New incremental SVM algorithms for human activity recognition in smart homes

Yala Nawal, Mourad Oussalah, Belkacem Fergani, Anthony Fleury

Journal of Ambient Intelligence and Humanized Computing (2023) 14:13433–13450
https://doi.org/10.1007/s12652-022-03798-w

ORIGINAL RESEARCH

New incremental SVM algorithms for human activity recognition in smart homes

Yala Nawal[1] · Mourad Oussalah[2] · Belkacem Fergani[1] · Anthony Fleury[3]

Received: 19 August 2021 / Accepted: 7 March 2022 / Published online: 24 March 2022

Abstract
Smart homes are equipped with several sensor networks to keep an eye on both residents and their environment, to interpret the current situation and to react immediately. Handling large scale dataset of sensory events on real time to enable efficient interventions is challenging and very difficult. To deal with these data flows and challenges, traditional streaming data classification approaches can be boosted by use of incremental learning. In this paper, we presented two new Incremental SVM methods to improve the performance of SVM classification in the context of human activity recognition tasks. Two feature extraction methods elaborated by refining dependency sensor extraction feature and focusing on the last sensor event only have been suggested. On the other hand, a clustering based approach and a similarity based approach have been suggested to boost learning performance of the incremental SVM algorithms capitalizing on the relationship between data chunk and support vectors of previous chunk. We demonstrate through several simulations on two major publicly available data sets (Aruba and Tulum), the feasibility and improvements in learning and classification performances in real time achieved by our proposed methods over the state-of-the-art. For instance, we have shown that the introduced similarity-based incremental learning is 5 to 9 times faster than other methods in terms of training performances. Similarly, the introduced Last-state sensor feature method induces at least 5% improvement in terms of F1-score when using baseline SVM classifier.

Keywords Smart home · Activity recognition · Incremental learning · Incremental SVM

1 Introduction

Smart home applications aim to make life easier and more convenient for individuals, especially those with restricted mobility Vischer (2007); Allameh et al. (2011). Some smart homes provide peace of mind to its inhabitants by transmitting regular reports regarding the status of key objects and any suspected activities. Others provide energy savings through intelligent management of user's location and activities. Smart home technologies also contributed to assist elderly raising the safety standards. Solaimani et al. (2013) and Wilson et al. (2014) reported that the majority of the surveyed smart home projects have an application on energy optimization, safety or health. Strictly speaking, activity is the primary type of context that characterizes the state of an individual within surrounding inhabitants Abowd et al. (1999); Cook et al. (2015). Hence, the need for activity recognition system is crucial. In the area of ubiquitous sensing, a flexible and transparent set of wireless sensors is embedded into everyday objects (e.g. fridges, door, cupboards, bed, etc.) such that the inhabitant's interactions with these objects provide insights for identifying the ongoing activities of daily living (such as cooking, leaving home, sleeping, eating, etc.).

Despite a large number of works in human activity of daily living recognition (HAR) Straczkiewicz et al. (2021), Demrozi et al. (2020), Fu et al. (2020), Liu et al. (2020), the recognition phase turns in a "delayed mode" (Tapia 2003; van Kasteren et al. 2008a; Krishnan and Cook 2012). That is to say, the system waits1 for a given time- lapse- to collect a certain number of new pieces of information to predict

✉ Mourad Oussalah
Mourad.Oussalah@oulu.fi

[1] LISIC Laboratory, Electronics and Computer Sciences Department, University of Science and Technology Houari Boumediene, Algiers 16000, Algeria

[2] Faculty of ITEE, CMVS, University of Oulu, PO Box 4500, Oulu 90014, North Ostrobothnia, Finland

[3] Computer Science and Automatic Control Department, University of Lille, IMT LIlle Douai, Lille 5900, Nord, France

the current activity. While in real-time recognition system, each new piece of information needs to be classified at the moment of its arrival. In smart home like sensor network, each event occurs within a specific daily living activity. Besides, in the delayed recognition mode, the system collects a number of events and predicts one single activity for them. However, since people do their activities in a sequential, interleaved and concurrent manner, it is not excluded that two consecutive events belong to different activities (see Fig. 2). As such, the delayed recognition mode cannot identify non-sequential activities properly. One solution would be to classify each event alone at the moment of its arrival. A motivation for doing so arises when a specific application tracks the execution of a daily living activity step-by-step for delivering in-home interventions to a person or for giving brief instructions describing the way a task should be performed for its successful completion (Pollack et al. 2003). On the other hand, the learning phase employed by most daily recognition activity algorithms requires a large and consistent training database. The annotation of large dataset is often very complex and presents a high proportion of noise, which, in turn, compromises its reliability. For instance, smart homes on which our work is based generate an average of 7000 sensor events each day, which, in view of the high makespan taken for learning activity models, is considered as a large-scale problem. Support Vector Machine (SVM) based classification has established itself as a well-respected standard in daily living activity recognition task (van Kasteren et al. 2008b; Ordóñez et al. 2013; Wilson and Atkeson 2005; Tapia et al. 2004b) due to its rigorous mathematical foundation, good generalization capabilities and high accuracy rate. Nevertheless, in case of large-scale class-imbalance dataset, the limitation and complexity of the training phase are well acknowledged and documented as well (Barger et al. 2005). For instance, in Krishnan and Cook (2012), a batch of SVMs takes four days to learn activity models. Indeed, SVM training requires solving a quadratic programming (QP) problem in a number of coefficients equal to the number of training examples, which, in turn, makes standard numerical techniques for QP infeasible for a such large dataset. To overcome this difficulty, some practical techniques decompose the problem into manageable sub-problems over part of the data Lester et al. (2005) or, perform component-wise optimization (Kim et al. 2013), or, to some extent, iterative pairwise comparison (Wang et al. 2011). Other researchers suggested to transform the batch SVMs to the incremental ones by adapting an incremental or online learning techniques (Cauwenberghs et al. 2001; Syed et al. 1999). According to Bao and Intille (2004), there are three classes of incremental learning methods: example-incremental learning, class-incremental learning, and attribute-incremental learning, which cooperate new examples, new classes, and new attributes to the trained learning system respectively. On the other hand, since the behavior of people can change over time, this can affect the way the activities are performed by the individual over time. In this case, an update of activity models is required. Strictly speaking, possibly because of its offline nature and the fact that the speed of the prediction/estimation at the online phase is more relevant from the end-user perspective, the learning phase is often overlooked. Learning phase involves feature selection and model selection where optimal parameter values should be found. Furthermore, if the model is to be trained over a mobile or a resource-constrained platform, reducing the training time would still make more sense. This highlights the importance and critical nature of the training phase in this respect. The main contribution of this paper is to present a novel solution for HAR, which achieves two main goals. First, it enables a real-time human activity recognition task (A1). Second, it learns activity models incrementally (A2).

More specifically, our contributions are fourfold:

- We provide a short and concise review of incremental SVM learning methods with a focus on both computational and memory requirement handling as well as dealing with imbalanced class dataset, which often occurs in human daily activity recognition problems.
- Acknowledging the importance of feature engineering in HAR problems and incremental learning methods, we present two extensions of the approach proposed in Krishnan and Cook (2012). The key is to divide sensor event sequence into segments of equal length. The first approach extends the dependency sensor feature extraction method by re-interpreting the concept of mutual information between two sensors as the probability that the two sensors fall on the same windows (of a fixed size) of events. The second approach promotes the importance of the last-state of the sensor within the given segment so that its feature representation is reduced to the status of this last event of the segment. By doing so, we significantly enhance the computational performance of the developed model to achieve target A1.
- We propose two methods to train SVMs incrementally to achieve target A2 by capitalizing on the similarity between support vectors of previous data chunk and current batch of data to reduce the size of training sample. The first method performs this using a clustering based approach by iterating k-clustering algorithms and using the concepts of pure cluster and hybrid clusters brought from molecular studies, so that datum associated with pure clusters are discarded and those with hybrid clusters are added to support vector list. The second method suggests to prone the training datum at each data chunk using a two-side similarity calculus process, where support vectors that are found highly similar to data in the

new chunk are discarded, and then, datums that bear similarity in the sense of Euclidean distance metric to the same support vector are reduced.

- The developed methods are then assessed using two publicly available datasets commonly employed in activity recognition task developed in CASAS smart-home project; namely, Aruba and Tulum datasets. Then, their performances are compared to some state-of-the-art HAR methods that used the same dataset. The choice of these dataset is motivated by the availability of comparative results using state-of-the-art methods. A general overview of the concept advocated in this paper is highlighted in Fig. 1.

The remainder of this paper is organized as follows. Section 2 briefly revisits and concisely summarizes the work that has been previously performed on data segmentation and feature extraction methods for HAR purpose with a focus on incremental learning based approaches. The developed approach is then reported in Sect. 3 where two new feature extraction methods and two novel SVM incremental learning modes are put forward. Section 4 presents the experimental setup and results for assessing and discussing the proposed algorithms. Conclusion and future works are reported in Sect. 5.

2 Background

For the sake of illustration purpose and link with our developed approach, we distinguish in this section data segmentation / feature extraction task in the context of daily activity recognition and SVM incremental learning-based approach.

2.1 Streaming data segmentation and features extraction

In the context of daily human activity recognition task, the activities are often performed on a regular basis, consecutively and, sometimes, through a concurrent and interleaved

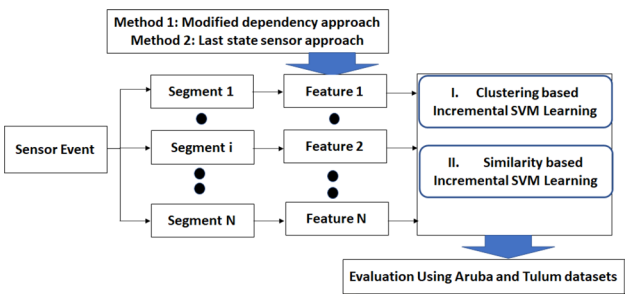

Fig. 1 Overall concept of the developed approach

activity execution, which makes it difficult to determine the exact boundaries between two instances of consecutive activities. Segmentation aims to distinguish various chunks / segments, each possibly corresponds to a single activity, from raw streaming sensor events taking into account time window, context and effects of uncertainty. Various segmentation approaches have been proposed in the literature depending on the nature of contextual information, sensory data and uncertainty framework. Especially, one distinguishes time-based (or time window-based) segmentation and sensor event based segmentation. Time-based segmentation divides data into fixed time windows. It is the most commonly used segmentation method for activity recognition Bao and Intille (2004); Tapia et al. (2004a); Wang et al. (2012). However, many of the classification errors using this method came from the selection of the window length Gu et al. (2009). If a small length is selected, there is a possibility that the window contains insufficient information to take an appropriate decision or train a machine learning based model. On the contrary, if the length is too wide, information of multiple activities can be embedded in one window, which reduces the ability to distinguish different activities. Another drawback of this technique arises when the sensors are discrete (e.g., in case of motion and door sensors that are "event-based"). For instance, in case of relaxing or sleeping activity, discrete sensors yield no change of their output values over a long period of time. The result of segmentation process in this case is either a number of silent windows (all sensors are off while segmentation process continues to provide empty windows) or repeated windows (no changes in the sensor values while segmentation process continues to provide windows with the same values of sensors). Therefore, time-based segmentation is more suitable for continuous sensors such as accelerometer which has a constant acquisition rate such that data for every time interval is always guaranteed. Sensor events-based segmentation divides sensor event sequence into windows of equal number of sensor events (Krishnan and Cook 2012). Typically, such windows have different time duration. Indeed, during the execution of activities, multiple sensors could be triggered, while at silent periods, whose time window can be much larger, a reduced number of sensors is fired. History of event occurrence provides insights to model and account for contextual information. Especially event-based segmentation is found to be more suitable in case of discrete sensors, which is the case for the study carried out in this paper. Figure 2 shows an example of stream sensor events during the acquisition stage, while Fig. 3 illustrates an example of time-windows and event-based segmentation. To characterize individual sensor event, previous sensor events are typically accounted for. More specifically, each event is described by the list of sensor events that precede it. Figure 4 illustrates this process. Nevertheless, a such method

Date	Time	IDs	Status	Activity
2010-11-04	17:57:06.97921	M026	ON	
2010-11-04	17:57:08.875092	M026	OFF	
2010-11-04	17:57:25.503492	M026	ON	
2010-11-04	17:57:29.514126	M026	OFF	
2010-11-04	17:57:30.478753	M026	ON	
2010-11-04	17:57:35.956698	M026	OFF	
2010-11-04	17:57:45.71581	M027	ON	
2010-11-04	17:57:46.296167	M026	ON	
2010-11-04	17:57:47.39778	M027	OFF	Work end
2010-11-05	00:00:11.187975	M003	ON	Sleeping begin
2010-11-05	00:00:14.292731	M003	OFF	
2010-11-05	00:00:14.680141	M002	OFF	
2010-11-05	00:01:08.87122	M003	ON	
2010-11-05	00:01:11.046478	M003	OFF	
2010-11-05	00:01:44.894195	M003	ON	

Fig. 2 Example of stream sensor events from Aruba dataset

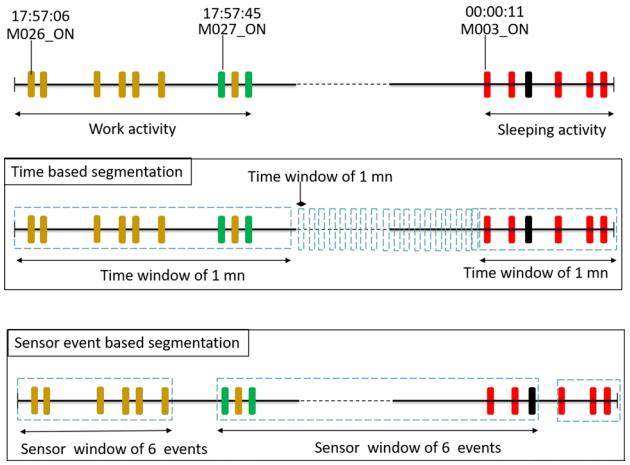

Fig. 3 Example of time-window and event-based segmentation

Fig. 4 Sensor event based segmentation for real time recognition

has also its own drawbacks. For example, consider the segment Seg_6 shown in Fig. 4. The last sensor event of this segment corresponds to the beginning of activity "Sleeping", while all events that precede it belong to another activity. We notice a significant gap between this event and the preceding. In fact, a such case represents a transition between two activities. Therefore, the relevance of the use of all sensor events in this segment with the last event in the same segment might be questionable considering the large elapsed time. Another drawback occurs when an inhabitant does two or more activities simultaneously (concurred activities). In a such scenario, one segment can contain sensor events of different activities. Therefore, a cautious attitude should be considered when dealing with last sensor events. Typically, once the sensor event window is defined, we transform this window into a feature vector that best captures its information content including temporal span, frequency of events and possibility first and last events. In this course, Krishnan and Cook Krishnan and Cook (2012) proposed a feature extraction method based on sensor dependency model to account for the relationship between the sensor events.

The detail of the feature extraction methods, including the newly proposed ones, is reported to Methodology section of this paper. Next, incremental SVM learning is presented.

2.2 Incremental SVM learning algorithms—background

As pointed out in the introduction section of this paper, despite the acknowledged advantages of SVM in terms of its rigorous mathematical foundation, resistance to overfitting by adjusting its regularization parameter, tackling nonlinearity issues through appropriate choice of kernel function, its training phase becomes computationally non-appealing for large scale dataset. Incremental learning is a suitable solution to speed up the training process or to handle the concept drift. In a such case, only a small subset of the data is considered at each step of the learning process and the solutions of the optimization problem are adapted when necessary. This learning mode is used when either the dataset is too large to be used at once or when all the data is not available at the training phase. An incremental learning algorithm, as defined by Polikar et al. (2001), follows the three procedures:

1. It learns from a new incoming data and adapts to changes in the data models in case of non-stationary data;
2. It does not require access to the original data used to train the new classifier;
3. It preserves the previously acquired knowledge.

SVM are known to be large margin classifiers that find a hyperplane to decide the class for a new data point. This hyperplane corresponds to the one with the largest margin between the classes. The dimension of a such hyperplane depends upon the number of features used for data representation. If the number of input features is 2, then the hyperplane is just a line, and becomes a two-dimensional plane if input features is 3, etc. Data points that are closer to the hyperplane and influence the position and orientation of the hyperplane are known as support vectors. Deleting a given support vector may ultimately change the position of the hyperplane. Therefore, the principle of SVM is to summarize

the training data using a set of relevant support vectors that enable hyperplanes with largest-margins between the corresponding classes. From the Karush–Kuhn–Tucker optimization condition perspective, this corresponds to training samples with non-null Lagrange multipliers. Since the majority of the training samples have zero Lagrange multipliers, e.g., they are non-support vectors and, thereby, have no influence on the SVM classification result, we can subsequently reduce the training process by dropping out a such data. This property yields great capacities to SVMs for their use into incremental learning scheme, by storing only the support vectors at each incremental step and discarding the remaining data (major part). Further relearning process (adaptation) is, in that case, simplified in terms of complexity.

2.3 Incremental SVM algorithms: state-of-the-art methods

We can distinguish two main approaches for extending classical SVM to accommodate incremental learning. The first one uses a recursive online algorithm (Cauwenberghs et al. 2001). To learn and adapt the model, a new instance is added to the learning set. If it is correctly classified by the current solution, no change will be necessary. Otherwise, an update of the current solution is performed by correcting the solution using the Lagrange multipliers while respecting the *Karush_Kuhn_Tucker* optimal conditions (Kuhn and Tucker 1951). An exact solution is constructed by this approach. However, its efficiency was questioned in case of large datasets as the update time could be non-negligible (comparing to the frequency of reception of the new data) (Gâlmeanu and Andonie 2008). To our knowledge, no successful practical application of this algorithm has been acknowledged.

The second approach is based on a set of adaptive algorithms Syed et al. (1999) where the training data is divided into chunks. At each incremental step, a significant amount of the training data is discarded while maintaining the set of support vectors describing the precedent decision boundary. Especially, once a new chunk of data is collected, there are different possibilities to update the current model. Syed et al. (1999) proposed a fixed-partition algorithm (*Fixed*) as shown in Fig. 5. Authors in Domeniconi and Gunopulos (2001) introduced the error-driven method *ERRD*, in which the new chunk of data is filtered at each incremental step. Previous model is used to classify the new chunk of data. If the data is misclassified, it will be maintained, otherwise it will be discarded. The support vectors of the previous incremental step together with the misclassified points are used as training data to obtain the new model. In this regards, misclassified data is considered as critical examples that have a higher likelihood to become support vectors during the next update step. If data is imbalanced and changes over time, this misclassification could be more important. This occurs for

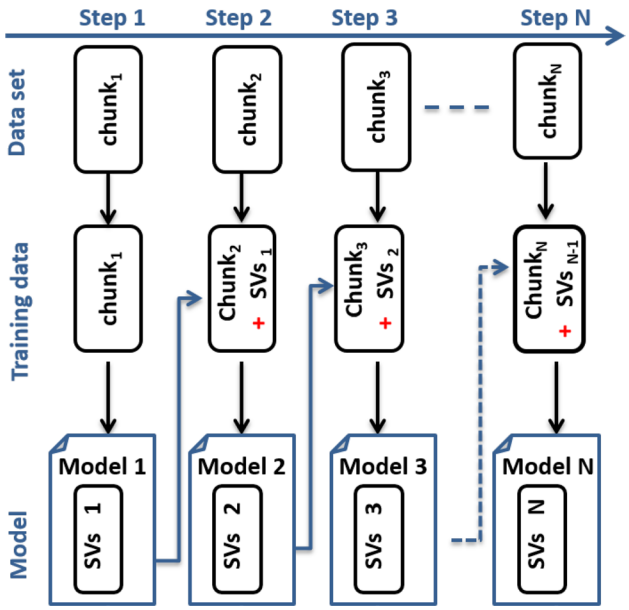

Fig. 5 Incremental SVM using adaptive process

instance when an inhabitant changes his/her way of living (e.g., to adapt to a new season, weather or accommodate new environmental conditions). Likewise, accommodating transitions between activities bears similar patterns. A such type of data instances prohibits updating the model correctly.

Furthermore, a common problem to *ERRD* and *Fixed* is that there is no limitation to memory growth. Fixed method adds all data in the new chunk and keeps all support vectors of the previous model. While *ERRD* filters data in a new chunk but keeps all support vectors of the previous model. As the incremental steps occur, the size of the reserved data increases and, hence, so is the learning time of the models as well. To cope with this problem, Pronobis et al. (2010) used Least Significant Support Vector Forget *(LSSVF)* to reduce memory footprint of the algorithm. This method discards the least relevant support vectors at each incremental step, i.e. the support vectors with the smaller value of Lagrange Multipliers. Authors in Domeniconi and Gunopulos (2001) used Oldest Support Vector Forget. This removes the oldest support vector of the current model. This sounds useful for applications in which the distribution of the data changes over time (non-stationary data). Least relevant Support Vectors are those having a very small Lagrange multiplier value. The support vectors can be bounded support vectors (BSVs) if their Lagrange multipliers values are equal to penalty C or unbounded support vectors (UBSVs) if their Lagrange multipliers values are smaller than C. BSVs are data vectors that lie beyond the margin of its class label. They can be approximated by the number of classification errors during training stage. Their number scales at least linearly with the number of training data. BSVs are associated with a

maximum Lagrange multipliers values. Therefore, they are not affected by the reduction of support vectors in the *LSSVF* method. Consequently, discarding support vectors with very small Lagrange multipliers does not speed up the training process in the case of large datasets. Furthermore, discarding an important amount of support vectors with small weight can decrease performance, especially the performances of small size classes, since a process similar to the unsampling has been applied. Our approach is also ultimately linked to the second approach (adaptive algorithm) pointed out earlier as we also aim to speed-up the training performance of the iterative learning of SVM model but with the introduction of new innovative approaches for both feature extraction and iterative learning method. In parallel to the preceding attempt in improving incremental learning of SVM, we shall also mention the growing effort to enhance the SVM computational performance by improving the efficiency of the underlined SVM optimization or by boosting parallelization of the associated software implementation. In this respect, Schlag et al. (2021) provided a recent up-to-date review of fast implementations of support vector machines, focusing on multi-level approaches. Fan et al. (2005) suggested a set selection methods to achieve faster convergence, while Osuna et al. (1997) put forward a decomposition algorithm that automatically compute the number of required of support vectors to achieve cost benefits. Yu et al. (2003) promoted the hierarchically clustered representation of the data by merging data points based on distance using linear classifiers, which are then extended to non-linear kernels. Similarly, Razzaghi and Safro (2015) suggested a graph representation instead of feature space representation and approximate k-nearest neighbors to yield a multilevel algorithm that trains the SVM where the support vectors of the previous coarser hierarchy level are used to train on the current level. The approach is also shown to be less sensitive to imbalanced data. In terms of software toolkit that boost the parallelization and GPU-based implementation, we shall mention the DC-SVM Hsieh et al. (2014) that implements a multilevel divide-and-conquer SVM that uses adaptive clustering, and Thunder SVM Wen et al. (2018), which implements a parallel SVM library that runs on GPUs as well as multi-core CPUs.

3 Method

3.1 Features extraction

3.1.1 Baseline feature extraction method (BL)

Before demonstrating the details of the *dependency sensors features method*, we shall present our baseline feature extraction method. Let us consider $[E_1, E_2, ..., E_N]$ a sequence of all sensor events collected from a given smart home. Each event is represented by its date and timestamp of day, sensor ID, sensor status and associated activity (see Fig. 2). Sensors IDs starting with *M* and *D* are motion and door sensors, respectively. Sensor status can be [ON, OFF, CLOSE, OPEN]. Usually, ON/OFF states are used for Television, Personal Computers, Cooking heater, and Cleaning appliances (e.g., ceiling light, vacuum), while CLOSE/OPEN states are used for Doors, Fridge (appliances with a door or a gate). Often, we can disregard the type of appliance in defining the states, which yields a binary state output only (so, ON/OFF become equivalent to CLOSE/OPEN). Following Krishnan and Cook (2012), in order to account for the context, the above sensor events are divided into equal number of sensor events. Let *m* be the number of events in the underlying window. Then a sensor event E_i is represented by the sequence $Seg_i = [E_{i-m}, E_{i-m+1}, ..., E_{i-1}, E_i]$. The hyperparameter *m* is calculated as follows: If $A = [a_1, a_2, ..., a_d]$ is the set of *d* activities performed in the smart home, $lA = [la_1, la_2, ..., la_d]$ is the average number of sensor events that were triggered during each activity. The hyperparameter *m* can take any value in the interval [*minA*, *maxA*] with which the classifier gives the best classification rate, such as $minA = min(lA)$ and $maxA = max(lA)$. To transform Seg_i to a F_i feature vector, we construct a fixed dimensional feature vector F_i containing:

1. The time-triggered of the first event in the window Seg_i, e.g., the time of event E_{i-m};
2. The triggering time of the last event in the window Seg_i, e.g., the time of event E_i;
3. The time duration of the window Seg_i, e.g., (time of E_i - time of E_{i-m});
4. A simple count of the different sensor events within the window Seg_i. For instance, if *l* is the number of sensors installed in the smart home, the dimension of the features vector F_i will be $l + 3$. F_i is tagged with label Y_i of E_iKrishnan and Cook (2012).

3.1.2 Dependency sensor features extraction method (DS)

AS explained above, we can observe a significant gap of time between two events E_{i-1} and E_i over the same segment. This time gap can be interpreted as a different location of sensors. Often, in activity recognition, two different locations means two different activities (e.g., transition between two activities or two residents do their activities in the same time in different locations). Furthermore, if two sensors have different locations, they often do not fire consecutively. Therefore, Krishnan and Cook (2012) suggested to use mutual information measure between sensors (refereed to as sensor dependency) as a weighting scheme in order to infer conclusions about last sensor event. This is defined as the likelihood that two sensors occur consecutively in the

entire sensor event sequence. In other words, if E_{i-1} and E_i occur consecutively at several occasions, the value of the dependency score will be high. This method aims to give a weight to each event in the segment Seg_i. The weight is nothing but the dependency between each event in Seg_i and the last event in the same segment. More specifically, if S_i and S_j are two sensors, then their dependency, or mutual-information, denoted $D(i; j)$, is defined as:

$$D(i,j) = \frac{1}{N}\sum_{k=1}^{N-1} \delta(\mathrm{S_k}, \mathrm{S_i})\delta(\mathrm{S_{k+1}}, \mathrm{S_j}) \tag{1}$$

$$\delta(S_k, S_i) = \begin{cases} 0 \text{ if } S_k \neq S_i \\ 1 \text{ if } \mathrm{S}_k = \mathrm{S}_i \end{cases} \tag{2}$$

The evaluation of $D(i; j)$ reaches a value one when the current sensor is S_i and the next sensor is S_j. The value of this dependency is linked to the proximity of both sensor events. To compute the feature vector F_i, instead of counting the different sensor events, we sum up the contributions of every sensor event based on dependency that defines the feature vector.

3.1.3 Two new approaches for feature extraction

Herein, we propose two new distinct approaches for feature extraction that can be utilized in our incremental learning HAR task.

Proposed method 1: Modified Dependency sensor feature extraction method (MDS)

The dependency between two sensors as previously described depends on the order of firing of sensors in the entire data sequence. For instance, let us assume, without loss of generality, four sensors, installed in a tight place of a smart home, that can be triggered simultaneously when a specific activity runs. Assume, for instance, that a given activity can be triggered using two distinct sequence of sensors; say, the sequence $\mathrm{S_1} \rightarrow \mathrm{S_2} \rightarrow \mathrm{S_3} \rightarrow \mathrm{S_4}$ or the sequence $\mathrm{S_1} \rightarrow \mathrm{S_3} \rightarrow \mathrm{S_2} \rightarrow \mathrm{S_4}$. Intuitively, one notices whenever the sensor $\mathrm{S_1}$ is fired, $\mathrm{S_2}$ is actioned as well, although a such firing is not necessarily successive, which makes the evaluation in Eq. (1) concludes the absence of dependency. To take into account this observation, we propose to compute the dependency between sensors $\mathrm{S_i}$ and $\mathrm{S_j}$ by calculating their frequency of occurrence within an interval of n sensor events along the entire data sequence (instead of tracking consecutive occurrences only as in Eq. 1), as quantified by the following equation:

$$D(i,j) = \frac{1}{W}\sum_{l=0}^{W-1} \nabla(\mathrm{S_i}, \mathrm{S_j}) \tag{3}$$

$$\nabla(S_i, S_j) = \begin{cases} 1 \text{ if } (S_i, S_j) \in [E_{l*n+1}, ..., E_{l*n+n}] \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

where W is the number of windows in which we compute the frequency of occurrence and n is the number of events in one single window. For a chosen window size n (number of events in the window), and a total number N of all sensor events collected from the given smart home, then it holds that $\mathrm{W} = N - n$.

The quantity $[\mathrm{E}_{l*n+1},...,\mathrm{E}_{l*n+n}]$ with l=0 refers to the first sequence of n events indicating the status of the sensors. Equation (4) indicates whether the event "occurrence of S_i is followed by occurrence of S_j" occurs in the sequence $[\mathrm{E}_{l*n+1},...,\mathrm{E}_{l*n+n}]$.

Proposed method 2: Last-State sensor based method (LS)

This method is motivated by our observation of inherent limitations in motion sensors installed in smart home, especially, small number of cone sensors (as shown in Fig. 6). Considering this fact, segments can contain sensors with active and inactive status (ON and OFF). Sensors with high sensitivity and detection capabilities can be triggered even if the inhabitant is not in the functional area of these sensors. This is because a smart home design often allows for a such situation to occur. For instance, motion sensor M020 is in an open surface. Therefore, we hypothesize that the last-state of a sensor within a segment Seg_i can be more informative and descriptive for the last event E_i. In this method, the feature vector is computed as follows: For each sensor S_i, if its last state within Seg_i is [ON/OFF] then it will be represented by respectively 1/-1 in the feature vector F_i, otherwise it will be represented by 0 (absent).

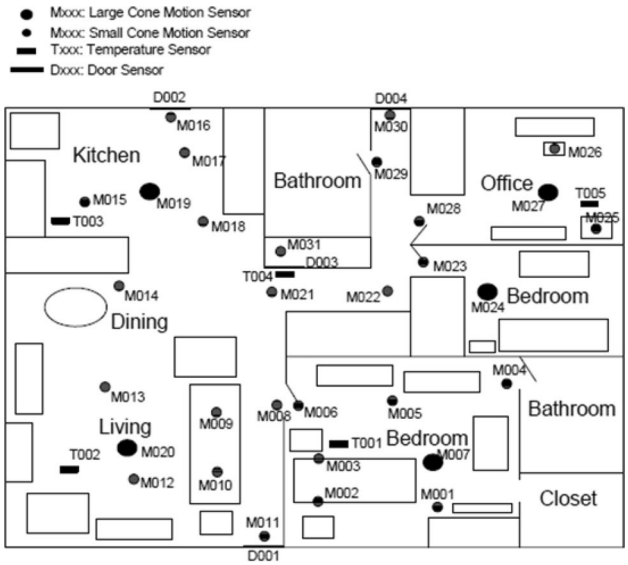

Fig. 6 Smart home floor plan in which Aruba dataset is collected Aruba and Tulum (2011)

The next step is feeding the learning machine with these features vectors, In the next section, we discuss our learning machine choice.

3.2 Proposed incremental SVM algorithm

Whatever the incremental SVM method, the model resulting from the first incremental step is considered as a preliminary version of the optimal decision plan. In the following incremental steps, the decision hyperplan will move in a way that is not *a priori* determined. Updating the model means to add new data as well as remove some old and/or obsolete data (remove some support vectors). We conjuncture the data that should be deleted must be either those that prevent the hyperplane from moving to a new "functioning mode"6 that is due to the evolution of the behavior or those that are redundant. Two new incremental SVM methods are described in this section to implement the aforementioned conjuncture. The first one aims to reduce the data size in new chunks, whereas the second aims to reduce both the number of support vectors preserved from the previous incremental step and the data size in new chunks.

3.2.1 Clustering-based incremental SVM method

This method aims to eliminate the data in new chunk that the system finds non-necessary to its evolution. The idea put forward here consists of a successive K-means clustering applied on the new chunk data. Specifically, we first apply K-means clustering on the new chunk data with a number of clusters taken equal to the number of activity classes in the new data. Then, we separate among the resulting clusters those containing only data with a unique activity label (pure clusters) from those containing multiple labels (hybrid clusters). Data from pure clusters are removed. Then, a second clustering is applied on the data belonging to hybrid clusters with a lower number of clusters to be generated. The new number of clusters (in the second k-means pass) is obtained by subtracting the number of distinct labels in identified pure clusters from the total number of activities available at the new chunk data. This process is repeated each time we find hybrid clusters. After a number of iterations, hybrid clusters data is kept and added to the support vectors of previous model to train the current one. The idea of discarding data can be explained by the fact that if the clustering algorithm has succeeded in producing pure clusters, it would then be easy to use SVM classifier to separate the two types of data. It is unlikely that the well separated data by clustering will be support vectors. On the contrary, hybrid cluster data represents critical points that are likely to become support vectors of the adapted model. The pseudo-code of successive clustering procedure is given in Algorithm 1, while the incremental SVM process is given in Algorithm 3. We denote this technique as *IncSucc* for future reference. In Algorithm 1, we deliberately reduced the number of iterations in hybrid-cluster to three, so that after a maximum of three applications of k-means (in a rare scenario, we may endup with large number of pure clusters and one or two hybrid-clusters after first or second k-means pass only, so that there is no need to more to third pass), all the resulting hybrid-clusters will be passed as support vectors in subsequent training operation. We shall also point out the following issues that might be raised.

- The question of the number of iterations or equivalently the number of times the k-means algorithm will be triggered is legitimate. Although, it can be possible to set up a global criterion, that can be associated, for instance, to the proportion of pure clusters, or hybrid clusters generated, two concerns maybe raised against a such approach. First, the call of several passes of k-means can substantially increase the computational complexity of the overall method, which may question its usefulness as the primarily target of the whole developed approach is to reduce a such burden complexity. Second, the convergence properties are not fully lied down yet, in the sense that there is no guarantee that an increased number of iterations would lead systematically to a reduction of number of hybrid clusters or an increase in the number of pure clusters. This would require further theoretical developments. Nevertheless, the starting guess of number of clusters (k) in k-means implementation where it was set to the number of available activity classes can be questioned. This will be further investigated in sensitivity analysis section.
- The question to apply a such clustering trick on the entire dataset, instead of each incremental data chunk only, to reduce the data and finally to train the traditional SVM could also be raised. Nevertheless, this is rather difficult to perform because k-means algorithm is known to struggle to find clusters in the case of large size discrepancy (which is the case for dataset employed in this study). In this situation, finding pure clusters becomes either rare or will take a lot of computational clustering resources. Therefore, a prudent manipulation of small size data expects to bring more efficiency. Moreover, using it with incremental classification algorithm will allow us, for real-time applications, to almost process the data online as they become available.

Algorithm 1 succKmeans procedure
An example of 3 successives clustering

Require: One data chunk, Kmeans() procedure, k1,k2,k3:number of clusters to be generated in the 1st,2nd and 3rd kmeans clustering (k1, k2, k3). ▷ get all Hybrid clusters and remove pure clusters

1: $HybridClusters \leftarrow kmeans(chunk, k1)$
2: $nbC \leftarrow$ number of clusters in $HybridClusters$
3:
4: for i = 1 to nbC do ▷ get all Hybrid clusters and remove pure clusters
5: $HybridClusters2 \leftarrow kmeans(HybridClusters_i, k2)$
6: $nbC2 \leftarrow number\ of\ clusters\ in HybridClusters2$
7:
8: for j = 1 to $nbC2$ do
▷ get all Hybrid clusters and remove pure clusters
9: $HybridClusters3 \leftarrow$ kmeans$(HybridClusters2_j, k3)$
10: end for
11: end for
12: return $data\ in\ HybridClusters3$

3.2.2 Similarity based incremental SVM method

The general idea of this method is as follows: the new data chunk may contain examples very similar to the support vectors of the previous model or the same. These examples are likely to be support vectors of the new model but do not serve to evolve the hyperplane. To avoid increasing the training time and the size of the model (the number of support vectors) to no avail, we propose an algorithm that tries to detect the useless learning data in current incremental step. First, the algorithm detects similarities between new chunk data and the support vectors preserved from the previous model. Then, the support vectors that are found to have a strong similarity to data in new chunk are discarded. Similarly, we carry out the reverse operation by checking on the data in new chunk that bears strong similarity to the same support vector (many data to one support vector relationship similarity), which will again be reduced to further shorten the training phase. This reduction is performed in a way to keep only those datum that exhibits high similarity with the associated support vector. To illustrate this reasoning, let us consider the example of Fig. 7, where the similarity calculation shows that d_1 has a strong similarity to the support vector SV_6. Similarly, d_2, d_3 and d_4 have a strong similarity to SV_7. d_5 has a strong similarity to SV_9. The vectors (SV_6, SV_7 and S_9) will then be removed as per first filtering process. For the second filtering stage, we observe that SV_7 has a strong similarity with 3 new chunk data. So, we select from (d_2, d_3, d_4) the ones that have high similarity values and discard the others. The amount to be discarded can be open to debate as it balances the classification rate and training time. In the sharp scenario, we can decide to retain only one datum that yields the highest similarity score. But this bears the risk to substantially reduce the training set to make it less relevant. Alternatively, we can decide to keep a certain percentage, say P, of the total number according to their similarity scores. This reasoning is advocated in our method (where the percentage of deletion can be either fixed manually or tuned automatically according to some overall

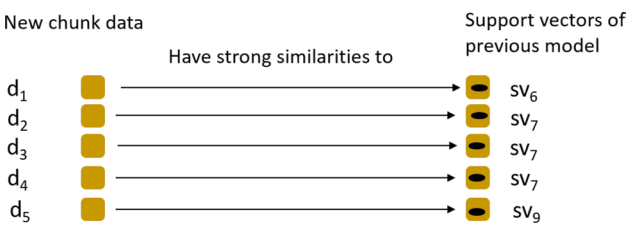

Fig. 7 Example of similarity based data reduction through incremental SVM method

criterion). The remaining dataset, after the aforementioned two deletion stages, is used to train a new model. In summary, with this method, data is eliminated in both directions. Especially, if sz is the size of a given activity class in the new chunk, it is easy to see that the size of the discarded support vectors from this activity class is less or equal than sz (In reality, often less than sz). Figure 7 and summary below illustrate how this method work. To compute the datum-support vector similarity scores, we used the Euclidean distance as in the following equation:

$$Eucl_d(d_j, sv_k) = \sqrt{\sum_{i=1}^{dm}(d_{ji} - sv_{ki})^2} \quad (5)$$

where dm is the dimension of data (and support vector as well). Reasonably, we shall consider datum d_j similar to support vector sv_i if and only if $Eucl_d(d_j, sv_k) \geqslant 0.9$

Setting a similarity threshold higher than 0.9 will result in smaller number of datums to be considered for data reduction, while setting the value too low will results in a subsequently larger number of datums.

(a) Building the initial model:

A small portion of data is used to build the initial model. Traditional SVM classifier is used for this step. The data selected to build the initial model is the first arrived data.

(b) Incremental steps (updating the model)

At each incremental step:

- New data is received (new chunk data).
- A set of support vectors are extracted from the previous model.
- Calculate similarities between each data sample in new chunk and the set of the support vectors of the class that the data sample belongs to.
- All support vectors with strong similarities to new data are removed. In the example of the Fig. 7, SV_6, SV_7 and SV_9 will be removed.
- If one support vector is found to be similar to more than one new sample data, we keep only the new data that exhibits highest similarity value. In the example of the Fig. 7, d_2, d_3 and d_4 have a strong similiraty with one support vector SV_7. We suppose that the similarity value $\sqrt{\sum_{i=1}^{dm}(d_{2i} - SV_{7i})^2} > \sqrt{\sum_{i=1}^{dm}(d_{3i} - SV_{7i})^2} > \sqrt{\sum_{i=1}^{dm}(d_{4i} - SV_{7i})^2}$. So, we keep only P% of $[d_2, d_3, d_4]$. P is a determined in order to find an optimal trade-off between classification rate and training time.
- The returned data is used to train the new model
- A test phase is applied at the end of the incremental step.

A pseudo-code version of the method is given in Algorithm.2 . We denote this technique as SimInc for future reference.

Algorithm 2 IncSim

```
Require: data chunks, number of training
    steps N, multiclassSVM() procedure, G
    = Gamma kernel parameter, Penality C,
    calculEuclideanSimilarity() procedure,
    P: number of new chunk data to be
    removed.
    ▷ Building the initial model
 1: classifier_0 ← multiclassSVM(chunk_0, Gamma, C)
                                  ▷ Incremental step
    (updating the model)
 2:
 3: for i = 1 to N-1 do
 4:    SVs ← extract support vectors from
    classifier_{i-1}                              ▷
    V contains the closet Support Vectors to
    chunk_i.D contains distance between data
    in chunk_i and V
 5:    D,V ← calculateEuclideanSimilarity(
    chunk_i, SVs)
 6:    S = unique(V)
 7:
 8:    for each support vector s in S do
 9:       [indices,num] ← occurrence number
    of s in V
10:
11:       if num > 1 then
12:          redundancyData ←
    chunk_i(indices)
13:          RD ← ascendingSort(redundancyData)
    ▷ delete the P half data in RD
14:          dataLower ← RD[1:P]
15:       end if
16:    end for
17:    presevedSVs ← delete S from SVs
18:    newChunkData ← delete dataLower from
    chunk_i
19:    trainingData_i ← presevedSVs+
    newChunkData
20:    classifier_i ← multiclassSVM(trainingData_i, G, C)
21: end for
```

Algorithm 3 IncSucc

Require: data chunks, number of training steps N, multiclassSVM() procedure,G = Gamma kernel parameter, Penality C, succKmeans() procedure, k1,k2,k3: number of clusters to be generated in the 1st,2nd,3rd clustering. ▷

Building the initial model

1: $classifier_0 \leftarrow$ multiclassSVM$(chunk_0, Gamma, C)$ ▷ Incremental step (updating the model)

2:

3: for i = 1 to N-1 do

4: SVs ← support vector from $classifer_{i-1}$

5: HybridData ← succKmeans$(chunk_i, k1, k2, k3)$

6: $\text{trainingData}_i \leftarrow$ HybridData + SVs

7: $classefier_i \leftarrow$ multiclassSVM$(\text{trainingData}_i, G, C)$

8: end for

4 Simulations and results

4.1 Datasets

To evaluate the proposed methods, we chose the Aruba and Tulum (2009) real-world datasets from the Washington State University CASAS smart-home project (Aruba and Tulum 2011). In Aruba smart home, the activities of daily living of an elderly woman were recorded. 34 motion and door sensors have been installed in the smart home. Sensor data were collected over a period of 220 days. Tulum contains sequential and simultaneous activities data of two inhabitants. 16 motion sensors are attached to objects. Sensor data were collected over a period of 4 months. Tables 1 and 2 summarize the characteristics of Aruba and Tulum datasets, respectively. We shall notice the so-called "Other events" are the ones with missing labels. This class represents a kind of an exclusion class that gathers the data that cannot be segmented or the data that have not been indexed because of inherent ambiguity that cannot be solved by the user. This covers 55% and 50 % of the entire Aruba and Tulum data sequence respectively. Therefore, missing labels activities are discarded in this work. Accuracy, F-measure, execution time and the size of the classifier are the metrics used for evaluating the effectiveness of classifiers. Accuracy shows the percentage of correctly classified instances, while the F-measure is defined as the harmonic mean of recall and precision. Precision measures the percentage of inferred activities correctly recognized while recall measures the percentage of ground truth activities correctly recognized. F-measure is important when we face a highly unbalanced' dataset as in our case. The size of the classifier is the number of resulting support vectors. For the incremental learning, the size of the data presented to the training system at each incremental step is used too to know which classifier eliminates more data.

Table 1 Aruba dataset statistics

Activity (label)	# of events
Bed-to-Toilet (1)	1330
Eating (2)	16,037
Enter-Home (3)	2018
Housekeeping (4)	10,583
Leave-Home (5)	1922
Meal-Preparation (6)	285,149
Relax (7)	354,585
Respirate (8)	542
Sleeping (9)	32,682
Wash-Dishes (10)	10,464
Work (11)	16,321
Other events (12)	871,320
# of events without other events	731,633

Table 2 Tulum (2009) data set's statistics

Activity (label)	# of events
Cook-Breakfast(1)	23,748
Cook-Lunch(2)	8915
Enter-Home(3)	597
Group-Meeting(4)	22,554
Leave-Home(5)	1574
R1-Eat-Breakfast(6)	10,395
R1-Snack(7)	95,089
R2-Eat-Breakfast(8)	12,234
Wash-Dishes(9)	24,392
Watch-TV(10)	50,250
Other events(11)	203,484
# of events without Other events	249,748

4.2 Results and discussion

To assess the data reduction performance using the clustering-based incremental SVM method, Table 3 shows the quantity of data reduced when this method is applied incrementally and non-incrementally. We applied 8 consecutive clustering processes on the entire datasets (non-incrementally) and 3 consecutive clustering processes on each data chunk (incrementally), where about two-third of the initial training dataset has been red.

Next, we conducted two experiments sets. The first set aims to evaluate the features extraction methods, presented

in Sects. 3.1 and 3.1.3. SVM traditional (batch algorithm) is used to train activity models. While the second set (reported in Sect. 4.5) aims to evaluate the incremental SVM methods, in which the features extracted by the Last-state sensor based method LS were used. We choose this feature method to evaluate the incremental learning classifiers since it outperforms the other methods and it does not require any offline calculation unlike the dependency sensor based features extraction method, which requires calculating the dependency matrix offline (in the presence of all data). The one vs. one SVM paradigm is used for learning activity models, since it is the appropriate strategy to counter unbalanced data and because it is the multiclass method that has the best ratio between efficiency, rapidity and global performances as highlighted in Krishnan and Cook (2012). A radial basis function kernel was used with a width parameter of 1. The penalty parameter C was set to 100. Training data samples were normalized before being presented to the classifier (explaining the width parameter). Whatever the learning type (batch, incremental), 75% and 25% of data are used for training and testing, respectively. The data test remains the same in both traditional and incremental algorithms. Algorithms presented in this paper are coded in python. SciKit Learn package is used to train the initial SVM models Pedregosa et al. (2011). The Single CPU(2.5 GHz) is used in the evaluation with 8 GB of RAM.

4.3 Discussing results obtained by the batch Learning

We began our experiment by testing BL feature extraction method with different number of events per window. We obtained the best performances (classification) with a number that is lower than the average number of sensor events that span all the instances of activities in the entire data. Then, we tested DS approach Krishnan and Cook (2012) and our two proposed feature extraction approaches MDS and LS. Table 5 summarizes the results obtained from the two datasets.

The *LS* proposed feature method outperforms the other methods over the two datasets. As it is shown in Table 4, the F-measure, Accuracy scores resulted from the proposed

Table 3 Process of reducing training data using successive clustering method

Datasets	Non-incrementally		Incrementally	
	Aruba	Tulum	Aruba	Tulum
Training data	671,407	341,880	631,164	305,562
Deleted training	152,089 (23%)	96,869 (29%)	422,594 (67%)	170,700 (56%)

Table 4 Batch SVM: evaluation of the features extraction methods. **MDS** and **LS** are the proposed methods

Datasets	Aruba				Tulum			
	BL	DS	MDS	LS	BL	DS	MDS	LS
Accuracy (in%)	90.14	88.00	91.37	**93.00**	54.96	72.06	72.96	**73.56**
F-measure (in %)	63.81	60.71	65.88	**67.06**	44.89	53.98	54.65	**57.37**
Training time (h)	40.56	34.32	47.28	73.44	1.24	1.00	1.08	1.25
# of Support vectors	92220	106968	99289	87091	92433	92974	90461	92467

Table 5 Batch SVM: individual F-measure (In percentages)

Activity label	Aruba				Tulum			
	BL	DS	MDS	LS	BL	DS	MDS	LS
1	62.33	42.95	**87.25**	85.87	66.79	66.03	67.06	**67.93**
2	74.17	62.75	**74.17**	70.14	**31.19**	12.185	16.15	15.67
3	**76.92**	76.18	75.24	68.22	39.89	37.17	39.40	**43.84**
4	34.36	22.72	32.97	**41.92**	72.90	78.39	80.06	**80.20**
5	33.98	52.19	56.86	**63.68**	57.50	58.23	54.38	**58.58**
6	89.56	89.61	90.96	**93.03**	14.16	04.16	05.21	**16.14**
7	93.29	92.39	94.45	**96.01**	69.40	78.21	78.99	**79.05**
8	37.04	**49.54**	19.82	20.00	6.39	44.90	45.92	**46.32**
9	96.48	90.97	96.71	**97.48**	0.21	51.54	51.27	**56.44**
10	0.00	0.00	0.00	0.00	0.64	85.60	86.72	**87.34**
11	81.39	7.58	86.08	**88.78**				

methods are higher than *BL* and *DS* methods, for most of the activities. For instance, in the case of Aruba dataset, the *LS* proposed feature extraction method achieves an F-measure of 67.06%, outperforming *BL* and *DS* with F-measure 63.81% and 60.71%, respectively. Similarly, in the case of Tulum dataset, *LS* achieves an F-measure of 57.37%, whereas *BL* and *DS* yield an F-measure of 44.89% and 53.98%, respectively. On the other hand, our proposed *MDS* method aims to correct some errors classification that can occur using the *DS* features method. Although, it has done so successfully, it still show a less performance than of the proposed *LS* feature extraction method.

When looking at the result from a single activity task, Table 5 shows individual F-measure score for each activity performed in the two smart homes. For Aruba dataset, it reveals that no method could identify the dishes(10) activity, where most of its test instances are identified as Meal-prepartion(6) activity. This is because the two activities run in the same location and trigger the same sensors.

Likewise, most of the test instances of the Test-to-Bed(1) activity were correctly identified by the proposed methods *MDS* and *LS*. This is not the case with *BL* and *DS* state-of-the-art methods. The *DS* method identified correctly the most test instances of the Respirate(8) rare activity; however, it provided a lower performance of recognizing Work(11) activity. This is mainly because of its misclassification as Relax(7) activity. Classifier under our proposed *LS* feature method yields the smaller number of resulting support vectors (Not-sparse solution). However it took more time to train activity models (73 h, about 3 days), since the *LS* feature matrix is a not sparse matrix as opposed to other feature matrices.

As for Tulum smart home, inhabitants performed theirs activities simultaneously. Classifier with *BL* feature method failed in recognizing R2-Eat-Breakfast(8), Wash-Dishes(9) and Watch-TV(10) activities. These activities run generally simultaneously in the smart home and a simple count of trigger sensors was not enough to identify them. The three other methods could identify them in varying proportions. In the average, training activity models lasted 48.9 and 1.14 h for Aruba and Tulum datasets, respectively. This seems quite high for Aruba dataset.

4.4 Sensitivity analysis of the parameters

In this section we shall analyze the sensitivity of the developed approach with respect to the two key identified parameters that impact the construction of the developed models as well number of clusters in clustering based incremental learning method. The first parameter consists of the window size n in the modified dependency sensor feature extraction method (MDS). While the second parameter corresponds to the proportion P, in the similarity based incremental SVM method (IncSim), which controls the extent to which datums are preserved when they judged similar to the same support vector from the previous incremental step. In both cases, we evaluated the effect of these parameters on the performance of the model when assessed using F1-measure. The choice of the F1-measure metric is justified by its good summarization of the overall performance of the system as it explicitly takes into account both the precision and recall performance metrics. We varied the parameter n from 1 till 10. The use of higher values seems not practical as at some incremental steps, the number of event sensors available cannot be that high. Similarly, we varied the ratio P from 5% till 100 %. The latter corresponds to the case where all datums are considered in the training even if some are highly similar to the same support vector from previous increment step. Figure 8 exhibits the overall performance using batch-algorithm when using MDS feature for different values of n for both Aruba and Tulum dataset. The plot shows that choosing a size window $n = 4$ and $n = 5$ provide best performance for Aruba and Tulum dataset, respectively.

Similarly, Fig. 9 exhibits the overall F1-measure performance of similarity-based incremental SVM method for different values of proportion P for both Aruba and Tulum datasets. The plot indicates that proportions $P = 45\%$ and $P = 40\%$ achieve the best F1 performance, in Aruba and Tulum dataset, respectively. This indicates that the best performance is rather achieved when we discard nearly half of the datums that are found similar to the same support vector in similarity-based incremental SVM method. This also shows that a prudent attitude should be followed when deciding to prone the initial data as a too small training sample may not be robust enough to capture all data variability, which yields negative consequences. Next, we attempt to

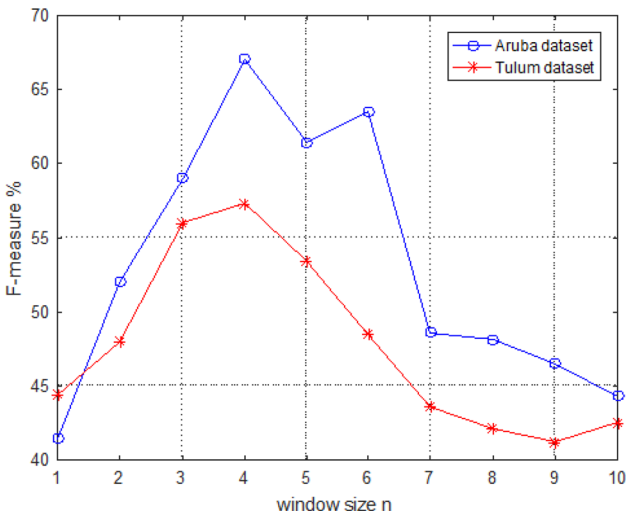

Fig. 8 Sensitivity with respect to size of window in MDS feature extraction method

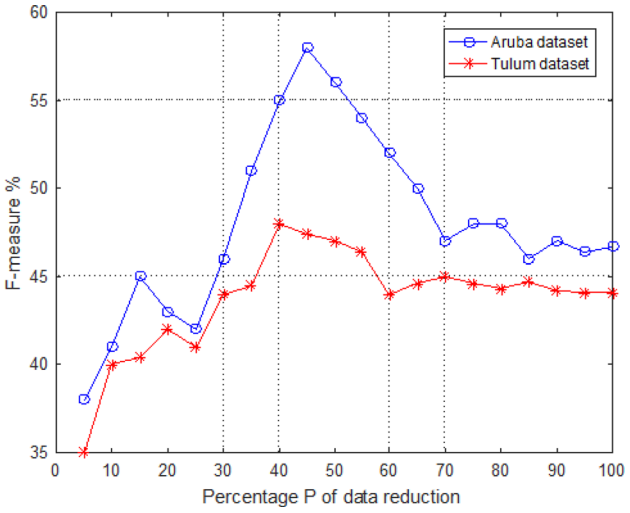

Fig. 9 Sensitivity with respect to proportion of deletion P in the similarity based incremental SVM

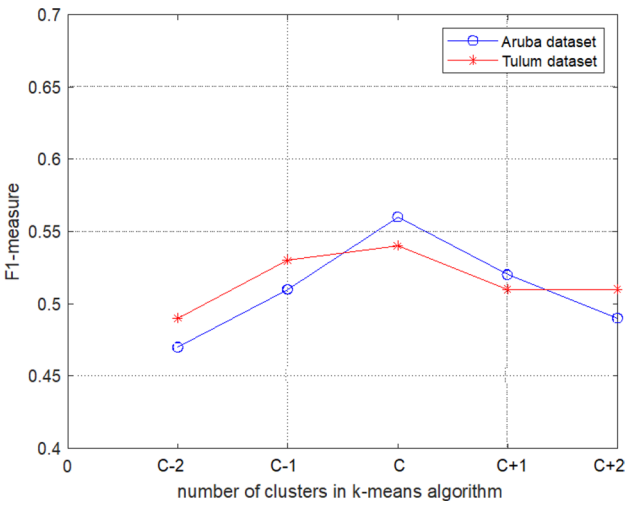

Fig. 10 Sensitivity with respect to number of cluster in IncSucc method

investigate the sensitivity of the clustering-based incremental SVM method with respect to the initial seed of the number of clusters k in the first application of k-means algorithm. For this purpose, and for practicality considerations, we restrict the choice of the value of k close to the number of activity classes C available at the given data chunk. Specifically, we took a window of size five around C, so that we test, whenever available, scenarios, $k = C + 2$, $k = C + 1$, $k = C$, $k = C - 1$, $k = C - 2$.

Figure 10 shows the average variations (over all increment steps) of F1-measure with respect to each choice of cluster seed value when LS feature is employed. The reading of the result exhibited in the plot indicates a slight preference of the case where the k-means algorithm is initialized with a number of clusters taken equal to the number of activities $k = C$.

4.5 Discussing results obtained by the incremental learning

In this set of experiments, our objective is to compare the proposed incremental SVM methods with those of the state-of-the-art on one hand and to compare incremental and traditional SVM learning, on the other hand. We evaluated three state-of-the-art incremental SVM learning (*Fixed*, *ERRD* and *LSSVF*) and our two proposed methods (*IncSim* and *IncSucc*). In this experiment the features extracted by the Last-state sensor based method *LS* were used. We choose this feature method to evaluate incremental learning classifiers since it outperforms the other methods and it does not require any offline calculation as the dependency sensor based features extraction method which requires calculating the dependency matrix offline. The datasets were divided into chunks of 1 day data. Data of the first 6 days were then used to train the initial model (classifier at the step 0). These data were retrieved from the original dataset using timestamp information. The values of RBF and C penalty parameters were chosen equal to the ones used for the traditional algorithms (default parameters in baseline SVM model). For *LSSVF* technique, we obtained the best results when 20% of the least significant support vectors were discarded. This corresponds to a trade-off between accuracy and training time. For the *IncSucc* proposed method, three successive k-means processes were applied. For the *IncSim* proposed method, we obtained the best results when P, in Algorithm.2, is close to 50%, i.e, 50% of data in new chunk are removed at each incremental step when they are found similar to the same support vector (this is highlighted in sensitivity analysis section as well). Table 6 summarizes the results obtained from the two datasets. The bold values represent the best results. Figures 11 and 12 illustrate the evolution of Accuracy and F-measure over incremental steps, while Fig. 12 shows the amount of training data fed into the classifier at each incremental step. We note that the Tulum dataset is not a large scale data; however, it is used here to evaluate our methods. Two Times are reported in the Table 6. The first one is the completion time taken by the system for training all chunks data, i.e, the sum of training time taken by the system for training each chunk data. The second one is the average time (over successive incremental steps) taken by the system to learn one model. We report these two Times because we can either use incremental learning for online training by using the resulting model at each incremental step for recognition or for speeding up the training phase by using only the last incremental step model for recognition. Then we

report the average Accuracy and F-measure over successive incremental steps as well as the number of support vectors and the classification performances of the last incremental step. The model of the last incremental step is considered as the solution to which the incremental algorithm converges.

Regarding Aruba dataset, one notices the following. The Fixed method is the one that uses all the chunk data together with the previous model support vectors to learn a new model. It exhibits the best performances in terms of Accuracy and F-measure (90.5 % and 61.5 %) but it takes longer time to train the algorithm, among the other presented incremental methods. For instance, it takes 53 h to train the models of all steps, an average of 19.87 min to train one model. This is almost impractical since we can not use it to speed up the algorithm and, even in the online setting its time complexity increases over steps (it trains last model in 1 h).

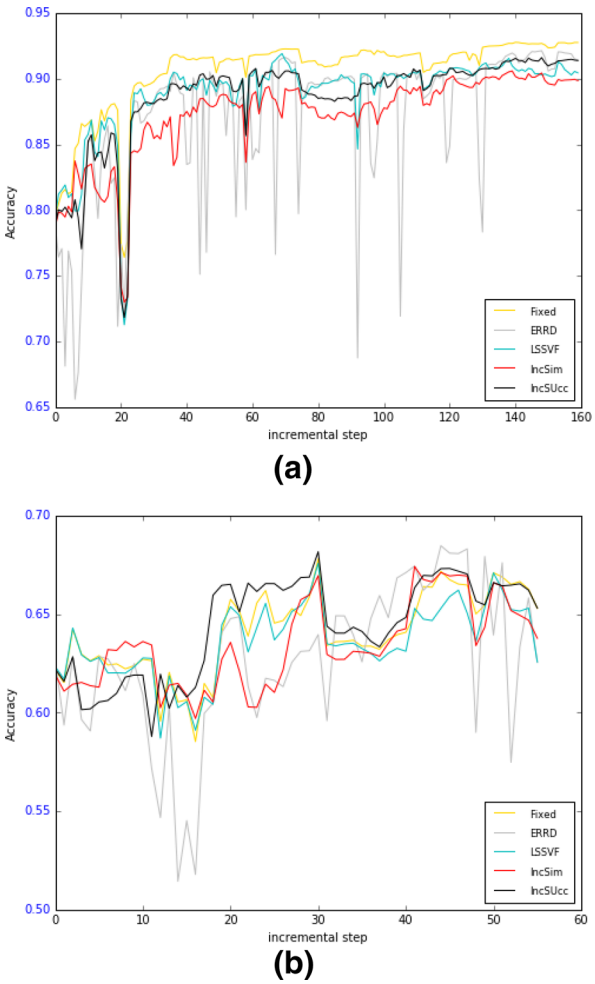

Fig. 11 Accuracy for each incremental step. **a** Aruba dataset; **b** Tulum dataset

The *ERRD* method has less learning time compared to *Fixed* method. It takes 21.06 h to train models of all steps, and an average of 7.90 mins to train one model. Despite the fact that the F-measure for the last step model is close to that obtained by *Fixed* method, *ERRD* method still suffer from the instability of its classification performances. Indeed, when an incremental step yields a high recognition rate, in the next step, the rate can drop significantly as it can be seen in Fig. 11. This is mainly due to its limited filtering process as was discussed in Sect. 2.2. Their instability makes it unfit to work online.

The *LSSVF* method discards a part of support vectors of the previous model. It runs faster than the two first methods and the *IncSucc* proposed method, but it yields lower F-measure scores. This is because discarding support vectors process does not consider imbalanced data and is rather considered as a resampling procedure on majority and minority classes. This partly explains why it has lower F-measure values.

The *IncSucc* proposed method runs faster than the *Fixed* and *ERRD* state-of-the-art methods. It takes 14.51 h to train models of all steps, an average of 5.44 min to train one model. It generated classification performances better than all other methods, if we exclude the *Fixed* method.

The proposed *IncSim* method is 5 to 9 times faster than other methods presented here and yields an almost a horizontal line in the training data (constant value) as shown in Fig. 13. It takes 3.56 h to train models of all steps, and an average of 1.3 mins to train one model. It shows a classification performances better than those obtained by *ERRD* and *LSSVF* state-of-the-art methods. We observe that by evaluating Aruba dataset, all classifier methods (except *Fixed*

Table 6 The performance measures of incremental methods. T1:Time to train all chunks. T2: Time Average to train one model. A.: Average; L.M: Last Model

	Aruba					Tulum				
Metrics	Fixed	ERRD	LSSVF	IncSucc	IncSim	Fixed	ERRD	LSSVF	IncSucc	IncSim
T1 (h)	53.00	21.06	9.66	14.51	**3.56**	6.84	3.04	3.87	3.36	**0.715**
T2 (minutes)	19.87	7.90	3.62	5.44	**1.33**	7.33	3.25	4.15	3.19	**0.715**
A. Accuracy	**90.5**	87.14	88.83	**88.87**	87.19	64.08	62.58	63.61	**64.34**	63.42
A. F-measure	**61.5**	57.64	54.98	**58.45**	57.10	50.09	49.04	48.49	**51.04**	50.62
L.M Accuracy	**92.76**	91.45	90.44	**91.44**	89.85	65.29	62.55	62.57	**66.22**	63.78
L.M F-measure	**64.31**	**62.17**	51.01	58.78	58.04	56.29	55.85	52.15	**57.06**	54.74
# of SVs L.M	64580	41477	21519	33189	**15280**	59914	39356	46214	50957	**27034**

method) has a lower execution time than those of batch algorithm runs.

In summary, the *IncSim* is the only method where the training time of all chunk data of Tulum dataset is less than the training time taken by the batch algorithm. In fact, it is expected from this method to perform well when the distribution of data changes over time, since it discards at each step a set of selective support vectors. Interpretations of Aruba results are still valid for Tulum. The proposed *IncSucc* method outperforms all other methods including the Fixed method, in term of Accuracy and F-measure.

To recap, the *IncSucc* proposed incremental method speeds up significantly the training time while it keeps the classification performances close or better than those obtained by state-of-the-art methods and it is expected to perform well when the distribution in data change over time.

5 Conclusion and future work

In order to provide a prompt service for dependent people in their own home, a real time system recognizing daily human activities from sensor readings is required. Most of the techniques used in the literature have inherent limitations due to the high execution time or to the constraints governing the model construction that are imposed by the employed classification method. In this paper, we propose and evaluate an extension of a sensor window approach to perform activity recognition in a streaming way, i.e. recognizing activities when a new sensor event is recorded. As our experiments deal with large scale dataset, training data offline become impractical. For this, we introduced two new incremental SVM techniques as extension to Krishnan and Cook (2012). The first one refines the dependency sensor feature extraction method by re-interpreting the concept of mutual information between two sensors as the probability that the two sensors

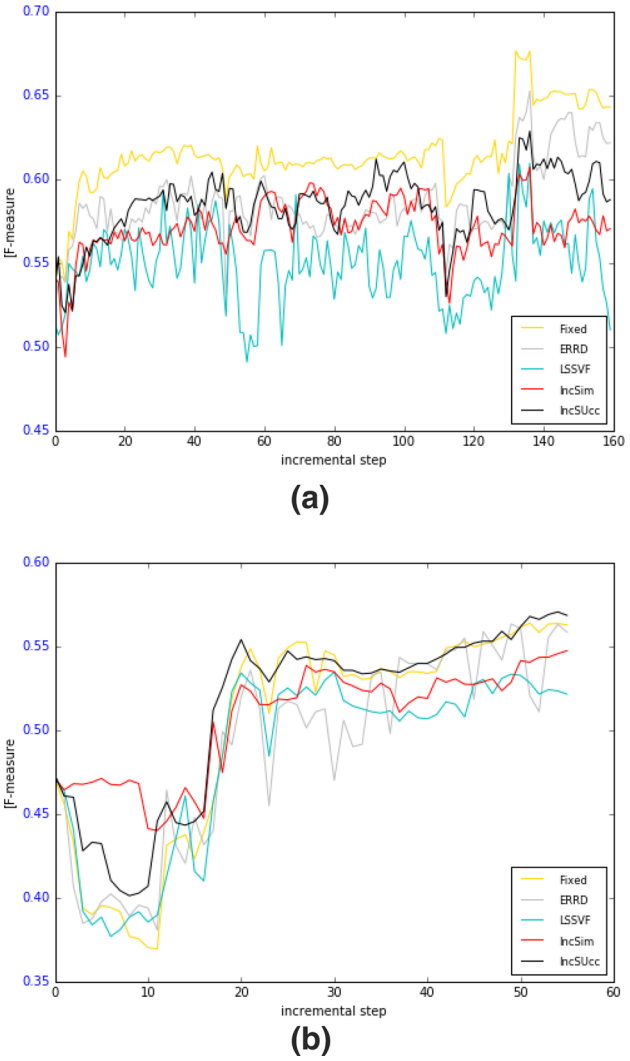

Fig. 12 F-measure for each incremental step. **a** Aruba dataset; **b** Tulum dataset

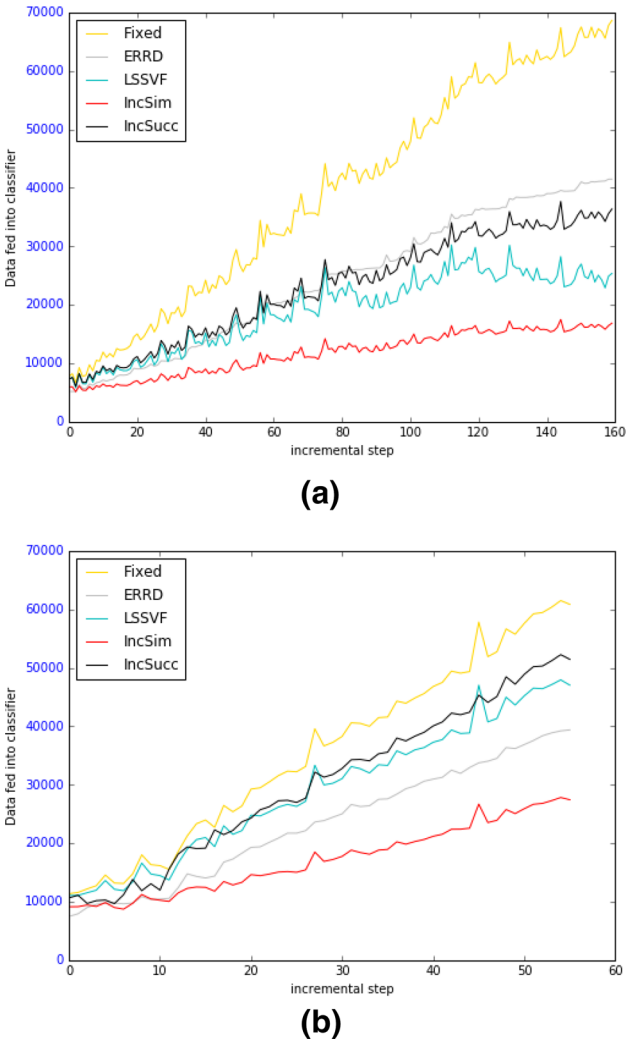

Fig. 13 Size of data fed into classifier at each incremental step. **a** Aruba dataset; **b** Tulum dataset

fall on the same windows (of a fixed size) of events. The second one acknowledges the importance of the last-state of the sensor within a given segment so that its feature representation is reduced to the status of this last event of the segment, which significantly boosts the computational efficiency of the feature extraction method. Next, we also proposed two candidate solutions for SVM incremental learning methods by capitalizing on the similarity between support vectors of previous data chunk and current batch of data to reduce the training phase. The first method builds on the concept of pure and hybrid cluster to apply iterative k-means algorithms so that datum associated with pure clusters are discarded and those with hybrid clusters are added to support vector list. The second method suggests to prone the training datum at each data chunk using a two-side similarity calculus process, through an Euclidean metric, where support vectors that are found highly similar to data in the new chunk are discarded, and then, datums that bear similarity to the same support vector are reduced. The evaluation of the developed approached have been carried out using publicly available Aruba and Tulum dataset. The results were also compared to the state-of-the-art methods; namely, Fixedm ERRD and LSSVF methods. The findings confirm both the feasibility and the high performance of the proposals in terms of accuracy, F1-score and data simplification. First, from the feature engineering perspective, and using baseline SVM method reveals that the introduced Last-State sensor based method outperforms all other feature extraction methods, including the introduced MDS approach. Second, the proposed clustering based-incremental methods are found to run faster than the Fixed and ERRD state-of-the-art methods. Especially, the proposed similarity based incremental learning is found to be 5 to 9 times faster than other presented methods, while achieving good performance in terms of F1-measure and accuracy as well. As perspective work, the detailed convergence properties of the introduced incremental learning algorithms are still to be investigated. This would provide a solid theoretical framework for further development of empirical research in human activity recognition. On the other hand, the detailed investigation of the effect of imbalanced class dataset is part of our planned future investigations as the currently developed method would fail if high class imbalance occurs. We hypothesize that data segmentation into small chunks would constitute a first asset to tackle a such phenomenon. Therefore, re-examining other related datasets with complex class distribution is pivotal. Another challenge is to update the model at each incremental step without system retrain and then to create a real-world model that can be used and tested online in a smart home. This will serve as input to some profiling system to create some optimized version tailored to each individual Karami et al. (2016). This will be investigated as part of perspective work.

Acknowledgements This work is partly supported by European project YoungRes (#823701), and Academy of Finland DigiHealth projects No.326291 which are gratefully acknowledged.

Author Contributions YN has performed the initial implementation and produced draft under supervision of BF. MO reshaped the draft and checked overall implementation. AF interacted with BF to extent their initial framework of incremental SVM.

Funding Open Access funding provided by University of Oulu including Oulu University Hospital. This work is partly supported by European project YoungRes (#823701), and Academy of Finland DigiHealth projects No.326291 which are gratefully acknowledged.

Declarations

Conflict of interest None.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Availability of data and materials All employed dataset are open sources

Code availability. Code can be made on reasonable request.

References

Aruba and Tulum (2011) Aruba dataset from wsu casas smart home project

Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P (1999) Towards a better understanding of context and context-awareness. In: Gellersen H-W (ed) Handheld and Ubiquitous Computing, pp 304–307, Springer, Berlin

Allameh E, Heidari Jozam M, Vries B. d, Timmermans H, Beetz J (2011) Smart home as a smart real estate: a state of the art review. In: 18th International Conference of European Real Estate Society, Eindhoven, The Netherlands Eindhoven: ERES

Bao L, Intille SS (2004) Activity recognition from user-annotated acceleration data. In: Pervasive computing, pp 1–17, Springer, Berlin

Barger TS, Brown DE, Alwan M (2005) Health-status monitoring through analysis of behavioral patterns. Trans Syst Man Cyber Part A 35(1):22–27

Cauwenberghs G, Poggio T (2001) Incremental and decremental support vector machine learning. In: Advances in neural information processing systems (NIPS*2000), volume 13

Cook DJ, Khrishnan N (eds) (2015) Activity learning, discovering, recognizing and predicting, Wiley, London, 1st edn

Demrozi F, Pravadelli G, Bihorac A, Rashidi P (2020) Human activity recognition using inertial, physiological and environmental sensors: a comprehensive survey. IEEE Access 8:210816–210836

Domeniconi C, Gunopulos D (2001) Incremental support vector machine construction. In: IEEE international conference, pp 589–592

Fan R-E, Chen P-H, Lin C-J (2005) Working set selection using second order information for training support vector machines. J Mach Learn Res 6:1889–1918

Fu B, Damer N, Kirchbuchner F, Kuijper A (2020) Sensing technology for human activity recognition: a comprehensive survey. IEEE Access 8:83791–83820

Gu T, Wu Z, Tao X, Pung H. K, Lu J (2009) epsicar: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition. In: 2009 IEEE international conference on pervasive computing and communications, pp 1–9

Gâlmeanu H, Andonie R (2008) Implementation issues of an incremental and decremental svm. In: Artificial Neural Networks - ICANN. Lecture Notes in Computer Science, vol 5163, Springer, Berlin

Hsieh C.-J, Si S, Dhillon IS (2014) A divide-and-conquer solver for kernel support vector machines. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, pages 566—-574, China. JMLR

Karami AB, Fleury A, Boonaert J, Lecoeuche S (2016) User in the loop: Adaptive smart homes exploiting user feedback – state of the art and future directions. Information 7(2):35

Kim S-C, Jeong Y-S, Park S-O (2013) Rfid-based indoor location tracking to ensure the safety of the elderly in smart home environments. Person Ubiquit Comput 17(8):1699–1707

Krishnan NC, Cook DJ (2012) Activity recognition on streaming sensor data. Pervasiv Mob Comput, pp 1–17

Kuhn HW, Tucker AW (1951) Nonlinear programming. In: Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1950, number 0047303 (13,855f), pages 481–492, Berkeley and Los Angeles

Lester J, Choudhury T, Kern N, Borriello G, Hannaford B (2005) A hybrid discriminative/generative approach for modeling human activities. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, pages 766–772

Liu J, Liu H, Chen Y, Wang Y, Wang C (2020) Wireless sensing for human activity: a survey. IEEE Commun Surv Tutor 22:1629–1645

Ordóñez FJ, de Toledo P, Sanchis A (2013) Activity recognition using hybrid generative/discriminative models on home environments using binary sensors. Sensors 13(5):54–60

Osuna E, Freund R, Girosi F (1997) An improved training algorithm for support vector machines. In: Neural Networks for Signal Processing VII. Proceedings of the 1997 IEEE Signal Processing Society Workshop, pages 276–285

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Polikar R, Upda L, Upda SS, Honavar V (2001) Learn++: an incremental learning algorithm for supervised neural networks. Trans Syst Man Cybern Part C 31(4):497–508

Pollack ME, Brown L, Colbry D, McCarthy CE, Orosz C, Peintner B, Ramakrishnan S, Tsamardinos I (2003) Autominder: an intelligent cognitive orthotic system for people with memory impairment. Robot Auton Syst 44(3):273–282

Pronobis A, Luo J, Caputo B (2010) The more you learn, the less you store: memory-controlled incremental svm for visual place recognition. Image and vision computing

Razzaghi T, Safro I (2015) Scalable multilevel support vector machines. Procedia Comput Sci 51(C):2683–2687

Schlag S, Schmitt M, Schulz C (2021) Faster support vector machines. 26(15)

Solaimani S, Keijzer-Broers W, Bouwman H (2013) What we do–and don't–know about the smart home: an analysis of the smart home literature. Indoor Built Environ, pp 1420326X13516350

Straczkiewicz M, James P, Onnela J (2021) A systematic review of smartphone-based human activity recognition methods for health research. npj Digital Medicine, 148

Syed NA, Liu H, Sung KK (1999) Incremental learning with support vector machines. In: Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden

Tapia EM (2003) Activity recognition in the home setting using simple and ubiquitous sensors. PhD thesis, Massachusetts Institute of Technology

Tapia EM, Intille S, Larson K (2004a) Activity recognition in the home using simple and ubiquitous sensors. In: In Pervasive, pp 158–175

Tapia EM, Intille SS, Larson K (2004b) Activity recognition in the home using simple and ubiquitous sensors, pp 158–175

van Kasteren T, Noulas A, Englebienne G, Kröse B (2008a) Accurate activity recognition in a home setting. In: Proceedings of the 10th international conference on ubiquitous computing, UbiComp '08

van Kasteren T, Noulas A, Englebienne G, Kröse B (2008b) Accurate activity recognition in a home setting. In: Proceedings of the 10th international conference on ubiquitous computing, UbiComp '08, pages 1–9

Vischer JC (2007) The concept of environmental comfort in workplace performance. Ambiente Construído 7(1):21–34

Wang L, Gu T, Tao X, Chen H, Lu J (2011) Recognizing multi-user activities using wearable sensors in a smart home. Pervasive Mob Comput 7(3):287–298

Wang L, Gu T, Tao X, Lu J (2012) A hierarchical approach to real-time activity recognition in body sensor networks. Pervasive Mob Comput 8(1):115–130

Wen Z, Shi J, Li Q, He B, Chen J (2018) Thundersvm: a fast svm library on gpus and cpus. J Mach Learn Res 19(1):797–801

Wilson C, Hargreaves T, Hauxwell-Baldwin R (2014) Smart homes and their users: a systematic analysis and key challenges. Pers Ubiquit Comput 19(2):463–476

Wilson DH, Atkeson C (2005) Simultaneous tracking and activity recognition (star) using many anonymous, binary sensors. In: Proceedings of the Third International Conference on Pervasive Computing, PERVASIVE'05, pp 62–79

Yu H, Yang J, Han J (2003) Classifying large data sets using svms with hierarchical clusters. page 306–315, Washington, D.C. Association for Computing Machinery