Chapter 8

Human Activity Recognition from Wireless Sensor Network Data: Benchmark and Software

T.L.M. van Kasteren, G. Englebienne, and B.J.A. Kröse[1]

Intelligent Systems Lab Amsterdam, Science Park 107, 1098 XG, Amsterdam, The Netherlands

Abstract

Although activity recognition is an active area of research no common benchmark for evaluating the performance of activity recognition methods exists. In this chapter we present the state of the art probabilistic models used in activity recognition and show their performance on several real world datasets. Our results can be used as a baseline for comparing the performance of other pattern recognition methods (both probabilistic and non-probabilistic). The datasets used in this chapter are made public, together with the source code of the probabilistic models used.

8.1 Introduction

Automatically recognizing human activities such as cooking, sleeping and bathing, in a home setting allows many applications in areas such as intelligent environments [1, 2] and healthcare [3–5]. Recent developments in sensing technology has led to wireless sensor networks which provide a non-intrusive, privacy friendly and easy to install solution to in-home monitoring. Sensors used are contact switches to measure open-close states of doors and cupboards; pressure mats to measure sitting on a couch or lying in bed; mercury contacts for movement of objects such as drawers; passive infrared sensors to detect motion in a specific area and float sensors to measure the toilet being flushed.

Recognizing activities from this sensor data requires us to solve the following issues. First, the start and end time of a performed activity is unknown. When activities are performed around the house there is no clear indication when one activity ends and another one starts. Second, there is ambiguity in the observed sensor data with respect to which activity is

[1]Corresponding author T.L.M. van Kasteren can be reached at tim0306@gmail.com.

taking place. For example, cooking and getting a drink both involve opening the fridge. Sensors can measure that the fridge is opened, but not which item is taken from the fridge. Third, activities can be performed in a large number of ways, making it difficult to create a general description of an activity. Fourth, observed sensor data is noisy. Noise can either be caused by a human making a mistake, for example opening a wrong cupboard, or by the sensor system which fails to transmit a sensor reading due to a network glitch.

These issues make activity recognition a very challenging task. Many different models have been proposed for activity recognition, but no common benchmark for evaluating the performance of these models exists. In this chapter we present the state of the art probabilistic models used in activity recognition and show their performance on several real world datasets. Our results can be used as a baseline for comparing the performance of other pattern recognition methods (both probabilistic and non-probabilistic). The datasets used in this chapter are made available to the public, together with the source code of the probabilistic models used.

The rest of this chapter is organized as follows. Section 8.2 presents the probabilistic models used for activity recognition. Section 8.3 presents the datasets and the sensor and annotation system used for recording the datasets. Section 8.4 presents our experiments and the results of these experiments. Section 8.5 discusses the outcome of our experiments. Section 8.6 discusses related work and how future work can possibly improve on the models discussed in this chapter.

8.2 Models

In this section we describe the probabilistic models that we use to provide a baseline recognition performance. First, we define the notation used for describing the models, then we present the naive Bayes model, hidden Markov model, hidden semi-Markov model and conditional random field. We present the model definitions and their inference and learning algorithms. The code of these models is publicly available for download from `http://sites.google.com/site/tim0306/`.

8.2.1 *Notation*

The models presented in this chapter require discretized time series data, this means data needs to be presented using timeslices of constant length. We discretize the data obtained from the sensors into T timeslices of length Δt. Each sensor corresponds to a single feature denoted as x_t^i, indicating the value of feature i at timeslice t, with $x_t^i \in \{0,1\}$. Feature values can either represent the raw values obtained directly from the sensor, or can be transformed according to a particular feature representation procedure. In the Section 8.4.2 we present various feature representations that can be used.

In a house with N sensors installed, we define a binary observation vector $\vec{x}_t = (x_t^1, x_t^2, \ldots, x_t^N)^T$. The activity at timeslice t is denoted with $y_t \in \{1, \ldots, Q\}$ for Q possible states. Our task is to find a mapping between a sequence of observations $\mathbf{x}_{1:T} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T\}$ and a sequence of activity labels $\mathbf{y}_{1:T} = \{y_1, y_2, \ldots, y_T\}$ for a total of T timeslices.

8.2.2 *Naive Bayes*

The naive Bayes model can be considered as one of the most simplistic probabilistic models. Unlike the other models in this chapter, the naive Bayes model assumes all data points are independently and identically distributed (i.i.d.), that is it does not take into account any temporal relations between datapoints. The model factorizes the joint probability over the datapoints as follows

$$p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\vec{x}_t \mid y_t) p(y_t). \tag{8.1}$$

The term $p(y_t)$ is the prior probability over an activity, that is, how probable an activity is to occur without taking any observation into account. The observation distribution $p(\vec{x}_t \mid y_t)$ represents the probability that the activity y_t would generate observation vector $\vec{x}_t$. If we were to model the distribution of the observation vector exactly, we would need to consider all possible combinations of values in the vector's dimensions. This would require 2^N parameters per activity, with N being the number of sensors used. This easily results in a large number of parameters, even for small numbers of N. Instead, we apply the naive Bayes assumption, which means we model each sensor reading separately, requiring only N parameters for each activity. The observation distribution therefore factorizes as

$$p(\vec{x}_t \mid y_t = i) = \prod_{n=1}^{N} p(x_t^n \mid y_t = i) \tag{8.2}$$

where each sensor observation is modeled as an independent Bernoulli distribution, given by $p(x_t^n \mid y_t = i) = \mu_{ni}^{x_t^n} (1 - \mu_{ni})^{1 - x_t^n}$.

Using naive Bayes is a very strong model assumption, which most likely does not represent the true distribution of the data (i.e. it is very likely that two sensors are dependent on each other with respect to a particular activity). However, naive Bayes has been shown to give very good results in many domains, despite its overly simplistic assumption [6].

8.2.3 *Hidden Markov model*

Hidden Markov models (HMM) apply the Markov assumption to model the temporal correlation between consecutive timeslices; it relies on the following two independence assumptions.

- The hidden variable at time t, namely y_t, depends only on the previous hidden variable y_{t-1} (*first order Markov assumption* [7]).
- The observable variable at time t, namely $\vec{x}_t$, depends only on the hidden variable y_t at that time slice.

The joint probability therefore factorizes as follows

$$p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(\vec{x}_t \mid y_t) p(y_t \mid y_{t-1})$$

where we have used $p(y_1 \mid y_0) = p(y_1)$ for the sake of notational simplicity. For the observation model $p(\vec{x}_t \mid y_t)$ we use the same assumptions as we did with the naive Bayes model. The transition probability distribution $p(y_t \mid y_{t-1})$ represents the probability of going from one state to the next. This is modeled as Q multinomial distributions, one for each activity. Individual transition probabilities are denoted as $p(y_t = j \mid y_{t-1} = i) \equiv a_{ij}$.
For further reading about hidden Markov models we refer the reader to a tutorial by Rabiner [7]. The tutorial covers many different aspects of HMMs and explains their application in the task of speech recognition.

8.2.4 *Hidden semi-Markov model*

Semi-Markov models relax the Markov assumption by explicitly modeling the duration of an activity. In conventional Markov models, such as the HMM, the duration of an activity is modeled implicitly by means of self-transitions of states, and this entails a number of important limitations [7, 8]. Hidden semi-Markov models explicitly model the duration of a state and therefore any distribution can be used to model the duration.
To model the duration we introduce an additional variable d_t, which represents the remaining duration of state y_t. The value of d_t is decreased by one at each timestep, and as long as the value is larger than zero the model continues to stay in state y_t. When the value of d_t reaches zero a transition to a new state is made and the duration of the new state is obtained from the duration distribution. The joint probability of the HSMM is factorized as

$$p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T}, \mathbf{d}_{1:T}) = \prod_{t=1}^{T} p(\vec{x}_t \mid y_t) p(y_t \mid y_{t-1}, d_{t-1}) p(d_t \mid d_{t-1}, y_t)$$

where we have used $p(y_1 \mid y_0, d_0) = p(y_1)$ and $p(d_1 \mid d_0, y_1) = p(d_1 \mid y_1)$ for the sake of notational simplicity. The observation model $p(\vec{x}_t \mid y_t)$ is the same as with the naive Bayes model and the HMM. Transitions between states are modeled by the factor $p(y_t \mid y_{t-1}, d_{t-1})$, defined as:

$$p(y_t = i \mid y_{t-1} = j, d_{t-1}) = \begin{cases} \delta(i,j) & \text{if } d_{t-1} > 0 \text{ (remain in same state)} \\ a_{ij} & \text{if } d_{t-1} == 0 \text{ (transition)} \end{cases} \tag{8.3}$$

where $\delta(i, j)$ is the Kronecker delta function, giving 1 if $i = j$ and 0 otherwise. The parameter a_{ij} is part of a multinomial distribution. Note, some model definitions of semi-Markov models force the value of a_{ii} to zero, effectively disabling self-transitions. However, in this work we do allow self-transitions. The use of self-transitions in semi-Markov models allows convolutions of duration distributions and makes it possible that separate instances of activities immediately follow each other.
State durations are modeled by the term $p(d_t \mid d_{t-1}, y_t)$, defined as

$$p(d_t \mid d_{t-1}, y_t = k) = \begin{cases} p_k(d_t + 1) & \text{if } d_{t-1} == 0 \text{ (generate new duration)} \\ \delta(d_t, d_{t-1} - 1) & \text{if } d_{t-1} > 0 \text{ (count down duration)} \end{cases} \tag{8.4}$$

where $p_k(d_t + 1)$ is the distribution used for modeling the state duration. In this work we use a histogram using 5 bins to represent the duration distribution. Note, d_t is defined as the *remaining* duration of a state, therefore the actual duration, when generating a new duration, is $d_t + 1$ timeslices.
For further details on hidden semi-Markov models we refer the reader to a tutorial from Murphy [8]. It provides a unified framework for HSMMs and describes possible variations to the HSMM described above.

8.2.5 *Conditional random fields*

The naive Bayes model, the HMM and HSMM are generative models in which parameters are learned by maximizing the joint probability $p(\mathbf{y}_{1:T}, \mathbf{x}_{1:T})$. Conditional random fields are discriminative models. Rather than modeling the full joint probability, discriminative models model the conditional probability $p(\mathbf{y}_{1:T} \mid \mathbf{x}_{1:T})$. Conditional random fields represent a general class of discriminative models. A CRF using the first-order Markov assumption is called a linear-chain CRF and most closely resembles the HMM in terms of structure.
We define the linear-chain CRF as a discriminative analog of the previously defined HMM, so that the same independence assumptions hold. These assumptions are represented in the

model using feature functions, so that the conditional distribution is defined as

$$p(\mathbf{y}_{1:T} \mid \mathbf{x}_{1:T}) = \frac{1}{Z(\mathbf{x}_{1:T})} \prod_{t=1}^{T} \exp \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \vec{x}_t)$$

where K is the number of feature functions used to parameterize the distribution, λ_k is a weight parameter and $f_k(y_t, y_{t-1}, \vec{x}_t)$ a feature function. The product of the parameters and the feature function $\lambda_k f_k(y_t, y_{t-1}, \vec{x}_t)$ is called an energy function, and the exponential representation of that term is called a potential function [9]. Unlike the factors in the joint distribution of HMMs, the potential functions do not have a specific probabilistic interpretation and can take any positive real value.

The partition function $Z(\mathbf{x}_{1:T})$ is a normalization term, that ensures that the distribution sums up to one and obtains a probabilistic interpretation [10]. It is calculated by summing over all possible state sequences

$$Z(\mathbf{x}_{1:T}) = \sum_{\mathbf{y}} \left\{ \prod_{t=1}^{T} \exp \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \vec{x}_t) \right\}. \tag{8.5}$$

The feature functions $f_k(y_t, y_{t-1}, \vec{x}_t)$ for the CRF can be grouped as observation feature functions and transition feature functions. In defining the feature functions we use a multi-dimensional index to simplify notation, rather than the one-dimensional index used above. This gives the following feature function definitions:

Observation: $f_{vin}(x_t^n, y_t) = \delta(y_t, i) \cdot \delta(x_t^n, v)$.
Transition: $f_{ij}(y_t, y_{t-1}) = \delta(y_t, i) \cdot \delta(y_{t-1}, j)$.

A thorough introduction to CRFs is presented by Sutton *et al.* [10]. It includes a general comparison between discriminative and generative models, and it explains how HMMs and CRFs differ from each other.

8.2.6 *Inference*

The inference problem for these models consists of finding the single best state sequence that maximizes $p(\mathbf{y}_{1:T} \mid \mathbf{x}_{1:T})$. For the naive Bayes model we can simply calculate the probability for all state and observation pairs and select the state with the maximum probability at each timeslice.

In the other models we need to consider every possible sequence of states, which grows exponentially with the length of the sequence. We use the Viterbi algorithm, which uses dynamic programming, to find the best state sequence efficiently. By using Viterbi we can discard a number of paths at each time step, for HMMs and CRFs this results in a computational complexity of $O(TQ^2)$ for the entire sequence, where T is the total number of timeslices and Q the number of states [7]. For HSMMs we also need to iterate over all possible durations and therefore the computational complexity is $O(DTQ^2)$, where D is the maximum possible duration [11].

8.2.7 *Learning*

For learning the model parameters we assume that we have fully labeled data available. The parameters of the naive Bayes model, HMM and HSMM can be learned in closed form. In the case of the CRF, parameters have to be estimated iteratively using a numerical method.

8.2.7.1 *Closed Form Solution*

We learn the model parameters using maximum likelihood. Finding the joint maximum likelihood parameters is equivalent to finding the maximum likelihood parameters of each of the factors that make up the joint probability.
The observation probability $p(x_t^n \mid y = i)$ is a Bernoulli distribution whose maximum likelihood parameter estimation is given by

$$\mu_{ni} = \frac{\sum_{t=1}^{T} x_t^n \delta(y_t, i)}{\sum_{t=1}^{T} \delta(y_t, i)} \tag{8.6}$$

where T is the total number of time slices.
The transition probability $p(y_t = j \mid y_{t-1} = i)$ is a multinomial distribution whose parameters are calculated by

$$a_{ij} = \frac{\sum_{t=2}^{T} \delta(y_t, j)\delta(y_{t-1}, i)}{\sum_{t=2}^{T} \delta(y_{t-1}, j)} \tag{8.7}$$

where T is equal to the number of time slices.

8.2.7.2 *Numerical Optimization*

The parameters $\theta = \{\lambda_1, \ldots, \lambda_K\}$ of CRFs are learned by maximizing the conditional log likelihood $l(\theta) = \log p(\mathbf{y}_{1:T} \mid \mathbf{x}_{1:T}, \theta)$ given by

$$l(\theta) = \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \vec{x}_t) - \log Z(\mathbf{x}_{1:T}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}$$

where the final term is a regularization term, penalizing large values of λ to prevent overfitting. The constant σ is set beforehand and determines the strength of the penalization [10]. The function $l(\theta)$ is concave, which follows from the convexity of $\log Z(\mathbf{x}_{1:T})$ [10]. A useful property of convex functions in parameter learning is that any local optimum is also a global optimum. Quasi-Newton methods such as BFGS have been shown to be suitable for estimating the parameters of CRFs [12, 13]. These methods approximate the Hessian, the matrix of second derivatives, by analyzing successive gradient vectors. To analyze the gradient vectors, the partial derivative of $l(\theta)$ with respect to λ_i is needed, it is given by

$$\frac{\partial l}{\partial \lambda_i} = -\frac{\lambda_i}{\sigma^2} + \sum_{t=1}^{T} f_i(y_t, y_{t-1}, \vec{x}_t) - \sum_{t=1}^{T} \sum_{y_t, y_{t-1}} p(y_t, y_{t-1} \mid \vec{x}_t) f_i(y_t, y_{t-1}, \vec{x}_t).$$

Because the size of the Hessian is quadratic in the number of parameters, storing the full Hessian is memory intensive. We therefore use a limited-memory version of BFGS [14, 15].

8.3 Datasets

We have recorded three datasets consisting of several weeks of data recorded in a real world setting. In this section we describe the sensor and annotation system that we used to record and annotate our datasets and we give a description of the houses and the datasets recorded in them.

8.3.1 *Sensors Used*

In this work we use wireless sensor networks to observe the behavior of inhabitants inside their homes. After considering different commercially available wireless network kits, we selected the RFM DM 1810 (fig. 8.1(a)), because it comes with a very rich and well documented API and the standard firmware includes an energy efficient network protocol. An energy saving protocol efficiently handles the wireless communication, resulting in a long battery life. The node can reach a data transmission rate of 4.8 kb/s, which is enough for the binary sensor data that we need to collect. The kit comes with a base station which is attached to a PC through USB. A new sensor node can be easily added to the network by a simple pairing procedure, which involves pressing a button on both the base station and the new node.

The RFM wireless network node has an analog and digital input. It sends an event when the state of the digital input changes or when the analog input crosses some threshold. We equipped nodes with various kinds of sensors: reed switches to measure whether doors and cupboards are open or closed; pressure mats to measure sitting on a couch or lying in bed; mercury contacts to detect the movement of objects (e.g. drawers); passive infrared (PIR) to detect motion in a specific area; float sensors to measure the toilet being flushed. All

(a) Network node

(b) Bluetooth headset

Fig. 8.1 Wireless sensor network node used for sensing the environment and bluetooth headset used for annotation.

sensors give binary output.

8.3.2 *Annotation*

Annotation was performed using either a handwritten activity diary or a bluetooth headset combined with speech recognition software. The starting and end point of an activity were annotated in both cases.

8.3.2.1 *Handwritten diary*

In one of our datasets we annotated the activities using a handwritten diary. Several sheets of papers were distributed throughout the house at locations where activities are typically performed. The subject would annotate the start and end time of activities by reading the time of his watch and writing it down on one of the sheets. The advantage of this method is that it is very easy to install and use by the subject. The disadvantage is that it is time consuming to process the annotated data (i.e. enter the handwritten sheets into a computer) and that the time stamps from the watch might slightly differ from the timestamps recorded by the computer saving the sensor data.

Table 8.1 Details of recorded datasets.

	House A	House B	House C
Age	26	28	57
Gender	Male	Male	Male
Setting	Apartment	Apartment	House
Rooms	3	2	6
Duration	25 days	14 days	19 days
Sensors	14	23	21
Activities	10	13	16
Annotation	Bluetooth	Diary	Bluetooth

8.3.2.2 *Bluetooth method*

In the bluetooth annotation method a set of predefined commands is used to record the start and end time of activities. We used the Jabra BT250v bluetooth headset (fig. 8.1(b)) for annotation. It has a range up to 10 meters and battery power for 300 hours standby or 10 hours active talking. This is more than enough for a full day of annotation, the headset was recharged at night. The headset contains a button which we used to trigger the software to add a new annotation entry.

The software for storing the annotation is written in C and combines elements of the bluetooth API with the Microsoft Speech API[1]. The bluetooth API was needed to catch the event of the headset button being pressed and should work with any bluetooth dongle and headset that uses the Widcomm[2] bluetooth stack.

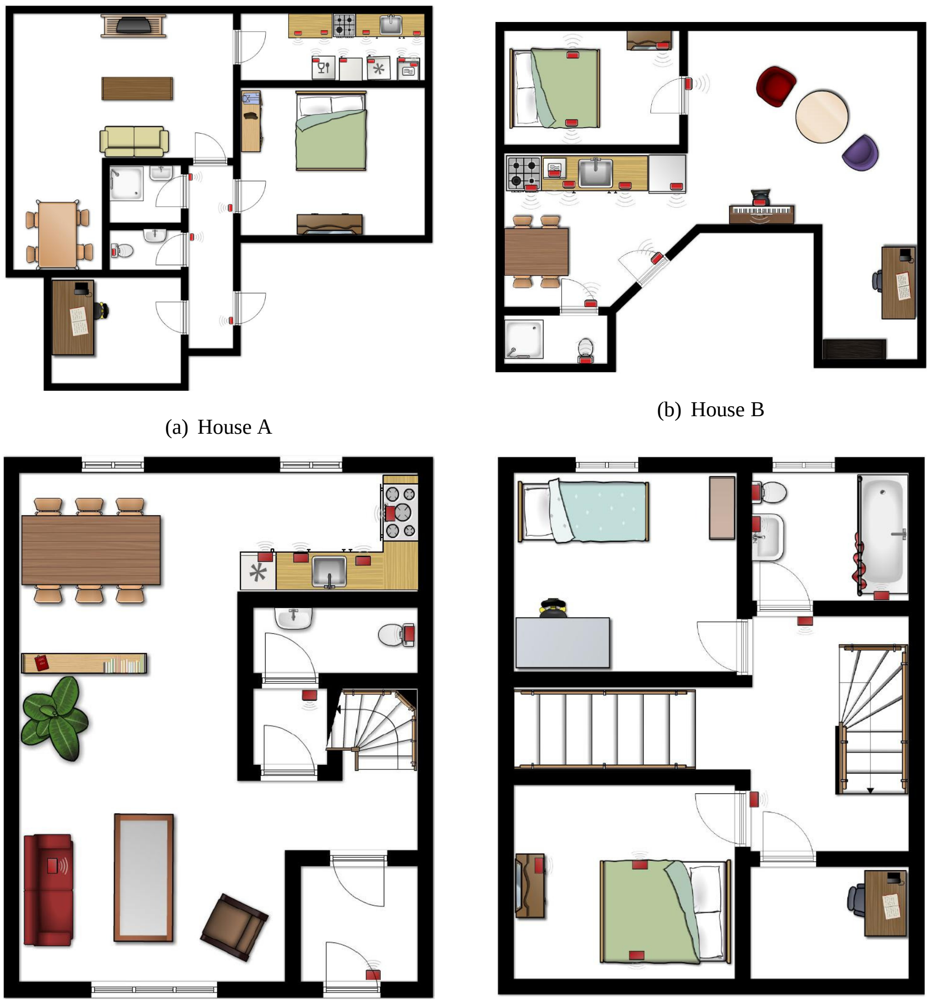
The Microsoft Speech API provides an easy way to use both speech recognition and text to speech. When the headset button is pressed the speech recognition engine starts listening for commands it can recognize. We created our own speech grammar, which contains a limited combination of commands the recognition engine could expect. By using very distinctive commands such as 'begin use toilet' and 'begin take shower', the recognition engine had multiple words by which it could distinguish different commands. This resulted in near perfect recognition results during annotation. The recognized sentence is outputted using the text-to-speech engine. Any errors that do occur can be immediately corrected using a 'correct last' command.

Because annotation is provided by the user on the spot this method is very efficient and accurate. The use of a headset together with speech recognition results in very little interference while performing activities. The resulting annotation data requires very little post

[1]For details about the Microsoft Speech API see: http://www.microsoft.com/speech/

[2]For details about the Widcomm stack see: http://www.broadcom.com/products/bluetooth_sdk.php

processing and is timestamped using the same clock (the internal clock of the computer) that is used to timestamp the sensor data.



(a) House A

(b) House B

(c) House C, First floor

(d) House C, Second floor

Fig. 8.2 Floorplan of houses A, B and C, the red boxes represent wireless sensor nodes.

8.3.3 *Houses*

A total of three datasets was recorded in three different houses. Details about the datasets can be found in Table 8.1. Floorplans for each of the houses, indicating the locations of the sensors, can be found in Figure 8.2.

Table 8.2 Confusion Matrix

	Inferred			
True	1	2	3	
1	TP_1	ε_{12}	ε_{13}	TT_1
2	ε_{21}	TP_2	ε_{23}	TT_2
3	ε_{31}	ε_{32}	TP_3	TT_3
	TI_1	TI_2	TI_3	$Total$

Confusion Matrix showing the true positives (TP), total of true labels (TT) and total of inferred labels (TI) for each class.

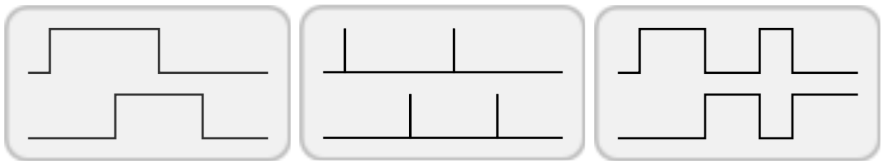
These datasets are publicly available for download from `http://sites.google.com/site/tim0306/`.

8.4 Experiments

Our experiments provide a baseline recognition performance in using probabilistic models with our datasets. We run two experiments, in the first experiment we discretize our data using various timeslice lengths and in the second experiment we compare the performance of the different models and a number of feature representations.

8.4.1 *Experimental Setup*

We split our data into a test and training set using a 'leave one day out' approach. In this approach, one full day of sensor readings is used for testing and the remaining days are used for training. We cycle over all the days and report the average performance measure. We evaluate the performance of our models using precision, recall and F-measure. These measures can be calculated using the confusion matrix shown in Table 8.2. The diagonal of the matrix contains the true positives (TP), while the sum of a row gives us the total of true labels (TT) and the sum of a column gives us the total of inferred labels (TI). We calculate the precision and recall for each class separately and then take the average over all classes. It is important to use these particular measures because we are dealing with unbalanced datasets. In unbalanced datasets some classes appear much more frequent than other classes. Our measure takes the average precision and recall over all classes and therefore considers the correct classification of each class equally important. To further illustrate the importance of these measures we also include the accuracy in our results. The accuracy represents the percentage of correctly classified timeslices, therefore more frequently occurring classes have a larger weight in this measure.



(a) Raw (b) Changepoint (c) Last-fired

Fig. 8.3 Differen feature representations.

$$\text{Precision} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TI_i} \tag{8.8}$$

$$\text{Recall} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TT_i} \tag{8.9}$$

$$\text{F-Measure} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{8.10}$$

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} TP_i}{Total} \tag{8.11}$$

8.4.2 *Feature Representation*

The raw data obtained from the sensors can either be used directly, or be transformed into a different representation form. We experiment with three different feature representations:

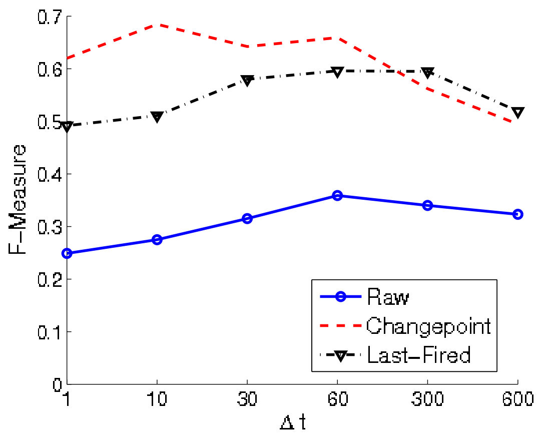
Raw: The raw sensor representation uses the sensor data directly as it was received from the sensors. It gives a 1 when the sensor is firing and a 0 otherwise (Fig. 8.3(a)).

Changepoint: The change point representation indicates when a sensor event takes place. That is, it indicates when a sensor changes value. More formally, it gives a 1 when a sensor changes state (i.e. goes from zero to one or vice versa) and a 0 otherwise (Fig. 8.3(b)).

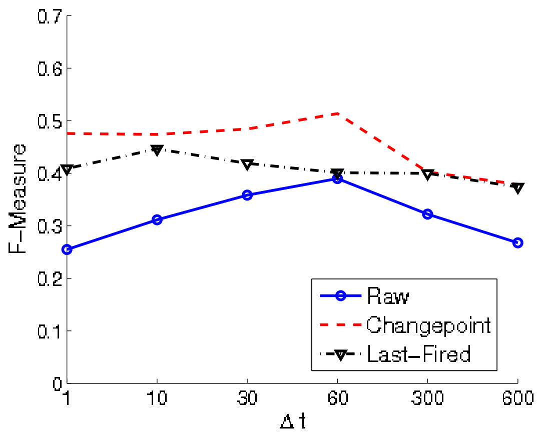
Last-fired: The last-fired sensor representation indicates which sensor fired last. The sensor that changed state last continues to give 1 and changes to 0 when another sensor changes state (Fig. 8.3(c)).

8.4.3 *Experiment 1: Timeslice Length*

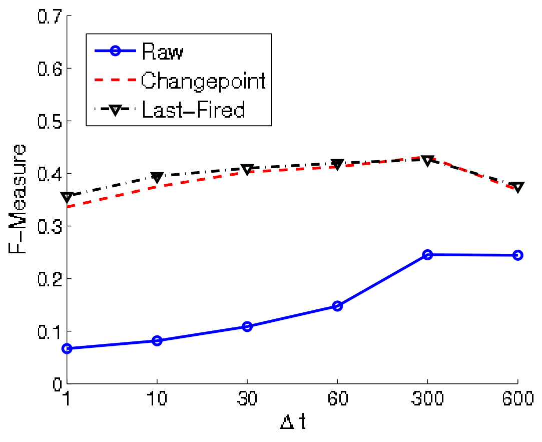
Here we present our findings for determining the ideal timeslice length for discretizing the sensor data. Experiments were run using the HMM. We experimented using all the feature



(a) House A



(b) House B



(c) House C

Fig. 8.4 F-Measure performance of the HMM for the three houses using different timeslice length to discretize the data. Δt is given in seconds.

representations, to rule out any bias towards any of the representations.
The sensor data and the ground truth activity labels are discretized using the same timeslice length. During this discretization process it is possible that two or more activities occur within a single timeslice. For example, an activity might end somewhere halfway of the timeslice and another activity can start immediately after. In this case we let the timeslice represent the activity that takes up most of the timeslice. However, a consequence of this is that the discretized ground truth differs from the actual ground truth. To express the magnitude of this difference, we introduce the discretization error. The discretization error represents the percentage of incorrect labels in the discretized ground truth, as a result of the discretization process.
The discretized ground truth labels are used to learn the model parameters. However, if we were to use the discretized ground truth for calculating the performance measures of the model as well, our measure would not take into account the discretization error. Therefore, we evaluate the performance of our models using the actual ground truth, which was obtained with a one second accuracy.
The F-measure values for the various timeslice lengths are plotted in Figure 8.4. We see that within a single house no timeslice length achieves consistent maximum performance for all feature representations. Furthermore, we see that across all three houses no timeslice length consistently outperforms the others for a particular feature representation. Overall we see that the timeslice lengths of $\Delta t = 30$ seconds and $\Delta t = 60$ seconds give a good performance.
Table 8.3 shows the discretization error for this experiment. As expected, small timeslice lengths result in a small discretization error. We see that a significant error occurs for timeslice length of $\Delta t = 300$ seconds and $\Delta t = 600$ seconds.

Table 8.3 Discretization Error in percentages.

Length	**House A**	**House B**	**House C**
$\Delta t =$ 1 **s**	0.0	0.0	0.0
$\Delta t =$ 10 **s**	0.2	0.2	0.2
$\Delta t =$ 30 **s**	0.6	0.6	0.9
$\Delta t =$ 60 **s**	1.3	1.1	1.7
$\Delta t =$ 300 **s**	5.9	4.0	8.1
$\Delta t =$ 600 **s**	10.6	17.4	13.7

8.4.4 *Experiment 2: Feature Representations and Models*

This experiment shows the performance of the models for the different feature representations. Data was discretized using the timeslice length of $\Delta t = 60$ seconds. The discretized ground truth was used to calculate the performance measures. The measures and their standard deviation for house A are shown in Table 8.4, for house B in Table 8.5 and for house C in Table 8.6.

In comparing the performance of the different feature representations we see that the raw feature representation performs by far the worst. The change and last representation both perform well, although the change representation manages to outperform the last representation significantly in a number of cases. In terms of model performances, we see that the HMM generally outperforms the naive Bayes model and that the HSMM generally outperforms the HMM. CRFs manage to outperform the other models in some cases, but the HSMM using the changepoint representation consistently achieves a very high F-measure (the highest in two of the three datasets). When looking at the precision and recall we see that the generative models (naive Bayes, HMM and HSMM) generally score higher in terms of recall, while the discriminative model (CRF) generally outperforms the other models in terms of precision. Finally, in terms of accuracy CRFs have a very high score and outperform the other models in almost all cases.

8.5 Discussion

The results from experiment 1 show that the choice of timeslice length can strongly influence the recognition performance of a model. A timeslice length of $\Delta t = 30$ seconds and $\Delta t = 60$ seconds gives a good performance and results in a small discretization error. This means these timeslice lengths are long enough to be discriminative and short enough to provide high accuracy labeling results.

Experiment 2 shows that the choice of feature representation strongly affects the recognition performance of the model. Feature representations are used to restructure the data with the intention of making the classification problem easier to solve. The raw feature representation does not give good results because the binary state of a sensor does not provide good information about the activity that is currently performed. For example, people tend to leave doors open after they walk through them. The raw representation can indicate that a shower door is open long after the person finished using the shower. Because the shower door can practically be open or closed when someone is not taking a shower it is not a good

Table 8.4 Results of experiment 2, House A

Model	Feature	**Precision**	**Recall**	**F-Measure**	**Accuracy**
NB	**Raw**	48.3 ± 17.7	42.6 ± 16.6	45.1 ± 16.9	77.1 ± 20.8
	Change	52.7 ± 17.5	43.2 ± 18.0	47.1 ± 17.2	55.9 ± 18.8
	Last	67.3 ± 17.2	64.8 ± 14.6	65.8 ± 15.5	95.3 ± 2.8
HMM	**Raw**	37.9 ± 19.8	45.5 ± 19.5	41.0 ± 19.5	59.1 ± 28.7
	Change	70.3 ± 16.0	74.3 ± 13.3	72.0 ± 14.2	92.3 ± 5.8
	Last	54.6 ± 17.0	69.5 ± 12.7	60.8 ± 14.9	89.5 ± 8.4
HSMM	**Raw**	39.5 ± 18.9	48.5 ± 19.5	43.2 ± 19.1	59.5 ± 29.0
	Change	70.5 ± 16.0	75.0 ± 12.1	72.4 ± 13.7	91.8 ± 5.9
	Last	60.2 ± 15.4	73.8 ± 12.5	66.0 ± 13.7	91.0 ± 7.2
CRF	**Raw**	59.2 ± 18.3	56.1 ± 17.3	57.2 ± 17.3	89.8 ± 8.5
	Change	73.5 ± 16.6	68.0 ± 16.0	70.4 ± 15.9	91.4 ± 5.6
	Last	66.2 ± 15.8	65.8 ± 14.0	65.9 ± 14.6	96.4 ± 2.4

Different feature representations using naive Bayes (NB), HMM, HSMMs and CRFs.

Table 8.5 Results of experiment 2, House B

Model	Feature	**Precision**	**Recall**	**F-Measure**	**Accuracy**
NB	**Raw**	33.6 ± 10.9	32.5 ± 8.4	32.4 ± 8.0	80.4 ± 18.0
	Change	40.9 ± 7.2	38.9 ± 5.7	39.5 ± 5.0	67.8 ± 18.6
	Last	43.7 ± 8.7	44.6 ± 7.2	43.3 ± 4.8	86.2 ± 13.8
HMM	**Raw**	38.8 ± 14.7	44.7 ± 13.4	40.7 ± 12.4	63.2 ± 24.7
	Change	48.2 ± 17.2	63.1 ± 14.1	53.6 ± 16.5	81.0 ± 14.2
	Last	38.5 ± 15.8	46.6 ± 19.5	41.8 ± 17.1	48.4 ± 26.0
HSMM	**Raw**	37.4 ± 16.9	44.6 ± 14.3	39.9 ± 14.3	63.8 ± 24.2
	Change	49.8 ± 15.8	65.2 ± 13.4	55.7 ± 14.6	82.3 ± 13.5
	Last	40.8 ± 11.6	53.3 ± 10.9	45.8 ± 11.2	67.1 ± 24.8
CRF	**Raw**	35.7 ± 15.2	40.6 ± 12.0	37.5 ± 13.7	78.0 ± 25.9
	Change	48.3 ± 8.3	51.5 ± 8.5	49.7 ± 7.9	92.9 ± 6.2
	Last	46.0 ± 12.5	47.8 ± 12.1	46.6 ± 12.0	89.2 ± 13.9

Different feature representations using naive Bayes (NB), HMM, HSMMs and CRFs.

feature to base classification on. In the changepoint representation this issue is resolved because this feature represents when a sensor changes state and thus indicates when an object is used. The last representation gives an indication of the location of an inhabitant. The idea is that as long as people remain in the same location they do not trigger any sensors. As soon as people start moving around the house they are likely to trigger sensors, which will provide an update of their current location. This representation works best with a large number of sensors installed, so that the chance of triggering a sensor when moving around

Table 8.6 Results of experiment 2, House C

Model	Feature	**Precision**	**Recall**	**F-Measure**	**Accuracy**
NB	**Raw**	19.6 ± 11.4	16.8 ± 7.5	17.8 ± 9.1	46.5 ± 22.6
	Change	39.9 ± 6.9	30.8 ± 4.8	34.5 ± 4.6	57.6 ± 15.4
	Last	40.5 ± 7.4	46.4 ± 14.8	42.3 ± 6.8	87.0 ± 12.2
HMM	**Raw**	15.2 ± 9.2	17.2 ± 9.3	15.7 ± 8.8	26.5 ± 22.7
	Change	41.4 ± 8.8	50.0 ± 11.4	44.9 ± 8.8	77.2 ± 14.6
	Last	40.7 ± 9.7	53.7 ± 16.2	45.9 ± 11.2	83.9 ± 13.9
HSMM	**Raw**	15.6 ± 9.2	20.4 ± 10.9	17.3 ± 9.6	31.2 ± 24.6
	Change	43.8 ± 10.0	52.3 ± 12.8	47.4 ± 10.5	77.5 ± 15.3
	Last	42.5 ± 10.8	56.0 ± 15.4	47.9 ± 11.3	84.5 ± 13.2
CRF	**Raw**	17.8 ± 22.1	21.8 ± 20.9	19.0 ± 21.8	46.3 ± 25.5
	Change	36.7 ± 18.0	39.6 ± 17.4	38.0 ± 17.6	82.2 ± 13.9
	Last	37.7 ± 17.1	40.4 ± 16.0	38.9 ± 16.5	89.7 ± 8.4

Different feature representations using naive Bayes (NB), HMM, HSMMs and CRFs.

the house is large.

The probabilistic models we used in the experiments differ in complexity. Naive Bayes is the simplest model, HMMs add temporal relations and HSMMs add the explicit modeling of state durations. CRFs are basically similar to HMMs, but the parameters are learned using a different optimization criterion. The results show that an increase in model complexity generally results in an increase of performance as well. However, this is not always the case, increasing the model complexity involves making certain model assumptions. For example, to model the temporal relations in HMMs we use the first order Markov assumption. This assumption might be too strong, it is very well possible that in a complex activity such as cooking dinner there are dependencies between timeslices at the start of the cooking activity and the end of the cooking activity. On the other hand, explicitly modeling these long term dependencies results in a large number of model parameters and therefore requires a large amount of training data to accurately learn those parameters. The best performing model therefore needs to be designed with a careful trade-off between complexity and amount of training data needed.

Precision and recall gives us further insight into the quality of the classification. Precision tells us which percentage of inferred labels was correctly classified, while recall tells us which percentage of true labels was correctly classified. CRFs score higher on precision because during learning the set of parameters is chosen that favors the correct classification of frequently occurring activities (activities that take up many timeslices). As a result infrequent classes are only classified as such when there is little or no confusion with the frequent class. This results in a high precision for the infrequent classes. The recall on the other hand is low because many infrequent classes are incorrectly classified as frequent. In the case of the generative models (naive Bayes, HMM and HSMM) model parameters are learned without consideration of the class frequency. This results in more confusion among the infrequent classes, but also results in more true positives for the infrequent classes and therefore a higher recall.

Finally, in terms of the accuracy measure CRFs generally score the highest, because of its preference to correctly classify frequent classes. In the accuracy measure frequent classes have a higher weight. However, because in most applications the correct classification of each class is equally important, we suggest the use of our precision, recall and F-measure. If for some application it is clear with which weight of importance the activities should be measured, it is possible to use those weights in the calculation of the precision and recall accordingly.

8.6 Related and Future work

The probabilistic models discussed in this chapter represent the state of the art models used in activity recognition. Tapia *et al.* used the naive Bayes model in combination with the raw feature representation on two real world datasets recorded using a wireless sensor network [16]. HMMs were used in work by Patterson *et al.* and were applied to data obtained from a wearable RFID reader in a house where many objects are equipped with RFID tags [17]. Wireless sensor network data is more ambiguous than RFID data with respect to activities. For example, when using RFID it is possible to sense which object is used, while with wireless sensor networks only the cupboard that contains the object can be sensed. Because there are usually multiple objects in a cupboard the resulting data is ambiguous with respect to the object used. In work by van Kasteren *et al.* the performance of HMMs and CRFs in activity recognition was compared on a realworld dataset recorded using a wireless sensor network [18]. Duong *et al.* compared the performance of HSMMs and HMMs in activity recognition using a laboratory setup in which four cameras captured the location of a person in a kitchen setup [19]. One type of model that we have not included in our comparison are hierarchical models. They have been successfully applied to activity recognition from video data [20], in an office environment using cameras, microphones and keyboard input [21] and on data obtained from a wearable sensing system [22].

The models and datasets presented in this chapter provide a good benchmark for comparing future models. Possible extensions for future work include different feature representations. In the representations presented in this chapter each feature corresponds to a single sensor. However, a representation which captures properties between sensors (e.g. sensor A fired after sensor B) might further improve the recognition performance. Another extension can be the use of a different observation model. In this chapter we used the naive Bayes assumption in all models to minimize the number of parameters needed. Structure learning can be used to learn which dependencies between sensors are most important for recognizing activities, therefore automatically finding a nice trade-off between the number of parameters and the complexity of the model. Finally, more elaborate dynamic Bayesian networks can be used in which variables relevant to activity recognition are modeled to increase recognition performance.

A clear downside of the use of probabilistic models is the need for carefully labeled training data to learn the model parameters. Because human activities are quite well understood and wireless sensor networks provide easily interpretable data, there is a large body of work in the logic community focusing on activity recognition. The use of a logic system allows the inclusion of hand crafted rules and therefore limits the need for training data. Many logic systems for activity recognition have been proposed using various kinds of formal logic, such as event calculus [23], lattice theory and action description logic [24] and temporal logic [25]. The benchmark datasets in this chapter can be used to compare the performance of such logic systems to the performance of state of the art probabilistic models in activity recognition.

8.7 Conclusion

This chapter presents several real world datasets for activity recognition that were recorded using a wireless sensor network. We introduced an evaluation method using precision, recall and F-measure, which provides insight into the quality of the classification and takes unbalanced datasets into account. The performance of state of the art probabilistic models was evaluated and these results provide a baseline for comparison with other pattern recognition methods. The code and datasets needed for repeating and extending on these experiments are publicly available to the community.

References

[1] J. C. Augusto and C. D. Nugent, Eds. *Designing Smart Homes, The Role of Artificial Intelligence*, vol. 4008, *Lecture Notes in Computer Science*, (2006). Springer. ISBN 3-540-35994-X.

[2] D. J. Cook and S. K. Das, *Smart Environments: Technology, Protocols and Applications*. (Wiley-Interscience, 2004).

[3] G. Abowd, A. Bobick, I. Essa, E. Mynatt, and W. Rogers. The aware home: Developing technologies for successful aging. In *Proceedings of AAAI Workshop and Automation as a Care Giver*, (2002).

[4] R. Suzuki, M. Ogawa, S. Otake, T. Izutsu, Y. Tobimatsu, S.-I. Izumi, and T. Iwaya, Analysis of activities of daily living in elderly people living alone, *Telemedicine*. **10**, 260, (2004).

[5] D. H. Wilson. *Assistive Intelligent Environments for Automatic Health Monitoring*. PhD thesis, Carnegie Mellon University, (2005).

[6] I. Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, pp. 41–46, (2001).

[7] L. R. Rabiner, A tutorial on hidden markov models and selected applications in speech recognition, *Proceedings of the IEEE*. **77**(2), 257–286, (1989). URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=18626`.

[8] K. P. Murphy. Hidden semi-markov models (hsmms). Technical report, University of British Columbia, (2002).

[9] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. (Springer, August 2006). ISBN 0387310738. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20&path=ASIN/0387310738`.

[10] C. Sutton and A. McCallum, *Introduction to Statistical Relational Learning*, chapter 1: An introduction to Conditional Random Fields for Relational Learning, p. (Available online). MIT Press, (2006).

[11] T. van Kasteren, G. Englebienne, and B. Kröse, Activity recognition using semi-markov models on real world smart home datasets, *Journal of Ambient Intelligence and Smart Environments*. **2** (3), 311–325, (2010).

[12] H. Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, (2002). URL `http://citeseer.ist.psu.edu/wallach02efficient.html`.

[13] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pp. 134–141, Morristown, NJ, USA, (2003). Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1073445.1073473.

[14] R. H. Byrd, J. Nocedal, and R. B. Schnabel, Representations of quasi-newton matrices and their use in limited memory methods, *Math. Program.* **63**(2), 129–156, (1994). ISSN 0025-5610.

[15] D. C. Liu and J. Nocedal, On the limited memory bfgs method for large scale optimization, *Math. Program.* **45**(3), 503–528, (1989). ISSN 0025-5610. doi: http://dx.doi.org/10.1007/BF01589116.

[16] E. M. Tapia, S. S. Intille, and K. Larson. Activity recognition in the home using simple and ubiquitous sensors. In *Pervasive Computing, Second International Conference, PERVASIVE 2004*, pp. 158–175, Vienna, Austria (April, 2004).

[17] D. J. Patterson, D. Fox, H. A. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *ISWC*, pp. 44–51. IEEE Computer Society, (2005). ISBN 0-7695-2419-2. URL `http://doi.ieeecomputersociety.org/10.1109/ISWC.2005.22`.

[18] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pp. 1–9, New York, NY, USA, (2008). ACM. ISBN 978-1-60558-136-1. doi: http://doi.acm.org/10.1145/1409635.1409637.

[19] T. Duong, D. Phung, H. Bui, and S. Venkatesh, Efficient duration and hierarchical modeling for human activity recognition, *Artif. Intell.* **173**(7-8), 830–856, (2009). ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2008.12.005.

[20] S. Luhr, H. H. Bui, S. Venkatesh, and G. A. West, Recognition of human activity through hierarchical stochastic learning, *percom*. **00**, 416, (2003). doi: http://doi.ieeecomputersociety.org/10.1109/PERCOM.2003.1192766.

[21] N. Oliver, A. Garg, and E. Horvitz, Layered representations for learning and inferring office activity from multiple sensory channels, *Comput. Vis. Image Underst.* **96**(2), 163–180, (2004). ISSN 1077-3142. doi: http://dx.doi.org/10.1016/j.cviu.2004.02.004.

[22] A. Subramanya, A. Raj, J. Bilmes, and D. Fox. Hierarchical models for activity recognition. In *IEEE Multimedia Signal Processing (MMSP) Conference*, Victoria, CA (October, 2006).

[23] L. Chen, C. Nugent, M. Mulvenna, D. Finlay, X. Hong, and M. Poland. Using event calculus for behaviour reasoning and assistance in a smart home. In *ICOST '08: Proceedings of the 6th international conference on Smart Homes and Health Telematics*, pp. 81–89, Berlin, Heidelberg, (2008). Springer-Verlag. ISBN 978-3-540-69914-9. doi: http://dx.doi.org/10.1007/978-3-540-69916-3_10.

[24] B. Bouchard, S. Giroux, and A. Bouzouane. A Logical Approach to ADL Recognition for Alzheimer's patients. In *Smart Homes And Beyond: Icost 2006, 4th International Conference on Smart Homes and Health Telematics*, p. 122. IOS Press, (2006).

[25] A. Rugnone, F. Poli, E. Vicario, C. D. Nugent, E. Tamburini, and C. Paggetti. A visual editor to support the use of temporal logic for adl monitoring. In *ICOST*, pp. 217–225, (2007).