UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Informatica

SEMI-SUPERVISED LEARNING FOR MULTI-INHABITANT ACTIVITY RECOGNITION

Relatore: Prof. Claudio BETTINI

Correlatore: Dott. Gabriele CIVITARESE

Tesi di Laurea di:
Luca ARROTTA
Matricola: 914098

Anno Accademico 2019/2020

Contents

Introduction

In the last few years, technological progress has led to the growth of low-cost sensors having computing and communication capabilities. Sensors of this kind installed in a home environment can interact with each other creating what is defined as smart-home. A smart-home is a habitation equipped with a network of sensors able to collect data related to inhabitants and to the environment which surrounds them. Inside this network, it is possible to find environmental (magnetic, pressure and temperature sensors) and wearable sensors (inertial sensors installed on devices like smartwatches). Furthermore, the spread of smartphones and the research in the ubiquitous sensing field, whose purpose is to extract knowledge from sensors data, led to the development of pervasive systems and context-aware applications. The purpose of the latter is to adapt their behavior based on the context that surrounds the user (e.g., the weather, the activity he is performing, the time of day). Among the context-aware applications, there are Human Activity Recognition (HAR) systems, whose purpose is to detect human activities through the analysis of videos containing human motions or data coming from environmental and wearable sensors. HAR applications can be used in a variety of areas, including the surveillance-based security and the healthcare fields. HAR systems in the healthcare area, for example, can be built to monitor and assist the residents of a smart-home. Inside such a habitation, the interactions of the users with environmental sensors are detected and exploited to recognize the actions the inhabitants are performing. If the smart-home is inhabited by more than one person, we talk about multi-inhabitant HAR systems. In such a field, the system has to be able to detect both the activities performed individually by the users and the concurrent ones. In a multi-inhabitant setting, the environmental sensors can't automatically identify the user who interacted with them. For this reason, in the smart-home multi-inhabitant activity recognition field, an important issue called data association consists of assigning the events generated by the available environmental sensors to the correct users. For instance, if an environmental sensor suggests that the fridge has been opened, how can the system detect which user performed this action?

The purpose of this work is to extend an existing multi-inhabitant HAR system [1] [2] developed at EveryWare Lab that recognizes high-level activities, such as cooking or watching the television, starting from data coming from smartwatches and environmental sensors. This system uses context data to handle the data association problem. A positioning infrastructure composed by BLE beacons and WiFi access points is used to detect the position of the users inside the habitation. Such information is useful to assign an environmental event only to the users who are in the same smart-home area of the triggered sensor. Streams of inertial and environmental data associated in this context-aware manner and regarding each user of the habitation are then given as input to a single-inhabitant activity recognition classifier. The predictions of this classifier consist of vectors containing a probability value for each activity detectable by the system. In the existing system, these vectors are then weighted based on the probability that each detectable activity has to be performed in the area in which the user is.

In this work, another source of context data is added to tackle the data association problem. The inertial data of the smartwatches worn by the users are used to detect the posture of each smart-home inhabitant. This context information can be useful to correctly assign the events generated by the pressure mats installed inside the smart-home. For instance, when a pressure mat is being activated inside the habitation, such an event will be assigned only to the users whose posture is sitting and who are in the same smart-home area of the triggered sensor. Furthermore, when a user is sitting on a pressure mat, environmental events that can't be generated while sitting (e.g., the opening of the fridge) will not be assigned to him. Context data are also used to discard from the inferences of the single-inhabitant activity recognition classifier the activities that are not context-consistent. To perform this process, different context data are used: the position and the posture of the users, the position and the status of the environmental sensors. For instance, the activity watching the television will be discarded by the system, unless the user is sitting on a pressure mat, he is in the same smart-home area of the television smart-plug and the status of this sensor is active. The experimental results of this work show that the use of the posture to tackle the data association problem doesn't improve the overall performances of the system. This happens because the dataset examples used to train and evaluate the activity recognition classifier do not contain cases in which the posture is necessary to correctly assign the environmental events to the users. The deployment of the system in a real-time setting helped to be aware of this issue and showed how the posture information is necessary to correctly associate pressure mats in some multi-inhabitant scenarios. Furthermore, the experimental results suggest that the use of different sources of context information to refine the inferences of the activity recognition classifier helps to improve the recognition rate of the system.

In the existing system, traditional machine learning techniques such as Support Vector Machine were used to perform the activity recognition task. In the last years, deep learning algorithms were widely used in HAR systems because they can improve the recognition rate of more traditional solutions. The purpose of this work includes the use of deep learning algorithms to improve the inference capabilities of the system. The performances of these models have been validated through the leave-one-subject-out cross-validation technique. The dataset used to train and validate the classifiers was previously collected and annotated at EveryWare Lab in previous works. In this thesis, different types of deep learning algorithms are evaluated. The greatest improvements of the system performances are obtained thanks to a deep neural network composed of both convolutional and LSTM layers.

Another consideration is that the system was implemented through supervised learning techniques, a standard solution in the activity recognition field that typically allows achieving a high recognition rate. These methods present different issues. First of all, to train an activity recognition algorithm with a high level of robustness, it is necessary to collect and label a huge quantity of data to build a training set. This process in a smart-home setting is tedious, time-consuming and not always feasible. Furthermore, it is known that training a classifier with labeled data regarding the subjects on which it will be used increases the recognition rate. But, for the same previously mentioned considerations, it is not reasonable to collect and annotate data for each possible user of the activity recognition model. To solve these supervised learning issues, it is possible to use semi-supervised techniques. The idea is to initially train the learning algorithm with a small-sized training set and then to automatically label through appropriate methods new data coming from smartwatches and environmental sensors in real-time. These newly labeled data can be used to increase the size of the training set and then to retrain the statistical model. Another possibility is to exploit the dynamic technique called incremental learning that can be used to directly update the classifier through newly annotated examples without retraining the model on the whole training set. Machine learning models that allow using such a technique are called incremental algorithms. In this work, the semi-supervised active learning technique is used in combination with deep neural networks, that are incremental algorithms. Thanks to active learning, when the system is uncertain about its prediction, it interacts with the users, asking them which is the activity they are performing between the two most likely ones. The feedbacks of the users are then exploited to label new data that will be used to update the deep learning algorithm. In this way, the statistical model is updated over time, customizing the behavior of the system based on the users who interact with it. In this work, the semi-supervised solution allows achieving similar performances of the supervised one on our dataset. The benefit of this solution is that only the examples of a single subject has to be annotated. Its drawback is that, before achieving results similar to the supervised solution, we have to wait for a certain number of classifier updates. Furthermore, experimental results show that the use of context data to refine the prediction of the semi-supervised classifier allows increasing its recognition rate and decreasing over time the number of interactions with the users, necessary to apply the active learning technique.

Along with the development of this work, a demo was deployed to test the system in a real-time setting. During the performed tests, the division of the smart-home into separate areas was different than the one used to collect the dataset. Furthermore, the position of some environmental sensors changed, even if each of them was placed in the same area of the habitation in which it was during the dataset collection process. These changes are useful to make the deployment of the system in a real-world scenario more realistic. The system had to detect the collaborative and individual activities performed by two subjects, using the inertial and environmental data streams coming in real-time from smartwatches and smart-home sensors. The deployment of the demo allows discovering that, in the scenario performed by the two subjects, context data regarding the users' posture were necessary to correctly assign the environmental events generated by the pressure mats installed inside the habitation. Furthermore, refining the classifier predictions through context data is ineffective when a user is in transition between two subsequent activities. In such cases, the statistical classifier makes an uncertain prediction, because the user is actually performing an unknown activity. Discarding the context-inconsistent activities and re-normalizing the probabilities vector emitted by the statistical model can make the inference certain on an incorrect activity. For this reason, instead of refining the predictions of the classifier, context data were used to map each not context-consistent activity to a transition activity.

To summarize, this work focused on the following points:

- improve through other context sources how the system assigns the environmental events to the users who populate the smart-home
- use deep learning algorithms to perform the activity recognition task
- use context data to refine the prediction of the activity recognition classifier
- apply semi-supervised learning techniques
- test the system in a real-time setting

The chapters of this work are organized in this way. Chapter 1 analyzes the state of the art related to HAR systems, semi-supervised techniques and deep learning algorithms relevant for this research field. Chapter 2 illustrates the high-level architecture of the system developed for multi-inhabitant activity recognition. Chapter 3 presents implementation details about the developed system, starting from the preliminary operations applied to the data, passing through the ways used by the system to solve the data association issue, arriving to the activity recognition predictions obtained by the deep learning algorithms and the context refinement applied to these inferences. Chapter 4 describes how the activity recognition system has been extended with semi-supervised learning techniques. Chapter 5 concerns the experimental evaluation of the effectiveness of how the system performs the data association and the context refinement, of the deep learning algorithms used for the activity recognition task and of the developed semi-supervised techniques. Chapter 6 illustrates details related to the real-time system realized to test the proposed solution. The last chapter summarizes the obtained results and proposes possible future works which can improve the system.

Chapter 1

Related work

Human Activity Recognition (HAR) is a research topic of strong interest for the realization of context-aware systems, which can adapt themselves based on user behavior. Examples of these systems include applications in the security, medical and military fields. In the medical field, it is possible to cite the smart-homes for assisted living, i.e. habitations equipped with a network of sensors and actuators which allow monitoring the health condition and the level of self-sufficiency of patients and old people. In literature, detectable activities which can be performed by a user are divided in

- low-level activities: physical activities such as running and walking
- high-level activities: complex actions as cooking, watching the television and eating

Based on the type of sensors used to monitor the users, HAR systems can be classified into two categories [3]: vision-based and sensor-based. The first approach typically involves the use of cameras and computer vision techniques, while the second one exploits a network of sensors which can be

- wearable: placed directly or indirectly on the body of the user, useful to detect his movements
- environmental: attached on objects to detect possible interactions of the user with his surrounding area

Generally, to infer the users' activities it is possible to use two different approaches: data-driven and knowledge-driven. The first involves the use of machine learning techniques, which allow predicting the performed activities through statistical models built thanks to large datasets containing data collected monitoring the users. The second takes advantage of prior knowledge about the application domain of interest to infer the actions performed by the users through logical reasoning. Most of HAR systems which use the data-driven approach work in a supervised way [4]: the activity recognition model is trained with a training set of examples that must be labeled through a time-consuming process. Another possibility is to use unsupervised techniques that work with huge datasets of unlabeled data. Other systems work in a semi-supervised way, i.e. at the beginning, they use a relatively small-sized training set which is then extended over time with new labeled data.

Section 1.1 illustrates a view of sensor-based HAR systems for high-level activities' detection. Section 1.2 concerns data-driven approaches, presented in their supervised and semi-supervised manners. Sections 1.3 and 1.4 describe knowledge-driven and hybrid activity recognition approaches. Finally, Section 1.5 illustrates how deep learning algorithms can be used in the activity recognition field.

1.1 Sensor-based systems for high-level HAR

1.1.1 Mobile devices based activity recognition

Wearable sensors allow detecting the movements of a user. HAR systems based on mobile devices typically exploit the following inertial sensors installed on devices such as smartphones and smartwatches.

- **Accelerometer**: sensor useful to determine the device linear acceleration in the three-dimensional space. Furthermore, it allows identifying the direction in which the device is moving

- **Gyroscope**: sensor useful to determine the orientation of the device in the three-dimensional space

- **Magnetometer**: sensor ables to detect the magnetic field intensity variations along the axes of the three-dimensional space. So, it allows to obtain information about the device orientation

In [5] for example, the performances achieved by two activity recognition classifiers that use data coming respectively from smartphones and smartwatches are compared. It is underlined that smartwatch data are useful to detect activities that involve distinctive hands movements (as eating and using the computer). Furthermore, unlike smartphones, it is more likely that smartwatches are constantly worn during the daily activities of a person. For this reason, they can be exploited in the HAR field.

1.1.2 Environmental sensors based activity recognition

Environmental sensors are useful to detect possible interactions of the user with his surroundings. Examples of environmental sensors used in the HAR field are:

- **Magnetic sensors**: they allow to detect opening and closing events of objects like doors
- **Pressure mats**: placed on chairs, allow to detect the moment in which a user sits down or stands up
- **Smart plugs**: allow measuring the electric power absorbed by a household appliance
- **Passive InfraRed sensors (PIR)**: allow to detect the presence of a subject inside an area

In a multi-inhabitant setting, environmental sensors can't automatically identify the users who interact with them. So, it is crucial to understand how to assign events generated by these sensors to the users who are inside the smart-home. This problem is called data association. This issue doesn't exist in single-inhabitant solutions since, in such cases, each event is always generated by the same subject. In literature, different works propose to handle the data association problem through probabilistic graphical models. In [6] for example, a Hidden Markovian model is used to determine which is the most probable activity performed by a user based on the observed environmental events sequence.

1.1.3 Hybrid approaches

The combination of data coming from wearable and environmental sensors is being tested in different works in literature. In [7], for example, the high-level activities are recognized using the environmental events to identify the room in which is each user and the smartphones' inertial sensors to detect the low-level actions performed by the inhabitants of the smart-home. Even in [8] is being developed a multi-inhabitant HAR system exploiting the combination of inertial and environmental sensors. The data association problem, in this case, is handled through the RFID technology, which however forces the user to wear specific readers of these signals. Furthermore, it is necessary to place an RFID tag on each object the system wants to monitor.

1.2 Data-driven models

Data-driven models involve the use of machine learning techniques, that allow predicting the performed activities through statistical models built thanks to data collected monitoring the users. The advantage of using statistical models is that they can handle sensor data uncertainty and complexity. The drawback of this solution comes from the necessity to have a large dataset to robustly perform the training and learning processes. Most of data-driven HAR systems work in a supervised way: the activity recognition model is trained through a training set of labeled examples. In a smart-home setting, collect and annotate a significant amount of data to build an appropriate training set is a tedious, expensive, time-consuming and not always feasible process. Furthermore, to improve the recognition rate of the system, it is useful to add to the training set data related to that user and then retrain the activity recognition classifier. Since it is not reasonable to collect and annotate data related to each possible user of the system, some applications work in a semi-supervised way, i.e. at the beginning, they use a relatively small-sized training set which is then extended over time with newly labeled data [4].

1.2.1 Supervised approach

In the past years, traditional machine learning models have been widely used in the HAR systems. For instance, in [9] are compared the performances obtained by three classifiers for smart-home activity recognition based on environmental data. The performance of their system achieved good results using Naïve Bayes (NB) and Support Vector Machine (SVM) methods, but better performances were reached using a Random Forest (RF) classifier. Also in [10], an RF classifier is used to detect activities in a smart-home single-inhabitant setting. In the last years, deep learning algorithms have been explored since their performances are promising for different machine learning-based solutions. Section 1.5 will discuss about this type of classifiers.

1.2.2 Semi-supervised approach

Semi-supervised learning techniques can be used to mitigate some supervised approach problems. First of all, supervised approaches request to collect and annotate a significant amount of examples to train a classifier. Furthermore, a person can change the way to perform a specific activity, so statistical models that can evolve and adapt themselves over time based on the users behavior should be used. In realistic conditions, large amounts of unlabelled data are easily collected while a small set of labeled training data is available. Semi-supervised learning techniques' purpose is to annotate part of these unlabelled data in order to expand the training set and improve the recognition system. Generally, machine learning algorithms need to be retrained on the whole training set, while incremental algorithms can be updated using only new available data.

Different approaches such as self-learning, co-learning and active-learning have been applied for semi-supervised learning. In the self-learning paradigm, the predicted label with the highest confidence can be added to the training set or directly used to update an incremental classifier. In [11] different semi-supervised approaches are used to improve activity classification on mobile phones, but in this analysis self-learning never demonstrated any improvement. The co-learning technique uses multiple classifiers, each trained on a different view of the dataset. First, the models are trained with the labeled data. Then, the most confident predictions of each classifier on the unlabeled data are used to increase the training set size. For example, [12] presented a method to recognize user activities from smartphone sensors using the co-learning technique. Unlike self-learning and co-learning, active learning requires user feedbacks to label data with their true label. In [13], a combination of context-aware reasoning and semi-supervised learning is used for activity recognition. Both self-learning and active-learning techniques are used to improve the classifier performances over time.

1.3 Knowledge-driven models

The knowledge-driven approach consists of building activity recognition models through logical reasoning, exploiting the prior knowledge related to the application domain of interest. For example, it is commonsense that the activity "cook pasta" consists of a sequence of actions involving a pot, hot water, a plate, pasta, salt, oil and a sauce. On the other hand, people may perform this activity in different ways [3]. For instance, one may add grated cheese on pasta and another doesn't. Even for the same type of activity, different people may use different items (e.g., different types of sauces) and in different orders (e.g., adding the sauce inside the pot and then fill the plate with pasta, or vice versa). The drawback of this approach is that it could have problems in handling the data uncertainty and that the models could be static and incomplete because, as already mentioned, people can perform the same action in different ways and the same person can change the way he executes an activity. So, the prior knowledge is not sufficient to handle the variability with whom the activities can be performed.

The ontologies are the most used formalism to build reliable knowledge-based activity recognition models. For instance, [14] proposed a knowledge-driven approach for activity recognition based on ontological modeling and semantic reasoning. In [15], an OWL 2 activity ontology is used for modeling, representing and reasoning with complex human activities.

1.4 Hybrid models

There are hybrid approaches that exploit the advantages of both data-driven and knowledge-driven methods. An example is COSAR [16], a mobile HAR system. Statistical techniques are used to identify a set of activities that could be the ones performed by the user. Afterward, a symbolic reasoning module infers between these which are the most logically probable, through the analysis of context data that surround the user. The work [17] presents an approach that uses data-driven techniques to evolve a knowledge-driven activity model. An initial and incomplete knowledge model is used to help a clustering process to detect action clusters that represent possible activities. Based on those action clusters, a learning process is then designed to learn and model different ways of performing activities.

1.5 Deep learning for activity recognition

Conventional machine learning algorithms can achieve satisfying results in the sensor-based activity recognition field. Generally, deep learning approaches are emerging in the literature, since they tend to overcome the performances of standard machine learning solutions on a wide range of research fields. Furthermore, conventional machine learning methods rely on heuristic handcrafted feature extraction, which is usually limited by human domain knowledge and tends to obtain from data only statistical information [18]. Deep learning algorithms tend to overcome this problem because the features can be learned automatically through the neural network. Finally, neural networks can be updated incrementally with new data labeled thanks to semi-supervised techniques.

1.5.1 Artificial Neural Networks theory

1.5.1.1 Introduction

Artificial Neural Networks (ANN) are computational models inspired by the biological neural network. An ANN is a collection of layers. Each layer contains several nodes called artificial neurons. An artificial neuron (Figure 1.1) computes a linear combination of its inputs and gives it to an activation function, that is used to introduce

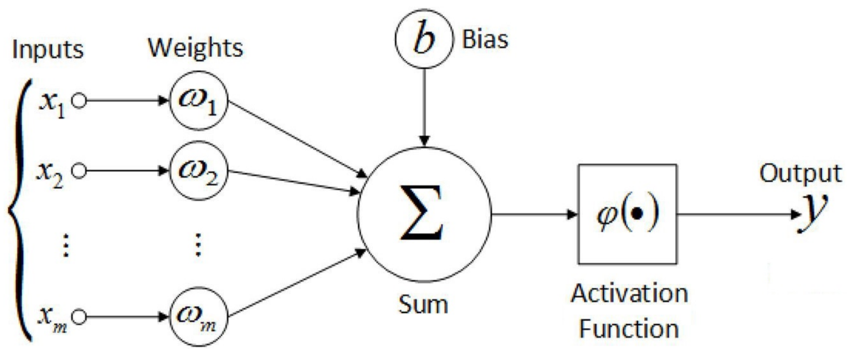


Figure 1.1: An example of artificial neuron.

non-linearity in the neural network. Without activation functions, the neural network would be a linear model, so it would be too simple, generating the underfitting problem illustrated with an example in Figure 1.2. ReLU is currently the most used activation function. It maps negative values to zero and keeps the positive ones. This activation function allows speeding up the training of the model [19]. The output of the activation function of a neuron will be the input of the other neurons connected to the first one. In this way, neurons belonging to different layers are connected. In a neural network, there is an input layer, an output layer and a variable number of hidden layers. A deep learning model is an ANN which contains more than a single hidden layer. The input layer receives as input the data we want to exploit to solve our classification or regression problem. The output layer emits the label that the neural network associates with the received input or a vector containing a probability distribution over the classes detectable by the neural network. The neurons of each layer (excluded the input layer) receive as input the combination of outputs emitted by the neurons of the previous layer.

1.5.1.2 Artificial Neural Network layers

In this section, we are going to describe some of the typical neural network layers used in the Activity Recognition field.

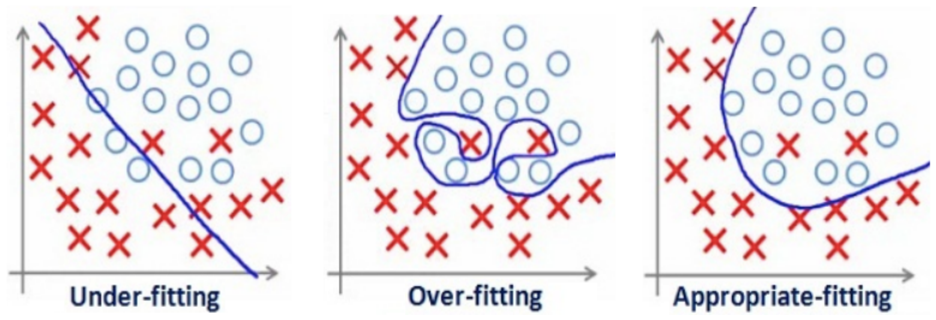


Figure 1.2: Examples of underfitting, overfitting and appropriate-fitting in binary classification.

Fully connected layers In a Dense or Fully Connected (FC) layer, each neuron is connected to all the neurons of the adjacent layers. Typically, the output layer of the network consists of an FC layer that uses the Softmax activation function, whose output is a value between 0 and 1 for each class that the neural network has to recognize. The sum of these values is equal to 1. In this way, the output of the classifier will be a probability distribution over the detectable classes.

Dropout technique The overfitting problem occurs when a machine learning algorithm fits excessively on the training set and can't classify correctly new data. This problem is illustrated with an example in Figure 1.2 and can be reduced with the dropout technique shown in Figure 1.3. When the dropout is applied to a layer, during the training stage some of its neurons will be randomly chosen and disabled. In this way, it is possible to reduce the dependency between neurons of the same layer, increasing the robustness of the neural network.

Batch normalization layers The Internal Covariate Shift is the change in the distributions of layers' inputs. Because of this change, the layers need to continuously adapt themselves to the new distribution they receive. It is possible to reduce the Internal Covariate Shift problem and the training time by fixing the distribution of the layer inputs. In particular, Batch Normalization layers transform the distribution of a layer input to maintain its mean and standard deviation respectively close to 0

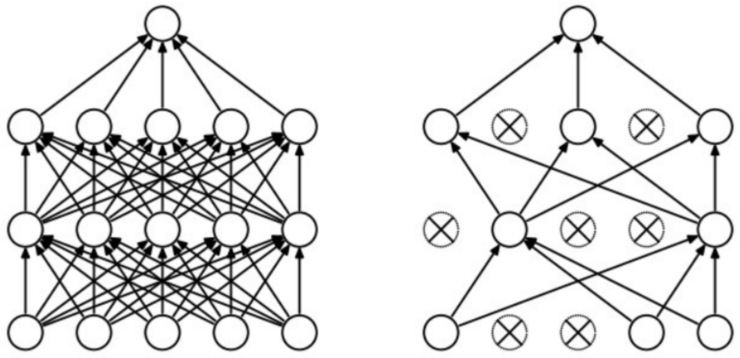


Figure 1.3: Left: a neural network that doesn't use the dropout technique. Right: the same network, using the dropout technique.

and 1 [20]. The application of the batch normalization is similar to the normalization and standardization processes that are typically applied in standard machine learning solutions.

Convolutional layers Convolutional layers are used to extract features from the received input, by applying to it a certain number of filters. The application of each filter allows a convolutional layer to perform a convolution operation on its input, generating what is called a feature map. The convolutional layer output will be the stack of feature maps obtained by applying different filters.

Pooling layers A pooling layer can be used to reduce the dimensionality of the feature maps generated by a convolutional layer, in order to maintain only the most important information. In this way, it is possible to reduce the complexity of the classifier, lightening the computation time and the overfitting problem. One of the most used pooling layers, called Max Pooling, slides a filter on its input and, for each position of the filter, only the maximum input value affected by the filtering operation is kept.

Flatten layers Flatten layers are used to obtain a mono-dimensional vector starting from a multi-dimensional input. So, the output of these layers will be a vector which contains the same number of elements contained in the input. The application of this layer is necessary when the multi-dimensional output of a layer must be mapped into a vector since the following layer needs to receive a mono-dimensional input.

LSTM layers Recurrent layers such as LSTM are used to create networks containing loops. This allows connecting the current network input with information related to past examples, through the following process. Each LSTM layer contains a cell state, whose purpose is to maintain values over arbitrary time intervals during the training stage. Three structures inside the layer called gates are then used to manage the information memorized by the cell state. These gates will decide what information will be thrown away from the cell state, what new information will be used to update it and how to filter it to generate the cell output.

1.5.2 Deep learning for HAR systems

In the last years, deep learning methods have been widely used in HAR systems. Typically, the following types of neural networks have been tested:

- Deep feedforward neural network: contains a certain number of FC layers
- Convolutional Neural Network (CNN) + FC layers: contains convolutional layers followed by FC layers
- Long-Short Term Memory network (LSTM): contains mainly LSTM layers
- CNN + LSTM

In [21], LSTM outperform CNN when the system has to deal with short activities which have a natural order. CNN is better for long-term and repetitive activities such as walking and running. The reason is that recurrent layers can exploit the time-order relationship between data, while CNN is more capable of learning features contained in recurrent patterns. Technically, no model outperforms all the others in all situations [18]. In [22] is shown that the combination of *CNN + LSTM* achieves better performances than the *CNN + FC layers* one. The first one contains four convolutional layers followed by two FC layers, while the second one presents the same structure but with LSTM layers instead of FC ones.

Chapter 2

System architecture

The purpose of the developed system is to infer the high-level activities performed by the users of a smart-home. This system receives data from two different sensing sources: environmental and wearable sensors. The firsts are located inside the smart-home to detect possible users' interactions with drawers, chairs and household appliances; the seconds are the inertial sensors of each smartwatch provided to the smart-home users, who place this device on their dominant arm to allow the software to monitor their movements. Furthermore, the smartwatches receive positioning information about the users coming from a dedicated infrastructure. This information will be used to detect in which smart-home area is each user. Inside a smart-home environment, the usage of smartwatches in place of smartphones is appropriate because it is more likely that the first would be constantly worn by a person in his own home. For example, the smartphone could be placed on the table by the user while he's cooking or eating his lunch. Furthermore, the arm movements detectable thanks to the smartwatch can help the system to distinguish activities that involve only the use of the arms, like eating or wash dishes.

One of the main problems in the multi-inhabitant field is called data association, namely the ability of the system to associate the smart-home environmental events to the different inhabitants. For example, when a pressure mat is activated in the kitchen, how the system can recognize which user sat on it? Context data, like the users' posture and position inside the smart-home or information about the environmental sensors, can help to achieve this task.

Figure 2.1 illustrates the architecture of the developed system. It is possible to notice that it consists of five main modules:

- **Positioning Infrastructure**: it collects positioning data that depend on the positioning technology used by the system. These data are then received by the users' smartwatches and given to the Context Aggregation module
- **Context Aggregation**: it collects context data as position and status of the environmental sensors and as position and posture of the users. The environmental sensors position is known, while their state is derived based on the events generated inside the smart-home. About the other context information, two submodules are used:
  - **Micro-localization**: it recognizes the position of the users inside the smart-home based on positioning data coming from the smartwatches which obtain this information thanks to the dedicated Positioning Infrastructure
  - **Posture**: it recognizes the posture (sitting or not-sitting) of the users based on the inertial data coming from the smartwatches
- **Data Association**: it assigns the environmental events to the users based on the context data collected by the Context Aggregation module
- **Activity Recognition**: it recognizes the high-level activities performed by the users
- **Context Refinement**: it refines through context data the predictions given by the Activity Recognition module

Data acquired by the different devices are used by the Micro-localization and Posture modules which, through machine learning techniques, infer respectively the position of the users inside the smart-home and their posture (sitting or not sitting). This information is used by the Data Association module to determine which users interacted with the sensors installed in the smart-home. The environmental events data, associated to the users, and the measurements coming from the inertial sensors of the smartwatches are used by the Activity Recognition module to infer the actions

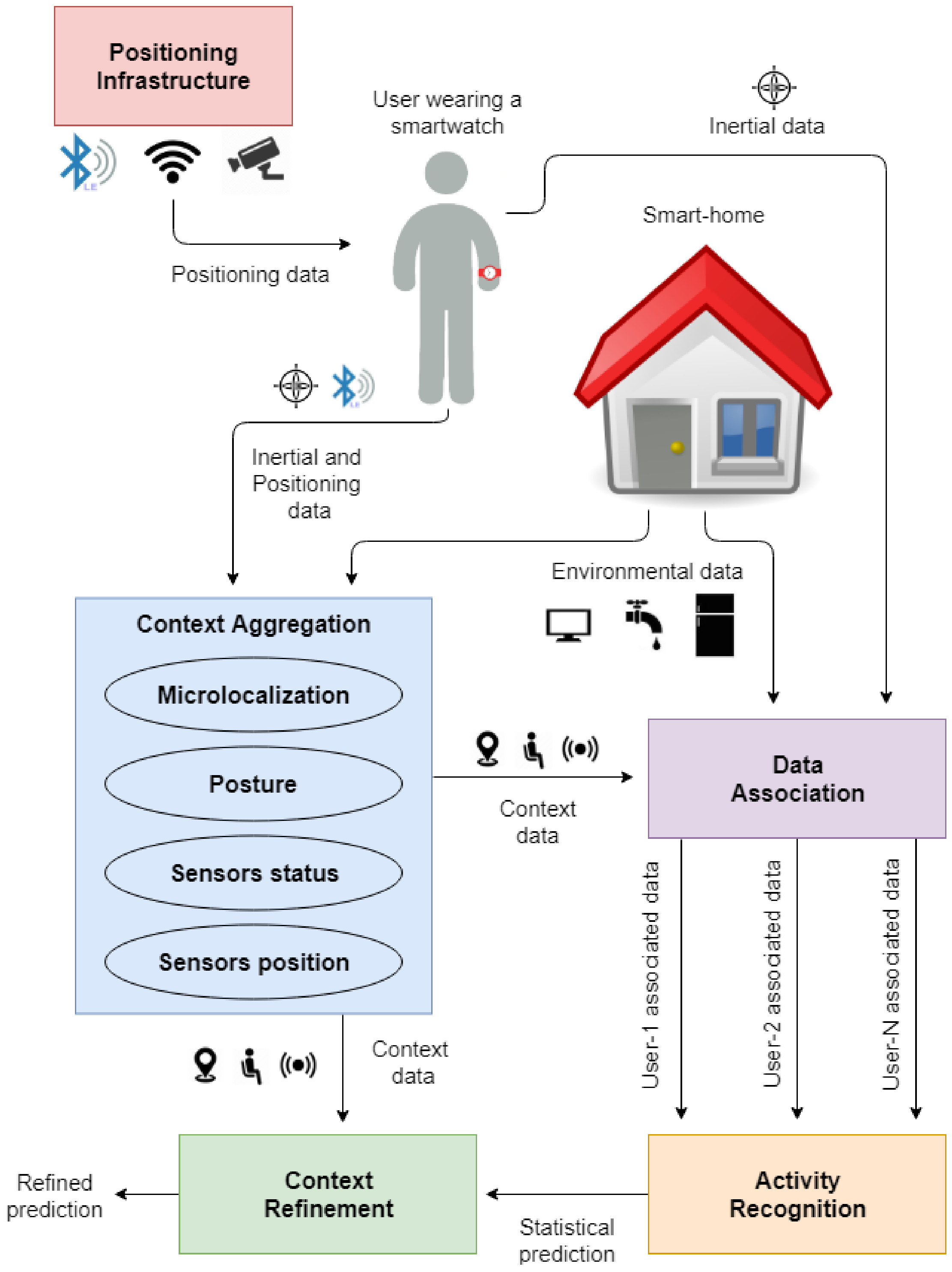


Figure 2.1: Architecture of the developed system.

performed by the users, through deep learning algorithms. Finally, the Context Refinement module refines the responses of the system, based on context data related to environmental sensors position and status, and related to users' position and posture. In the following sections, the different modules of the system are described in detail.

2.1 Sensing devices

2.1.1 Smartwatches and environmental sensors

The smartwatches worn by the users are used by the system to monitor the movements of their dominant arm. This is possible thanks to the inertial sensors (accelerometer, gyroscope and magnetometer) installed inside these devices. Furthermore, Bluetooth and WiFi antennas mounted on smartwatches can be exploited to locate the users inside the smart-home if the positioning infrastructure concerns the use of BLE beacons or WiFi access points.

In this system, different environmental sensors are used to monitor the interaction of the users with their surroundings. The sensors of this type installed inside the smart-home are magnetic sensors, pressure mats and smart-plugs. Magnetic sensors are useful to detect, for instance, the opening of drawers. Pressure mats are exploited to recognize when someone is sitting on a chair. Smart-plugs can be used, for example, to detect when household appliances are turned on.

2.1.2 Positioning infrastructure

The smart-home is divided into semantic areas. The granularity of this division depends on how accurate is the localization technology used by the Positioning Infrastructure. A high-level division could assign every room of the smart-home (living room, kitchen, dining room and office) to a semantic area. A low-level division could assign regions of the same room to different semantic areas: for example, the kitchen could be divided into the cooker area, the fridge area and the sink area. Positioning data are then used by the Micro-localization module to detect the users' position inside the habitation.

This module can implement one of the possible localization technologies known in literature. For example, it can use signals coming from BLE Beacons or WiFi access points or a combination of them.

2.2 Context Aggregation

2.2.1 Environmental sensors position and status

The environmental sensors position inside the smart-home is known (e.g., the cooker is in the kitchen semantic area). Instead, their status is detected and memorized by the system every time an environmental event is generated (e.g., the fridge has been opened/closed).

2.2.2 Micro-localization

This module receives positioning data from the users' smartwatches which obtained them thanks to the Positioning Infrastructure. With this information, it can detect through machine learning techniques the position of the users inside the smart-home. So, its output will be the semantic area in which each user currently is. The smart-home can be divided into semantic areas with different granularity levels. A finer granularity allows providing context information more detailed to the Data Association and Context Refinement modules. Tendentially, this will increase the performances of these modules.

2.2.3 Posture

In this module, a low-level activities classifier is used to detect the posture of a user, starting from the inertial data obtained by his smartwatch. The output of such a classifier is a probability distribution over the low-level activities detectable by the system. Users' posture can help the system during the data association and the context refinement stages. Even in this case, based on the number of low-level activities that the system can detect, it is possible to provide these two modules information more or less sophisticated. The Posture module, for example, could detect only two low-level activities (sitting and not-sitting) or more (sitting, standing, walking).

2.3 Data Association

While the data related to the inertial sensors can be associated automatically to the users, thanks to unique user IDs assigned to each smartwatch, in a multi-inhabitant environment we need to use heuristics to assign the environmental events of the smart-home to the different users. When the inertial data coming from the smartwatches and the environmental data coming from the smart-home arrived as input to the system, it performs the data association exploiting the context data collected by the Context Aggregation module: the environmental events generated inside the smart-home are assigned only to the users whose context is consistent with the event (e.g., only users which are in the kitchen could activate the cooker). In this way, for each user is created a stream of inertial data to which associated environmental events are appended. The main rules used to associate the environmental events are:

- To assign an environmental event to a user, he must be in the same area in which is placed the sensor that generated that event
- When the posture of a user becomes sitting, the system checks if there are unassigned and active pressure mats in the same area of that user. In this case, one of these sensors is associated with the user
- When a pressure mat is assigned to a user, this sensor must not be associated with other users as long as it remains active
- Some environmental events (as the opening of the fridge) can't be assigned to sitting users
- If an environmental event not generated by a pressure mat occurs when more users are in the same area of the sensor, such an event is associated with every user that could have generated it.

Context data received by the Context Aggregation module can be used and combined to perform correctly the data association. For instance, when a sensor is activated in the living room area, the system is sure that this event couldn't be generated by a user which is in the kitchen area. The granularity level used to divide the smart-home into semantic areas could affect the effectiveness of the Data Association module. We can consider the case in which two users are simultaneously in the kitchen and the fridge sensor is activated. In case of high-level areas division, the system will never be sure to assign such an event to one of these users based only on their position. In case of low-level areas division, if only one of the users would be in the fridge area, then the system would be sure about who generated the environmental event. The posture context information is useful to assign correctly the events generated by the pressure mats: when a pressure mat is activated in an area where there are more users, this event can be assigned only to the users whose detected posture becomes sitting. Furthermore, when a user is sitting on a pressure mat, the system can avoid assigning to him events that cannot be triggered in this posture. Indeed, is unlikely that a sitting user can open the fridge or turn on the cooker.

2.4 Activity Recognition

Streams of inertial and associated environmental data for each user are used by the Activity Recognition module to predict in real-time the activities that the subjects are performing inside the smart-home. To perform this task, a single-inhabitant activity recognition classifier is used. Its output will be a probability distribution over the activities detectable by the system. Different machine learning algorithms can be used inside this module: from standard solutions for the HAR field, such as Support Vector Machines, to deep neural networks that have been widely explored in the last years.

2.5 Context Refinement

This module's purpose is to improve the reliability of the system predictions, exploiting context information collected by the Context Aggregation module. We use different kinds of context data for this purpose: environmental sensors status, position of environmental sensors and users inside the smart-home and posture of users.

An example related to the use of sensors and users' position is the following: a user can watch the television only if he is in the same area in which the television smart-plug is placed. The following example instead is related to the status of the environmental sensors and to the users' posture: a user can watch the television only if he is sitting and the television smart-plug is turned on. It is possible to note that, increasing the number of low-level activities detectable by the Posture module will affect the prediction refinement process. For instance, if the system could detect multiple low-level activities, it would know that the user is not sitting, but also that he is walking. In this case, it is possible to discard also the activities which typically are not performed walking, like washing dishes. The following example combines all the context information available to the system: a user can watch the television only if he is sitting, he is in the same area of the television smart-plug and the television smart-plug is turned on.

Chapter 3

Proposed solution

This chapter describes the techniques developed to detect the activities performed by the users in a multi-inhabitant setting, based on the data streams regarding the smartwatch inertial sensors and the smart-home environmental ones. The software modules realized were developed using the language **python** with 3.5.2 version.

Section 3.1 describes preliminary operations applied to the data before the data association process. These operations were already developed in system [1] and consist of data time alignment, data segmentation and the application of a median filter. Section 3.2 describes how the environmental events generated inside the smart-home can be associated with the users, using context information such as the position of sensors and users, and the posture of the latter. Section 3.3 outlines how features were extracted from inertial and environmental sensors data streams. Furthermore, it is described how streams of raw data were created to give them as input to deep learning algorithms letting them handle the feature extraction task. Section 3.4 presents an overview of the deep learning algorithms tested in this work. Finally, Section 3.5 outlines how context data can be used to refine and improve the predictions of the activity recognition classifier.

3.1 Data preprocessing

This section outlines the preliminary operations already developed in a past work [1] which are applied to the data before the data association process. The following

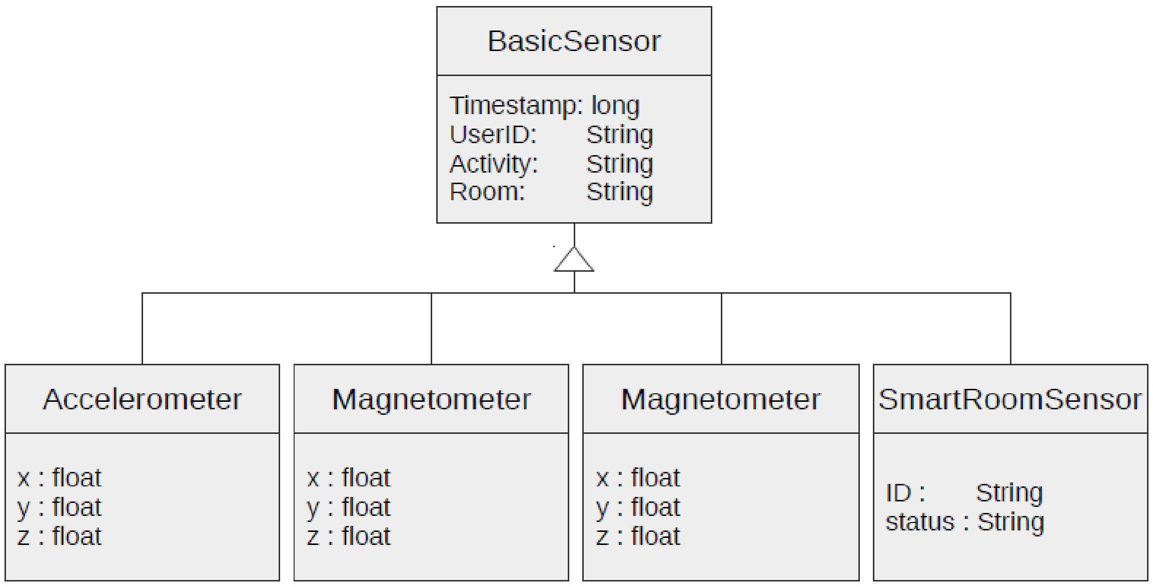


Figure 3.1: Classes hierarchy which represents inertial sensors measurements and environmental events.

sections describe these operations, which are data time alignment, data segmentation and median filtering.

3.1.1 Data time alignment

In this stage, the data structure that represents inertial and environmental sensors data is modified. Indeed, during the acquisition of the dataset, these data were organized in a database inside a structure called **sensorData** which groups them based on the sensor type that generated the measurements. Furthermore, a class for each type of sensor was created and inserted in the classes hierarchy shown in Figure 3.1. At the top of this hierarchy, there is the **BasicSensor** class which stores information commons to all the types of sensors. This information regards the **userID** of the user and the **timestamp** related to the sensor measurement, the **label** regarding the activity performed at that moment and the **room** in which the user was in. In the lowest level of the hierarchy, there are classes that represent the different sensor types. Each of them specifies information related to a particular type of sensor. All of these classes are inside the **sensor** package.

This representation is not convenient to perform a temporal segmentation of the data obtained by all the sensors. For this reason, these data are reorganized in a single structure, ordered based on the timestamp of the measurements related to all the different types of sensors. This process was made for each scenario performed during the dataset acquisition and the results were stored in specific files to avoid the repetition of this operation.

3.1.2 Data segmentation

With data segmentation, smartwatch inertial sensors and environmental sensors measurements are grouped in consecutive temporal windows. In this way, the system can recognize the activities using both data regarding the users' gestures and the history of the last manipulations of the inhabitants with monitored smart-home objects. The segmentation windows are created through a two-steps process.

A first temporal window is defined by a couple *(a, l)*, where *a* represents an environmental event (if present) and *l* identifies an inertial measurements temporal neighbourhood of *a* with a fixed width. This width is determined by two parameters: *past* and *future*, which represent respectively the number of seconds preceding and following the environmental event. Empirically, the best values obtained in [1] on the EveryWare Lab dataset for these parameters were 16 and 0. Exploiting these first type of windows, it is possible to create another kind of data segments that will be used to perform the feature extraction. Each segment is composed by the last *k* environmental events detected and the measurements contained in the last first type of window created as explained above. The *k* parameter allows defining how much the system is interested in the history of environmental events. Empirically, in [1] the best results on the EveryWare Lab dataset were obtained using a *k* parameter equals to 11.

Furthermore, to help the system to detect transitions between activities, an *overlap* percentage between consecutive windows of the second type was introduced. This increases the number of examples presented in the training set as well. Empirically, in [1] 80% of *overlap* was the value that led the system to the best performances on the EveryWare Lab dataset.

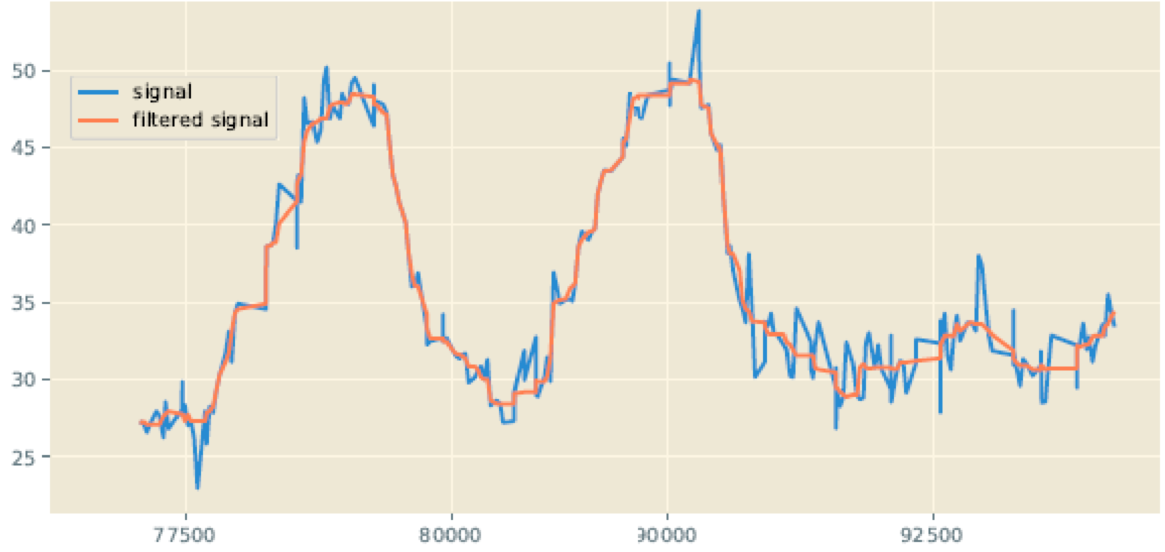


Figure 3.2: Median filtering applied to the signal obtained by the x-axis of the magnetometer.

3.1.3 Median filtering

During this stage, the signal noise of each axis of each sensor inside a temporal window is cleaned through the application of a median filter. In this way, the value of each signal point is approximated with the median value of a neighborhood centered in the point itself and with a fixed width. Empirically, in [1] the best results on the EveryWare Lab dataset were obtained with a width equals to 11. This filtering operation was added in the system using the **medfilt** function of the python **scipy.signal** library. Figure 3.2 shows how the application of a median filter can change a temporal signal.

3.2 Data association

In a multi-inhabitant setting, the system has to detect the activities performed by the users inside the smart-home, recognizing if they are performing independent or group actions. The main problem is the correct association of environmental events to the users which inhabit the smart-home. Wrong associations could create feature vectors containing noisy data: for example, they could hold for a user information about events generated by another user. This makes the learning process more complicated for the activity recognition classifier since inertial data are not sufficient to distinguish high-level activities, especially in front of noisy data regarding the environmental events. In this section, it is described how the system uses context data to tackle the data association problem.

3.2.1 Users' micro-localization and sensors' position

The smart-home is divided into semantic areas. The Micro-localization module developed in [2] provides to the system information about the area in which is each user. To achieve this goal, the module uses the signals detected by the user's smartwatch coming from BLE Beacons and WiFi access points installed inside the smart-home. These data are segmented in 5 seconds windows with a 50% of overlap and filtered with a Savitzky-Golay filter. For each data segment, it is extracted a feature vector containing the mean value of the signal coming from each WiFi and Beacon source. These feature vectors are then given as input to a Random Forest Classifier.

The information about the semantic area in which is placed each environmental sensor is known to the Context Aggregation module. Combining this information with the users micro-localization, it is possible to enhance the data association performed by the system using the heuristics that will be described in Section 3.2.3

3.2.2 Users' posture

To detect the posture of the users which inhabit the smart-home, a low-level activity recognition algorithm was trained receiving as input only the handcrafted features that will be described in Section 3.3.1.2 related to the inertial sensors of the data segments. These features are dimensionally reduced to 84 through the ANOVA technique that will be described in Section 3.3.1.3 and then standardized. Instead, for each data segment, the information coming from environmental sensors was discarded. In this way, the detected posture depends only on the user's gestures. This is important because the posture will be used to perform the data association, so it can't be based on information coming from environmental events. In the training set, the label of each high-level activity was mapped in one of the two postures detectable by this classifier:

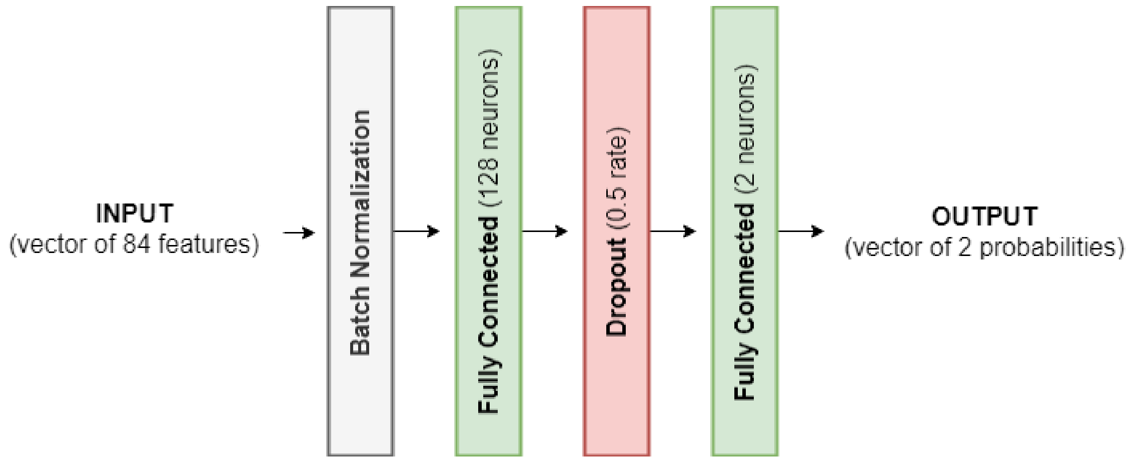

INPUT
(vector of 84 features)
Batch Normalization
Fully Connected (128 neurons)
Dropout (0.5 rate)
Fully Connected (2 neurons)
OUTPUT
(vector of 2 probabilities)

Figure 3.3: Structure of the low-level classifier used to detect the users' posture.

SITTING and NOT_SITTING. Examples of activities that can be performed in both of these postures (such as answering or making a phone call) were not used to train the algorithm.

Figure 3.3 shows the structure of the neural network used to detect the users' posture. At the beginning, the input is given to a *Batch Normalization* layer, which maintains input mean and standard deviation respectively close to 0 and 1. This allows reducing the training time. Then, data flow through two *Fully Connected* (FC) layers, separated by a *Dropout* layer with a rate equals to 0.5, which is necessary to reduce the overfitting problem. The first FC layer is composed of 128 neurons and uses the *ReLU* activation function. The last FC layer contains two neurons, one for each of the detectable postures. In order to obtain a vector with a probability for each of these two postures, the last FC layer uses a *Softmax* activation function. The performances reached by this classifier are explained in Section 5.3.1. The model was developed using the **keras** python library.

3.2.3 Association of events to inhabitants

As already mentioned in this work, inertial data obtained by the smartwatches can be directly associated with the users, thanks to the use of uniques userIDs. Instead, environmental data has to be associated with the people who inhabit the smart-home. The effectiveness of the handcrafted feature extracted from this type of data depends on how the data association is performed.

We can imagine a setting in which we are able to perform a perfect data association that can correctly assign each environmental event to the user who generated it. As we will describe in Section 3.3.1.1, in the feature vectors there are features that count how many times an environmental sensor is being activated or disabled. Furthermore, a time decay function is applied to environmental features regarding disabled sensors to reduce their weight over time. In a multi-inhabitant setting, there can be a problem when the same sensor is activated and disabled by different users. We can consider as U1 the user who activated the sensor, while U2 will be the user who disabled it. In this situation, wrong feature vectors could be created for both the users.

- U1: in his feature vector we have the information about the activation of the sensor, but we don't have the one regarding its deactivation. So, the features related to that sensor will be incorrect: the time decay function will not be applied to a feature related to the sensor since it is still considered active, while the feature which counts the deactivation events will not be increased.
- U2: here we can consider two cases:
  - if the activity of U2 doesn't depend on the sensor, then the feature vector will be right because the activity he is performing is not based on the manipulation of that sensor. This can happen when U2 closes a drawer which U1 forgot to close.
  - if the activity of U2 depends on the sensor, then also his feature vector will be wrong because it doesn't contain information about the activation event of the sensor. This can happen when U1 turns on the tv, he watches it with U2 and then U2 turns it off.

We can now analyze a naive solution in which any environmental event is associated with all the users. The problem described above is solved because the activation and deactivation events will be associated with both U1 and U2. However, in this situation to a user we could associate events belonging to other users, creating noisy feature vectors.

After the previous analysis, we can conclude that

- we should assign the deactivation event of a sensor to the user who generated the event, but also to the user who previously activated the sensor
- we shouldn't assign any environmental event to any users

So, when the system has to create a feature vector starting from a data segment related to a user, it follows this scheme:

- at the beginning, associate with the user any environmental event, even without knowing who generated it
- then, remove the assignements that are not context-consistent

In this work, three different context sources were used to tackle the data association problem. These can be also combined in order to obtained hybrid approaches.

- **Micro-areas (M)**
- **Posture (P)**
- **PosturePlus (P+)**

The results obtained with the combination of these three approaches are described in Section 5.3.2.

Micro-areas approach This approach uses the following heuristic to decide how to perform the data association exploiting positioning context data: if a sensor is being activated or disabled when the user is not in its own semantic area, then this event is not considered. This heuristic is not applied in two cases: when the event regards activities which can be performed in any smart-home area (such as answering or making a phone call) and when it concerns the deactivation of the pressure mat on which the user was sitting. This choice allows the system to recognize when to unallocate a pressure mat from a user in cases in which the detected user position is wrong. This is important when the Micro-areas approach is combined with the Posture or the PosturePlus one.

Posture approach This approach uses the following heuristics to decide how to perform the data association exploiting context data concerning the users' posture:

- once a pressure mat is assigned to a user, that sensor will not be assigned to other users as long as it remains active
- if a user is not sitting on a pressure mat, the system uses the posture classifier to detect his posture based on the inertial data coming from his smartwatch
  - if the detected posture is NOT_SITTING, then if the user in the past was sitting on a pressure mat, only the features related to that pressure mat will not be zeroed. In this way, the time decay function will continue to be applied to that sensor features. If the user in the past wasn't sitting on a pressure mat, then all the features related to pressure mats will be zeroed
  - if the detected posture is SITTING, then the last activated pressure mat is assigned to the user. Only the features related to that pressure mat will not be zeroed

This approach leads to the following problems:

- when the system assigns to the user the latest activated pressure mat, it could assign a pressure mat on which actually another user is sitting. Combining this approach with the Micro-areas one, this problem is solved because we assign to the user one of the pressure mats placed in his own smart-home area and, as we are going to discuss in Section 3.3, features concerning pressure mats of the same area will be then grouped.
- the posture classifier sometimes confuses the inertial data regarding the COOKING activity with the ones concerning the WATCHING_TV activity. So, in these cases, the detected posture will be SITTING and this can lead to a wrong data association. The same problem occurs with the inertial data concerning the activities EATING and TAKING_MEDICINES. Within this list, this is the only unresolved problem, whose solutions will be discussed in Chapter 7.
- some high-level activities can be performed both with a sitting and a not-sitting posture. These activities are ANSWERING_PHONE and

MAKING_PHONE_CALL. Also in these cases, the system could generate wrong feature vectors. To solve this issue, the heuristics of the posture approach shouldn't be applied to feature vectors related to these activities during the training of the high-level activities classifier.

Because of the last two problems, the high-level activities classifier which uses the posture approach was trained without applying the previously described heuristics for feature vectors related to ANSWERING_PHONE, MAKING_PHONE_CALL, COOKING and TAKING_MEDICINES activities.

PosturePlus approach This approach uses the following heuristic to decide how to perform the data association exploiting context data concerning the users' posture: some environmental events shouldn't be assigned to sitting users. These events are all the detectable environmental events (such as the opening of the fridge or of a drawer) excluding turning on the television, answering and making a phone call. The idea is that, in a real setting, it is unlikely that a user would open the fridge while he is sitting on a chair. The problem of this approach is related to the posture classifier issues described during the analysis of the Posture method. With this approach, for example, the system could not assign the turning on of the cooker to the user who is performing the activity COOKING, because his detected posture is SITTING.

3.3 Feature extraction

In this work, deep learning algorithms are trained and tested in two ways: giving them as input handcrafted features or raw data. Handcrafted features are extracted both from environmental and inertial sensors. Then they are dimensionally reduced through the application of the ANOVA technique and finally standardized. This approach was already developed in [1] and it is explained in Section 3.3.1. In this work, only the environmental sensors feature extraction was modified in order to group features that represent the same semantic information. Instead, when raw data are used, the input of the activity recognition model consists of matrices in which the first row contains the environmental sensors handcrafted features, while each of the other rows holds raw data coming from the axis of a single inertial sensor. This approach is explained in Section 3.3.2.

3.3.1 Handcrafted features

3.3.1.1 Environmental sensors feature extraction

In order to define the environmental sensors features, we can consider the following intuitions:

- We need to count how many times a sensor is activated and disabled
- Sensors which were activated more recently are more important than the others
- If the status of a sensor is active, this indicates that there is an ongoing interaction between the user and the object monitored by that sensor. So, this sensor must have greater importance than the sensors activated in the past
- Features related to pressure mats placed in the same smart-home area can be synthesized in a single feature. The idea is that, for the activity recognition classifier, it is important to know that there is at least one active pressure mat in a specific semantic area and it shouldn't be important to know which is. This because pressure mats of the same smart-home area communicate the same semantic information. This consideration can be done when the same type of environmental sensors (e.g, pressure mats, smart-plugs) is used to monitor the same kind of smart-home objects (e.g., the chairs, the stoves)

So, two types of features were used: one based on counting environmental events, the other based on the status of the sensors. To both types of features is applied a time decay function, that reduces the weight of the feature over time.

- **Temporal extraction counting events**: for each environmental sensor, two features of this type are defined. One counts the number of activation events, the other counts the number of deactivation events of that sensor. To each of these features (if non-zero) is applied a time decay function. Considering that in the smart-home there are 16 environmental sensors and 2 smartphone events are used (making of an outgoing call and answering to an incoming call), so 36 features of this type are defined. Grouping features related to pressure mats placed in the same area, this number is reduced to 24.

- **Temporal extraction dependent on the status**: it counts the number of activation events related to a sensor and applies a time decay function only if the sensor status is disabled. Grouping features related to pressure mats placed in the same area, the number of this type of feature is 12.

So, 36 environmental sensors' features are defined in total.

3.3.1.2 Inertial sensors feature extraction

In order to characterize users' movements through inertial sensors, the work [1] considered features that are well-known in the activity recognition literature [4]. The following features were extracted from each axis of each smartwatch inertial sensor considered (accelerometer, gyroscope and magnetometer).

- **Mean**: it computes the mean value of the signal. It helps to detect if an activity is static (lower mean value) or not (higher mean value)
- **Difference between max and min**: it computes the difference between the maximum and the minimum values contained in a data segment
- **Variance**: it computes the signal variance, which provides a measure of the variability of the signal
- **Standard Deviation**: it computes the signal standard deviation, which provides a measure of the stability of the signal
- **Median**: it represents the value which separates the upper half from the lower half of the signal
- **Root Mean Square (RMS)**: it computes the square root of the mean of the signal values squares

- **Kurtosis**: it represents information regarding the shape of the signal values distribution. It expresses how this distribution deviates from a normal distribution
- **Symmetry**: it expresses how the values distribution is symmetric. A zero value indicates a symmetric distribution, while non-zero values indicate an asymmetric distribution
- **Zero-crossing rate**: it computes the number of times in which the signal crosses a specific value, called zero. In this case, the zero value is the value which corresponds to half of the signal range. This feature helps to detect oscillatory and repetitive movements, typical of activities such as eating or drinking
- **Number of peaks**: it counts the number of maximum and minimum signal pitches. This feature helps to distinguish dynamic activities from static ones
- **Energy**: it computes the sum of the squares of the signal points absolute value

Furthermore, for each inertial sensor the following correlation indices are computed for each combination of its axes:

- **Pearson correlation**: it is used to detect if there is a linear relationship between two signals, i.e. it expresses the attitude of a signal to change based on the variation of another signal.
- **Cross correlation**: it provides a measure of similarity between two signals

Finally, for each inertial sensor, the following features is computed on all of its axes:

- **Magnitude**: this feature is used to detect strong changes during the execution of the activities

In this way, 40 features for each inertial sensor are defined, obtaining 120 total features related to the inertial data.

3.3.1.3 Dimensionality reduction

This stage allows finding between all the previously defined features which are the most meaningful ones. These will be the features that distinguish better the activities and they will be given as input to the deep learning algorithms. To select the k best features, the work [1] used a technique based on the variance analysis called ANOVA (ANalysis Of VAriance) [23]. This approach allows comparing groups of data based on the variability of each group (within variance) and the variability between the groups (between variance). In this case, each group consists of all the feature vectors related to the same activity. To compute this comparison, for each feature is computed the ratio between the between and the within variance of the groups through the statistical test called F-test. The k most meaningful features are selected based on the ones which obtained the highest F-test values. Empirically, in [1] the use of 84 features led to the best results on the EveryWare Lab dataset. This means that, when the deep learning algorithms are trained with handcrafted features, they receive as input a vector containing 84 values for each data segment generated by the system. The implementation of this stage is realized through the **SelectKBest** class of the **sklearn.feature_selection** python library.

In this work, environmental sensors features related to pressure mats placed in the same smart-home areas are collapsed in a single feature. This approach led to an improvement of the system performances. This could have happened because, in [1], the application of the ANOVA technique discarded the features related to the pressure mats that were activated less frequently during the dataset collection. For this reason, the information about the activation of these pressure mats would not be given as input to the activity recognition classifier and maybe this affects its recognition rate. Now, probably, all the features related to pressure mats remain in the feature vectors. Due to time constraints, it was not possible to verify this hypothesis during the development of this work. Furthermore, it should be done again the tuning of the k ANOVA parameter since some environmental features are changed.

3.3.1.4 Standardization

Standardization consists of a transformation which makes the data distribution equals to a distribution with zero mean and variance equals to 1. This avoids that features with a higher variance dominates the learning function of the activity recognition classifier. The standardisation operation is realized using the **StandardScaler** class of the **sklearn.preprocessing** python library.

3.3.2 Raw data for deep learning feature extraction

When we want to let the deep learning algorithms handle the feature extraction operation, data coming from the smartwatch inertial sensors are segmented, manipulated through a median filter and then given directly as input to the learning models. Instead, features regarding the environmental sensors are extracted in a handcrafted way, as described in Section 3.3.1.1. This choice comes from the idea that maybe it is not necessary to use too much detailed information about environmental events to correctly predict an activity. For example, to distinguish the activity ”setting up the table” from the activity ”washing dishes” it doesn't seem required to know for each sample of time if the cutlery drawer was opened or closed. Probably it is sufficient to know if it was opened at least once in the last temporal window of data.

So, when the deep learning algorithms are trained with raw data, they receive as input, for each data segment generated by the system, a vector containing the 36 environmental handcrafted features and a matrix with shape *(9, samples_num)*, where *samples_num* is the mean number of samples measured by the inertial sensors for each data segment created using a specific window size. At the beginning, it is created a matrix of 9 rows. Each of the 9 rows contains the measurements coming from each axis of each smartwatch inertial sensor. The system needs to apply to this matrix a function called **interpolation_or_subsampling** because of two related reasons. The first reason is that the number of samples coming from the different inertial sensors changes based on their sampling rate. So, each of these 9 rows contains a different number of values. The second reason is that the shape of the input received by the deep learning algorithms must be always the same. The **interpolation_or_subsampling** function receives as input the 9 rows regarding the inertial data and the *samples_num* value. Then it forces each input row to have a number of samples that is equal to *samples_num*, trough the interpolation (to add samples) or the subsampling (to discard samples) techniques. Both techniques, when applied to a row, require the generation of a random number r that represents a possible position of the row. The interpolation inserts in the r-th position of the row the mean value between its r-th and $r+1$-th values. The subsampling, instead, discards the r-th value of the row. These techniques are applied iteratively to each row until it contains a number of values equal to *samples_num*. A data segment is discarded if its rows contain less than *samples_num - 100* samples. In this way, the system doesn't consider data segments that contain a number of samples too low compared to the *samples_num* value.

3.4 Deep learning for activity recognition

Once the environmental events are associated with the different smart-home users, it is possible to use deep learning algorithms in order to detect their activities. In this work, two different shapes of input were used to train the activity recognition models. In one case, the input consists of vectors containing 84 handcrafted features based on environmental and inertial sensors, dimensionally reduced through the ANOVA technique and finally standardized. In the other case, features are manually extracted only for the environmental sensors, while inertial data are directly used in their raw shape. In both cases, we tested four different artificial neural networks based on the type of layers they use:

- a network which contains mainly *Fully Connected* layers
- a network which contains *Convolutional* and *Fully Connected* layers
- a network which contains *LSTM* and *Fully Connected* layers
- a network which contains *Convolutional*, *LSTM* and *Fully Connected* layers

At the beginning of Section 1.5 is available a theoretical introduction to neural networks and their layers. Each of these networks was trained using the **keras** python

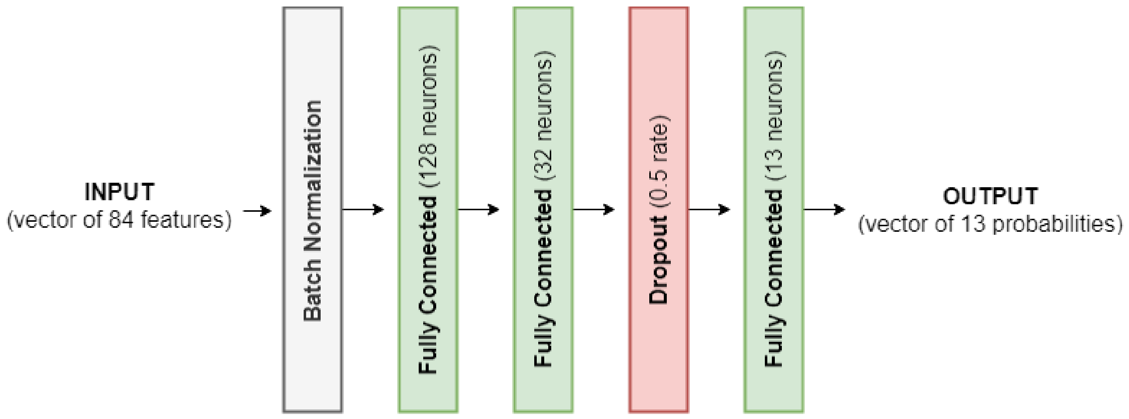

INPUT
(vector of 84 features)
Batch Normalization
Fully Connected (128 neurons)
Fully Connected (32 neurons)
Dropout (0.5 rate)
Fully Connected (13 neurons)
OUTPUT
(vector of 13 probabilities)

Figure 3.4: Structure of the activity recognition algorithm which uses handcrafted features and mainly *Fully Connected* layers.

library and presents one of the typical neural network structures used in the activity recognition field (as described in Section 1.5). In this thesis, the parameters of each neural network layer were experimentally tuned. The results obtained using these classifiers are illustrated in Section 5.3. Section 3.4.1 describes the structure of the neural networks trained using as input vectors of 84 handcrafted features, while Section 3.4.2 illustrate the structure of the deep learning algorithms trained using as input matrices of 9 rows containing raw data generated by inertial sensors and a vector holding the 36 environmental handcrafted features.

3.4.1 Use of handcrafted features

In this section the neural networks trained using handcrafted features are described. Figure 3.4 shows the structure of the classifier which uses mainly *FC* layers. The first is a *Batch Normalization* layer, as already mentioned, useful to reduce the training time. After that, data flow through two *FC* layers containing respectively 128 and 32 *neurons*, and both using the *ReLU* activation function. Before the last *FC* layer there is a *Dropout* layer, used to reduce the *overfitting* problem. Finally, another *FC* layer uses 13 *neurons* (one for each of the 13 high-level detectable activities by the system) and the *Softmax* activation function.

Figure 3.5 describes the model which uses *Convolutional* and *FC* layers. After

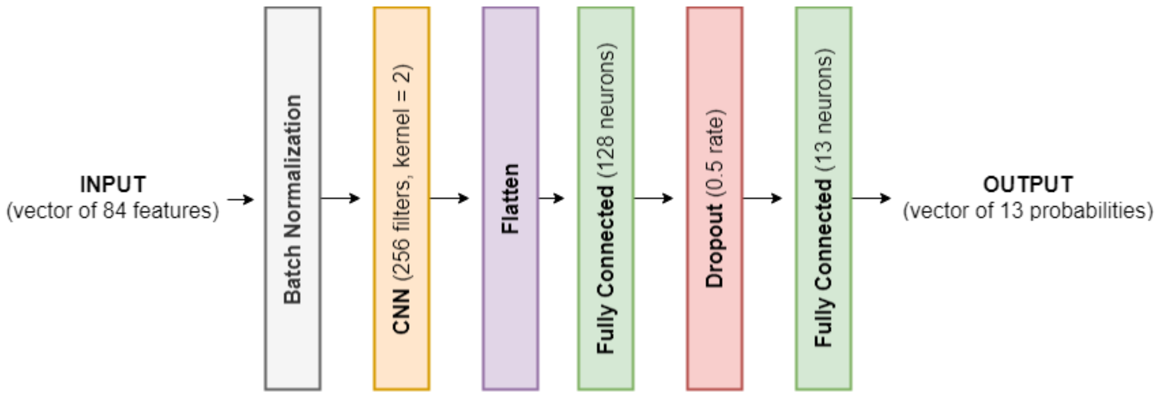

INPUT
(vector of 84 features)
Batch Normalization
CNN (256 filters, kernel = 2)
Flatten
Fully Connected (128 neurons)
Dropout (0.5 rate)
Fully Connected (13 neurons)
OUTPUT
(vector of 13 probabilities)

Figure 3.5: Structure of the activity recognition algorithm which uses handcrafted features, *Convolutional* and *Fully Connected* layers.

the *Batch Normalization*, a one-dimensional *Convolutional* layer is used in order to identify patterns between features close to each other contained in the input vector. A flatten layer is necessary to obtain a monodimensional vector to give as input to the next layers. Again, we have a 128 neurons *FC* layer with a *ReLU* activation function, a *Dropout* layer and the output *FC* layer with the *Softmax* activation function.

Figure 3.6 describes the activity recognition algorithm which uses *LSTM* and *FC* layers. An *LSTM* layer with 128 *units* is used to maintain inside the network information about past examples of the training set. Each *LSTM* layer in **keras** library has a boolean parameter called *stateful*. If the layer is stateful, its memory will consider the examples of the whole training set. If the layer is stateless, its memory will be reset after each batch during the training stage. This means that the layer will learn only the temporal patterns present inside the single batches. In this work, each *LSTM* layer is stateless. This choice is based on the following idea: if the system needs to predict the activity CLEARING_TABLE, it should be enough for it to know that the last activities performed by the user were COOKING and EATING, it shouldn't be necessary for it to know the activities he performed during the whole day. Finally, Figure 3.7 presents a combination of the last two described structures.

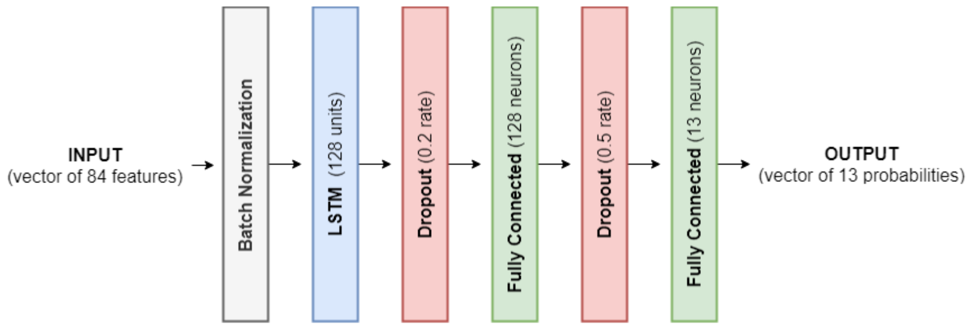


Figure 3.6: Structure of the activity recognition algorithm which uses handcrafted features, *LSTM* and *Fully Connected* layers.

3.4.2 Use of raw data

In this section the deep learning algorithms trained using raw data coming from inertial sensors and handcrafted features extracted from environmental ones are described. It is important to clarify that the size of the windows use to segment data will affect these classifiers' input shape. We have already mentioned that the inertial data will be given as input through a matrix that depends on a variable called *samples_num*. This variable value indicates the number of inertial measurements collected inside a single segmentation window. When we'll describe the experimental results obtained by the following classifiers in Chapter 5, we will consider that reducing the window size, also the *samples_num* value will be decreased. So, using the same neural network structure varying the window size will affect the relative complexity of the model. For example, if the first FC layer of the classifier contains 512 neurons, the complexity of the model will be relatively greater when segmentation windows of 14 seconds are used instead of 16 seconds ones.

Figure 3.8 shows the structure of the classifier which uses mainly *FC* layers. It is possible to notice that the model at the beginning is divided into two stacks of layers. The idea is to split the input since data coming from inertial sensors are raw, while information related to environmental sensors is obtained through handcrafted features. The first stack receives as input the matrix containing the inertial information. These data flow through different *FC* layers before being flattened in order to obtain a vector of values needed by the *Concatenate* layer to combine the outputs of

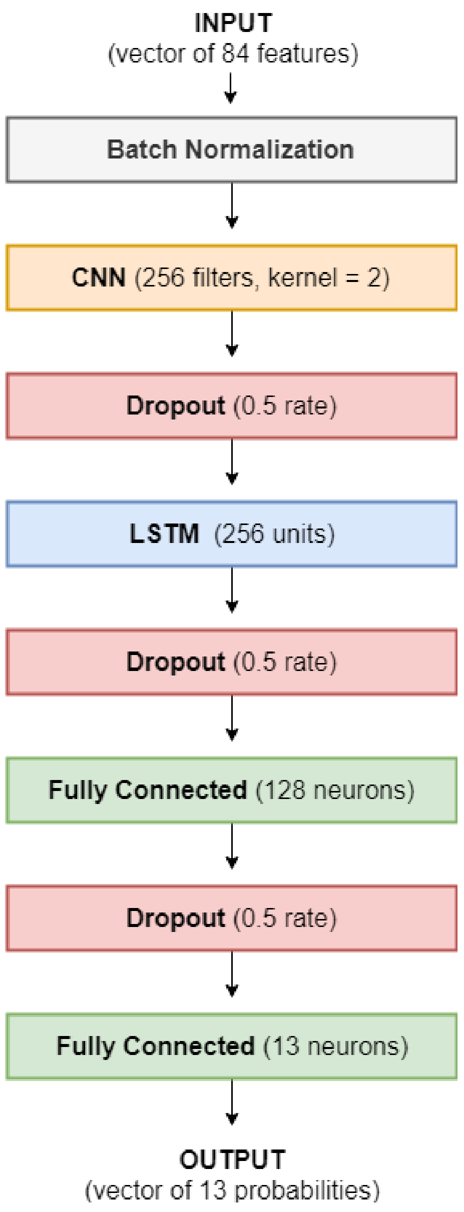


Figure 3.7: Structure of the activity recognition algorithm which uses handcrafted features and a combination of *Convolutional*, *LSTM* and *Fully Connected* layers.

the two stacks which must be of the same shape. The second stack receives as input the vector containing the environmental handcrafted features. After the *Concatenate* layer, another couple of *FC* layers are used before the last one which uses the *Softmax* activation function.

Figure 3.9 describes the model which uses *Convolutional* and *FC* layers. In this case, inertial information flows through bi-dimensional *Convolutional* layers, useful to extract features from the received data. Kernels with shape (2,2) are used to find patterns between data contained in rows close to each other. This is important because in this way the convolutional filters work with information provided by pairs of axes of the same inertial sensor. This is another reason which explains why data coming from inertial and environmental sensors were divided into two different stacks: the application of these convolutional layers to matrix rows containing semantically different information shouldn't lead to performance improvements. A *Batch Normalization* layer is added before each *Convolutional* one to speed up the model training. After the *Convolutional* layers, inertial data flow through different *FC* layers before reaching the *Concatenate* one. Environmental data already consist of handcrafted features, so they don't pass through any *Convolutional* layer.

Figure 3.10 shows the structure of the classifier which uses *LSTM* and *FC* layers. After the *LSTM* layers, raw and environmental data pass through *FC* layers. The only difference is that environmental information flows through two recurrent layers instead of only one. Finally, Figure 3.11 presents a combination of the last two described structures.

3.5 Context refinement

Information provided by the Context Aggregation module can be used to improve the system performances. In this work, two approaches were tested. In the first case, once the activity recognition classifier generates its prediction, the Context Refinement module refines this inference using context information. In the second case, context data such as position and posture of the users are included directly in the feature vectors used to train the machine learning algorithms. The following sections describe the implementation of these approaches.

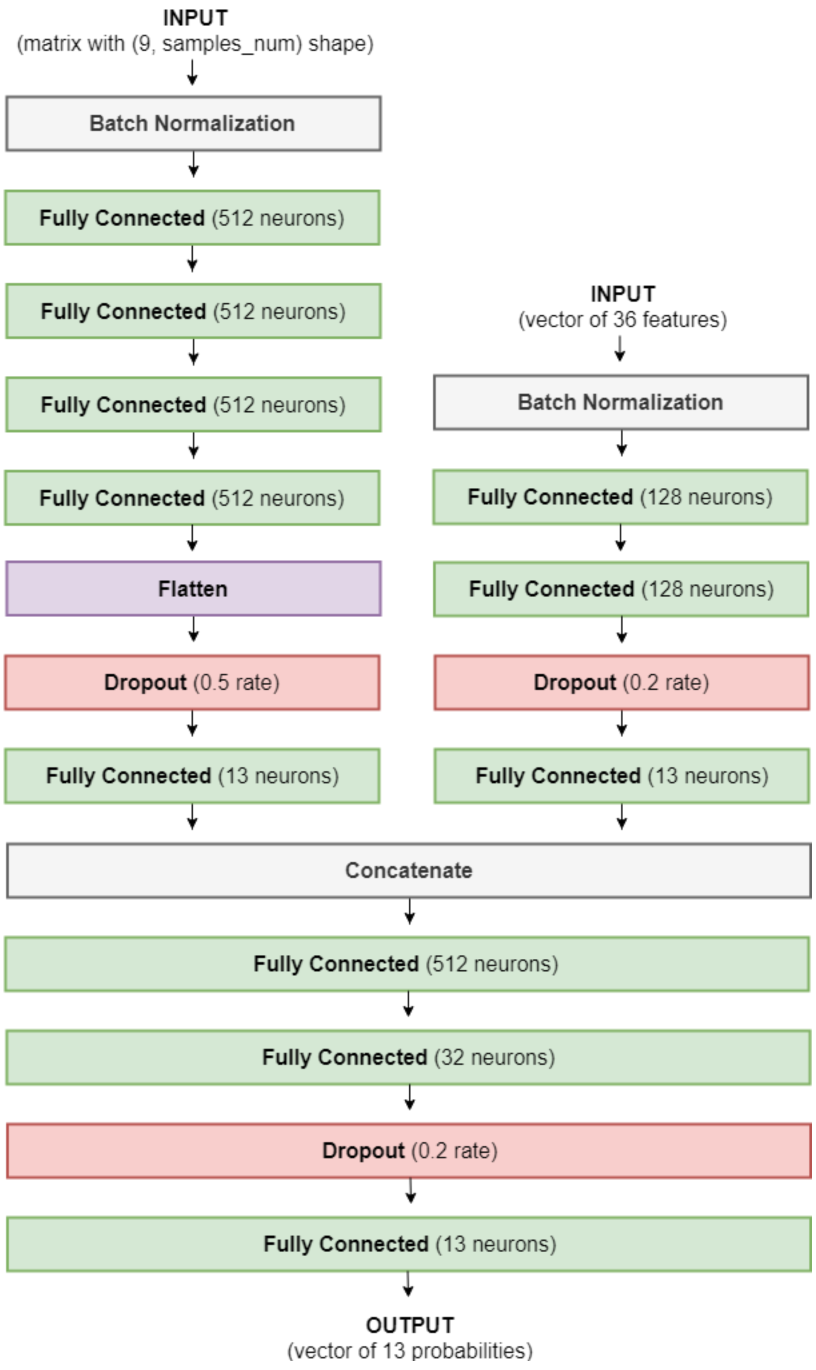

INPUT
(matrix with (9, samples_num) shape)
Batch Normalization
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Flatten
Dropout (0.5 rate)
Fully Connected (13 neurons)
INPUT
(vector of 36 features)
Batch Normalization
Fully Connected (128 neurons)
Fully Connected (128 neurons)
Dropout (0.2 rate)
Fully Connected (13 neurons)
Concatenate
Fully Connected (512 neurons)
Fully Connected (32 neurons)
Dropout (0.2 rate)
Fully Connected (13 neurons)
OUTPUT
(vector of 13 probabilities)

Figure 3.8

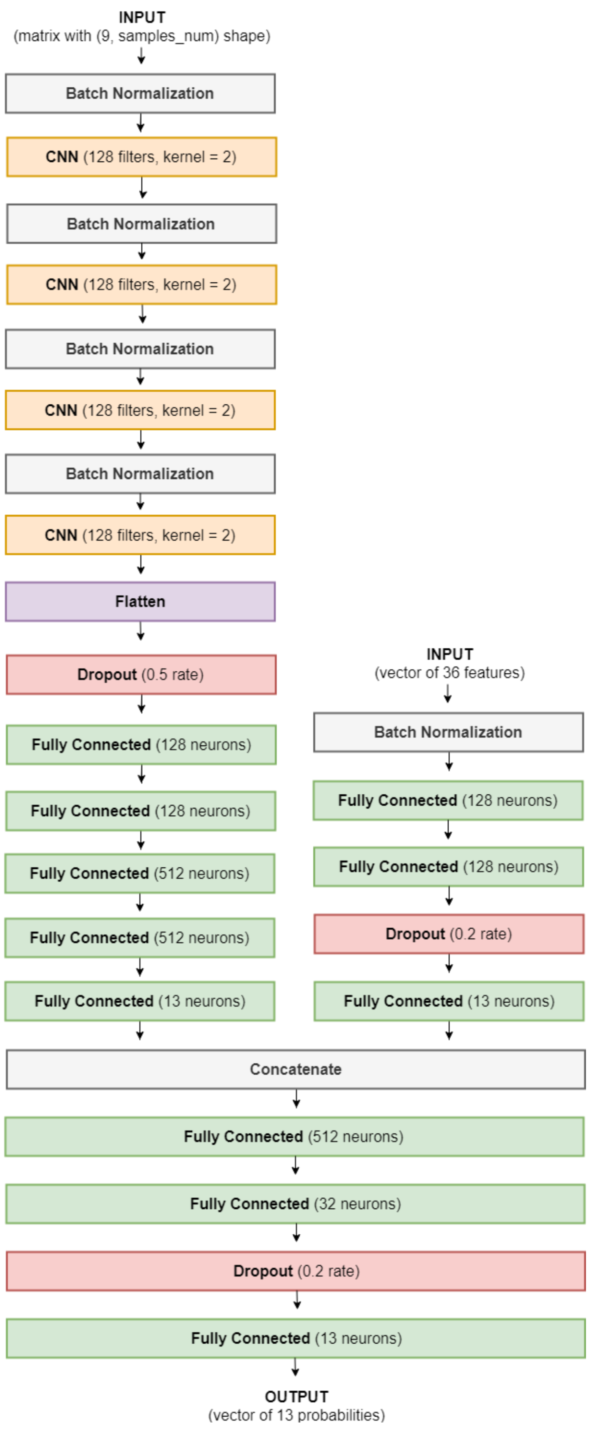

INPUT
(matrix with (9, samples_num) shape)
Batch Normalization
CNN (128 filters, kernel = 2)
Batch Normalization
CNN (128 filters, kernel = 2)
Batch Normalization
CNN (128 filters, kernel = 2)
Batch Normalization
CNN (128 filters, kernel = 2)
Flatten
Dropout (0.5 rate)
Fully Connected (128 neurons)
Fully Connected (128 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (13 neurons)
INPUT
(vector of 36 features)
Batch Normalization
Fully Connected (128 neurons)
Fully Connected (128 neurons)
Dropout (0.2 rate)
Fully Connected (13 neurons)
Concatenate
Fully Connected (512 neurons)
Fully Connected (32 neurons)
Dropout (0.2 rate)
Fully Connected (13 neurons)
OUTPUT
(vector of 13 probabilities)

Figure 3.9

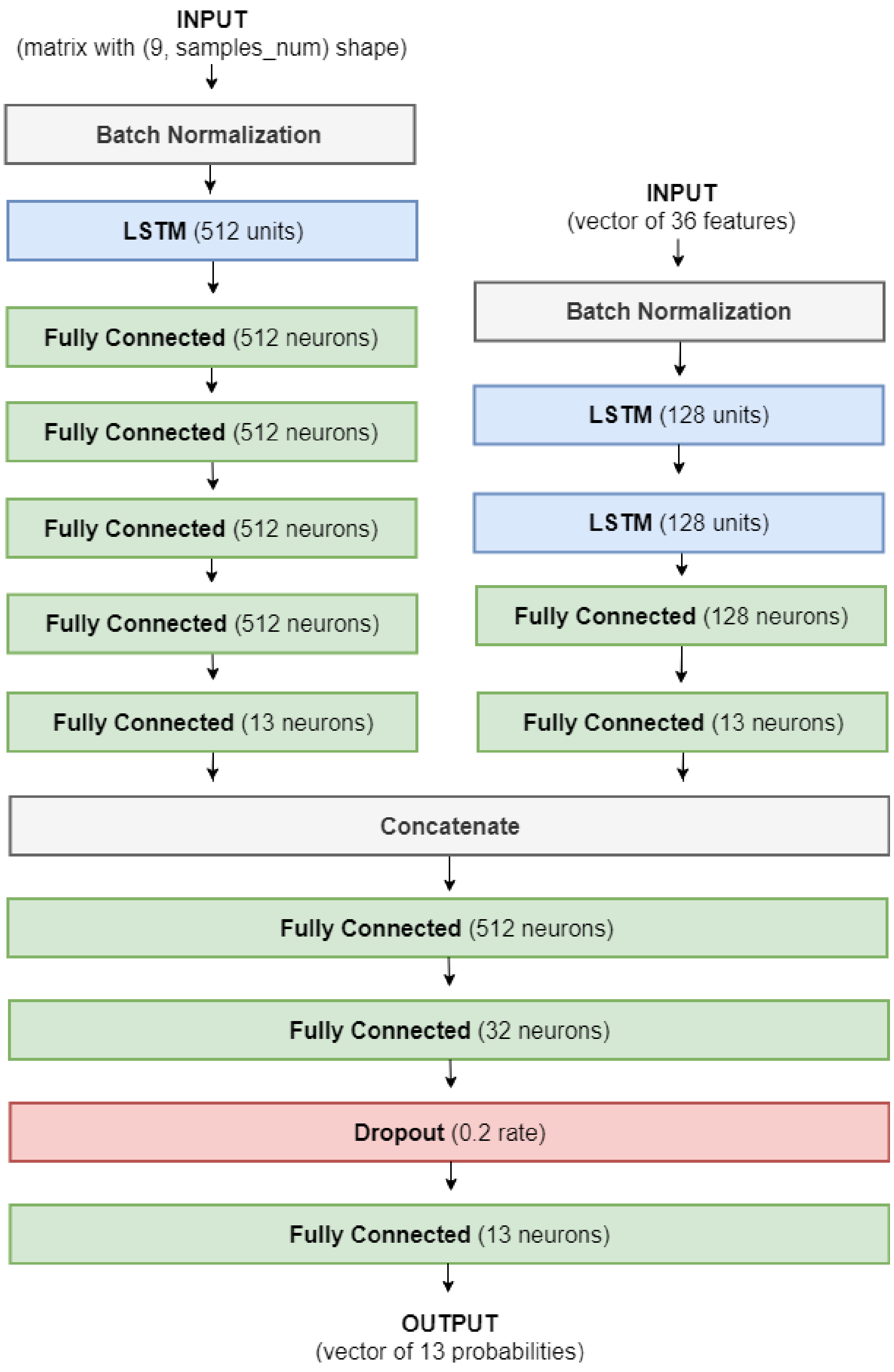

INPUT
(matrix with (9, samples_num) shape)
Batch Normalization
LSTM (512 units)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (13 neurons)
INPUT
(vector of 36 features)
Batch Normalization
LSTM (128 units)
LSTM (128 units)
Fully Connected (128 neurons)
Fully Connected (13 neurons)
Concatenate
Fully Connected (512 neurons)
Fully Connected (32 neurons)
Dropout (0.2 rate)
Fully Connected (13 neurons)
OUTPUT
(vector of 13 probabilities)

Figure 3.10

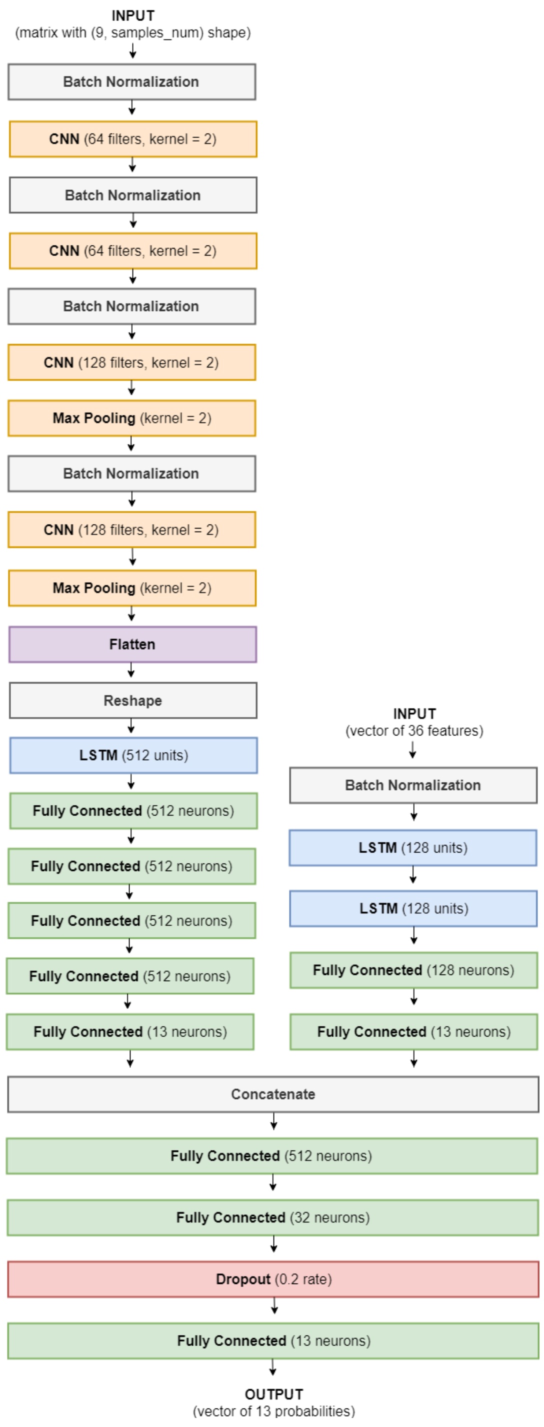

INPUT
(matrix with (9, samples_num) shape)
Batch Normalization
CNN (64 filters, kernel = 2)
Batch Normalization
CNN (64 filters, kernel = 2)
Batch Normalization
CNN (128 filters, kernel = 2)
Max Pooling (kernel = 2)
Batch Normalization
CNN (128 filters, kernel = 2)
Max Pooling (kernel = 2)
Flatten
Reshape
LSTM (512 units)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (512 neurons)
Fully Connected (13 neurons)
INPUT
(vector of 36 features)
Batch Normalization
LSTM (128 units)
LSTM (128 units)
Fully Connected (128 neurons)
Fully Connected (13 neurons)
Concatenate
Fully Connected (512 neurons)
Fully Connected (32 neurons)
Dropout (0.2 rate)
Fully Connected (13 neurons)
OUTPUT
(vector of 13 probabilities)

Figure 3.11

3.5.1 Use of context data to refine the classifier predictions

The activity recognition classifier can predict a wrong activity, that is not consistent with the context surrounding the user. To mitigate this problem, the Context Refinement module applies knowledge-based reasoning on context data in order to exclude from the probability distribution resulting from the activity recognition algorithm those activities which are not context-consistent. After this operation, the probability distribution is re-normalized.

In this work, three approaches were developed and combined:

- **Micro-areas (M)**
- **Sensors (S)**
- **Posture (P)**

Using the Micro-areas approach, the probability related to an activity is discarded if the user is not in one of the semantic areas in which the activity can be performed (WASHING_DISHES in KITCHEN for example). This positioning information is known a priori by the system and will be illustrated in Figure 5.2. Furthermore, for some activities the positioning information used is based on the semantic area in which are placed environmental sensors: for example, a user can perform WATCHING_TV only if he is in the same semantic area in which is placed the television smart-plug.

With the Sensors approach, the status of the sensors is used to perform the context refinement. For example, a user can perform EATING if he is sitting on a chair or can perform WATCHING_TV if the television smart-plug is active. Such reasoning can't be used for COOKING and the cooker smart-plug, because not necessarily a user stop to perform this action at the same moment in which the sensor is turned off.

Using the Posture approach, the activities probabilities are discarded based on the users' posture. For example, a user can't perform WASHING_DISHES if he is sitting and can't perform EATING if he is not sitting.

3.5.2 Use of context data as features

Another possible usage of context data consists of providing this information to the activity recognition classifier directly as input together with the environmental sensors handcrafted features. Even in this case, different approaches were tested and combined.

- **Micro-areas (M)**: a new feature value is added generating a digit that identifies one of the smart-home areas. This value is then mapped in a number between 0 and 1.
- **Posture (P)**: a new feature value is added generating a digit that identifies the sitting posture (1) and the not_sitting one (0).

The comparison between the results obtained by these approaches and the ones described in Section 3.5.1 are illustrated in Section 5.3.3.

Chapter 4

Semi-supervised extension of the system

The system described in Chapter 2 has been extended with semi-supervised learning techniques. The idea is to train the activity recognition algorithm with a set of examples which is relatively small. In this way, it is possible to lighten the dataset collection and annotation operations. Through the active learning technique is then possible to label new data in real-time thanks to the directly interaction with the users. Incremental machine learning algorithms do not need to be trained receiving as input the whole dataset at once, but they can be updated over time with these newly labeled examples. This is important because retrain a classifier on the whole dataset to which are added new instances would be inconvenient. Furthermore, the active learning technique enables the system to adapt itself to the users' behavior. This happens because, when the system is uncertain about its prediction, it can ask the user related to this uncertainty which is the activity he is performing between the two most likely ones. The feedback of the user is then used to update the activity recognition classifier.

Section 4.1 illustrates how the system architecture was extended to introduce semi-supervised learning techniques. Section 4.2 instead describes how such a solution was implemented.

4.1 Architecture extension

To extend the system with semi-supervised learning techniques, it was necessary to introduce a new module called Prediction Confidence Evaluation (PCE). In Figure 4.1 it is possible to notice how the architecture described in Chapter 2 has been extended. Once the Context Refinement module refines the predictions emitted by the activity recognition classifier, the PCE module has to evaluate the uncertainty level of this inference. This is measured through the entropy value related to the probabilities vector generated as output by the Context Refinement module. This vector contains the probabilities related to the activities detectable by the system after the not context-consistent ones were discarded. If the system is uncertain between two possible activities, then it is explicitly asked to the user which of those is the correct one through an Android application developed for tablet in parallel works [24, 25]. The user can also answer through the graphic interface that none of the two activities is the correct one. This technique that allows labeling new data in real-time based on the users' feedbacks is called active learning. Since the deep learning classifiers are incremental algorithms, then it is possible to update the neural networks through these new annotated instances.

The refinement of the predictions emitted by the Activity Recognition module before evaluating the uncertainty of the system allows reducing the number of interactions with the users. This result is demonstrated in Section 5.4.3. This happens because the refinement based on context data avoids asking something to the users when at least one of the two most probable activities is logically impossible. For example, if the classifier is uncertain about the activities WATCHING_TV and COOKING, then it is easy to detect if one of them is logically impossible based on context data such as the position of the user inside the smart-home. So, if the user is in the KITCHEN area, then the activity WATCHING_TV will be discarded because the television smart-plug is in the LIVING_ROOM area, while the activity COOKING will be maintained in the probabilities vector since the cooker is in the KITCHEN area, too. Discarding one activity, after the normalization of the probabilities vector, all the other activities will be more likely. In this case, the probability of the COOKING activity will increase and is less probable that the system will remain uncertain about the action performed by the user.

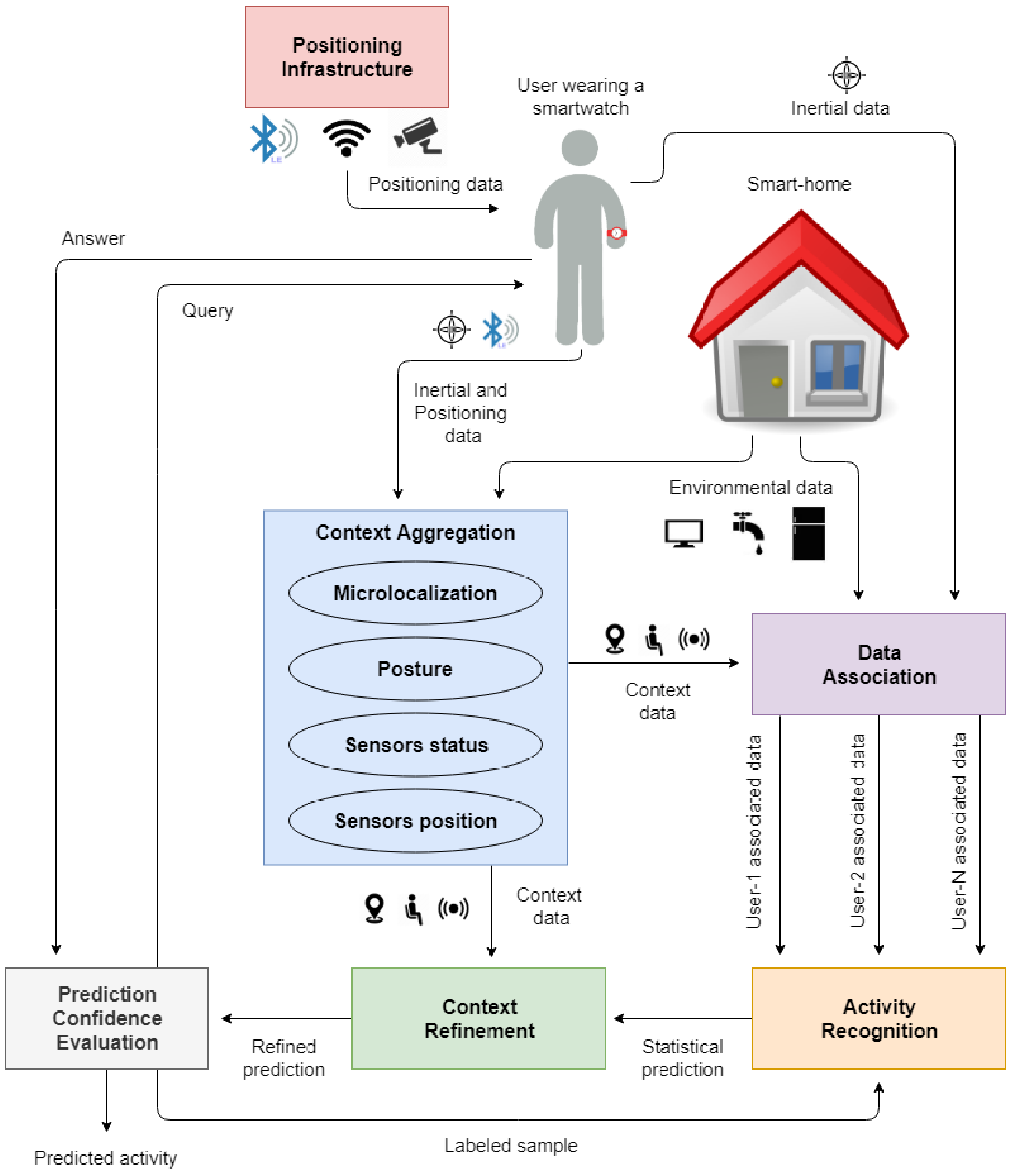


Figure 4.1: Extended system architecture.

In this work, we also introduced into the system the self-learning paradigm. With this technique, when the system is very confident about its inference, the feature vector related to the predicted activity can be used to update the incremental classifier, using the prediction itself as label. Even in this case, the predictions context-refinement is applied before passing the probabilities vector emitted as output by the activity recognition classifier to the PCE module. In this way, the uncertainty of the system should decrease thanks to the elimination of probabilities related to logically impossible activities. Unfortunately, as already happened in another mentioned work [11], this approach doesn't improve the performances of the system. Such a result will be described in detail in Section 5.4.

4.2 Implementation

To extend the system architecture with semi-supervised learning techniques, it is necessary to add to the system the new module called PCE that will work on the refined prediction emitted by the Context Refinement module. This means that the operations applied by the other modules of the system don't change with this extension. In this work, two methods were developed and evaluated: active learning and self-learning. As already mentioned, thanks to the use of incremental algorithms, it is possible to update the activity recognition classifier with data labeled with these techniques.

When the Context Refinement module refines the probabilities vector predicted by the activity recognition classifier, then the PCE module computes the entropy value of this vector using the python library **scipy.stats**. The entropy value of a probability distribution can be used to measure the uncertainty of the distribution itself. It measures the heterogeneity of the values contained in the distribution. The uncertainty of the prediction will be lower when the entropy value related to the probabilities vector tends to zero. This happens when, inside the vector, only one probability has a value that tends to 1, while the other probabilities tend to zero. The uncertainty of the prediction will be higher when the entropy value is greater than zero. This happens when inside the vector different probabilities have similar values greater than zero.

The level of uncertainty of a prediction can be used to determine whether a semi-supervised technique could be applied. If the entropy value is lower than a fixed threshold σ, then the prediction is considered certain and the PCE module applies the self-learning method: the feature vector that generated the prediction is labeled using the most likely activity contained in the vector emitted by the Context Refinement module. This newly annotated example can be used to update the incremental activity recognition classifier. It is important to note that computing the entropy of the vector after the context refinement operation should reduce the uncertainty of the inference. If the entropy value is greater than a fixed threshold α, then the prediction is considered uncertain and active learning is performed. Through the Android application developed for tablet, it is asked to the user which is between the two most likely activities the correct one. The user can also affirm that none of them is the activity he was performing. If the user answers with one of the two proposed activities, then his feedback is used to label the feature vector related to the statistical inference. Then, this annotated example can be used to update the activity recognition algorithm. Even in this case, it is useful to apply the context refinement before the uncertainty evaluation performed by the PCE module. This choice allows reducing the number of interactions between the system and the users. This result will be shown in Section 5.4.3.

Once a feature vector is labeled thanks to one of the previously described techniques, it is possible to decide whether to update immediately the activity recognition classifier with it or to add it to a batch that will be filled with other examples. In the second case, when the batch reaches a certain size, it can be used to update the machine learning algorithm. Updating the classifier with a single example at a time allows improving faster the inference performances related to the activity of that instance. On the other hand, the neural network will modify its weights based on the example of a single activity and this means that there could be a worsening of the performances regarding the other activities. If the classifier is updated with a batch that contains examples of different activities, then the just described issue should be avoided. On the other hand, the performance improvement will be slower since the neural network has to update its weights considering examples related to different detectable activities. The evaluation of these methods is described in Section 5.4.

Chapter 5

Experimental evaluation

In this chapter the results of the experimental evaluation performed during this work are presented. Sections 5.1 and 5.2 respectively describe the dataset and the metrics used to experimentally evaluate the system. Sections 5.3 reports the multi-inhabitant classifier evaluation, by comparing the results varying the type of neural network and the context data used to perform the data association and the context refinement. Section 5.4 describes the evaluation of the semi-supervised extension of the system.

5.1 Considered dataset

To evaluate the system we used a dataset already collected in past works at Every-Ware Lab, whose openspace was used to simulate a smart-home environment. The lab was divided into 6 semantic areas, each representing a different room of the habitation (hall, kitchen, dining room, medicine room, living room and office). Different environmental sensors were installed to monitor the interaction of the users with their surroundings: 14 BLE beacons, 5 magnetic sensors, 9 pressure sensors and 2 smart-plugs. Furthermore, incoming and outgoing mobile calls necessary to perform some high-level activities were simulated through a dedicated Android application. In Figure 5.1 it is shown how the smart-home was divided into semantic areas and how environmental sensors and beacons were placed during the dataset acquisition. During the acquisition, 12 different subjects were involved. Both single- and multi-inhabitants scenarios were simulated, recording activities performed at the same time by one, two or four subjects. Before the acquisition, the users were instructed about the sequence of activities that they had to perform. To increase the dataset variability, the users were free to execute each activity in their own way. Overall, the dataset was built through the acquisition of 12 scenarios in a single-inhabitant setting, 10 scenarios with 2 subjects inside the habitation and 10 scenarios with 4 subjects involved. During the deployment of the system in a real-time setting, we realized that in these scenarios the posture of the users was never useful to correctly perform the data association problem. In particular, when a user is sitting on a pressure mat in a specific smart-home area performing a certain activity, all the other users present in the same area are sitting on other pressure mats and are performing the same action. For this reason, in the experimental results the posture context information seems to be useless, but in Chapter 6 we'll see that in other multi-inhabitant scenarios it is essential. The activities collected in the dataset are the following:

- ANSWERING_PHONE: the user takes a smartphone from his pocket and presses a button that generates an event associated with an incoming call. Finally, he puts the smartphone to his ear

- CLEARING_TABLE: the user collects plates, glasses and cutlery from the table, puts the bottles in the fridge and takes the tablecloth

- COOKING: the user can warm the milk or cook pasta. In the first case, he takes a pot from the dedicated drawer, he fills it with milk taken from the fridge and warm through the cooker. In the second case, he takes a pot from the dedicated drawer, he fills it with tap water warm through the cooker. In a second moment, salt and pasta taken from the pantry are added. The user waits for the pasta cooking and then he can drain the pasta and eventually dress it with condiments taken from the fridge

- EATING: the user sits on a DINING_ROOM chair and simulates to eat what he cooked

- GETTING_IN: the user enters in the smart-home, takes his coat off and hangs it on the coat hanger

- GETTING_OUT: the user puts on the coat taken from the coat hanger and then he leaves the smart-home
- MAKING_PHONE_CALL: the user takes a smartphone from his pocket and simulates the composition of a telephone number. Then he presses a button that generates an event associated with an outgoing call. Finally, he puts the smartphone to his ear
- PREPARING_COLD_MEAL: the user can prepare a sandwich or a salad. In the first case, he takes some bread from the pantry, he cuts it with a knife taken from the dedicated drawer and then he fills it with food taken from the fridge. In the second case, he puts some salad from the fridge into a bowl. Oil and salt taken from the pantry are added to the salad, which is then mixed with cutlery taken from the dedicated drawer
- SETTING_UP_TABLE: the user puts the tablecloth on the table, then he puts on it also plates, glasses, napkins, cutlery and one or more bottles taken from the fridge
- TAKING_MEDICINES: the user takes medicine from the dedicated drawer and assumes it. Eventually, he can perform this action using a glass filled with water taken from the fridge
- USING_PC: the user sits on the OFFICE chair and simulates the usage of the computer writing on the keyboard and moving the mouse
- WASHING_DISHES: the user simulates to wash dishes with his hands
- WATCHING_TV: the user turns on the television and sits on a LIVING_ROOM chair

Figure 5.2 shows the semantic areas in which each activity could or couldn't be performed (respectively through numbers 1 and 0) by the users during the dataset acquisition. This modeling derives from common sense knowledge and it is exploited by the Context Refinement module to refine the inferences of the activity recognition classifier.

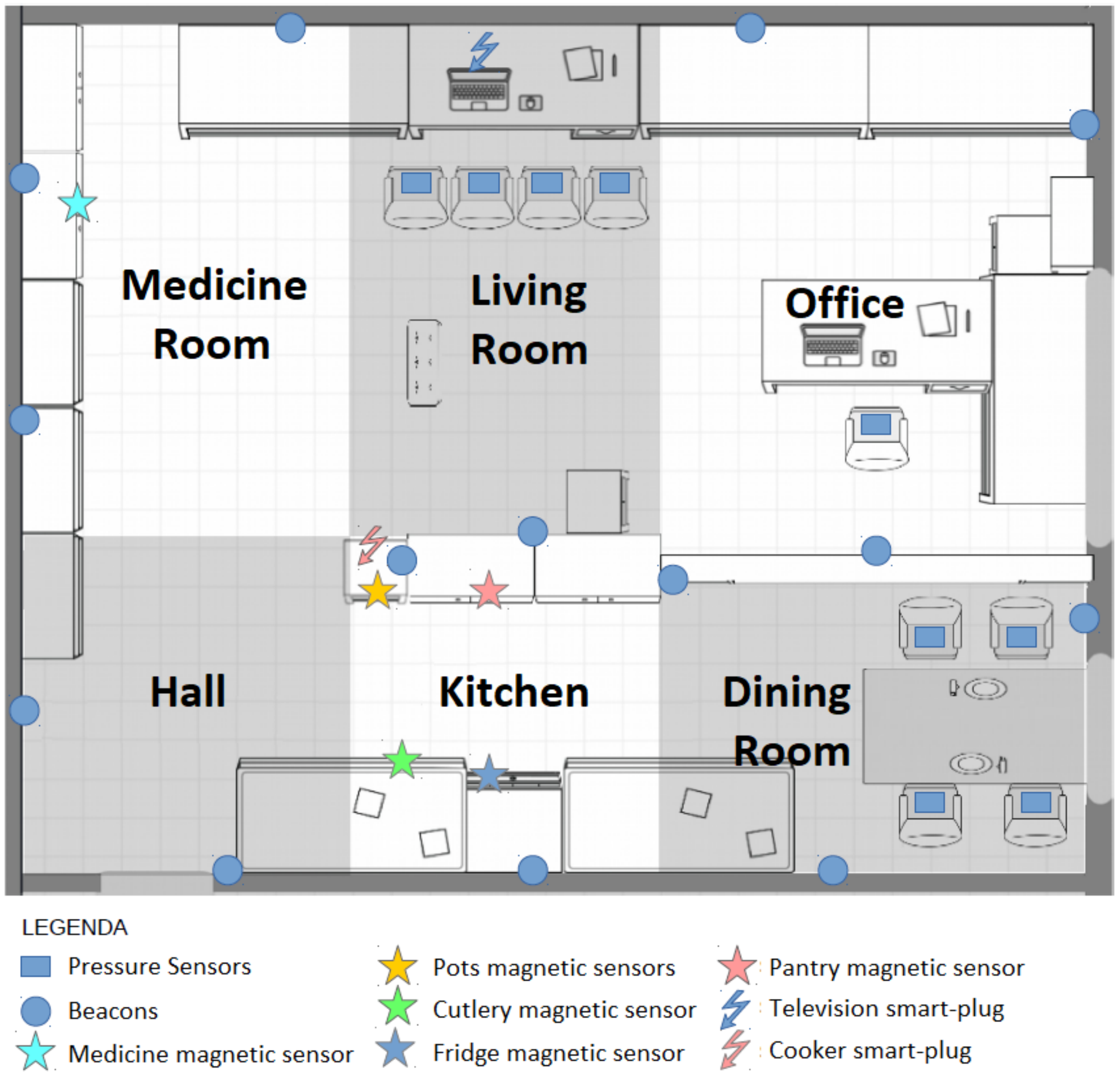
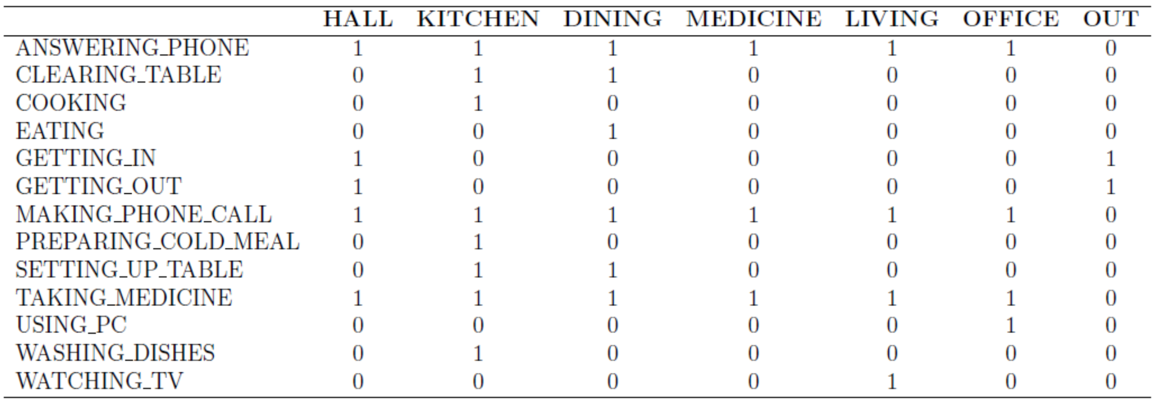


Figure 5.1: Division of the smart-home into semantic areas and position of the environmental sensors.



	HALL	KITCHEN	DINING	MEDICINE	LIVING	OFFICE	OUT
ANSWERING_PHONE	1	1	1	1	1	1	0
CLEARING_TABLE	0	1	1	0	0	0	0
COOKING	0	1	0	0	0	0	0
EATING	0	0	1	0	0	0	0
GETTING_IN	1	0	0	0	0	0	1
GETTING_OUT	1	0	0	0	0	0	1
MAKING_PHONE_CALL	1	1	1	1	1	1	0
PREPARING_COLD_MEAL	0	1	0	0	0	0	0
SETTING_UP_TABLE	0	1	1	0	0	0	0
TAKING_MEDICINE	1	1	1	1	1	1	0
USING_PC	0	0	0	0	0	1	0
WASHING_DISHES	0	1	0	0	0	0	0
WATCHING_TV	0	0	0	0	1	0	0

Figure 5.2: Semantic areas in which each dataset activity can (1) or cannot (0) be performed.

5.2 Metrics

To evaluate the predictions of a learning model, it is necessary to give it as input some examples and then compare the classifier's output with the ground truths of such examples. In this way, it is possible to compute:

- **TP (True Positive)**: how many times an activity is predicted in the right way
- **FP (False Positive)**: how many times an activity is predicted even if it wasn't performed by the user
- **FN (False Negative)**: how many times an activity is not predicted

Based on these values, it is possible to compute metrics which are well known in literature, such as:

- **Precision**: the percentage of an activity correct predictions on the total number of predictions which have the same activity as result. It is computed as

$$Precision = \frac{VP}{VP + FP} \tag{1}$$

- **Recall**: the percentage of an activity correct predictions related to the total number of predictions performed on examples of the same activity. It is computed as

$$Recall = \frac{VP}{VP + FN} \tag{2}$$

- **F1**: the harmonic mean of precision and recall, which allows to obtain a summary about the quality of the classifier's predictions. It is computed as

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

5.3 Multi-inhabitant classifier evaluation

The evaluation of the activity recognition classifier is computed through the leave-one-subject-out cross-validation technique. With this approach, shown in Figure 5.3, the dataset examples are divided based on the user who generated them. At each iteration of the process, data related to one of the users are used as test set, while the others are the training set used to train the classifier. At the end of each step, the mean of the activities f1 score is computed based on the predictions made by the classifier on the test set. At the end of the process, the mean of the f1 scores of each step is computed. This technique provides an evaluation of the quality of the predictions, but also on the generalization ability of the system, i.e. how much the system is effective when it receives as input examples of activities performed by users not considered during the training.

The best performances classifier obtained during the evaluation process has the following properties:

- as input, it receives handcrafted feature both for inertial and for environmental sensors, segmented into 14 seconds temporal windows
- to perform the data association and the context refinement, it uses only the context data provided by the Micro-localization module
- the deep neural network contains a combination of convolutional and recurrent layers

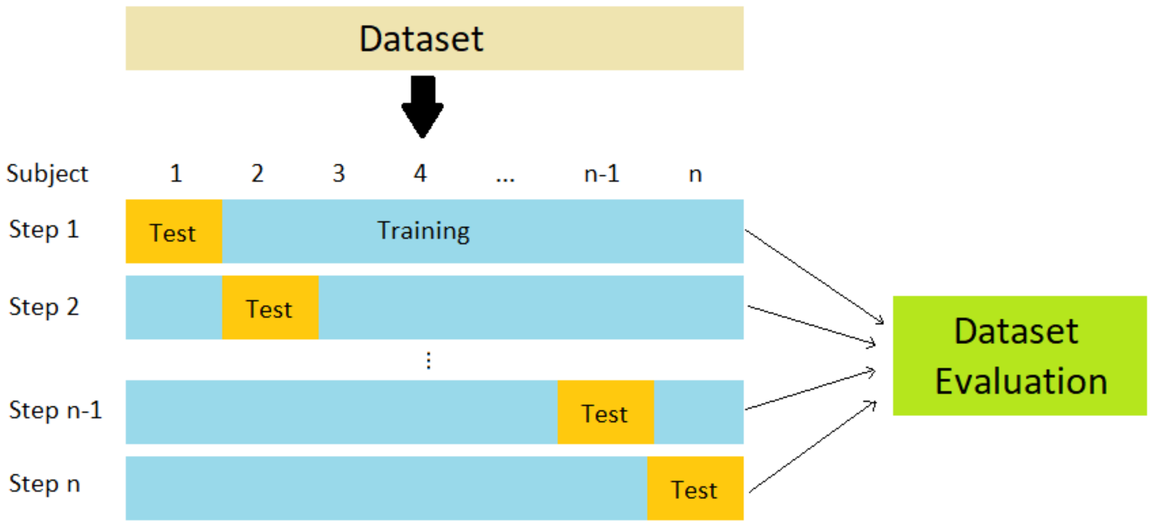


Figure 5.3: Example of leave-one-subject-out cross-validation.

We already mentioned why the posture information is useless with our dataset to perform the data association. In Section 5.3.3, it will be explained why this happens also for the context refinement process.

Figure 5.4 and Table 5.1 show respectively the confusion matrix and the f1 score values obtained by the most performing classifier. In the confusion matrix, the value in position *(i,j)* describes the percentage of times in which the *j* label was predicted while the ground truth was the *i* label. The confusion matrix allows making different remarks:

- ANSWERING_PHONE, EATING, MAKING_PHONE_CALL, USING_PC and WATCHING_TV: these activities tend to be detected without problems, probably because each of them is recognizable based on the environmental sensors with whom the user has to interact when he performs the action. For example, EATING could be easy to recognize, because it's the only activity which is performed in DINING_ROOM sitting on a pressure mat

- CLEARING_TABLE, COOKING, PREPARING_COLD_MEAL, SETTING_UP_TABLE e WASHING_DISHES: all of these activities can be performed inside the KITCHEN area and require the interaction of the user with some environmental sensors. Furthermore, some activities share the interaction with the same sensors (e.g., both preparing a cold meal and setting up the table require the opening of the fridge). Probably, these activities are confused because the system hasn't enough detailed context information to associate for sure the environmental events generated in the KITCHEN area when more users are there. So, it could happen that while a user is performing the action WASHING_DISHES the system associate to him the event related to the turning on of the cooker. In such a case, the system could infer the COOKING activity for this user

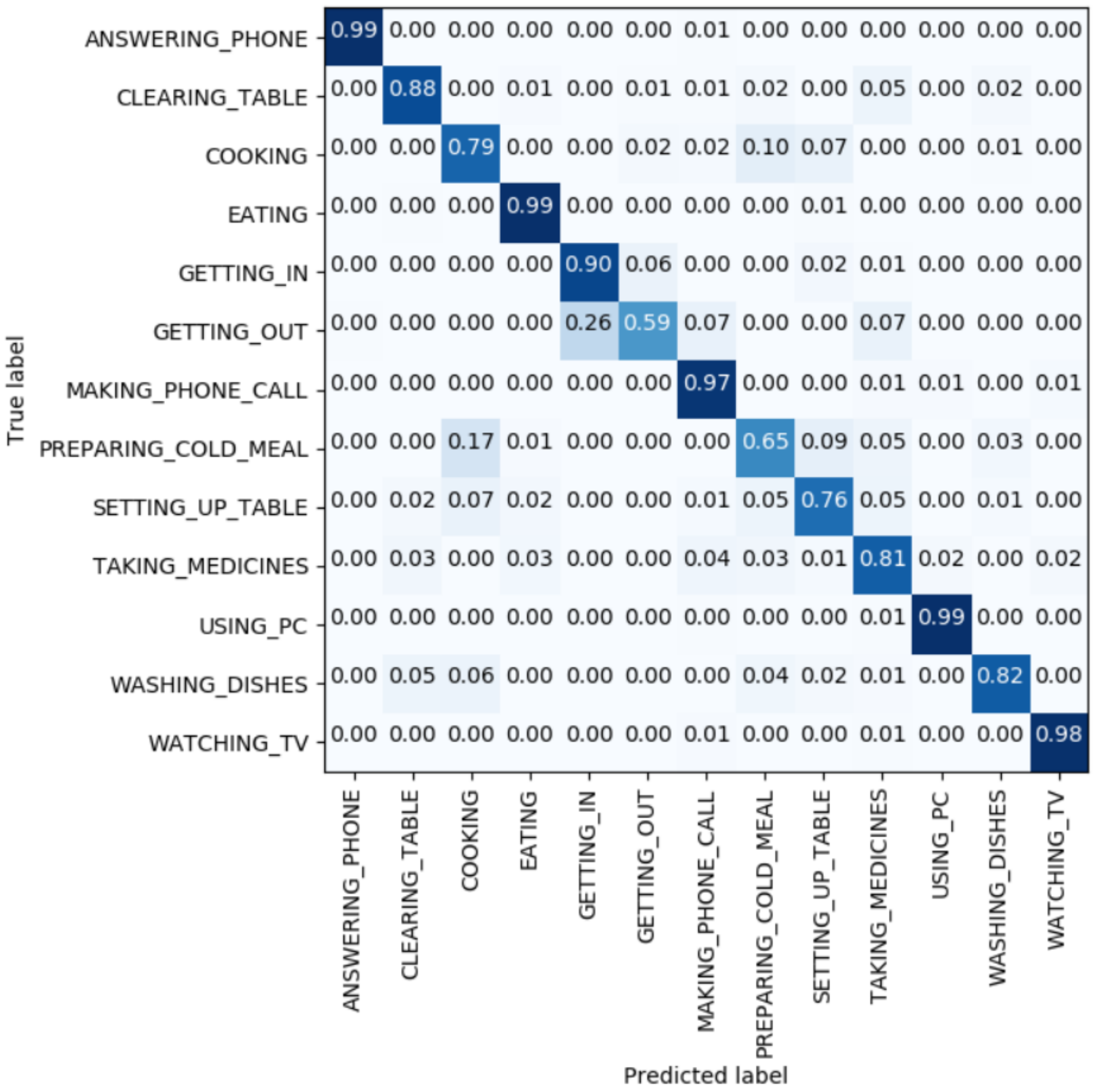
- GETTING_IN e GETTING_OUT: these activities don't require an interaction of the user with environmental sensors, but they have similar gestures and the system can confuse them

- TAKING_MEDICINES: this activity is confused with a low percentage with different dataset activities, probably because it is an action which can be performed in different semantic areas of the smart-home (e.g., sitting on a chair of the dining room or standing near the medicine cabinet of the medicine room)

From the previous analysis, we can notice that, generally, the system confuses activities such that context information is not sufficient to perform a certain data association of the environmental events characteristic for those activities (the cooker for COOKING, the fridge for SETTING_UP_TABLE or PREPARING_COLD_MEAL).

5.3.1 Posture classifier evaluation

To detect the users' posture, a low-level activities classifier was trained. It can detect two possible postures: SITTING and NOT_SITTING. To train this statistical model, the same dataset used to build the high-level activities classifier was considered. The only difference is that, in this case, only inertial data were used. Each feature vector was labeled based on its high-level activity annotation. EATING, USING_PC and WATCHING_TV labels were mapped into the SITTING one. The vectors related to the activities MAKING_PHONE_CALL and ANSWERING_PHONE were discarded since those activities can be performed both with a sitting and with a not sitting posture. The other labels were mapped into the NOT_SITTING annotation.



True label \ Predicted label	ANSWERING_PHONE	CLEARING_TABLE	COOKING	EATING	GETTING_IN	GETTING_OUT	MAKING_PHONE_CALL	PREPARING_COLD_MEAL	SETTING_UP_TABLE	TAKING_MEDICINES	USING_PC	WASHING_DISHES	WATCHING_TV
ANSWERING_PHONE	0.99	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
CLEARING_TABLE	0.00	0.88	0.00	0.01	0.00	0.01	0.01	0.02	0.00	0.05	0.00	0.02	0.00
COOKING	0.00	0.00	0.79	0.00	0.00	0.02	0.02	0.10	0.07	0.00	0.00	0.01	0.00
EATING	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
GETTING_IN	0.00	0.00	0.00	0.00	0.90	0.06	0.00	0.00	0.02	0.01	0.00	0.00	0.00
GETTING_OUT	0.00	0.00	0.00	0.00	0.26	0.59	0.07	0.00	0.00	0.07	0.00	0.00	0.00
MAKING_PHONE_CALL	0.00	0.00	0.00	0.00	0.00	0.00	0.97	0.00	0.00	0.01	0.01	0.00	0.01
PREPARING_COLD_MEAL	0.00	0.00	0.17	0.01	0.00	0.00	0.00	0.65	0.09	0.05	0.00	0.03	0.00
SETTING_UP_TABLE	0.00	0.02	0.07	0.02	0.00	0.00	0.01	0.05	0.76	0.05	0.00	0.01	0.00
TAKING_MEDICINES	0.00	0.03	0.00	0.03	0.00	0.00	0.04	0.03	0.01	0.81	0.02	0.00	0.02
USING_PC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.99	0.00	0.00
WASHING_DISHES	0.00	0.05	0.06	0.00	0.00	0.00	0.00	0.04	0.02	0.01	0.00	0.82	0.00
WATCHING_TV	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.98

Figure 5.4: Confusion matrix of the activities obtained by the most performing classifier.

	precision	**recall**	**f1-score**	**support**
ANSWERING_PHONE	0.99	0.99	0.99	1100
CLEARING_TABLE	0.86	0.88	0.86	554
COOKING	0.83	0.79	0.79	996
EATING	0.99	0.99	0.99	2897
GETTING_IN	0.83	0.90	0.86	289
GETTING_OUT	0.82	0.59	0.67	212
MAKING_PHONE_CALL	0.90	0.97	0.93	1034
PREPARING_COLD_MEAL	0.71	0.65	0.66	640
SETTING_UP_TABLE	0.77	0.76	0.75	674
TAKING_MEDICINES	0.72	0.81	0.74	443
USING_PC	0.99	0.99	0.99	1681
WASHING_DISHES	0.95	0.82	0.88	994
WATCHING_TV	1.00	0.98	0.99	4475
avg / total	0.94	0.92	0.93	15989

Table 5.1: F1 scores obtained by the most performing classifier.

Figure 5.5 and Table 5.2 show respectively the confusion matrix and the f1 score values obtained by this low-level classifier, using data segmented into 16 seconds window size. We can notice that in the 11% of cases, the posture NOT_SITTING is confused with the SITTING one. This is mainly due to the following problem: very static activities as COOKING (that during its execution requires to wait until the water boils) sometimes are confused with WATCHING_TV by the low-level classifier. In such cases, the wrong detected posture will be SITTING, when instead it should be NOT_SITTING. To solve this problem, it should be necessary to train the low-level classifier giving it also examples labeled as STILL_STANDING, i.e. activities in which the user is still standing and not walking. This should reduce the number of times in which standing and static activities as COOKING bring the low-level classifier to predict SITTING. Another problem is that sometimes the low-level classifier confuses TAKING_MEDICINES with EATING, because of their common gesture of drinking from a glass of water. Also in these cases the detected posture will be SITTING instead of NOT_SITTING. Another possible solution to this problem will be explained in Chapter 7.

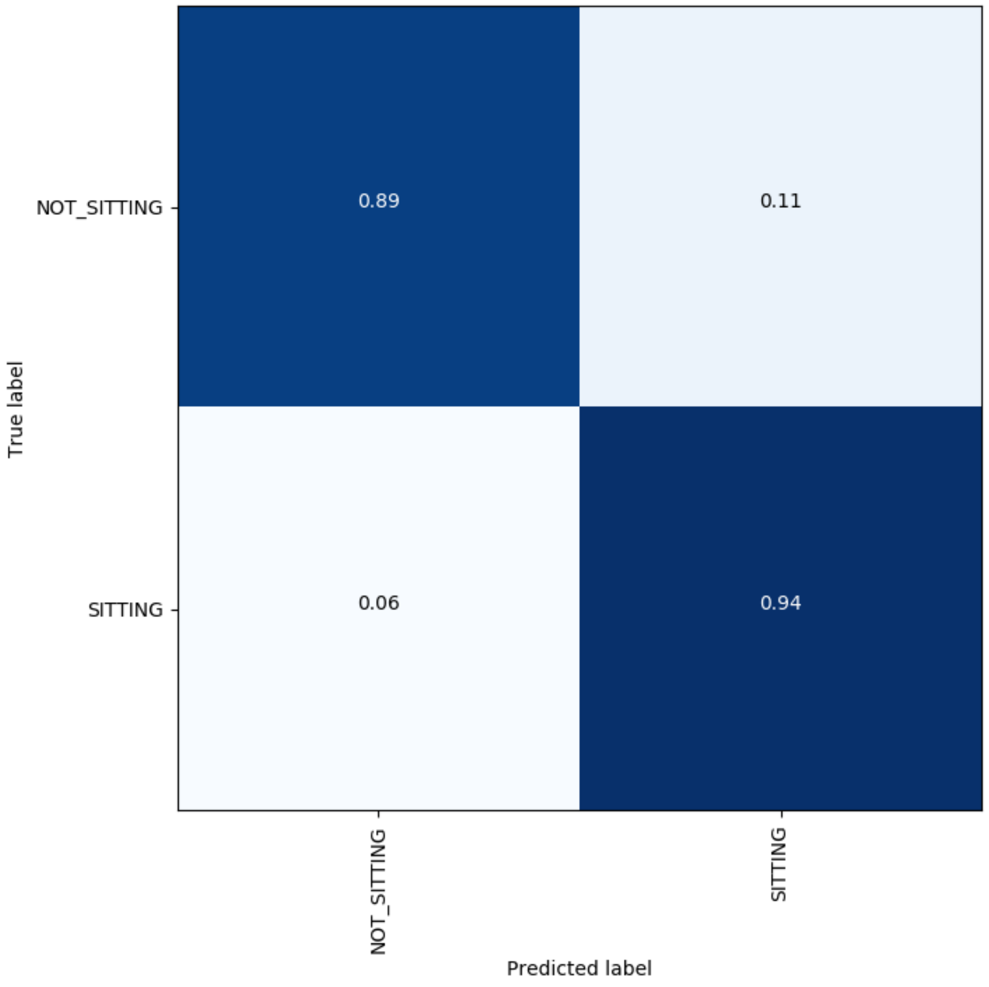


Figure 5.5: Confusion matrix of the postures obtained by the low-level classifier.

	precision	**recall**	**f1-score**	**support**
NOT_SITTING	0.89	0.89	0.89	4150
SITTING	0.94	0.94	0.94	7796
avg / total	0.92	0.92	0.92	11946

Table 5.2: F1 scores obtained by the low-level classifier.

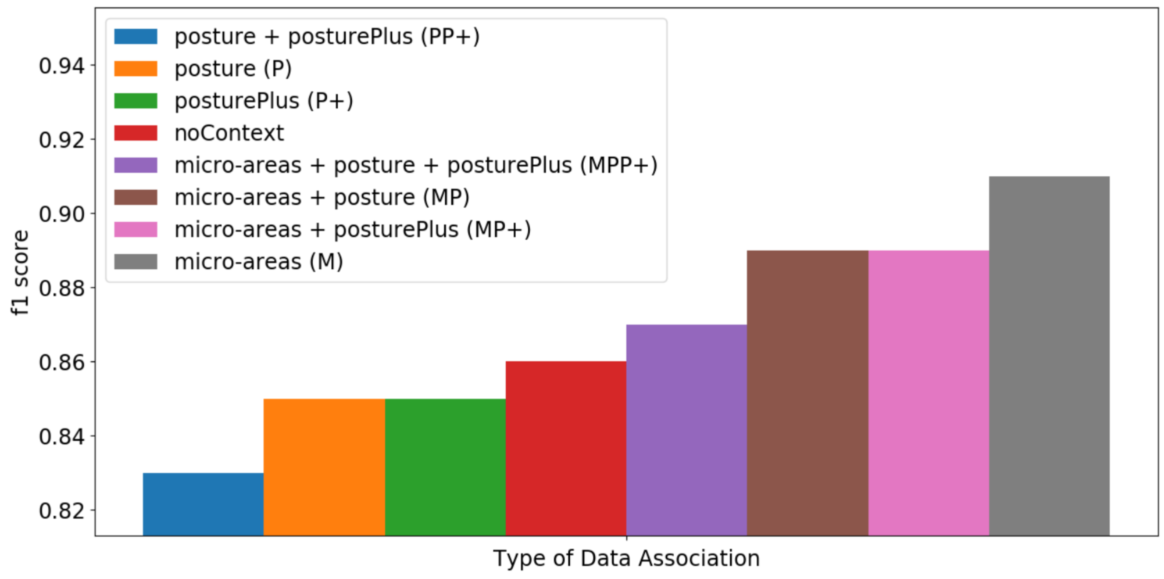


Figure 5.6: Comparison between different types of context information used to perform the data association.

5.3.2 Data association evaluation

This section analyzes the impact of context information to perform the data association. The different types of context information described in Section 3.2.3 were considered and combined. Figure 5.6 shows the f1 scores obtained by the classifier using different combinations of context information in order to perform the data association. It is possible to notice that the best performances are obtained using only the information provided by the Micro-localization module. This means that the usage of the posture worsens the results. We already mentioned that the scenarios used to collect the dataset do not contain examples in which the user posture is useful to correctly perform the data association. This leads to the result that posture information doesn't improve the performances of the system on this dataset. Furthermore, the classification errors of the low-level classifier generates wrong feature vectors during the data association process. These errors will be propagated to the activity recognition task, worsening the results.

Figure 5.7 compares the most promising result with hypothetical results, which can be obtained with a precisely detected posture of the users or with a perfect data association. It's important to note that, the perfect data association suffers

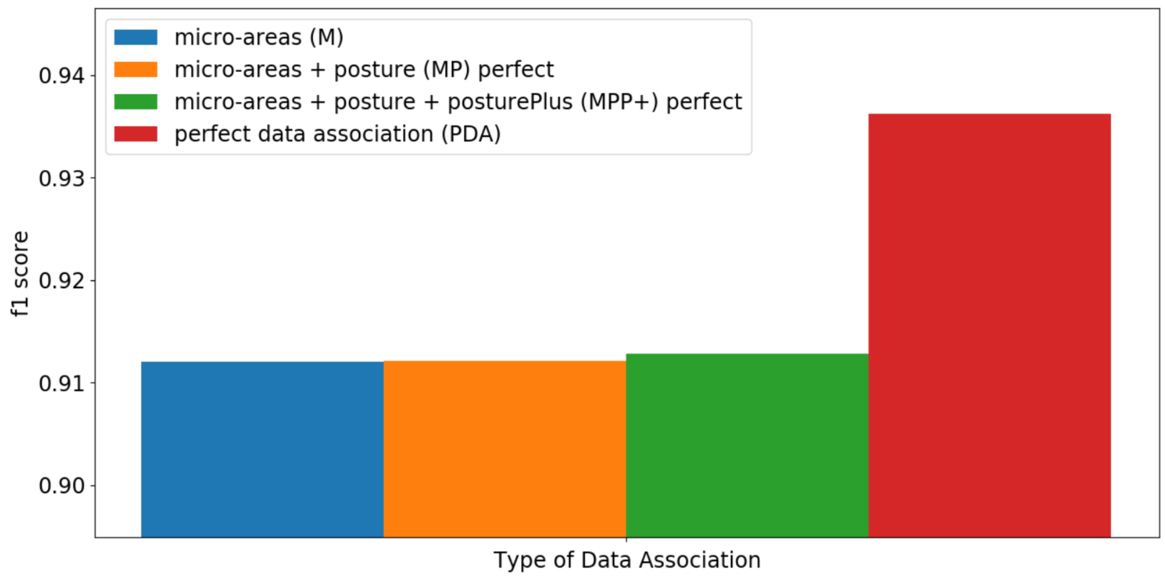


Figure 5.7: Comparison between the most promising results related to data association using context information.

from the following problem: when a user activates a sensor which is then disabled by another user, then the feature vectors related to the first user can be wrong, as explained in Section 3.2.3. The figure shows that the posture, even if perfect, doesn't improve the performances of the data association process obtained using only the information provided by the Micro-localization module. This happens for the previously discussed reasons. Furthermore, the bar related to the Perfect Data Association (PDA) represents the f1 score obtained using a perfect assignment of the environmental events to the users.

5.3.3 Context refinement evaluation

This section analyzes the usage of context data to perform the context refinement. The analysis considers all the possible combinations between the different types of data association with the different types of context refinement. The three types of context refinement approaches already described in Section 3.5.1 were considered and combined. Figure 5.8 shows the f1 scores obtained with the Perfect Data Association varying the context information used to perform the context refinement. It is possible to notice that the combination of all the context data (with perfect information

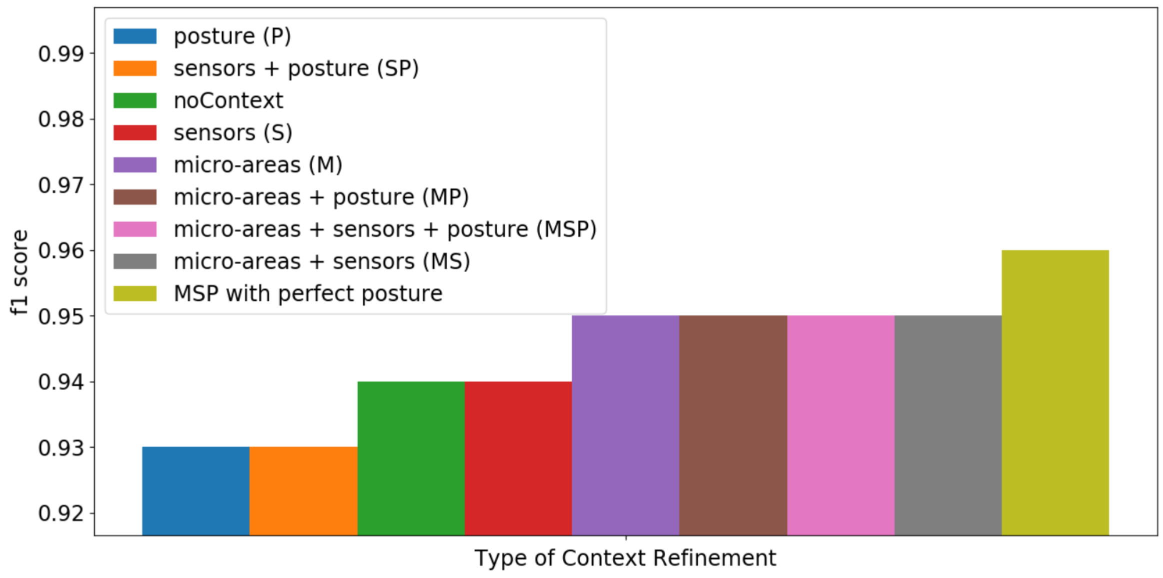


Figure 5.8: Comparison between different types of context information used to perform the context refinement, combining them with the Perfect Data Association.

about the posture of the users) could bring to the best results. Obviously, these results are hypothetical because they use perfect data about the assignment of the environmental sensors and the users' posture. Anyway, this allows us to understand that, in the future, an improvement of the data association and the detection of the users' posture can allow the system to reach better performances.

The Figures 5.9 and 5.10 show the use of different types of context information in order to perform the context refinement, combining them respectively with several combinations of context information for data association. Since the best results concern the data association performed using the micro-areas approach, we can focus on Figure 5.11, which is only about these results. We can notice that the best not-hypothetical result is obtained using only context data about the micro-localization both for the data association and the context refinement. Combine this context source with the one regarding the environmental sensors' status doesn't change much the overall f1 score. This probably happens because of cases in which there are wrong environmental sensors associations: for example, if the television smart-plug active status is not assigned to the user who is performing WATCHING_TV, then the context refinement will remove that activity from the possible results of the system.

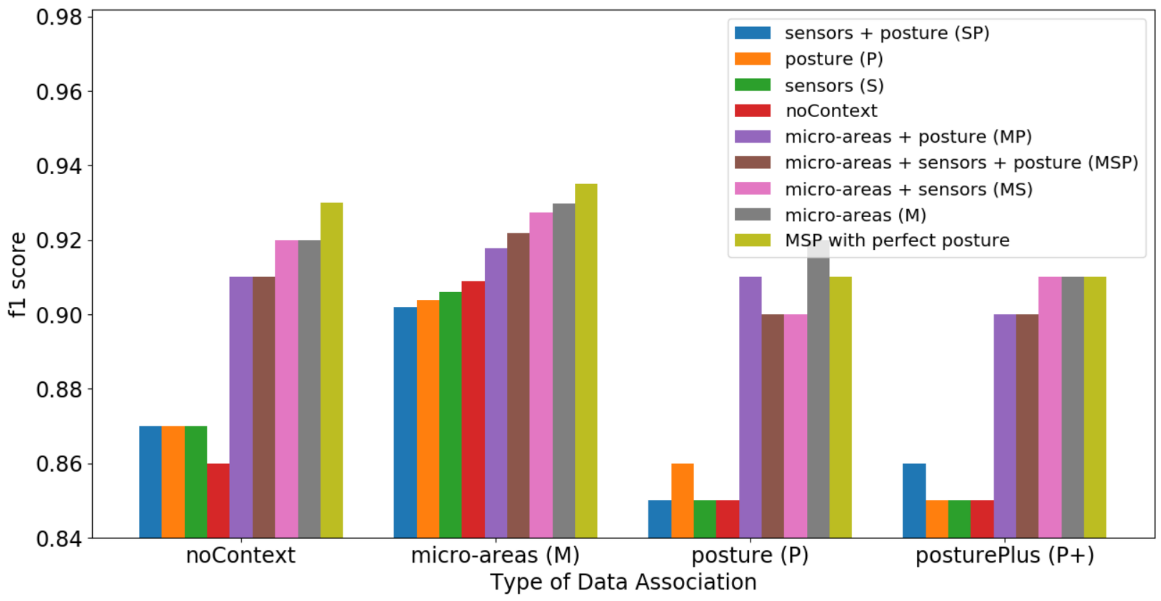


Figure 5.9: Comparison between different types of context information used to perform the context refinement, combining them with individual context approaches for data association.

Furthermore, the addition of posture information generates a deterioration of the results for the same reasons explained in Section 5.3.1. Finally, also in this case, a perfect information about the posture of the users could help the system to improve its performances.

Figure 5.12 compares the most promising result with hypothetical results, which can be obtained with combinations of a perfect detected users' posture and a perfect data association. The best not-hypothetical result uses only the context information provided by the Micro-localization module both for the data association and the context refinement. We already noticed that with perfect information about the users' posture, this result would be more positive. The last bar of the figure shows the best f1 score value which we can reach considering the Perfect Data Association and all the available context data (including the information about the perfect users' posture) to perform the context refinement. We can notice how the proposed solution achieves results that are not so far from the best performances that can be hypothetically reached on our dataset. Furthermore, these results can be improved with the solutions that will be proposed in Chapter 7.

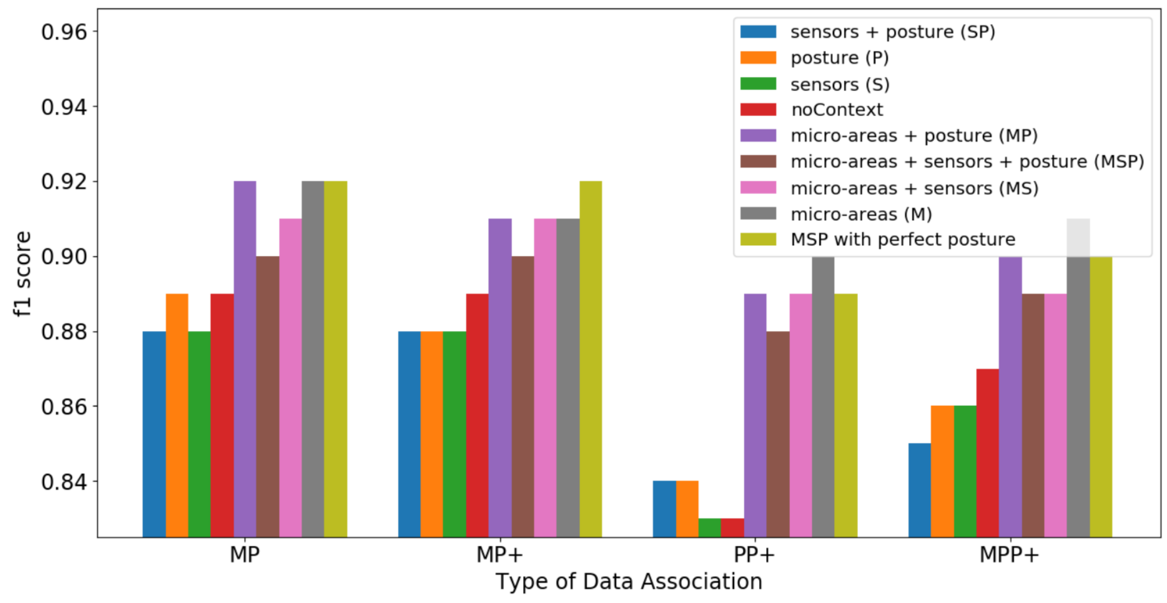


Figure 5.10: Comparison between different types of context information used to perform the context refinement, combining them with hybrid context approaches for data association.

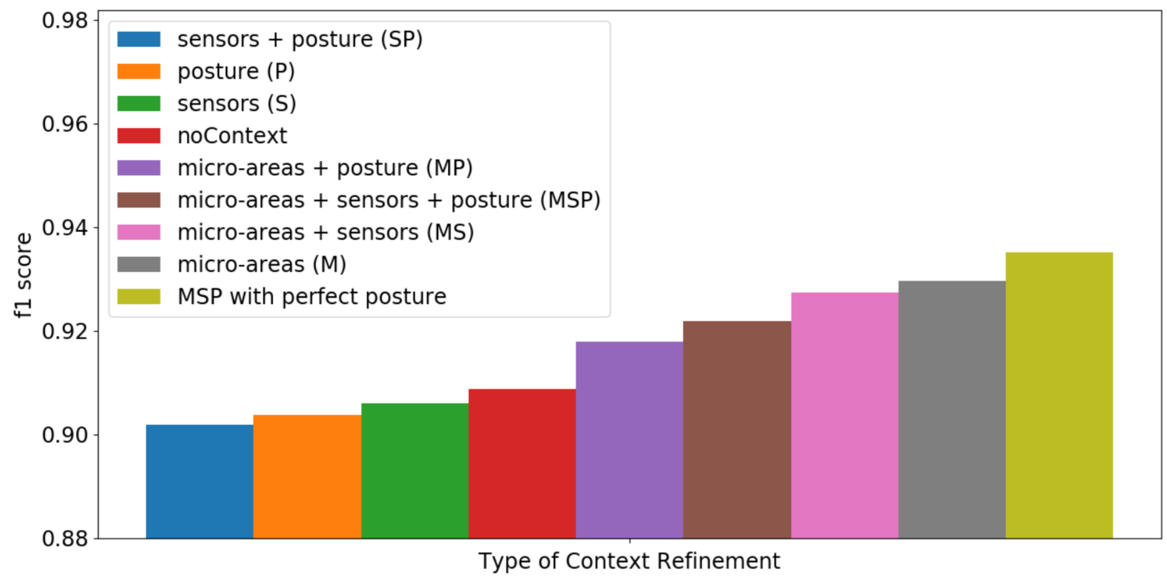


Figure 5.11: Comparison between different types of context information used to perform the context refinement, combining them with the micro-areas context approach for data association.

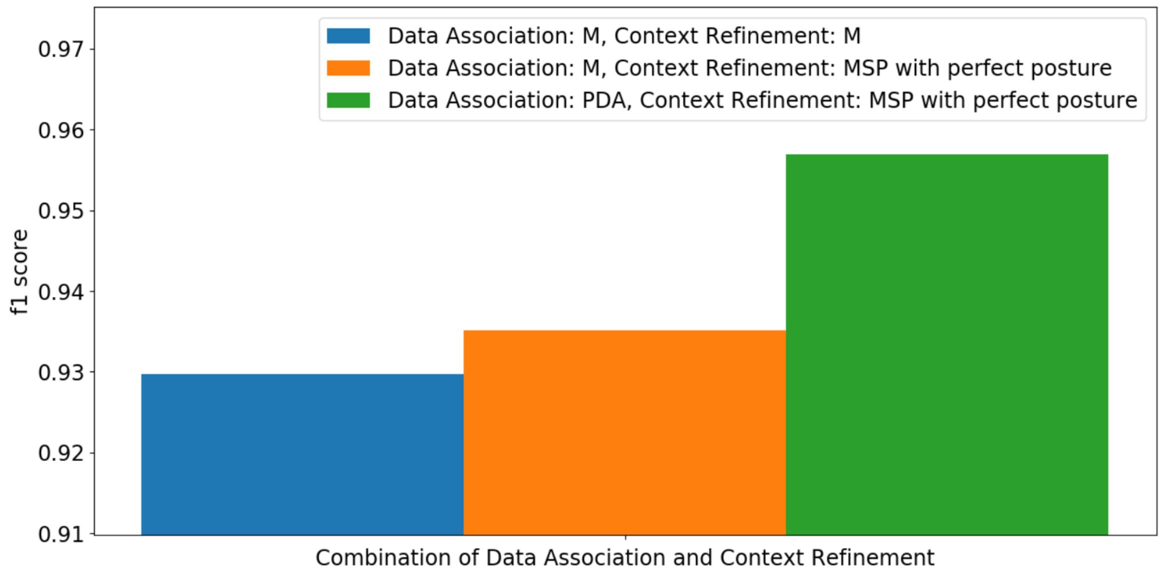


Figure 5.12: Comparison between the most promising results related to the combination of data association and context refinement.

5.3.4 Deep learning evaluation using handcrafted features

In this section different deep learning algorithms are analyzed using handcrafted features and the best not-hypothetical combination of data association and context refinement described in the previous sections. So, both the data association and the context refinement are obtained using only the information related to the users' position inside the smart-home. Four different types of deep learning models (already described in Section 3.4) were evaluated varying the data segmentation windows size:

- **FC Layers**: a model which contains only fully connected layers
- **CNN + FC Layers**: a model which contains convolutive layers followed by fully connected layers
- **LSTM**: a model which contains only LSTM recurrent layers
- **CNN + LSTM**: a model which contains convolutive layers followed by LSTM recurrent layers

Figure 5.13 shows the results of this analysis. We can notice that the CNN + LSTM model achieves a little improvement with respect to the other models, allowing also

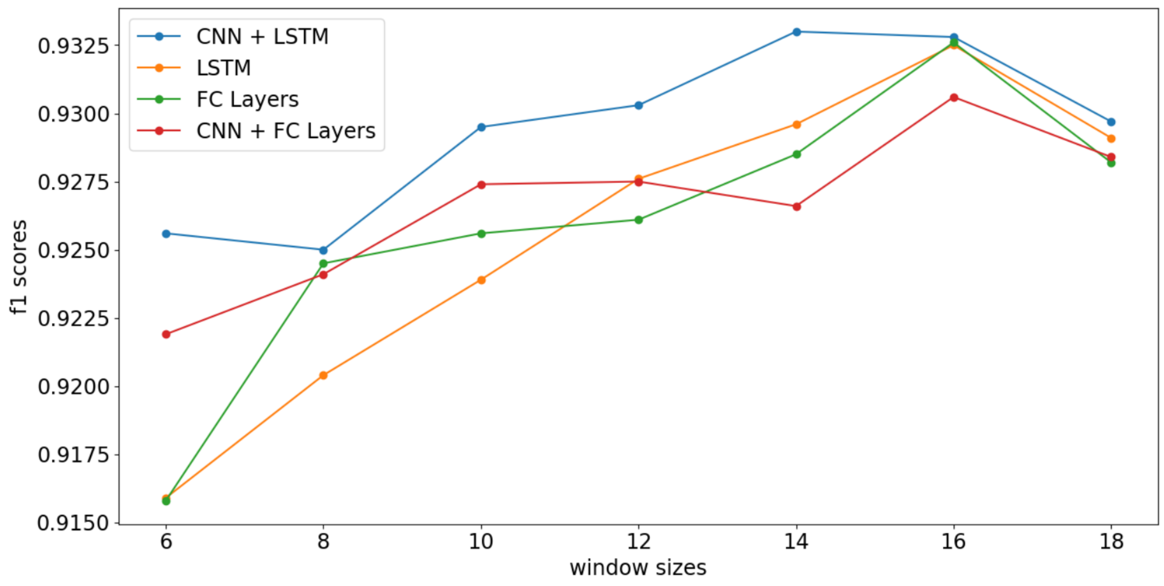


Figure 5.13: F1 score values related to the four neural networks tested with hand-crafted features, varying the data segmentation window size.

the use of a shorter window size. Furthermore, the models using convolutive layers seem to have better performances when a small window size is considered with respect to the algorithms which don't use this type of layers.

The performances of the CNN + LSTM model which uses 14 seconds segmentation windows are shown in Figure 5.4 and in Table 5.1. The explaination of such results can be found at the beginning of the Section 5.3.

Context data as features Figure 5.14 shows the comparison between the f1 score values obtained using context data for the context refinement or/and directly during the classifier training. We can notice that, also giving the context information directly in the feature vectors, the use of only micro-areas data allows obtaining better results rather than using also users posture information. Anyway, using context data as features, the f1 score values are increased of a couple of percentage points.

On the other hand, the context refinement leads to even better results. The last two bars of the figure show the difference between using context data both for context refinement and as features and using them only to perform the context refinement. This last option allows the model to achieve the best performances. Furthermore,

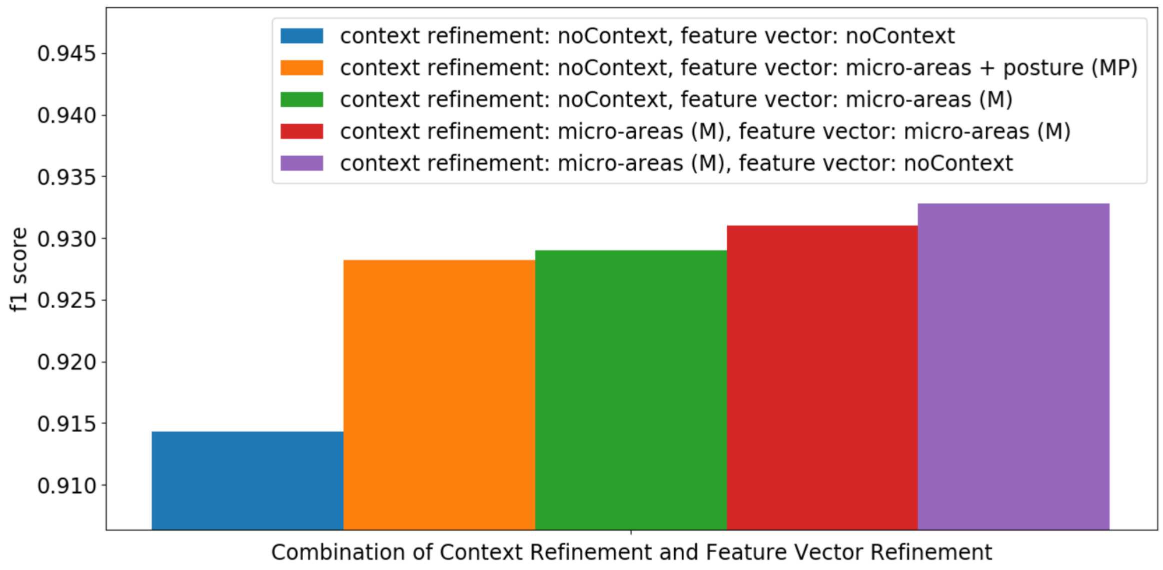


Figure 5.14: Comparison between f1 score values obtained using context data for the context refinement or/and directly during the training of the activity recognition model.

the use of context data as features has an intrinsic scalability problem: the growth of the number of context sources will make the activity recognition classifier ever more complex. Instead, the refinement of the statistical predictions will be always applied on a recognition algorithm trained only on inertial and environmental data.

5.3.5 Deep learning evaluation using raw data

It is important to clarify that the size of the windows used to segment data will affect the activity recognition classifiers' input shape. We have already mentioned that the inertial data will be given as input through a matrix that depends on a variable called *samples_num*. This variable value indicates the number of inertial measurements collected inside a single segmentation window. We have to consider that reducing the window size, also the *samples_num* value will be decreased. So, using the same neural network structure, varying the window size will affect the relative complexity of the model. For example, if the first FC layer of the classifier contains 512 neurons, the complexity of the model will be relatively greater when are used segmentation windows of 14 seconds instead of 16 seconds ones. This happens since in the first case

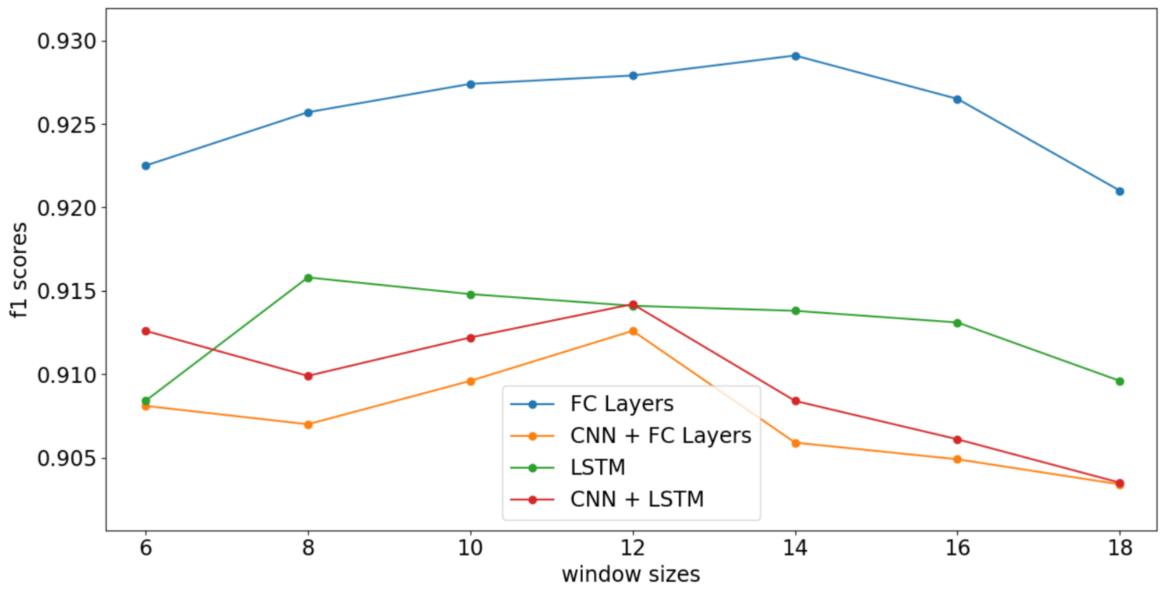


Figure 5.15: F1 score values related to the four neural networks tested with raw data, varying the data segmentation window size.

the input will be composed by matrices of shape (9, 444), while in the second case the matrices shape will be (9, 508). Furthermore, the results of this section are influenced by the hardware constraints of the machine used to perform the tests. CNN + FC Layers, LSTM and CNN + LSTM structures couldn't be too complex during the experiments, so their results would be increased without these constraints. Despite these issues, neural networks trained with raw data allow us to make some interesting considerations if compared with the ones trained with handcrafted features. These observations will be described in the next section.

Figure 5.15 shows the f1 scores obtained by the four different neural networks. Because of the hardware constraints, the FC layers structure obtained the best results. The most important thing to notice is that the LSTM neural networks achieve their best performances with a lower segmentation window size.

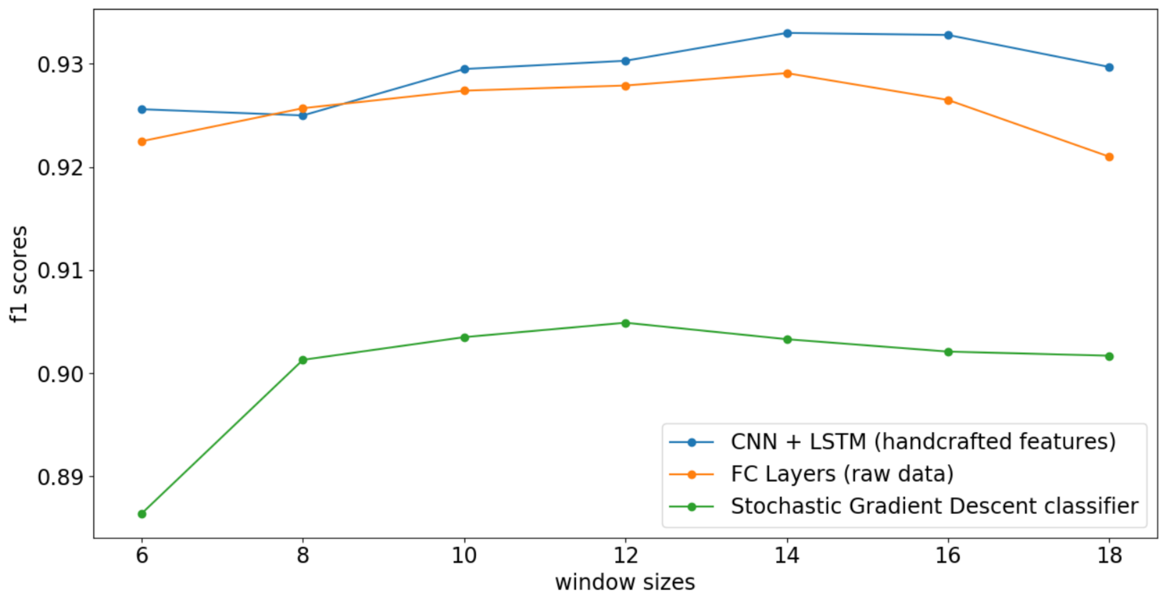


Figure 5.16: Comparison between the best model obtained using handcrafted features, using raw data and using another incremental algorithm, such as the Stochastic Gradient Descent classifier.

5.3.6 Comparison between handcrafted features and raw data

In this section, we compare the best results obtained with handcrafted features and raw data. Furthermore, we even consider the performance obtained with an incremental algorithm different from the neural networks. Figure 5.16 shows these results. We can notice that the use of handcrafted features or raw data allows the system to achieve similar results. This means that the handcrafted features chosen for this work can be considered good enough in the activity recognition field. Furthermore, neural networks can achieve better performances rather than other incremental algorithms such as Stochastic Gradient Descent (SGD) or Naïve Bayes classifiers. In Figure 5.16 only the SGD classifier is considered among other incremental algorithms since it obtained the best results excluding deep learning methods.

5.4 Semi-supervised extension evaluation

In this section, the best model obtained by the previous analysis is used to perform the hyperparameters tuning regarding semi-supervised techniques. Three hyperparameters were tuned:

- **Active learning entropy treshold**: when the probabilities vector emitted by the activity recognition model has an entropy value greater than this threshold, then the system asks the user to confirm which activity he's doing. The feature vector related to this prediction is labeled based on the user's feedback and used to update the classifier. The obtained tuned value for this parameter is 0.6
- **Batch size**: the number of samples labeled with the user's feedbacks used to fill the batch which will be used to update the classifier. The obtained tuned value for this parameter is 32
- **Self learning entropy treshold**: when the probabilities vector emitted by the activity recognition model has an entropy value lower than this threshold, then the feature vector related to this prediction is labeled based on the most probable inferred activity and used to update the classifier. During the experiments, the use of this parameter didn't lead to performances improvements

Furthermore, some experiments have shown the benefits that the context refinement introduces when semi-supervised techniques are used.

To conduct these experiments, at the beginning, the activity recognition classifier is trained only on data regarding a single subject of the dataset. Then, it is evaluated how the model improves its performances thanks to incrementally updates made on data concerning all the other subjects. This process is performed through the leave-one-subject-out cross-validation technique to obtain more robust results. In particular, these are the steps followed to conduct the experiments:

- for each subject of the dataset, all its data are used to train an activity recognition classifier. In this way, 12 different classifiers are built.
- for each classifier, it is built a test set with all the dataset examples that are not related to the user whose data were already used to train the classifier. These

examples are shuffled and divided into windows of 800 samples, built with an 80% of overlap. Each sample of each window is used to test the classifier. The statistical model is eventually updated through semi-supervised techniques during these tests, based on the hyperparameters mentioned before. When all the examples of a single window has been used to test the classifier, then it is possible to compute the f1 score obtained by the classifier on such examples.

- finally, the f1 mean score of each window is computed as the mean of the f1 scores obtained by the 12 classifiers related to that same window

In Figure 5.17 we can notice how the combination of active learning (with an entropy threshold of 0.6) and context refinement allows the system to achieve better performances, using a batch size parameter equals to 32 and without using any self-learning techniques. Furthermore, in Figures 5.18 and 5.19 we can see that the context refinement reduced also the number of user's feedbacks required by the system during the experiments.

Another important aspect is that we can compare the classifier trained on the whole training set, with the classifier trained through semi-supervised techniques. The first classifier is the one obtained through the experiments conducted before this semi-supervised section. It reached an overall f1 score of 0.9330 (as we have already seen in Figure 5.14). Thanks to Figure 5.17, we can notice that the classifier trained through semi-supervised techniques reached an overall f1 score of about 0.9250. Generally, fully supervised algorithms reach better performances than the semi-supervised ones. Their drawback is the annotation of the examples of the whole dataset. This semi-supervised solution allows achieving similar performances of the supervised one on our dataset. The benefit of this solution is that only the examples of a single subject has to be annotated. Its drawback is that before achieving results similar to the supervised solution, we have to wait for a certain number of classifier updates.

5.4.1 Active learning

Figure 5.20 shows how the f1 mean score trend changes over time, varying different active learning entropy thresholds. It is possible to notice that the best results are obtained with thresholds between 0.4 and 0.6. The Figures 5.21 and 5.22 show how

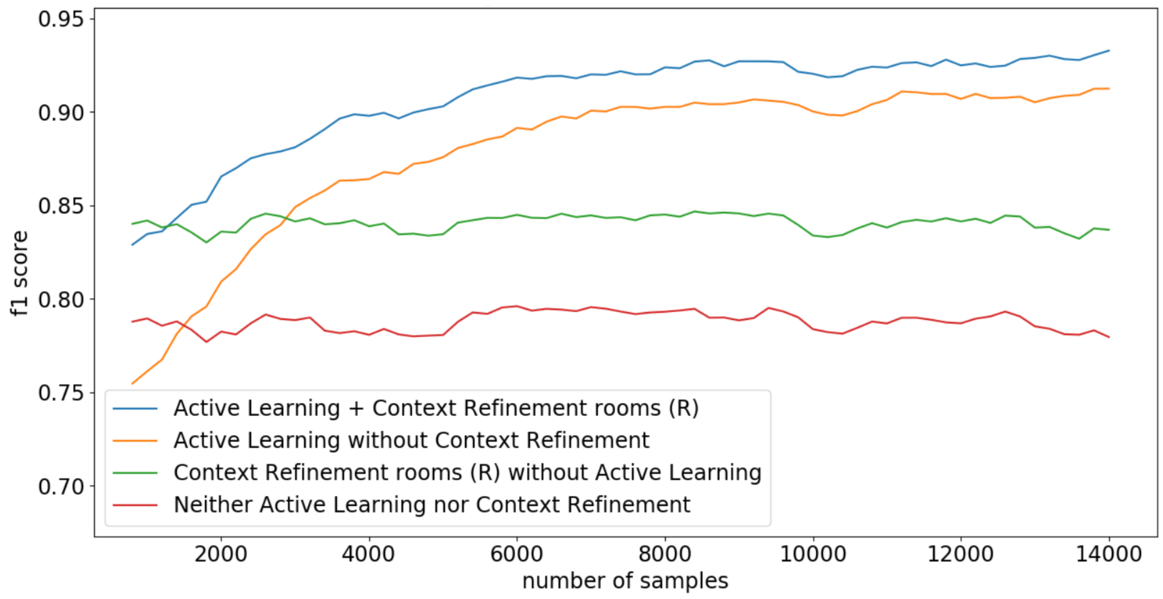


Figure 5.17: F1 score mean trend, comparing the usage of active learning and/or context refinement.

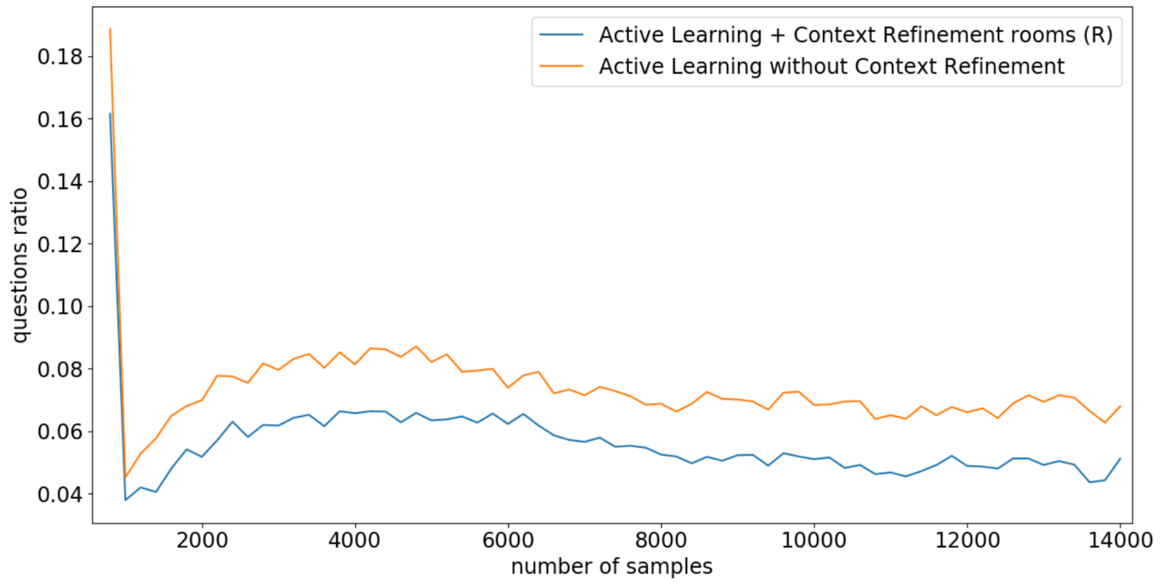


Figure 5.18: Variation over time of the user's feedback percentage requests, comparing the usage of active learning and/or context refinement.

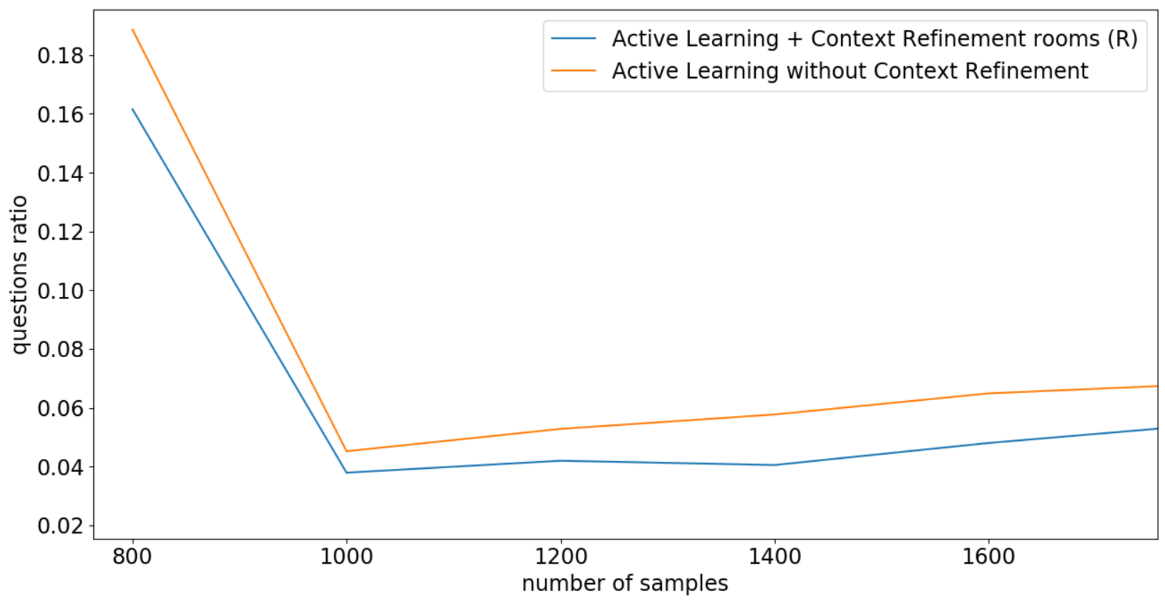


Figure 5.19: A zoomed version of the Figure 5.18.

the user's feedback percentage requests trend changes over time, varying different active learning entropy thresholds. Between the best results of Figure 5.20, here the value 0.6 is the one which reduces more the questions ratio. It is important to see that the greatest decrease in the number of feedback requests came after the first window of 800 samples. This means that, during the evaluation of this window, the system tends to ask frequently feedbacks to the user (about 17% of times with a 0.6 threshold). After the first 800 samples, the activity recognition model seems to be already enough certain about its predictions thanks to the previous requests, indeed the percentage is reduced until about the 5% of the number of samples.

5.4.2 Batch size

This parameter establishes how many feature vectors labeled thanks to user's feedbacks will fill the batch used to update the activity recognition classifier. Figure 5.23 shows that a batch size equals to 1 creates difficulties during the updates and doesn't improve very much the model performances over time. Probably, this happens since the classifier at each update changes its weights based on the example of a single activity and this can modify weights that are important for the prediction of other

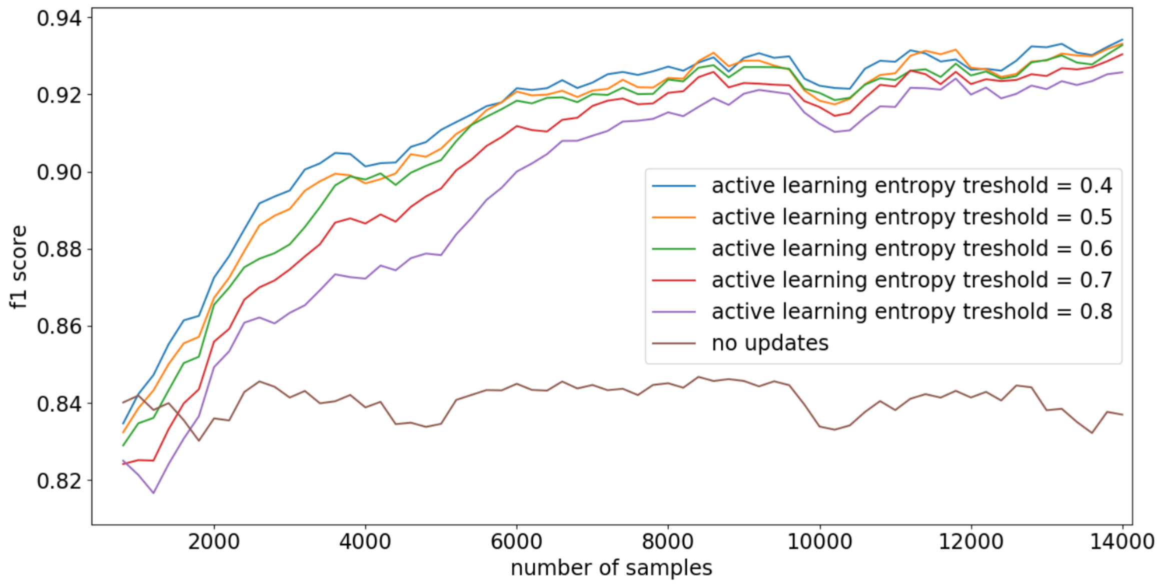


Figure 5.20: F1 score mean trend, comparing different active learning entropy thresholds.

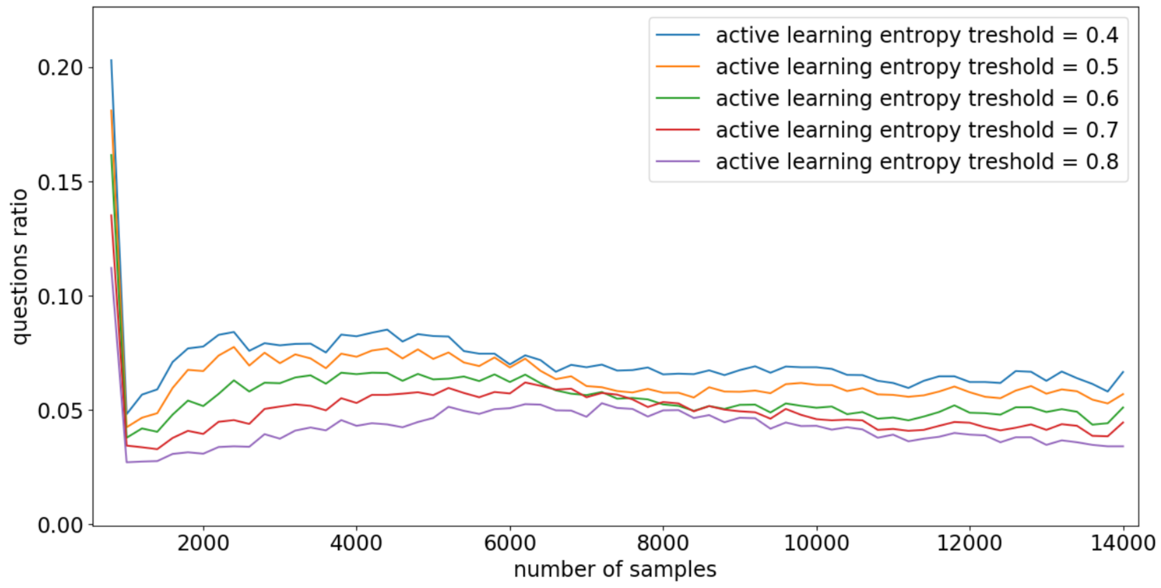


Figure 5.21: Variation over time of the user's feedback percentage requests, comparing different active learning entropy thresholds.

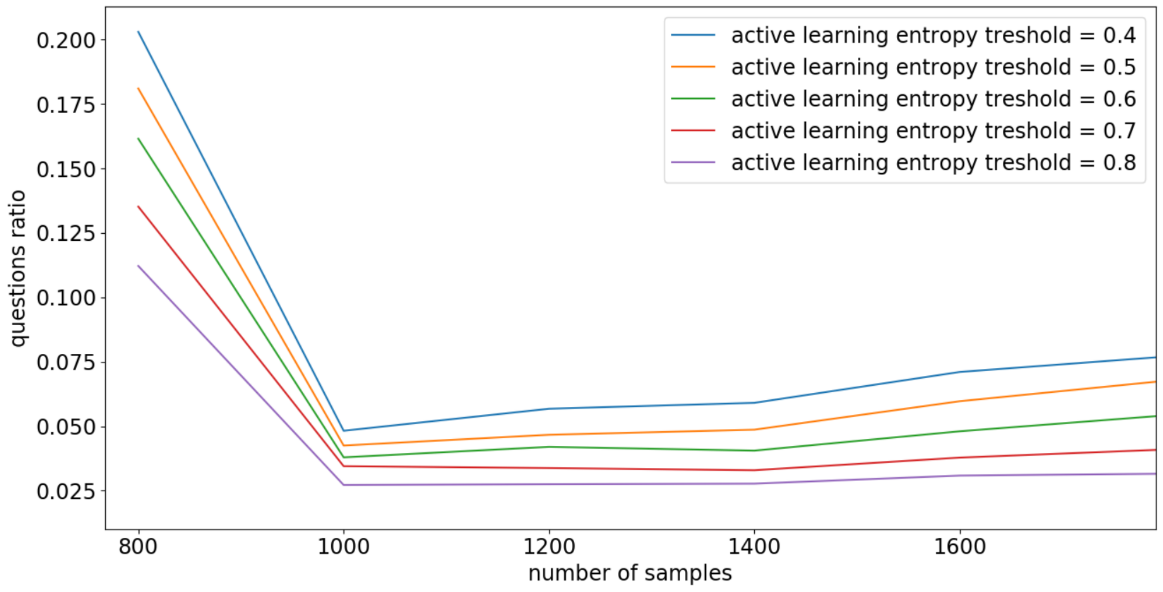


Figure 5.22: A zoomed version of the Figure 5.21.

activities. The best results are instead obtained with a batch size value equals to 32.

On the other hand, we can see in Figures 5.24 and 5.25 that a batch size value equals to 1 decreases a lot the percentage of user's feedback requests. This happens probably because the classifier receives updates more frequently since it doesn't need to fill the batch with more labeled examples. In this way, frequent updates generate a faster improvement of the performances at the beginning of the process, as we can see in Figure 5.23 for the f1 score values related to about the first 2000 samples. This improvement allows the system to reduce faster also the questions ratio. At the same time, the figures suggest that a batch size equals to 1 can lead the classifier to be sure about wrong predictions. Indeed, the decreasing number of user requests indicates that the algorithm is certain about its inferences, while the f1 score obtained by the classifier does not improve very much over time.

5.4.3 Prediction refinement based on context data

As already mentioned at the beginning of Section 5.4, the context refinement allows the system to improve the performances and to reduce the user questions ratio. In Figure 5.17, we can see that the context refinement without the use of active learning

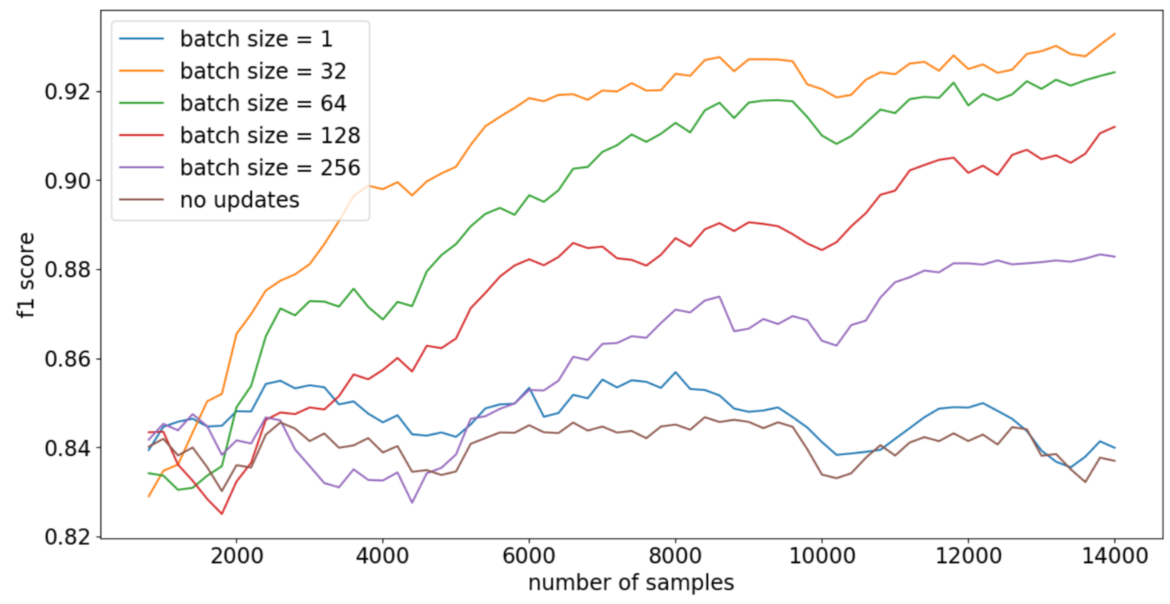


Figure 5.23: F1 score mean trend, comparing different batch size values.

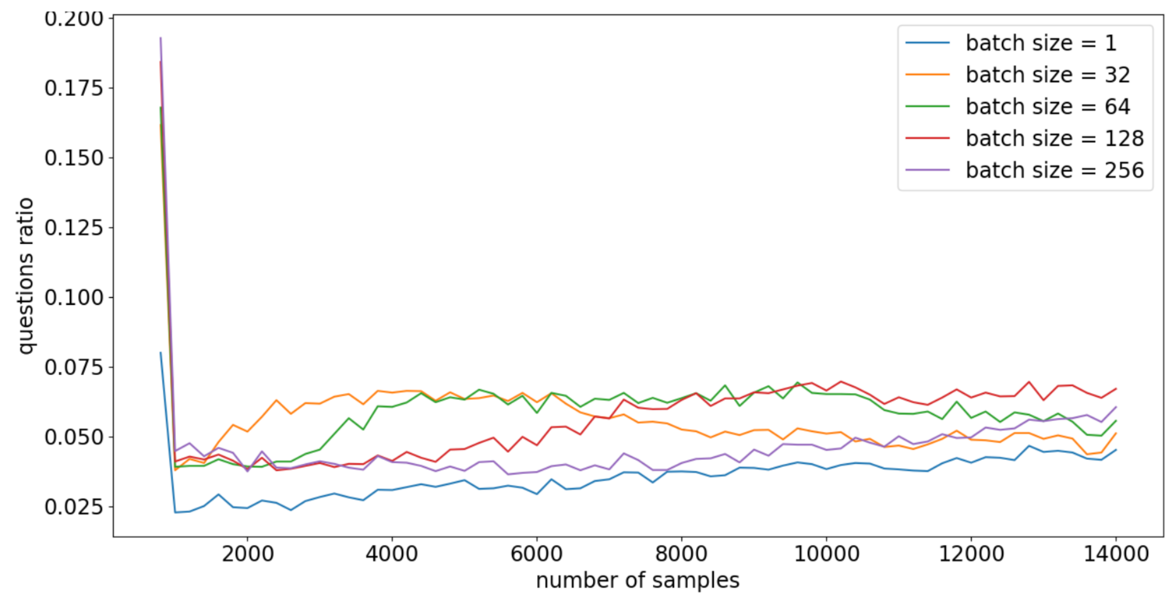


Figure 5.24: Variation over time of the user's feedback percentage requests, comparing different batch size values.

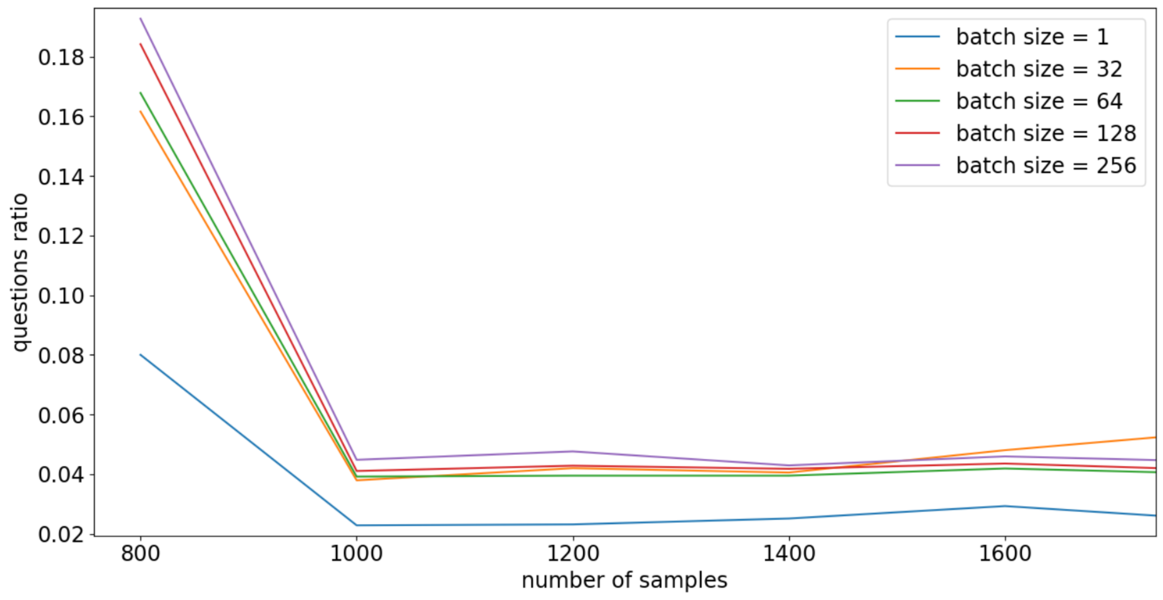


Figure 5.25: A zoomed version of the Figure 5.24.

increases the classifier performances of about 5 percentage points. It improves by about 2 percentage points the f1 mean score over time if used with active learning as well. Figures 5.18 and 5.19 show that context refinement reduces also the user questions ratio, generally with a 2 percentage points difference.

5.4.4 Self-learning

In Figure 5.26 we can see that using only the self-learning technique described in Chapter 4 leads to lower performance. This behavior can be explained through two examples. In the first example, the classifier is sure about the prediction of an activity which is already easy to infer for the classifier. So, if the model is already very good at predicting the activity WATCHING_TV, another labeled example of this activity doesn't improve the performances. Furthermore, it is possible that, in this way, the statistical model is updated in an unbalanced manner since the self-learning technique will be applied only to activities that are already detectable with an high recognition rate. In the second example, the classifier is sure about a wrong prediction. This could happen because of activities related to similar patterns of inertial and environmental data. So, updating the model with a feature vector labeled as COOKING while

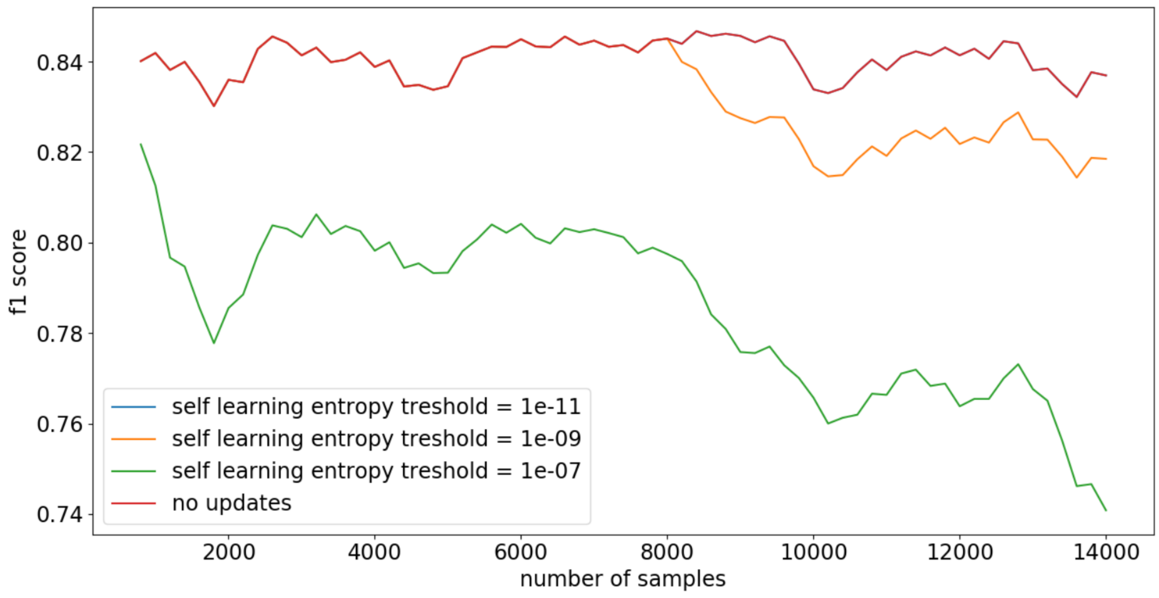


Figure 5.26: F1 score mean trend, comparing different self-learning entropy thresholds. Note that there wasn't any self-learning update using a threshold equals to 1e-11.

its ground truth is WATCHING_TV leads to a deterioration of the results. Similar results are obtained in [11], where self-learning never improved the activity recognition performances.

Figures 5.27, 5.28 and 5.29 show the results obtained with the combination of self-learning and active learning techniques. The idea is that active learning can improve the performances of the classifier and thanks to this improvement the self-learning could become effective. Actually, from these figures, we can understand that this is not happening since the combination of the techniques and the use of the active learning technique only lead to very similar results. For these reasons, the self-learning technique was not included in the system.

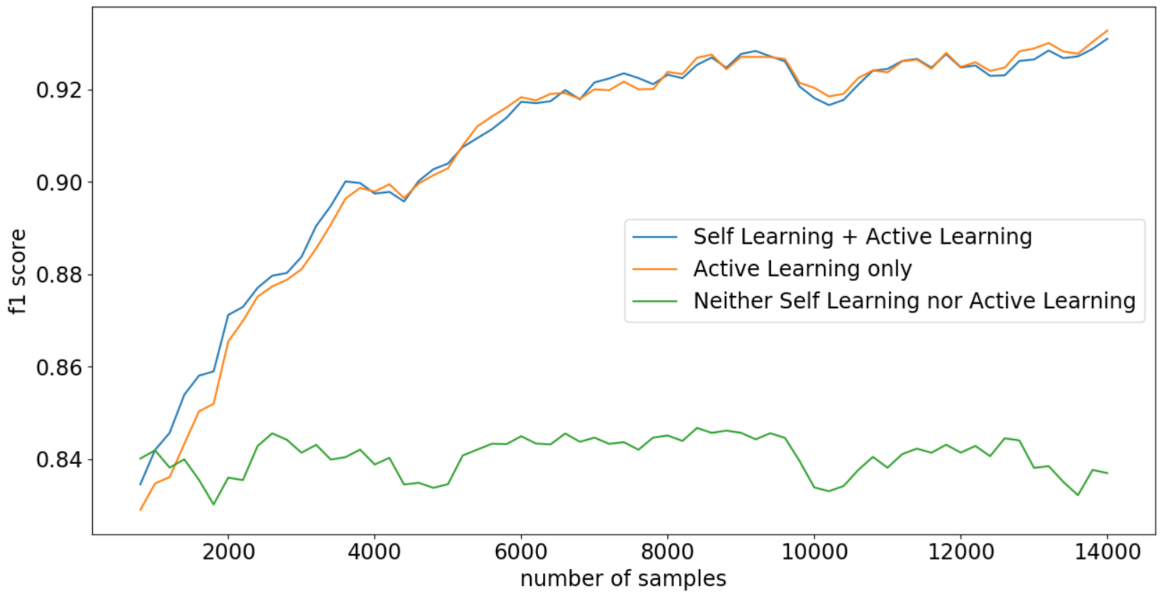


Figure 5.27: F1 score mean trend, comparing the use of self-learning technique alone or in combination with the active learning technique.

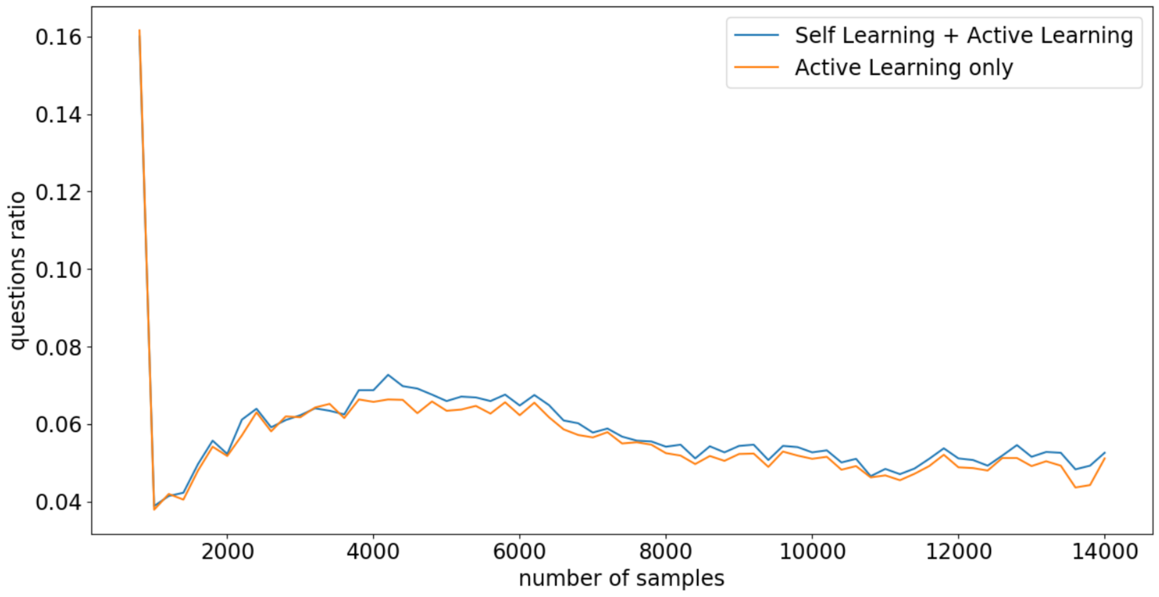


Figure 5.28: Variation over time of the user's feedback percentage requests, comparing the use of self-learning technique alone or in combination with the active learning technique.

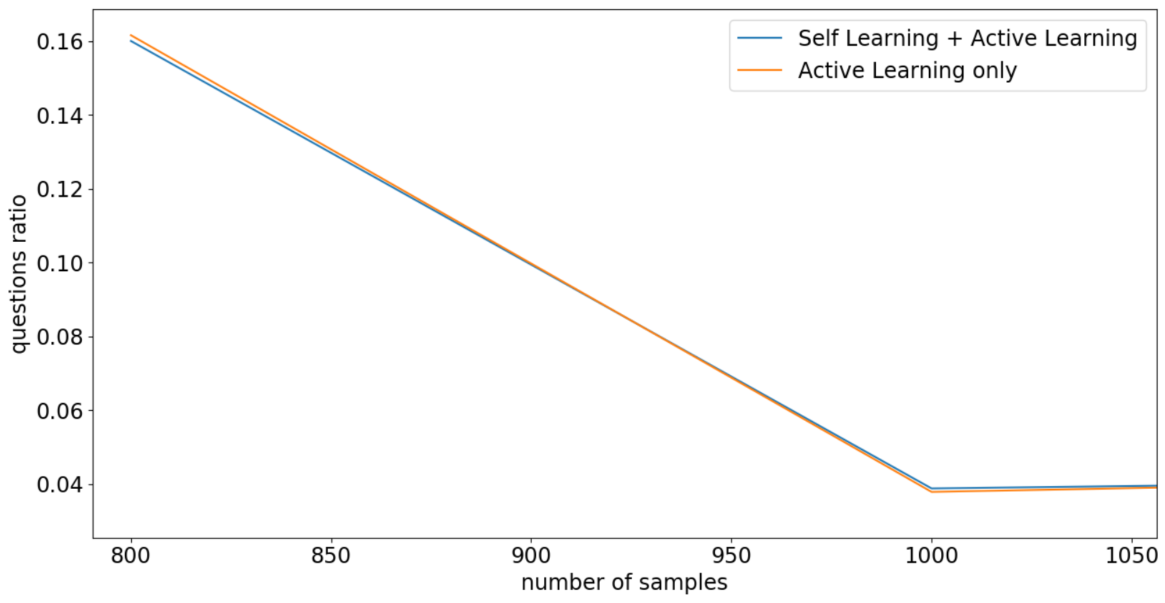


Figure 5.29: A zoomed version of the Figure 5.28.

Chapter 6

Implementation of a demo

A demo was set up at EveryWare Lab to test the system in a real-world setting, in which, instead of dataset examples, the system processes data coming in real-time from environmental and inertial sensors while the users are performing different activities. The division of the smart-home into the semantic areas during the deployment of this prototype was different than the one used to collect the dataset (e.g., the KITCHEN and the DINING_ROOM areas were collapsed to the only DINING_ROOM area). In this work, the existing demo is being modified following what the experimental results described in Chapter 5 suggested. During the development of the demo, some issues emerged conflicting with these experimental results, mainly because of the data used to train the activity recognition classifier. These problems led us to modify the behavior of the system and paved the way for new future works.

Section 6.1 explains how the demo was realized using as suggestions the experimental results obtained in this work. Section 6.2 shows the issues encountered during the development of the demo and how they were solved. Finally, Section 6.3 describes a multi-inhabitant scenario used to test the system in a real-time setting.

6.1 Demo structure

In this work, the existing demo realized at EveryWare Lab is being modified following what the experimental results described in Chapter 5 suggest. Figure 6.1 shows the architecture of the real-time system. Inertial and positioning data are collected by the application installed on the smartwatch of each user, while environmental sensors send information about the activated events to the smart-home gateway. All these data are sent to the Data Collection server that records them on a database. The Demo App gets from the database the information it needs to predict the high-level activities performed by the users. The Tablet Broker module developed in [25] collects the inferences of the Activity Recognition module and evaluate if they are certain or uncertain. Based on this decision, the Tablet App developed in [24] will display the activities performed by the users or asks them feedbacks regarding what they are doing. These feedbacks are used to update the activity recognition classifier. The following sections will describe in detail how each software component of the system works in a real-time setting.

6.1.1 Smartwatch App

Home Watch is the name of the Android application installed on the smartwatch of each user and used to collect inertial and positioning data. Each user has to wear the smartwatch on his dominant arm and, after the application is started, he has to insert a unique numeric ID. This process allows assigning automatically to the users inertial and positioning data collected by each smartwatch. The gathered inertial data come from accelerometer, gyroscope and magnetometer while positioning data are collected through the Bluetooth and WiFi antennas present on the smartwatches, that can detect in this way the signals emitted by BLE Beacons and WiFi access points installed inside the smart-home. These data are then sent to the Data Collection server that will record them on a database.

6.1.2 Smart-home gateway

The environmental sensors are handled through the Z-Wave wireless protocol. Thanks to a DomoticZ client, each environmental sensor is configured so that, when an event occurs, this information is sent using REST methods to a JavaScript server deployed on the smart-home gateway. This server, executed through NodeJS, has to format the received data and to send them to the Data Collection server, that will record them on a database.

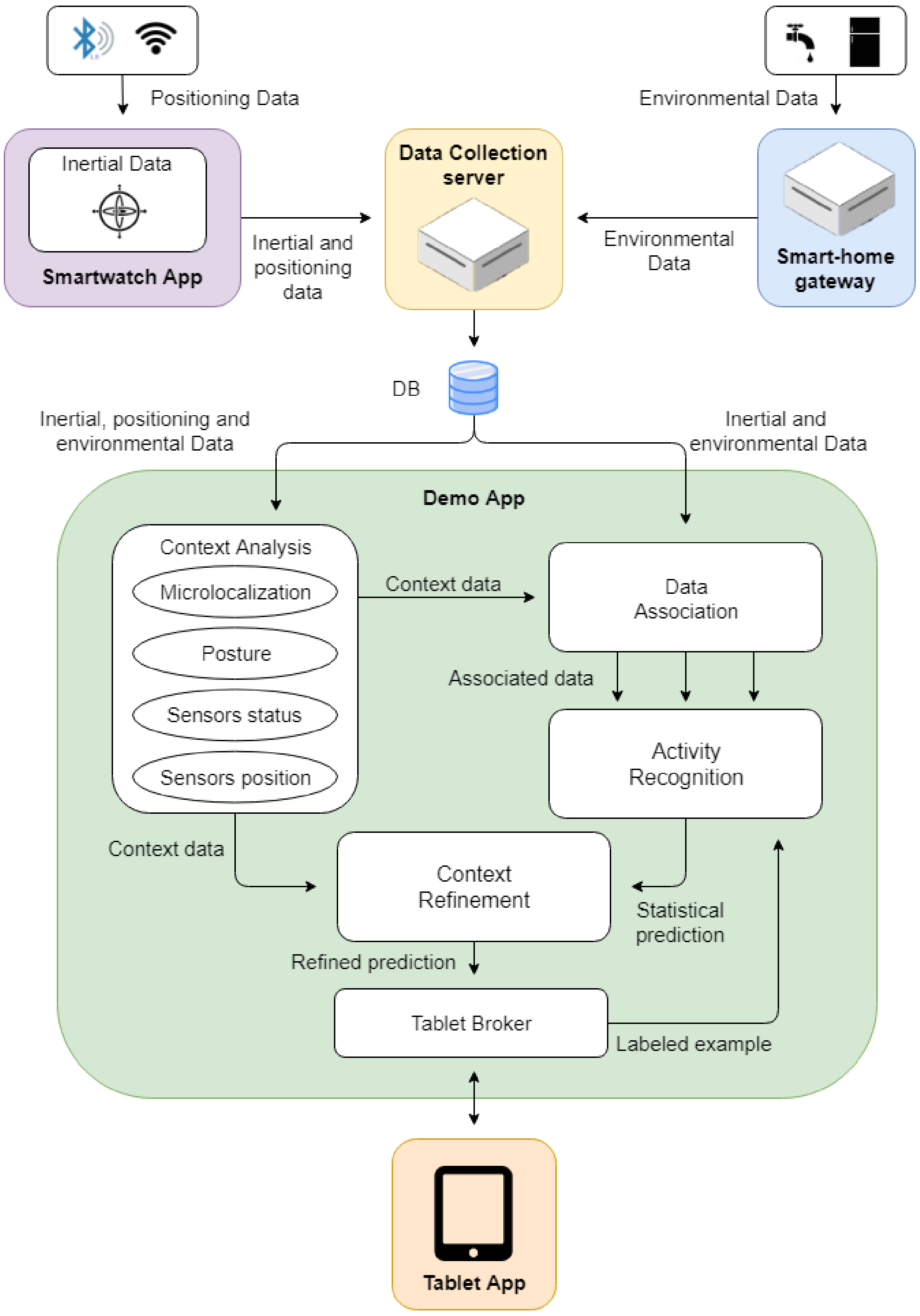


Figure 6.1: Architecture of the real-time system.

6.1.3 Data Collection server

The Data Collection server is implemented in Python and its purpose is to record on a MongoDB database the data that it receives from the smartwatches of the users and the smart-home gateway. Inertial, positioning and environmental data are recorded in different structures. Data coming from smartwatches are recorded into a MongoDB collection called *sensorData*. Each document of this collection contains information and measurements regarding inertial and positioning data detected by a smartwatch in a specific time interval. Data coming from environmental sensors are recorded into a MongoDB collection called *smartRoomData*. Each document of this collection contains information about an environmental event generated inside the smart-home.

6.1.4 Demo App

The Demo App is a script developed in Python, whose purpose is to monitor the users, detecting the activities they are performing. For each user monitored by the system, the Demo App starts two thread: *followUser* and *analizeUser*. The first one has to detect in real-time the position of the user inside the smart-home, while the second one uses a deep learning algorithm to recognize the high-level activities he performs. A data structure called *HomeStatus* is created to record information regarding the status of the smart-home. It memorizes the current position, posture and activity that each user is performing, the status and the position of the environmental sensors and who each pressure mat is assigned to. Furthermore, the *HomeStatus* data structure records a buffer that contains couples of inferences related to the last 5 seconds regarding the position and the activity of each user. These couples are appended to the buffer if the high-level activity inference has a probability that overcomes a fixed threshold. This process will be described in detail during the analysis of the *analizeUser* thread.

followUser In this thread, the micro-localization classifier is loaded, indicating the level of granularity that the system will use to divide the smart-home into semantic areas. When the same semantic area is predicted by the classifier with the highest probability for at least 4 times, then the *HomeStatus* is updated with the new position of the user.

analizeUser When the *analizeUser* thread starts, the single-inhabitant activity recognition classifier is loaded by the system. Then, a series of python iterators implements the operations described in Chapter 3 necessary to segment data extracted in real-time from the MongoDB database into windows of 14 seconds. From each window, handcrafted features are extracted using a context-aware data association that considers the position of the user inside the smart-home.

Once the Context Refinement module discards the logically impossible inferences emitted by the activity recognition classifier through the Micro-areas approach and renormalizes the vector of probabilities associated with each detectable activity, then the Tablet Broker module has to evaluate the uncertainty of the system about its prediction. When the most likely activity hasn't a value that at least twice overcomes the 50% of probability, then the inference is discarded. Otherwise, the status of the smart-home is updated with a new couple of inferences that contains the last micro-localization prediction recorded in the *HomeStatus* and the not-discarded most likely activity. If the probability of this activity is greater than 65%, then the prediction is considered certain by the Tablet Broker. Otherwise, it is considered uncertain. It is important to notice that the uncertainty of the system should be evaluated using the entropy value of the probabilities vectors emitted by the Activity Recognition module and then refined by the Context Refinement one. The focus of this demo was not to test the active learning technique, so it has inherited the use of probability thresholds from the existing application already developed at EveryWare Lab.

6.1.5 Tablet App

The Tablet App receives from the Tablet Broker module JSON data containing the probability distributions regarding the position and the activities performed by the users and information about possible uncertainties of the system. If the system is certain about its inference, the Android application will display the activities that the users are performing. As shown in Figure 6.2, if the system is uncertain about a prediction, the Tablet App will ask the user related to this doubt which is between his two most probable activities the correct one. Hence, the user can answer in two ways: through a tap on the screen of the tablet or through voice control. Eventual feedbacks of the users are then sent back by the Tablet App to the Tablet Broker

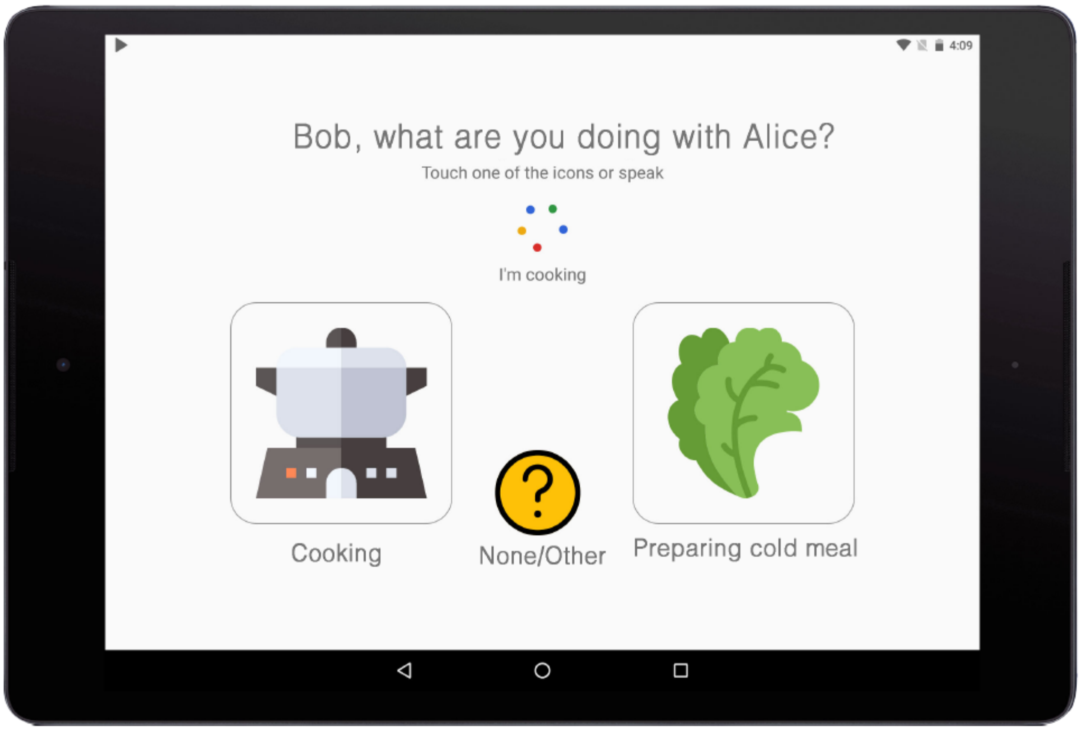

Bob, what are you doing with Alice?
Touch one of the icons or speak
I'm cooking
Cooking
None/Other
Preparing cold meal

Figure 6.2: Android activity of the Tablet App that is shown when users' feedbacks are required because of the uncertainty of the system.

module that can use them to update the activity recognition classifier, exploiting in this way the active learning technique.

6.2 Demo issues

During the deployment of the real-time demo, different issues arose:

- the use of the Context Refinement module can deteriorate the overall performance of the system
- during the execution of some scenarios, the posture context information is necessary to correctly assign pressure mats to users
- the use of recurrent neural networks could be tricky in a real-time setting

- some actions tend to be recognized when the user is going to change activity

The following four sections describe in detail each of these issues.

6.2.1 Change the prediction refinement process

In a real-time setting, the Context Refinement module currently developed in our solution can lead to a worsening of the activity recognition performances. This problem arises since the number of activities recognizable by the system is limited. For this reason, when a subject is performing an activity that is not detectable by our framework, then it could happen that the system is enough certain about a prediction that will be necessarily wrong. The same problem arises when a user is not performing any activity because he is in "transition" between two actions. The Context Refinement module makes the occurrences of this issue more probable. Indeed, when the prediction refinement process is applied, there is a good chance that the most likely activity is enough probable that it leads the Tablet Broker module to consider the system prediction as certain.

To solve this problem, it is necessary to take into account a new class of activities that we are going to call TRANSITION. Then, it is possible to proceed in two ways:

- train the activity recognition classifier considering even data examples regarding the TRANSITION action
- instead of discarding logically impossible activities through the Context Refinement module, these actions should be mapped to the TRANSITION activity

In our real-time system, we chose the second solution. An example can help to explain how this solution was implemented: the WATCHING_TV activity will be mapped to the TRANSITION one if the user is not in the same semantic area of the television smart-plug or if this sensor is not being activated. Furthermore, with this arrangement, the TRANSITION activity will never be displayed on the Tablet App, neither in case of a certain prediction nor if an uncertain inference occurs.

6.2.2 Pressure mats association

The experimental results described in Chapter 5 suggest that the context information regarding the users' posture is not necessary to improve the performances of the data association process. This happens since, in the collected dataset, every time a subject is acting while he is sitting on a pressure mat, all the other users who are in the same semantic area are performing the same action while sitting on other pressure mats. This problem was discovered thanks to the real-time implementation of the system and it suggests how, in machine learning solutions, it is not feasible to collect a dataset enough rich to consider all the possible scenarios that will be faced by the system after its deployment. Furthermore, in each dataset semantic area, at most only one activity could be performed while sitting on a chair. For this reason, when a pressure mat is assigned to a subject, the activity recognition classifier tends to predict the only activity that can be performed while sitting inside the semantic area in which the user currently is. Finally, the importance of using the users' posture to perform the data association process is more evident during the demo deployment because the division of the smart-home into semantic areas used in the real-time experiments was different than the one used to collect the dataset. For instance, the KITCHEN and the DINING_ROOM areas were collapsed to the only DINING_ROOM area. This means that the number of activities that can be performed in the DINING_ROOM area during the demo is greater than the one regarding the collected dataset. So, it is more likely that wrong assignements of pressure mats occur when different users are performing actions inside the DINING_ROOM semantic area.

At the beginning, the Demo App assigned an activated pressure mat to the first user whose *analizeUser* thread discovered that the sensor was being activated. For this reason, a pressure mat can be wrongly assigned to a user who is not sitting on a chair. For the reason previosuly explained, once the pressure mat is assigned to the user, the activity recognition classifier tends to predict the only activity that can be performed while sitting on a chair inside the user semantic area. We can consider the following example: user U1 is EATING in DINING_ROOM and he is sitting on a chair, while user U2 is WASHING_DISHES in the same semantic area. If the system assigns the pressure mat to U2, then the predicted activity of him will wrongly be EATING. To solve this problem, the low-level activity recognition classifier that can detect the users' posture was introduced in the Demo App through the *analizeSitting* thread. At the same time, the combination of Micro-areas, Posture and PosturePlus approaches was used to perform the context-aware data association. It is important to notice that the low-level activity recognition classifier presents some issues, already described in Chapters 3 and 5. Possible solutions that can be developed in the future will be mentioned in the last chapter.

analizeSitting When this thread is started, the low-level activity classifier is loaded to detect the posture of the user. As input, it receives the inertial handcrafted features extracted by each segmentation window. Since the data association is performed considering the users' posture, when the detected posture is SITTING, then it is checked if there is an active pressure mat that is not assigned to any user. In such a case, the sensor will be associated with the user until it will be disabled.

6.2.3 Use of an LSTM-based neural network

The neural network used inside the *analizeUser* threads exploits a combination of convolutional and LSTM layers. The use of recurrent layers could be tricky since they increase the probability that the inference of the activity recognition classifier is overfitted on data used to train the statistical model. For instance, the activity CLEARING_TABLE is easily detected by the system if it is performed by a user after the EATING action. This happens because in the training data often the subjects cleared the table after they ate. It is important to notice that CLEARING_TABLE is one of the activities that the system hardly recognize when a neural network that doesn't contain recurrent layers is used. Unfortunately, there can be also cases in which the recurrent layers lead to wrong predictions because the same pattern of subsequent activities can be found in different scenarios used to collect the dataset. For example, after that a user acts WASHING_DISHES, the system tends to detect the activity USING_PC. This particular situation is solved thanks to the prediction refinement that maps USING_PC to TRANSITION since the user is washing the dishes in DINING_ROOM and the computer can be used only in the OFFICE area.

6.2.4 Delayed predicitons

The last issue concerns the fact that the system tends to recognize some actions when the user is going to change activity. This happens for activities that typically require not much time to be performed (e.g., GETTING_IN, GETTING_OUT, SETTING_UP_TABLE and CLEARING_TABLE). To solve this problem, the performances of the high-level activities classifier should be increased when shorter segmentation windows are used. Decisions that regard handling the previously mentioned activities depend on the purpose of the deployed system. For instance, if it is important for the users to know in real-time the system predictions, then it would be improper to display on the tablet these activities after their execution.

6.3 Considerations about a tested scenario

The scenario described in Table 6.1 was executed several times by two subjects during the development of this work. The activity SETTING_UP_TABLE is hardly recognized by the system. Typically, its probability is not high enough to obtain an inference that can be considered certain by the Tablet Broker. Furthermore, the cases in which the SETTING_UP_TABLE activity is detected and considered as a certain prediction occur only after that Alice starts to prepare a cold meal. The activities PREPARING_COLD_MEAL, WATCHING_TV, COOKING and EATING are nearly always detected correctly and within a reasonable time. Rarely Alice and Bob are confused by the system while they are respectively performing EATING and COOKING. This happens when the pressure mat on which Alice is sitting is wrongly assigned to Bob. This is all because the low-level activities classifier sometimes improperly detect as SITTING the posture of Bob. If this happens before that Alice's posture is recognized as SITTING, so the first *analizeSitting* thread that will check if there are available and active pressure mats will be Bob's one. Since Bob will be considered seated, when the cooker is activated, it will be assigned erroneously to Alice. In the last chapter, possible solutions to improve the performances of the posture classifier are described. Furthermore, CLEARING_TABLE is correctly detected by the system when an LSTM-based neural network is used. Otherwise, its prediction leads to the

Alice	**Bob**
SETTING_UP_TABLE (DR)	WATCHING_TV (LR)
PREPARING_COLD_MEAL (DR)	WATCHING_TV (LR)
EATING (DR)	COOKING (DR)
EATING (DR)	EATING (DR)
WATCHING_TV (LR)	CLEARING_TABLE (DR)
WATCHING_TV (LR)	WASHING_DISHES (DR)

Table 6.1: The tested scenario. DR and LR mean respectively DINING_ROOM and LIVING_ROOM.

same considerations made for the activity SETTING_UP_TABLE. Finally, WASHING_DISHES is correctly detected by the system when Bob is performing it, but sometimes it is also predicted in place of other actions (e.g., SETTING_UP_TABLE). This happens typically at the beginning of the scenario, when the two subjects start to perform their activities.

Chapter 7

Conclusion and future work

The focus of this work was to extend an existing multi-inhabitant activity recognition system with semi-supervised learning techniques, using deep learning algorithms. Furthermore, the use of context data has been extended to tackle the data association problem and to refine the inferences of the deep learning algorithms. Particularly, a module was added to the system to detect the posture of the users. This system can detect the actions performed by the users who inhabit a smart-home, using inertial data coming from their smartwatches and environmental events generated by the sensors installed inside the habitation. The main issue of a multi-inhabitant activity recognition system is to associate the environmental events to the users who could have generated them. The developed system uses context data such as the position of the users inside the smart-home, their posture and the position and the status of environmental sensors to address this problem. The position of the users is computed by a random forest classifier that receives as input data coming from BLE beacons and WiFi access points. The posture of the users is detected by a neural network that receives as input data coming from the smartwatches of each user. The position of the environmental sensors instead is known and fixed inside the smart-home. Once the environmental events are associated, it is possible to create for each user feature vectors containing both inertial and environmental data. These feature vectors are used by a deep learning algorithm which combines convolutional and recurrent layers to recognize the high-level activities performed by the users. The predictions of the activity recognition classifier are then refined through a reasoning module that discards the logically impossible activities based on the available context data. Finally, when the system is uncertain about its inference, it is asked to the user through a tablet application which is between the two most probable activities the one he is performing. The feedback of the users is then exploited to annotate new data that can be used to update the activity recognition classifier. The interaction between the system and the users is possible thanks to an Android tablet application developed in parallel works.

The developed system has been tested in a real-time setting. This setting was different than the one simulated to collect the dataset used to train the activity recognition classifier and to evaluate all the modules of the system. Particularly, the division of the smart-home into semantic areas was less sophisticated and the position of the environmental sensors has been changed, even if they were still placed in the same semantic areas as before. These adjustments allow making the deployment of the system more realistic. In the tested multi-inhabitant scenario described in Chapter 6, most of the performed activities were correctly recognized by the system, even if sometimes there is a subtle difference between the detectable actions (e.g., the system can distinguish the COOKING activity from the PREPARING_COLD_MEAL one). The use of positioning and posture context information to tackle the data association problem allows improving the assignments of environmental events to users and, therefore, even the recognition rate of the deep learning algorithms. The behavior of the Context Refinement module has been changed in order to make it more effective in a real-world setting. This adjustment will be further explained in this chapter. However, the deployment of the system shows how the context data are useful to mitigate possible classification errors generated by the statistical model. The active learning technique has been implemented in the real-time system, but the focus of the demo was not on its semi-supervised extension. So, the effectiveness of the active learning technique in a real-time setting cannot be evaluated in this work.

In a real-time setting, different issues should be solved. When the user is not performing any activity detectable by the system, the Context Refinement module shouldn't be applied. As already mentioned in Chapter 6, the reason is that, for instance, the system can't detect an activity that indicates that the user is in transition between two detectable actions. So, if the refinement is applied in these situations, usually the most likely activity will be enough probable to be indicated to the user as the correct one even if he is not performing any detectable action. To solve this problem, the activity recognition classifier should be trained even with examples regarding an action such as TRANSITION. Currently, the behavior of the Context Refinement module is being changed in this way: instead of discarding not context-consistent activities, it maps them to the TRANSITION activity. Such an activity will never be displayed by the tablet application. This is very useful to avoid showing to the users activities that couldn't be performed.

The problem of the posture classifier is that it was trained mapping the high-level activities' labels to two possible low-level annotations: SITTING and NOT_SITTING. Troubles come when a high-level activity that is mapped to SITTING is confused by the posture classifier with another one that is mapped to NOT_SITTING, or vice versa. This happens for example when COOKING is confused with WATCHING_TV because they are both very static actions. In such cases, the detected posture will be SITTING instead of NOT_SITTING. To solve this problem there are two possible solutions.

- The low-level classifier can be trained with examples of the activity STANDING_STILL in which the user is standing without performing any action. This solution solves the issue when the involved activities are COOKING and WATCHING_TV, but it doesn't handle all the instances of the problem.
- A more robust solution consists of creating the training set for the posture classifier following these steps: for each user, we label as SITTING only the feature vectors obtained with inertial data temporarily contained between the activation and the subsequent deactivation event of the same pressure mat; all the other feature vectors will be annotated as NOT_SITTING.

Furthermore, the low-level activities classifier could be trained to detect more than two postures. Such a solution would be used by the Context Refinement module to discard activities that can't be performed with the current posture of a user.

Some activities are confused because inertial data are not sufficient to distinguish them. In this case, it is possible to install inside the smart-home other environmental sensors that should help to make some activities more identifiable through them. For instance, a sensor could detect if the tap of the kitchen is being opened or not. This would help to recognize the WASHING_DISHES activity.

Some activities are confused because the system hasn't context information enough accurate to perform correctly the data association task. This happens for the activities that can be performed in the KITCHEN and the DINING_ROOM semantic areas. To solve this problem, there could be two possible solutions:

- train a gesture detection classifier that recognizes when a user opens a drawer or turns on the stove
- use a finer-grain positioning infrastructure that allows the system to divide the smart-home into smaller semantic areas. In this way, if the kitchen of the smart-home is divided into different semantic areas, when the fridge will be opened the system will be uncertain only about users who are in the FRIDGE_AREA and not about all the people present in the KITCHEN

Bibliography

[1] Daniele Guarnieri. Riconoscimento real-time di attività multi-inhabitant in ambiente smart-home. Master degree thesis, 2018.

[2] Chiara Franceschetti. Micro-localizzazione e sue applicazioni in ambiente smart-home multi-inhabitant. Master degree thesis, 2018.

[3] Liming Chen, Jesse Hoey, Chris D Nugent, Diane J Cook, and Zhiwen Yu. Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):790–808, 2012.

[4] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209, 2012.

[5] Gary M Weiss, Jessica L Timko, Catherine M Gallagher, Kenichi Yoneda, and Andrew J Schreiber. Smartwatch-based activity recognition: A machine learning approach. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 426–429. IEEE, 2016.

[6] Geetika Singla, Diane J Cook, and Maureen Schmitter-Edgecombe. Recognizing independent and joint activities among multiple residents in smart environments. *Journal of ambient intelligence and humanized computing*, 1(1):57–63, 2010.

[7] Nirmalya Roy, Archan Misra, and Diane Cook. Ambient and smartphone sensor assisted adl recognition in multi-inhabitant smart environments. *Journal of ambient intelligence and humanized computing*, 7(1):1–19, 2016.

[8] Liang Wang, Tao Gu, Xianping Tao, Hanhua Chen, and Jian Lu. Recognizing multi-user activities using wearable sensors in a smart home. *Pervasive and Mobile Computing*, 7(3):287–298, 2011.

[9] Tobias Nef, Prabitha Urwyler, Marcel Büchler, Ioannis Tarnanas, Reto Stucki, Dario Cazzoli, René Müri, and Urs Mosimann. Evaluation of three state-of-the-art classifiers for recognition of activities of daily living from smart home ambient data. *Sensors*, 15(5):11725–11740, 2015.

[10] Hadi Tabatabaee Malazi and Mohammad Davari. Combining emerging patterns with random forest for complex activity recognition in smart homes. *Applied Intelligence*, 48(2):315–330, 2018.

[11] Brent Longstaff, Sasank Reddy, and Deborah Estrin. Improving activity classification for health applications on mobile devices using active and semi-supervised learning. In *2010 4th International Conference on Pervasive Computing Technologies for Healthcare*, pages 1–7. IEEE, 2010.

[12] Young-Seol Lee and Sung-Bae Cho. Activity recognition with android phone using mixture-of-experts co-trained with labeled and unlabeled data. *Neurocomputing*, 126:106–115, 2014.

[13] Gabriele Civitarese, Riccardo Presotto, and Claudio Bettini. Context-driven active and incremental activity recognition. *arXiv preprint arXiv:1906.03033*, 2019.

[14] Liming Chen, Chris D Nugent, and Hui Wang. A knowledge-driven approach to activity recognition in smart homes. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):961–974, 2011.

[15] Daniele Riboni and Claudio Bettini. Owl 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing*, 7(3):379–395, 2011.

[16] Daniele Riboni and Claudio Bettini. Cosar: hybrid reasoning for context-aware activity recognition. *Personal and Ubiquitous Computing*, 15(3):271–289, 2011.

[17] Gorka Azkune, Aitor Almeida, Diego López-de Ipiña, and Liming Chen. Extending knowledge-driven activity models through data-driven learning techniques. *Expert Systems with Applications*, 42(6):3115–3128, 2015.

[18] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[19] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[21] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016.

[22] Francisco Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[23] MH Subhashini and M Arumugam. Analysis of variance (anova). *CMFRI Special Publication*, pages 169–170, 1981.

[24] Andrea Cremonesi. Studio di un'interfaccia conversazionale per ambienti smart-home multiutente. Master degree thesis, 2019.

[25] Pierpaolo Albrici. Integrazione di un'interfaccia locale con un sistema di riconoscimento di attività multi utente. Bachelor degree thesis, 2019.