Learning rules from user behaviour

Domenico Corapi[1], Oliver Ray[2], Alessandra M. Russo[1], Arosha K. Bandara[3], and Emil C. Lupu[1]

[1] Imperial College London, Department of Computing
{d.corapi, a.russo, e.c.lupu}@imperial.ac.uk
[2] University of Bristol, Department of Computer Science
oray@cs.bris.ac.uk
[3] The Open University, Department of Mathematics and Computing
a.k.bandara@open.ac.uk

Abstract. Pervasive computing requires infrastructures that adapt to changes in user behaviour while minimising user interactions. Policy-based approaches provide adaptability but, at present, require policy rules to be provided by users. This paper presents preliminary work on using Inductive Logic Programming (ILP) to automatically acquire such knowledge from observational data. We show how a non-monotonic ILP system called XHAIL can incrementally learn and revise rules of user behaviour and we briefly discuss how this approach might be exploited within a wider pervasive computing framework.

1 Introduction

Pervasive computing is being enabled by the development of increasingly complex devices and software infrastructures that accompany users in everyday life. Such systems must autonomously adapt to changes in user context and behaviour, while operating seamlessly with minimal user intervention. They must, therefore, be able to learn from sensory input and user actions. Yet user acceptance requires them to be predictable, capable of explaining their actions, and providing some way for users to understand and amend what has been learnt.

This directs us towards techniques that use logical rules for knowledge representation and reasoning. Even though some statistical pre-processing of raw sensor data will inevitably be required, there are considerable advantages to adopting a core logical formalism; such as simple and modular enforcement through policy frameworks [1, 2] and principled representations of space and time [3]. Logic programs are an ideal choice of knowledge representation from a computational point of view; and they are also supported by powerful tools, developed in the field of Inductive Logic Programming (ILP) [4] that permit the learning of logic programs from examples.

Learning rules of user behaviour through inductive reasoning poses several challenges. First, learning must be incremental: as examples of user behaviour are continuously added, the system must permit periodic revision of the rules and knowledge learnt. Second, the system must cater for temporal aspects, expressing both persistence and change through time, and exceptions to previously learnt rules. For this, the system must be capable of non-monotonic reasoning [5]. Third, the system must reason with partial information whilst providing fine grained control of the resoning process to satisfy user defined language and search biases (such as minimising the changes made to the initial theory).

This paper reports preliminary work using a nonmonotonic ILP system called XHAIL to learn and revise models of user behaviour. To illustrate the approach we consider a simplified example consisting of learning the circumstances in which users accept, reject or ignore calls on a mobile phone. Such rules could then be enacted automatically avoiding user intervention, and could be periodically reviewed and amended by the user. Although we focus here on the learning process, this work is part of a larger project [6] that also seeks to exploit this knowledge in conjunction with suitable privacy policies in real mobile devices.

The paper is structured as follows: in Section 2 we describe the ILP concepts necessary for the understanding of the rest of the paper; Section 3 discusses learning in pervasive systems, introduces the basic concepts in our framework, describes the theory revision aspects of XHAIL and presents our example. Section 4 compares our approach with other ILP systems. Section 5 summarises our conclusions and describes directions for future work.

2 Background

Inductive Logic Programming (ILP) [4] is concerned with learning logic programs from *examples* consisting of factual data that partially describes concepts to be learned. In our case, examples describe instances of user actions in time. From examples we aim to derive more general rules that can classify unobserved examples and therefore predict user actions. More precisely, given a background theory B that specifies given knowledge and a set of examples E, we attempt to find a hypothesis H such that $B \cup H \models E$ (i.e., the examples are logical consequences of the derived hypotheses together with the background theory).

Formally, a *(normal) logic program* [7] is a set of *clauses* of the form:

$$a_0 \leftarrow l_1 \wedge l_2 \wedge ... \wedge l_n$$

where a_0 is an *atom* and each *literal* l_i is either an atom a_i or a negated atom $\neg a_i$. a_0 is the head of the clause and $l_1 \wedge l_2 \wedge ... \wedge l_n$ is its body. The meaning of a clause is that if all the literals in the body are true, then the head must be true. An atom $p(t_1, ..., t_n)$ is the application of an n-ary predicate p to n terms that represent entities in the domain. Any variables (which, by convention, begin with capital letters) are assumed to be universally quantified over all objects in the domain. Clauses and literals are *ground* if they do not contain variables. For example, the following clause is not ground because it contains the variable T:

$$do(select\_mode(silent), T) \leftarrow after(T, 09{:}00) \wedge \neg holdsAt(device\_is\_off, T)$$

This clause predicts that "*the user will select the silent mode at time T, if T is after* 09:00 *and the device is not off.*" *Functions* like *select_mode* are convenient because they allow a more immediate representation of data structures, permitting the encoding of, for example, recursive data types.

The ability to use negative literals in clause bodies gives logic programs a powerful non-monotonic inference mechanism which is useful for practical applications, such as ours, that involve reasoning under uncertainty. As in real-life, non-monotonic reasoning allows previous conclusions (e.g., a bird can fly) to be retracted on the basis of additional information (e.g., that bird is a penguin). In what follows, we will show how non-monotonic ILP can also be used to perform the general theory revision task of finding a new theory B' (which is not necessarily an extension of the original theory B) to correctly account for newly acquired examples E. In particular, we show how a nonmonotonic ILP system called XHAIL (eXtended Hybrid Abductive Inductive Learning) [8] can learn and revise user theories from examples of their behaviour. XHAIL uses abductive reasoning to construct a set of initial ground hypotheses that are inductively generalised afterwards.

3 Learning User Behaviour

Our use of mobile devices and our actions as sensed by the pervasive computing environment implicitly provide precious information that applications could learn in order to operate autonomously or to improve usability and user acceptance. To achieve this, we need to represent knowledge about user behaviour and a learning system capable of continuously revising this knowledge.

The system we propose learns and revises a *user theory*, U, accessible by other applications or by policy management systems. U is a normal logic program defining the conditions in which a user action is performed. Queries can thus be performed on U to determine whether an action needs to be performed in response to events or requests. For example, U can be queried to determine if the user would allow access to his current location in response to a request:

$$?- do(allow(current\_location, request\_id), time).$$

The answer to this query will be based on (a) properties of the request such as the ID of the requester, the time of the request, proofs of identity, etc., (b) contextual information at the particular *time* such as the location of the user, the profile active on the phone or the number of devices nearby and (c) other application-specific knowledge defined as logical predicates.

To represent dynamic systems, we use the Event Calculus [3]. This temproal reasoning framework allows us to infer which properties (or *fluents*) are true at any given time (denoted $holdsAt(F, T)$) based on which actions (or *events*) have previously occurred (denoted $happens(E, T)$), along with a knowledge of how fluents are affected by events.

Conceptually, we consider the hypotheses derived by XHAIL as prescriptions for changes in the current theory to entail examples, rather than as additional hypotheses. This permits revising the current theory by adding or deleting entire rules and/or literals in the body of existing rules.

This result is obtained in three *phases*, as shown in Figure 1. The inputs to each step i are: a logic program B_i containing sensory data, available information on actions and static concept definitions; the set of current examples E_i representing user actions; and the current U_i comprising previously learnt rules.

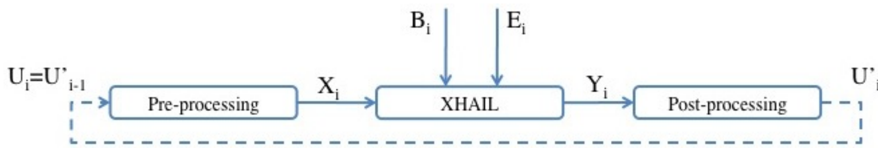

Fig. 1. Revision phases with inputs and outputs at the i-th revision step

B_i is updated when new information is available and the most recent version is used in the learning step. To cater for drifting concepts [9, 10], examples are buffered by means of a sliding window, so that only the most recent ones are used in each learning step. We currently use a simple fixed window size, but we plan to revisit this issue in future work.

A *revision step* starts with a *pre-processing* phase (explained later) that restructures U_i into X_i. During the subsequent phase, XHAIL is executed to find hypotheses Y_i. Hypotheses are then used as new rules and to revise existing rules in a *post-processing phase* that generates a revised theory U'_i.

We illustrate this method through a simple scenario where we aim to learn rules that define *the context in which a user accepts incoming calls.* This scenario has been derived from real data on mobile phone usage collected in the course of the Cityware project [11]. We are showing examples over three days and, for simplicity, the learning computation is performed at the end of each day considering only new examples. Figure 2 illustrates the subset of the data provided to the learning system that appears in rules after the learning is performed.

During the first day (step 0) the user does not answer a call from a contact in her $personal$ contact list while she is at home at 07:00 ($\neg do(accept\_call(c1\_1, alice)$, 07:00)) but subsequent calls are accepted at 07:30 and 11:00. B_0 contains, for example, information about user movements and nearby bluetooth devices together with properties about the calls like $in\_group(alice, home)$.

U_0 is the initial theory and it is empty thus no pre or post-processing is needed. XHAIL learns the following rule (see [8] for a description of the learning algorithm):

$$
\begin{aligned}
&U'_0 = U_1 = \{ \\
&\quad do(accept\_call(CallId, From), T) \leftarrow T \geq 07{:}30. \quad \}
\end{aligned}
\tag{1}
$$

On the second day the user rejects calls from contacts in her *home (H)* and *friends (F)* groups while at Imperial College, but accepts calls from colleagues in the *college (C)* group — which violates the existing rule. A new learning step

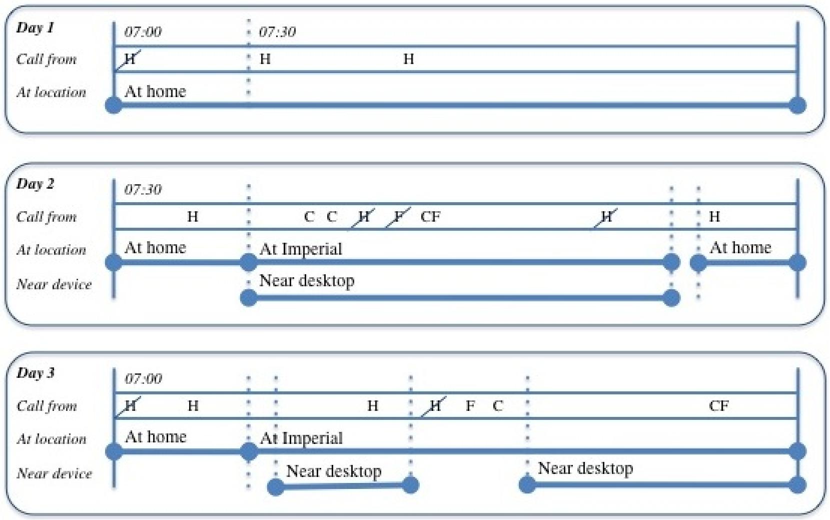

Fig. 2. Example scenario (C, H and F denote incoming calls from the user's college, home, and friends contact lists respectively. Refused calls are marked with a " ⁄ ")

is executed where rule (1) is rewritten during pre-processing as:

$$
\begin{aligned}
X_1 = \{ \;\; & \ldots \\
& do(accept\_call(CallId, From), T) \leftarrow \quad try(1, 1, T \geq 07{:}30) \wedge \\
& \quad \neg exception(1, do(accept\_call(CallId, From), T)). \quad \}
\end{aligned} \tag{2}
$$

where $try/3$ and $exception/3$ are special predicates (detailed later) introduced in every revisable clause. Learning exceptions to a rule is equivalent to adding new conditions to that rule. If the system learns an exception with an empty body then this is equivalent to deleting the rule.

XHAIL is executed with $B_i \cup X_i$ as background theory to learn changes needed to entail examples E_i processed in the current step. The output of the *XHAIL phase* for the second day is:

$$
\begin{aligned}
Y_1 = \{ & \\
& exception(1, do(accept\_call(CallId, From), T)) \leftarrow \quad \neg in\_group(From, college) \wedge \\
& \quad holdsAt(status(location(imperial)), T). \quad \}
\end{aligned}
$$

This is a correct solution because $X_1 \cup Y_1 \models E_1$. According to the result obtained, the *post-processing phase* rewrites rules (2) into:

$$U'_1 = U_2 = \{$$

$$do(accept\_call(CallId, From), T) \leftarrow T \geq 07{:}30 \wedge in\_group(From, college). \tag{3a}$$

$$do(accept\_call(CallId, From), T) \leftarrow T \geq 07{:}30 \wedge \neg holdsAt(status(location(imperial)), T). \tag{3b}$$

$$\}$$

Note that the choice between learning a new rule or revising an existing one, when both solutions are acceptable, is driven by minimality. Generally, every revision step minimally changes U_i i.e., every learning step adds/deletes the minimum number of conditions to/from existing rules, and creates new rules with the minimum number of literals. This is reasonable since, at each step, we would like to preserve the knowledge learnt from previous examples.

Day 3 examples trigger a new step to revise rules since two calls from contacts not included in the *College* contact list are answered while at Imperial College and no rules can explain this. Rule (3b) is pre-processed as follows (and rule 3a is treated analogously):

$$\begin{aligned}
&X_2 = \{ \ \ldots \\
&\quad do(accept_call(CallId, From), T) \leftarrow \\
&\qquad try(2, 1, T \geq 07{:}30) \wedge \\
&\qquad try(2, 2, \neg holdsAt(status(location(imperial)), T)) \wedge \\
&\qquad \neg exception(2, do(accept_call(CallId, From), T)). \quad \}
\end{aligned} \tag{4}$$

During pre-processing *try* is added to every literal in X_i. Every literal is uniquely identified by the first two arguments of $try(c, r)$, where c refers to the clause and r indexes the literal in the body of the clause. For instance, the clause added in X_2 for the first condition in the rule (4) is:

$$\begin{aligned}
&X_2 = \{ \ \ldots \\
&\quad try(2, 1, T \geq 07{:}30)) \leftarrow use(2, 1) \wedge T \geq 07{:}30. \quad \}
\end{aligned}$$

As explained in more detail in [8], using $try(c, r, condition)$ instead of *condition* in the rule weakens the condition so that if U_i is not consistent with examples, XHAIL can learn that certain literals must be deleted. $use/2$ is defined by the rule $use(I, J) \leftarrow \neg del(I, J))$ meaning that a literal is kept in U_i if not deleted. This indirection is needed since XHAIL computes minimal revisions (a minimal number of *del* facts should be learnt in order to delete the lowest possible number of literals). $del(c, l) \in Y_i$ means that in the consistent revision computed by XHAIL the literal indexed by c and l can be deleted. In the scenario, the hypotheses computed by XHAIL for the third day are

$$\begin{aligned}
&Y_2 = \{ \\
&\quad exception(2, do(accept_call(CallId, From), T)) \leftarrow \\
&\qquad holdsAt(status(bluetooth_near(desktop_computer)), T). \\
&\quad del(2, 1). \quad \}
\end{aligned}$$

Following from the definition of *try*, this means that the examples can be explained by deleting the literal $(2, 1)$. After theory refactoring in the last phase, this results in the following rule:

$$\begin{aligned}
&U_3 = U'_2 = \{ \\
&\quad do(accept_call(CallId, From), T) \leftarrow T \geq 07{:}30 \wedge in_group(From, college). && \text{(5a)} \\
&\quad do(accept_call(From), T) \leftarrow T \geq 07{:}30 \wedge \\
&\qquad \neg holdsAt(status(bluetooth_near(desktop_computer)), T). \quad \} && \text{(5b)}
\end{aligned}$$

U_3 can be queried, for example, to know if the answer-phone should be activated or the ring turned off for future incoming calls. The answer will be based on the context at the time of the call.

We have shown how a set of two rules describing the conditions under which the user answers to phone calls, U_3, has been learnt incrementally in three steps from scratch, based on information about user location, co-location with other devices and properties of the incoming calls, namely the contact list associated with the caller and the time of the call.

Each revision computed by XHAIL took a couple of seconds on a Pentium laptop PC. The solutions shown above were selected by hand from half a dozen or so alternative minimal hypotheses returned by XHAIL.

4 Discussion

Our preliminary results suggest that this approach to theory revision based on the integration of non-monotonic abductive and inductive reasoning can be exploited to learn rules describing user behaviour.

Although statistical techniques [12] are necessary to process, classify and aggregate raw sensor data upstream, a core logical methodology provides significant advantages as a subsequent step: descriptive rules constitute executable policies that users can query, understand, and amend; such policies are modular and expressed using user-level abstractions; and logic allows principled representations of space, time and causality.

XHAIL has several advantages compared with other ILP systems. Whilst Progol5 [13] and Alecto [14], also employ abduction to learn non-observed predicates, they do not have a well-defined semantics for non-monotonic programs and their handling of negation is limited. Compared to first-order theory revision systems like *INTHELEX* [15], *Audrey II* [16] and *FORTE* [17], the proposed extension of XHAIL has a more expressive language, more efficient inconsistency detection and exploits existing domain specific information to accurately constrain possible revisions. Rules like (3b) and (5b) cannot be learnt by FORTE because it can only learn Horn clauses that do not allow negated literals in the body. Furthermore the three theory revision systems constrain hypotheses to function-free clauses and this implies a less compact representation and complications in the axiomatization and use of Event Calculus.

5 Conclusions and future work

Although other theory revision systems exist, none has succeeded in expressing contextual conditions in dynamic systems and constructing compact rules using negated conditions. We have shown that by extending a non-monotonic ILP learning system it is possible to learn incrementally and revise rules describing user behaviour. Rules are learnt based on past examples, which consist of positive and negative conditions under which the user performs actions.

This work is part of a more ambitious effort to learn privacy policies on mobile devices. Learning rules is thus part of a framework that also includes: policy enforcement [1], statistical learning and classification. Further work, includes the development of components for efficient access to sensory data, caching and acceleration of the learning process, windowing techniques for the examples, handling of noisy data as well as more studies on the theory revision aspects.

We are mindful of the complexity of the implementations of such algorithms but our experience with policy enforcement, and distributed abductive reasoning on mobile devices [18] has provided valuable lessons on techniques for improving the scale-down and efficiency of the implementations.

Acknowledgements

This work is funded by the UK EPSRC (EP/F023294/1) and supported by IBM Research as part of their Open Collaborative Research (OCR) initiative and Research Councils UK.

References

1. Ponder2: The ponder2 policy environment. www.ponder2.net
2. Lupu, E., et al.: AMUSe: Autonomic Management of Ubiquitous Systems for e-health. J. Conc. and Comp.: Practice and Experience **20**(3) (2008) 277–295
3. Shanahan, M.: The event calculus explained. LNCS **1600** (1999)
4. Muggleton, S., de Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming **19/20** (1994) 629–679
5. Minker, J.: An overview of nonmonotonic reasoning and logic programming. Technical Report UMIACS-TR-91-112, CS-TR-2736, University of Maryland, College Park, Maryland 20742 (August 1991)
6. Bandara, A.K., Nuseibeh, B.A., Price, B.A., Rogers, Y., Dulay, N., et al.: Privacy rights management for mobile applications. In: 4th Int. Symposium on Usable Privacy and Security, Pittsburgh (July 2008)
7. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
8. Ray, O.: Nonmonotonic abductive inductive learning. In: Proc. Int. Workshop on Abduction and Induction in Artificial Intelligence and Bioinformatics. To appear in Journal of Applied Logic. (2008)
9. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T.M.A., Mauro, N.D.: Incremental learning and concept drift in inthelex. Intell. Data Anal. **8**(3) (2004) 213–237
10. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**(1) (1996) 69–101
11. Cityware: Urban design and pervasive systems. http://www.cityware.org.uk/
12. Eagle, N., Pentland, A.: Reality mining: sensing complex social systems. Personal and Ubiquitous Computing **10**(4) (2006) 255–268
13. Muggleton, S.: Learning from positive data. In: 6th Int. Workshop on Inductive Logic Programming, London, U.K., Springer Verlag (1996) 358–376
14. Moyle, S.: An investigation into theory completion techniques in inductive logic. PhD thesis, University of Oxford (2003)
15. Esposito, F., Semeraro, G., Fanizzi, N., Ferilli, S.: Multistrategy theory revision: Induction and Abduction in INTHELEX. Mach. Learn. **38**(1-2) (2000) 133–156
16. Wogulis, J., Pazzani, M.J.: A methodology for evaluating theory revision systems: Results with Audrey II. In: 13th IJCAI, Chambery (1993) 1128–1134
17. Richards, B.L., Mooney, R.J.: Automated refinement of first-order horn-clause domain theories. Machine Learning **19**(2) (1995) 95–131
18. Ma, J., Russo, A., Broda, K., Clark, K.: DARE: a system for Distributed Abductive REasoning. J. Autonomous Agents and Multi-Agent Systems **16** (2008) 271–297