DCR: a new distributed model for human activity recognition in smart homes

Amina Jarraya, Amel Bouzeghoub, Amel Borgi, Khedija Arour

Version of Record: https://www.sciencedirect.com/science/article/pii/S0957417419305512
Manuscript_c5827bfc5dc8eb68291670a73919b238

DCR: a New Distributed Model for Human Activity Recognition in Smart Homes

Amina Jarraya[a,b], Amel Bouzeghoub[a], Amel Borgi[b], Khedija Arour[b,c]

[a] *SAMOVAR, Télécom SudParis, CNRS, Paris-Saclay University, Evry, France*
[b] *University of Tunis El Manar, Faculty of Sciences of Tunis, LIPAH, Tunis, Tunisia*
[c] *College of Business, University of Jeddah, Jeddah, Saudi arabia*

Abstract

Human Activity Recognition (HAR) is an important research issue for pervasive computing that aims to identify human activities in smart homes. In the literature, most reasoning approaches for HAR are based on centralized approach where a central system is responsible for processing and reasoning about sensor data in order to recognize activities. Since sensor data are distributed, heterogeneous, and dynamic (i.e., whose characteristics are varying over time) in the smart home, reasoning process on these data for HAR needs to be distributed over a group of heterogeneous, autonomous and interacting entities in order to be more efficient. This paper proposes a main contribution, the DCR approach, a fully Distributed Collaborative Reasoning multi-agent approach where agents, with diverse classifiers, observe sensor data, make local predictions, communicate and collaborate to identify current activities. Then, an improved version of the DCR approach is proposed, the DCR-OL approach, a distributed Online Learning approach where learning agents learns from their collaborations to improve their own performance in activity recognition. Finally, we test our approaches by performing an evaluation study on Aruba dataset, that indicates an enhancement in terms of accuracy, F-measure and G-mean metrics compared to the centralized approach and also compared to a distributed approach existing

Email addresses: amina.jarraya@telecom-sudparis.eu (Amina Jarraya), amel.bouzeghoub@telecom-sudparis.eu (Amel Bouzeghoub), Amel.Borgi@insat.rnu.tn (Amel Borgi), kbarour@uj.edu.sa (Khedija Arour)

in the literature.

Keywords: Human activity recognition, Distributed reasoning, Multi-agent system, Smart homes, Online learning.

1. Introduction

HAR is an emerging research field in smart environments (Ranasinghe et al., 2016; Ziaeefard and Bergevin, 2015) through which the performed human activities can be identified. Various approaches use reasoning techniques for activity modeling and recognition such as ontological or semantic reasoning (Noor et al., 2016; Chen et al., 2012), probabilistic reasoning (Riboni et al., 2016; Wang and Ji, 2014), evidential reasoning (McKeever et al., 2010; Sebbak et al., 2014) or fuzzy reasoning (Rodriguez et al., 2014; Abdelhedi et al., 2016). The common characteristic of these approaches is that they are centralized for processing and reasoning with sensor data. The centralized approach integrates a recognition model already built and identifies activities as the environment changes. Nevertheless, connections between sensors and the centralized system are not guaranteed. Moreover, handling the huge incoming sensor data from different locations in smart homes, decreases the system performance. To rectify these shortcomings, a fully decentralized approach seems to be a necessity for distributing both sensor data and reasoning process. Due to the dynamic and open nature of smart homes, the following main challenges must be considered when dealing with distributed HAR: *C1: sensor data management*: how to deal with distributed data arrival from deployed sensors in different locations of the smart home? *C2: data freshness*: how to avoid outdated data? *C3: identification*: how to identify current activities based on past person behaviors? *C4: accuracy*: how to increase the activity recognition accuracy? *C5: heterogeneity*: how to handle the nature of sensor data? *C6: uncertainty*: how to trust data coming from other sensors ? *C7: Conflict*: how to deal with contradictory data coming from different sensors? The literature review revealed few approaches that have considered distributed reasoning for HAR in smart homes (Cicirelli

et al., 2016; Wang and Ji, 2014; Mosabbeb et al., 2013; Ramakrishnan et al., 2013; Amft and Lombriser, 2011; Marin-Perianu et al., 2008). However, these few works only deal with some of the challenges discussed above. In this work, we consider the six aforementioned challenges as requirements to achieve. The aim of this paper is to propose a totally Distributed Collaborative Reasoning (DCR) approach for HAR. DCR is a novelty for distributed single-user activity recognition issue.

The layout of this paper is structured as follows. Section 2 gives recent studies on distributed HAR approaches. In section 3, our main contribution, the DCR approach, is introduced. In Section 4, the improved version of DCR is presented (the DCR-OL approach). In Section 5, several experiments are conducted over the Aruba dataset to evaluate the effectiveness of our approaches. Finally, the conclusion and future works are briefly summarized in Section 6.

2. Related works

Few recent studies have focused on distributed HAR approaches that can be grouped according to their architecture.

1)- *Client-server approach*: authors in (Cicirelli et al., 2016; Fortino et al., 2015) proposed a framework that mainly relies on the *Cloud-assisted Agent-based Smart home Environment* (CASE) architecture which is composed of three layers: IoT Layer, Virtualization Layer and Analytics Layer. The latter consists of several nodes. Each node contains a server agent that hosts quite a few agents and permits them to execute their functionalities. Agents evolve the activity recognition task which is in charge of processing the features and recognizing the high level activities in real time.

2)- *Hierarchical distributed approaches*: authors in (Amft and Lombriser, 2011) introduced an activity-event-detector (AED) for distributed activity recognition systems based on directed acyclic graphs. It is a set of distributed sensing and detection nodes (detectors) where each detector performs local data acquisition, atomic activity spotting and communicates event-type information to other net-

work nodes. In addition, authors in (Marin-Perianu et al., 2008) proposed an activity recognition architecture based on fuzzy logic, through which multiple nodes collaborate to produce a reliable recognition result from unreliable sensor data. The three main steps are: the detection of simple events, the combination of events into basic operations and the final activity classification.

3)- *Totally distributed approaches*: authors in (Wang and Ji, 2014) proposed a distributed abnormal activity detection approach (*DetectingAct*) which employs sensor nodes to detect abnormal activities. In *DetectingAct*, a node detects normal activities using frequent pattern mining technique (FP-tree algorithm). Then, the activity is labeled by each node it passes through. Its status becomes abnormal as soon as it is tagged as abnormal by at least one node. In addition, authors, in (Mosabbeb et al., 2013), proposed a multi-view distributed SVM model for camera sensor networks where different learning nodes integrate SVM classifiers. In order to pull the learning results of different views together, a regularization is used to concatenate all views. Finally, authors in (Ramakrishnan et al., 2014, 2013) proposed a modular and distributed Bayesian framework which is a collection of Bayesian Networks (BN) where each individual BN models a unique high-level context. The parent node in each individual BN is the high-level context node to be inferred and their respective children nodes correspond to the sensors used to infer them.

Table 1 summarizes distributed HAR systems discussed above from the recent to the oldest one.

Despite their major differences regarding their architecture, all these approaches propose a distributed HAR where sensor data are processed in a bottom-up manner to detect activities. Moreover, in some approaches, entities (nodes or agents) use data-driven models (KNN and SVM classifiers, FP-tree) for data reasoning that allow the use of large-scale datasets of sensors to learn activity models. However, they present some limitations:

− All entities adopt the same type of reasoning model. The choice of this one depends on the nature of sensor data. However, sensor data differ from a location to another and then the reasoning model has to be different from an entity

Table 1. Distributed HAR approaches

System	*Distributed system*	*Reasoning model*	*Architecture*	Communication	*Input*	Collaboration
CASE system (Cicirelli et al., 2016; Fortino et al., 2015)	Multi-agent system	KNN classifiers	Client-Server	Server agent-Agents	A set of events {date, timestamp, sensorId, status}	No collab. between server agents
DetectingAct System (Wang and Ji, 2014)	A set of sensor nodes	FP-tree algorithm	Totally distributed	A node-a subset of nodes	Trajectory and duration data	Yes (sharing with neighbors to mark a normal or an abnormal activity)
Multi-view Distributed SVM System (Mosabbeb et al., 2013)	A set of camera sensor nodes	SVM classifiers	Totally distributed	A node-all nodes	Camera data (IXMAS dataset)	Yes (regularization of different views)
Distributed Bayesian Framework (Ramakrishnan et al., 2014, 2013)	A set of nodes	Bayesian networks	Totally distributed	A node-all nodes	Tri-axial accelerometer, binary sensors, etc.	Yes (generating a global view)
Distributed recognition system (Amft and Lombriser, 2011)	A set of nodes	AED graph	Hierarchical distributed	Parent-children	Detectors data	Yes (detected events are communicated)
Distributed HAR fuzzy-enabled system (Marin-Perianu et al., 2008)	A set of sensor nodes (detectors)	Rule-based fuzzy inference	Hierarchical distributed	Parent-children	Trial accelerometers	No collab. between nodes in a same layer
Our approach	Multi-agent system	Different type of classifiers built upon different training data	Totally distributed	An agent-a set of agents	A set of events {date, timestamp, sensorId, status} (Aruba dataset)	Yes (aggregating the final activity)

to another.

− For totally distributed approaches, 3^{rd} and 4^{th} approaches present a communication with all entities. On the other hand, in the 2^{nd} approach, the communication is possible with some predefined entities established in advance. In order to, respectively, overcome the high costs and to meet the real-time needs, the communication should be dynamically defined for a set of entities.

− Uncertainty is not commonly addressed in these approaches. Entities trust all data provided from the others when collaborating.

Table 2 positions these different approaches according to the challenges addressed in the *Introduction* section. These approaches deal with some of challenges.

In this regard, we propose the DCR approach that tackles the seven challenges by providing the following contributions:

−**C1 (sensor data management)** → **P1**: a fully distribution of reasoning process and data over heterogeneous, autonomous and interacting entities.

−**C2 (data freshness)** →**P2**: a bottom-up approach should guarantee data freshness.

−**C3 (identification)** → **P3**: enriching agents with classifiers as activity models.

−**C4 (accuracy)** → **P4**: ensuring communication and collaboration with a dynamic set of agents in order to increase the recognition accuracy.

−**C5 (heterogeneity)** → **P5**: adopting agents with different types of activity models built upon different training data regarding the nature of sensor data.

−**C6 (uncertainty)** → **P6**: assigning a trust degree for recognized activities.

−**C7 (conflict)** → **P7**: three conflict resolution strategies are proposed.

3. The DCR approach

The DCR approach was briefly described in (Jarraya et al., 2018). However, this paper describes in detail the DCR model and its algorithms.

Table 2. Position of the different approaches according to cited challenges

Systme	*D1: sensor data management*	*D2: data freshness*	*D4: accuracy*	*D5: heterogeneity*	*D6: uncertainty*	*D7: conflict*
CASE system Cicirelli et al. (2016); Fortino et al. (2015)	✓	✗	✓	✗	✗	✗
DetectingAct system Wang and Ji (2014)	✓	✓	✓	✗	✗	✓
Multi-view Distributed SVM system Mosabbeb et al. (2013)	✓	✗	✓	✗	✗	✓
Distributed Bayesian Framework ?Ramakrishnan et al. (2014, 2013)	✓	✗	✓	✗	✗	✓
Distributed Recognition System Amft and Lombriser (2011)	✗	✓	✗	✗	✗	✓
Distributed HAR fuzzy-enabled system Marin-Perianu et al. (2008)	✗	✗	✗	✗	✗	✗
Our approach	✓	✓	✓	✓	✓	✓

3.1. System overview

The whole framework for distributed HAR is illustrated in Figure 1. The agent paradigm was adopted to architect our distributed system that aims at supporting activity recognition process in smart homes. Multi-agent systems are a well-adapted paradigm in modeling smart homes, facilitating local data processing and the final decision is made by a collaboration (Maciel et al., 2015).

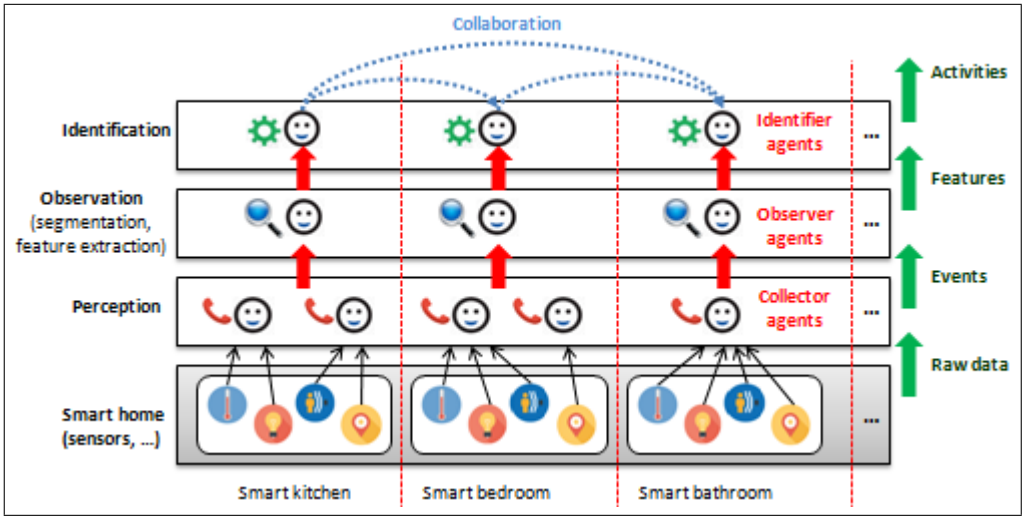


Figure 1. System overview of distributed HAR

We assume that a smart home is composed of different smart zones that deploys heterogeneous sensors (e.g, motion sensors, presence sensors). A zone can be the set of bedrooms or the living room plus the corridor or simply a kitchen.

The architecture of the proposed framework is composed of three main modules:

Perception: more than one collector agent can be deployed in the same zone that incorporate a CEP (Complex Event Processing) engine (Luckham, 2008). These agents gather raw data sensor coming from one or more sensors and produce events represented by a quadruple {date, timestamp, sensorId, status (on/off)}.

Observation: one observer agent is deployed by zone. Observer agents segment the incoming event list into segments. Then, they extract a feature vector from each segment (segmentation and feature extraction techniques are detailed in this survey (Alzahrani and Kammoun, 2016)).

Identification: aims to recognize the occurring activities. In this work, the first two modules are not in the scope and the focus is on the **identification** module which is the core of our proposed DCR approach.

Actually, DCR models a smart home as a multi-agent system $MAS = \{A_{l_1}, A_{l_2}, ..., A_{l_n}\}$ where each agent A_{l_i} is assigned to a zone l_i. Agents are autonomous and have the same functionalities. Each agent $A_{l_i} \in MAS$ is defined as a tuple $A_{l_i} = (id_{A_{l_i}}, clf_{A_{l_i}}, ACT_{A_{l_i}}, ACQ_{A_{l_i}})$ where:

– $id_{A_{l_i}}$: is the identifier of the agent A_{l_i} in the location l_i.

– $clf_{A_{l_i}}$: is the classifier associated to the agent A_{l_i}. It is built from past feature vectors labeled with performed activities in the zone l_i. Thus, agents in MAS hold different type of classifiers built upon different training data. The choice of the classifier type is detailed in the *Simulation and evaluation* section.

– $ACT_{A_{l_i}} = \{< a_{1_{(l_i)}}, d_{1_{(l_i)}} >, < a_{2_{(l_i)}}, d_{2_{(l_i)}} >, ..., < a_{m_{(l_i)}}, d_{m_{(l_i)}} >\}$: is the set of activities $a_{1_{(l_i)}}, a_{2_{(l_i)}}, ..., a_{m_{(l_i)}}$ known by the classifier $clf_{A_{l_i}}$ with their trust degrees $d_{1_{(l_i)}}, d_{2_{(l_i)}}, ..., d_{m_{(l_i)}}$. These ones express the truth degree of an activity. This list is fixed when building the classifier $clf_{A_{l_i}}$ and the trust degree is determined using the *F1-score* metric (Ting, 2010) for each performed activity in l_i.

– $ACQ_{A_{l_i}}$: is the acquaintance list of the agent A_{l_i}. It's dynamically adjusted and contains agents who can recognize the local predicted activity by A_{l_i}.

Each agent A_{l_i} has a basic life cycle (Figure 2) which includes the following steps to recognize activities:

– Given as input a feature vector FV arrived over time in l_i, the agent A_{l_i} interrogates its classifier $clf_{A_{l_i}}$ to get the local predicted activity $pa_{A_{l_i}}$.

– After that, the agent A_{l_i} determines the trust degree d_{pa} related to the activity $pa_{A_{l_i}}$ from $ACT_{A_{l_i}}$ list.

– According to d_{pa} value, we distinguish two cases:

- If $pa_{A_{l_i}}$ is well recognized with a degree $d_{pa} >= \delta$ (δ is a detection threshold chosen by the designer), the agent will not collaborate with other agents and will generate as an output the activity $Fpa_{A_{l_i}}$ which is $pa_{A_{l_i}}$.
- If $pa_{A_{l_i}}$ is recognized with a degree $d_{pa} < \delta$, the agent proceeds by the following steps:

- Building its acquaintance list $ACQ_{A_{l_i}}$. This one contains agents who can recognize $pa_{A_{l_i}}$.
- Sending its input to some agents in $ACQ_{A_{l_i}}$. These ones are selected if their trust degrees of the related activity are higher than d_{pa}.
- Receiving foreign activity predictions with their trust degrees from selected agents.
- Applying conflict resolution strategies when the local prediction $pa_{A_{l_i}}$ and foreign predictions are in a disagreement.
- Generating as an output the final activity $Fpa_{A_{l_i}}$ which can be different from $pa_{A_{l_i}}$.

The following subsection provides more detailed information through the proposed algorithms.

3.2. The DCR algorithms

An agent in the DCR system can be triggered in two cases as follows:

- When there is a new incoming feature vector (the input). In this case, it's called the *starter agent*.
- Following the request of another agent. In this case, it's called the *receiver agent*. The request is set off for two reasons:
  - In order to determine its acquaintance list $ACQ_{A_{l_i}}$, the starter agent A_{l_i} requests all other agents to check if its predicted activity belongs to their activities list $ACT_{A_{l_j}}$ $(j \neq i)$.
  - In order to determine the final output, the starter agent A_{l_i} collaborates with some agents in its acquaintance list $ACQ_{A_{l_i}}$ by sending its input to get their predictions.

The sequence diagram of DCR approach is described in Figure 3.

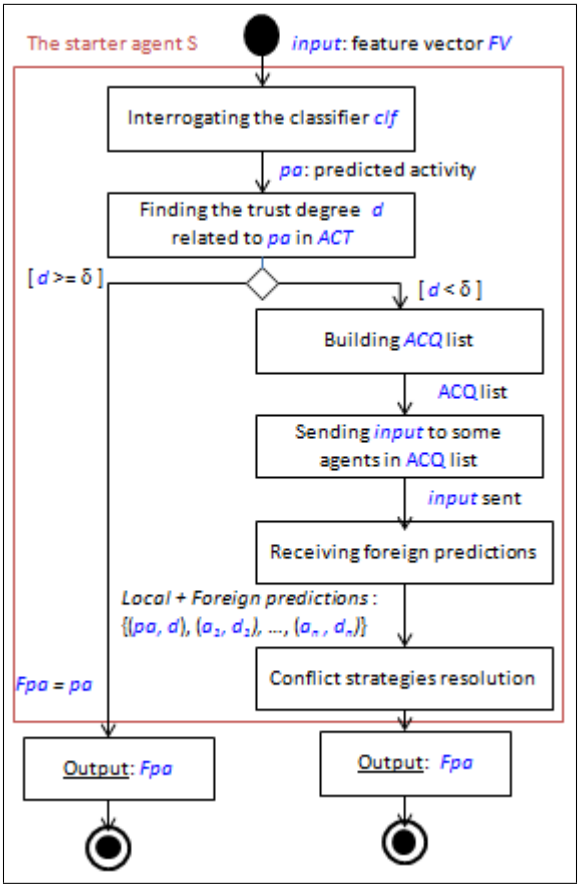


Figure 2. The state diagram of the DCR approach

3.2.1. The starter agent algorithms (three algorithms)

Processing feature vector input (Algorithm 1) This algorithm takes as input the feature vector FV and generates a list of predicted activities, source agents and their trust degrees $pAct\_list$. This agent S starts by loading its classifier clf (line 3), predicting the activity pa_S (line 4) and determining its trust degree $d_{pa_{(S)}}$ (line 5). If this one is greater than the threshold δ then the starter agent S recognizes successfully the activity pa_S (line 6). It's useless to collaborate with other agents because it is time consuming and may decrease the recognition accuracy of pa_S. If the trust degree is less than the threshold δ then the agent S sends a message containing the predicted activity pa_S to all

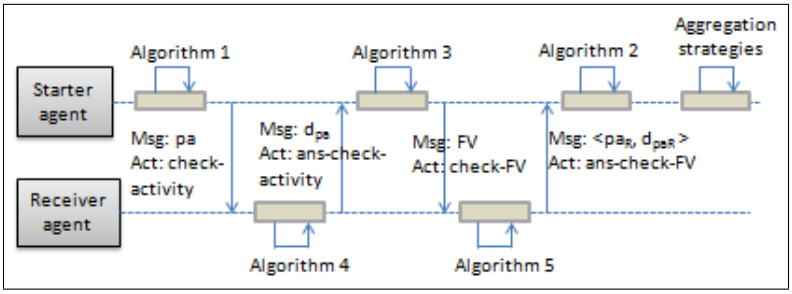


Figure 3. Sequence diagram of the DCR approach

other agents in order to find agents that are able to recognize it (lines 9-10).

Algorithm 1. Processing feature vector algorithm

Input:

FV: a feature vector; S: a sender agent; O: all other agents; δ: a threshold

$ACT_S = \{< a_1, d_1 >, < a_2, d_2 >, ..., < a_m, d_m >\}$: a list of recognized activities with their trust degrees

Output:

$pAct\_list = (< a_1, S, d_1 >, < a_2, R_1, d_2 >, ..., < a_n, R_n, d_n >)$: list of predicted activities, source agents and their trust degrees

```
1  begin
2      initialise(pAct_list)
3      clf = loadClassifier()
4      pa_S = predictActivity(clf, FV)
5      d_pa(S) = getDegreeFrom(pa_S, ACT_S)
6      if d_pa(S) ≥ δ then
7          pAct_list.add(< pa_S, S, d_pa(S) >)
8      else
9          for o_i ∈ O do
10             Send message(pa_S, S, o_i, act : "check − activity")
11     return(pAct_list)
```

Processing answer for request "check-activity" (Algorithm 3) This algorithm aims at determining the acquaintance list of the starter agent $ACQ_{(S)}$. This list contains agents that are able to recognize the predicted activity pa_S. The algorithm takes as input the reply message from the receiver agent R during a period Δt. If R can recognize pa_S, the message will contain the related trust degree $d_{pa_{(R)}}$ of pa_S. After Δt, no message is processed.

Processing answer for request "check-FV" (Algorithm 2) The starter agent receives all predicted activities with their trust degrees from selected agents in $ACQ_{(S)}$. In fact, the message msg contains the answer from the selected agent that correctly identifies the pa_S (line 3). The starter agent S adds the answer to $pAct\_list_{(S)}$ and overwrites the last line of the result file $rFILE$ by concatenating old predicted activities with the new one. The result file $rFILE$ contains rows where each row is the list of predicted activities, the source agents and their trust degrees for each FV.

Algorithm 2. Processing answer for request *check-FV*

Input:
msg: received message; S: sender agent; R: receiver agent
$pAct\_list_S$: list of predicted activities, the source agents and their trust degrees
Output:
$rFILE$: result file

```
1 begin
2     if act = "ans − check − FV" then
3         < pa_(R), R, d_(R) >= msg.getContent()
4         pAct_list_(S).add(< pa_(R), R, d_(R) >)
5         last_line = read_last_line(rFILE)
6         concat(last_line, < pa_(R), R, d_(R) >)
7         replaceLastLine(rFILE, last_line)
8     return(rFILE)
```

3.2.2. The receiver agent algorithms (two algorithms)

Processing request "check-activity" (Algorithm 4): this algorithm determines if the receiver agent can recognize the predicted activity of the sender agent $pa_{(S)}$. In fact, the receiver agent checks if $pa_{(S)}$ is in its activity list $ACT_{(R)}$. If it is the case, it answers the sender agent by sending the related trust degree $d_{(R)}$ (lines 5, 6 and 7). Otherwise, it does not send anything.

Processing request "check-FV" (Algorithm 5): the selected receiver agent who can recognize the predicted activity $pa_{(S)}$ better than the starter agent, will receive a message from the latter. This message is a request to check $FV_{(S)}$ which activity corresponds. Thus, the receiver agent loads its classifier $clf_{(R)}$ (line 4), predicts the activity $pa_{(R)}$ according to the $FV_{(S)}$ (line

Algorithm 3. Processing answer for request *check-activity*

Input:

msg: received message; R: receiver agent; S: sender agent; act: action; FV: feature vector; $d_{pa_{(S)}}$: trust degree of pa_S; $startTime$: processing start time; Δt: max period of processing

Output:

$ACQ_{(S)}$: acquaintance list

```
1  begin
2      period = 0
3      if (act = "ans − check − activity") then
4          while period < Δt do
5              d_pa(R) = msg.getContent()
6              if d_pa(R) is not ∅ then
7                  ACQ(S).add(R)
8                  if d_pa(R) > d_pa(S) then
9                      Send message(FV, S, R, act : "check − FV")
10             period = getCurrentTime() − startTime
11     return(ACQ(S))
```

5), determines its trust degree $d_{(R)}$ (line 6) and answers the starter agent by sending a message containing the couple of its predicted activity with its trust degree $< pa_{(R)}, d_{(R)} >$ (line 7).

Algorithm 4. Processing request *check-activity*

Input:

msg: received message; S: sender agent; R: receiver agent; act: action

Output:

$d_{pa_{(R)}}$: trust degree

```
1  begin
2      if (act = "check − activity") then
3          pa(S) = msg.getContent()
4          ACT(R) = R.getACTlist()
5          if pa(S) ∈ ACT(R) then
6              d_pa(R) = R.getDegreeFrom (pa(S), ACT(R))
7              Send message(d_pa(R), R, S, act : "ans − check − activity")
8      return(d_pa(R))
```

Algorithm 5. Processing request *check-FV*

Input:

msg: received message; S: sender agent; R: receiver agent; act: action; $clf_{(R)}$: receiver classifier

Output:

$< pa_{(R)}, d_{(R)} >$: couple of predicted activity and its trust degree of the receiver agent

```
1 begin
2     if (act = "check − FV") then
3         FV_(S) = msg.getContent()
4         clf_(R) = loadClassifier()
5         pa_(R) = predictActivity(clf_(R), FV_(S))
6         d_(R) = getDegreeFrom(ACT_(R), pa(R))
7         Send message(< pa_(R), d_(R) >, S, act : "ans − check − FV")
8     return(< pa_(R), d_(R) >)
```

3.3. Properties of the DCR algorithms

In this section, we discuss some properties of the DCR algorithms regarding their termination, their number of messages and their complexity.

Termination: we assume a multi-agent system $MAS = \{A_{l_1}, A_{l_2}, ..., A_{l_n}\}$ with a finite number of agents in the system.

Number of messages: Given n as the total number of agents, the exchange of messages between the starter agent and receiver agents entails the following two steps:

- When fixing acquaintance list: exchange of messages between the starter agent and all other agents. In this case, the number of messages is equal to $2 \times (n-1)$.
- During the collaboration with some agents within the acquaintance list: in the worst case, all agents in the acquaintance list identify properly the predicted activity. Thus, the number of messages stands at $2 \times (n-1)$.

Therefore, the number of messages exchanged between the starter agent and the other receiver agents for a feature vector is equal to: $NbrMsg = 4 \times (n-1)$. Assuming that we have m_{FV} feature vectors, the total number of message exchanged is: $ToT = m_{FV} \times NbrMsg = 4 \times (n-1) \times m_{FV} \simeq \mathcal{O}(n \times m_{FV}) = \mathcal{O}(m_{FV})$ where $|n| << |m_{FV}|$

Complexity analysis: in the worst case, the computational complexity of the DCR algorithms on a single agent is proportional to the number of agents n and to the number of feature vectors m_{FV}: $\mathcal{O}(n \times m_{FV}) = \mathcal{O}(m_{FV})$ where $|n| << |m_{FV}|$.

3.4. Conflict resolution strategies

In DCR, the conflict may occur when the starter agent S receives foreign predicted activities from receiver agents R_j which are different from the local predicted activity $pa_{(S)}$. These received activities exist in the list $pAct\_list_{(S)} = \{(pa_{(S)}, S, d_{pa_{(S)}}), (pa_{(R_1)}, R_1, d_{pa_{(R_1)}}), ..., (pa_{(R_{(n-1)})}, R_{n-1}, d_{pa_{(R_{(n-1)})}})\}$ including $pa_{(S)}$.

Starter agents adopt a conflict resolution based on *self-modification* strategy (Barber et al., 2000). Thus, three aggregation methods are used by a starter agent S to resolve conflicts as follows:

1)−The *maximum trust degree* method (max-trust): the starter agent chooses the most confident activity which means the one having the higher mean of trust degrees.

2)−The *most frequent* method (max-freq.): the starter agent considers activity frequency. Thus, it chooses the most frequent activity in the $pAct\_list_{(S)}$ list.

3)−The *stacking* method (Wolpert, 1992) (stack.): the starter agent contains a meta-classifier which is trained on a meta-dataset. This one contains as features the set of local and foreign predictions ($pAct\_list_{(S)}$) and as the class the correct activity. The logistic regression method (Gromping, 2016) is used to build the meta-classifiers. The *stacking* method was not presented in (Jarraya et al., 2018).

3.5. Discussion and Limitations

The DCR system is a set of agents with similar functionalities that communicate and interact with each other to identify the participant's current activity in the smart home. These agents incorporate heterogeneous classifiers and predefined activities lists $ACT_{A_{l_i}}$. These elements are initialized once and will not

be updated over time. However, the DCR system must deal with the following changes that may occur over time:

1. Environment changes (e.g, a new deployed sensor, a new activity to discover, etc.)
2. The behavior change of the participant. For DCR, we assume that the frequency of the participant's behavior change is very low.

In this paper, we are interested in the second point described above. Especially, DCR must consider the participant's behavior changes by applying an online learning. Indeed, an agent must learn online and consider feedbacks resulted from collaborations with others. In other words, agents must evaluate their final predicted activity by comparing it with the real one and take actions in the future to improve the recognition accuracy.

Therefore, a new challenge appears and join the set of challenges cited in the *Introduction* section: *C8: learning*: how agents can learn from their experiences and improve their performances ?

The following section aims to propose an improved version of DCR to deal with this new challenge, by applying an online learning using learning agents. These latters are capable of learning from their collaborations. They start with some basic knowledge of the MAS system and are then able to act and adapt autonomously, through learning, to improve their own performance. This new distributed online approach is called *DCR-OL (DCR with Online Learning)*.

4. DCR-OL: the DCR approach with an online learning

In this section, we present our new DCR-OL approach by applying an online learning. First, we review two recent existing works on the online learning and discuss their differences. Then, we present the *DCR-OL* system overview. Finally, we describe the *DCR-OL* algorithm.

4.1. Related works on the online learning

When working with data streams, we should expect an infinite amount of data captured from distributed and heterogeneous sensors. Observations come in one by one and are being processed in that order. The effective use of such high data also involves a significant challenge: the nature of observed data, their labels and their relationships can change over time, known as *concept drift* (Zliobaite, 2010).

Several works have been proposed to deal with this challenge. We discuss two existing works as below:

1. Authors, in (Canzian et al., 2015), propose the *Perceptron Weighted Majority (PWM)*, a distributed online ensemble learning approach to classify observed data from distributed data sources. They focus on binary classification problems. Their system consists of multiple distributed local learners, which analyze different data streams that are correlated to a common event that needs to be classified. Each learner uses a local classifier to make a local prediction. The local predictions are then collected by each learner and combined using a weighted majority rule to output the final prediction. After making the final prediction, the learner compares the final prediction and the real one. The learner updates the aggregation weights adopting a perceptron learning rule. Indeed, if the comparaison fails, the weights of the learners that reported a wrong prediction are decreased by one unit, whereas the weights of the learners that reported a correct prediction are increased by one unit. Otherwise, the model is not modified.

   Their approach only learn an aggregation rule to handle the dynamic data streams. Classifiers maintain an aggregation rule up-to-date and are able to deal with the concept drift. They process each observation on arrival only once, without the need for storage and reprocessing chunks of data. The local classifiers are not centrally retrained (e.g., in a distributed scenario it may be expensive to retrain the local classifiers or unfeasible if

the learners are operated by different entities).

2. Authors, in (van Rijn et al., 2018), study the use of heterogeneous ensembles for data streams in a centralized way and propose the Online Performance Estimation framework, which dynamically weights the votes of individual classifiers in an ensemble. This approach is called BLAST (short for best last) which is an ensemble embodying the performance estimation framework. In fact, it consists of a group of diverse base-classifiers. Each classifier maintains a performance value which is updated by a performance estimation function. During the learning phase, the performance estimation function estimates the performance value of each classifier based on the result of comparison between the local prediction and the real one. Then, the BLAST algorithm updates the classifier with the current training instance. For each test instance, the ensemble selects a classifier with the highest performance value to make the prediction.. This selected classifier is considered as the active classifier. When several classifiers obtain the same estimated performance value, an arbitrary classifier can be selected as the active classifier.

Table 3 compares the difference points of the two approaches described above.

Table 3. (Canzian et al., 2015) vs (van Rijn et al., 2018)

Criteria	*(Canzian et al., 2015)*	*(van Rijn et al., 2018)*
Field	Binary classification problems	Multi-class classification problems
Identification of the final prediction	Combining the local predictions of learners and aggregating the final prediction	Selecting an active classifier with the highest performance value to make the final prediction
Online learning	If the final prediction is different from the real prediction, the weights of the learners are modified	The performance values of base-classifiers are instantly up-to-date during the learning phase
Updating classifiers	No update for the distributed classifiers, only the aggregation rule is updated	Instant update of the classifiers when observations arrive during the learning phase

In this regard, we present our *DCR-OL* approach which addresses the fol-

lowing propositions:

- The focus is on activity recognition field (multi-class identification problem).
- Distributed learning agents are used, each has a performance value.
- The final activity is the aggregation of several predictions (local and received predictions) resulted from a new aggregation strategy called *the maximum weighted performance* strategy.
- The performance values of agents are updated if the final prediction is different from the real prediction.
- Classifiers will not be modified. Only the performance values of agents change when new feature vectors arrive over time.

4.2. The DCR-OL model

The *DCR-OL* model adopts the same representation as the DCR system. Except, it replaces intelligent agents by learning agents and modifies the definition of the agent A_{l_i} by adding a fifth component. This latter is a list of the performance values of all agents in MAS. The purpose of adding this list is to evaluate the performance of agents in terms of their predictions comparing to the real one.

Thus, *DCR-OL* is a set of distributed learning agents LA_i ($i \in n; n$ is the number of learning agents in MAS) in the smart home. Each learning agent LA_i is defined as a tuple $LA_i = (id_{LA_i}, clf_{LA_i}, ACT_{LA_i}, ACQ_{LA_i}, PERF_{LA_i})$ where $PERF_{LA_i} = \{p_{LA_1}, p_{LA_2}, ..., p_{LA_n}\}$ is a list of the performance values of each agent LA_i in the MAS system. We note that $p_{LA_i} \in [0, 1]$, with better performing agents obtaining a higher score in terms of prediction.

The $PERF_{LA_i}$ list is specific for each agent LA_i and depends only on agent's context. It is different from an agent to another. When initializing the *DCR-OL* system, this list is initialized with the value 1; we assume that all learning agents in MAS are performant.

The learning agent LA_i reasons with the same way as the agent A_{l_i} in the DCR system. When a new feature vector FV arrives, the learning agent LA_i named the starter learning agent SL_i, asks its local classifier to get the local predicted activity with its trust degree. If the latter is higher than the threshold δ, the learning agent will not collaborate with other agents and will generate as an output the corresponding activity.

If the trust degree is less than the threshold δ, the agent SL_i collaborates with some agents in its acquaintances list to get their predictions. If the local predicted activity is different from the received predictions, the agent SL_i will apply a conflict resolution strategy, called *the maximum weighted performance max-wPerf* strategy to make a final decision.

After generating the final prediction, the agent SL_i must learn from feedbacks received from other agents. It has to adapt its performance list $PERF_{SL_i}$. Thus, it starts by comparing its final prediction Fpa_{SL_i} with the real prediction $RP_{(FV)}$. We assume that the latter is retrieved during the interaction with the participant who confirms or contradicts the final predicted activity. For this purpose, we distinguish two cases:

1. If the final prediction Fpa_{SL_i} is different from the real prediction $RP_{(FV)}$, $PERF_{SL_i}$ will be modified. In the following, we describe how to update the performance values of different learning agents.
2. If the final prediction Fpa_{SL_i} is the same to the real prediction $RP_{(FV)}$, $PERF_{SL_i}$ will not be modified.

Updating the performance list $PERF_{SL_i}$

Performance values of different learning agents in $PERF_{SL_i} = \{p_{LA_1}, p_{LA_2}, ..., p_{LA_n}\}$ list will be updated according to the performance measure *Perf-measure* used by the $BLAST$ approach in van Rijn et al. (2018) as follow:

$$p_{LA_i} = (p_{LA_i} \times \alpha) + \{(1 - Loss(pa_{LA_i}(FV_{(SL_i)}), RP(FV_{(SL_i)}))) \times (1 - \alpha)\}$$

Where:

- α : the fading factor gives a high importance to recent predictions, whereas the importance fades away when they become older. This parameter is chosen by the expert and is initialized with the value 0.999. The choice of this value is detailed in (van Rijn et al., 2018).

- $Loss()$: is a zero/one loss function, giving a penalty of 1 to all misclassified feature vectors.

- $pa_{LA_i}(FV_{(SL_i)})$: the local predicted activity of the agent LA_i according to the corresponding feature vector $FV_{(SL_i)}$.

- $RP(FV_{(SL_i)})$: the real activity of the corresponding feature vector $FV_{(SL_i)}$.

Thus, the learning agent SL_i considerates feedbacks received from other learning agents by using the performance measure *Perf-measure* and updates its performance list $PERF_{SL_i}$.

The maximum weighted performance strategy max-wPerf

The performance list $PERF_{SL_i}$ of the starter learning agent SL_i will be used by the new conflict resolution strategy *max-wPerf*. The latter aggregates the final prediction by computing the maximum weighted performance of each predicted activity in the $pAct\_list(SL_i)$ list.

Indeed, the weighted performance $PP_{(pa_j)}$ of each activity $PP_{(pa_j)}$ in the list $pAct\_list(SL_i)$ is defined as below:

$$PP_{(pa_j)} = \frac{\sum_{k=1}^{K} p_{LA_k}(pa_j)}{\frac{|pAct\_list_{(SL_i)}|}{\sum_{f=1} p_{LA_f}}}$$

where:

- j: index of the predicted activity in the list $pAct\_list(SL_i)$.

- K: number of learning agents who predicted the activity pa_j.

The final predicted activity Fpa_{SL_i} is the predicted activity pa_j with the highest value of $PP_{(pa_j)}$ in the $pAct\_list(SL_i)$.

Figure 4 describes two steps: the learning step in which the learning agent compares its final prediction with the real prediction. If this comparison fails, the agent SL_i updates its $PERF_{SL_i}$ list by computing the performance value p_{LA_i} of each learning agent LA_i existing in the list $pAct\_list(SL_i)$. The testing step consists in applying the maximum weighted performance strategy *max-wPerf* considering the updated list $PERF_{SL_i}$.

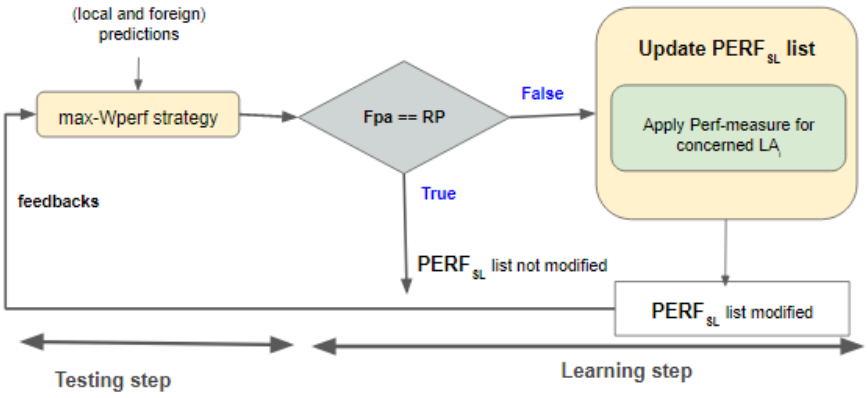


Figure 4. The DCR-OL approach

Figure 5 describes the life cycle of a starter learning agent SL_i in the *DCR-OL* approach and shows its contribution according to the *DCR* approach by adding the feedbacks loop.

4.3. The DCR-OL algorithm

The DCR-OL algorithm (Algorithm 6) describes the new online conflict resolution method which is the maximum weighted performance strategy *max-wPerf*. The starter learning agent SL_i applies this online algorithm to generate the final prediction from the $pAct\_list(SL_i)$ list which contains the local prediction and all the received predictions. This algorithm consists of two main steps after the initialization step of the $PERF_{SL_i}$ list (line 2):

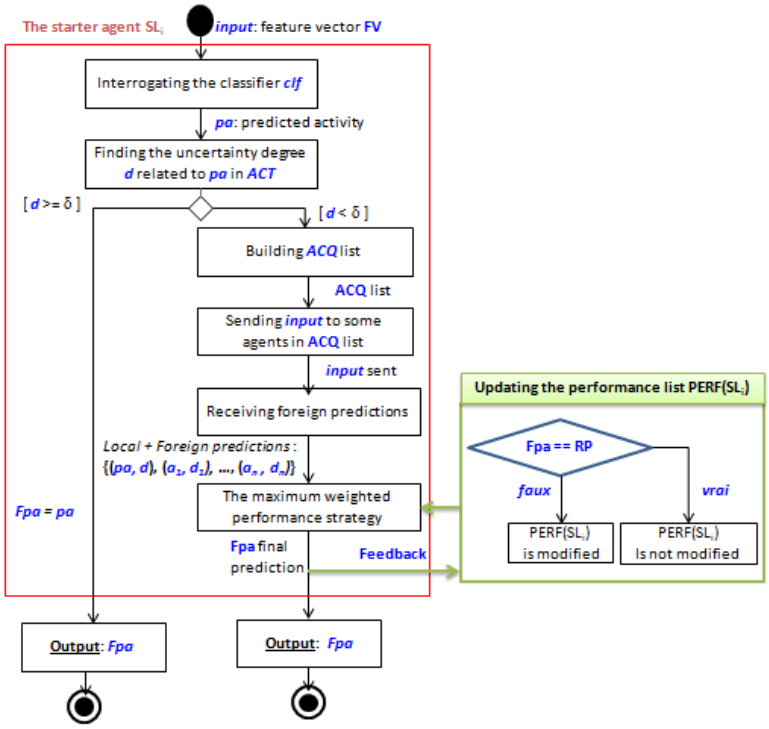


Figure 5. The state diagram of the DCR-OL approach

1. Generate the final prediction Fpa_{SL_i} by applying the *max-wPerf* strategy (line 4). This step is performed via the $maxWeightedPerformance$ method (algorithm 9).
2. Update the $PERF_{SL_i}$ list by evaluating the concerned agents int $pAct\_list(SL_i)$ list (line 5). This step is performed via the $updatePerformanceValuesmethod$ (algorithm 10).

We detail these two methods as follows:

- The $maxWeightedPerformance$ method: computes the weighted performance of each activity in the $pAct\_list(SL_i)$ list (line 4) and generates the final activity with the highest weighted performance (line 6).
- The $updatePerformanceValues$ method: updates the the $PERF_{SL_i}$ list of the starter agent SL_i. Indeed, this method retrives predictions of agents

Algorithm 6. DCR-OL: the DCR approach with an online learning

Input:

SL_i : the starter learning agent; n : number of agents in MAS; $pAct\_list_{(LS_i)}$: list of triplets (predicted activity, source agent, trust degree); α : fading factor;

```
1  begin
       // Initialization of PERF_{SL_i}
2      set PERF_{SL_i} = {p_{SL_i}, p_{LA_2}, ..., p_{LA_{n-1}}} ← 1
       // For all feature vectors
3      for all observations o = (FV_{SL_i}, RP(FV_{SL_i})) do
4          Fpa_{SL_i} = maxWeightedPerformance(pAct_list_{(SL_i)}, PERF_{SL_i})
5          if Fpa_{SL_i} <> RP_{(FV_{SL_i})} then
6              PERF_{SL_i} =
                 updatePerformanceValues(pAct_list_{(SL_i)}, PERF_{SL_i}, RP(FV_{SL_i}), α)
```

in the $pAct\_list(SL_i)$ (lines 2 and 3), determines their performance values according to the *Perf-measure measure* (line 4) and finally updates the $PERF_{SL_i}$ list for concerned agents (line 5).

Algorithm 7. the $maxWeightedPerformance$ method

Input:

$pAct\_list(SL_i)$: list of triplets (predicted activity, source agent, trust degree);

$PERF_{SL_i} = \{p_{SL_i}, p_{LA_2}, ..., p_{LA_{n-1}}\}$: list of performance values of all agents in MAS.

Output:

Fpa_{SL_i}: the final predicted activity

```
1  begin
       // List of predicted activities PaList_{SL_i}
2      PaList_{SL_i} = getDistinctPredictedActivities(pAct_list_{SL_i})
3      for each a_k in PaList_{SL_i} do
           // Compute the weighted performance value of each activity
4          PP_{a_k} = computePP(pAct_list_{SL_i}, PERF_{SL_i})
5          PPlist.add(< a_k, PP_{a_k} >)
       // Fpa_{SL_i} is the activity with the highest weighted performance value
6      Fpa_{SL_i} = maxPP(PPlist)
7      return(Fpa_{SL_i})
```

These algorithms are implemented and tested in the next section.

Algorithm 8. The $updatePerformanceValues$ method

Input:

$pAct\_list(SL_i)$: list of triplets (predicted activity, source agent, trust degree);

$PERF_{SL_i} = \{p_{SL_i}, p_{LA_2}, ..., p_{LA_{n-1}}\}$: list of performance values of all agents in MAS.

$RP_{(FV_{SL_i})}$: the real activity of FV; α : fading factor;

Output:

$PERF_{SL_i}$: performance list updated of the agent SL_i

```
1 begin
2     for all agents LA_i in pAct_list_SL_i do
          // Retrieve local predictions of agents LA_i in pAct_list_SL_i
3         LP_LA_i(FV_SL_i) = getLocalPrediction(pAct_list_SL_i, LA_i)
          // Compute the new performance value of the agent LA_i
4         p_LA_i = (p_LA_i × α) + {(1 − Loss(LP_LA_i(FV_SL_i), RP(FV_SL_i))) × (1 − α)}
          // Update PERF_SL_i of the agent SL_i
5         PERF_SL_i = updatePerformanceVector(PERF_SL_i, LA_i, p_LA_i)
```

5. Simulation and evaluation

5.1. Aruba dataset

To simulate our DCR approach, this paper uses the Aruba open data collected from CASAS smart homes, a project of Washington State University (CASAS Project, 2007) (Cook and Schmitter-Edgecombe, 2009). Aruba dataset records the daily life of an elderly person living alone. Data collected from Aruba dataset was obtained using 31 motion sensors, three door sensors, five temperature sensors and three light sensors. In this paper, we are only interested in motion and door sensors. 11 activities were performed for 220 days (7 months). These data are all represented as a sequence of time-stamped sensor events with annotated activities, as shown in Figure 6.

```
2010-11-04 05:40:46.310862 M003 OFF
2010-11-04 05:40:51.303739 M004 ON Bed_to_Toilet begin
2010-11-04 05:40:52.342105 M005 OFF
2010-11-04 05:40:57.176409 M007 OFF
2010-11-04 05:43:29.711796 M005 ON
2010-11-04 05:43:30.279021 M004 OFF Bed to Toilet end
2010-11-04 05:43:45.7324 M003 ON   Sleeping begin
2010-11-04 05:43:52.044085 M003 OFF
2010-11-04 05:43:53.185335 M002 ON
2010-11-04 05:43:53.253809 M003 ON
```

Figure 6. An example of timestamped sensor events with annotated activities

Table 4. Activities statistics of Aruba dataset

Id	Activity	number of events	Id	Activity	number of events
1	Bed_to_Toilet	844	7	Relax	173 337
2	Eating	10 421	8	Resperate	211
3	Enter_Home	1 097	9	Sleeping	21 247
4	Housekeeping	10 583	10	Wash_Dishes	8 021
5	Leave_Home	1 161	11	Work	9 927
6	Meal_Preparation	149 875	12	Other	493 274

The dataset is imbalanced, as some of the activities occur more frequently than others. Table 4 presents the number of sensor events per activity in the Aruba dataset. *Other* activity contains events with missing labels. It covers 54% of the entire sensor events.

5.2. Preprocessing of Aruba dataset

The purpose of Aruba dataset preprocessing is to prepare the acquired data for performing the DCR approach. As mentioned above, this process involves four main steps: segmentation, feature extraction, adding *location* feature and creating sub datasets per *location* feature. Figure 7 summarizes these steps. In this work, we used data results of *segmentation* and *feature extraction* steps provided by (Yala et al., 2017).

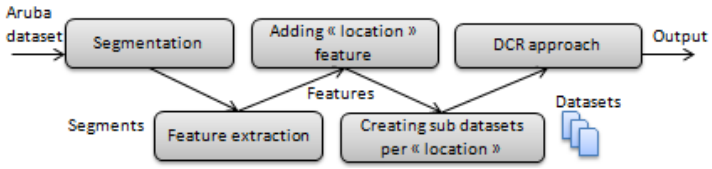


Figure 7. Preprocessing of Aruba dataset for DCR approach

5.2.1. Segmentation step

Segmenting a continuous sensor sequence is usually a prerequisite process for activity recognition. It aims to split the data into segments or windows which

contain information about activities. Retrieving important and useful information from continuous stream of sensor data is a difficult issue for continuous activity (Alzahrani and Kammoun, 2016). In (Yala et al., 2017), authors used the sensor-based windowing technique to deal with streaming sensor data. Each window contains an equal number of events which is fixed to 10 events. The choice of this number is influenced by the average number of sensor events that span the duration of different activities.

5.2.2. Feature extraction step

It aims to extract features as the main characteristics of a raw data segment. In other words, it is the transformation of large input data into a reduced representation of the set of features, which can also be referred as a *feature vector* (Alzahrani and Kammoun, 2016). Once the segmentation technique is performed, each window can be transformed into a feature vector. In (Yala et al., 2017), authors used the *baseline method* as a feature extraction method (Krishnan and Cook, 2014). Thus, from a window, one feature vector FV_i is built with a fix dimension containing the start time of the first event, the last time of the last event, the duration of the window and the occurrence number N $(N_{(D001)}, N_{(D002)}, ..., N_{(M031)}$ where $Dxxx$ are door sensors and $Mxxx$ are motion sensors) of each sensor within the window. Given Aruba dataset, if we have 34 sensors installed in the smart home, the feature vector's dimension FV_i will be 34+3. FV_i is tagged with the label y_i of the last sensor event in the window. Each label y_i corresponds to an activity class (Figure 8).

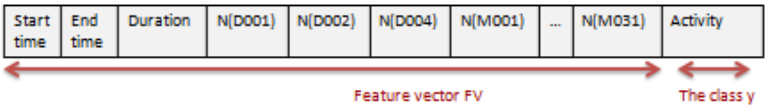


Figure 8. Feature vector and its corresponding class

Thus, a collection of FV_i and the corresponding y_i become the training data on which the classifier of each agent is built.

Table 5. List of sensors for each location

Location	Sensors id	Location	Sensors id
Living	M009,M010,M012,M013,M020	Bedroom 1	M023,M024
Dining	M014	Bedroom 2	M001,M002,M003,M005,M006,M007
Kitchen	M015,M017,M018,M019	Bathroom 1	M031
Office	M025,M026,M027,M028	Bathroom 2	M004
Corridor	M021,M022, M008, M029	Exit	D001,D002, D004, M011, M016, M030

5.2.3. Adding ***location*** *feature*

Our DCR architecture holds a set of agents assigned to different zones or locations in the smart home. Therefore, in order to simulate our approach, we have to determine the location of each feature vector in the map (Figure 9). We distinguish 10 locations (10 zones): living room, kitchen, dining, bedroom 1, bedroom 2, bathroom 1, bathroom 2, exit, corridor and office.

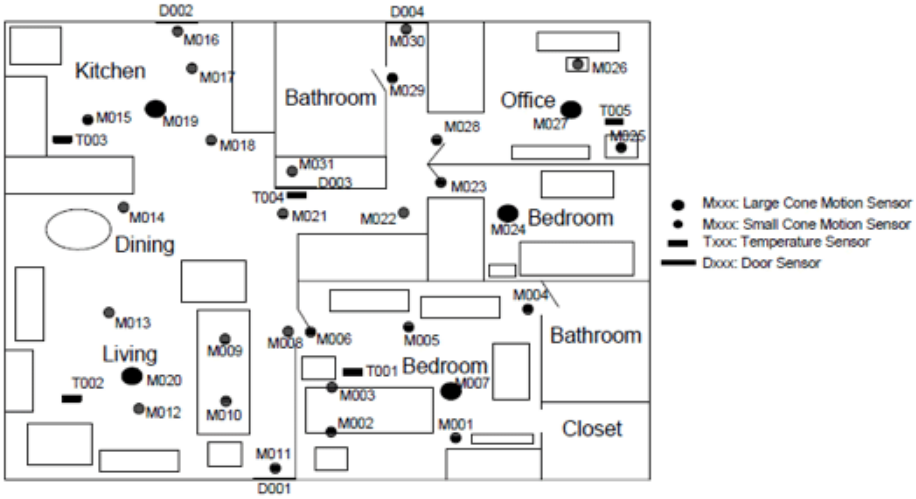


Figure 9. Smart home map of Aruba dataset

Table 5 summarizes the set of sensors installed in each location.

The *location* of each feature vector is determined by the following strategy: FV contains the occurrence number of each triggered sensor within the window.

These triggered sensors belong to different locations in the smart home. The final location of the FV is the location of the triggered sensors having the maximum mean occurrence number. The *location* feature is added to each feature vector (Figure 10).

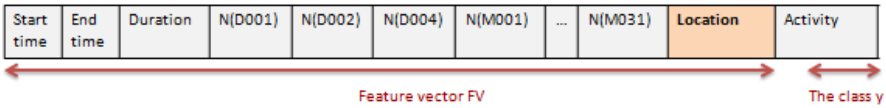


Figure 10. Adding *location* feature

5.2.4. Creating sub datasets per location

Adding location feature in the vector aims to split the Aruba dataset into different sub datasets according to the location feature. Thus, each sub dataset corresponds to a specific location, contains all related feature vectors and then can be assigned to an agent in that location. Feature vectors in sub datasets do not include the *location* feature.

5.3. Simulation environment

In order to test and evaluate our DCR and DCR-OL approaches on Aruba dataset, the simulation environment was employed with two setups:

Hardware setup: tests were performed on a virtual machine Windows Server 2012. This one is an Intel(R) Xeon(R) CPU E5-2673 v3 @2.40 GHz 2.39 GHz (endowed with 8 cores hyper threading enabled), 28.0 GB of RAM and 126 GB of hard disk.

Software setup: we have chosen the Anaconda[1] V3, the leading open source data science platform including Python language. For data analysis, we used the open source python library **Scikit-learn**[2] that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms.

[1] https://www.continuum.io/anaconda-overview
[2] http://scikit-learn.org/

For the multi-agent setup, we used SPADE[3] V2.3 (Smart Python multi-Agent Development Environment) which is a Multi-agent Platform based on the XMPP/Jabber technology and written in Python.

5.4. Initialization of DCR and DCR-OL systems

In this section, we describe the initialization step of the two models DCR and DCR-OL.

5.4.1. Initialization of DCR system

DCR models the smart home (Figure 9) as a multi-agent system $MAS = \{A_l, A_d, A_k, A_o, A_c, A_{bed1}, A_{bed2}, A_{bath1}, A_{bath2}, A_c\}$ where agents are respectively assigned to livingroom, dining, kitchen, office, corridor, bedroom1, bedroom2, bathroom1, bathroom2 and corridor locations. The threshold δ is chosen at 80%. Agents in MAS have to be initialized with their classifiers $clf_{A_{l_i}}$ and their activities lists $ACT_{A_{l_i}}$ as follows.

1)-Building $clf_{A_{l_i}}$: we have to analyze sub datasets by classifying activities with machine learning algorithms (Alpaydin, 2010): Random Forest (RF), Decision Tree (DT), Extra-tree (ExT) and Naive Bayes (NB) (these four classifiers were chosen according to the accuracy and processing time, e.g, SVM takes much time to provide the results). The classification was performed with default parameters and using 10-fold cross-validation. This analysis aims to choose the best classifier for each sub dataset in terms of accuracy and F-measure metrics.

Table 6 denotes the accuracy value for all sub datasets per location by employing the four different classifiers. In fact, we notice that the RF classifier presents the best accuracy and also for the F-measure and G-mean metrics for all sub datasets (due to the lack of space, we did not describe the F-measure and G-mean details). Therefore, all agents will adopt an activity model built upon the RF classifier. It should be noted that the choice of a classifier to build an activity model depends mainly on the sub dataset. Thus, in our case, we

[3]https://pypi.python.org/pypi/SPADE

Table 6. Accuracy of different classifiers for Aruba sub datasets

Dataset	Location	Instance number	RF (%)	DT (%)	ExT (%)	NB (%)
D_l	Living	322530	75.96	72.36	60.3	59.73
D_d	Dining	33789	66.04	61.46	64.72	58.71
D_k	Kitchen	241218	58.96	55.67	56.43	51.38
D_o	Office	30485	66.04	61.54	65.55	65.55
D_c	Corridor	52600	84.46	77.82	83.15	26.66
D_{bed1}	Bedroom 1	36127	66.98	63.31	66.28	42.12
D_{bed2}	Bedroom 2	146002	91.84	89.89	90.99	48.61
D_{bath1}	Bathroom 1	990	81.23	70.01	79.91	64.92
D_{bath2}	Bathroom 2	2667	92.7	90.94	92.44	58.32
D_e	Exit	13582	74.05	68.64	73.06	35.68

have by chance obtained the same type of classifiers for all sub datasets but we can have activity models built upon different type of classifiers in other cases (using other datasets).

2)-Building $ACT_{A_{l_i}}$: each agent incorporates this list which is a set of recognized activities by the classifier and their trust degrees. The latter corresponds to the F1-score measure for each activity. The $ACT_{A_{l_i}}$ list content of each agent is sketched in Table 7.

5.4.2. Initialization of DCR-OL system

The DCR-OL approach is the DCR approach with an online learning over time. Thus, DCR-OL adopts the same representation and the same initialization of data as those of the DCR system. For the performance list $PERF_{LA_i} = \{p_{LA_1}, p_{LA_2}, ..., p_{LA_n}\}$, it is initialized with the value 1. All the p_{LA_i} values are equal to 1. To illustrate the DCR approach and the DCR-OL approach, we present in the following an example of application for each system.

Table 7. List of recognized activities $ACT_{A_{(l_i)}}$ of each agent

Agent A_{l_i}	Dataset D_{l_i}	List $ACT_{A_{(l_i)}}$
A_l	D_l	{(2, 42.19%), (4, 43.64%), (5, 21.62%), (6, 61.34%), (7, 86.21%), (9, 89.03%), (10, 24.14%), (11, 30.0%), (12, 83.92%)}
A_d	D_d	{(2, 73.23%), (4, 31.02%), (5, 44.44%), (6, 51.91%), (7, 75.71%), (9, 0.0%), (10, 28.57%), (11, 0.0%), (12, 84.62%)}
A_k	D_k	{(2, 19.62%), (3, 100.0%), (4, 25.2%), (5, 11.11%), (6, 74.6%), (7, 64.37%), (9, 81.58%), (10, 23.56%), (11, 0.0%), (12, 68.78%)}
A_o	D_o	{(2, 0.0%), (4, 64.29%), (5, 42.86%), (6, 68.93%), (7, 82.84%), (8, 77.86%), (9, 90.91%), (10, 64.52%), (11, 69.71%), (12, 85.38%)}
A_c	D_c	{(2, 37.93%), (3, 23.53%), (4, 33.95%), (5, 48.97%), (6, 59.0%), (7, 63.13%), (8, 0.0%), (9, 83.93%), (10, 22.86%), (11, 5.56%), (12, 94.62%)}
A_{bed1}	D_{bed1}	{(2, 66.67%), (4, 39.51%), (5, 54.55%), (6, 70.44%), (7, 69.77%), (9, 70.09%), (10, 50.0%), (11, 47.2%), (12, 84.63%)}
A_{bed2}	D_{bed2}	{(1, 51.58%), (2, 0.0%), (4, 31.2%), (6, 61.08%), (7, 52.0%), (9, 89.84%), (10, 44.44%) (11, 46.15%), (12, 96.87%)}
A_{bath1}	D_{bath1}	{(4, 56.67%), (12, 22.22%), (6, 93.71%)}
A_{bath2}	D_{bath2}	{(1, 57.78%), (4, 56.25%), (6, 0.0%), (7, 66.67%), (9, 0.0%), (12, 97.5%)}
A_e	D_e	{(2, 50.0%), (3, 61.13%), (4, 55.56%), (5, 21.62%), (6, 80.1%), (7, 75.0%), (9, 98.25%), (10, 30.77%), (11, 75.0%), (12, 89.16%)}

5.5. Application examples of DCR and DCR-OL systems

5.5.1. Application example of DCR system

In this subsection, we aim to simulate an example of DCR approach. Given the office agent A_o as an example (Figure 11), it incorporates its sub dataset D_o. **First step**: it takes as input the first feature vector from D_o, interrogates its classifier clf_{A_o} and gets its predicted activity pa_o (e.g. pa_o is identified by id 5 which corresponds to the *leave_home* activity). Then, it determines its trust degree d_{pa_o} from ACT_{A_o} (see Table 7) which is 42.86%. This one is less than 80% which means that the agent A_o is not able to recognize accurately the *leave_home* activity and has to collaborate with some agents who are capable

to recognize it with a higher precision. **Second step**: it starts finding its acquaintance agents and determines its ACQ_{A_o} list that includes agents who can recognize pa_o. In our case, ACQ_{A_o} list contains agents A_l, A_d, A_k, A_c, A_{bed1} and A_e. **Third step**: the agent A_o sends its input to some selected agents in ACQ_{A_o} . Selected agents are agents who recognized pa_o better than A_o which means that their trust degree for pa_o is higher than d_{pa_o} (42.86%). Selected agents are A_d, A_{bed1} and A_c (see Table 7). **Fourth step**: the agent A_o receives foreign predictions and aggregates them with its local prediction pa_o by applying the *max-trust.* aggregation method as an example. The output is a set of predicted activities built upon a collaboration mechanism. Thus, the agent A_o generates two activities: the *leave_home* activity (id 5) with a trust degree equal to 47.28% and the *other* activity (id 12) with a trust degree equal to 94.62%. As a final result, the agent A_o chooses the most confident activity which is the *other* activity (id 12). This process is applied for all feature vectors existing in the sub dataset D_o.

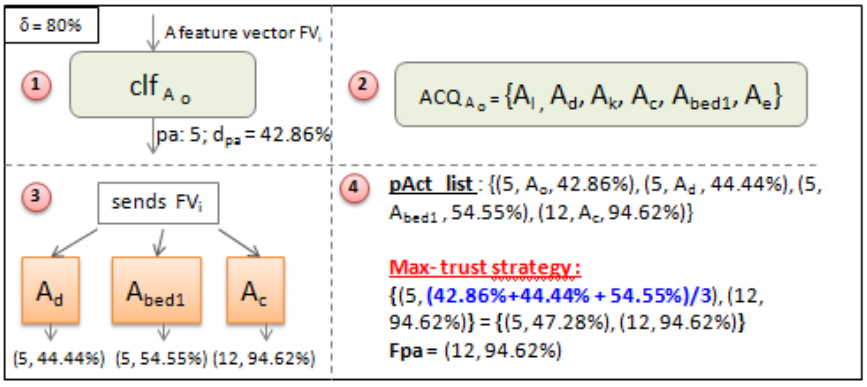


Figure 11. An example of DCR applied to the office agent A_o

5.5.2. Application example of DCR-OL system

We keep the same example of the office agent A_o previously described (i,e. the same example of data). Thus, we keep the same process of 1, 2 and 3 steps (Figure 12).

Fourth step: we apply the DCR-OL approach by using learning agents. These agents learn over time by considering feedbacks from others. Specifically, they apply the maximum weighted performance strategy to resolve conflicts when the local prediction is different from those received.

We assumed that at all agents in MAS are performant in the initialization step and therefore the $PERF_{A_o}$ list of the agent A_o is initialized with the value 1 (Figure 12). The agent A_o receives foreign predictions from selected receiver agents and inserts them in the $pAct\_list_{A_o}$ list including its local prediction pa_o. It proceeds by computing the weighted performance value for each activity in $pAct\_list_{A_o}(PP(id5)$ and $PP(id12))$. With the *max-wPerf* strategy, the agent A_o selects the activity with the maximum weighted performance value which is the activity id5 with a performance value of 75%. Thus, the final predicted activity Fpa_o is the activity *leave_home* (id5).

Fifth step: the agent A_o starts with the learning phase of the $PERF_{A_o}$ list. Indeed, it compares the final predicted activity Fpa_o with the real activity RP_{FV} (Figure 12). In our case, the comparison test fails and then the $PERF_{A_o}$ list must be modified.

Sixth step: this step aims to update of the $PERF_{A_o}$ list. We apply the *Perf-measure* measure for each agent in the $pAct\_list_{A_o}$ list. As a result, the performance values of agents that misclassify the real activity RP_{FV} decrease. These agents are A_o, A_d and A_{bed1} whose performance values p_{A_o}, p_{A_d} and $p_{A_{bed1}}$ are respectively 0.999. The performance value of the agent that correctly predicted RP_{FV} remains the same. This agent is the agent A_c whose performance value p_{A_c} is respectively 1. The $PERF_{A_o}$ list is updated with the new performance values of concerned agents which are A_o, A_d and A_{bed1}. On the next arrival of a feature vector, the *max-wPerf* strategy will considerate the updated $PERF_{A_o}$ list.

In the next section, we start with the experimental evaluation of the DCR system on Aruba dataset.

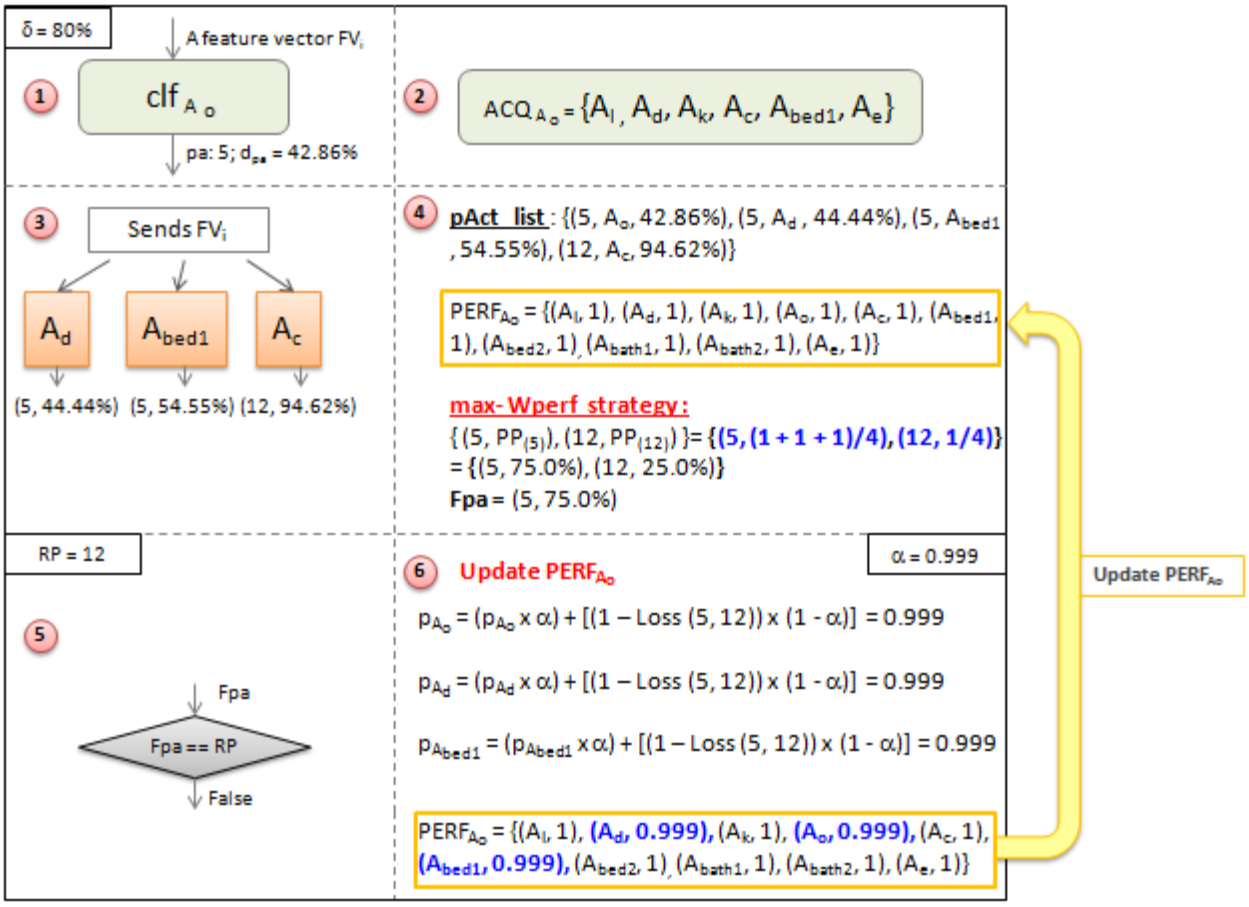


Figure 12. An example of DCR-OL applied to the office agent A_o

5.6. Experimental evaluation of the DCR system

After *MAS* initialization step, the DCR approach can be launched. In fact, each starter agent has its sub dataset and performs the DCR approach with 10 folds cross-validation. Therefore, each sub dataset is split in 10 folds. Each fold represents the testing dataset and the merge of the others represents the training dataset. Thus, each starter agent has 10 folds of training and testing data respectively and then has 10 RF classifiers. A starter agent takes as input each feature vector from a testing dataset, launches the DCR program and generates as output the set of predicted activities with their trust degree. This

task is repeated 10 times for all testing datasets and for each starter agent. To evaluate the performance of our DCR approach, evaluation metrics (Ting, 2010) used in this paper are: *accuracy* – shows the percentage of correctly classified instances; *F-measure* – is the harmonic mean of precision and recall (Ting, 2010); *G-mean* – is the geometric mean measure that tries to maximize the accuracy on each of the classes while keeping these accuracies balanced (Kubat and Matwin, 1997); *processing time* – the time taken to run a program and *message number* – the number of messages exchanged between agents.

5.6.1. Evaluation with accuracy, F-measure and G-mean metrics

Given as an example the office agent A_o, Table 8 denotes in details accuracy, F-measure and G-mean metrics with 10 folds for DCR by applying the three aggregation strategies. According to this comparative table, DCR (stack.) is the best aggregation method in terms of the global average of accuracy and F-measure metrics. DCR (max-freq.) is the best aggregation method in terms G-mean metric. The evaluation details of the 10 folds of the remaining agents are not mentioned in this paper due to the lack of space. We only show the global average of accuracy, F-measure and G-mean metrics of each agent's sub dataset in the whole MAS (Table 9). DCR is evaluated according to the following comparisons:

– **DCR vs central.**: highlights the contribution of the *DCR* approach compared to a centralized approach. This one consists of building the activity model (RF classifier) upon the whole Aruba dataset. According to Table 9, *DCR* outperforms the centralized approach in terms of accuracy, F-measure and G-mean metrics, especially with DCR (max-freq. and stack.).

– **DCR (max-trust) vs DCR (max-freq.) vs DCR (stack.) vs DCR (UB)**: highlights the choice of the aggregation method. Table 9 shows that DCR (stack.) is the best aggregation method compared to both max-trust and max-freq. in terms of accuracy values. However, DCR (max-freq.) is the best aggregation method compared to both max-trust and stack. in terms of F-measure and G-mean values. DCR (UB) represents the maximum values

computed (upper bound) of metrics that an aggregation method can achieve. Therefore, a new aggregation method can be implemented to get more closer to DCR (UB).

690 − **DCR vs W-DCR**: highlights the collaborative aspect of DCR compared to W-DCR. This latter is a degraded version of DCR without considering agent collaboration. Especially, each agent interrogates its classifier with its FVs and directly generates as an output one activity. Table 9 denotes that the accuracy is slightly improved with DCR (max-freq.) compared to W-DCR. However, DCR 695 (stack.) outperforms W-DCR up to 2.3%. This is thanks to the collaboration between the starter agents and their selected agents that consolidate the recognition rate of the correct activity. The F-measure and G-mean metrics values decrease for DCR (max-trust, max-freq. and stack.) compared to W-DCR.

− **DCR vs CASE system (Cicirelli et al., 2016)**: highlights a comparison 700 of DCR with CASE, a related work discussed above. The obtained metrics of DCR outperform the ones obtained by CASE (Table 9).

5.6.2. Evaluation of processing time

Figure 13 denotes the average processing time in minutes of different sub datasets for each agent when launching the DCR program (without considering 705 the strategies max-trust. and max-freq.). The kitchen agent A_k presents the highest processing time that exceeds 300min and achieves 22 hours. This is due to the huge collaboration with other agents when predicting locally the *other* activity (id 12) and the *meal_preparation* activity (id 6). These activities present respectively 41% and 53% of activities in the dataset and their trust 710 degrees are less than 80%. The other agents present a processing time that does not exceed 3 hours. This is due to the *other* activity that presents more than 50% of activities in the dataset and has a trust degree more than 80%. Thus, the collaboration with other agents is not required.

Table 8. Agent A_o: evaluation of *office* dataset D_o

D_o		DCR		
		max-trust	max-freq.	stack.
1-fold	Acc.	53.02%	59.25%	59.48%
	F-meas.	36.94%	53.02%	53.18%
	G-mean	49.89%	57.62%	53.18%
2-fold	Acc.	26.54%	57.55%	67.06%
	F-meas.	11.38%	57.54%	54.20%
	G-mean	44.14%	69.89%	46.54%
3-fold	Acc.	51.35%	66.7%	67.58%
	F-meas.	36.02%	62.53%	62.63%
	G-mean	49.69%	67.37%	67.39%
4-fold	Acc.	76.67%	75.3%	79.46%
	F-meas.	69.90%	75.95%	70.57%
	G-mean	45.16%	64.34%	40.36%
5-fold	Acc.	97.47%	61.15%	98.13%
	F-meas.	96.87%	74.29%	97.20%
	G-mean	13.48%	50.13%	13.54%
6-fold	Acc.	90.06%	62.07%	90.26%
	F-meas.	86.02%	69.95%	85.82%
	G-mean	32.91%	67.40%	29.35%
7-fold	Acc.	62.43%	54.59%	64.11%
	F-meas.	49.81%	52.18%	50.36%
	G-mean	47.75%	55.3%	48.15%
8-fold	Acc.	65.65%	68.64%	70.05%
	F-meas.	54.02%	66.22%	66.35%
	G-mean	46.76%	59.06%	59.12%
9-fold	Acc.	50.62%	61.91%	62.98%
	F-meas.	35.02%	57.55%	58.12%
	G-mean	49.66%	63.75%	64.19%
10-fold	Acc.	53.0%	64.79%	70.28%
	F-meas.	40.87%	65.07%	68.10%
	G-mean	47.62%	64.57%	66.94%
Global	Acc.	62.68%	63.19%	72.93%
	F-meas.	54.38%	63.43%	66.65%
	G-mean	42.70%	61.94%	48.87%

5.6.3. Evaluation of message number

Agents exchange messages when collaborating together. Messages can be request or answer messages. It should be noted that the message exchange starts when the local predicted activity has a trust degree less than the threshold. Figure 14 describes the average of the message number exchanged between the starter agent and the other agents. The kitchen agent A_k presents the highest message number that exceeds 50000 messages and achieves 652726 messages. This result is meaningful since its processing time is high.

Table 9. DCR vs other approaches

Agents		Central.	CASE System	W-DCR	DCR			
					max-trust	max-freq.	stack.	UB
A_l	Acc.	-	63.52%	76.04%	76.38%	76.4%	76.36%	76.97%
	F-meas.	-	63.46%	75.43%	75.38%	75.61%	75.1%	76.37%
	G-mean	-	66.77%	78.07%	69.9%	77.99%	77.74%	78.38%
A_d	Acc.	-	51.54%	65.81%	67.25%	67.44%	68.01%	70.60%
	F-meas.	-	51.32%	64.24%	62.62%	64.13%	61.9%	68.25%
	G-mean	-	55.27%	65.57%	62.67%	64.23%	61.76%	68.36%
A_k	Acc.	-	52.17%	58.81%	50.41%	51.45%	61.31%	83.17%
	F-meas.	-	52.5%	58.97%	49.77%	50.88%	52.24%	83.16%
	G-mean	-	56.55%	63.14%	54.45%	57.56%	52.89%	85.97%
A_o	Acc.	-	53.51%	62.76%	62.68%	63.19%	72.93%	76.52%
	F-meas.	-	54.61%	63.16%	54.38%	63.43%	66.65%	74.08%
	G-mean	-	52.27%	68.21%	42.7%	61.94%	48.87%	69.51%
A_c	Acc.	-	75.67%	84.13%	85.21%	85.24%	85.65%	86.89%
	F-meas.	-	75.29%	80.76%	79.71%	80.3%	79.48%	82.67%
	G-mean	-	45.70%	46.97%	37.78%	41.03%	35.7%	48.13%
A_{bed1}	Acc.	-	49.82	65.81%	65.16%	67.01%	68.40%	77.97%
	F-meas.	-	51.22%	65.75%	58.37%	64.7%	60.48%	76.04%
	G-mean	-	52.40%	64.11%	53.04%	61.27%	51.55%	71.24%
A_{bed2}	Acc.	-	85.72%	91.99%	92.57%	92.52%	92.48%	92.77%
	F-meas.	-	85.51%	91.18%	91.25%	91.35%	91.0%	91.60%
	G-mean	-	74.23%	83.56%	82.9%	83.41%	83.02%	83.63%
A_{bath1}	Acc.	-	78.18%	80.10%	77.78%	83.53%	81.78%	86.36%
	F-meas.	-	76.79%	76.48%	75.69%	78.36%	78.56%	83.03%
	G-mean	-	39.42%	33.46%	42.36%	33.68%	39.9%	53.05%
A_{bath2}	Acc.	-	90.92%	92.34%	93.39%	93.99%	93.69%	94.07%
	F-meas.	-	91.02%	91.71%	92.31%	92.55%	91.38%	92.66%
	G-mean	-	46.58%	56.15%	55.33%	55.41%	35.45%	55.52%
A_e	Acc.	-	57.4%	73.86%	73.17%	74.24%	74.26%	76.97%
	F-meas.	-	57.25%	72.34%	68.52%	72.35%	70.37%	74.98%
	G-mean	-	56.12%	70.27%	66.07%	70.16%	66.5%	72.00%
Global	Acc.	72.94%	65.84	75.16%	74.4%	75.5%	77.48%	82.23%
	F-meas.	71.80%	65.89%	74.0%	70.8%	73.36%	72.71%	80.28%
	G-mean	55.27%	54.53%	62.95%	56.72%	60.66%	55.33%	68.57%

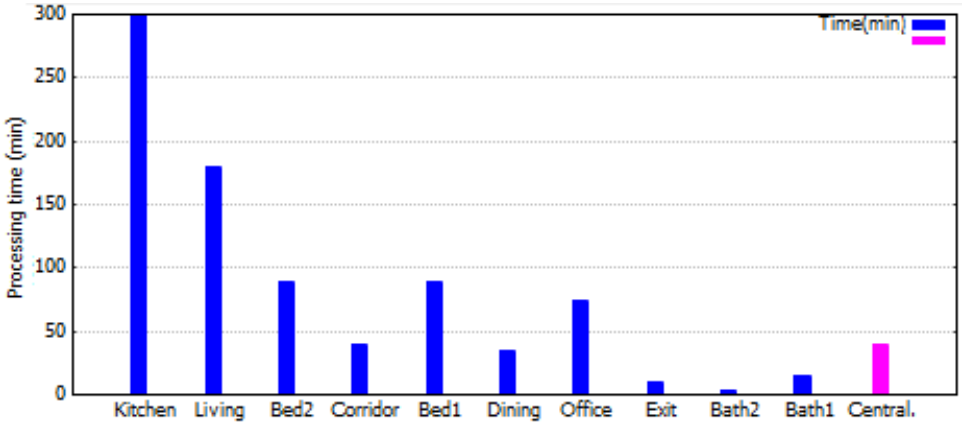


Figure 13. Processing time of DCR's agents and the centralized approach

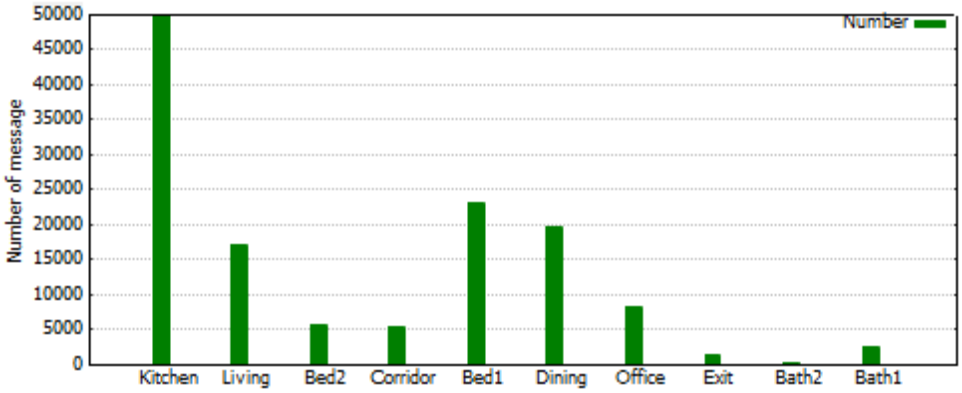


Figure 14. Message number exchanged between each starter agent and others

5.7. Experimental evaluation of the DCR-OL system

The simulation of the DCR-OL approach is performed with 10 folds cross-validation and it is the same as well as the DCR approach. A starter agent SL_i takes as an input each feature vector from the testing dataset, launches the DCR-OL system and generates as an output the final predicted activity with its weighted performance value (this value is considered as a trust degree). This task is repeated 10 times for all testing datasets and for each starter agent in MAS.

5.7.1. Evaluation with accuracy, F-measure and G-mean metrics

Considering the same example of the office agent, Table 10 adds a column for the DCR-OL approach and especially the new *max-wPerf* conflict resolution strategy. The aim is to evaluate this new online strategy against the other three conflict resolution strategies of the DCR approach. The evaluation details of the 10 folds of the remaining agents are not mentioned in this paper due to the lack of space.

We deduce that the *max-wPerf* method improves the accuracy compared to the two methods *max-freq* and *max-trust*. However, the *stacking* method remains the best method in terms of accuracy. Considering the F-measure and G-mean metrics, the *max-wPerf* method exceeds the *max-trust* method but it is less performant than the two other methods: *max-freq* and *stacking*.

Given the MAS system, Table 11 summarizes the global average of metrics (accuracy, F-measure and G-mean) of all agents. We highlight *DCR-OL*'s contribution to *DCR* as follows:

– **DCR-OL (max-wPerf) vs DCR (max-trust) vs DCR (max-freq.) vs DCR (stack.) vs DCR (UB)**: the *max-wPerf* method improves the accuracy compared to the two *max-freq* and *max-trust* methods. However, the *stacking* method is still the best method in terms of accuracy. For the F-measure and G-mean metrics, the *max-wPerf* method exceeds the *max-trust* and the *stacking* methods but it is less performant than the *max-freq* method.

To summarize, the *stacking* method is the best method of conflict resolution followed by the *max-wPerf* online method in terms of accuracy metric and the *max-freq.* method is the best conflict resolution method followed by the *max-wPerf* online method in terms of F-measure and G-mean metrics.

5.7.2. Accuracy evolution over time

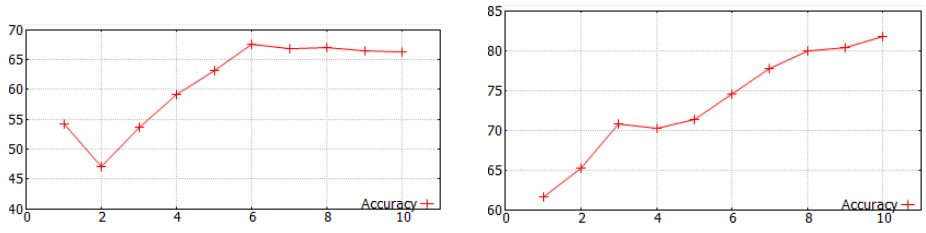
This subsection analyzes the evolution of the accuracy of different agents over time. In fact, this study highlights the impact of the evolution of each agent's performance measure on the accuracy over time.

Table 10. Agent LA_o: evaluation of *office* dataset D_o with *DCR-OL*

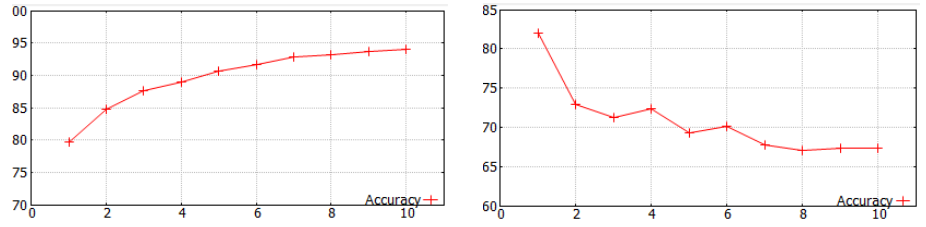
D_o		DCR			DCR-OL
		max-trust	max-freq.	stacking	max-wPerf
1-fold	Acc.	53.02%	59.25%	59.48%	54.27%
	F-meas.	36.94%	53.02%	53.18%	39.72%
	G-mean	49.89%	57.62%	53.18%	51.24%
2-fold	Acc.	26.54%	57.55%	57.58%	37.57%
	F-meas.	11.38%	57.54%	54.20%	33.26%
	G-mean	44.14%	69.89%	46.54%	52.69%
3-fold	Acc.	51.35%	66.7%	67.58%	57.28%
	F-meas.	36.02%	62.53%	62.63%	48.75%
	G-mean	49.69%	67.37%	67.39%	56.31%
4-fold	Acc.	76.67%	75.3%	79.46%	77.0%
	F-meas.	69.90%	75.95%	70.57%	70.28%
	G-mean	45.16%	64.34%	40.36%	45.7%
5-fold	Acc.	97.47%	61.15%	98.13%	97.57%
	F-meas.	96.87%	74.29%	97.20%	96.92 %
	G-mean	13.48%	50.13%	13.54%	13.49%
6-fold	Acc.	90.06%	62.07%	90.26%	89.86%
	F-meas.	86.02%	69.95%	85.82%	85.72%
	G-mean	32.91%	67.40%	29.35%	31.13%
7-fold	Acc.	62.43%	54.59%	64.11%	62.57%
	F-meas.	49.81%	52.18%	50.36%	50.08%
	G-mean	47.75%	55.3%	48.15%	47.91%
8-fold	Acc.	65.65%	68.64%	70.05%	65.94%
	F-meas.	54.02%	66.22%	66.35%	54.66%
	G-mean	46.76%	59.06%	59.12%	47.3%
9-fold	Acc.	50.62%	61.91%	62.98%	55.48%
	F-meas.	35.02%	57.55%	58.12%	44.70%
	G-mean	49.66%	63.75%	64.19%	54.73%
10-fold	Acc.	53.0%	64.79%	70.28%	67.18%
	F-meas.	40.87%	65.07%	68.10%	67.15%
	G-mean	47.62%	64.57%	66.94%	66.34%
Global	Acc.	62.68%	63.19%	72.93%	66.51%
	F-meas.	54.38%	63.43%	66.65%	59.12%
	G-mean	42.7%	61.94%	48.87%	46.68%

Table 11. DCR vs DCR-OL

Agents		DCR			DCR-OL	upper bound
		max-trust	max-freq.	stacking	max-wPerf	
LA_l	Acc.	76.38%	76.4%	76.36%	76.51%	76.97%
	F-meas.	75.38%	75.61%	75.1%	75.42%	76.37%
	G-mean	69.9%	77.99%	77.74%	77.96%	78.38%
LA_d	Acc.	67.25%	67.44%	68.01%	67.99%	70.60%
	F-meas.	62.62%	64.13%	61.9%	64.21%	68.25%
	G-mean	62.67%	64.23%	61.76%	64.01%	68.36%
LA_k	Acc.	50.41%	51.45%	61.31%	58.82%	83.17%
	F-meas.	49.77%	50.88%	52.24%	57.22%	83.16%
	G-mean	54.45%	57.56%	52.89%	58.9%	85.97%
LA_o	Acc.	62.68%	63.19%	72.93%	66.51%	76.52%
	F-meas.	54.38%	63.43%	66.65%	59.12%	74.08%
	G-mean	42.7%	61.94%	48.87%	46.68%	69.51%
LA_c	Acc.	85.21%	85.24%	85.65%	85.46%	86.89%
	F-meas.	79.71%	80.3%	79.48%	80.29%	82.67%
	G-mean	37.78%	41.03%	35.7%	39.63%	48.13%
LA_{bed1}	Acc.	65.16%	67.01%	68.40%	66.96%	77.97%
	F-meas.	58.37%	64.7%	60.48%	61.98%	76.04%
	G-mean	53.04%	61.27%	51.55%	56.97%	71.24%
LA_{bed2}	Acc.	92.57%	92.52%	92.48%	92.6%	92.77%
	F-meas.	91.25%	91.35%	91.0%	91.3%	91.60%
	G-mean	82.9%	83.41%	83.02%	82.97%	83.63%
LA_{bath1}	Acc.	77.78%	83.53%	81.78%	84.43%	86.36%
	F-meas.	75.69%	78.36%	78.56%	79.36%	83.03%
	G-mean	42.36%	33.68%	39.9%	36.69%	53.05%
LA_{bath2}	Acc.	93.39%	93.99%	93.69%	93.99%	94.07%
	F-meas.	92.31%	92.55%	91.38%	92.55%	92.66%
	G-mean	55.33%	55.41%	35.45%	55.41%	55.52%
LA_e	Acc.	73.17%	74.24%	74.26%	73.31%	76.97%
	F-meas.	68.52%	72.35%	70.37%	68.71%	74.98%
	G-mean	66.07%	70.16%	66.5%	66.29%	72.00%
Global	Acc.	74.4%	75.5%	77.48%	76.64%	82.23%
	F-meas.	70.8%	73.36%	72.71%	73.01%	80.28%
	G-mean	56.72%	60.66%	55.33%	58.55%	68.57%



(a) Agent LA_o (b) Agent LA_{bath1}



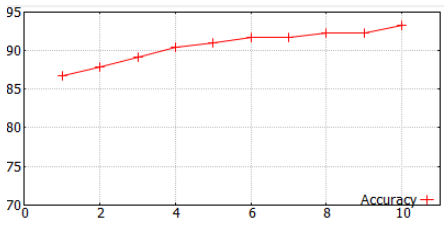
(c) Agent LA_{bath2} (d) Agent LA_{bed1}

Figure 15. Evolution of accuracy over time for agents LA_o, LA_{bath1}, LA_{bath2} and LA_{bed1}

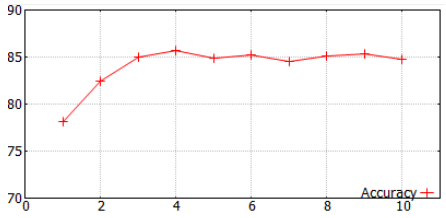
In order to achieve this, the dataset of each agent is divided into 10 sub-datasets. We start with the first sub-dataset received over time by launching the DCR-OL system and especially by applying the *max-wPerf* method. Then, we measure the accuracy. Next, we add the second sub-dataset received over time by keeping the ascending chronological order. We launch the DCR-OL system, apply the *max-wPerf* method and measure the accuracy. Next, we add the third sub-dataset received over time by keeping the ascending chronological order and so on.

Figures 15 and 16 denote the evolution of the accuracy by merging sub-datasets one by one each time in the ascending chronological order. For example, the x-axis with the value 3 means the merge of the three sub-datasets.

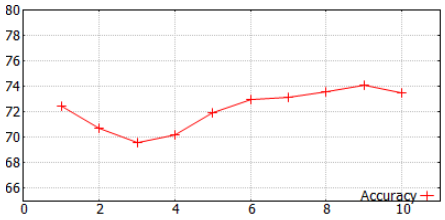
Discussion: the more agents learn, adapt and evolve their performance values, more they correctly recognize the activities and thus the accuracy improves. There is only the LA_{bed1} agent whose accuracy is decreased over time.



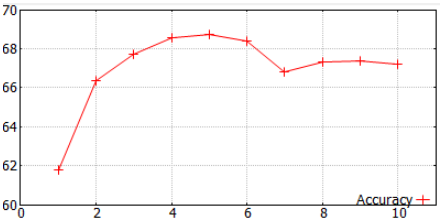
(a) Agent LA_{bed2}



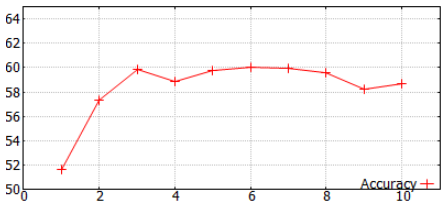
(b) Agent LA_c



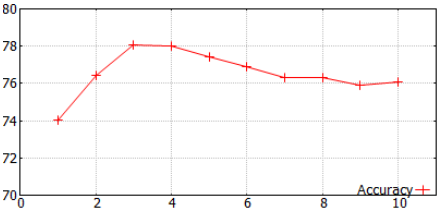
(c) Agent LA_e



(d) Agent LA_d



(e) Agent LA_k



(f) Agent LA_l

Figure 16. Evolution of accuracy over time for agents LA_{bed2}, LA_c, LA_e, LA_d, LA_k and LA_l

6. Conclusion and future works

This paper proposes DCR as a novel Distributed Collaborative Reasoning approach to recognize human activities from a continuous sensor sequence in smart homes. DCR consists of several agents, with diverse classifiers based on reasoning mechanism, that are able to analyze data coming from deployed sensors and collaborate together in order to identify activities. After segmentation and feature extraction steps of the continuous sensor sequence, agents analyze feature vectors according to their location in the smart home, predict a local activity and collaborate with each other in order to achieve the correct activity based on three strategies of decision making (max-trust, max-freq. and stack.).

Our DCR approach achieves some requirements as follows: **C1**→**P1**- the whole distributed HAR system (Figure 1) deals with continuous sequence of sensor data through three modules namely *Perception*, *Observation* and *Identification*, from raw sensor data to identifying activities. The *Identification* task is performed through DCR which is a totally distributed architecture including heterogeneous, autonomous and interacting agents. **C2**→**P2**- our approach adopts a bottom up strategy (from raw data, event, segments, feature vectors to activities) which guarantees the data freshness; **C3**→**P3**- agents incorporate classifiers built upon past history; **C4**→**P4**- we defined a collaboration strategy that consists of selecting agents who identify the activity with a more valuable accuracy than the starter agent; **C5**→**P5**- agents hold different types of classifiers built upon different training data regarding their locations in the smart home; **C6**→**P6**- agents assign a trust degree to each recognized activity which help the starter agent to make a decision; **C7**→**P7**- the *stacking* method is the best method of conflict resolution in terms of accuracy metric and the *max-freq.* method is the best conflict resolution method in terms of F-measure and G-mean metrics. In addition to these achievements, we note that our approach is generalizable and can be extended to other datasets for HAR that contain a set of events with annotated activities.

Moreover, a new version of DCR with an online learning, which is the DCR-

OL approach is proposed. This approach satisfies the challenge **C8** by using learning agents. These latters can learn from their collaborations, act and adapt their performance over time in order to improve the recognition accuracy.

In our ongoing work, for the DCR approach, the number of messages exchanged between agents and the processing time are important. Therefore, we need to define a new communication strategy in order to reduce them. Considering the DCR-OL approach, we plan to evolve the trust degrees defined in the ACT_{LA_i} list over time. Furthermore, we can use the performance values of agents as weighting factors for the two aggregation methods: *max-trust* method (the weighting factor of the concerned agent will be applied to its trust degree) and *max-freq* method (the weighting factor of the concerned agent will be applied to its frequency). We are currently working in this direction to ensure the model adaptability and evolution.

Acknowledgment

We would like to thank authors in (Yala et al., 2017) for providing us their results of segmentation and feature extraction steps related to the Aruba dataset.

References

Abdelhedi, S., Wali, A., and Alimi, A. M. (2016). *Fuzzy Logic Based Human Activity Recognition in Video Surveillance Applications*, pages 227–235. Springer International Publishing, Cham.

Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2nd edition.

Alzahrani, M. and Kammoun, S. (2016). *Human Activity Recognition: Challenges and Process Stages*.

Amft, O. and Lombriser, C. (2011). Modelling of distributed activity recognition in the home environment. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 1781–1784.

Barber, K. S., Liu, T. H., and Han, D. C. (2000). *Strategic Decision-Making for Conflict Resolution in Dynamic Organized Multi-Agent Systems*.

Canzian, L., Zhang, Y., and van der Schaar, M. (2015). Ensemble of distributed learners for online classification of dynamic data streams. *IEEE Transactions on Signal and Information Processing over Networks*, 1(3):180–194.

Chen, L., Nugent, C. D., and Wang, H. (2012). A knowledge-driven approach to activity recognition in smart homes. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):961–974.

Cicirelli, F., Fortino, G., Giordano, A., Guerrieri, A., Spezzano, G., and Vinci, A. (2016). On the design of smart homes: A framework for activity recognition in home environment. *Journal of Medical Systems*, 40(9):200.

Cook, D. J. and Schmitter-Edgecombe, M. (2009). Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(5):480.

Fortino, G., Giordano, A., Guerrieri, A., Spezzano, G., and Vinci, A. (2015). *A Data Analytics Schema for Activity Recognition in Smart Home Environments*, pages 91–102. Springer International Publishing, Cham.

Gromping, U. (2016). Practical guide to logistic regression. 71.

Jarraya, A., Bouzeghoub, A., Borgi, A., and Arour, K. (2018). Distributed collaborative reasoning for HAR in smart homes. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 1971–1973.

Krishnan, N. C. and Cook, D. J. (2014). Activity recognition on streaming sensor data. *Pervasive and Mobile Computing*, 10(Part B):138 – 154.

Kubat, M. and Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann.

Luckham, D. (2008). The power of events: An introduction to complex event processing in distributed enterprise systems. In Bassiliades, N., Governatori, G., and Paschke, A., editors, *Rule Representation, Interchange and Reasoning on the Web*, pages 3–3, Berlin, Heidelberg. Springer Berlin Heidelberg.

Maciel, C., de Souza, P. C., Viterbo, J., Mendes, F. F., and El Fallah Seghrouchni, A. (2015). A multi-agent architecture to support ubiquitous applications in smart environments. In *Agent Technology for Intelligent Mobile Services and Smart Societies*, pages 106–116, Berlin, Heidelberg. Springer Berlin Heidelberg.

Marin-Perianu, M., Lombriser, C., Amft, O., Havinga, P., and Tröster, G. (2008). *Distributed Activity Recognition with Fuzzy-Enabled Wireless Sensor Networks*, pages 296–313. Springer Berlin Heidelberg, Berlin, Heidelberg.

McKeever, S., Ye, J., Coyle, L., Bleakley, C. J., and Dobson, S. (2010). Activity recognition using temporal evidence theory. *JAISE*, 2(3):253–269.

Mosabbeb, E. A., Raahemifar, K., and Fathy, M. (2013). Multi-view support vector machines for distributed activity recognition. In *2013 Seventh International Conference on Distributed Smart Cameras (ICDSC)*, pages 1–2.

Noor, M. H. M., Salcic, Z., and Wang, K. I.-K. (2016). Enhancing ontological reasoning with uncertainty handling for activity recognition. *Knowledge-Based Systems*, 114(Supplement C):47 – 60.

Ramakrishnan, A. K., Preuveneers, D., and Berbers, Y. (2013). A modular and distributed bayesian framework for activity recognition in dynamic smart environments. In *Ambient Intelligence - 4th International Joint Conference, AmI 2013, Dublin, Ireland, December 3-5, 2013. Proceedings*, pages 293–298.

Ramakrishnan, A. K., Preuveneers, D., and Berbers, Y. (2014). A bayesian framework for life-long learning in context-aware mobile applications. In *Context in Computing - A Cross-Disciplinary Approach for Modeling the Real World*, pages 127–141.

Ranasinghe, S., Machot, F. A., and 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).

Riboni, D., Sztyler, T., Civitarese, G., and Stuckenschmidt, H. (2016). Unsupervised recognition of interleaved activities of daily living through ontological and probabilistic reasoning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 1–12, New York, NY, USA. ACM.

Rodriguez, N. D., Cuellar, M. P., Lilius, J., and Calvo-Flores, M. D. (2014). A fuzzy ontology for semantic modelling and recognition of human behaviour. *Knowledge-Based Systems*, 66(Supplement C):46 – 60.

Sebbak, F., Benhammadi, F., Chibani, A., Amirat, Y., and Mokhtari, A. (2014). Dempster–shafer theory-based human activity recognition in smart home environments. *annals of telecommunications - annales des télécommunications*, 69(3):171–184.

Ting, K. M. (2010). *Precision and Recall*, pages 781–781. Springer US, Boston, MA.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2018). The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–176.

Wang, X. and Ji, Q. (2014). Context augmented dynamic bayesian networks for event recognition. *Pattern Recognition Letters*, 43(Supplement C):62 – 70. ICPR2012 Awarded Papers.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241 – 259.

Yala, N., Fergani, B., and Fleury, A. (2017). Towards improving feature extraction and classification for activity recognition on streaming data. *Journal of Ambient Intelligence and Humanized Computing*, 8(2):177–189.

Ziaeefard, M. and Bergevin, R. (2015). Semantic human activity recognition: A literature review. *Pattern Recognition*, 48(8):2329 – 2345.

Zliobaite, I. (2010). Learning under concept drift: an overview. *CoRR*, abs/1010.4784.