Activity recognition in multiple contexts for smart-house data

Kacper Sokol and Peter Flach

Intelligent Systems Laboratory, University of Bristol, UK,
{k.sokol, Peter.Flach}@bristol.ac.uk

Abstract. Predicting human behaviour from smart house data became a recent topic of interest. Applications include but are not limited to healthcare — detecting falls, and improving quality of life e.g. smart thermostats. This kind of technology is becoming increasingly popular and in most of the cases making a house smarter is as simple as installing off-the-shelf devices. In this paper we focus on predicting future location of a person in a simulated smart house environment. We consider three different occupant types who have varying working patterns: *normal*, *part-time*, and *shift*. We develop a *versatile model* which is capable of adjusting to significant change in occupant behaviour without the need of retraining. We model the problem as a simple event calculus task. To this end, we build a framework based on the Aleph Inductive Logic Programming system. Event calculus helps to handle time and persisting sensor states. Background knowledge allows to encode important information about the smart house that is otherwise difficult to learn and facilitates transferability of the model to different house layouts. Finally, rule models being white-box (human readable) are easy to inspect and tune, providing useful insights during the feature design and training phases.

Keywords: activity recognition, smart-house, versatile model, event calculus, Aleph

1 Introduction

We consider a problem of controlling a heating system in a living room based on behaviour (location) of a resident. We want to predict the location of a resident half-an-hour in advance — the time needed to preheat the room to a selected temperature — to turn on the heating system only when necessary, hence, improve the energy footprint of the house. The majority of off-the-shelf thermostats have daily resolution with only two working modes available: *working day* and *free day*; where each setting allows two different temperature settings: *leaving* and *returning* to the house. Some companies like *Nest* have built thermostats that "learn" daily routines from repetitive temperature changes made manually. Unfortunately, such systems perform poorly on households with unstructured and irregular working patterns which are increasingly common these days. All these create a need for truly intelligent thermostat [3, 5].

In most of the cases smart house data is in form of a time series consisting of sensor activations. The events are usually not evenly spaced through time as they are caused by a change in sensor state. This property of time in our data causes representational difficulties well known in logic. To overcome this issue we use *Simple Event Calculus* [7].

Simple Event Calculus is a formalisation of representing actions and their effects in logic. This framework allows for *events* (sensor state changes) occurring in a single time point that activate or terminate *fluents* — properties of system that persist through time. Additionally, event calculus provides an approach to evaluate the model over time and not events what guarantees best possible model for a set of events [8].

In this paper we consider a simple scenario of a house with 2 rooms (`living_room` and `bedroom`: one motion sensor per room) and days discretised into 12 distinctive blocks. We chose this approach as modelling a smart two-bedroom flat for a week for the three different resident types has proven to be infeasible due to overwhelming amount of data for the Aleph learning system. All three toy datasets are presented in Tables 2, 3, and 4 shown in Appendix. In the toy example we focus on predicting not being in a selected room — `living_room` — during the next time block. To this end, we build a *vanilla* and *versatile* rule-based models and show that the latter achieves top-ranked performance regardless of context of operation (working pattern) [2]. We use *Aleph* Inductive Logic Programming framework to build our models [4].

2 Variable Context Time Sequence Completion

We assume that a smart house fitted with n motion sensors at each time point t provides a feed of boolean (activated/deactivated) data in a form of a binary vector $\mathbf{x}_t = (x_{t1}, \ldots, x_{tn})$ where $x_{ti} \in \{0,1\}$ is a state of a sensor i at time t. These observable events $\mathbf{x}_t$ for $t \in [1,T]$ create a sequence $[\mathbf{x}_1, \ldots, \mathbf{x}_t] \in \mathscr{X}$ — a smart house state over time.

Furthermore, this sequence can be divided into a series of consecutive *non-observable* blocks $B = [B_{1m}, \ldots, B_{zT}]$ for $1 < m < z < T$ by a series of time points $[m, \ldots, z]$; and a block B_{ab} is defined as $B_{ab} = [\mathbf{x}_a, \ldots, \mathbf{x}_b]$ where $1 < a < b < T$. Each block can be characterised by some unique label $L_{ab} = L(B_{ab})$ from a finite set of labels $\mathscr{L}$. In the toy example $\mathscr{L} = \{\texttt{sleep}, \texttt{work}, \texttt{leisure}\}$.

Additionally, a sub-sequence of events $[\mathbf{x}_r, \ldots, \mathbf{x}_s]$ for some $1 \leq r < s \leq T$ occurs in an implicit, *non-observable context* $C_{rs} = C([\mathbf{x}_r, \ldots, \mathbf{x}_s])$. We assume that every complete series $[\mathbf{x}_1, \ldots, \mathbf{x}_T]$ is a mixture of several contexts. In our example possible contexts are $\mathscr{C} = \{\texttt{working\_day}, \texttt{working\_night}, \texttt{working\_morning}, \texttt{working\_afternoon}, \texttt{free\_day}\}$.

Due to the above property of our data, a naïvely fitted model would implicitly depend on the sequence of contexts. Such a model is therefore ineffective if in deployment the context is not as expected. To remedy this, we propose to use a *versatile* model that can adapt to constantly changing contexts and ensure accurate predictions throughout [1].

Example 1. An example rule from a versatile model:

```
not_visiting_room_in_the_next_time_block(living_room,A) :-
    holdsAt(not_in_room(living_room),A),
    holdsAt(time_block(10),A),
    contex(free_day,A).
```

where not being in the living room in the next time block is defined as: not being in the living room currently and current time block (hour of the day) being 10 given that current context is a free day. In this case the context is defined as an additional feature.

Finally, we define the binary target variable to be $y_t \in \{0,1\}$ — where $[y_1,\dots,y_T] \in \mathscr{Y}$; and y_t indicates not being in a selected room (`living_room`) in the next $t+\delta$ time block for $\delta > 0$. In the toy example we fix $\delta = 1$. With the input space $\mathscr{X}$ as defined above our data are represented as $\mathbf{D} := \mathscr{X} \times \mathscr{Y}$ and we want to learn a function $f : \mathscr{X} \to \mathscr{Y}$ describing these data.

In the smart house scenario presented above the model output $y_{t+1} = f([\mathbf{x}_1,\dots,\mathbf{x}_t]) =$ `versatile_model`$(C_{tt}, [\mathbf{x}_1,\dots,\mathbf{x}_t])$ is necessary. Therefore two predictions have to be made: the current context C_{tt} dependent on the data block structure is predicted; then it is used together with the partial observations $[\mathbf{x}_1,\dots,\mathbf{x}_t]$ to predict the label y_{t+1}. It is worth noting that the context is at the top of the hierarchy and is latent while the block structure is somehow in the middle of the time series structure.

For the problem defined above we want to show that a vanilla model performs quite well on the data where the implicit context does not change. Nevertheless, once it is changed the model's performance drops significantly while a versatile model performance does not exhibit similar behaviour.

Finally, variable context time sequence completion problem is not limited to smart environments: houses and cities. It can be used with any kind of spatio-temporal data with hierarchical structure and varying context.

3 The toy dataset

Working on real and artificial smart house data has proven to be infeasible due to large amount of data points — ~60k samples per week for a 2-bedroom flat — leading to too long computation time. Therefore, we decided to use a simple representation of the problem which exhibits all the necessary properties. For this study we use artificial data generated by a highly customisable smart house data generator [9]. We create a week of data with 12 samples a day for a house with 2 rooms, each fitted with 1 motion sensor. The data are generated for each of the three working patterns — see Tables 2, 3 and 4 in Appendix for reference.

3.1 Data representation

Spatio-temporal data generated by a smart house pose representation issues due to time variable being unbounded and sensor events occurring in a single time point but causing a change persisting through time. These properties lead naturally to use of *Simple Event Calculus* which addresses all of these issues [6]. Therefore, the data have to be formatted in event calculus syntax as well as being preprocessed for the Aleph framework requiring: positive examples, negative examples, and background knowledge (Aleph configuration, raw data, feature extractors, etc.) files. Sensor activations are represented as

```
happensAt(sensor(m06, on), 13).
```

fluents describing properties of the smart house (sensors state, signal features) as

```
holdsAt(sensor(m01, on), 13).
```

and annotations are in a form

```
not_visiting_room_in_the_next_time_block(living_room,14).
```

3.2 Contexts

We use five different contexts listed above (`working_day`, `working_night`, `working_morning`, `working_afternoon`, `free_day`) and indicating different type of a day, mixture of which creates one of the three different working patterns: *normal* (full-time), *part-time*, and *shifts*. The general structure of each working pattern is given in Tables 2, 3 and 4 and described below.

Normal A typical working day consists of basic morning activities, 8 hours of work, followed by leisure time and 8 hours of sleep. A typical free day consists mainly of leisure activities and having a night out.

Part-time The free days are exactly the same as above, but working days are either *working mornings* or *working afternoons* where the work period is reduced to 4 hours and the remaining 4 hours are allocated with leisure activities.

Shifts The free days are again the same as above though they can appear during weekdays. Additionally there are two types of working patterns: *working days* which are exactly the same as in normal working schedule and *working nights* — mirror image of working days.

3.3 Features

Rules extracting features form the raw sensor state data are the most important building block of any rule-based system. Our model is built with real-time application in mind, hence, all of the features can use sensor states from the past but they cannot see future system state. All of the features are implemented as event calculus *fluents*.

Location-based features Majority of used features is location-based: either current location or one of the previous locations.

- Being in a given room,
- not being in a given room,
- visiting a room in a sliding time window of fixed length,
- sequence of visited rooms.

Time-based features Time-based features consist of all the information that can be extracted from a UNIX timestamp.

Date: year, month, day;
time: hour, minute, second;
timestamp derivatives: season of the year, day type, week number, week day, time of the day (morning, afternoon, etc.).

Finally, we implemented some additional features neither related to time nor place like *type of day* encoding contextual information — one of 5 possible day types.

4 Results

The main goal of this study is to show the importance of versatile models in complex scenarios like smart house activity recognition. Non-versatile models use implicit information about the house layout, number of residents and lifestyle patterns. A model performing well for one person's house is not guaranteed to have similar performance on a different house. A versatile model can handle multiple contexts without need of time and resource consuming retraining — features in such models need to generalise well by using *markers* specific to given routines rather than fixed time points.

A great advantage of ILP rule-based systems is possibility to tweak the model by changing the background knowledge. If a person moves from one house to another and does not change living patterns, once learnt model can be used in the new environment by simply changing sensors bindings in the background knowledge; no need for model retraining. Additionally, the background knowledge can be used to "inform" the model about the learning problem properties which with any other learning system would have to be inferred first.

4.1 Models

Below we present two model types: *vanilla* and *versatile*; we compare and contrast them against *majority class classification*. For comparison we use data for all 3 working patterns as well as *merged* 3-weeks long data where each week comes from a different working pattern. Additionally, we use 3-weeks long data where number of days from every context is the same as in merged data but the order is shuffled — we call it *shuffled*. Results are presented in Table 1. These are accuracies (in %) averaged over 3 different realisations of given working pattern from the same distribution.

Vanilla A non-versatile model (*Merged* rule list) is one that does not use the contextual information hidden in the data. Given our features, the most basic vanilla model learnt by Aleph is highly dependant on the *time* structure of the series. Due to repetitive patterns in the data the model memorises what happened during each *day of the week* at given *time of the day* therefore achieving high performance (92.46% ACC on *Merged*) on data from the same working patter distribution. Unfortunately, this model does not perform similarly well when working patterns are shifted or shuffled e.g. Monday for normal working person is no longer a working day but is a free day (74.21% ACC on *shuffled*). See Table 1 for details.

```
not_visiting_room_in_the_next_time_block(living_room,A) :-
   holdsAt(in_room(bedroom),A),
   holdsAt(time_block(3),A),
   holdsAt(day_number(1),A).
not_visiting_room_in_the_next_time_block(living_room,A) :-
   holdsAt(not_in_room(living_room),A),
   holdsAt(time_block(7),A).
```

	Rules:	Normal	Shift	Part	Merged	Versatile	Majority
Data:	Normal	**89.29**	72.62	76.19	90.48	*96.43*	75.00
	Shift	80.95	**82.14**	67.86	89.29	*90.48*	66.67
	Part	80.95	77.38	**85.71**	*97.62*	88.10	63.10
	Merged	83.73	77.38	76.59	***92.46***	91.67	68.25
	Shuffled	83.73	77.38	75.40	74.21	*84.76*	68.25

Table 1. Accuracies (in %) averaged over 3 realisations of given dataset for *vanilla*, *versatile*, and *majority class* approaches. Rows are data and columns are rule sets. Italic numbers indicate best rule set for given dataset and bold figures indicate test set accuracies.

Versatile A versatile model picks up patterns and parametrises them based on context. Most of the patterns in our data are variations in time of the same event structure: stretched, shifted or mirror imaged. Therefore, such model learns structure of sequences in given context rather than its detailed dependence on the time. The versatile model always outperforms the self-learnt rules for each of the 3 working patterns. Additionally, it performs very close to the *vanilla* model on *Merged* data — 91.67% ACC; and does not suffer significant loss in accuracy for the shuffled data: 84.76% ACC on *shuffled*. This 7% drop in performance is caused by the `context` rule not always correctly recognising the type of the day from the partial data it gets. In order to build this model *multi-tier learning* approach was used. First of all, a definition of `context` predicate has been learnt; then, it has been used as a feature in the main predicate learning. The versatile model presented below uses approach called *context as a feature*.

```
not_visiting_room_in_the_next_time_block(living_room,A) :-
    holdsAt(not_in_room(living_room),A),
    holdsAt(time_block(6),A),
    contex(working_day,A).
not_visiting_room_in_the_next_time_block(living_room,A) :-
    holdsAt(not_in_room(living_room),A),
    holdsAt(time_block(10),A),
    contex(free_day,A).
```

5 Conclusions and future work

The work presented in this paper shows the importance of context — often implicit — that our data is produced in. Moreover, context is not always a monolithic object, it often can be divided into smaller entities. Furthermore, we showed the significance of event calculus framework in logic when handling spatio-temporal data; and that it can be implemented in Aleph by using *multi-tier learning* to achieve a versatile model in particular. Finally, we showed that by recognising context the model does not suffer significant loss in performance when the context of operation changes, hence, no need of time and resource consuming model retraining phase.

This work provides only experimental evaluation on a largely simplified real life problem and acts as a proof of concept. We plan to study outlined problem in more detail with both real-like synthetics and real datasets and provide evaluation of both vanilla and versatile models on these large scale datasets.

References

1. Al-Otaibi, R., Prudêncio, R.B., Kull, M., Flach, P.: Versatile decision trees for learning over multiple contexts. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 184–199. Springer (2015)
2. Baryannis, G., Woznowski, P., Antoniou, G.: Rule-based real-time adl recognition in a smart home environment
3. Diethe, T., Twomey, N., Flach, P.: Active transfer learning for activity recognition. In: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (2016)
4. Hernández-Orallo, J., MartÍnez-UsÓMartinez-Uso, A., Prudêncio, R.B., Kull, M., Flach, P.: Reframing in context: A systematic approach for model reuse in machine learning. In: AI Communications (to appear) (2016)
5. Kafalı, Ö., Romero, A.E., Stathis, K.: Activity recognition for an agent-oriented personal health system. In: International Conference on Principles and Practice of Multi-Agent Systems. pp. 254–269. Springer (2014)
6. Katzouris, N., Artikis, A., Paliouras, G.: Incremental learning of event definitions with inductive logic programming. Machine Learning 100(2-3), 555–585 (2015)
7. Kowalski, R., Sergot, M.: A logic-based calculus of events. In: Foundations of knowledge base management, pp. 23–55. Springer (1989)
8. Shanahan, M.: The event calculus explained. In: Artificial intelligence today, pp. 409–430. Springer (1999)
9. Sokol, K.: Shgen: v1.0 (Dec 2015), http://dx.doi.org/10.5281/zenodo.34710

Appendix: Toy dataset realisation

	t	Mon–Fri	Sat	Sun
	$C(\mathbf{x}_t)$	`w. day`	`free day`	`free day`
$\mathbf{x}_t$	`living_room` $= x_{t1}$	`000010000110`	`000010100000`	`000000110110`
	`bedroom` $= x_{t2}$	`111100000001`	`111100011000`	`011111001001`
	$L(B_t)$	`SSSSLWWWWLLS` (x5)	`SSSSLLLLLLLL`	`LSSSSSLLLLLS`
	y_t	`111011110011`	`111010111111`	`111110010011`

Table 2. Toy example (and dataset) for normal living pattern.

	t	Mon–Tue	Wed–Fri	Sat	Sun
	$C(\mathbf{x}_t)$	`w. morning`	`w. afternoon`	`free day`	`free day`
$\mathbf{x}_t$	`living_room` $= x_{t1}$	`000010011110`	`000011100110`	`000010100000`	`000000110110`
	`bedroom` $= x_{t2}$	`111100000001`	`111100000001`	`111100011000`	`011111001001`
	$L(B_t)$	`SSSSLWWLLLLS` (x2)	`SSSSLLLWWLLS` (x3)	`SSSSLLLLLLLL`	`LSSSSSLLLLLS`
	y_t	`111011000011`	`111000110011`	`111010111111`	`111110010011`

Table 3. Toy example (and dataset) for part-time living pattern.

	t	Mon–Tue	Wed	Thur–Fri	Sat	Sun
	$C(\mathbf{x}_t)$	`w. day`	`free day`	`w. night`	`w. night`	`free day`
$\mathbf{x}_t$	`living_room` $= x_{t1}$	`000010000110`	`000010110110`	`000110000010`	`000110011110`	`000001101110`
	`bedroom` $= x_{t2}$	`111100000001`	`111100001000`	`000001111100`	`000001100001`	`111110010001`
	$L(B_t)$	`SSSSLWWWWLLS` (x2)	`SSSSLLLLSLLW`	`WWWLLSSSSSLW` (x2)	`WWWLLSSLLLLS`	`SSSSSLLLLLLS`
	y_t	`111011110011`	`111010010011`	`110011111011`	`110011000011`	`111100100011`

Table 4. Toy example (and dataset) for shifts living pattern.