quad else\ if\ ((= Wrestling)\ x)\ then\ -2 \\ \quad else\ 0.\end{array}$$

This definition states that the user has a modest liking for animal programs, a strong dislike of animation programs, and so on. The information in this definition is obtained by an initial questionnaire completed by the user and by belief update, if the user later changes his/her preferences. (Here 'belief update' is to be understood in the sense of updating a logical theory, not in the sense of the term as used in statistical machine learning.) Our experiments showed that the learner was able to make good use of the information given by the function $genre$.

Another transformation obtained from the user is

$$classification : Classification \rightarrow Preference$$

that gives information about the user's liking for programs having a certain classification. A typical definition of $classification$ could be as follows.

$$\begin{array}{l}(classification\ x) = \\ \quad if\ ((= Y7)\ x)\ then\ -2 \\ \quad else\ if\ ((= Y)\ x)\ then\ -2 \\ \quad else\ 0.\end{array}$$

The information in this definition is also obtained by an initial questionnaire.

The remaining transformations that follow are generic ones which essentially come from the types employed in the application [7].

For each type α, there is a transformation $top : \alpha \rightarrow \Omega$ defined by $top\ x = \top$. The predicate top is the weakest predicate on individuals (of type α).

For each component of the type $Program$, there is an associated projection function. For example,

$$projTitle : Program \rightarrow Title$$

is defined by

$$(projTitle\ (t, t', d, g, c, s)) = t.$$

Similarly, there are projections $projSubtitle$, $projGenre$, $projClassification$, and $projSynopsis$.

For each constant C of type $Genre$, there is a transformation

$$(= C) : Genre \rightarrow \Omega$$

defined by

$$((= C)\ x) = x = C.$$

Similarly, for each string S, there is transformation $(= S)$ that returns true iff its argument is identical to S.

For each integer N, there is a transformation

$$(< N) : Int \rightarrow \Omega$$

defined by

$$((< N)\ m) = m < N.$$

In a similar way, one can define the transformations $(> N)$, $(\geq N)$, and $(\leq N)$.

The transformation

$$StringToText : String \rightarrow Text$$

takes a string as input and returns the list of words in the order that they occur in the string (discarding white space between words). Furthermore, words in the output text are stemmed. Thus

$$(StringToText\ \text{“}High\ Technology\text{”}) = [\text{“}high\text{”}, \text{“}technolog\text{”}].$$

The transformation

$$listExists_1 : (String \rightarrow \Omega) \rightarrow Text \rightarrow \Omega$$

is defined by

$$listExists_1\ p\ t = \exists x.((p\ x) \wedge (member\ x\ t)).$$

The predicate $(listExists_1\ p)$ checks whether some text (that is, a list of strings) contains a string that satisfies p.

The predicate rewrite system for the function $user\_likes\_tv\_program$ is given in Figure 3. The actual strings S used in rewrites of the form

$$top \rightarrowtail (= S)$$

are the titles and subtitles of all the programs in the (current) set of training examples. In rewrites of the form

$$top \rightarrowtail (listExists_1\ (= S)),$$

the strings S appearing are computed as follows. First, the set of all stemmed words appearing in titles, subtitles or synopses of programs in the (current) set of training examples that are not stopwords is formed. Then for each word in this set we compute the ratio of the number of positive training examples (plus one) in which it appears divided by the number of negative training examples (plus one) in which it appears. The set of words is decreasingly ordered by this ratio and the top 100 are used in the rewrites. The intuition is that these 100 words are good for discriminating between positive and negative examples. Note that the predicate rewrite system is constantly changing as new training examples arrive.

A typical learned decision list for the function $user\_likes\_tv\_program$ is given in Figure 4. Such decision lists usually contain over a hundred decision nodes.

Users also have some level of direct control over the function $user\_likes\_tv\_program$ since it is possible to add user-defined rules to the learned definition of the function. For example, a user can add a rule such as:

> If the title is "Rugby Union" and the word "Australia"
> is in the synopsis, then true.

(This rule assumes the predicate rewrite system is enriched by also allowing conjunctions of conditions.) In general, the predicate in a rule can be any one that is obtainable from the predicate rewrite system. Since the user-defined rules are checked before the learned part of the definition, TV programs that satisfy these conditions are guaranteed to be classified in the way the user desires.

Figure 5 gives screen shots from the TV recommender in action. They show extracts from the TV guide. Program titles highlighted in green are recommended to the user. Titles of programs for which the TV recommender requires further training before it can make a prediction are presented in amber. (There are none of these in Figure 5.) The tick and cross buttons next to a program description are used to generate a positive or negative training example for that program. Pressing the query button displays a short explanation of why a recommendation was made. The figure shows recommendations for two different users for the same time slot. Note that the recommendations are quite different.

5. EXPERIMENTS AND RESULTS

Here we present the results of a number of experiments measuring the performance of the TV recommender. The four authors used the system over a two-week period, training it to personalise to their individual viewing preferences. One author trained the system twice, in different ways. All experiments were carried out on a 10-channel subset of the full TV guide, chosen by each user.

5.1 Experiment 1 – learning under favourable conditions

The first set of experiments was designed to test whether the TV recommender could personalise to different users when good training data was made available. Two users used the system in a rigid way, collecting training examples for two hours from the TV guide each day for two weeks. They provided feedback indicating their preference for every program in those two hours.

Charts (a) and (b) in Figure 6 are learning curves showing the performance of the system as more training examples were supplied. 10-fold cross-validation experiments were performed after every 10 examples up to 50, and every 25 examples thereafter. Cover, recall, precision and accuracy were calculated in the usual way. The last three values were calculated on the covered examples only. A Bezier curve of best fit was plotted on the resulting data points. The charts show a rapid rise in performance up to approximately 100 examples and a gradual improvement as further examples were added. The performance on all measures is near 90% after 500 examples, for both users.

5.2 Experiment 2 – learning under real conditions

The second set of experiments tested personalisation to different users under more realistic conditions. Three users used the system on two weeks of TV programming. They requested recommendations from any time slot and provided training examples to improve the correspondence between these recommendations and their viewing preferences. Users generally supplied corrective training examples for incorrect recommendations or programs for which the TV recommender indicated it was unsure. Reinforcing training examples could also be supplied.

Charts (d)–(f) in Figure 6 are learning curves plotted for each user as in the previous set of experiments. They show a steep rise in performance after early training and a more gradual improvement towards the end of the training period. The number of examples is generally smaller for each user than in the previous experiment. The final performance is generally lower than in the previous experiment, although still broadly improving at this point.

5.3 Experiment 3 – comparison of learning algorithms

This set of experiments evaluates the decision-list algorithm against more sophisticated learning algorithms.

The table in Figure 7 records the final number of examples (Ex) collected for each user as well as their positive (Ex+) and negative (Ex−) breakdowns. Also shown are decision-list 10-fold cross-validation results for cover (DL Cov), recall (DL Rec), precision (DL Prec) and accuracy (DL Acc) on the final datasets.

To compare these results, the learning algorithm AdaBoost [6] was adopted. AdaBoost is arguably the best off-the-shelf algorithm available, and its performance gives a good indication of the kind of accuracy attainable by other state-of-the-art algorithms.

We used single predicates defined by the predicate rewrite system given in Figure 3 as base classifiers. The number of iterations was set at 400 after some experimentation.

10-fold cross-validation results on the final data sets generated by each user are reported in the table in Figure 7 alongside the corresponding results achieved by the decision-list algorithm. In general terms, the numbers suggest that AdaBoost performs slightly better, but the decision-list algorithm is not far behind. (However this comparison is unfair to AdaBoost which makes a prediction on every example.)

This confirms that, in this particular application, the use of a symbolic learning algorithm does not incur a significant cost in terms of accuracy.

5.4 Discussion

In all the experiments the performance of the system was evaluated using 10-fold cross-validations. One could also use an independent test set to collect performance statistics. This was in fact done for each user and the results closely correspond. As an example we have included the test-set results for user 2 in Chart (c).

The first set of experiments suggest that it is possible for the TV recommender to personalise well to different users, given sufficient training data. This is due in part to the nature of the problem. Television programming tends to be rather repetitive. Programs with the same title are scheduled cyclically at daily, weekly and sometimes hourly periods. In these circumstances instance-based learning is expected to work well. This form of learning is captured implicitly by the use of predicates that test for equality of program titles in the decision lists. This works well in combination with more conventional rule-based learning that exploits more general conditions in the hypothesis language.

The second set of experiments suggest that real-world learning is more problematic. Collecting sufficient training data is difficult. Users in general prefer to give minimal feedback, mostly correcting mistakes as they occur. They also expect the TV recommender to make useful predictions on times of the day not previously trained on.

The results for user 3 show that even after a moderately large period of training the TV recommender can fail to perform as well as in the first set of experiments. This user chose two specialist movie channels from the TV guide and noted that the system performed poorly on movies compared with the more day-to-day TV programming of other channels. One reason for this is that there is in general an insufficient overlap between a person's preference for movies and TV programs for it to be possible to learn a theory that models both simultaneously. This suggests the need for a sep-

arate movie recommender with a more appropriate representation of movie individuals. We also note that the meta-data for movies in the TV guide was not very rich. For example, the genre for many movies was often simply designated as 'movie'.

Users 4 and 5 reported that subjectively the system performed well enough, and that further training to improve performance seemed unnecessary.

6. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we described the application of symbolic, on-line learning to personalisation of an infotainment agent, concentrating particularly on a TV recommender. With the qualification that more experiments are needed particularly in realistic deployments of the agent with typical users, the results suggest that a high level of personalisation can be achieved.

In on-going work, we are applying the same techniques to personalising other agents, especially the music recommender which is more challenging than the TV recommender. We also plan to make the movie recommender available to the TV recommender.

One promising technique for improving the performance of the agent is that of active learning. The idea of this is that the agent should proactively seek training examples for the predictions it is most unsure about as measured by some confidence factor. In practice, this would mean that the TV recommender would occasionally directly ask the user about some particular program. This approach would relieve the user of some of the responsibility of giving good training examples to the agent.

7. ACKNOWLEDGMENT

This research was supported by the Smart Internet Technology Cooperative Research Centre.

8. REFERENCES

[1] L. Ardissono and M. Maybury, editors. Special Issue on User Modelling and Personalization for Television. *User Modelling and User-adapted Interaction*, 14(1), 2004.

[2] A. Bowers, C. Giraud-Carrier, and J. Lloyd. Classification of individuals with complex structure. In P. Langley, editor, *Machine Learning: Proceedings of the Seventeenth International Conference (ICML2000)*, pages 81–88. Morgan Kaufmann, 2000.

[3] S. Džeroski, L. De Raedt, and H. Blockeel. Relational reinforcement learning. In *Proceedings of the 15th International Conference on Machine Learning, ICML'98*, pages 136–143. Morgan Kaufmann, 1998.

[4] S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.

[5] S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer, 2001.

[6] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[7] J. Lloyd. *Logic for Learning*. Cognitive Technologies. Springer, 2003.

[8] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[9] R. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

$$
\begin{aligned}
&top \rightarrowtail projTitle \circ top\\
&top \rightarrowtail projTitle \circ StringToText \circ top\\
&top \rightarrowtail projSubtitle \circ top\\
&top \rightarrowtail projSubtitle \circ StringToText \circ top\\
&top \rightarrowtail projGenre \circ (listExists_1\ top)\\
&top \rightarrowtail projGenre \circ (listExists_1\ genre \circ top)\\
&top \rightarrowtail projClassification \circ top\\
&top \rightarrowtail projClassification \circ classification \circ top\\
&top \rightarrowtail projSynopsis \circ StringToText \circ top\\
\\
&top \rightarrowtail (=\ \text{“}The\ Bill\text{”})\\
&top \rightarrowtail (=\ \text{“}South\ Park\text{”})\\
&\qquad\vdots\\
&top \rightarrowtail (=\ \text{“}Seinfeld\text{”})\\
&top \rightarrowtail (=\ \text{“}The\ Cosby\ Show\text{”})\\
\\
&top \rightarrowtail (listExists_1\ (=\ \text{“}adventur\text{”}))\\
&top \rightarrowtail (listExists_1\ (=\ \text{“}coverag\text{”}))\\
&\qquad\vdots\\
&top \rightarrowtail (listExists_1\ (=\ \text{“}technolog\text{”}))\\
&top \rightarrowtail (listExists_1\ (=\ \text{“}war\text{”}))\\
\\
&top \rightarrowtail (=-2)\\
&\qquad\vdots\\
&top \rightarrowtail (=2)\\
\\
&top \rightarrowtail (=\ Action)\\
&\qquad\vdots\\
&top \rightarrowtail (=\ Wrestling)\\
\\
&top \rightarrowtail (=\ Y7)\\
&\qquad\vdots\\
&top \rightarrowtail (=\ NA)
\end{aligned}
$$

Figure 3: The predicate rewrite system for the TV recommender

APPENDIX

$$
\begin{aligned}
&(user\_likes\_tv\_program\ x) = \\
&\quad if\ (projTitle \circ (=\ \text{“}NFL\ Football\text{”})\ x)\ then\ \top \\
&\quad else\ if\ (projTitle \circ (=\ \text{“}English\ Premier\ League\text{”})\ x)\ then\ \top \\
&\quad else\ if\ (projGenre \circ (listExists_1\ genre \circ (<0))\ x)\ then\ \bot \\
&\quad else\ if\ (projGenre \circ (listExists_1\ (=Drama))\ x)\ then\ \bot \\
&\quad else\ if\ (projGenre \circ (listExists_1\ (=Comedy))\ x)\ then\ \bot \\
&\quad else\ if\ (projTitle \circ (=\ \text{“}Sky\ RaceNight\text{”})\ x)\ then\ \bot \\
&\quad else\ if\ (projTitle \circ StringToText \circ (listExists_1\ (=\text{“}sport\text{”}))\ x)\ then\ \top \\
&\quad else\ if\ (projSynopsis \circ StringToText \circ (listExists_1\ (=\text{“}war\text{”}))\ x)\ then\ \top \\
&\quad else\ if\ (projGenre \circ (listExists_1\ (=Current\_Affairs))\ x)\ then\ \bot \\
&\qquad\quad \vdots \\
&\quad else\ \bot.
\end{aligned}
$$

Figure 4: A typical definition for $user\_likes\_tv\_program$

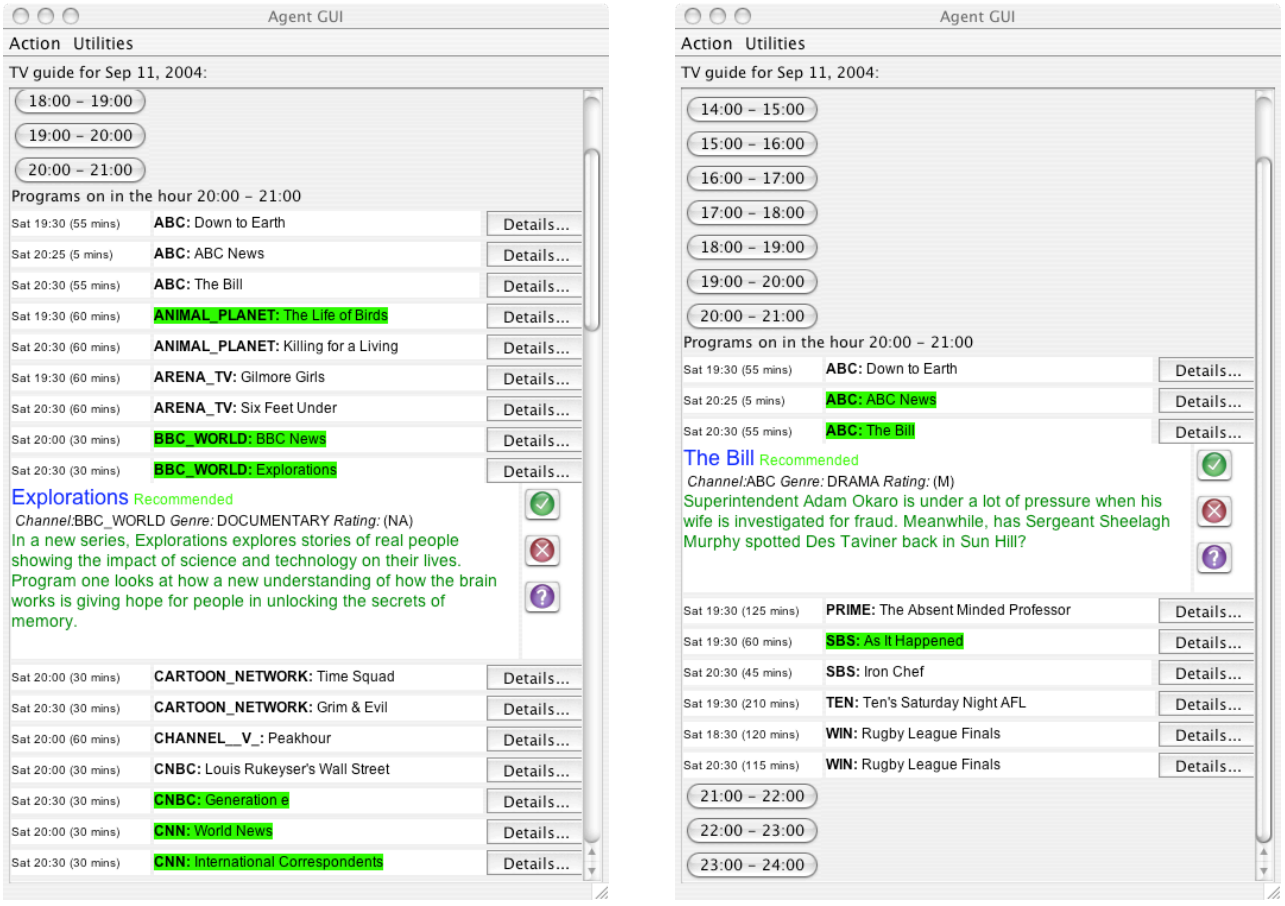

Figure 5: Recommendations presented to two different users for the same time slot.

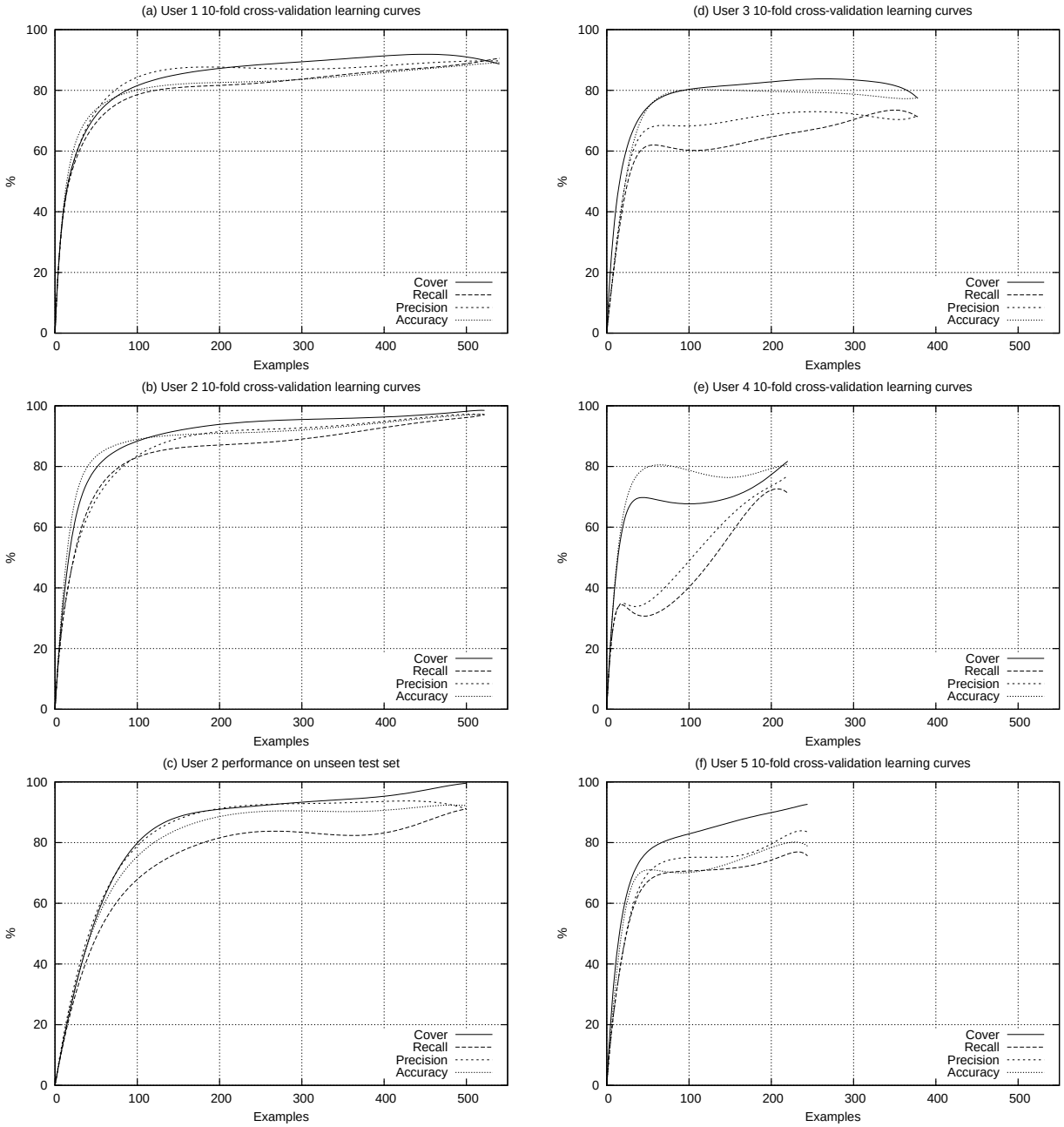

Figure 6: Charts of results

User	Ex	Ex+	Ex−	DL Cov	DL Rec	Boost Rec	DL Prec	Boost Prec	DL Acc	Boost Acc
User 1	540	283	257	88.67	90.57	80.24	89.69	91.16	89.13	90.12
User 2	520	232	288	98.46	97.15	92.67	97.02	94.64	97.27	94.23
User 3	377	146	231	77.43	71.37	54.98	71.44	79.09	77.40	74.79
User 4	220	68	152	81.75	71.21	63.48	77.05	81.97	80.79	84.55
User 5	248	126	122	92.60	75.63	82.50	83.52	84.63	78.85	83.03

Figure 7: Table of results