Journal of Ambient Intelligence and Smart Environments 0 (2013) 1–0
IOS Press

An unsupervised recommender system for smart homes

removed
removed

Abstract.
Inhabitants of today's smarter homes struggle with complicated user interfaces and inflexible home configurations. The proposed smart home recommender system addresses these issues by continuously interpreting the user's current situation and recommending services that would be beneficial for the user in some way, i.e. automate some action that the user would want to perform anyway. With these recommendations it is possible to build much simpler user interfaces that highlight the most interesting choices currently available. Configuration becomes much more flexible, since the recommender system automatically learns user habits. Evaluations on two smart home datasets show that the algorithm produces correct recommendations with 61% and 73% accuracy, respectively.

Keywords: context-awareness, personalization, behavior modeling, machine learning

1. Introduction

The smarter homes of tomorrow promise to increase comfort, aid elderly and disabled people, and help inhabitants save energy. Unfortunately, smart homes today are far from this vision. People who already live in such a home struggle with complicated user interfaces, in particular guests that are unfamiliar with the system are often unable to use it at all [5,17]. Rules (e.g. at 8 AM ring the alarm and open bedroom window blinds) and so-called scenes (e.g. at the press of one button, all lights are turned off) that should reflect user routines must today be manually programmed into the system. However, often user routines are subconscious and can not be described in the detail necessary to cover all variations [9]. Consequently, several inhabitants of smarter homes complain about inconveniences caused by the inflexibilities of rules and scenes [20,5,17,27].

The proposed recommender system for smart homes removes the need to manually describe user routines. In a training phase, the proposed system learns a model of the user habits, i.e. learns which actions the user typically performs in a given situation. The system can learn user routines from manually performed actions as well as from commands given through the smart home user interface (UI). This means that it is not necessary to use a complicated smart home UI for training the system, instead users can continue to live in their home just as normal. The training is unsupervised, i.e. no manual annotation of smart home data is necessary.

After the training phase, the system continuously interprets the user's current situation and generates personalized recommendations. The system tries to recommend services that would be beneficial for the user in some way, i.e. can automate some action that the user would want to perform anyway. In high-level terms, the output of the system are recommendations such as "in the current situation you typically perform actionX, would you like me to perform it for you?".

For example, consider that the user is preparing dinner, opening cupboards and the refrigerator, and using the stove. A large number of services will be fairly useless in this context, such as opening the garage door or closing the skylight. Which service is most useful depends largely on the user's habits. If the user likes to listen to the radio while cooking, then a service for turning on the radio might be relevant. If dinner is commonly eaten in the dining area, the smart home may already turn on the lights above the table. The system may also automate some tasks that are problematic for the user, e.g. opening high-mounted

cabinet doors. And if it starts raining, the previously irrelevant service for closing the skylight could suddenly become important.

The recommendations can be used to simplify user interfaces by removing complicated menus and instead allow the user to select from only the most relevant choices. They might also be used to improve recognition rates of alternative input methods such as speech recognition, or might even allow the smart home to act autonomously in some situations.

The proposed recommender system is evaluated on two publicly available smart home datasets. If the user is shown only the one best recommendation, the system reaches an accuracy of 61% and 73%, respectively. If the user has the choice between five recommendations, the accuracy reaches 90% for both datasets. The results are stable with regard to choice of system parameters, as long as extreme values are avoided.
The main contributions of the paper are:

- The paper proposes an unsupervised method for learning an inhabitant's habits in the smart home.
- The paper proposes a method for interpreting the current user situation and generating service recommendations based on the learned model.
- The paper outlines several strategies for utilizing the recommendations that allow users to decide how much direct control over their home they are willing to give up for an increase in usability and comfort.

Section 2 discusses user requirements, gives an overview of the system and describes different strategies for utilizing the recommendation results. Section 3 presents necessary definitions. In Sections 4 and 5, the method's training and recommendation phases are described. The method is evaluated in Section 6 and compared to related work in Section 7. The paper concludes with a short summary and some ideas for future work.

2. Background

2.1. User expectations and experiences

A number of qualitative and quantitative studies have been performed to find out how people view the home of the future and what expectations they have of living in such a home. It turns out that comfort tasks and home control are the two most interesting applications for many potential users [24,11,4]; e.g. remote control of heating, lights and windows, and help with cleaning tasks are commonly mentioned applications.

Complicated user interfaces Several study participants that already today live in a smarter home remark that increased comfort is indeed one of their favorite aspects [5,17]. However, today's smart home inhabitants struggle with complicated user interfaces. One participant complains that "things must be simpler to do than in a normal house ... I don't want to work through a menu just to turn off the lights" [20, p. 232]. Eight of the fourteen participants in a 2011 study cite complex user interfaces as one of the main downsides of their smart home [5]. Especially guests that are unfamiliar with the system are often unable to use it at all; one smart home inhabitant describes how his/her mother sat in the dark during her visit because she was too scared to touch the controls [5]. Similar sentiments are also reported in a study from 2012 [17].

Inflexibility Several problems arose in the studied smart homes because of inflexible rules and scenes. One inhabitant mentioned that "It bothers me when it turns on the light ten times and I actually don't need it" [17, p. 156]. Another simply accepts the problems "I just accept that the shades are down and then I just go to the door to look outside" [17, p. 156]. One user discussed, how he scaled back from his initial ideas of a wake scene with music playing and lights switching on automatically because his life is much less structured than implied by the setup of the scene [5]. He concluded "So I don't think the routineness of automation is what I was really wanting." [5, p. 6].

Fear of control loss It is important that researchers are aware of the fear of a loss of control that is prevalent amongst potential smart home users [24,11,22,4]. Often potential users are not comfortable with the idea that the system autonomously performs some action on their behalf. Eggen et al. strongly stress that "people want control over when and how things are done, and to what degree the home takes over" [11, p. 7]. Other

participants worry that there may be no way to manually override any automation [24].

Addressing these issues The proposed recommender system addresses the inflexible configuration of today's smart homes by removing the need to manually describe user routines. The system continuously produces service recommendations that are fitted to the user's current situation and updated whenever the situation changes. Several strategies for utilizing the recommendations will be presented, so that users themselves can decide how much control they are willing to give up for an increase in usability and comfort.

2.2. Terminology

Researchers stress that smart homes must be context-aware, i.e. must be attentive to the current situation and the current needs of the inhabitant. The context-aware home is always ready to assist the inhabitant without needing complicated instructions.

User context Information about user context can be provided by cheap and easy-to-deploy sensors. User context can contain knowledge about the user (e.g. location, medical information) and the user's environment (e.g. temperature). The latter also includes the status of electronic devices (e.g. whether the media center is currently paused or playing), furniture and house components (e.g. are doors/windows currently open or closed?) and usage of items (e.g. whether the pan is at its normal location). Any sensor event can be interpreted as a change in the user context.

User action A user action is any action that is manually performed by the user. The action may include some interaction with the smart home system (opening a door, using an item) or be self-contained (the user walks into a different room). User actions cannot be observed directly by the smart home system. Instead, it can only observe sensor events (i.e. context changes) that are caused by the action. Consider opening the fridge: unless the smart home is equipped with a video camera system with live-image recognition, the only way to know that this user action happened is from a sensor event "fridge=open". If the user action does not cause any context changes (e.g. due to lack of suitable sensors), the system has no way of knowing that the action occurred.

System action/service A system action is any action that can be automatically performed by the system, typically in form of a service. A service is called context-altering, if executing it changes the user context in some way. For example, a service "Turn on dining room lights" would result in a context-change "dining_lights=on". A system action matches a user action, if both result in the same context changes. The idea of matching system actions to user actions was inspired by Bellotti and Edwards [3].

2.3. Overview of the recommender system

Recommender systems are widely used in e-commerce applications to provide users with hints on products or services they could be interested in. These recommendations are based on the user's personal history in using the application (e.g. which products the user bought) and are oftentimes cross-referenced with other users' histories (e.g. those who bought book1, also bought book2).

We propose a recommender system for the smart home that works analogously. In high-level terms, the output of the system are recommendations such as "in the current situation you typically perform actionX, would you like me to perform it for you?". The method draws only from the user's previous behavior in the smart home and does not cross-reference with other users.

Limitations The proposed recommender system can currently only be used in single-person households. The reason is that the recommendations are personalized to one inhabitant's context and routines. The final section of this paper considers how it can be extended to multi-person households.

Example scenario The basic idea for the recommender system is demonstrated in Figure 1 using a smart home dataset made available by van Kasteren et al. [25]. The figure shows the four most common actions the user performs within five minutes of closing the front door of the smart home.

Surprisingly, the most common user action is to immediately open the front door again. However, this action is only common within the first 40 seconds of closing the door – possibly the user has forgotten something in the car or can not carry all bought groceries inside at once. If a longer time has passed since closing the door, the user often heads either to the bathroom or for the fridge. If

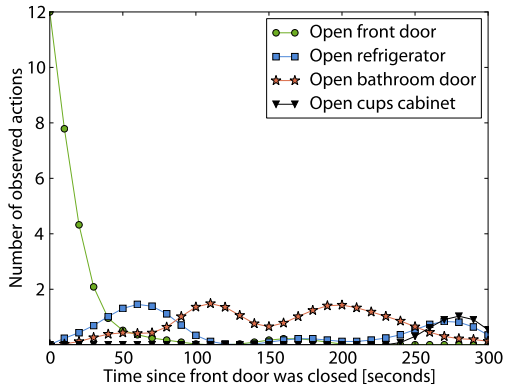


Fig. 1. Typical actions after closing front door

one looks even further into the future, say 20 minutes since closing the front door, no dominating actions can be found. Similar observations can be made in this dataset for many other user routines. The data supports rather detailed observations, e.g. after opening the dishwasher, actions concerning the plate and the cups cabinet are more likely than other kitchen actions.

The two phases of the system The proposed method extracts these temporal relationships between user actions from a user's smart home history. For any current user context, the system then tries to identify what the user most likely wants to do next and recommends some system actions that best match the user's intentions. The recommender system has two phases:

Training phase (Section 4) Input is a sequence of sensor events which is collected while a user is living in the smart home. The recommender system extracts, how often and under which temporal relations specific events happen in any given situation.

Recommendation phase (Section 5) Input is the current user context (i.e. the current status of all available sensors). The recommender system predicts the most likely next context changes based on the learned behavior model. The predicted context changes represent potential next user actions. The system then identifies which context-altering services result would result in one of the predicted changes, i.e. which service can automate possible next user actions. Output is a ranked list of service recommendations, plus a measure of recommendation uncertainty and conflict.

2.4. Utilizing the service recommendations

This section presents several strategies for utilizing the service recommendations and discusses their impact on the issues of control loss and usability.

2.4.1. Active context-awareness

Chen and Kotz [6] make the distinction between active and passive context-awareness. An active context-aware application "automatically adapts to discovered context, by changing the application's behavior" [6, p. 3]. An active context-aware smart home continuously senses the user context, interprets it using the recommender system and autonomously acts on the recommendation results by executing the best recommendation. Active context-awareness requires that very precise service recommendations are generated. The smart home must be able to decide *(a) if a service should be executed in the current situation*, *(b) which service should be executed* and also *(c) exactly when to execute it*.

Pinpointing the best time of execution may be even more difficult than identifying the correct service to execute. For example, imagine that the smart home inhabitant has just opened the refrigerator to take out some breakfast ingredients. It is not very hard to predict that one of the inhabitant's next actions will be to close the refrigerator, and hence to identify the corresponding service *close refrigerator* as the best service recommendation. However, the home should avoid closing the refrigerator before the inhabitant is finished with retrieving the needed groceries. Even if the chosen execution time is based on additional data (e.g. the user is not near the refrigerator anymore), the user may get annoyed if the refrigerator closes prematurely (actually the user was just transporting some ingredients to the stove and planned to return to the refrigerator).

There is a risk that active context-awareness elicits feelings of control loss even if recommendations and timings are perfect. With imprecise recommendations and timings, this loss of control may no longer be compensated by an increased level of convenience. The inhabitant could quickly become frustrated and switch off the system.

2.4.2. Passive context-awareness

Instead, we are turning towards what Chen and Kotz call passive context-awareness [6]. With this strategy the smart home also continuously senses the user context and interprets it using the recommender system. However, the home does not autonomously execute some service, but instead waits for input from the user.

To achieve passive context-awareness, the service recommendations are used to build a very simple graphical user interface (GUI) that shows only the currently most relevant choices. In case the user wants to perform some other, less typical action, a list of all services may be hidden behind an additional button. This solution increases usability but keeps the user still fully in control.

2.4.3. More natural interaction

Even with a simplified GUI, standard touch-screen operated devices such as tablets or smart phones will likely involve too much overhead for most users. In many cases it will be much simpler to perform the desired action manually than to retrieve the device from the pocket or wherever it was left previously, switch it on, and select the correct service. Such an input device may be most interesting for users that have trouble performing some tasks manually, e.g. opening heavy drawers.

Several more natural solutions for interacting with smart homes have been proposed, e.g. using wearable computing devices [15], speech recognition [28] and gestures [19]. In wearable computing devices, display space is often severely limited. Here it becomes even more important to reduce the selection choice to only those services that are currently relevant for the user. In speech and gesture recognition system, service recommendations could be used to improve recognition rates. Whenever recognition confidence is low and there are several potential service matches for the user's command, the system can cross-check with the recommendations to decide which of the options is most probable given the user's situation.

2.4.4. Adding some active context-awareness

If there are several alternative services that all fulfill some similar goal, a GUI might show only the most recommended of these services and indicate that there are alternatives. One might even combine passive context-awareness with some aspects of active context-awareness. Instead of letting the user select from the alternatives, the GUI presents a summary of the choices and the smart home chooses which alternative to execute. For example, the smart home senses that it is too dark in the room and recommends that one or several lamps should be switched on. Instead of listing specific recommendations, the GUI could simply display a virtual light switch. When the user selects this option, the smart home decides which lamp to turn on.

This mixed strategy avoids the timing issues of active context-awareness, since the cue for executing a service comes from the user. It also lends itself very well for the use with speech and gesture recognition systems. Instead of having to specify exactly which device should execute some service, the user could control the home using much more imprecise commands.

3. Definitions

The recommender system is built on a formal model of context that was first proposed in [14]. Some parts of this model are shortly summarized here, with small adjustments to allow reasoning about temporal data.

3.1. Context dimension

We model context as a multi-dimensional space. Each sensor maps to a context dimension d, which has some numeric or nominal value domain v. In this paper we restrict ourselves to nominal sensors. The reason is that most user actions involve changing the internal status of some device or home component, which is typically expressed as a nominal dimension, e.g. $d = \mathit{fridge}$ with a value domain $\{\mathit{open}, \mathit{closed}\}$. The user can rarely directly alter any numeric dimensions (e.g. a change in indoor temperature is a result of interacting with other smart home infrastructure such as heating and windows).

3.2. Context setting

A context setting on dimension l is expressed as $d_l = v_k$, e.g. $\mathit{fridge} = \mathit{open}$. Let C be the set of all possible context settings, e.g. $C = \{\mathit{fridge} = \mathit{open}, \mathit{fridge} = \mathit{closed}, \mathit{radio} = \mathit{on}, \mathit{radio} = \mathit{off}, \ldots\}$.

3.3. Current user context and context change

The current user context c^u is a collection of the context information from all available sensors:

Definition 1 (User context) *The current context of a user is a tuple* $c^u = \langle (c_0, \Delta t_0), \ldots, (c_n, \Delta t_n) \rangle$, *where each* $c_i \in C$ *is a context setting and* Δt_i *is the time elapsed since this setting has been active.*

The current user context also contains information about the time that the user has been in the current situation. All temporal information is given in relative terms, i.e. as the time since something happened. All timing information will be given in seconds throughout this paper.

A context change is the result of a sensor event and is denoted as Δc. Δc is a context setting with $\Delta c \in C$. After a context change, the user context c^u has to be updated with the new context setting.

3.4. Example

Figure 2 shows as an example a short sequence of six sensor events, which could have been recorded while the user prepares some light snack.

Fig. 2. 10 minutes of smart home activity

Table 1 lists some example user contexts based on this sequence of sensor events. After the fourth event, the fridge has been closed since 235 seconds, the cabinet open since 210 seconds and the dishwasher open since 0 seconds. Five seconds later, the settings remain the same, but the elapsed times increased by five seconds. With the fifth sensor event, the user context is updated with a context change $cabinet = closed$ and the elapsed time for this specific context dimension is reset.

Directly after the fourth sensor event
$c^u = \langle (fridge = closed, 235), (cabinet = open, 210), (dishwasher = open, 0) \rangle$
Five seconds after the fourth sensor event
$c^u = \langle (fridge = closed, 240), (cabinet = open, 215), (dishwasher = open, 5) \rangle$
Context change caused by fifth sensor event (105 seconds after fourth sensor event)
$\Delta c = cabinet = closed$
$c^u = \langle (fridge = closed, 340), (cabinet = closed, 0), (dishwasher = open, 105) \rangle$

Table 1
Examples for user context and context change

4. Training phase

Input to the training phase of the recommender system is a sequence of sensor events which is collected while a user is living in the smart home. The system extracts, how often specific events happen in any given situation. Since learning is performed on the basis of observed sensor events, the system can learn the user behavior from manually performed user actions as well as from service invocations. This means that users can continue to live in their home just as normal and do not have to switch to a smart home user interface for all of their interactions with the home.

Typically, there will not be enough training data to exhaustively cover all possible user situations. With only ten binary context dimensions, already 2^{10} different user contexts could occur. For this reason, the recommender system looks at each context dimension individually, analogously to established algorithms such as Naïve Bayes[1]. For example, it will extract information about which context changes are common if $cabinet = open$, which changes are common if $cabinet = closed$, if $fridge = open$, etc.

4.1. Extracting observation tuples

From the training data a set of tuples of the form $(c_i, \Delta c_j, \Delta t_i)$ is extracted, where c_i is a context setting, Δc_j is an observed context change and Δt_i is the time elapsed since c_i has changed at the moment where Δc_j occurred. For example, at 10:06 the user from Figure 2 opens the fridge and 30 seconds later closes it again, so the extracted tuple is $(fridge = open, fridge = closed, 30)$. Afterwards, $fridge = closed$; while this setting is active, four other sensor events happen, so four further tuples can be extracted:

[1] See [13] for an introduction to Naïve Bayes.

(*fridge=closed*, 25, *cabinet=open*)
(*fridge=closed*, 235, *dishwasher=open*)
(*fridge=closed*, 340, *cabinet=closed*)
(*fridge=closed*, 360, *dishwasher=closed*)

The first tuple expresses that the cabinet door was opened 25 seconds after the fridge was closed. The other three tuples express that 235 seconds after closing the fridge the dishwasher was opened, that the cabinet was closed 340 seconds after fridge was closed, and that the dishwasher was closed 360 seconds after the fridge was closed.

4.2. Dealing with temporal data

We showed earlier that any prediction about the next context change must be based on the current user situation, as well as on the time that the user has been in this situation. Thus, for any given context setting c_i we want to know how likely a change Δc_j is, given that the user has been in c_i since Δt_i time. Let Δt_{max} be the maximum Δt for which such temporal information is collected. This maximum reflects the loss of correlation between current context and subsequent context changes if the user has been in the situation for a long time.

Avoid over-fitting and over-generalizing When extracting temporal relationships between user situations and typical context changes, it is important to avoid over-fitting to the training data. There will always be some variations in user activities that should be tolerated by the algorithm. For example, it should not matter whether the user re-opens the front door after 3 or after 5 seconds. On the other hand, there can be a risk of over-generalizing the user behavior, e.g. if no difference is made between opening the front door instantly or after 5 minutes. The test smart home datasets show a need to be more precise for small Δt and more general for large Δt. To this end, the $[0, \Delta t_{max}]$ interval is divided into a number of intervals with increasing width. For example, for the Kasteren houseA dataset, $\Delta t_{max} = 300$ and an interval width of 10 seconds for $\Delta t \in [0, 60]$ and 30 seconds for $\Delta t \in (60, 300]$ were used.

Interval lookup A function $b(\Delta t)$ is defined to look up the correct interval for a given Δt (Equation 1).

$$b(\Delta t) = \begin{cases} -1 & \text{if } \Delta t > \Delta t_{max} \\ \textit{index of interval} & \text{otherwise} \\ \textit{which includes } \Delta t & \end{cases} \tag{1}$$

4.3. Counting the observations

The final step of the training phase is to aggregate the extracted tuples $(c_i, \Delta c_j, \Delta t_i)$ into the defined intervals. To make the intervals comparable with each other, the number of occurrences in each interval is divided by the interval's width. Moving average smoothing with window size 10 is applied to the intervals to further reduce the risk for over-fitting. A number of look-up functions for the smoothed counts are defined:

$\text{count}(\Delta c_j \mid c_i)$	How often Δc_j has been observed in c_i
$\text{count}(\Delta c_j \mid c_i, b)$	How often Δc_j has been observed in c_i in time interval with index b, $\text{count}(\Delta c_j \mid c_i, -1) = \text{count}(\Delta c_j \mid c_i)$

These functions return the number of observations of Δc_j in a situation c_i in total and in the different time intervals. For large Δt_i, temporal information is not useful, so the system falls back onto the total counts. Finally, some sums of the counts are defined to simplify later calculations:

$\text{csum}(c_i)$	Total number of observations for c_i, $\text{csum}(c_i) = \sum_{\forall \Delta c_j} \text{count}(\Delta c_j \mid c_i)$
$\text{csum}(c_i, b)$	Total number of observations for c_i in interval b $\text{csum}(c_i, b) = \sum_{\forall \Delta c_j} \text{count}(\Delta c_j \mid c_i, b)$

maxtotal	Maximum number of total observations for any c_i $\text{maxtotal} = \arg\max_{\forall c_i} \text{csum}(c_i)$
maxtemp	Maximum number of observations for any c_i in any interval $\text{maxtemp} = \arg\max_{\forall c_i, \forall b \neq -1} \text{csum}(c_i, b)$

5. Recommendation phase

Input to the recommendation phase is the current user context (i.e. the current status of all available sensors). The recommender system predicts the most likely next context changes based on the learned behavior model (Section 5.1). It then identifies, which context-altering services result would result in one of the predicted changes, i.e. which service can automate possible next user actions (Section 5.2). Output is a ranked list of service recommendations, plus a measure of recommendation uncertainty and conflict (Section 5.3).

5.1. Predicting context changes

5.1.1. Dempster-Shafer theory of evidence

Dempster-Shafer theory is a framework for achieving a consensus between several different information sources. It is a generalization of Bayesian probability theory. The popular Naïve Bayes algorithm (which is compared to our method in Section 6) is one application of Bayesian probability theory.

In contrast to Bayesian theory, Dempster-Shafer allows attribute belief mass not only to individual hypotheses, but also to combinations of hypotheses. This makes Dempster-Shafer theory a valuable tool in human-centered environments, which are typically governed by variations, and thus uncertainties, in the subject's behavior. In the following a short introduction to the theory is given, more details can be found in [23].

In Dempster-Shafer theory, the frame of discernment Θ is the set of all hypotheses of interest. In our case, the frame of discernment is made up from all context changes that could occur given the user's current situation. This means that Θ changes if c^u changes, and thus the frame of c^u is denoted as Θ_{c^u}. For example, if $c^u = \langle(\textit{fridge} = \textit{closed}, 15), (\textit{cabinet} = \textit{open}, 120), (\textit{bedroomdoor} = \textit{closed}, 3400)\rangle$, then the frame of discernment will be $\Theta_{c^u} = \{\textit{fridge} = \textit{open}, \textit{cabinet} = \textit{closed}, \textit{bedroomdoor} = \textit{open}\}$. For each available sensor, there is at least one (in case of binary sensors), but potentially several (for more complicated sensors) such possible changes in Θ_{c^u}.

5.1.2. Information sources

Each part of the current user context is considered a source of information. Above example context contains three information sources: (*fridge=closed*, 15), (*cabinet=open*, 120) and (*bedroomdoor=closed*, 3400). A belief distribution is then calculated for each source. This means that each source attributes some masses to context changes in Θ_{c^u}, which reflect how probable each change is according to the source. E.g., the source (*fridge=closed*, 15) would probably attribute large masses to *fridge* = *open* and *cabinet* = *closed*, but small masses to *bedroomdoor=open*, since the former two are much more probable in this situation.

Uncertainties are expressed by attributing some mass to combinations of elements or to Θ_{c^u} itself. For example, if it is quite likely that some context change will happen in the kitchen, but there is high uncertainty exactly which change will happen, one could attribute a large mass to the combination of all kitchen-related context changes, but a small mass to each of the individual changes. The masses attributed by one source must always add up to 1.

5.1.3. Source weight

The evidences of a source with a high number of observations should be trusted more than one with only a few observations. Sources for which no temporal information is available, i.e. $\Delta t > \Delta t_{max}$, should have a very small weight. Such sources should only be relevant for finding the combined belief if no source with $\Delta t \leq t_{max}$ is available. i.e. temporal knowledge always trumps general knowledge.

Equation 2 calculates the weight of a source i with $(c_i, \Delta t_i)$. If the source has temporal knowledge, its weight is based on the number of observations in the current interval, scaled by the maximum number of observations of any source in any interval. Otherwise, the weight is a based on the total number of observations of this source, scaled by the number of observations of the overall best

source. In this case, i is additionally discounted by a small ϵ_w, e.g. $\epsilon_w = 10^{-4}$.

$$weight_i = \begin{cases} \dfrac{csum(c_i, b(\Delta t_i))}{maxtemp} & \text{if } b(\Delta t_i) \neq -1 \\ \dfrac{csum(c_i)}{maxtotal} * \epsilon_w & \text{otherwise} \end{cases} \tag{2}$$

5.1.4. Attributing the masses

The masses attributed by a source are based on the counts extracted from the data during the training phase. A context change with a high number of observations should be attributed a larger mass than one with a low number of observations. Equation 3 calculates the mass for one context change Δ_{c^u} for a source i with $(c_i, \Delta t_i)$. The mass attributed to a change $\Delta c_j \in \Theta_{c^u}$ is the ratio of the number of observations of Δc_j in interval $b(\Delta t_i)$ to the total number of observations in this interval. The calculated mass is then discounted by the weight of the source. In a slight abuse of notation, we write $m(\Delta c_j)$ instead of $m(\{\Delta c_j\})$.

$$m_i(\Delta c_j) = \frac{count(\Delta c_j | c_i, b(\Delta t_i))}{csum(c_i, \Delta t_i)} * weight_i \tag{3}$$

Finally, the mass put on Θ_{c^u} can be calculated as the remaining masses for reaching a sum of 1 (Equation 4).

$$m_i(\Theta_{c^u}) = 1 - \sum_{\Delta c_j \in \Theta_{c^u}} m_i(\Delta c_j) \tag{4}$$

It holds that $m_i(\Theta_{c^u}) = 1 - weight_i$. For a large weight, much mass is put on the context changes and only a small mass is put on Θ_{c^u}, i.e. the uncertainty of the source is low. For a small weight, less mass is put on the context changes, more mass is put on Θ_{c^u}, i.e. the uncertainty of the source is high.

5.1.5. Combining the sources

The different pieces of evidence are combined using Dempster's rule of combination [23]. Dempster's rule requires information sources to be independent. This assumption holds true for the test datasets. In case of dependent information sources, the cautious rule of combination can be used instead [10]. Dempster's rule is given in Equation 5a.

$$m_{1\oplus 2}(A) = \sum_{B \cap C = A, A \neq \emptyset} m_1(B) * m_2(C) \tag{5a}$$

Since we only assign masses to singleton elements and to Θ_{c^u}, the calculation can be simplified as shown in the example in Equation 5b.

e.g., combine sources 1 and 2 for change Δc_j

$$\begin{aligned} m_{1\oplus 2}(\Delta c_j) = {} & m_1(\Delta c_j) * m_2(\Delta c_j) \\ & + m_1(\Delta c_j) * m_2(\Theta_{c^u}) \\ & + m_1(\Theta_{c^u}) * m_2(\Delta c_j) \end{aligned} \tag{5b}$$

The conflict between the sources, i.e. the masses that can not be attributed to any elements or subsets of Θ_{c^u}, can be calculated according to Equation 6a [23], an example is given in Equation 6b.

$$K_{1,2} = \sum_{B \cap C = \emptyset} m(B) * m(C) \tag{6a}$$

e.g. for sources 1 and 2 and $\Theta_{c^u} = \{\Delta c_j, \Delta c_k\}$:

$$\begin{aligned} K_{1,2} = {} & m_1(\Delta c_j) * m_2(\Delta c_k) \\ & + m_1(\Delta c_k) * m_2(\Delta c_j) \end{aligned} \tag{6b}$$

More than two sources can be combined iteratively, i.e. combine the first two sources, then combine the result with the third source, etc. After combining all sources in c^u, the plausibility that a context change Δc_j will occur next can be calculated: $pls_{c^u}(\Delta c_j) = m_{c^u}(\Delta c_j) + m_{c^u}(\Theta_{c^u})$. One also obtains K_{c^u} as an overall measure of the conflict between the sources and $m_{c^u}(\Theta_{c^u})$ as an overall measure of the uncertainty.

In the implementation we chose to transform masses into commonalities and use Dempster's rule of commonalities to combine sources [23]. This step converts the combination equation into a different form, which can be calculated much faster. The final outcome of the calculation is the same.

5.2. Service matching

The output of the previous step is a list of possible context changes with their plausibilities, plus the measures of conflict and uncertainty. Each of these context changes can be representative of some user action, i.e. the context change is a result of performing this action. The higher the plausibility of a context change, the more common is also the respective user action. The next step is now to match available services to the predicted context changes. We say that a context-altering service matches a context change, if executing it has as effect the desired context change. We assume that information about service effects is available to the recommender system; see [21] on how the smart home can automatically learn service effects.

Matching between service effect and predicted context change is trivial for nominal context dimensions. In this case, there can only be "match" or "no match". The interested reader may refer to [14], where matching between context descriptions is also defined for numeric context data, were even partial overlaps are possible. For now, assume that a context change $fridge = open$ was predicted. Then a service with effect $fridge = open$ would be a match. A service with effect $fridge = closed$ would not be a match, neither would a service with effect $tv = on$.

For each of the predicted context changes the algorithm goes through the list of available context-altering services. In some cases, no matching service will be found, either because the context change happens automatically (e.g. the toilet flush sensor will automatically switch to off) or because there is no service that corresponds to the user action (e.g. user walks into different room). If a matching service is found, it will be added to a list of service recommendations. If several services match the same context change, the services are *alternatives*, and can be marked as such in the user interface.

Output of the matching step is a list of service recommendations. The list is ordered by the plausibilities of the respective context changes, i.e. first in the list is the service that matches the context change that has the highest plausibility. Only the order of the recommendation list is used at the moment, the actual plausibility values are currently not used.

The calculated recommendations are only valid for the current user context, and have to be recalculated whenever the context changes. Because the time passed since a context setting changed is part of the user context, this means that recalculation has to be performed every second (assuming that time resolution is one second, i.e. each Δt in the user context is updated every second). In reality, the service recommendations will be stable for a longer time frame, depending on the selected time intervals. However, since the algorithm can calculate recommendations within a few milliseconds (see Section 6), it is very feasible to simply execute it every second.

5.3. Interpreting conflict and uncertainty

The previous steps returned not only a list of service recommendations, but also some measure of the conflict and the uncertainty in the system. In the following the edge cases low/high uncertainty and low/high conflict are discussed. These considerations will be revisited in the evaluation section.

High uncertainty, low conflict No temporal sources are available, therefore there is not enough information to make reliable recommendations. The user-interface should present a layout which allows to select from the full list of services. More common services could be highlighted.

High uncertainty, high conflict Does not occur.

Low uncertainty, low conflict There are one or several temporal sources that are in agreement. This is the best-case scenario. The user-interface should display only the few best recommendations. The other recommendations should still be accessible, but can be hidden behind an extra button in the interface. Only in this case, active context-awareness could be feasible.

Low uncertainty, high conflict There are several temporal sources that are in disagreement. The user-interface should display several of the best recommendations. Again, the other recommendations should still be accessible, but can be hidden behind an extra button in the interface.

6. Evaluation

In this section the proposed algorithm is evaluated and compared to a Naïve Bayes classifier using two publicly available smart home datasets. It is also evaluated how the choice of time intervals influences the recommendation results and the usefulness of the conflict and uncertainty measures is explored. The section finishes with an evaluation of the algorithm run-times.

6.1. Procedure

Input to the experiments is a sequence of sensor events. Since the order of the events is important, the dataset cannot be randomized. Instead 10-fold cross- validation is performed as follows: in the first fold, the first 10% of events are test data, the rest are training data; in the second fold the second 10% of events are test data, the rest are training data, etc. Following procedure is applied to each of the ten folds:

1. Train the model on the training data using the selected method.
2. Then for each sensor event e in the test data:

(i) Update the current user context and calculate possible context changes.
(ii) Calculate the probabilities for these context changes using the selected algorithm.
(iii) Identify from e which user action a was performed, if e cannot be matched to any user action, then do not proceed further for e.
(iv) Calculate service recommendations based on the predicted context changes.
(v) Compare the highest ranked recommendations with a.

The necessary descriptions of service effects and user actions were provided manually, see the sections on the individual datasets for more details. The method was fully implemented in Python, the source code is available online[2]. Experiments were performed on a PC with an Intel Core 2 Duo CPU 3 GHz and 4 GB RAM.

[2]The source code is available at `https://github.com/krasch/smart-assistants`

6.2. Metrics

The recommendation results are evaluated using the standard metrics of precision, recall and F1 [3]. For each service, count the number of true positives (tp), false positives (fp) and false negatives (fn). Table 2 explains true positives, false positives and false negatives using as example the service/action *Open frontdoor*.

	recommended service	**actual action**
true positive	*Open frontdoor*	*Open frontdoor*
false positive	*Open frontdoor*	any other action
false negative	any other service	*Open frontdoor*

Table 2
True positives, false positives and false negatives for example *Open frontdoor*

Recall corresponds to the true positive rate ($recall = \frac{tp}{tp+fn}$): whenever the *Open frontdoor* user action occurs in the test data, how often is the correct service recommended? Precision measures, how relevant the recommendations are ($precision = \frac{tp}{tp+fp}$) – whenever *Open frontdoor* is the best recommendation, how often does it actually occur? F1 is the harmonic mean of precision and recall ($F1 = 2 * \frac{precision*recall}{precision+recall}$). The overall precision, recall and F1 are calculated as the average over all services, weighted by number of their occurrences. Precision, recall and F1 all fall into range $[0.0, 1.0]$. All results are listed using the 90% confidence interval over 10 folds.

6.3. Used datasets

Kasteren houseA dataset This dataset was made available by van Kasteren et al. [25]. It contains observations of the inhabitant of a smart home over a period of 25 days. Fourteen binary sensors are used for the open/close status of frontdoor, kitchen cabinets, bedroom door, fridge, freezer, dishwasher and washing machine, plus whether the toilet flush is on or off. For all sensors except the toilet flush, two matching services and human actions were identified: open door/fridge/etc and close door/fridge/etc. The context change *toi-*

[3]Further information on these standard metrics can be found in [2, Chapter 4]

letflush=off happens automatically, so the toilet flush can only be described by one service/human action "activate flush". In total the dataset contains thus 28 context settings and 27 services. Since the sensor are binary, at any given moment there are at most 14 possible context changes and possible user actions. The experiments use $\Delta t_{max} = 300$ and an interval width of 10 seconds for $\Delta t \in [0, 60]$ and 30 seconds for $\Delta t \in [60, 300]$.

Kasteren houseB dataset The second test dataset was also made available by van Kasteren et al. [26]. This dataset was recorded over the course of 13 days, in a different smart home, which was equipped with 23 binary sensors. Some of these sensors (pressure mats on bed and chairs, infrared sensors) can not be mapped to any services. The toilet flush and mercury switches (used to detect if there is some movement at drawers and doors) can only be mapped to one service each. Overall there are 44 possible sensor statuses, but only 26 available services. The same interval setup as for houseA dataset was used in this experiment.

6.4. Comparison with Naïve Bayes

The proposed method is compared with a Naïve Bayes classifier. Since Dempster-Shafer theory is a generalization of Naïve Bayes, this method is equivalent to the proposed method without the temporal properties and without any non-specificity. A random classifier is used as a baseline in the comparison.

Only the best service recommendation is used Table 3 lists evaluation results when only the one best service recommendation is used, i.e. the user is shown only one service. On the houseA dataset our method achieves a recall of 0.61. This means that in 61% of cases the recommended service matches exactly the action that the user would perform next. For the houseB dataset, the recall is 0.73, i.e. the correct service is recommended in 73% of all cases[4].

Our method significantly outperforms Naïve Bayes for both datasets. The Naïve Bayes algorithm will always predict the most common context change without any regard for temporal relations, i.e. it will always predict "Close cupboard", regardless of how long the cupboard has been open. The evaluation results show that this approach is too coarse and results in much lower recall and precision when compared to our approach.

Method	Recall	Precision	F1
houseA dataset			
Our method	0.61 ± 0.02	0.63 ± 0.04	0.59 ± 0.02
Naïve Bayes	0.48 ± 0.03	0.38 ± 0.03	0.40 ± 0.03
Random	0.08 ± 0.01	0.30 ± 0.05	0.10 ± 0.02
houseB dataset			
Our method	0.73 ± 0.12	0.78 ± 0.10	0.72 ± 0.12
Naïve Bayes	0.55 ± 0.20	0.50 ± 0.21	0.51 ± 0.21
Random	0.11 ± 0.04	0.49 ± 0.19	0.16 ± 0.07

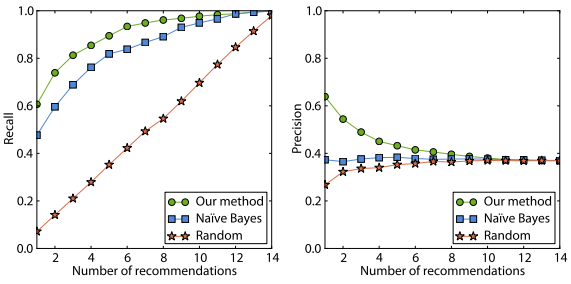
Table 3
Results for both datasets when only the best service recommendation is used

Results when showing several services In the next experiment, the user can select from several services. If several service recommendations are displayed in the user interface, then often recall increases (only one of the recommendations has to be correct) and precision decreases (several of the recommendations are irrelevant).

The results of the experiment for the houseA dataset are shown in Figure 3. On the left-hand side, it can be seen that the recall increases with the number of service recommendations for all methods. If the user is shown five services, the recall reaches 0.9, i.e. in 90% of cases the correct service is included in the top five recommendations. For less than 13 recommendations, our method always outperforms Naïve Bayes. In this dataset, the user can typically be recommended at most 14 services [5], thus for 14 recommendations all algorithms achieve similar results. For up to 9 recommendations, our method has higher precision than Naïve Bayes; for more than 9 recommendations both algorithms have similar precision.

[4] For the houseB dataset some confidence intervals are rather wide. The reason is that the dataset is partially dominated by one of the mercury switch sensors. In some of the folds, events for this sensor occur extremely often, which makes prediction easier.

[5] An exception is the initial phase of the evaluation, where the initial status of the devices is not known. For example, since it is not known, whether a door is open or closed at the beginning of the dataset, the user is shown both services for opening and closing this door.



(a) Recall (b) Precision

Fig. 3. Number of recommendations vs precision and recall for houseA dataset

The same experiment was performed with the houseB dataset, and it was found that in 90% of cases the correct service is contained in the four best recommendations calculated by our method.

6.5. Choice of time intervals

It is important to evaluate, how susceptible prediction accuracy is to the choice of time intervals. Table 4 lists recall, precision and F1 for the houseA dataset for several different interval choices. In the first half of the table, shorter and longer settings for Δt_{max} are evaluated. When decreasing Δt_{max}, the algorithm is quite stable and recall degrades only slowly. Only very short $\Delta t_{max} <= 30$ should be avoided. Increasing the Δt_{max} to 1200 has no significant effect on the algorithms performance.

Intervals	**Recall**	**Precision**	**F1**
baseline (Table 3)	0.61 ± 0.02	0.63 ± 0.04	0.59 ± 0.02
Influence of Δt_{max}			
$\Delta t_{max} = 60$	0.59 ± 0.02	0.66 ± 0.03	0.58 ± 0.02
$\Delta t_{max} = 30$	0.58 ± 0.02	0.66 ± 0.02	0.57 ± 0.01
$\Delta t_{max} = 20$	0.55 ± 0.02	0.66 ± 0.03	0.54 ± 0.02
$\Delta t_{max} = 10$	0.21 ± 0.04	0.11 ± 0.03	0.11 ± 0.03
$\Delta t_{max} = 1200$	0.61 ± 0.02	0.64 ± 0.03	0.58 ± 0.02
Influence of interval widths (same width for all intervals)			
width=2s	0.60 ± 0.02	0.66 ± 0.03	0.59 ± 0.02
width=4s	0.61 ± 0.02	0.67 ± 0.04	0.59 ± 0.02
width=6s	0.61 ± 0.02	0.66 ± 0.03	0.59 ± 0.01
width=30s	0.58 ± 0.02	0.58 ± 0.02	0.55 ± 0.02
width=100s	0.54 ± 0.02	0.59 ± 0.02	0.52 ± 0.02

Table 4

Influence of time intervals on prediction accuracy for houseA dataset

In the second half of Table 4, the algorithm is tested with different interval widths, while $\Delta t_{max} = 300$. The results show that even very short intervals do not lead to over-fitting to the test dataset. However, for long intervals, the prediction accuracy degrades due to the over-generalization of the user behavior. The results indicate that the algorithm is quite stable with regard to the choice of time intervals, as long as very wide intervals and very small Δt_{max} are avoided.

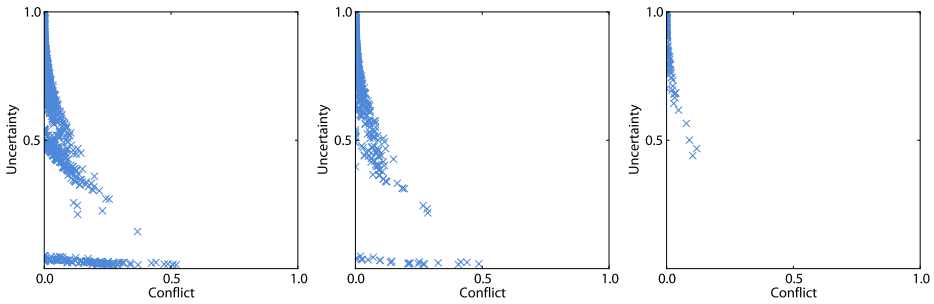
6.6. Exploring conflict and uncertainty

The potential use of the measures of conflict and uncertainty was discussed earlier. In the following a first evaluation of the actual information value of these measures is presented. Please note that a partially supervised approach is used for this initial evaluation, in contrast to the rest of this chapter.

6.6.1. Exploration experiment

For this experiment, the algorithm was applied the whole Kasteren houseA dataset, without any cross-validation. Each cross in the scatter-plots in Figure 4 represents one service recommendation; the x-coordinate of a cross represents how much conflict the system had when making this recommendation and the y-coordinate represents how much uncertainty there was in the system. Scatter-plot (a) shows only the most successful service recommendations, where the service with highest probability matches the actually performed user action. It is not possible to identify any conflict-uncertainty region where the algorithm is more or less successful.

Scatter-plot (b) shows less successful recommendation results, where the correct service was not in the best two recommendations, i.e. the user would have to be shown three or more service recommendations. There is a much lower density of data points in the lower-lefter corner of the plot compared to plot (a), while there are still a large number of points in the upper-left corner. The final scatter-plot (c) shows even less successful results, where the correct service was not in the best four recommendations. It can be seen that there are no longer any data points in the lower-left corner of the plot, i.e. for low conflict and low uncertainty, the correct service is found within the four best recommendations.



(a) First rec. is correct (b) At least 3 rec. needed (c) At least 5 rec. needed

Fig. 4. Recommendation success vs conflict and uncertainty

6.6.2. Using conflict and uncertainty to reduce the number of recommendations

A second experiment was performed to show how the number of recommendations that are presented to the user can be reduced based on conflict and uncertainty. Let *cutoff* be the maximum number of recommendations that are shown to the user. However, if uncertainty<0.4, show only *dynamic cutoff*<*cutoff* recommendations, i.e. make the selection easier for the user if we can be reasonably sure that the correct service will be included in the first *dynamic cutoff* recommendations. Table 5 compares for *cutoff=5* and *cutoff=10* the success of this strategy in terms of average number of recommendations shown, recall and precision.

Cutoff	# of Recs.	Recall	Precision
Maximum 5 recommendations (cutoff=5)			
fixed cutoff=5	5.00 ± 0.00	0.89 ± 0.01	0.43 ± 0.02
dynamic cutoff=4	4.86 ± 0.03	0.89 ± 0.01	0.43 ± 0.02
dynamic cutoff=2	4.57 ± 0.09	0.88 ± 0.01	0.44 ± 0.02
Maximum 10 recommendations (cutoff=10)			
fixed cutoff=10	10.00 ± 0.00	0.98 ± 0.01	0.38 ± 0.03
dynamic cutoff=4	9.13 ± 0.19	0.98 ± 0.01	0.38 ± 0.03
dynamic cutoff=2	8.84 ± 0.25	0.96 ± 0.01	0.39 ± 0.03

Table 5

Results for dynamic selection of number of recommendations

For *cutoff=5*, without any dynamic cutoff, on average five recommendations are shown to the user. This average decreases to 4.86 if the list is cut at four items if uncertainty is low, while still upholding the same high recall. The average decreases further to 4.57 for an *dynamic cutoff=2*. In this case, the recall decreases as well, i.e. the user will less often find the correct service within the shown recommendations. However, one can argue the overall effect is still positive, since recall decreases only by 2%, while the number of recommendations decreases by 8.6%. The usefulness of *dynamic cutoff* is even more pronounced for *cutoff=10*. With an *dynamic cutoff=4* the user is shown only 9.13 instead of 10 items, while recall is equal and precision increases slightly. We think that these initial results are promising and plan to further expand in this direction in the future.

6.7. Runtimes

Table 6 compares the training and prediction times for the methods. All algorithms process the datasets in less than one second. Our method extracts not only overall counts, but also has to sort occurrences into time intervals and perform smoothing; the training times are thus the highest.

Method	Training time [ms]	Prediction time r_{predict} [ms]	
		total	per instance
houseA dataset			
Our method	460.21 ± 11.82	152.48 ± 4.78	0.66 ± 0.02
Naïve Bayes	52.29 ± 2.63	37.35 ± 1.44	0.16 ± 0.01
Random	0.01 ± 0.00	12.46 ± 0.15	0.05 ± 0.00
houseB dataset			
Our method	810.91 ± 35.86	250.22 ± 59.69	0.81 ± 0.19
Naïve Bayes	91.33 ± 6.08	67.91 ± 8.75	0.22 ± 0.03
Random	0.03 ± 0.00	17.04 ± 0.51	0.06 ± 0.00

Table 6

Training and prediction times for both datasets, in milliseconds

More interesting is the time needed for recommendation. In order to be usable in pervasive envi-

ronments, the algorithms must be able to process new context information and calculate service recommendations quickly. In particular, there should be no noticeable delay in the user interface, e.g. as soon as a lamp has been switched off, the corresponding button to switch of the lamp should disappear from the UI or should be grayed out. According to the literature, delays of up to 200 milliseconds are not noticable by users [7]. The results in Table 6 show that while our algorithm has the longest recommendation times, one set of recommendations is calculated in less than one milliseconds.

To test scalability of the algorithm, a synthetic dataset was generated that imitates the data seen in the houseA and houseB datasets. Evaluations performed with the synthetically generated data show that the algorithm scales well also for very large smart homes, e.g. for 500 context dimensions and 2500 services, one set of recommendations is calculated in 43 milliseconds. This means that even for large-scale environments recommendations are generated sufficiently fast for an online recommender system.

7. Related work

7.1. Behavior prediction

Das et al. propose the Smart Home Inhabitant Prediction (SHIP) algorithm for predicting the next action of a user [8]. Given a user's most recent commands, the algorithm identifies matching sequences from the collected history and uses them to predict the next command. Temporal relations between user actions are not considered. To deal with small variations in user routines, the user can set a parameter called inexact threshold that represents the maximum percentage of allowed mismatches; however, no evaluation of the effectiveness of this parameter was performed. In contrast, our algorithm does not require users to perform their actions in regular sequences. Additionally, the SHIP algorithm learns only from the commands that are issued to the home devices, i.e. it learns a history of service invocations. Our method works at the context level and can therefor learn user behavior from manually performed actions as well as from service invocations.

Gopalratnam and Cook propose ActiveLeZi for predicting which device the user will interact with next [12]. The algorithm builds a tree of observed sequences of these interactions. To predict the next action, the algorithm tries to match the most recent inhabitant–home interactions with previously observed sequences. The algorithm was tested in a dataset collected in a test smart home (different dataset than used in this paper) and achieved a prediction accuracy of 47%.

Aipperspach et al. propose an approach for predicting arbitrary sensor events [1]. Their approach is also related to sequence matching; sensor events are modeled as words and tools from Natural Language Processing are used to build a language model of the sensor data. This model can calculate how probable some word is, given the most recent words. Longer pauses in the sequence of sensor events are modeled as a special PAUSE event, other temporal relations are not considered. The authors report 51% prediction accuracy on a smart home dataset (again using a different dataset). They briefly discuss how the predictions can be used, e.g. to predict where the user is heading and turn on the lights in preparation. However, no generalized approach for making use of the predictions is proposed.

Mozer [18] built a neural network that predicts, which zone in the smart home will become occupied in the next two seconds. The predictions are used to switch on the lights in the to-be-occupied zones before the user enters them. The author reports promising initial results. However, the approach is tailored to one specific home installation and supports only location prediction. It remains uncertain, how well the approach is transferable to other smart homes and whether the neural network can scale for predicting further attributes of user context.

The proposed recommender system has several advantages compared to these approaches. It does not rely on the user executing their activities in orderly sequences, instead it predicts the next event based on the current user context. The proposed approach also takes temporal relations between events into account. Instead of learning from user commands, the method works at the context level and can learn user behavior from manually performed actions as well as from service invocations. Finally, the recommender system also includes the

important step of transforming the predictions into personalized service recommendations.

7.2. Use of temporal information in activity recognition

Some recent works in activity recognition also make use of temporal information to improve recognition accuracy. Ye et al propose the usage of absolute temporal information as well as relative temporal information for activity recognition [29] and show that temporal awareness can significantly increase the activity recognition accuracy. Mckeever et al. propose the usage of activity durations in activity recognition [16] and implement a learning algorithm using Dempster-Shafer theory. Evaluation results show a 70% improvement of the recognition f-measure, compared to a non-temporal Naïve Bayes classifier. Both works perform supervised learning.

8. Conclusions and future work

This paper presented an unsupervised recommender system for the smart home. Based on the user's current situation, the proposed system tries to recommend services that match the user's intended next actions. Several options of making use of the recommendation results and their implications on convenience, ease-of-use and potential control loss were discussed. Evaluations were performed on two publicly available test datasets and it was shown that the method *(i)* can produce correct recommendations with 61% and 73% accuracy, *(ii)* significantly outperforms a Naïve Bayes classifier and *(iii)* is stable with regard to choice of parameters, as long as extreme values are avoided.

One avenue to pursue in the future is the utilization of conflict and uncertainty. The evaluations indicated that conflict and uncertainty can be useful tools for identifying how many recommendations should be presented to the user. However, at the moment, important parameters such as conflict and uncertainty thresholds have to be set manually. Further research should investigate how this process can be improved.

A second avenue for future work is to enable the use of the recommender system in multi-person households. One challenge is that the recommender system must be able to build a different behavior model for each inhabitant. This means it must be able to distinguish which user is performing which action, e.g. through the use of indoor positioning systems. A second challenge is that the recommender system should not send recommendations to inhabitant1 that only interest inhabitant2 (e.g. inhabitant1 is in the bedroom, inhabitant2 is in the kitchen, both receive kitchen recommendations, since the smart home detected recent activity in the kitchen)

A final idea for future work is to move towards active context-awareness. The method presented in this paper makes a step towards answering the question of *which service would be most useful in the current situation.* However, additional work must be done to make the smart home able to decide *if a service should be executed* and *when exactly to execute it.* Any solutions for these problems must be tested with actual smart home inhabitants, with a special focus on evaluating whether the increased usability can compensate for potential feelings of control loss.

Acknowledgements The author would like to thank Rassul Ayani and Christian Schulte from KTH Stockholm as well as Johan Schubert from FOI Swedish Defense Research for their helpful comments on this paper.

References

[1] R. Aipperspach, E. Cohen, and J. F. Canny. Modeling human behavior from simple sensors in the home. In *Proceedings of the 4th international conference on Pervasive Computing*, pages 337–348, 2006.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval, Second Edition.* Addison-Wesley, Boston, MA, USA, 2012.

[3] V. Bellotti and K. Edwards. Intelligibility and accountability: human considerations in context-aware systems. *Hum.-Comput. Interact.*, 16(2):193–212, 2001.

[4] S. Ben Allouch, J. A. Dijk, and O. Peters. The acceptance of domestic ambient intelligence appliances by prospective users. In *Proceedings of the 7th International Conference on Pervasive Computing*, pages 77–94, 2009.

[5] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon. Home automation in the wild: challenges and opportunities. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2115–2124, 2011.

[6] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA, 2000.

[7] J. R. Dabrowski and E. V. Munson. Is 100 milliseconds too fast? In *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '01, pages 317–318, 2001.

[8] S. Das, D. Cook, A. Battacharya, E. Heierman III, and T. Lin. The role of prediction algorithms in the MavHome smart home architecture. *Wireless Communications, IEEE*, 9(6):77–84, 2002.

[9] S. Davidoff, M. K. Lee, C. Yiu, J. Zimmerman, and A. K. Dey. Principles of smart home control. In *Proceedings of the 8th international conference on Ubiquitous Computing*, pages 19–34, 2006.

[10] T. Denceux. The cautious rule of combination for belief functions and some extensions. In *9th International Conference on Information Fusion*, pages 1 –8, 2006.

[11] B. Eggen, G. Hollemans, and R. van de Sluis. Exploring and enhancing the home experience. *Cognition, Technology & Work*, 5(1):44–54, 2003.

[12] K. Gopalratnam and D. J. Cook. Online sequential prediction via incremental parsing: The Active LeZi algorithm. *IEEE Intelligent Systems*, 22(1):52–58, 2007.

[13] D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In C. Nédellec and C. Rouveirol, editors, *Machine Learning: ECML-98*, pages 4–15. 1998.

[14] F. Li, K. Rasch, H.-L. Truong, R. Ayani, and S. Dustdar. Proactive service discovery in pervasive environments. In *Proceedings of the 7th International Conference on Pervasive Services*, pages 126–133, 2010.

[15] U. Maurer, A. Rowe, A. Smailagic, and D. P. Siewiorek. ewatch: A wearable sensor and notification platform. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, BSN '06, pages 142–145, 2006.

[16] S. Mckeever, J. Ye, L. Coyle, C. Bleakley, and S. Dobson. Activity recognition using temporal evidence theory. *J. Ambient Intell. Smart Environ.*, 2(3):253–269, 2010.

[17] S. Mennicken and E. Huang. Hacking the natural habitat: An in-the-wild study of smart homes, their development, and the people who live in them. In *Proceedings of the 10th international conference on Pervasive Computing*, pages 143–160. 2012.

[18] M. C. Mozer. Lessons from an adaptive home. In *Smart Environments*, pages 271–294. John Wiley & Sons, Inc., 2005.

[19] A. Pentland. Perceptual environments. In *Smart Environments*, pages 345–359. John Wiley & Sons, Inc., 2005.

[20] D. Randall. Living inside a smart home: A case study. In R. Harper, editor, *Inside the Smart Home*, pages 227–246. Springer London, 2003.

[21] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Automatic description of context-altering services through observational learning. In *Proceedings of the 10th international conference on Pervasive Computing*, pages 461–477, 2012.

[22] C. Röcker, M. Janse, N. Portolan, and N. Streitz. User requirements for intelligent home environments: a scenario-driven approach and empirical cross-cultural study. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 111–116. ACM, 2005.

[23] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[24] N. Stolzoff, E. Chuan-Fong Shih, and A. Venkatesh. The home of the future: an ethnographic study of new information technologies in the home. *Project NOAH*, 2000.

[25] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 1–9, 2008.

[26] T. L. M. van Kasteren, G. Englebienne, and B. J. A. Kröse. Transferring knowledge of activity recognition across sensor networks. In *Proceedings of the 8th international conference on Pervasive Computing*, pages 283–300, 2010.

[27] A. Woodruff, S. Augustin, and B. Foucault. Sabbath day home automation: "it's like mixing technology and religion". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 527–536, New York, NY, USA, 2007. ACM.

[28] R. Xu, G. Mei, Z. Ren, C. Kwan, J. Aube, C. Rochet, and V. Stanford. Speaker identification and speech recognition using phased arrays. In Y. Cai and J. Abascal, editors, *Ambient Intelligence in Everyday Life*, pages 227–238. Springer-Verlag, 2006.

[29] J. Ye, A. K. Clear, L. Coyle, and S. Dobson. On using temporal features to create more accurate human-activity classifiers. In *Proceedings of the 20th Irish conference on Artificial intelligence and cognitive science*, pages 273–282, 2010.