Expert Systems With Applications 185 (2021) 115641

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Embedding-based real-time change point detection with application to activity segmentation in smart home time series data

Unai Bermejo [a], Aitor Almeida [a,*], Aritz Bilbao-Jayo [a], Gorka Azkune [b]

[a] DeustoTech - University of Deusto, Avenida de las Universidades 24, 48007 Bilbao, Spain
[b] IXA NLP Group, Faculty of Computer Science, Euskal Herriko Unibertsitatea (EHU/UPV), M. Lardizabal 1, 20008 Donostia, Spain

ARTICLE INFO

Keywords:
Activity transition detection
Change point detection
Activity segmentation
Smart homes
Action embeddings
Sensor embeddings

ABSTRACT

Human activity recognition systems are essential to enable many assistive applications. Those systems can be sensor-based or vision-based. When sensor-based systems are deployed in real environments, they must segment sensor data streams on the fly in order to extract features and recognize the ongoing activities. This segmentation can be done with different approaches. One effective approach is to employ change point detection (CPD) algorithms to detect activity transitions (i.e. determine when activities start and end). In this paper, we present a novel real-time CPD method to perform activity segmentation, where neural embeddings (vectors of continuous numbers) are used to represent sensor events. Through empirical evaluation with 3 publicly available benchmark datasets, we conclude that our method is useful for segmenting sensor data, offering significant better performance than state of the art algorithms in two of them. Besides, we propose the use of retrofitting, a graph-based technique, to adjust the embeddings and introduce expert knowledge in the activity segmentation task, showing empirically that it can improve the performance of our method using three graphs generated from two sources of information. Finally, we discuss the advantages of our approach regarding computational cost, manual effort reduction (no need of hand-crafted features) and cross-environment possibilities (transfer learning) in comparison to others.

1. Introduction

The development of systems capable of monitoring human activities in smart homes is crucial in order to build many human-centered applications that are used in a wide range of subjects, from preventive medicine and security to sustainability and entertainment (Ranasinghe, Al Machot, & Mayr, 2016). More concretely, in the field of healthcare and ambient assisted living, these activities are often called Activities of Daily Living (ADLs) and they are used by health professionals to measure the functional status of dependent patients, essentially elderly people and persons with disabilities (Bennett, Rokas, & Chen, 2017). Their monitoring also serves to perform medical interventions to people with risk conditions (e.g. diabetes) and to evaluate the evolution of diseases, including their early detection, such as cancer, Alzheimer's or depression.

These activity-aware systems are often composed by low-cost sensors that capture the actions performed by users over time. With this information, those systems can recognize activities by using different types of algorithms. Likewise, it is possible to employ cameras to gather images or videos and recognize the activities being carried out (Weinland, Ronfard, & Boyer, 2011). However, vision-based solutions are not that popular, as they tend to generate privacy concerns (Chernbumroong, Cang, Atkins, & Yu, 2013; Yilmaz, Javed, & Shah, 2006).

When sensors are used to collect environmental information, those systems must accomplish the task of recognizing activities from a stream of sensor events. Without exception, in real applications, these sensor events are unsegmented. Hence, the first step in human activity recognition (from now on, HAR) is to segment sensor events in order to extract features and recognize the ongoing activities.

This is a demanding task, since the activities do not always have the same number of sensor events, nor do they always have to present them in the same order. User behavior is difficult to predict. For example, to prepare breakfast, different sensors can be activated (different actions can be performed) and they do not always have to be activated in the same order. One day, the person may open the refrigerator, take out a brick of milk and activate the coffee maker to prepare a milky coffee. On another day, perhaps the brick of milk is out of the fridge and the person only has to turn on the coffee maker, or does not feel like making coffee and prepares a juice. This leads to a very varied,

* Corresponding author.
E-mail addresses: unai.bermejo@deusto.es (U. Bermejo), aitor.almeida@deusto.es (A. Almeida), aritzbilbao@deusto.es (A. Bilbao-Jayo), gorka.azcune@ehu.eus (G. Azkune).

https://doi.org/10.1016/j.eswa.2021.115641
Received 1 December 2020; Received in revised form 11 June 2021; Accepted 19 July 2021
Available online 24 July 2021

rich and complex stream of sensor events to segment and from which recognize activities.

There exists multiple approaches to perform activity segmentation. A very promising technique that has recently begun to be used is change-point detection (CPD) (Aminikhanghahi & Cook, 2017a). In this approach, different algorithms are used to detect activity transitions (change points) that mark the start and end of activities. Capturing such starts and ends makes possible to separate the sensor events in non-overlapping segments that encapsulate all the information of the activity being carried out, without introducing information from others activities, what can ease the job of the recognition module on both inference and training.

In this sense, segmenting sensor events with CPD algorithms can improve the recognition performance of many classifiers (Aminikhanghahi & Cook, 2019), in comparison to the most popular alternative, the window-based approach, which separates the data in overlapping or non-overlapping segments of fixed size. Moreover, detecting activity transitions enables many applications on its own like timed notification systems or automatic intervention systems that can be deployed in homes to address health risks (Sprint, Cook, Fritz, & Schmitter-Edgecombe, 2016).

In this paper, we propose an unsupervised novel methodology based on Mikolov, Sutskever, Chen, Corrado and Dean (2013) popular Word2vec embedding models to detect activity transitions and perform activity segmentation. This methodology relies on representing sensor events with embeddings and computing similarities between them to find activity transitions in real-time. We propose three alternatives to compute such similarities. Through empirical evaluation, we demonstrate the usefulness of every alternative to detect activity transitions in three benchmark datasets.

Furthermore, we glimpse several advantages of our method in comparison to other CPD-based activity segmentation algorithms. First, our method is simple, easy to interpret and more efficient in terms of computational cost. Second, it does not need hand-crafted features, as sensor embeddings, generated in a semi-supervised fashion, are used as features. Third, it is ϵ-real-time configurable, what permits to balance the model between delay and accuracy depending on the application requirements (see Section 2, Related Work). Fourth, the representation of sensor events with embeddings opens the possibility to perform direct transfer learning between environments for activity segmentation.

In addition, as we believe that our methods require high-quality embeddings to function properly, we propose to use a hybrid technique called retrofitting (Faruqui et al., 2014) to post-process sensor embeddings. This graph-based technique allows us to introduce expert knowledge and improve their representational capabilities with barely any effort. We show empirically that our CPD methods' performance improves after adjusting the embeddings with this approach, employing different knowledges graphs generated from two distinct sources of information. Our approach is inspired by the past work of other authors (Triboan, Chen, Chen, & Wang, 2017, 2019) who have also explored the usage of external information to tackle the segmentation problem.

The rest of the paper is organized as follows. Section 2 summarizes the related work on activity segmentation and CPD. Section 3 explains our approach. Section 4 shows the experimental results and Section 5 discusses them. Finally, Section 6 presents our conclusions and lines of future work.

2. Related work

Common HAR approaches employ a fixed window length to sequentially segment the incoming sensor events. This technique has the problem of determining the optimum size of the window. For small values, activity information can be lost, making the task of activity recognition difficult. For large values, more than one activity's information can be part of the same segment, which creates a classification error in the first step of the process. To overcome this problem, some authors have explored adaptive, dynamic window size methods (Laguna, Olaya, & Borrajo, 2011; Okeyo, Chen, Wang, & Sterritt, 2014). These methods try to find significant events that mark the beginning and the end of one activity, so the window size is adjusted to fit those boundaries. However, as in real scenarios activities may overlap and erratically transition from one to another, these approaches still encounter too many issues to segment sensor data accurately.

Another classical approach to sensor event segmentation is to employ frequent pattern mining techniques. These methods try to find patterns by iterating multiple times across the data, which can be used later to recognize activities (Cook, Krishnan, & Rashidi, 2013; Rashidi & Cook, 2013). Although they show good results for multiple datasets, they are completely offline. As in this paper we focus on real-time segmentation, no further mention will be made to this type of algorithms.

Authors in the past had also augmented their solutions for event segmentation using additional semantic information. In Triboan et al. (2017, 2019), authors use domain knowledge for discerning sensor data into multiple threads of activity of daily living. The authors use ontologies to perform terminology box and assertion box reasoning, along with logical rules to infer whether the incoming sensor event is related to a given sequences of the activity.

In contrast to window-based segmentation, CPD-based segmentation has to deal with the task of detecting state transitions.

A time series data stream $S = \{x_1, \dots, x_i, \dots\}$ can be defined as an infinite sequence of data points indexed in time order, where x_i is a vector of arbitrary dimension. They are commonly used to describe the current state of systems and processes. When the data points do not differ or change during a certain period, it can be assumed that the process remains in the same state during such period. Two consecutive, distinct states are separated by a change point. In smart home time series data, we can translate state to activity, data point to sensor event and change point to activity transition.

Therefore, in this approach, the activity segmentation is performed by detecting change points that mark the start and end of activities (transitions). Fig. 1 shows the differences between the window-based segmentation approach and the CPD-based approach.

CPD can be performed in real-time or offline. To define how real-time is an algorithm, the concept of ϵ-real-time algorithm is used in the literature (Aminikhanghahi & Cook, 2017a). An algorithm is said to be ϵ-real-time when it makes a decision at time t after a delay of ϵ data points. In other words, and as explained by Aminikhanghahi, Wang, and Cook (2018), a ϵ-real-time algorithm needs to inspect data points $x_t, x_{t+1}, \dots, x_{t+\epsilon}$ to decide if there exists a change point in t. Hence, according to this definition, an offline algorithm is ∞-real-time and a completely-online algorithm is 0-real-time. As can be understood, the lower the ϵ value is, the more responsive the system can be. However, reducing the value of ϵ usually impacts on the model's accuracy. In this paper, we focus our efforts on developing a configurable ϵ-real-time method that can be adapted to any responsiveness or accuracy requirements.

Multiple CPD algorithms have been developed that behave differently in terms of response time. They can be supervised or unsupervised.

A supervised CPD approach employs machine learning algorithms in form of binary or multi class classifiers. In the first case, all the state transitions at any time t represent one class and all of the within-state points a second class. This has been demonstrated to be a complex learning problem as there can exist too many possible types of transitions (Cook & Krishnan, 2015; Feuz, Cook, Rosasco, Robertson, & Schmitter-Edgecombe, 2014). In the second case, a multi class classifier learns to find state boundaries if the number of possible states is given. Again, it is a hard training problem as the data must be diverse

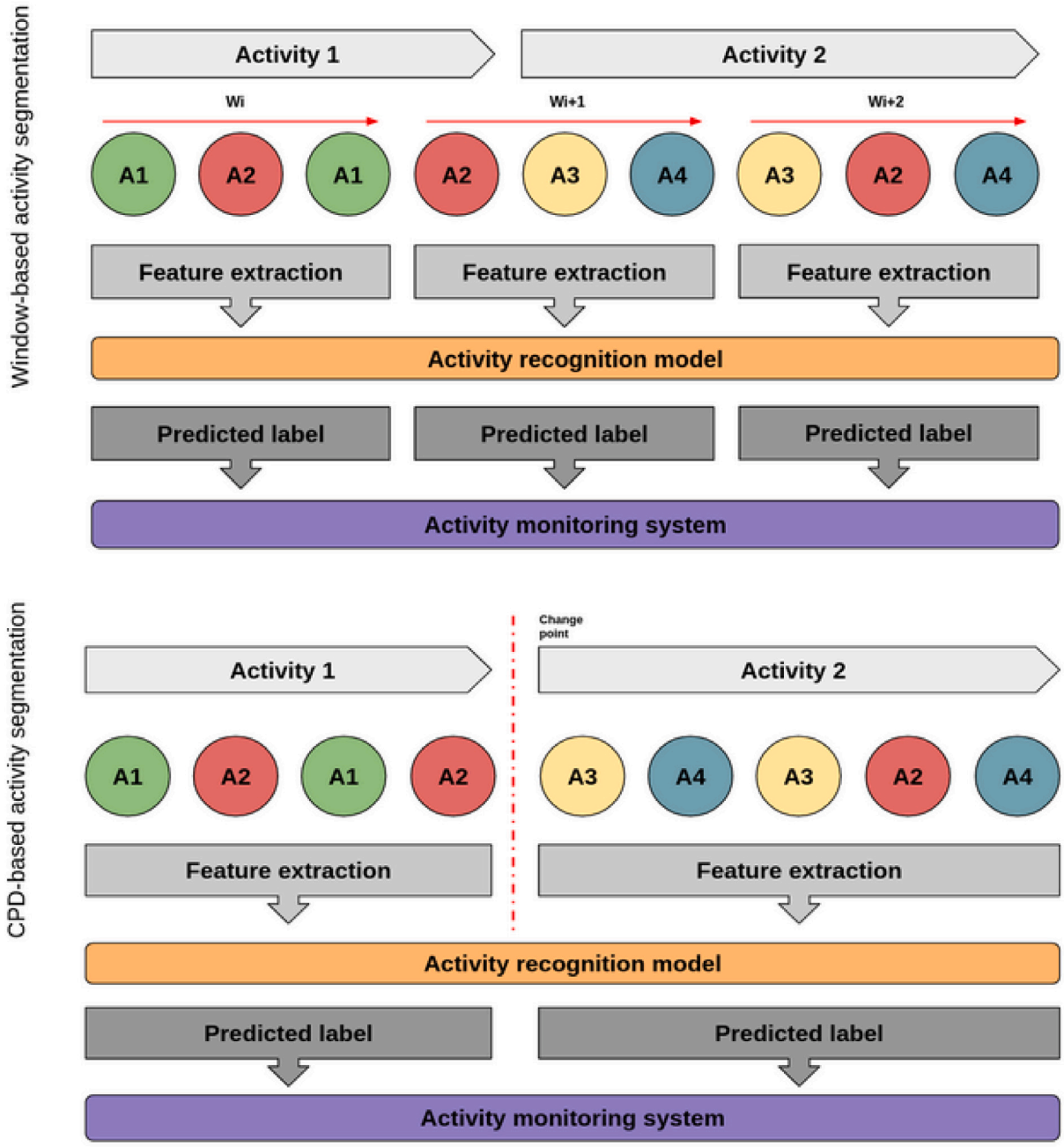

Fig. 1. Window-based and CPD-based segmentation. *Ai* is the action identifier (sensor event). The extracted features from each segment are used to predict the ongoing activity.

enough to represent all the possible states but also all the possible transitions between them (Cleland et al., 2014; Han, Lee, Lee, et al., 2012; Synnott, Chen, Nugent, & Moore, 2014). Apart from these two approaches, virtual classifiers can also be used to detect change points in a supervised manner (Hido, Idé, Kashima, Kubo, & Matsuzawa, 2008). Although most of them perform on 1-real-time, labeled data is hard to find in real applications (Kwon, Kang, & Bae, 2014; Szewcyzk, Dwan, Minor, Swedlove, & Cook, 2009). Due to this fact, the method we propose is fully unsupervised.

Unsupervised algorithms find change points without the need of labeled data. These techniques rely on statistical features of the time series to determine if a change point exists at time t. In the literature, they are often differentiated in six families (Aminikhanghahi & Cook, 2017a):

- **Subspace model**. The CPD is based on the analysis of subspaces. The time series is represented by state spaces and the change points are detected by identifying state space distances between consecutive sliding windows.
- **Probabilistic methods**. The algorithms that fall in this category (e.g. bayesian approaches) estimate probability distributions for each new sliding window based on the previous ones to detect change points.
- **Clustering**. These methods focus on grouping similar data points into clusters that represent states and identify change points by comparing its features.
- **Kernel based methods**. This kind of algorithms map observations onto a higher-dimensional feature space and compare the homogeneity of consecutive windows to detect change points.
- **Graph based methods**. By representing the time series data points as nodes and edges, these methods calculate similarities

between observations that can be used to detect change points when compared to a threshold.

- **Likelihood ratio**. These techniques assume that the probability density of two consecutive sliding windows are the same if they belong to the same state. Then, they rely on calculating density ratios between windows to find change points.

As the nonparametric likelihood ratio algorithms uLSIF, RuLSIF and SEP have shown to obtain the best results on CPD in smart home time series data (Aminikhanghahi & Cook, 2017b; Aminikhanghahi et al., 2018), are totally unsupervised and can work in ϵ-real-time, we compare our method with them.

Regarding the use of embeddings for change detection as we propose in this research work, it has been already applied in other domains. Ceci, Corizzo, Japkowicz, Mignone, and Pio (2020) proposed an embedding based change detection approach to predict the behavior of smart grids. To do so, time series' embeddings were created through stacked auto-encoders and Euclidean distance was used with a predefined threshold in order to detect if future observations were deviating from the expected data distribution. Huang, Kong, and Huang (2014) proposed the use of Kernel Mean Discrepancy in order to detect process changes in the high dimensional process monitoring field. In contrast to previously mentioned works, we propose the use of embeddings created using Word2Vec algorithm and the cosine similarity for activity segmentation.

3. Embedding-based real-time CPD

In this section, we describe our method, which has three variants. We present a naive approach to perform CPD with embeddings as baseline, and two elaborated, more complex alternatives. We do also propose a technique to introduce expert knowledge in our particular embedding-based CPD task.

3.1. Sensor vs. action space

User behaviors are comprised of a large collection of defining elements, making them a complex structure. In order to properly describe it, we have defined a series of concepts on the basis of those proposed in Chaaraoui, Climent-Pérez, and Flórez-Revuelta (2012): actions, activities and behaviors. Actions describe the simplest conscious movements, while behaviors describe the most complex conduct. To be able to work with a more flexible representation of the information in the intelligent environments, we map the raw sensor data to actions like proposed in Chen et al. (2008). Other authors have proved that actions area suitable approach to model behaviors (Schank, 1983). The advantage of working on the action-space is that different sensor types may detect the same action type, simplifying and reducing the hypothesis space. This is even more important when using semantic embeddings to represent these actions in the model, as the reduced number of actions produces more significant embedding representations, as seen in Almeida and Azkune (2018).

3.2. Proposed method

First, our method requires to generate the actions embeddings offline. This step only has to be done once and does not require labeled data. As defined in our previous work (Almeida & Azkune, 2018), given a sequence of actions $S_{act} = \{a_1, \dots, a_i, \dots, a_{l_a}\}$ where l_a is the sequence length and $a_i \in R^{d_a}$ indicates the action vector of the ith action in the sequence, we let $Context(a_i) = [a_{i-n}, \dots, a_{i-1}, a_{i+1}, \dots, a_{i+n}]$ be the context of a_i, where $2n$ is the length of the context window. Being $p(a_i|Context(a_i))$ the probability of the ith action in the sequence for action a_i, the target of the model used to create the embeddings is to optimize the log maximum likelihood estimation (logMLE):

$$L_a(MLE) = \sum_{a_i \in S} \log p(a_i|Context(a_i)) \tag{1}$$

Table 1
Cosine similarity scores (Formula (2)) between sample actions embeddings. See Fig. 2.

Action	A1	A2	A3	A4
A1	1.00	0.86	0.34	0.27
A2	0.86	1.00	0.33	0.24
A3	0.34	0.33	1.00	0.91
A4	0.27	0.24	0.91	1.00

Second, and once the action embeddings are generated, we compute cosine similarity scores (CS) between them to find change points in real-time. In natural language processing, cosine similarity is used for tasks such as phrase analogy and synonymy (Mikolov, Chen, Corrado and Dean, 2013; Mikolov, Yih and Zweig, 2013). Inspired by this, and as activities are formally composed by actions, our first intuition was that calculating similarities between consecutive actions could be useful for detecting activity transitions (change points). Concretely, we thought that if two consecutive actions were not similar (low cosine similarity score), it could probably mean that the user changed to another activity. In this paper, we present and evaluate three main alternatives built upon this intuition.

CS-CPD. The first one is a naive approach that compares the computed cosine similarity between two consecutive actions to a threshold. Formally, if the cosine similarity score $CS(a_i, a_{i+1})$ computed between action a_i at time t_i and action a_{i+1} at time t_{i+1} is below a certain threshold th, a change point occurs at time t_{i+1}.

$$CS(A_i, A_j) = \frac{A_i \cdot A_j}{\|A_i\| \|A_j\|} \tag{2}$$

CCS-CPD. The second one is a more complex approach that defines the existence of a change point or activity transition as follows. Given an action a_i at time t_i, we determine that a change point exists at t_i if the context cosine similarity score $CCS(a_i, n)$ is above a certain threshold th, where $2n$ is the length of the context window.

$$CCS(A_i, N) = \frac{1}{2N} \sum_{j=1}^{N} CS(A_i, A_{i+j}) + (1 - CS(A_i, A_{i-j})) \tag{3}$$

The calculation of this score is based on the assumption that an action that is closely similar to the following actions, but which is clearly different from the actions that precede it, may be a transition between activities (change point).

CCSD-CPD. The third approach employs the context cosine similarity score differently. Instead of comparing it to a threshold, we search for descending patterns in the following way. Given an action a_i at time t_i, we determine that a change point exists at t_i if $CCS(a_i, n) > CCS(a_{i-1}, n) + d$, where d is the minimum distance to consider a change of trend and $2n$ is the length of the context window.

For a better understanding of the differences between the first and second approaches, consider the example depicted in Fig. 2. In this figure, a toy data stream composed of two consecutive activities, Activity 1 and Activity 2, which are frequently performed by executing Action 1 and Action 2, and Action 3 and Action 4 respectively is shown. The semantic relationships between actions are described in Table 1 with cosine similarity. The threshold value is set to 0.7 and the context window length is set to 2.

As can be seen for this example, the second method (CCS), which employs the context cosine similarity score, can deal with outliers (actions that not fit the activity in a normal situation). The first method (CS), instead, generates false positives when it encounters an outlier, as it does not consider the context of the action.

Likewise, we believe that the third method is less sensitive to outliers than the second method, since context cosine similarity scores decrease when approaching to an actual activity transition (change point) and extremely increase when it occurs. Namely, by searching for

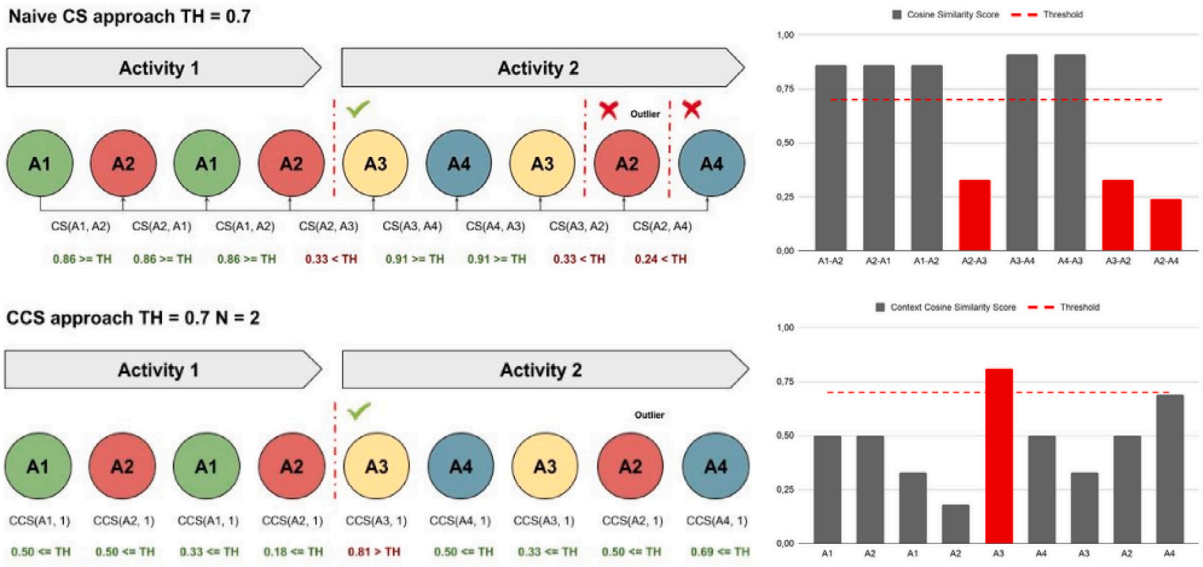

Fig. 2. Comparison between the naive CS approach and the CCS approach. The CS approach does not consider the context of the action, creating false positives when it encounters an outlier. The CCS approach calculates context cosine similarity scores (Formula (3)) and does not create false positives when an unexpected action is performed. See Table 1.

these changes of trend, we think that we can deal with outliers better than with a fixed threshold value.

Regarding the threshold value th and the minimum distance d value, it must be said that they have to be chosen based on optimal performance for a particular scenario (time series). This same thing happens with the aforementioned state of the art approaches that employ density ratio CPD algorithms to perform activity segmentation, as they also compare the obtained scores against a fixed threshold value (Aminikhanghahi et al., 2018). The context window length must be fixed accordingly too.

Finally, concerning response times, the first method (CS) is 1-real-time, as it only needs one sensor event to determine the existence of a change point. The second approach (CCS) and the third approach (CCSD) are both (n+1)-real-time where 2n is the length of the context window. We consider that being able to configure the context window length is a strength, because it allows to fit the model to any application requirements (balance between delay and accuracy) and adapt it to the nature of the time series.

3.3. Retrofitting action embeddings

The cosine similarity score between two action embeddings is significantly influenced by how well these are distributed in the vector space. Due to this fact, we hypothesize that the embeddings quality directly impacts our CPD methods performance. In other words, we believe that the embeddings must represent the actions well enough so our models behave as expected.

To ensure so, the embeddings have to be trained with sufficient, high quality data. Although it is possible to gather the necessary amount of training data in most occasions, smart home time series data is usually full of noise (outliers). These outliers can distort their generation process and consequently, they may end up not being sufficiently representative of the different actions. To overcome this, we propose to retrofit the trained action embeddings with knowledge-graphs.

Retrofitting is a graph-based learning technique that employs relational resources to obtain higher quality semantic embeddings (Faruqui et al., 2014). In short, it is a post-processing step that runs belief propagation on a knowledge graph to update embedding vectors so the new ones are similar to the vectors of related token types and similar to their purely distributional representations. The retrofitting software is publicly available at its GitHub repository.[1]

As occurs with the Word2vec algorithm, this method was originally thought for natural language processing. In the original paper, the authors demonstrated that retrofitting word embeddings with semantic lexicons can improve many NLP tasks. For instance, word similarity, synonym selection and sentiment analysis (Faruqui et al., 2014).

In this work, we propose to use this technique to update the trained action embedding vectors with three different knowledge graphs that have the actions as nodes and the relations between actions as edges.

Formally, each of these graphs is an undirected graph $G = (V, E)$ where V is a set of unique actions (sensor events) and E is a set of edges $[x, y]$ such that $x, y \in V$. An edge $[x, y]$ is constructed if two actions x, y share a common entity. For the first graph, we construct edges with activities. For the second graph, we construct edges with locations. The third graph has the combination of edges of both the activity and location entities.

Necessarily, these graphs must be generated by experts or from a labeled data source. Thus, the approaches that employ this post-processing step are not unsupervised, but hybrid. We provide insights about how the aforementioned graphs can be generated in Section 4.

4. Experiments and results

In this section, we describe the experiments we performed to validate our methods. We performed three types of experiments:

- **Type I - Cross-validated performance**. These experiments evaluate our embedding methods in three benchmark datasets. The aim of these experiments is to estimate the behavior of our methods in a real scenario. We employ leave-one-day-out cross-validation and report averages and standard deviations for the chosen metrics. We generate the embeddings and optimize parameters only with each split training data.
- **Type II - Performance comparison in 2-real-time**. These experiments compare our methods with three state of the art density ratio CPD algorithms: uLSIF, RuLSIF and SEP. The aim of these experiments is to confirm the usefulness of our methods against

[1] https://github.com/mfaruqui/retrofitting.

Table 2
Details of the datasets used for the evaluation.

	House A	House B	House C
Age	26	28	57
Setting	Apartment	Apartment	House
Rooms	3	2	6
Duration	28 days	14 days	19 days
Sensors	14	23	21
Activities	8	13	16

algorithms that have been proven as effective for activity segmentation. We decide to evaluate them in 2-real-time, leaving one week of data for testing and the rest for training (embedding generation and parameter optimization). As all the methods have random initialization, we perform several executions for the different configurations and provide averages and standard deviations for the chosen metrics.

- **Type III - Embeddings quality**. These experiments compare our embedding methods with and without retrofitting. The aim of these experiments is to confirm the hypothesis made in Section 3: 'Embeddings quality directly impacts our CPD methods performance'. For such purpose, we retrofit embeddings with three different knowledge graphs. We generate such graphs from two sources of information: Expert Activity Models (EAMs) and data labels (groundtruths). These experiments follow the same parameter optimization, embedding generation and cross-validation strategy as Type I experiments.

Before explaining those experiments in detail, we describe the datasets and metrics we chose for the evaluation.

4.1. Datasets

To make our results comparable and easily reproducible, we use the publicly available Van Kasteren activity datasets of Houses A, B and C (Van Kasteren, Englebienne, & Kröse, 2011; Van Kasteren, Noulas, Englebienne, & Kröse, 2008). These datasets are well known in the literature and have traditionally served as benchmark for different HAR tasks. Table 2 contains a summary of the datasets.[2]

In House A, an individual is monitored in a three-room apartment where 14 binary sensors were installed. Those sensors were installed in locations such as doors, plateboards, cupboards, freezer, refrigerator or toilet. Sensor data for 28 days was collected for a total of 2120 sensor events and 245 activity instances. The annotated activities were: 'Leave House', 'Use Toilet', 'Take Shower', 'Go To Bed', 'Prepare Breakfast', 'Prepare Dinner', 'Get Drink' and 'None'. It is important to say that although many authors remove the 'None' label (other activity or not recognized activity), we take it into account as an activity itself. In real scenarios this label will appear frequently, as a result of erratic behavior, annotation problems or sensor errors, so we decide to maintain it to test algorithms performance more realistically. Likewise, we must say that during preprocessing we removed day 2008-03-16 because it has annotation problems. In this period, no matter which, all sensor events were annotated with the 'Leave House' label.

House B and C present more challenging scenarios, with a greater number of different sensors and activities (for example, 'Brush Teeth', 'Shave' or 'Play Piano'), but with less recorded days. However, in House B and C, sensor event frequency is higher than in House A, having a total of 13 358 and 22 770 sensor events respectively. Another important detail regarding House C is that we divide the 'Use Toilet' activity into 'Use Toilet Upstairs' and 'Use Toilet Downstairs', as there are two different toilets in the house.

We think that the combination of these three datasets is a good reference to test our methods, since people with different ages, activities with different levels of detail and different house distributions are present. We do also think that having different sensor event frequencies is good to test our CPD algorithms rigorously, as sensor event frequency directly impacts the proportion between change points (activity transitions) and no change points. For more information of the three houses, please refer to Van Kasteren et al. (2011, 2008) where floorplans and other details such as annotation methodologies are provided.

Regarding sensor-action mappings, it must be said that we decided to apply a 1:1 mapping function, because the semantics of each sensor event can be clearly identified and considered unique with respect to the target activities (see Table 3).

4.2. Performance metrics

According to Aminikhanghahi and Cook (2017a), the most common metrics used to evaluate the performance of CPD algorithms are True Positive Rate (TPR or also known as Sensitivity), False Positive Rate (FPR) and Geometric Mean (G-Mean). In this paper, we use these metrics to evaluate our methods.

TPR (True Positive Rate, Sensitivity). The portion of positive examples (change points) that were recognized correctly. This metric indicates how effectively true state changes are detected by a CPD algorithm.

$$TPR = \frac{TP}{TP + FN} \tag{4}$$

FPR (False Positive Rate). The portion of negative examples (no change points) that are recognized as change points to the total number of negative examples. This metric indicates how many 'false alarms' generates a CPD algorithm.

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

G-Mean (Geometric Mean). As a CPD algorithm often encounters an imbalanced class distribution (understandably, change points are less in number than no change points), it is needed to use a metric that combines both Sensitivity and Specificity to evaluate its performance.

$$\text{G-Mean} = \sqrt{TPR \times TNR} = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{FP + TN}} \tag{6}$$

To determine if a detected change point is correct, we adopt the same strategy as in the literature (Aminikhanghahi et al., 2018; Feuz et al., 2014; Liu, Yamada, Collier, & Sugiyama, 2013). We consider that a detected change point at time t_i is correct if a true change point occurs in the time interval $[t_i - \lambda, t_i + \lambda]$. We chose $\lambda = 1$ for exact CPD evaluation and $\lambda = 10$ for evaluation with a small time offset (in seconds).

4.3. Type I - Cross-validated performance

In order to evaluate the performance of the proposed approaches, we have used leave-one-day-out cross-validation. In a similar manner to common cross-validation approaches, this evaluation methodology evaluates proposed approach's performance in each of the available days in the dataset with a model trained with all the days except the one being evaluated. Then, we report the average and standard deviation values for the chosen metrics in Table 4.

For parameter optimization, we perform sensitivity analysis with different values at regular intervals with the training days of each split. For thresholds th, we increment by 0.1 from 0.0 to 1.0 and choose the best value for the G-Mean metric in exact CPD. For minimum distances d, we test the following values: 0.0, 0.001, 0.01 and 0.1. Again, we choose the best one according to the G-Mean. Finally, for windows lengths, we test n values from 1 to 5.

[2] These datasets are available in our GitHub repository after preprocessing and conversion to CSV format: https://github.com/gazkune/generic_ar/tree/master/datasets.

Table 3
Extract of the House A dataset after preprocessing. Each sensor event has associated a timestamp and an activity label. We decided to apply a 1:1 mapping function between sensor events and actions. When the activity label changes between two consecutive actions (ordered by timestamp), a change point (activity transition) occurs. We mark change points with 1 and no change points with 0.

Timestamp	Sensor event (ON)	Action	Activity	Change point
09:12:53	HallBathroomDoor	HallBathroomDoor	UseToilet	0
09:13:55	ToiletFlush	ToiletFlush	UseToilet	0
09:14:01	HallBathroomDoor	HallBathroomDoor	UseToilet	0
09:14:06	ToiletFlush	ToiletFlush	UseToilet	0
09:15:03	PlatesCupboard	PlatesCupboard	PrepareBreakfast	1
09:15:18	GroceriesCupboard	GroceriesCupboard	PrepareBreakfast	0
09:15:28	Fridge	Fridge	PrepareBreakfast	0
09:18:05	PlatesCupboard	PlatesCupboard	PrepareBreakfast	0
09:57:21	HallBathroomDoor	HallBathroomDoor	UseToilet	1
09:59:06	ToiletFlush	ToiletFlush	UseToilet	0
10:00:25	HallBathroomDoor	HallBathroomDoor	UseToilet	0
10:00:30	HallToiletDoor	HallToiletDoor	UseToilet	0
10:31:18	HallToiletDoor	HallToiletDoor	TakeShower	1
10:31:23	HallToiletDoor	HallToiletDoor	TakeShower	0
10:40:42	HallToiletDoor	HallToiletDoor	TakeShower	0
10:40:47	HallToiletDoor	HallToiletDoor	TakeShower	0
10:49:25	Frontdoor	Frontdoor	LeaveHouse	1
17:13:24	Frontdoor	Frontdoor	LeaveHouse	0

Table 4
Cross-validated performance experiments. Leave-one-day-out cross-validation. Exact and 10s offset evaluation CPD. Averages and standard deviations are reported. Best results in bold. Embeddings were generated only with each split's training data. Optimum parameters (threshold, minimum distance, context window length) were selected according to best G-Mean on each split's training data (exact CPD evaluation).

	Model	Exact CPD			10s offset CPD		
		TPR	FPR	G-MEAN	TPR	FPR	G-MEAN
A	CS	**0.89** ± 0.14	0.43 ± 0.15	0.70 ± 0.09	**0.90** ± 0.12	0.40 ± 0.15	0.72 ± 0.09
	CCS	0.72 ± 0.13	0.27 ± 0.07	0.72 ± 0.09	0.75 ± 0.11	0.24 ± 0.07	0.75 ± 0.07
	CCSD	0.77 ± 0.12	**0.23** ± 0.06	**0.77** ± 0.06	0.78 ± 0.11	**0.21** ± 0.06	**0.78** ± 0.06
B	CS	**0.57** ± 0.12	0.10 ± 0.06	**0.71** ± 0.09	0.67 ± 0.09	0.09 ± 0.05	0.78 ± 0.05
	CCS	0.55 ± 0.12	0.12 ± 0.06	0.69 ± 0.08	**0.70** ± 0.09	0.10 ± 0.05	**0.79** ± 0.05
	CCSD	0.51 ± 0.15	**0.08** ± 0.06	0.67 ± 0.12	0.61 ± 0.08	**0.08** ± 0.05	0.75 ± 0.06
C	CS	0.40 ± 0.15	**0.03** ± 0.03	0.60 ± 0.16	0.44 ± 0.15	**0.03** ± 0.03	0.63 ± 0.17
	CCS	**0.56** ± 0.16	0.09 ± 0.08	**0.69** ± 0.17	**0.63** ± 0.17	0.07 ± 0.05	**0.74** ± 0.18
	CCSD	0.42 ± 0.13	0.04 ± 0.03	0.62 ± 0.15	0.46 ± 0.13	0.04 ± 0.03	0.64 ± 0.16

For generating the action embeddings according to the definition given in Section 3, we employed Mikolov, Sutskever et al. (2013) Word2vec algorithm on its Gensim implementation[3] with its default parameter values (size=50, window=1, iter=5, etc.) and the training days data. These same default parameters were found optimum in our previous papers to obtain high quality semantic action embeddings for behavior modeling (next action prediction task) in the same datasets (Almeida & Azkune, 2018; Almeida, Azkune, & Bilbao, 2018).

4.4. Type II - Performance comparison in 2-real-time

In these experiments, we compare our embedding-based methods that calculate cosine similarities between actions (CS, CCS, CCSD) against state of the art density ratio algorithms (uLSIF, RuLSIF, SEP) in the three described datasets. Those generalistic methods for change point detection have been proven effective for activity segmentation in 2-real-time recently (Aminikhanghahi et al., 2018).

Probably, the biggest limitation of these density ratio CPD methods is their computational cost (Aminikhanghahi et al., 2018). Due to this fact, we have divided the experiments of this section in two steps. First we compare the computation times of our approach (CS, CCS, CCSD), cosine similarity, with the one used by the current state of the art methods (uLSIF, RuLSIF, SEP), density ratio. We do this by increasing the number of features of two random vectors. We report the results for both approaches in Table 5 and Fig. 3. Please note that the results are grouped depending on the distance used to compare the vectors, as it is the main component of the algorithms regarding the computational costs. To compute density ratios, we employed the open-source library *densratio*[4] for the Python programming language. To calculate cosine similarity values between vectors, we employed SciPy library's cosine distance function.[5]

The second step is the performance evaluation. As the results of the previous step show, we cannot afford doing leave-one-day-out cross-validation for density ratio algorithms (due to the time constraints). Therefore, we split each dataset in train and test of consecutive days, instead of adopting a cross-validation strategy. We leave the last week for testing and the rest of the days for training (parameter optimization and embedding generation). The optimum parameters we found for the three houses are shown in Table 6. We employed the same sensitivity analysis strategy as Type I experiments for such parameter optimization.

To extract features and model density ratios between windows, we followed the same feature engineering methodology as Aminikhanghahi et al. (2018). We slid a window that looks at $k = 30$ actions and extracts a feature vector with time information, action information and general window characteristics. We tried to employ other, simpler feature extraction approaches (only action event counts, action event counts plus window features, action event counts plus time features) without improving the results obtained by the referenced approach. We did also test different values of k (5, 10, 15, 20, 25). This feature

[3] https://radimrehurek.com/gensim/models/word2vec.html.

[4] https://github.com/hoxo-m/densratio_py.

[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cosine.html.

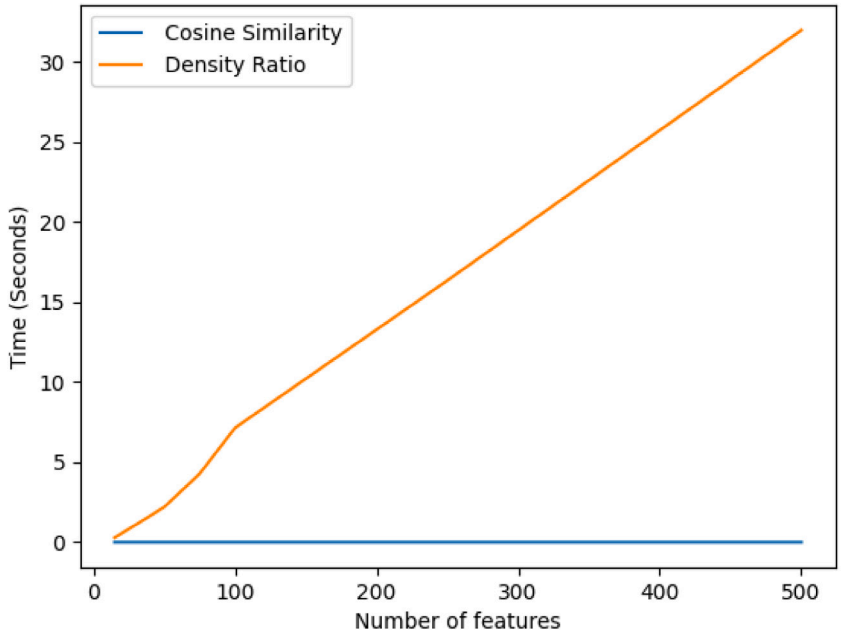

Fig. 3. Evolution of the required computation time as we increase the number of features of two vectors for Cosine Similarity (CS, CCS, CCSD) and Density Ratio (uLSIF, RuLSIF, SEP).

Table 5
Time required in seconds for the computation of Cosine Similarity (CS, CCS, CCSD) and Density Ratio (uLSIF, RuLSIF, SEP) between two vectors as the number of features increases.

Nº. features	Cosine similarity (s)	Density ratio (s)
15	0.000086	0.294254
30	0.000061	1.092677
50	0.000050	2.194505
75	0.000048	4.274077
100	0.000045	7.142882
250	0.000047	16.358610
500	0.000052	32.000003

Table 6
Performance comparison in 2-real-time experiments. Parameter optimization with training sets. *th*: threshold, *n*: context window length, *d*: minimum change of trend distance. Please, consider that, as explained in Section 3, our CCS and CCSD embeddings methods are (n + 1)-real-time and density ratio algorithms are n-real-time. Thus, we are comparing the methods in 2-real-time, except the CS method which is always 1-real-time.

Model	House A			House B			House C		
	th	*n*	*d*	*th*	*n*	*d*	*th*	*n*	*d*
uLSIF	0.1	2	–	0.1	2	–	0.1	2	–
RuLSIF	0.3	2	–	0.2	2	–	0.2	2	–
SEP	0.3	2	–	0.1	2	–	0.1	2	–
CS	0.5	–	–	0.9	–	–	0.9	–	–
CCS	0.5	1	–	0.5	1	–	0.5	1	–
CCSD	–	1	0.1	–	1	0.0	–	1	0.0

extraction was performed offline to accelerate the experimentation process. In a real scenario, it can be performed in real-time.

Having said so, it is important to remark that both the density ratio methods and our methods have random initialization. Because of that, we executed each method 5 times with the optimum parameters[6] of Table 6 in the three test sets (House A, B and C) and calculated averages and standard deviations. We provide the results in Table 7.

[6] For the sensitivity analysis in the different training sets, we also performed 5 executions.

4.5. Type III - Embeddings quality

For these experiments, we follow the preprocessing and cross-validation methodologies described in the previous Type I experiments (Cross-validated performance). However, in this case, we compare our CPD embedding approaches with and without retrofitting to analyze if the embeddings quality has a direct impact in their performance. For making a fair comparison and getting rid off the randomness factor on the embeddings generation, we employed the same embeddings models generated in Type I experiments.

To retrofit the embeddings, we used 3 different knowledge graphs. For generating such graphs, we employed two different sources of information: Expert Activity Models (EAMs) and the labeled data of the three datasets' training sets. In our previous work (Azkune & Almeida, 2018), we defined an EAM as a standardized computational model of activities generated by experts that contains the following knowledge:

- **Actions**. The minimum number of actions that are usually executed to perform a given activity. Each action can be associated to one or more sensor events. Each sensor is placed in a location (e.g. Kitchen, Bathroom) and classified with a type (e.g. Cooking, Hygiene).
- **Duration**. A rough estimation of the typical duration of a given activity.
- **Starting time**. Approximate time ranges when a given activity is usually started (multiple time ranges are supported).
- **Locations**. Semantic tags for the places where a given activity is usually performed. For example, Bathroom or Bedroom (multiple locations for an activity are supported).

With this information, we are able to construct activity and location graphs, as we can determine what actions share locations and activities to generate edges. Likewise, we think that generating edges only from labeled data is a perfect alternative when expert knowledge is not available. An edge $[x, y]$ can be constructed if two actions share a common label l at any time in a common labeled data source. For example, considering Table 3, where an extract of House A dataset is shown, an edge between 'Groceries Cupboard' action and 'Fridge' action could be constructed as they share the 'Prepare Breakfast' activity (the

Table 7
Performance comparison in 2-real-time experiments. Test set (House A, B and C). Exact and 10s offset evaluation CPD. Averages and standard deviations for 5 executions. Best results in bold. Optimum parameters were selected empirically with each house training set (see Table 6).

	Model	Exact CPD			10s offset CPD		
		TPR	FPR	G-MEAN	TPR	FPR	G-MEAN
A	uLSIF	0.54 ± 0.04	**0.32** ± 0.03	0.61 ± 0.01	0.62 ± 0.04	**0.26** ± 0.03	0.68 ± 0.01
	RuLSIF	**0.61** ± 0.02	0.40 ± 0.01	0.60 ± 0.01	**0.69** ± 0.01	0.34 ± 0.01	0.68 ± 0.01
	SEP	0.57 ± 0.05	0.33 ± 0.03	**0.61** ± 0.01	0.64 ± 0.04	0.28 ± 0.02	**0.68** ± 0.01
	CS	**0.85** ± 0.12	**0.44** ± 0.13	0.68 ± 0.08	**0.86** ± 0.11	0.42 ± 0.13	0.70 ± 0.08
	CCS	0.71 ± 0.10	0.27 ± 0.03	0.72 ± 0.06	0.73 ± 0.09	0.25 ± 0.03	0.74 ± 0.06
	CCSD	0.82 ± 0.03	**0.27** ± 0.01	**0.78** ± 0.02	0.84 ± 0.03	**0.24** ± 0.01	**0.80** ± 0.02
B	uLSIF	0.18 ± 0.03	**0.05** ± 0.01	0.41 ± 0.03	0.26 ± 0.04	**0.04** ± 0.01	0.50 ± 0.04
	RuLSIF	0.34 ± 0.05	0.12 ± 0.01	0.55 ± 0.04	0.48 ± 0.05	0.12 ± 0.01	0.65 ± 0.03
	SEP	**0.42** ± 0.02	0.15 ± 0.00	**0.60** ± 0.02	**0.56** ± 0.02	0.14 ± 0.00	**0.69** ± 0.01
	CS	0.39 ± 0.04	0.09 ± 0.00	0.59 ± 0.03	0.54 ± 0.05	0.08 ± 0.00	0.70 ± 0.03
	CCS	0.39 ± 0.01	**0.06** ± 0.00	0.61 ± 0.01	0.49 ± 0.00	**0.05** ± 0.00	0.68 ± 0.00
	CCSD	**0.41** ± 0.02	0.08 ± 0.00	**0.62** ± 0.01	**0.55** ± 0.01	0.07 ± 0.00	**0.71** ± 0.01
C	uLSIF	0.21 ± 0.01	**0.01** ± 0.00	0.46 ± 0.02	0.23 ± 0.02	**0.01** ± 0.00	0.48 ± 0.02
	RuLSIF	0.39 ± 0.02	0.05 ± 0.00	0.61 ± 0.01	0.43 ± 0.03	0.05 ± 0.00	0.64 ± 0.02
	SEP	**0.45** ± 0.01	0.06 ± 0.00	**0.65** ± 0.01	**0.49** ± 0.01	0.06 ± 0.00	**0.68** ± 0.01
	CS	0.30 ± 0.01	**0.02** ± 0.00	0.54 ± 0.01	0.33 ± 0.01	**0.02** ± 0.00	0.57 ± 0.01
	CCS	0.37 ± 0.01	0.02 ± 0.00	0.60 ± 0.01	0.41 ± 0.01	0.02 ± 0.00	0.63 ± 0.01
	CCSD	**0.41** ± 0.01	0.03 ± 0.00	**0.63** ± 0.00	**0.44** ± 0.01	0.03 ± 0.00	**0.66** ± 0.01

label) at time 9:15:18 and time 9:15:28 respectively. The same logic applies to locations or any kind of entity in form of label.

It is important to remark that, when constructing activity graphs from labeled data, we do not consider the 'None' activity, as it could affect negatively to the results, modeling relations between actions that do not correspond with reality or any expert knowledge. However, remember that this 'None' label (other activity or not recognized activity) is considered in the CPD task evaluation. We made the same decision regarding 'Setting Up Sensors' and 'Install Sensor' activities (only performed in the first two days of House B), as they do not give any insight about relations between actions.

We show the results of retrofitting embeddings with different knowledge graphs in Table 8. Similarly, to illustrate how retrofitting can improve actions embeddings and provide more significant cosine similarity scores, we built Table 9.

5. Discussion

We discuss the results for each type of experiment separately.

5.1. Type I - Cross-validated performance

First of all, we will start analyzing the results from G-Mean metric's perspective. In House A, we can observe that the CCSD method achieves the best results in both exact and 10 offset evaluations, increasing in 5 and 3 points respectively the performance of CCS, and 7 and 6 points compared to CS. As it can be clearly seen analyzing these results, CCSD outperforms by a wide margin the other two proposed methods. However, this varies in the other two scenarios where CS and CCS gain importance.

With regard to House B, CS and CCS have an equiparable performance, being the first one the best approach in the exact evaluation (2 points better than CCS and 4 compared to CCSD) and CCS the best performing in the 10s offset scenario (1 point better than CS and 4 compared to CCSD).

In House C, the best performing approach is CCS by a wide margin with respect to CS and CCSD for both exact and 10s offset evaluations, achieving 9 and 7 points differences in the first one and 11 and 10 in the second one.

Therefore, after analyzing these results we can conclude that using the actions's context (CCS, CCSD) is useful in the three scenarios (House A, B and C). However, CCS and CCSD's performance varies depending on the testing scenario: House A for CCSD, and Houses B and C for CCS.

About G-Mean's standards deviations, it is important to remark that the three houses have relatively high values, particularly House C which achieves values up to 18 points. This means that there is a significant difference in the performance of the models depending on the day that it is being evaluated.

Regarding the TPR and FPR metrics, all methods seem to find a balance between detecting change points and not generating false alarms. This makes sense since we optimized parameters looking at the G-Mean metric. Nevertheless, it is remarkable that in House A higher TPRs are achieved in comparison to the other two houses. We believe that those differences are caused due to the complexity of each environment. House B and C are clearly more challenging scenarios than House A. They have higher number of activities, more sensors and less training data, what makes more demanding to detect activity transitions in real-time.

5.2. Type II - Performance comparison in 2-real-time

On the one hand, from a quantitative perspective, the results obtained in these experiments show that our embedding-based CPD methods outperform the latest density ratio algorithms in two of the three chosen scenarios. We start analyzing the results for the G-Mean metric, which is the most important one, as it provides information about the algorithm ability to find change points (activity transitions) without generating false alarms.

In House A, only with the CS naive approach, we are able to improve SEP's G-Mean (best density ratio algorithm) by 7 points for exact CPD. With the rest of the two approaches, CCS and CCSD, we are able to improve it by 11 and 17 points respectively. With a t-test that outputs ($p = 0.0074$) for CCS $\leftrightarrow$ SEP and ($p = 0.0000$) for CCSD $\leftrightarrow$ SEP, we confirm that their differences are statistically significant at the ($p<0.05$) level. For the 10s offset evaluation experiments, we also improve the likelihood algorithms G-Mean, but with smaller differences.

In House B, we improve SEP's G-Mean by 2 points ($p = 0.0495$) with the CCSD approach. We believe that the gap between House A and House B's G-Mean is because House B is a more challenging scenario and has less training data, what clearly impacts the embeddings generation and the parameter optimization. House B also presents some annotation problems, derived from the annotation technique that was used, a personal diary (Van Kasteren et al., 2011). We extract the same conclusions for 10s offset evaluation.

With respect to House C, we are not able to improve SEP's G-Mean. It outperforms CCSD by 2 points ($p = 0.0000$). However, it is important

Table 8
Embeddings quality experiments. Leave-one-day-out cross-validation. Exact evaluation CPD. Performance average changes with retrofitting for the G-Mean metric. Best improvements in bold. We use three types of graphs to retrofit embeddings. ACT: Activity graph, LOC: Location graph, ACT-LOC: Activity and Location graph. Graphs are generated from two sources of information: labeled data (from training sets) and Expert Activity Models (EAMs). Embeddings were generated only with each split's training data (same models of Table 4). Optimum parameters (threshold, minimum distance, context window length) were selected according to best G-Mean on each split's training data (exact CPD evaluation).

	Model	Labeled data (training set)			Expert activity models		
		ACT	LOC	ACT-LOC	ACT	LOC	ACT-LOC
A	CS	0.57 −13%	0.81 +11%	0.63 −7%	0.81 +11%	0.79 +9%	0.79 +9%
	CCS	0.77 +5%	0.79 +7%	0.72 +0%	0.80 +8%	0.80 +8%	0.80 +8%
	CCSD	0.77 +0%	**0.82 +5%**	0.75 −2%	**0.82 +5%**	**0.82 +5%**	**0.82 +5%**
B	CS	0.30 −41%	0.49 −22%	0.45 −26%	0.61 −10%	0.51 −20%	0.51 −20%
	CCS	**0.70 +1%**	0.69 +0%	0.69 +0%	0.69 +0%	**0.70 +1%**	**0.70 +1%**
	CCSD	0.67 +0%	0.66 −1%	0.64 −3%	0.67 +0%	0.64 −3%	0.64 −3%
C	CS	0.33 −27%	0.60 +0%	0.36 −24%	0.39 −21%	0.57 −3%	0.38 −22%
	CCS	0.69 +0%	**0.75 +6%**	0.69 +0%	0.68 −1%	**0.74 +5%**	0.68 −1%
	CCSD	0.62 +0%	0.62 +0%	0.62 +0%	0.61 −1%	0.62 +0%	0.61 −1%

Table 9
Cosine similarity scores between related actions from House A before and after retrofitting with an activity graph. HBD: HallBathroomDoor, TF: ToiletFlush, HTD: HallToiletDoor. The higher the cosine similarity scores, the better. Our methods rely on high cosine similarity scores between related actions to function properly. In this case, those actions are related because they share the activities 'TakeShower' and 'UseToilet' according to our EAMs and the ground truth labels of the dataset.

Action	HBD		TF		HTD	
	Before	After	Before	After	Before	After
HBD	1.00	1.00	0.50	**0.93**	0.53	**0.87**
TF	0.50	**0.93**	1.00	1.00	0.45	**0.82**
HTD	0.53	**0.87**	0.45	**0.82**	1.00	1.00

to say that we still improve the other two likelihood algorithms performance: RuLSIF and uLSIF. The good results of SEP in comparison to the other likelihood algorithms seem logical to us, since SEP has been demonstrated to be the most efficient CPD algorithm for other smart home time series datasets (Aminikhanghahi et al., 2018). Likewise, we think that as happens with House B, the differences with House A's performance are due to the complexity of the scenario and the lack of high-quality, big enough training sets to create the embeddings.

Analyzing standard deviations, we see that the embedding-based methods' stability is comparable to density ratio methods. The standard deviations range from 1 to 5 points. The only configurations that surpass this range are the naive CS approach (8) and the CCS approach (6) in House A.

In respect of the embedding methods and continuing the analysis of the G-Mean, the results suggest again that considering the action's context (CCS, CCSD) is useful to determine the existence of an activity transition (change point), in contrast of the naive approach (CS), which only considers the next action. The results also confirm our intuition that the CCSD approach, which searches for changes of trend between consecutive context similarity scores, can perform better than the CCS approach, which compares the context similarity scores to a threshold.

On the other hand, with respect to the qualitative differences between methods, it is particularly interesting that the embedding-based methods outperform the density ratio algorithms without feature engineering. In other words, we consider promising the possibility of finding change points by capturing the semantic meaning of actions (sensor events) with unsupervised embeddings models and without the need of hand-crafted features, which are difficult to design and often a potential source of problems as they are usually thought for a concrete environment (not generalizable).

In relation to this last insight, we also consider positive the great performance of embeddings models as they could enable transfer learning (pre-trained embeddings models, finetuning) for this particular task and serve as inspiration to solve many challenges on the field regarding that concept, which is a well-known open question in the literature (Feng & Duarte, 2019; Ma et al., 2020).

Last but not least, it must be said that our method is fast and efficient in comparison to density ratio algorithms. Estimating density ratios in real-time is known to be computationally expensive (Aminikhanghahi et al., 2018) compared to the computation of cosine similarities (see Table 5, Fig. 3). As it can be seen in Fig. 3, while cosine similarity computation times do not suffer a noticeable variation as the number of features increases, the computation time of density ratio increases linearly as we add new features, needing 2 s for its calculation with only 50 features (the dimensionality of our embeddings). This could be a clear advantage to consider when deploying models in edge devices with scarce computational resources (Gómez-Carmona, Casado-Mansilla, Kraemer, López-de Ipiña, & García-Zubia, 2020) or that use rechargeable batteries (De La Concepción, Morillo, Gonzalez-Abril, & Ramírez, 2014).

5.3. Type III - Embeddings quality

The results of these experiments confirm that retrofitting embeddings can be useful to improve their meaning and find change points more precisely. In most cases, retrofitted embeddings improve the G-Mean of all our methods significantly.

Nevertheless, in some occasions, and mainly for the naive CS approach, performance drastically worsens. We think that this performance degradation has to do with the optimal threshold values going above 0.9 when embeddings are adjusted with graphs (consider that 0.9 is the last interval we test). In relation to this and looking at Table 9, we see that cosine similarity scores between actions increase when performing retrofitting, so we hypothesize that optimum thresholds must be increasing too. Thus, it is normal for the CS naive approach, the one more sensible to the threshold, to be so negatively affected.

Regarding CCS method's performance, it is noteworthy to mention that is always improved at least by one point with the proposed graphs for retrofitting. In House C, the CCS method has a very significant improvement using retrofitted embeddings with both labeled data and EAMs location graphs. Introducing location information into the action embeddings seems useful for CCS in this scenario.

Now, we focus the discussion on the CCSD method. In House A, retrofitted embeddings improve G-Mean up to 5 points with location graphs generated from labeled data and all the graphs generated from EAMs. On the contrary, there is no improvement on CCSD's performance using this type of graphs in Houses B and C.

We believe that the performance differences between graphs are caused by the quality of the datasets annotations and the quality of the EAMs in each house. Namely, the quality of information sources is critical to adjust the embeddings effectively. For example, House B, as mentioned before, has annotation problems, what makes logical not to obtain significant improvements with graphs generated from its labeled data. Likewise, the best EAMs are the ones of House A. It is the simpler scenario and it is easier to model its expert knowledge precisely. So,

again, it is logical for us to obtain less improvements in Houses B and C, the most complex scenarios, with graphs generated from their EAMs.

In summary, we confirm that embeddings quality does affect to our CPD methods and that retrofitting could be a good technique to boost applications where abundant/excellent expert knowledge is available.

6. Conclusions and future work

From this research, we can conclude that our novel embedding-based approach is useful for unsupervised real-time CPD in smart home time series data. Moreover, as this method can be used in any situation where the time series is discretizable and does not require domain-specific feature extraction, we would like to explore its effectiveness in other fields as future work. In addition, after comparing its variants against three state of the art algorithms in three benchmark datasets, we observe that this method can segment sensor data more accurately than its competitors. In this sense, we believe that this segmentation technique can be used to boost activity recognition models even more, generating better, error-free segments and consequently, allowing those models to extract more representative features to classify activities in real-time.

We also think that, as computing cosine similarities is affordable in terms of computational cost, this method can enable many applications where computational cost is critical. Likewise, we think that the sensor/action representation with embeddings is useful to perform direct transfer learning between different environments, what other models cannot do easily. In future work, we plan to empirically explore these beliefs with cross-environment performance experiments.

Regarding possible improvements, we are aware that the hyperparameter selection (threshold, minimum distance, window length) has a great impact on the models performance. The optimal values for these hyperparameter are dataset-specific, and as explained in the previous section, are fine-tuned using the training data. Currently this is a user-guided process, which could be automatized for better results. We conclude that it would be interesting to use other methods like gradient descent to find its optimum values, instead of employing constant increments. This same conclusion was withdrawn by Aminikhanghahi et al. (2018) for density ratio CPD algorithms. Additionally, as we have shown that retrofitting embeddings to knowledge-graphs can improve the CPD results, we would like to explore other hybrid approaches to tweak the embeddings. Concretely, we plan to use NLP word-level embeddings to represent the actions as in our previous work (Azkune, Almeida, & Agirre, 2020). Finally, we would like to study the possibilities of deploying the algorithm in the edge (Díaz-de Arcaya, Miñón, Torre-Bastida, Del Ser, & Almeida, 2020), and studying its performance in platforms with constrained resources that could be installed in intelligent environments.

CRediT authorship contribution statement

Unai Bermejo: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Visualization. **Aitor Almeida:** Conceptualization, Methodology, Validation, Formal analysis, Writing - review & editing, Supervision, Project administration. **Aritz Bilbao-Jayo:** Conceptualization, Validation, Writing - review & editing, Visualization. **Gorka Azkune:** Validation, Data curation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was carried out with the financial support of FuturAAL-Ego (RTI2018-101045-A-C22) granted by Spanish Ministry of Science, Innovation and Universities.

References

Almeida, A., & Azkune, G. (2018). Predicting human behaviour with recurrent neural networks. *Applied Sciences*, *8*(2), 305.

Almeida, A., Azkune, G., & Bilbao, A. (2018). Embedding-level attention and multi-scale convolutional neural networks for behaviour modelling. In *2018 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computing, scalable computing & communications, cloud & big data computing, internet of people and smart city innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)* (pp. 439–445). IEEE.

Aminikhanghahi, S., & Cook, D. J. (2017a). A survey of methods for time series change point detection. *Knowledge and Information Systems*, *51*(2), 339–367.

Aminikhanghahi, S., & Cook, D. J. (2017b). Using change point detection to automate daily activity segmentation. In *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)* (pp. 262–267). IEEE.

Aminikhanghahi, S., & Cook, D. J. (2019). Enhancing activity recognition using CPD-based activity segmentation. *Pervasive and Mobile Computing*, *53*, 75–89.

Aminikhanghahi, S., Wang, T., & Cook, D. J. (2018). Real-time change point detection with application to smart home time series data. *IEEE Transactions on Knowledge and Data Engineering*, *31*(5), 1010–1023.

Díaz-de Arcaya, J., Miñón, R., Torre-Bastida, A. I., Del Ser, J., & Almeida, A. (2020). PADL: A modeling and deployment language for advanced analytical services. *Sensors*, *20*(23), 6712.

Azkune, G., & Almeida, A. (2018). A scalable hybrid activity recognition approach for intelligent environments. *IEEE Access*, *6*, 41745–41759.

Azkune, G., Almeida, A., & Agirre, E. (2020). Cross-environment activity recognition using word embeddings for sensor and activity representation. *Neurocomputing*, *418*, 280–290.

Bennett, J., Rokas, O., & Chen, L. (2017). Healthcare in the smart home: A study of past, present and future. *Sustainability*, *9*(5), 840.

Ceci, M., Corizzo, R., Japkowicz, N., Mignone, P., & Pio, G. (2020). ECHAD: Embedding-based change detection from multivariate time series in smart grids. *IEEE Access*, *8*, 156053–156066. http://dx.doi.org/10.1109/ACCESS.2020.3019095.

Chaaraoui, A. A., Climent-Pérez, P., & Flórez-Revuelta, F. (2012). A review on vision techniques applied to human behaviour analysis for ambient-assisted living. *Expert Systems with Applications*, *39*(12), 10873–10888.

Chen, L., Nugent, C. D., Mulvenna, M., Finlay, D., Hong, X., & Poland, M. (2008). A logical framework for behaviour reasoning and assistance in a smart home. *International Journal of Assistive Robotics and Mechatronics*, *9*(4), 20–34.

Chernbumroong, S., Cang, S., Atkins, A., & Yu, H. (2013). Elderly activities recognition and classification for applications in assisted living. *Expert Systems with Applications*, *40*(5), 1662–1674.

Cleland, I., Han, M., Nugent, C., Lee, H., McClean, S., Zhang, S., et al. (2014). Evaluation of prompted annotation of activity data recorded from a smart phone. *Sensors*, *14*(9), 15861–15879.

Cook, D. J., & Krishnan, N. C. (2015). *Activity learning: Discovering, recognizing, and predicting human behavior from sensor data*. John Wiley & Sons.

Cook, D. J., Krishnan, N. C., & Rashidi, P. (2013). Activity discovery and activity recognition: A new partnership. *IEEE Transactions on Cybernetics*, *43*(3), 820–828.

De La Concepción, M. Á., Morillo, L. S., Gonzalez-Abril, L., & Ramírez, J. O. (2014). Discrete techniques applied to low-energy mobile human activity recognition. A new approach. *Expert Systems with Applications*, *41*(14), 6138–6146.

Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., & Smith, N. A. (2014). Retrofitting word vectors to semantic lexicons. In *Proceedings of NAACL*.

Feng, S., & Duarte, M. F. (2019). Few-shot learning-based human activity recognition. *Expert Systems with Applications*, *138*, Article 112782.

Feuz, K. D., Cook, D. J., Rosasco, C., Robertson, K., & Schmitter-Edgecombe, M. (2014). Automated detection of activity transitions for prompting. *IEEE Transactions on Human-Machine Systems*, *45*(5), 575–585.

Gómez-Carmona, O., Casado-Mansilla, D., Kraemer, F. A., López-de Ipiña, D., & García-Zubia, J. (2020). Exploring the computational cost of machine learning at the edge for human-centric Internet of Things. *Future Generation Computer Systems*, *112*, 670–683.

Han, M., Lee, Y.-K., Lee, S., et al. (2012). Comprehensive context recognizer based on multimodal sensors in a smartphone. *Sensors*, *12*(9), 12588–12605.

Hido, S., Idé, T., Kashima, H., Kubo, H., & Matsuzawa, H. (2008). Unsupervised change analysis using supervised learning. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 148–159). Springer.

Huang, S., Kong, Z., & Huang, W. (2014). High-dimensional process monitoring and change point detection using embedding distributions in reproducing kernel Hilbert space. *IIE Transactions*, *46*(10), 999–1016.

Kwon, Y., Kang, K., & Bae, C. (2014). Unsupervised learning for human activity recognition using smartphone sensors. *Expert Systems with Applications*, *41*(14), 6067–6074.

Laguna, J. O., Olaya, A. G., & Borrajo, D. (2011). A dynamic sliding window approach for activity recognition. In *International conference on user modeling, adaptation, and personalization* (pp. 219–230). Springer.

Liu, S., Yamada, M., Collier, N., & Sugiyama, M. (2013). Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, *43*, 72–83.

Ma, Y., Campbell, A. T., Cook, D. J., Lach, J., Patel, S. N., Ploetz, T., et al. (2020). Transfer learning for activity recognition in mobile health. arxiv preprint arXiv:2007.06062.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arxiv preprint arXiv:1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 746–751).

Okeyo, G., Chen, L., Wang, H., & Sterritt, R. (2014). Dynamic sensor data segmentation for real-time knowledge-driven activity recognition. *Pervasive and Mobile Computing*, *10*, 155–172.

Ranasinghe, S., Al Machot, F., & Mayr, H. C. (2016). A review on applications of activity recognition systems with regard to performance and evaluation. *International Journal of Distributed Sensor Networks*, *12*(8), Article 1550147716665520.

Rashidi, P., & Cook, D. J. (2013). COM: A method for mining and monitoring human activity patterns in home-based health monitoring systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *4*(4), 1–20.

Schank, R. C. (1983). *Dynamic memory: A theory of reminding and learning in computers and people*. cambridge university press.

Sprint, G., Cook, D. J., Fritz, R., & Schmitter-Edgecombe, M. (2016). Using smart homes to detect and analyze health events. *Computer*, *49*(11), 29–37.

Synnott, J., Chen, L., Nugent, C. D., & Moore, G. (2014). The creation of simulated activity datasets using a graphical intelligent environment simulation tool. In *2014 36th annual international conference of the IEEE engineering in medicine and biology society* (pp. 4143–4146). IEEE.

Szewcyzk, S., Dwan, K., Minor, B., Swedlove, B., & Cook, D. (2009). Annotating smart environment sensor data for activity learning. *Technology and Health Care*, *17*(3), 161–169.

Triboan, D., Chen, L., Chen, F., & Wang, Z. (2017). Semantic segmentation of real-time sensor data stream for complex activity recognition. *Personal and Ubiquitous Computing*, *21*(3), 411–425.

Triboan, D., Chen, L., Chen, F., & Wang, Z. (2019). A semantics-based approach to sensor data segmentation in real-time activity recognition. *Future Generation Computer Systems*, *93*, 224–236.

Van Kasteren, T. L., Englebienne, G., & Kröse, B. J. (2011). Human activity recognition from wireless sensor network data: Benchmark and software. In *Activity recognition in pervasive intelligent environments* (pp. 165–186). Springer.

Van Kasteren, T., Noulas, A., Englebienne, G., & Kröse, B. (2008). Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on ubiquitous computing* (pp. 1–9).

Weinland, D., Ronfard, R., & Boyer, E. (2011). A survey of vision-based methods for action representation, segmentation and recognition. *Computer Vision and Image Understanding*, *115*(2), 224–241.

Yilmaz, A., Javed, O., & Shah, M. (2006). Object tracking: A survey. *Acm Computing Surveys (CSUR)*, *38*(4), 13–es.