# A Dynamic Sliding Window Approach for Activity Recognition

Javier Ortiz, Angel García Olaya, and Daniel Borrajo

Departamento de Informática, Universidad Carlos III de Madrid,
Avda. de la Universidad. 30, 28911 Leganés (Madrid), Spain
jortiz@inf.uc3m.es,agolaya@inf.uc3m.es,dborrajo@ia.uc3m.es
http://www.uc3m.es

**Abstract.** Human activity recognition aims to infer the actions of one or more persons from a set of observations captured by sensors. Usually, this is performed by following a fixed length sliding window approach for the features extraction where two parameters have to be fixed: the size of the window and the shift. In this paper we propose a different approach using dynamic windows based on events. Our approach adjusts dynamically the window size and the shift at every step. Using our approach we have generated a model to compare both approaches. Experiments with public datasets show that our method, employing simpler models, is able to accurately recognize the activities, using fewer instances, and obtains better results than the approaches used by the datasets authors.

**Keywords:** Human Activity Recognition, Sliding Window, Sensor Networks, Wearable Systems, Ubiquitous Computing

## 1 Introduction

Generally speaking, human activity recognition (AR) can be defined as the automatic recognition of an activity or a state of one or more persons based on observations coming from sensor readings. Usually, AR problems are tackled as a machine learning problem where the observations collected by the sensors are the inputs, the performed activities are the classes, and the learning techniques generate classifiers. Sensors produce data streams that can be seen as a simple time series, a collection of observations made sequentially in time. So, the recognition system must process the inputs to extract the learning instances, their feature values and the classes. The features depend on the available sensors. Thus, in [12] RFID sensors are used and the features extracted are the RFID tags detected by the RFID reader. In [8], two-state sensors are used and the features are the states of all sensors. Other types of sensors like accelerometers produce continuous data streams and the features must be extracted from those. For instance, the features extracted in [1] are the mean, energy, frequency-domain entropy, and correlations of the other features. In [6], they use similar features as well as the magnitude of the mean, variance, energy, spectral entropy, and the discrete Fast Fourier Transform (FFT) coefficients.

Independently of the sensors used, in the feature extraction step most AR systems use a sensory sequence segmentation based on a fixed-size sliding window [1, 15, 14]. In those cases, many of the classification errors come from the selection of the sliding window length [4]. For instance, an incorrect length may truncate an activity instance. In many cases, errors appear at the beginning or at the end of the activities, when the window overlaps the end of one activity and the beginning of the next one. In other cases, the window length may be too short to provide the best information for the recognition process. In [6], the authors studied different features and window lengths. They showed that the best performance is achieved when different window lengths and features are chosen separately for each activity.

Besides, the static sliding window approach generates many identical consecutive temporal windows with exactly the same features and the same activity performed when the user executes the same activity during a long period of time. Those repetitive instances do not contribute to solve the problem better. Instead, they produce higher scores of the activities during which those instances are generated. But they do not help to recognize other activities, and the systems have to classify the identical instances over and over again.

For those reasons, we hypothesize that a different segmentation approach based in non-fixed length windows may achieve better results. Thus, we propose an approach based in events to generate dynamic sliding windows to infer the activities. So, instead of defining a static fixed-length window, we define the events that will be used to define the boundaries of the dynamic windows employed to extract the features. Hence, when a specific event in the sensors readings is detected, we extract the features to classify what the user did between that event and the previous one. Those features are always the same, but the size of the windows changes based on when the events happen. So, the size of the window is dynamically established by the events. Thus, the windows are dynamic in time although the number of events in a window is always the same. In addition, our method does not create any temporal window if no events are detected.

The events we use are domain dependent and rely on the sensors the system uses. In the case of Radio Frequency Identification (RFID) or reed switch sensors, an event could be any sensor state change. That is the case of the datasets used in this paper. Using sensors producing continuous data like accelerometers, magnetometers, gyroscopes or GPS's, one or several thresholds could be set in order to detect the events. Since there is no evidence that this method would work using this kind of sensors, in the near future, we plan to test the method using data from this type of sensors.

The goal of this paper is to learn the actions that users perform, using the dynamic window method based on state changes on public datasets, and to compare the results with other approaches. For our experiments, we used data from two different sources. The first dataset used was presented in [8]. It uses a set of two state sensors deployed in a house. The second dataset is the one used in [12]. In this dataset, RFID readers and a set of RFID tags installed in the environment are used to detect the activities. We built models using some

state-of-the-art algorithms for classifying the activities and models used by the authors of the datasets in order to compare their models and ours.

The rest of the paper is organized as follows. In Section 2 we describe models generated for testing. In Section 3 we report on the results of the approaches used. Section 4 discusses related work. Finally, in Section 5 we summarize our findings.

## 2 Dynamic Windows Based on State Changes

Given a network of $N$ sensors, $N$ sequences will be generated by the AR system where each sequence can be represented as a vector $X^s =< .., x_i^s, .. >$ of readings of sensor $s$. $x_i^s$ is the sensor reading at time $i$ of sensor $s$. So, the first task consists of defining a function $f_1$ that takes the $N$ sequences and produces $Z$ vectors of features $\boldsymbol{F_i}$. Each vector is labeled with the activity $a$ that the user performed during the period of time from $i$ to $i+l$ in which the features in $\boldsymbol{F_i}$ were extracted. The second task is to learn a function $f_2$ that takes as inputs those vectors $\boldsymbol{F_i}$ produced by $f_1$ and builds a classifier to infer the activities performed.

The static sliding window approach uses fixed-length temporal windows that shift to create instances. Each window position produces a segment that is used to isolate data for later processing. It uses two parameters: the windows length $l$ and the shift $r$. Figure 1 shows an example of sliding windows where $l = r$, $i$ is the timestamp at which the first window starts and $i + l$ is the timestamp at which it finishes and the next temporal window starts.

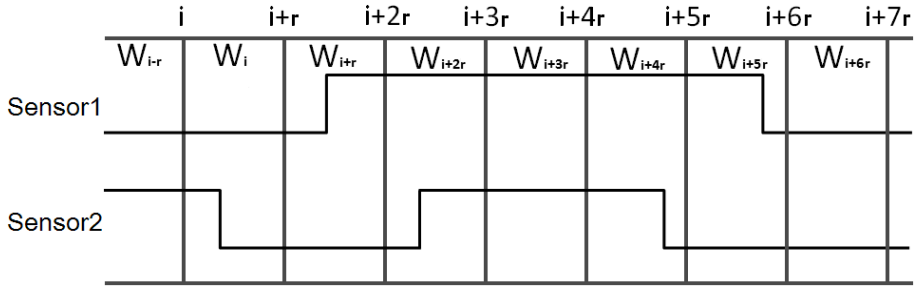


**Fig. 1.** Temporal segmentation on time series of 2 sensors by the sliding window method.

So, using this method, the function $f_1$ may be defined as follows. Given a network of $N$ sensors, $N$ sequences of data are generated. Each sequence is segmented in $Z$ temporal windows or time slices of $l$ seconds in length defined as $W_i^s =< x_i^s, ..., x_{i+l-1}^s >$ of contiguous readings from the sensor $s$ starting at time $i$. The window shift, $r$, defines the next temporal window as $W_{i+r}^s =< x_{i+r}^s, ..., x_{i+r+l-1}^s >$. The segments that start at time $i$ are grouped in the matrix

$W_i =< W_i^1, .., W_i^N >$. These temporal windows are represented as squares in the figure 1. Then, the features are extracted from $W_i$ to build the vector $F_i$ which is labeled with the activity $a \in A = \{a_1, a_2, ..., a_n\}$ that the user performed during $W_i$. Thus, given a set of $n$ activities $A = \{a_1, a_2, ..., a_n\}$ (classes), every temporal window $W_i$ produces a vector $\boldsymbol{F_i}$ that is labeled with an activity $a \in A$. Then, the function $f_2$ builds a classifier to find the mapping between $\boldsymbol{F_i}$ and the activity in $A$ that was performed by the user.

In contrast, our approach generates the learning instances, given by $f_1$, from a temporal window created by using as boundaries what we call *significant* events. So, it uses the last $m$ *significant* events to generate the learning instances. Also, instead of sliding, it uses the next events to fix the boundaries of the next instance. Hence, the approach relies on the events detected by the sensors that the system uses instead of the window length. Figure 2 shows an example of our method using as significant events all the changes in the values of the sensors.

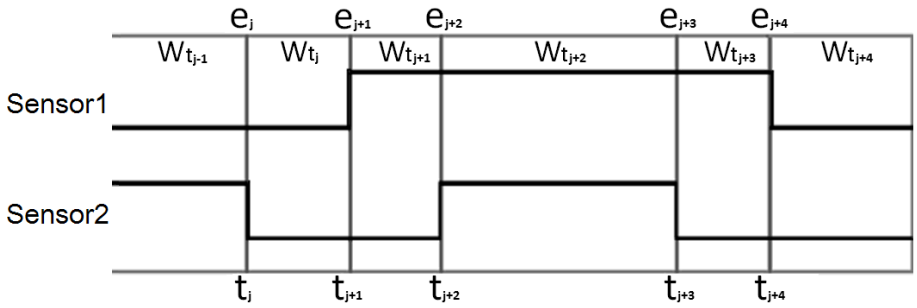


**Fig. 2.** Temporal segmentation on a time series of 2 sensors by the dynamic window method.

Thus, our method does not set values for $l$, the size of the window, and $r$, the shift. These values change over the time. The function $f_1$ can be formulated as follows. Given $N$ sequences of sensors readings as above, $X^s$, we generate one sequence of significant events $E =< e_0, e_1, .., e_j, .. >$ that are detected at time steps $T =< t_0, t_1, .., t_j, .. >$ where $t_j$ is the timestamp of event $e_j$. The events are detected from all sensors readings, but they are merged in $E$. Then, the sequences of all sensors are segmented by the events $e_x \in E$, from all sensors. Those events will divide the sequences in $Z$ temporal windows $W_{t_j}^s =< x_{t_j}^s, .., x_{t_{j+m-1}}^s >$ where $W_{t_j}^s$ will contain all the readings of sensor $s$ from $t_j$ to $t_{j+m-1}$, being $m$ the number of significant events used to build the windows. The next window will be defined as $W_{t_{j+1}}^s =< x_{t_{j+1}}^s, .., x_{t_{j+1+m-1}}^s >$ so the shift at every step will be set dynamically as $r_{j+1} = t_{j+1} - t_j$. Then, $W_{t_j} =< W_{t_j}^1, .., W_{t_j}^N >$ will be the segments of the $N$ sensors in the time $t_j$. The figure 2 show this segments $W_{t_j}$ as squares where $m = 2$. Events will divide the sequences of all sensors, even the sequences of the sensors that did not detect such event.

Once the temporal windows are delimited, the features $\boldsymbol{F_{t_j}}$ are extracted and they are labeled with an activity $a_j \in A$. The activity that will label the window $W_{t_j}$ will be the activity that the user performed between the last two events $t_{j+m-2}$ and $t_{j+m-1}$. Finally, the function $f_2$ is performed.

One of the main differences between both methods is that our approach generates a new window just when a new event is detected, whereas the other approach continues creating new windows even when the sensor readings do not change; temporal windows created with no changes in the sensors are identical. It can be seen in the figure 1 where the window $W_{i+3r}$ does not contain any event, so the vector $\boldsymbol{F_{i+3r}}$ will be identical to the vector $\boldsymbol{F_{i+2r}}$ generated by the previous window $W_{i+2r}$.

In order to test our approach, we generated models based on the sensors used to record the datasets. Hence, for each dataset we chose different *significant* events to generate the classification instances. In the first dataset we generated models following the dynamic window approach. We compared our models with the fixed-length sliding-window approach employed by the authors. We used the second dataset in a similar way. We generated models using our approach and we compared our models with the model that got better results from the ones generated by the authors of the dataset. Next, we describe the models generated for our experiments in detail.

### 2.1 Kasteren Dataset

The first dataset was recorded and used by Kasteren and colleagues in [8]. The sensor network consists of wireless network nodes to which simple off-the-shelf sensors can be attached. Each sensor sends an event when the state of the digital input changes or when some threshold of the analog input is reached. The dataset was recorded in the house of a 26-year-old man living alone in a three-room apartment where 14 state-change sensors were installed. Locations of sensors included doors, cupboards, refrigerator and a toilet flush sensor. Sensors were left unattended, collecting data for 28 days in the apartment. The dataset contains 2638 sensor events and 245 activity instances. Activities were annotated by the subject himself using a bluetooth headset as described in [8]. Seven different activities were annotated, namely: *Leave house*, *Toileting*, *Showering*, *Sleeping*, *Preparing breakfast*, *Preparing dinner* and *Preparing a beverage*. Times where no activity is annotated are referred to as *Idle*. The dataset is public and can be downloaded with its annotations.

### 2.2 Kasteren Models

We executed three experiments with this dataset. In the first one, we reproduced the model that achieved the best results in [8]. They employed temporal probabilistic models and divided the data in slices of constant length, 60 seconds, without overlapping. So, the parameters used were $l = 60$ seconds and $r = 60$ seconds. A vector of features was generated for each slice. The vector contained one entry for each sensor, where the values of the sensors could be `0` or `1`. They

performed four experiments. We will focus in the one that got better results. In that experiment they used two representations in parallel that they called *change point* and *last*. In the *change point* representation the sensor gives a 1 to time slices where the sensor reading changed. In the *last* representation the last sensor that changed its state continues to give 1 and changes to 0 when a different sensor changes state. In our experiments we recreated this in the first experiment but we used a Dynamic Bayesian network (DBN) equivalent to the Hidden Markov Model (HMM) used by them since an HMM is a simple DBN. To do so, we added the id of the last activity performed to the vector of features.

In order to test our approach, we built some models using the same representation over the same dataset, but generating the instances using the dynamic window approach and using a different representation. To use this approach, we considered as a significant event any change in the sensor readings. That is, when a sensor changes its state from *1* to *0* or vice versa. Since we used changes in the sensors readings instead of slices of constant length, we generated our instances from the last *10* changes. A way to select the length of the temporal window is using the average time spent performing an activity in the data, as in [12]. Instead of using the time, we used the events. So, we divided the state changes of the dataset, `2638`, by the number of activities `245` and we obtain $m = 10$. There were 14 sensors, so the features were a vector $\boldsymbol{F_{t_j}} =< F_j^1, ..., F_j^{14} >$ for the event $e_j$ where $F_j^n \in \{0, 1\}$. The $F_j^n$ was set to 1 if the sensor $n$ changed its state at least once between $t_j$ and $t_{j+9}$ since $m$ is set to 10. Also, we kept to 1 the last sensor that changed its state. This way we reproduced the representations *change point* and *last* used by the author. Additionally, we added the id of the last activity performed to the vector of features to learn activity transitions like the HMM does. This experiment was called *DSW-1-K* for Dynamic Sliding Window using Kasterens dataset. For the third experiment, *DSW-2-K*, we created a new model using a different representation and features. Instead of using the state of all sensors during the temporal window, whether the sensor changes its state or not, in this model we construct the vector using the identifier of the sensors that produced the last 10 events. So, the features of the segment that starts in $e_i$ were $\boldsymbol{F_{t_j}} =< s_{t_j}, ..., s_{t_{j+9}} >$ where $s_{t_n} \in \{1, ..., 14\}$ since there were 14 different sensors. The $s_{t+9}$ is set to the identifier of the sensor that produced the event $e_{j+9}$ and $s_{t_j}$ is the identifier of the sensor that produced the event $e_j$. If a sensor is not responsible for any event in the last 10 ones it is not included. Also, if a sensor produced $k$ events, the identifier of that sensor is included $k$ times. So, for example, if in the current temporal window $W_{t_j}$ the sensors that changed its state were $1, 2, 3, 4, 1, 2, 3, 4, 5, 6$ then the features used would be $\boldsymbol{F_{t_j}} =< 1, 2, 3, 4, 1, 2, 3, 4, 5, 6 >$.

Then, we tried to reduce the number of features extracted from the sensors performing a feature selection method by computing the Information Gain Ratio [5] for each of the features and then ranking them from highest to lowest. Afterwards, we created and tested a model using all the features. The worst feature according to the ranking was eliminated and a new model was generated and tested with the remaining features. All the features were progressively

eliminated until none is left. All generated models are compared and the features employed in the best model were kept. Using this feature selection method, we reduced the size of the vector from 10 to the last two sensor state changes $\boldsymbol{F_{t_j}} =< s_{t_{j+8}}, s_{t_{j+9}} >$ being $s_{t_{j+8}}$ and $s_{t_{j+9}}$ the id of the sensors that detected the last 2 events. Later, we added the id of the last activity performed like in the other models. Since we are interested in comparing probabilistic models with others models more easily readable by humans, *DSW-1-K* and *DSW-2-K* were generated using *PART* [3], an algorithm that generates rule sets, and *C4.5* [13], a decision-trees learning algorithm, instead of a DBN. We use these two algorithms because the model they generate can be transformed into a simple computer program that may be used to decide the activity performed by the user.

### 2.3 Patterson Dataset

The second dataset used is the one in [12]. The experiments performed with this dataset focused on routine morning activities which used common objects and are normally interleaved. The 11 activities which were observed are: *Using the bathroom*, *Making oatmeal*, *Making soft-boiled eggs*, *Preparing orange juice*, *Making coffee*, *Making tea*, *Making or answering a phone call*, *Taking out the trash*, *Setting the table*, *Eating breakfast* and *Clearing the table*. To create the dataset, one of the authors performed each activity 12 times in two contexts: twice in isolation, and then on 10 mornings all of the activities were performed together in a variety of patterns.

In order to capture the identity of the objects being manipulated, the kitchen was outfitted with 60 RFID tags placed on every object touched by the user during a practice trial. Data are in the form: <objectID> <activityID>. This dataset is more challenging than the previous one; most tasks were interleaved with or interrupted by others during the 10 full data collection sessions. In addition, the activities performed shared objects in common. This made interleaved AR much more difficult than associating a characteristic object with an activity. The dataset is public and can be downloaded.

### 2.4 Patterson Models

We also executed three experiments using this dataset. In the first experiment we used the fixed-length sliding-window approach used by the authors of this dataset. We also used their features and representation. They divided the data in slices of constant length, where the mean length of each uninterrupted portion of the interleaved tasks was 74 seconds. At each second they generated a vector of features with the data of the last 74 seconds. So, the parameters used were $l = 74$ seconds and $r = 1$ seconds. The vector contained 74 entries, one for each second, where the values of each entry could be *object-X-touched* when an object was detected or *no-object-touched* when no objects were detected. They used temporal probabilistic models too. They tested four different models. We have reproduced one of the most accurate models they generated; an HMM equivalent

to the one used by Kasteren in the previous dataset. So, in order to replicate the results we created a DBN equivalent to the HMM used by the authors as with the Kasteren dataset.

In *DSW-1-P*, we used our approach with the same representation. We used a vector with the last 74 significant events to recognize the activity that was performed between the last two events. In this case, we considered as significant event when the RFIDs change the object detected or no objects are detected. So, the features used were the id of the new object detected or *no-object*. The dataset just provides information about the objects detected and the timestamps. Therefore, to detect the *no-object* state we assumed that when two readings are found in the dataset and the time interval between them is one second or more there is at least a state with no objects detected.

We also generated a new model using a simpler representation in *DSW-2-P*. We used a vector of features composed of the last 74 events like in the previous setup. Then, we applied the same feature selection method that we used in the previous dataset to find the optimal combination of features. That way we obtained a very accurate model using just the last two events instead of 74.

Like in the other dataset, we generated the models that use our approach, *DSW-1-P* and *DSW-2-P*, employing the *C4.5* and *PART* algorithms. We added the activity performed previously to the vector of features used in these experiments to learn activity transitions like we did in the previous dataset.

## 3 Experimental Results

In summary, we have performed six experiments, using the models described in the previous section. We used two metrics to evaluate the models using 10-fold cross-validation for estimating the error: precision and recall averaged over all activities. We have used these two metrics instead of accuracy, used by the authors of the datasets, because the datasets are unbalanced. So, accuracy is not a good metric as described in [2]. Hence, we have used precision and recall as recommended by Kasteren in [16, 2]. The measures are calculated as follows:

$$\text{Precision}=\frac{1}{C}\sum_{c=1}^{C}\left\{\frac{tp_c}{tp_c+fp_c}\right\} \qquad \text{Recall}=\frac{1}{C}\sum_{c=1}^{C}\left\{\frac{tp_c}{tp_c+fn_c}\right\}$$

in which $tp_c$ are the true positives of the class $c$, $fp_c$ are the false positives of the class $c$, $fn_c$ are the false negatives of the class $c$ and $C$ is the number of classes. So, what we call precision was used as a metric by Kasteren but he called it $ClassAccuracy$.

The learning algorithms we used were *DBN* to learn the models that replicate the models used by the authors of both datasets and *C4.5* and *PART* to learn our models.

Table 1 shows the average precision and recall obtained by each setup, and the number of instances generated by each model. In addition, we have included

a column with the percentage of times that the temporal window generated by the segmentation method contains at least one event able to change the state of the sensors and produce an instance different from the one produced in the previous window. That is, any change in the sensors readings that generates one instance different from the previous one. This counter measures the diversity of the temporal windows from which the instances are created since the values of the sensors in the temporal windows without state changes will be the same as in the previous window.

**Table 1.** Precision, recall, number of instances and diversity for all Setups.

| | Precision | Recall | N. Instances | Diversity |
|---|---|---|---|---|
| Kasteren DBN | 80.55 | 80.08 | 40003 | 3.16% |
| DSW-1-K C4.5 | 92.16 | 91.15 | 2638 | 68.35% |
| DSW-1-K PART | 92.28 | 90.85 | 2638 | 68.35% |
| DSW-2-K C4.5 | 93.05 | 91.34 | 2638 | 68.35% |
| DSW-2-K PART | 92.61 | 91.38 | 2638 | 68.35% |
| Patterson DBN | 78.90 | 86.57 | 16280 | 27.16% |
| DSW-1-P C4.5 | 94.80 | 94.48 | 5408 | 100% |
| DSW-1-P PART | 97.72 | 96.76 | 5408 | 100% |
| DSW-2-P C4.5 | 94.80 | 94.49 | 5408 | 100% |
| DSW-2-P PART | 97.69 | 95.32 | 5408 | 100% |

As we can see from the results of both datasets, the precision and recall of the static sliding-window approach, *Kasteren DBN* and *Patterson DBN*, are much lower than the dynamic window, *DSW-1-K*, *DSW-2-K*, *DSW-1-P* and *DSW-2-P*. The table shows that the different models generated using the dynamic window approach obtain similar results. The feature selection improved slightly the results in the first dataset, though not in the second dataset. The precision of *Kasteren DBN* is 80.55%, quite similar to the result reported by Kasteren in [8] which is 79.4%. The DBN we used needs some parameters to be defined, so the difference probably is due to dissimilarities in those parameters. We can not compare our results with those reported by Patterson in [12] since they do not report on prediction and recall.

The table shows as well that the number of instances created by each method is very different. The static sliding-window approach creates many more instances than our approach in both datasets. Our method creates an instance just when an specific event is detected, while the static sliding-window approach creates instances for every time slide even when the user is not at home, or is sleeping and no events are detected. That is the case of the first dataset. The behavior in the second dataset is similar; many instances are created when the

system does not detect any event, since one instance is created every second. So, most of the instances are generated without changes in the sensor readings. This fact is shown in the last column of the table where our models generated higher percentages of different temporal windows. Notice that the diversity of the dynamic window in the first dataset is not 100% like in the second dataset because some of the significant events produced the same feature. When a sensor changed its state from 1 to 0 or vice versa the value of the feature extracted was 1 in both cases because of the *change point* representation. Thus, some consecutive instances generated were identical. In any case, using near 5% of the instances generated by the sliding window approach, we obtain better precision and recall.

A deeper analysis of the models shows that the sliding window fails more often when the activity changes. Comparing the *Kasteren DBN* and *DSW-2-K PART* we see that the first one fails in the 96.37% of the instances when the activity changes, whereas the second one fails in 48.12%. The results are similar in the second dataset, so *Patterson DBN* fails in 86.58% whereas the *DSW-2-P PART* fails in 42.13%. Thus, the activities with fewer instances reach a lower precision and recall, whereas the activities with many instances like *leave house* and *sleep* in the first dataset and *Making soft-boiled eggs* in the second get better results.

The experiments show that the different classification algorithms obtained similar results. Although experiments with Pattersons dataset show that PART obtained better results than C4.5, the differences are small in both datasets.

## 4 Related Work

In relation to sensor data segmentation approaches, most AR systems use a sensory sequence segmentation based on a static sliding window for extracting the relevant features. In [1] the feature vectors were computed on 512 sample windows of acceleration data with 256 samples overlapping between consecutive windows. In others [15], the duration for each feature vector was the average duration for each activity computed from all the activities and the shift in the feature window was half of the duration of the quickest activity. In [14], the sensors used in [1, 15] are combined. Each feature is computed over a sliding window shifted in increments of 0.5 seconds. They evaluated the performance of the features both individually and in combination, and over different window lengths (0.5sec-128sec). In [6], different features and window lengths are studied to recognize some activities. They show that the best performance is achieved when different window lengths and features are chosen separately for each activity.

So, using different deployments or different types of sensors, the common approach is to extract the features over a fixed length sliding window. Since a fixed length temporal feature slice is not the optimal solution [6], in our system we do not use them to compute the features. Instead, we compute the features every time a *significant* event is detected in the system. In that way we avoid the limitations of the static sliding windows approach.

Although the static sliding window is the most commonly used method, it is not the only one. For example, an approach that does not use sliding windows and instead employs events can be found in [7]. They use a two-stage method which consists of a preselection stage that aims to localize and preselect sections in the continuous signal and a second stage that classifies the candidate sections selected in the previous stage. The preselection stage looks for relevant motion events, which is very similar to what we do with our *significant* events approach. Ours is different because we try to find the events that mark the beginning and the end of one activity and they try to find one event that marks the whole activity, since it tries to recognize very short actions. Other different approaches that are not based on sliding windows may be found in [10, 11].

## 5 Conclusions

In this paper we have presented a different approach to create the learning instances for AR. The novelty of this approach relies on the fact that our system uses the changes in the information captured by the sensors to create the instances for classification instead of the temporal sliding-window approach. We have compared our approach in public datasets used in the past by other researchers and we have shown a good performance. The results show that our approach obtains higher scores in precision and recall in the datasets used to test the approach.

The main advantage of the dynamic window approach is that it provides accurate models using much fewer number of learning instances and features. This makes this approach suitable to be used online and in situations where computation times are important, since fewer instances would be evaluated and the instances will be processed faster. Also, it is independent of the time granularity.

One limitation is that instances depend on the sensors accuracy. So, whenever the sensors do not capture a significant change in the environment, the system does not detect the state change and it does not create the corresponding instance. Anyway, this case is equivalent to the case when the temporal window is too long and contains two activities instead of just one. In [9] the author has shown some of the limitations of RFID in extensive use. However, other sensor modalities like accelerometers or motes can be used to recognize the activities.

We have described how models can be used for AR employing this method to extract the information from the sensor readings. The generated models are able to predict transition probabilities better by recording the last objects observed in each activity. So, good results can be obtained by using just the last two objects detected by the RFID or the last two reed switch sensor that changed the value. While there are still technical challenges to overcome, this work shows that AR using the significant events approach to generate the instances and just some of the changes in the states of the sensors as features can be a good choice.

For our future work, we will try our approach with sensors that produce continuous signals, like accelerometers, in order to test the method with other type of sensor and features. Also, we will test the method to recognize short length

activities like gestures. This is called activity spotting by some researchers [7]. We believe our approach may obtain good results in such task too.

## References


1. L. Bao and S. Intille. Activity recognition from user-annotated acceleration data. *Lecture Notes in Computer Science*, pages 1–17, 2004.
2. N. Chawla. Data mining for imbalanced datasets: An overview. *Data Mining and Knowledge Discovery Handbook*, pages 875–886, 2010.
3. E. Frank and I. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151. Citeseer, 1998.
4. T. Gu, Z. Wu, X. Tao, H. Pung, and J. Lu. epSICAR: An Emerging Patterns based approach to sequential, interleaved and Concurrent Activity Recognition. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications-Volume 00*, pages 1–9. IEEE Computer Society, 2009.
5. M. Hall and L. Smith. Practical feature subset selection for machine learning. *Computer Science*, 98:4–6, 1998.
6. T. Huynh and B. Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, page 163. ACM, 2005.
7. H. Junker, O. Amft, P. Lukowicz, and G. Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.
8. T. Kasteren, N. Athanasios, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 1–9, New York, NY, USA, 2008. ACM.
9. B. Logan, J. Healey, M. Philipose, E. Tapia, and S. Intille. A long-term evaluation of sensing modalities for activity recognition. *Lecture Notes in Computer Science*, 4717:483–500, 2007.
10. J. Modayil, T. Bai, and H. Kautz. Improving the recognition of interleaved activities. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 40–43. ACM, 2008.
11. J. Ortiz, A. García, and D. Borrajo. A Relational Learning Approach to Activity Recognition from Sensor Readings. In *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*, volume 3, 2008.
12. D. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Ninth IEEE International Symposium on Wearable Computers*, pages 44–51, 2005.
13. J. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 2003.
14. M. Stikic, T. Huynh, K. Van Laerhoven, and B. Schiele. ADL recognition based on the combination of RFID and accelerometer sensing. In *2nd International Conference on Pervasive Computing Technologies for Healthcare*, volume 2008, 2008.
15. E. Tapia, S. Intille, and K. Larson. Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors. *Lecture Notes in Computer Science*, pages 158–175, 2004.
16. T. Van Kasteren, G. Englebienne, and B. Kröse. Towards a Consistent Methodology for Evaluating Activity Recognition Model Performance.