UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Classification techniques for sensor-based activity recognition in smart homes

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Carlos Güemes Palau

Tutor: Eva Onaindía de la Rivaherrera

Course 2019-2020

Acknowledgements

First I would like to thanks Dr. Eva Onaindia for proposing this topic to me as well for her guidance along the entire development of the project. I would also like to thanks Emanuel Zaymus who was performing a traineeship at DSIC during Eva's supervision and helped in the task of choosing the most adequate parameters values for the SVM implementation in this paper, as well as how to scale the data and which kernel function to apply.

Resum

En els últims anys ha crescut l'interés en torn el reconeixement de l'activitat humana, especialment en els camps de la salut on es pot utilitzar per a augmentar la qualitat de vida, confort i seguretat de la població dependent i d'edat avançada. L'auge de l'internet de les coses (IoT) i la tecnologia associada permeten als investigadors entrenar models predictius a partir de cases intel·ligents equipades amb diferents tipus de sensors on els residents poden dur a terme la seua vida diària. L'objectiu d'este treball consistix en una anàlisi de l'efectivitat de diferents models de classificació a l'hora de reconéixer l'activitat humana a partir de conjunts de dades ja desenrotllats. Particularment, utilitzarem dades publicats pel projecte CASAS de la Universitat Estatal de Washington a l'hora d'examinar l'efectivitat i precisió de diverses tècniques de preprocessament i models. El dipòsit CASAS proporciona dades tant d'experiments controlats, on es busca examinar activitats específiques, com a dades arreplegats de voluntaris vivint en cases intel·ligents, on intenten dur a terme la seua rutina diària. En este treball es realitzarà una avaluació exhaustiva de diversos models diferents, des d'un classificador de bayes ingenues fins a boscos aleatoris, identificant les seues fortaleses i debilitats a l'hora de tractar amb diferents fonts de dades i diferents tècniques de preprocessament.

Paraules clau: reconeixement d'activitat humana, cases intel·ligents, models predictius, classificador de bayes ingenues

Resumen

En los últimos años ha crecido el interés en torno el reconocimiento de la actividad humana, especialmente en los campos de la salud donde se puede utilizar para aumentar la calidad de vida, confort y seguridad de la población dependiente y de edad avanzada. El auge del internet de las cosas (IoT) y la tecnología asociada permiten a los investigadores entrenar modelos predictivos a partir de casas inteligentes equipadas con diferentes tipos de sensores donde los residentes pueden llevar a cabo su vida diaria. El objetivo de este trabajo consiste en un análisis de la efectividad de diferentes modelos de clasificación a la hora de reconocer la actividad humana a partir de conjuntos de datos ya desarrollados. Particularmente, utilizaremos datos publicados por el proyecto CASAS de la Universidad Estatal de Washington a la hora de examinar la efectividad y precisión de varias técnicas de preprocesamiento y modelos. El repositorio CASAS proporciona datos tanto de experimentos controlados, donde se busca examinar actividades específicas, como datos recogidos de voluntarios viviendo en casas inteligentes, donde intentan llevar a cabo su rutina diaria. En este trabajo se realizará una evaluación exhaustiva de varios modelos diferentes, desde un clasificador bayesiano ingenuo hasta bosques aleatorios, identificando sus fortalezas y debilidades a la hora de tratar con diferentes fuentes de datos y diferentes técnicas de preprocesamiento.

Palabras clave: reconocimiento de actividad humana, casas inteligentes, modelos predictivos, clasificador bayesiano ingenuo

Abstract

Research on human activity recognition in smart homes has drawn lately much attention as a means to improve the quality of life, comfort, safety and health care of elderly and dependent people. The advancement of the technology in the Internet of Things (IoT) and smart homes enables the utilization of dataset benchmarks in experimental scenarios for testing appropriate predictive techniques of daily activities of residents at

home. The objective of this project goes in this research direction and presents a study of the performance of different classification techniques in a sensor-based dataset of human activity. Particularly, we use the public datasets provided by the CASAS project of the Washington State University and we examine the suitability and accuracy of various classification techniques in scheduled activities as well as in daily life recordings. We perform an exhaustive evaluation of various techniques that range from a Naive Bayes classifier to Random Forests, identifying the strengths and weaknesses of the tested techniques to each type of dataset.

Key words: human activity recognition, smart homes, predictive models, Naive Bayes classifier

Contents

CHAPTER 1
Introduction

1.1 Motivation

My first introduction to the field of Human Activity Recognition (HAR) is when Dr. Eva Onaindía, professor at the Department of Computer Systems and Computation (DSIC) and my tutor for this project, contacted me to participate in research in this field. The topic HAR interested me as it sounded as a novel and challenging problem. HAR is certainly a complex task due to the huge variety of data sources, data extraction methods and Machine Learning (ML) techniques to solve the task. In addition, there is not a clearly dominant technology that monopolizes the attention of the researchers, but instead an outstanding number of proposed algorithms.

Advances in the field of HAR have seen application in the medical field. For example, the work done by classifiers that use human vitals has been applied to monitor the rehabilitation of patients with cardiovascular problems. [19]. Additionally, research in activity recognition in smart homes puts specifically the focus on the development of solutions that improve the quality of life of elderly and dependent people. Precisely, one of the datasets used in this paper, the *Daily Life Recording* dataset, records and measures signal of daily life activities of an elderly person in a monitored smart home. [9]. The creators of this dataset have recently published a paper detailing the implementation of a robotic support system which monitors and assists individuals in certain activities [31].

The quality of life of elderly and dependent people is a growing issue. For example, in Spain recent estimates show that currently over 18.49% of the population is aged 65 years and older[1], and it is expected to grow. Personally, I have two grandmothers that, while not purely dependent, they do rely on their daughters for some tasks, such as certain parts of housekeeping or carrying heavy objects. The problem is that both of their daughters have full time jobs and can only offer help during their limited free time. Solutions such as the ones developed in [31] would allow people who to take care of their family members to do so without sacrificing so much of their personal time, as well as being more convenient for the person taken care of.

During the project development I also realized that despite the lots of feature extraction methods and ML algorithms, there are hardly attempts to delve into the suitability of classification techniques to specific datasets and to understand their strengths and shortcomings. In this sense, our aim is to cover this gap by analyzing three of the most well-known classifiers used in the HAR field. The three classification techniques were chosen according to three criteria: they are all compatible with the same set of features, they are fundamentally different in nature and they present a varying degree of popularity and success in HAR.

[1]The World Factbook - Central Intelligence Agency. Age Structure. https://www.cia.gov/library/publications/the-world-factbook/fields/341.html#SP. Acessed: 2020-07-01

1.2 Objectives

The objectives we pursue in this project are the following:

1. Assessment of different classification techniques in handling the task of Human Activity Recognition (HAR) in smart homes.
2. Identifying publicly available HAR datasets, how they differ in nature and how to extract relevant features; to utilize several different nature datasets under a similar representation during the evaluation of the classifiers so as to compare their performance.
3. To study how the raw data coming from the sensors are interpreted and formatted to be usable by different classifiers; to learn ways for adding contextual information to samples to be used by non-contextual classifiers, such as *sensor event based windowing*.
4. Investigating the type of features extractable from the sensor-based data, their usefulness and the type of information they attempt to capture; to experiment with different data pre-processing methods so as to increase the data quality, reduce data noise and/or identify discriminative information such as the use of sensor and time dependencies.
5. To compare the performance of a Naive Bayes, a Support Vector Machine and a Random Forest classifier on the the task of HAR using different metrics such as accuracy and F-score; analysis of the confusion matrices generated by the trained models to observe how the nature of different daily activities affect the classifiers' behaviour; to get the big picture on understanding the strengths and weaknesses of each classification technique.

1.3 Organization of the document

This work is structured as follows. First, in Chapter 2, we will discuss the current state of the art in Human Activity Recognition. In section 2.1 we will introduce the concept of Human Activity Recognition, followed by section 2.2 which mentions the currently available datasets for HAR. The chapter ends with section 2.3 that addresses the classification techniques used in the task of HAR.

Chapter 3 discusses the details of a HAR task applied to smart homes. First, in section 3.1 we formally introduce the problem of HAR. Second, in section 3.2, we detail the datasets used to evaluate the classification techniques in the following chapter. Sections 3.3 and 3.4 focus on the data processing and extraction of features from the datasets. Finally, section 3.5 presents the set of features that we will use to build the feature vectors.

In Chapter 4 we will evaluate the performance of three classification techniques on the datasets presented in Chapter 3. Initially, section 4.1 explains the evaluation setup and the metrics to use for the classifiers assessment. Each of the following sections is devoted to a thorough evaluation of a classification technique, providing details on the implementation and commenting the results across all the tested datasets. Section 4.2 is devoted to Naive Bayes, 4.3 to SVM and 4.4 to Random Forest.

Chapter 5 summarizes the take-home lessons from this project and outlines some future work directions.

CHAPTER 2
State of the art

In this chapter, we first briefly introduce the field of Human Activity Recognition. Then we present various research and academic projects on this topic as well as a summary of the most widely used state-of-the-art machine learning classification methods for activity recognition.

2.1 Human Activity Recognition

Human Activity Recognition (HAR) is the study about the identification of the action of a person based on sensor data. HAR is applicable to many contexts, mainly in smart homes for eldercare and healthcare monitoring in combination with other technologies like Internet of Things (IoT) [30].

Advances in HAR can be classified accordingly to the type of sensors utilized. Thus, we can distinguish:

- Applications that principally utilize wearable body sensors and mobile devices. These sensors can measure vitals from the participants, such as pulse meters, blood pressure, the speed of the participants using accelerometers or other physical properties. This type of sensors are commonly used in smart healthcare monitoring systems [28].
- Applications that rely on sensors located in the environment in which the activity takes place such as smart homes. These applications are meant to elderly care and the purpose is to recognize daily living activities so as to identify anomalous behaviours. Sensors typically used here are state-change sensors like motion or temperature sensors [26].

Our work is positioned within the second type of applications, that is, detection and recognition of activities in smart homes via sensors that monitor the presence and movement of people, the utilization of items and utensils and room temperature. This type of sensors are placed in the environment, typically take readings periodically or continuously, and data is stored only when the measurement value of the sensor changes (as is the case in the datasets used in this paper).

2.2 Projects in Human Activity Recognition

There exist various projects aimed at developing solutions for the task of Human Activity Recognition. Following we present the most relevant ones.

One of the most popular HAR projects is the one developed at the Center for Advanced Studies in Adaptive Systems (CASAS) at Washington State University (WSU) [1]. CASAS originally started in 2005 with a focus on the problem of gathering information from 'smart' environments (like environments equipped with an array of sensors) to build models of the behaviour of the people [33]. Since then CASAS has also expanded into the study of wearable sensors to monitor patients with cardiovascular diseases [19]. When performing research in smart environments, CASAS sets up several testbeds consisting in adapted homes and apartments equipped with a set of sensors where volunteers can perform daily activities. The CASAS project offers public access to the data collected from the home residents via sensors with the aim to be used by other researchers.

The Opportunity Project is also known as Activity and Context Recognition with Opportunistic Sensor Configurations [2]. This project is the result of the joint work of several institutions coordinated by the ETH Zürich University and the European Union. Its objective is researching ways to tackle the problem of HAR by developing general algorithms that are applicable independently of the availability, placement and nature of sensors. Researchers of the Opportunity project have published papers and organized conferences from 2009 to 2013, related to the use of both wearable and environment sensors.

There also exist several public datasets concerned with the study of HAR and published by researchers involved in the field:

- Van Kasteren[3]: Besides the datasets of the project CASAS, the ones published by Dr. Tim van Kasteren are the only public datasets that comprise data extracted exclusively from sensors located in the environment rather than from wearable sensors.
- MHEALTH dataset [2, 3]: published by several professors from the University of Granada, it encompasses data collected from participants wearing sensors that measure movements and vitals while performing activities such as "Cycling" and "Jogging".
- Human Activity Recognition Using Smartphones dataset (UCI HAR) [10]: a collaborative work of professors from the University of Genova and the Polytechnic University of Catalonia, this dataset is built specifically for the task of HAR where data is collected from a set of wearable sensors coordinated through a smartphone.

2.3 Classification techniques in HAR

The earliest publications on the application of classification techniques to the HAR task was published by the CASAS project in 2005 [33]. In these first publications, authors used Hierarchical Hidden Markov Models, a type of sequential model. Later in 2010, other sequential models where introduced and compared, including Hidden Markov Models (HMM), Hidden semi-Markov Models, Conditional Random Fields and semi-Markov Conditional Random Fields [29]. Overall, sequential models seem like a natural fit for HAR, as data captured by state-change sensors store data sequentially. However, the set of features and information that can be extracted from the data is more limited than in other models. Consequently, while they are still in use today, they are not the only viable classification techniques and will often depend on the dataset.

[1] Center of Advanced Studies in Adaptive System. http://casas.wsu.edu/

[2] Activity and Context Recognition with Opportunistic Sensor Configurations. http://www.opportunity-project.eu/

[3] Tim Van Kasteren. https://sites.google.com/site/tim0306/datasets

Among the early publications on HAR, the CASAS project published some works using Support Vector Machines (SVM). Originally as one of the most robust classifiers, SVM also became one of the most popular classification techniques to deal with HAR along with HMM and K-Nearest Neighbors algorithms. Unlike sequential models, SVMs are more flexible regarding sample representation and have consistently shown a superior performance across several datasets such as the Opportunity project, the Van Kasteren dataset and the UCI HAR dataset [14, 13, 17]. However, they feature some limitations on working with large datasets as the temporal cost of training an SVM increases rapidly with the number of samples [7].

A powerful classification technique that has not been much used in HAR, despite its wide success in other applications, are the Neural Networks. This may be due to a failed attempt in 2012 of using an artificial neural network to classify different activities, which ended up performing considerable worse and less consistently than other techniques [13]. Another explanation may be found in that current neural networks, with many more hidden layers than the network models used in 2012 had, require way more samples than the current datasets contain. However as of recently, since 2016, we have seen papers attempting to use Convolutional Neural Networks and Recurrent Neural Networks with a higher degree of success [16, 1]. Overall, in spite of their shortfalls, deep neural networks have been proven extremely effective in handling different problems, and they are also highly flexible with the representation of the samples.

We can also find several other classifiers that have seen more limited use. Among these, we can mention the Random Forests, which have only been used in the multiresident dataset of the CASAS project and in the Opportunity project [20, 15]. While Random Forests are not so widely used and they are less popular than other classification techniques such as SVM, they exhibit a strong performance in the research works that experimented with this technique. There are also clustering methods, such as K-Nearest Neighbors, which were originally introduced in 2014, but they have not kept up with the latest classification techniques [12, 8]. Finally, Naive Bayes classifiers have also been studied in several papers, although they are commonly used with the aim of comparing different feature extraction techniques or providing a baseline method to which compare other classifiers to [24, 8].

CHAPTER 3
Problem statement and data processing

In this chapter we will first give a formal definition of the problem to solve. Then we will introduce and analyze the smart home datasets used in the project. Later we will explain why it is necessary to process data prior to the application of the classification techniques and how this is done. Finally, the chapter presents various methods used to extract features from the datasets.

3.1 Overview of the problem

The problem addressed in this project is the recognition of human activity in smart homes. A smart home comes equipped with an array of sensors, which are distributed across the house and measure different factors such as movement, human interaction with objects like doors, or room temperature. When a sensor triggers, it emits what we call a *sensor event*. The sensor event contains information about the sensor, the measured value and the time it was triggered. Sensor events are stored in a file, as it may be seen in Figure 3.1, where each row describes a single sensor event. The annotations at the end of some of the sensor events represent the activity that was being done by the person monitored within the home when the sensor event was emitted (e.g. `Bed_to_toilet`). All the sensor events comprised between the begin and end of an activity correspond to the triggered sensors during the performance of such activity.

We tackle the recognition of home daily activities as a classification problem, where the collected sensor events are to become the samples and the activities to recognize are the classes. We work under the assumption that only one participant is at home at a time or, in the case of several residents, that they are all working in the same activity. The classification techniques must also be able to work with streaming sensor data; that is, when the sensor events are directly obtained from the array of sensors through a data stream rather than from a file.

Formally, a sensor event $e = \langle d, t, s, v \rangle$ is a four-valued tuple which includes the date d, the time t, the sensor identifier s, and the value measured by the sensor v. A dataset may be interpreted as a list of N sensor events $E = \{e_0, e_1 ..., e_N\}$ in chronological order. Datasets also contains annotations from a set of recognized activities A. Each sensor event $e \in E$ is uniquely associated with one of the activities $a \in A$.

```
49  2010-11-04 05:40:43.642664 M003 OFF Sleeping end
50  2010-11-04 05:40:44.223548 M003 ON
51  2010-11-04 05:40:45.939846 M005 ON
52  2010-11-04 05:40:46.310862 M003 OFF
53  2010-11-04 05:40:51.303739 M004 ON Bed_to_Toilet begin
54  2010-11-04 05:40:52.342105 M005 OFF
55  2010-11-04 05:40:57.176409 M007 OFF
56  2010-11-04 05:40:57.941486 M004 OFF
57  2010-11-04 05:43:24.021475 M004 ON
58  2010-11-04 05:43:26.273181 M004 OFF
59  2010-11-04 05:43:26.345503 M007 ON
60  2010-11-04 05:43:26.793102 M004 ON
61  2010-11-04 05:43:27.195347 M007 OFF
62  2010-11-04 05:43:27.787437 M007 ON
63  2010-11-04 05:43:29.711796 M005 ON
64  2010-11-04 05:43:30.279021 M004 OFF Bed_to_Toilet end
65  2010-11-04 05:43:34.261135 M005 OFF
```

Figure 3.1: Excerpt of sensor events belonging to the DLR dataset [9]

Figure 3.1 shows an excerpt of a listing of sensor events, each described with the date, the time, the sensor identifier and the value of the sensor. We can observe some sensor events are annotated with the begin/end of an activity. As we mentioned before, all sensor events comprised within the sensors labeled as `Bed_to_toilet begin` (sensor event #53) and `Bed_to_toilet end` (sensor event #64) are associated with the activity `Bed_to_toilet`.

We can also observe in Figure 3.1 that some sensor events are not comprised within the triggered sensors of any activity (e.g. the three sensor events before the event annotated with `Bed_to_toilet begin`). There are two ways to handle this: to associate all of these sensor events to a virtual class called "Other", or outright ignore them. In this project we chose the latter.

3.2 CASAS datasets

In this paper we will use the public datasets provided by the CASAS Project. In these datasets the status of the residents and their physical surroundings are captured through sensors and the environment is acted upon via controllers. As many other researchers we utilize CASAS datasets for Human Activity Recognition (HAR) tasks with the purpose of identifying the suitability and accuracy of various classification techniques to particular smart home environments.

The main reason why we chose CASAS over other datasets is that most researchers tend to create their own private datasets, so there are few publicly available data sources. However, CASAS offers a higher number of smart home environments as they vary in how data is collected, the types of sensors that are used and the measurable aspects of the sensed environments. While there are some other public repositories besides the CASAS and Van Kasteren datasets, they either use types of sensors that we do not consider in this paper, such as wearable sensors; they attempt to classify simple actions rather than more complex activities; or may lack annotations indicating the associated activities.

Specifically, we used four different smart home environments from the CASAS datasets:

- **Ordered Activities (OA)** [23]. This is an experiment in which data is collected from 23 volunteers. Participants were asked to perform five activities in a specific given order so as to be able to replicate common household activities. The activities were performed in the following order: 1) `Make a phone call`; 2) `Wash hands`; 3) `Cook`; 4) `Eat`; and 5) `Clean`. Volunteers were given precise instructions on how to do the activities. The number of sensor events per class may be seen in Table 3.1:

Activities	Number of sensor events	Relative frequency (%)
Cook	2172	35.15
Clean	1607	26.01
Make a phone call	1037	16.78
Eat	882	14.27
Wash hands	481	7.78

Table 3.1: Number of sensor events in the OA dataset

- **Ordered Activities with Errors (OA-E)** [23]. Data in this experiment is recollected from 20 volunteers. The experiment records the same five activities as in OA but this time participants were asked to commit a mistake while doing the activity. As in OA, volunteers are instructed on how to perform each activity, including the 'mistake'. For example, for the activity `Make a phone call`, participants would first dial an incorrect number before the correct one. The number of sensor events per class may be seen in Table 3.2:

Activities	Number of sensor events	Relative frequency (%)
Cook	2338	45.32
Make a phone call	1019	19.75
Eat	770	14.93
Clean	574	11.13
Wash hands	458	8.88

Table 3.2: Number of sensor events in the OA-E dataset

- **Interweaved Activities (IwA)** [25]. This dataset stems from an experiment with 20 volunteers where participants were requested to perform the following eight activities once in any order: `Fill medication dispenser`, `Watch DVD`, `Water plants`, `Answer the phone`, `Prepare birthday card`, `Prepare soup`, `Clean`, and `Choose outfit`. Participants were also allowed to start one activity before finishing the activity that was currently being done. This dataset was meant to compare the performance of classifiers when activities are not done in any specific order. As a side note and due to problems with the data of two participants, we will only work with the data of 18 residents. The number of sensor events per class may be seen in Table 3.3:

Activities	Number of sensor events	Relative frequency (%)
Clean	1611	23.46
Prepare soup	1066	15.53
Watch DVD	976	14.21
Water plants	975	14.20
Prepare birthday card	766	11.16
Fill medication dispenser	550	8.01
Choose outfit	519	7.56
Answer the phone	403	5.87

Table 3.3: Number of sensor events in the IwA dataset

- **Daily Life Recording (DLR)** [9]. Unlike the other datasets, data in DLR does not stem from a synthetic experiment, but instead captures data of only one volunteer when doing her daily routines. Specifically, the volunteer is an adult woman who lives in the house alone, although she is allowed to receive visits from her relatives. This dataset is meant to contain more *realistic* data. Instead of performing a detailed set of activities, the participant was simply asked to carry out her daily life while the research team collected the sensor data. This dataset recognizes 11 activities, which are manually annotated by the research team. The activities of DLR are `Meal_Preparation`, `Relax`, `Eating`, `Work`, `Sleeping`, `Wash_Dishes`, `Bed_to_Toilet`, `Enter_Home`, `Leave_Home`, `Housekeeping`, and `Resperate` (usage of a portable electronic device that promotes slow, deep breathing). The number of sensor events per class may be seen in Table 3.4:

Activities	Number of sensor events	Relative frequency (%)
Relax	374743	47.28
Meal Preparation	292158	36.86
Sleeping	63792	8.05
Work	17637	2.23
Eating	16651	2.10
Housekeeping	10938	1.38
Wash Dishes	10594	1.34
Enter Home	2041	0.26
Leave Home	1954	0.25
Bed to Toilet	1483	0.19
Respirate	571	0.07

Table 3.4: Number of sensor events in the DLR dataset

Each dataset is recorded within a so-called testbed. A testbed is a home adapted with different sensors in order to record any activity within it. CASAS owns several testbeds each one named after cities such as Kyoto, Aruba or Cairo. The OA, OA-E and IwA datasets are built from the Kyoto testbeds, while the DLR dataset is built from the Aruba testbed. The Kyoto testbed replicates a two-story house although more sensors were added to IwA, while the Aruba testbed replicates a single-floor bungalow.

3.2.1. Datasets OA and OA-E

The OA and OA-E datasets use the same testbed with the same sensor configuration shown in Figure 3.2. In this figure, the blueprint on the left is the ground floor of the house where activities take place and the map on the right shows the cabinet located above the kitchen sink. These two datasets feature the following sensors:

- 26 motion sensors, referred to as M-01 through M-26 in Figure 3.2.
- nine sensors associated to different type of objects identified with names starting by 'I'. Some of the objects are located in the kitchen cabinet while others in the living room:
  - within the kitchen cabinet we can find:
    - middle shelf: a medicine cabinet (I-06).
    - bottom shelf: boxes with ingredients such as oatmeal (I-01), raisins (I-02), sugar (I-03) and kitchen utensils such as a bowl (I-04) and measuring spoon (I-05).
  - a pot (I-07) located under the cabinet.
  - a phone (indicated with an asterisk in the dataset but not shown in Figure 3.2, and a phone book (I-08), both in the living room.
- one sensor attached to the door of the kitchen cabinet (D-01).
- one sensor attached to the stove (burner) in the kitchen (AD1-A).
- one sensor attached to the hot water faucet in the kitchen (AD1-B) and another one to the cold water faucet (AD1-C).

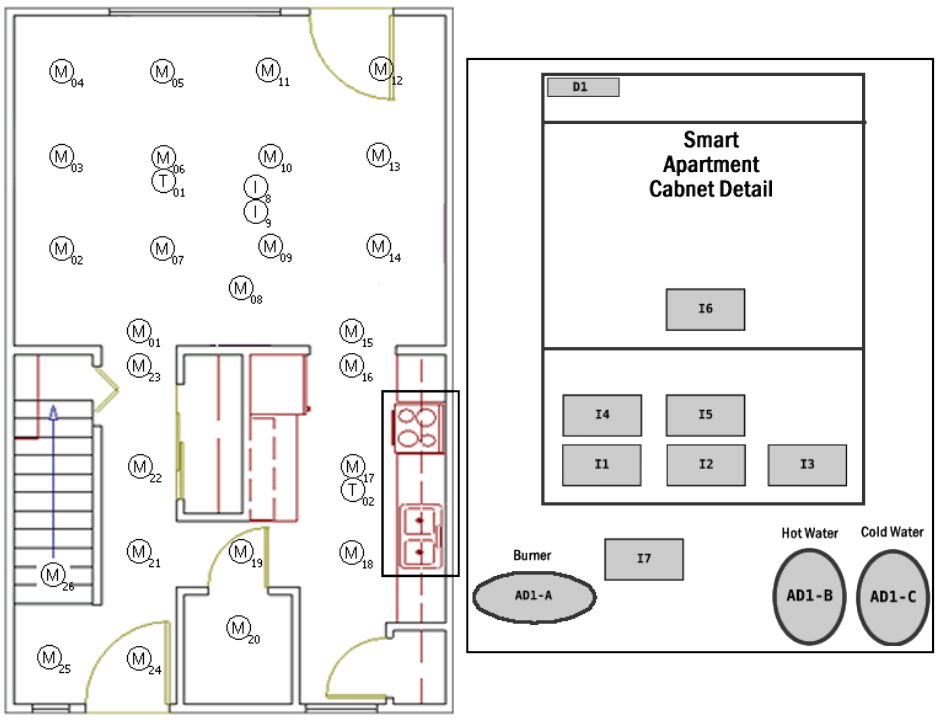


Figure 3.2: Sensor distribution for the testbed in OA and OA-E [23]

We can also observe in Figure 3.2 that there is another type of sensor, labeled as T, which represents temperature sensors. The researchers made the conscious choice to not include the temperature sensors although the motivation behind it is unknown.

3.2.2. Dataset IwA

In the case of the IwA dataset, the blueprint of the house (see Figure 3.3) is the same as in datasets OA and OA-E but the set of sensors is slightly different. The blueprint of Figure 3.3 is the ground floor of the house, and the sensors readings in the IwA dataset are:

- the same 26 motion sensors as described for OA and OA-E datasets plus another sensor, M-51, located in the bottom right corner which denotes a kitchen supply closet.
- ten sensors attached to different objects. One of them, the phone in the living room, is identified as sensor 'P-01', while the other nine are identified with initial letter 'I'. These are attached to the following objects:
  - although not explicitly displayed on the figure, the kitchen features a cabinet with a middle shelf (the medicine cabinet) that has two sensors (I-06 left and I-04 right), and a bottom shelf with two sensors (I-01 left and I-02 right); there is also a sensor I07 associated to the kitchen counter.
  - sensors I-03 and I-05 are attached to the right and left TV shelf, respectively.
  - I-08 is the phone book sensor, located at the living room.
  - it is unclear what the object I-09 refers to; by analyzing the data we found that this sensor triggers only when participants perform the activity `Prepare birthday card`.
- five sensors attached to the doors of the house, labeled in Figure 3.3 from D-07 to D-12, which indicate if the door is opened or closed. Sensors referred in the figure as D-01 to D-06 are attached to the doors in the house, but the recorded sensor events do not report readings on these sensors. The collected sensor events associated to doors are the following:
  - the kitchen's cabinet door (D-07).
  - the freezer's door (D-08).
  - the fridge's door (D-09).
  - the microwave's door (D-10).
  - the door of the closet at the bottom right of the figure (D-11). We checked the dataset and noticed this sensors triggers when participants are either performing the `Water plants` activity or the `Clean` activity, so we may safely assume this closet contains cleaning products and gardening tools.
  - the door of the closet behind the kitchen (D-12). This sensor only triggers during the activity `Choose outfit`, so it clearly functions as a wardrobe.
- one sensor attached to the stove in the kitchen (AD1-A).
- one sensor attached to the hot water faucet in the kitchen (AD1-B) and another one to the cold water faucet (AD1-C).
- two temperature sensors, identified as T-01 and T-02, which measure the temperature of the living room and the kitchen, respectively. These sensors are shown in the blueprint of the ground floor shown in the right part of Figure 3.3.

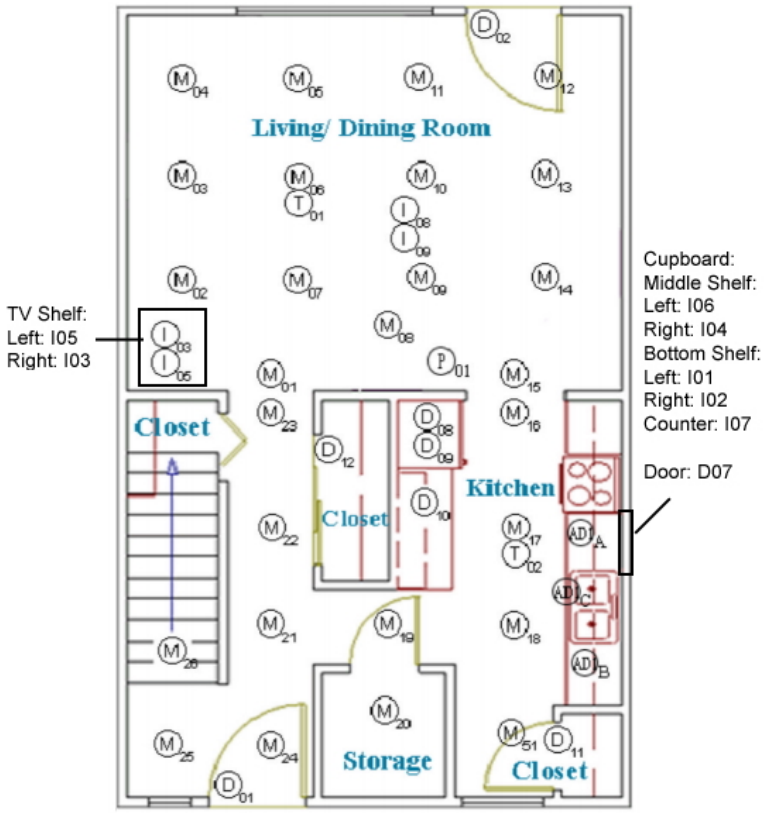


Figure 3.3: Sensor distribution for the testbed in lwA (ground floor) [25]

3.2.3. Dataset DLR

Finally, the DLR dataset uses a completely different testbed that features a different layout as shown in Figure 3.4. This dataset is named Aruba in the CASAS project:

- 31 motion sensors, labeled in the figure from M-01 up to M-31. Out of these, sensors M-019, M-020, M-007, M-024 and M-027, which are shown by their shades in Figure 3.4, have a detection radius larger than the rest of motion sensors. These far-reaching sensors aim to measure the presence of the resident in one of the rooms of the house.
- four sensors attached to the doors of the house, where three of them are doors to the outside (D-001, D-002 and D-004), and one of them is a door located within the house (D-003).
- five temperature sensors, each one of them located in a different room:
  - the bedroom (T-001)
  - the living room (T-002)
  - the kitchen (T-003)
  - the main hall (T-004)
  - the office (T-005)

The DLR testbed also reports two additional readings indicating if the participant leaves or enters the house. However, the provided specifications do not indicate whether these readings are added manually or they are also captured from sensors.

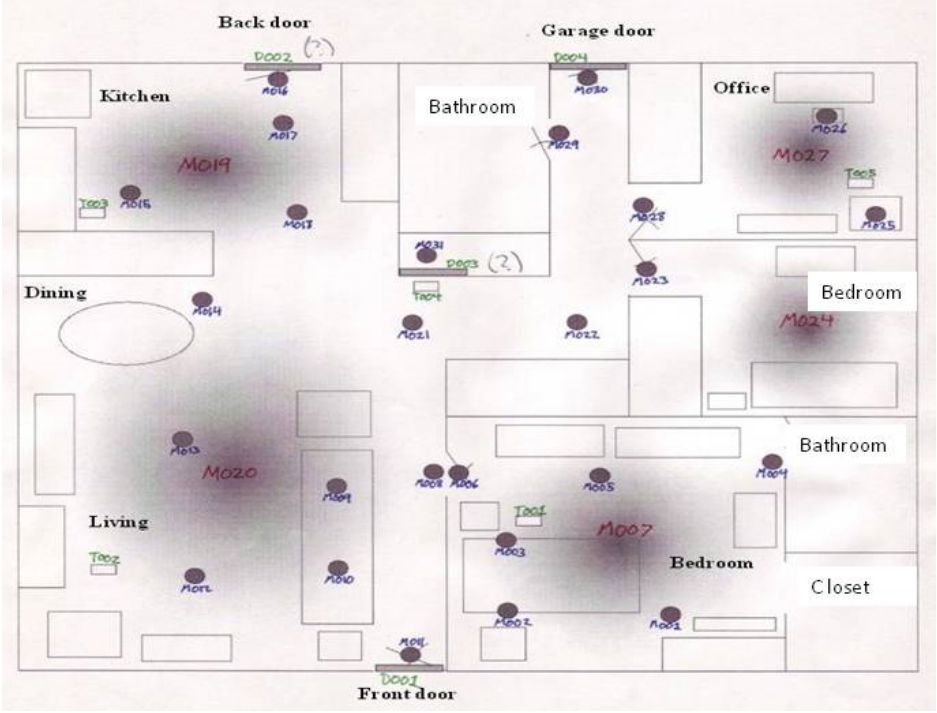


Figure 3.4: Sensor distribution for the testbed in DLR [9]

3.2.4. Similarities and differences between datasets

When choosing the datasets to work with, our aim was to test different levels of difficulty, using data coming from controlled experiments to data obtained from daily human activity. In the latter, the time of the activities will be the usual time a person performs an activity (e.g. sleep at night) but in the controlled experiments the time of the activities is not necessarily an indication of the activity nature since participants perform each activity only once either in a predetermined order or in any order at the participant's choice. We also decided to bound the scope of the work and consider only data sets that collect data from one participant at a time, or that two activities may not be performed simultaneously (not to be confused with interleaved, where one activity may be halted in order to advance another one). This is to simplify the theoretical model in which our classifiers are based upon.

The OA dataset is gathered from a controlled environment where participants were asked to do the five activities in a same given order. The only source of variation between participants is in how they perform the activities themselves, but this is minimized as participants are given detailed instructions on how to perform each activity. Given the simplification of the contextual model of the activities in OA, this experiment may not bring a significant insight towards activity recognition but serves as a basis to test the predicatability of linear models.

The experiments with OA-E and IwA attempt to relax some of the controlled aspects in the OA experiment and to better reflect real human activity: in the OA-E dataset participants force mistakes while performing the activities, and in IwA participants are allowed to change the order of the activities. As such, we can study the effects of each restriction in the effectiveness of each classifier and how much they distort the results. The DLR dataset, however, collects data that come from daily human activity, i.e., from a participant performing the same tasks she would normally do in her daily life. In this case, the participant may or may not do the same activities in the same order, make mistakes while performing them or repeat the same tasks at given days and times. This dataset would allow us to see how different classifiers will perform in real scenarios.

3.3 Data segmentation

As commented in section 3.1, the classification techniques must be able to perform activity recognition with sensor events stored in a file as well as in an online or streaming fashion. In the first case, the sensor events of a file unambiguously identify the performed activity. In the case of streaming data, data are continuously generated by the different sensors. However, a single sensor event does not contain enough information on its own to allow a classifier to make a reasonable prediction of the activity associated to it. This is why we need to divide up the sequence of sensor events in groups in order to form the samples that will be used to feed the classification algorithms. This process is called *data segmentation*.

The aim of data segmentation is to divide data into segments or *sensor windows* which are appropriate for activity recognition. Then we will extract features on each sensor window to build the feature vectors that will be used as instances for the learning and testing phases of the classification algorithms. There are several ways of creating these sensor windows such as picking the set of sensor events comprised within a fixed duration or time slot [18]. Some works also suggest using unsupervised methods to create the sensor windows for the classifier [22].

The method we chose instead is the *sensor event based windowing*, which consists in creating windows with a fixed number of consecutive sensor events [18]. Given an ordered list of events E and one single event $e_i \in E$, we define a window of size k as the list of events $\{e_{i-k+1}, e_{i-k+2}, \ldots, e_i\}$. For example, in the sequence of sensor events shown in Figure 3.1, if we select a window size of six sensor events, we would extract one window ranging from sensor event 49 to 54, another window from sensor event 50 to 55 and so on.

Sensor event based windowing also determines that the associated activity to a sensor window is the activity that the last sensor event of the window is associated to, independently of the other sensor events. Therefore, in the previous examples both sensor windows would be associated with the activity `Bed to Toilet` since the last sensor event in the window [49-54], (i.e, 54), and the last sensor event in the window [50-55], (i.e., 55) are both associated to such activity, even though these two windows include sensor events not associated with the activity `Bed to Toilet`.

We note that the definition of a sensor-event windowing does not prevent a sensor event from appearing in more than one window. It is also in our best interest to extract the largest amount of samples that follow this definition from the datasets. To do so, given a window size k, we extract as many segments of k consecutive sensor events as possible. In the case of obtaining the sensor events from a file, where we would have a finite list of N sensor events, $\{e_1, e_2, ..., e_N\}$, we would extract $N - k$ segments, starting at $\{e_1, e_2, ..., e_k\}$, $\{e_2, e_3, ..., e_{k+1}\}$ and ending at $\{e_{N-k}, e_{N-k+1}, ..., e_N\}$. In contrast, in the case of handling sensor events from a data stream, we need to store the latest k sensor events, and each time a new sensor event is received we will replace it with the oldest sensor and build a new sensor window out of the currently stored events. This approach consisting in creating a new sensor window each time one sensor event is advanced at a time is known as a *sliding window*.

Note that applying this type of windowing will not affect the prior probabilities of the activities. This is because there will be one window created per each sensor event, with the exception of the first k events. As such, given enough sensor events, the effect of having k fewer samples will be insignificant on the relative frequency of each activity.

3.4 Feature extraction

In this section we present the features to extract on the sensor windows explained in the preceding section. There exists a large variety of features that can be extracted from the information contained in a window. The ones we propose in this project are applicable in all the considered datasets and require the least possible data pre-processing. The basic features we chose are:

1. Day of the week of the last sensor event of the window;
2. Number of seconds elapsed between midnight and the last sensor event of the window;
3. Number of seconds elapsed between midnight and the first sensor event of the window;
4. Seconds elapsed between the first and last sensor events of the window;
5. A count of the times that each sensor is triggered within the window.

The first feature, **the day of the week**, is aimed to differentiate activities according to the week day they take place. It may be expected that daily routines of a resident would change across the days of the week, especially between weekdays and weekends. We note that this feature represents a category, not a continuous value. Since categorical variables are incompatible with the continuous features we will work with in this project, we will use *one-hot encoding* to convert them to numerical variables. In one-hot encoding, we split a categorical feature into a set of dummy features that take on an integer value 0 or 1. For this feature, we used 7 dummy features, each one representing one day of the week. A dummy feature takes on the value 1 if it corresponds to the day of the week in which the activity was performed or 0 otherwise. For example, Monday would be represented as [1,0,0,0,0,0,0], Tuesday as [0,1,0,0,0,0,0] and so on.

The second and third feature, **number of seconds elapsed from midnight to the last and first sensor event**, respectively, aim to measure the time of the day at which the last and first event in the window occurs. These are used to differentiate activities according to the time of the day they are performed. We believe this is specially important for certain activities that require to be done at given times of the day, such as, for instance, the activity of *sleeping*. We expect these two features to be particularly useful in the DLR dataset since the activities of the other datasets are realized at arbitrary dates and times and thereby no meaningful temporal information is retrievable about the activities of the participants from these datasets.

We note that using either the first and last event of the window provides similar temporal information. One could argue that using the last event is marginally better than using the first event since the latter may belong to another activity all together. In practice, however, the difference in seconds between both events tend to be insignificant to the number of seconds in the day. Additionally, by using any of these two features plus the seconds elapsed in the sensor window, we can derive the other one. We found in the literature that other researchers also use both of them in their own classifiers. When we started developing our code, we quickly tested how using the first sensor event, the last one or both would affect the accuracy of a Gaussian Naive Bayes. The results revealed that the accuracy was the same using either of them but slightly improved when using both. In the end, this seems to be the main reason for using both. It is also worth mentioning that the number of features in our problem is not too large so adding one more dimension does not have a big impact in the accuracy of the classifier.

The fourth feature measures **the duration of the window**. Given that the number of sensor events within a window is always constant by our data segmentation, the duration of the window also tells us how often sensors are triggered during the current performed activity. Thus, a short duration window is an indication that the activity triggers a large number of sensors. This allows a classifier to differentiate between *active* activities and *passive* activities. By *active* we mean activities that trigger sensors frequently, therefore creating windows with a smaller time span than *passive* activities, which are the ones that trigger sensors infrequently. For example, the activity `Cooking` in the OA dataset frequently triggers the item sensors connected to all the ingredients within the kitchen, so we will commonly have several dozens of sensor events in less than a minute. On the other hand, the activity `Sleeping` of the DLR dataset is likely not to trigger any sensor during its duration, leading to windows with sensor events which are minutes, even hours, apart from each other.

The last feature, **the count of sensor triggers**, counts the number of times each sensor appears within the window. We stress that we interpret a triggering when the sensor emits an event, independently of the emitted value. As such, the count for a given sensor is the number of events within the window that are emitted by such sensor. Since the total number of sensor events per window is fixed, this feature also indicates the proportion of triggers for each sensor within the window. The sensor count is used by the classifiers to identify the most frequently triggered sensors in one activity.

```
1  2008-02-27  12:43:27.416392 M08 ON  Phone_Call begin
2  2008-02-27  12:43:27.8481   M07 ON
3  2008-02-27  12:43:28.487061 M09 ON
4  2008-02-27  12:43:29.222889 M14 ON
5  2008-02-27  12:43:29.499828 M23 OFF
6  2008-02-27  12:43:30.159565 M01 OFF
7  2008-02-27  12:43:30.28561  M07 OFF
```

Figure 3.5: Sample window of sensor events belonging to the OA dataset [23]

Let's see an example on how the BASE features are extracted from a sample window as the one shown in Figure 3.5:

- The last sensor event took place on February 27, 2008, which was a Wednesday. Thus, the feature indicating the day of the week would be encoded as [0,0,1,0,0,0,0].
- The last sensor event triggered at 12:43 and 30 seconds. Therefore, the amount of seconds elapsed since last midnight is equal to $12 \times 360 + 43 \times 60 + 30 = 6930$.
- The first sensor event triggered at 12:43 and 27 seconds. Therefore, the amount of seconds elapsed since last midnight is equal to $12 \times 360 + 43 \times 60 + 27 = 6927$.
- Three seconds elapsed between the first and last sensor event of the window.
- As we can see from the sensor identifiers, sensor M-07 triggered twice, sensors M-01, M-08, M-09, M-14, and M-23 triggered once, and the remaining sensors were never triggered in this sensor window.

We want to highlight that none of these features actually use the values of the sensor events. This is a conscious choice as we wish to compare the results across datasets and we need a common set of features to do so. Since each dataset uses a different set of sensors with a different set of ranges, it is not possible to include all of them in the common representation.

Besides this basic set of features, other researchers investigated the use of additional levels of pre-processing so as to improve the samples obtained from the feature extraction. In the following we will explore other techniques to enrich the feature vector. Ultimately, we aim to have various features configurations and test the classifiers with all of them in order to check whether some configurations offer advantages over others and why.

3.4.1. Time Dependency

The concept of time dependency (TD) refers to the time elapsed between two sensor events and it is used to measure how relevant the two events are to each other [18]. When applying TD we make the assumption that two sensor events that are temporally close to each other are more likely to belong to the same activity than two sensors that are wide apart. For example, two sensor events that are milliseconds from each other are probably caused by the same action, but two sensor events separated minutes and even hours apart are more likely to be part of different activities.

For the application of the TD we will refine the feature that represents the count of times each sensor is triggered in the window. Since the idea behind sensor event based windowing is that all sensor events within the window are to provide context for the last event, we will include the temporal information that TD brings inside the count. Now, given a window w and a sensor event e, each time e appears in w, i.e. $e \in w$, instead of increasing the count by one we will increase it by the coefficient $C(e, e')$, where e' is the last event in the window w. This coefficient is a value between zero and one, which becomes exponentially smaller as the number of seconds elapsed between e and e' increases. Specifically, the formula used to define this coefficient is:

$$C(e, e') = exp(-\chi\Delta(e, e'))$$

where $\Delta(e, e')$ is the number of seconds elapsed between the events e and e' and χ is a normalization parameter. We will use the value $\chi = 2^{-3}$ in our experiments as this is reported value in the scientific literature that returns the best results [18].

3.4.2. Sensor Dependency

The idea behind sensor dependency (SD) is to measure how often a set of sensors is triggered together across several windows as an indication that such set of sensors is being used in the same activity. For example, we can imagine that the activity `Cooking` is likely to trigger the sensors of the water faucet and the burner. Therefore, it is likely they will commonly appear together in several windows. On the other hand, it is reasonable to assume that the item sensor of the TV remote is needed for the activity `Watching TV` but not for `Cooking` and so it is unlikely this sensor appears in the same window that the burner or the water faucet. And even in the case they do appear in the same window, this is a highly probable indication that the two sensors belong to two different activities.

The way sensor dependency is implemented in this work is inspired by the ideas presented in two topic-related papers. The first paper is authored by the same researchers that proposed the TD feature introduced in section 3.4.1 [18]. The second paper, developed by Nawel Yala, Belkacem Fergani and Anthony Fleury, presents some modifications over the contribution of the first paper in order to design a more flexible feature [32]. Our proposal inherits from the ideas presented in [32] but unlike this work we will use the sliding windows that result from the data segmentation process explained in section 3.3.

The sensor dependency is modeled through the creation of a data structure named the Window Mutual Information (WMI) matrix. Let S be the set of the sensors identifiers used by a dataset and $\{w_1, w_2, ..., w_N\}$ a set of N windows of sensor events. The WMI matrix has $|S|$ rows and $|S|$ columns, where each value at row i and column j shows the dependency between the sensor s_i and sensor s_j, $s_i, s_j \in S$. The values in WMI range from zero to one: a value of one indicates that sensors s_i and s_j are highly dependant, meaning they always appear together in the N windows of the dataset, while a value of zero means that the sensors never triggered within the same window.

Specifically, the value of a cell in the WMI matrix is obtained through the following formula:

$$WMI(s_i, s_j) = \frac{1}{N} \sum_{k=0}^{k=N} \delta(w_k, s_i)\delta(w_k, s_j)$$

where

$$\delta(w, s) = \begin{cases} 1 & \text{if} \quad \exists e, s \in e \wedge e \in w \\ 0 & \text{otherwise} \end{cases}$$

When adding the sensor dependency to our features we first need to calculate the WMI matrix. This is done offline using the windows that will be later used for the training of the classification models. Once the WMI matrix is obtained, we use the values of the matrix to modify the count of the sensor triggers. Now, given a window w, a sensor event $e = \langle d_e, t_e, s_e, v_e \rangle \in w$ and the last sensor event of the window, $e' = \langle d_{e'}, t_{e'}, s_{e'}, v_{e'} \rangle$, each time the sensor s_e appears in w, instead of increasing the count of s_e by one we will increase it by the coefficient $WMI(s_e, s_{e'})$

Note that it is possible to apply both the TD and SD features to the count of sensor triggers as the two techniques are compatible to each other. In the scenarios where we want to apply so, given a window w and a sensor event $e = \langle d_e, t_e, s_e, v_e \rangle$, each time e appears in w, instead of increasing the count by one we will increase it by the product $C(e, e') \cdot WMI(s_e, s_{e'})$.

3.5 Sample Generation

Now that we have defined several features that can be extracted from the sensor windows we will explain the sample generation that our models will use. For each window of sensor events, we will obtain one sample made up of a set of features. Each sample will also have one associated class. The associated class of the sample is the activity associated to the last sensor event of the window. In this paper we will test four different configurations of features:

1. **BASE** configuration: consists of
   (a) Day of the week of the last sensor event of the window;
   (b) Number of seconds elapsed between midnight and the last sensor; event of the window;
   (c) Number of seconds elapsed between midnight and the first sensor event of the window;
   (d) Seconds elapsed between the first and last sensor events of the window;
   (e) A simple count of the times that each sensor is triggered within the window.

2. **BASE + TD** configuration: same set of features as the **BASE** configuration plus the Time Dependency applied to the count of sensor triggers;
3. **BASE + SD** configuration: same set of features as the **BASE** configuration plus the Sensor Dependency features applied to the count of sensor triggers;
4. **BASE + TD + SD** configuration: same set of features as the **BASE** configuration but applying both Time Dependency and Sensor Sependency to the count of sensor triggers.

As a result, given a dataset made out of readings from S different sensors and for all the set of features above, we have a feature vector with $S + 10$ dimensions:

- seven dimensions for the day of the week using the one-hot encoding conversion explained in section 3.4;
- one dimension for the number of seconds elapsed between midnight and the last sensor event;
- one dimension for the number of seconds elapsed between midnight and the first sensor event;
- one dimension for the duration of the window;
- S dimensions for the counts of each sensor.

CHAPTER 4

Classification Techniques for Human Activity Recognition

In this chapter we will examine the performance of different classifiers on the task of human activity recognition using the datasets and features presented in chapter 3. First the chapter starts with a quick review of which metrics we will use to evaluate each classifier. Then we will go through each classification technique, explaining in detail how it works and how it fared. The classifiers to be commented are naive Bayes, Support Vector Machine (SVM) and Random Forests.

4.1 Evaluation setting

As we detailed in Chapter 3, the collection of samples we will work with in this project is a set of labeled samples, that is, the class of the samples is known, and hence all the studied classifiers are examples of supervised learning. The labeled samples are partitioned into a training set and a testing set. A supervised classifier uses the training set to build a model from (training phase) and once this is complete the learnt model is used to classify the samples of the testing set (testing phase).

We ran the classifiers using the datasets and features as proposed in Chapter 3, and measured their *accuracy*, *precision*, *recall* and *F-score*. Since the ideal window size depends on both the dataset and features extracted from the data, we tested the classification techniques along a range of window sizes and calculated the average values across them. In order to define the accuracy, precision, recall and F-score we first review the terms of true positives, false positives, true negatives and false negatives:

- true positives (TP): samples that belong to a class and that the classifier correctly classifies as belonging to such a class;
- true negatives (TN): samples that do not belong to a class that the classifier correctly classifies as not belonging to such a class;
- false positives (FP): samples that do not belong to a class and that the classifier incorrectly classifies as belonging to such a class;
- false negatives (FN): samples that belong to a class and that the classifier incorrectly classifies as not belonging to such a class.

Let A be the set of activities in a dataset (e.g. for the OA dataset it would be $A = \{\texttt{Make a phone call}, \texttt{Wash hands}, \texttt{Cook}, \texttt{Eat}, \texttt{Clean}\}$ and $|A| = 5$), N the total number of samples and TP_a the number of correctly classified samples for an activity a (e.g. a = `Wash hands`). The classification accuracy is the percentage of correctly classified samples over all the activities:

$$accuracy = \frac{1}{N} \sum_{a \in A} TP_a$$

In unbalanced datasets, the accuracy measure may be highly influenced by the correct classification of the most populated classes in the dataset. This is the case, for instance, if we have only 10 samples of the activity `Clean`, and half of them correctly classified, and we have 200 samples of `Cook`, and all of them are correctly classified. The obtained accuracy would be equal to 205/210, which reflects how well the classifier performs globally, but it does not reflect how it performs individually on each class. This means that if the dataset has unbalanced classes and a classifier obtains a high accuracy we may not distinguish if it is due to correctly classifying samples from all classes or instead only samples from the most common classes.

In this paper we will also evaluate the classifiers using the *F-score*. The F-score shows the average percentage of correctly classified samples per class and is calculated on the *precision* and *recall* for a given activity a:

$$precision_a = \frac{TP_a}{TP_a + FP_a} \qquad recall_a = \frac{TP_a}{TP_a + FN_a}$$

Precision shows how many samples that the classifier returns as belonging to the class or activity a actually belong to activity a. *Recall* gives an idea of completeness of the classifier for a particular activity a; that is, it measures how many samples out of all the samples that belong to class a are actually correctly classified as belonging to class a.

The *F-score* of an activity a is interpreted as a weighted average of the precision and recall of such activity. Specifically, in this work we will use the balanced F-score, which gives equal weight to both values and it is formally defined as:

$$\text{F-score}_a = \frac{2 \cdot precision_a \cdot recall_a}{precision_a + recall_a}$$

Then, to obtain the global *F-score* we compute the *weighted* average of the individual F-score for each activity a. The weight of an activity a in *F-score* is proportional to the number of samples of such activity a in respect to the total amount of samples. This is known as a weighted-averaged F-score:

$$\text{F-score} = \sum_{a \in A} \frac{N_a}{N} \cdot \text{F-score}_a$$

where N_a is the number of samples of class a and N is the total amount of samples overall. Note that this measurement is also biased to the performance of the classes with a higher number of classes in the case of unbalanced datasets. Therefore, it raises the question of why considering the *F-score* as well. In a nutshell, accuracy gives a precise view of the correctly classified samples but it fails to account for the non-correctly classified ones. That is, accuracy gives more importance to TP and TN while *F-score* is used when knowing what the classifier has done wrong (FP + FN) is also relevant.

Additionally, the *F-score* emphasizes that the importance of having a high value for *precision* or *recall* does not exclude having a low value for the other. For example, it is clear that an algorithm that returns a low *precision* and high *recall* for a class means that the classifier is unable to correctly identify the samples belonging to such class. However, a class with high *precision* and low *recall* means that the classifier does not necessarily identify all the samples of that class but instead classifies most samples it receives as belonging to such class. To make sure that a classifier correctly identifies a class it must have both high values of *precision* and *recall*.

Furthermore, in order to evaluate in detail how classifiers handle each individual class we will examine the generated confusion matrices. Samples in the diagonal of the matrix are the ones that are correctly classified. The row cells of the confusion matrix associated to an activity a, excluding the diagonal cell, shows the percentage of samples of class a that are classified in each of the possible classes; i.e, the set of FN for activity a. And the values of the column of activity a represent all the samples of class a that are incorrectly classified as such; i.e., the set of FP of class a.

Finally, we will evaluate the classifiers using cross-validation with 10 blocks. This means that, given a classifier and a set of samples, we divide the samples into 10 blocks. This division is done at random as to make sure each block has a balanced number of samples of each class and avoid over-training. Then we make one run per block, where one block is used for the evaluation of the model and the remaining ones for training. The results obtained from the evaluation are then the average of the partial results obtained at each run. Unlike simply partitioning data into one set for training and another for testing, this method makes a better use of the data and it is less prone to over-fitting[1].

4.2 Naive Bayes

Naive Bayes (NB) is one of the simplest probabilistic models, typically used as a baseline to compare other models against [11]. A NB classifier is one of the fastest models to train, it is highly scalable and the number of features barely affects the time taken to train and run the model, while still being reasonably accurate compared to other more complex classification techniques.

As the name suggests, Naive Bayes is an example of a Bayesian classifier. A Bayesian classifier attempts to obtain the class c^* of a sample $x \in X$ using the posterior probability. The posterior probability is the conditional probability that a given sample x belongs to the activity c. This is expressed as $P(c|x)$. If we obtain the posterior probability for all classes then the classifier makes the prediction that the correct class is the one with the highest posterior probability:

$$c^* = \arg\max_{c \in C} P(c|x)$$

Given that the posterior probability is a conditional probability, we are able to decompose it by using Bayes' rule:

$$c^* = \arg\max_{c \in C} P(c|x) = \arg\max_{c \in C} \frac{P(x,c)}{P(x)} = \arg\max_{c \in C} P(x,c)$$

Note that, we are simply trying to find the class which maximizes the posterior probability and that $P(x)$ is a constant factor for all classes. Therefore, calculating c^* comes down to calculating the joint probability $P(x,c)$ since obtaining the class that maximizes the joint probability is the same as the one that maximizes the posterior probability.

[1] Scikit-learn. Cross-validation: evaluating estimator performance. scikit-learn.org/stable/modules/cross_validation.html. Accessed: 2020-05-21

Naive Bayes is a type of generative classifier since it performs this classification by obtaining the joint probability. This joint probability can be split as:

$$c^* = \arg\max_{c \in C} P(x, c) = \arg\max_{c \in C} P(x|c)P(c)$$

In our work, the prior probability of a class, $P(c)$, is obtained from the data collected in each dataset. The likelihood that the sample x belongs to class c, $P(x|c)$, is computed by the NB classifier. Naive Bayes makes the assumption that all the features within the feature vector x are independent. That is, given a feature vector composed of n features, $\forall i \in [1, n]$, $P(x_i|c, x_1, ..., x_n) = P(x_i|c)$. Therefore, we may simplify the expression as:

$$c^* = \arg\max_{c \in C} P(x|c)P(c) = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(x_i|c)$$

The likelihood of the individual features is calculated by using a probability distribution, which will depend on the nature of the features. For continuous variables a Gaussian distribution is usually the best choice but for categorical variables the most popular probability distributions are Multinomial NB or Bernoulli NB.

4.2.1. Design

The implementation of Naive Bayes is carried out with the API Scikit-learn 0.22.2 [6]. Specifically, we chose to use a Gaussian Distribution to model our features. In a Gaussian distribution, we assume that the likelihood for each combination of feature and class follows a normal distribution: $p(x_i|c) \sim \mathcal{N}(\mu_{c,i}, \sigma_{c,i})$:

$$p(x_i|c) = \sqrt{\frac{1}{2\pi\sigma_{c,i}^2}} exp(-\frac{(x_i - \mu_{c,i})^2}{\sigma_{c,i}^2})$$

During the training phase, the probability distributions are obtained in terms of the mean and the standard deviation according to the maximum likelihood estimation method. A Gaussian distribution has the advantage of being compatible with all the features proposed in our work, such as the day of the week represented through a one-hot encoding, the sensor count which is represented by an integer number, and the sensor count with dependencies which use real numbers instead.

While training the Gaussian model we also must introduce variable smoothing over the variance of each distribution. Variable smoothing is introduced as to avoid over trained classifiers, but more importantly it avoids computational errors. For example, in a Gaussian Distribution the variance cannot be equal to zero, otherwise it would cause a division by zero. However, it may be the case that with a particular set of training samples we obtain a variance of zero, or a too small value to be represented with a floating point representation, thus leading to execution errors.

The variable smoothing included in the Scikit-learn library is achieved by increasing the variance of the modelled Gaussian distributions. Specifically, our classifier will model for every feature a Gaussian distribution for each of the classes in the data set. For each feature, the variance of all classes is increased by adding a factor equal to the product of the largest variance for the given feature and a parameter $\lambda \in [0, 1]$. In our case we use a value of $\lambda = 10^{-7}$ as it prevents over fitting while causing minimum disturbance to the outcomes.

4.2.2. Results

In this section we will evaluate the performance of the Naive Bayes classifier to the four datasets introduced in the preceding chapter. Table 4.1 shows the accuracy obtained for every combination of dataset, feature configuration and window size, as well as the averaged accuracy across all window sizes for each combination of dataset and set of features. Likewise Figure 4.1 displays the average F-score across all window sizes for each combination dataset+features.

This section is structured as follows. First, we will discuss global trends, specifically how the application of the features sensor dependency (SD) and time dependency (TD) affects the results. Then we will comment on the impact of the various window sizes across all datasets. Finally, we will provide a thorough discussion on the relative performance on each dataset, OA, OA-E, IwA and DLR, commenting on the efficiency in terms of global accuracy and F-score as well as studying the confusion matrices for each configuration.

Dataset	Set of Features	Window Sizes					
		10	20	30	40	50	Average
OA	Base	0.7309	0.7830	0.8355	0.8470	0.8488	0.8090
	Base + TD	0.6661	0.6682	0.6705	0.6844	0.7051	0.6789
	Base + SD	0.6087	0.6806	0.6887	0.7243	0.7680	0.6941
	Base + TD + SD	0.5399	0.5674	0.5822	0.6087	0.6390	0.5874
OA-E	Base	0.7998	0.8274	0.8131	0.8269	0.7971	0.8128
	Base + TD	0.7751	0.7759	0.7777	0.7771	0.7901	0.7792
	Base + SD	0.6937	0.7554	0.7759	0.7536	0.7545	0.7466
	Base + TD + SD	0.6220	0.6591	0.6796	0.6974	0.7310	0.6778
IwA	Base	0.6259	0.6131	0.5534	0.5247	0.5062	0.5647
	Base + TD	0.6123	0.6019	0.5911	0.5912	0.5894	0.5972
	Base + SD	0.4138	0.4689	0.4680	0.4560	0.4584	0.4530
	Base + TD + SD	0.3429	0.4228	0.4784	0.4940	0.4920	0.4460
DLR	Base	0.8929	0.8855	0.8695	0.8538	0.8386	0.8680
	Base + TD	0.6251	0.5600	0.5348	0.5268	0.5238	0.5541
	Base + SD	0.5996	0.6855	0.7411	0.7731	0.7871	0.7173
	Base + TD + SD	0.4161	0.4411	0.4485	0.4564	0.4619	0.4448

Table 4.1: Accuracy (%) obtained with naive Bayes by dataset and set of features

I. Impact of the feature configurations

We start by analyzing the impact of the features TD and SD across all datasets. By observing the numbers in Table 4.1 and Figure 4.1, we realize that using TD and SD affect negatively both the accuracy and the F-score of the models. This effect worsens when both dependencies are used simultaneously. This may be due to unforeseen consequences of the implementation of these dependencies into the feature vector.

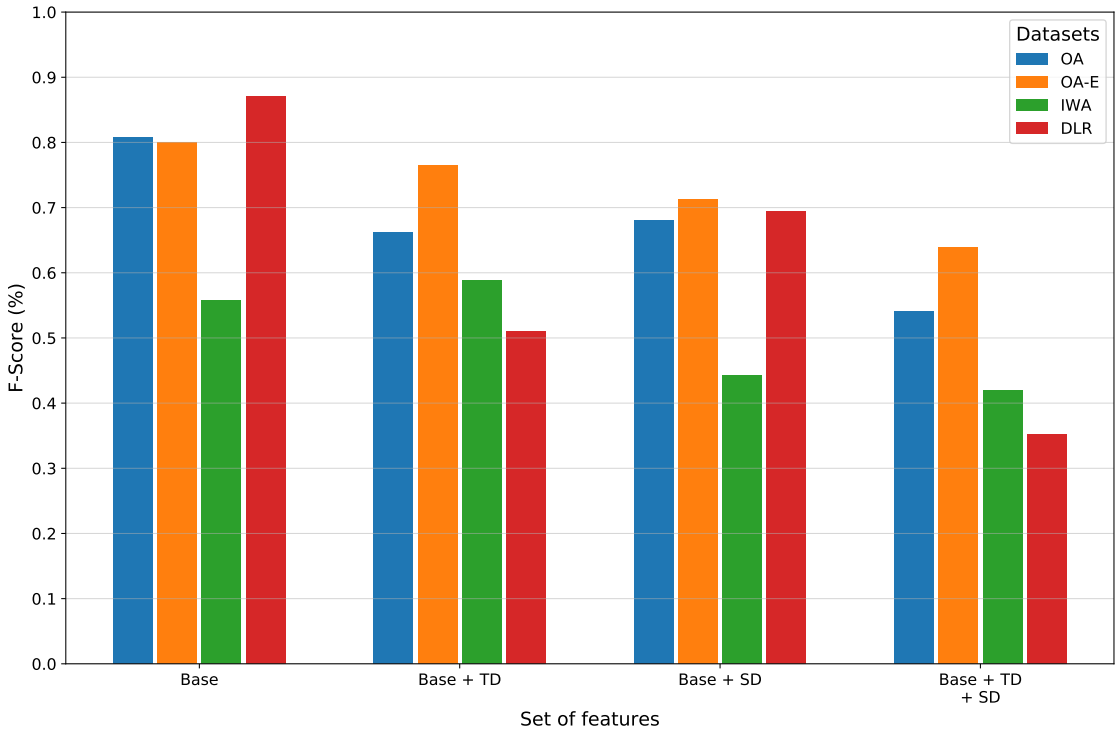


Figure 4.1: F-Score obtained with a Naive Bayes classifier by dataset and set of features

We recall that calculating the value of TD and SD consists in multiplying a factor between zero and one to the count of the sensors. As a result the average sensor count decreases independently of the class the sensor belongs to. Additionally, the average sensor count is already close to 0 due to the number of sensor events in a window is usually very small compared to the number of existing sensors. The resulting effect is that, for a given sensor, the application of the TD and SD will likely yield a very similar average sensor value over all classes, thus preventing the classifier from correctly differentiating them.

Interestingly enough it seems that the effect of TD or SD is also highly dependent on the dataset. The OA and DLR datasets benefit more from using SD than TD than OA-E and IwA datasets. The large number of sensors of DLR is a factor that clearly benefits this dataset from using the feature SD. Still, we can observe in the figures of Table 4.1 and Figure 4.1 that the Base configuration is the one that provides the best results across the board. Exceptionally, applying TD is generally more effective than the Base configuration in the IwA dataset. We can also see in Table 4.1 that when compared with the Base set of features, using TD diminishes the negative effect of increasing the window size. The purpose of TD is to reduce the impact of the sensor events of a window that belong to an activity other than the activity associated to the window. And it is in the IwA dataset where participants are more prone to keep changing the activity they perform, and therefore it is likely that the sensor windows of IwA contain a higher number of events that belong to other classes. As such, it is reasonable to assume that this is the scenario where applying TD outweighs the negative effect over the data.

II. Impact of the window size

According to Table 4.1, the accuracy of OA and OA-E datasets increase with the window size. In the OA dataset the accuracy increases along with the increment of the window size, while the OA-E reaches its highest accuracy at a window size of 40. On the other hand the IwA and DLR datasets behave completely the opposite, their accuracy decreases as the window size increases.

Before explaining the impact of the window size in the datasets, we first need to introduce the concept of **noisiness** in a sensor window. We define *noise* as the sensor events within a window associated to an activity other than the window's activity. A high noise level in a window affects the features of the sample, making it harder to differentiate from samples of other classes. Ideally, the more noisy the windows, the harder is for the classifier to differentiate the activities from each other.

Overall, we expect the accuracy to increase as the window size increases. Larger windows promote sensor counts to be less disperse, carry more information and increase the variance that the feature "number of seconds between the first and last sensor events" brings. However, the larger the window the higher the chance that sensor events from other activities are included within a window, therefore increasing their noise and making it harder to classify.

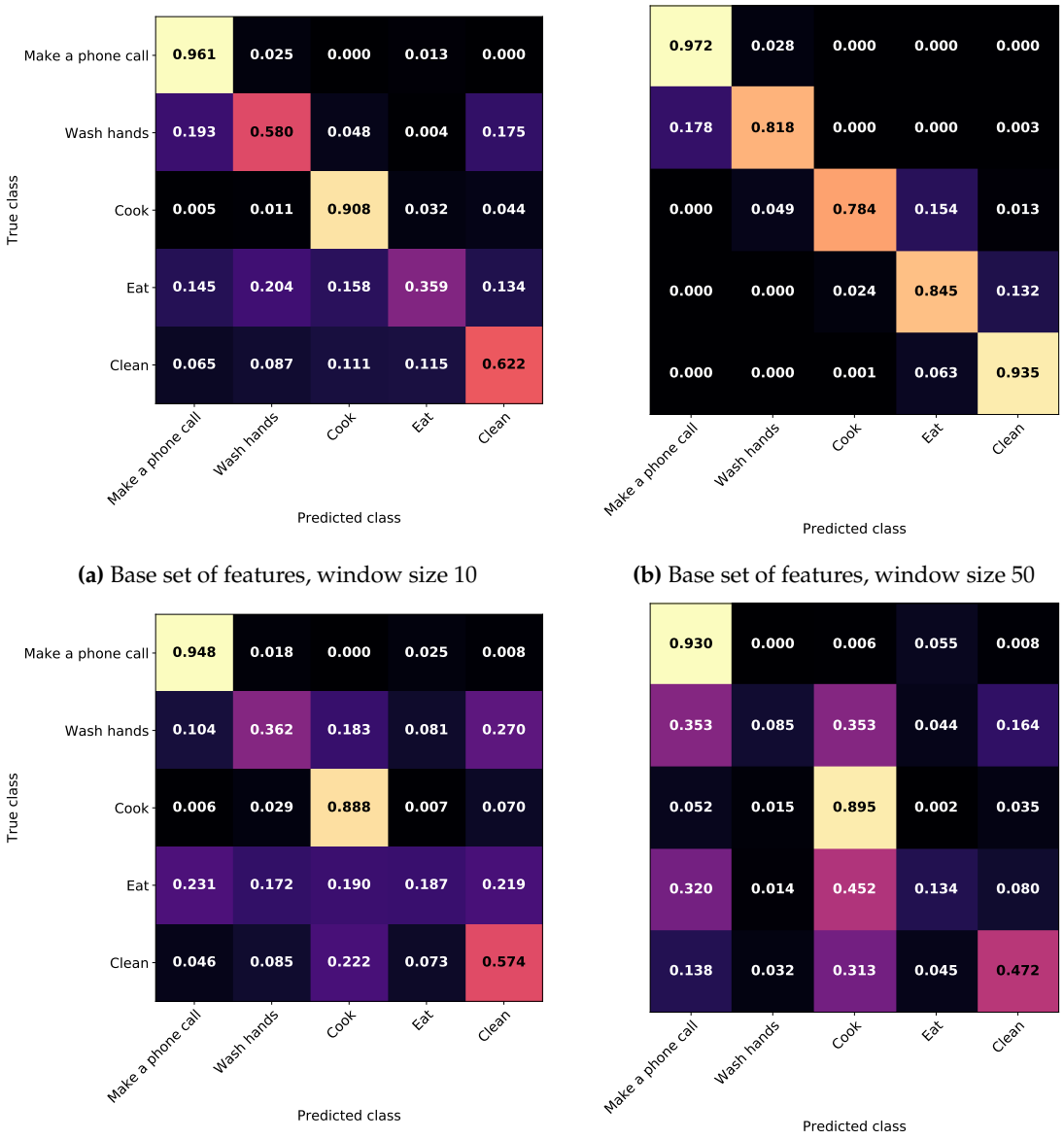
On the one side, the least amount of noise is found in the windows extracted from OA and OA-E. In these datasets, the activities are completed in one go and in a specific order. Therefore, the chance that a window has sensor events belonging to other activities is low, and even if sensor events from other activities appear is easy to predict from which activity they come from. For example, a window associated to the activity `Wash Hands` may have sensors associated to the activity `Wash Hands` or `Make a phone call` (given that `Make a phone call` is ordered before `Wash Hands`), and therefore the sensor events coming from other windows are easy to predict and differentiate. Consequently, these two datasets benefit from using a larger windows.

On the other side, the DLR and IwA datasets produce the noisiest windows. In DLR although activities are usually completed in one go, the order at which they are completed is not predefined. For example a window associated to the activity `Relax` may comprise sensors associated to any of the 11 possible activities instead of only 2. As a result the windows in DLR will contain a higher level of noise than the OA and OA-E datasets. On the other hand, the activities in the IwA are not completed in one sitting, and instead the participants tend to change quickly between one activity and another, resulting in very noisy windows. As a result, DLR and IwA benefit from small windows as this diminishes the amount of noise in them.

III. Analysis of the OA and OA-E datasets

As explained in section 3.2, the data in OA and OA-E come from the same five activities, which are `Make a phone call`, `Wash hands`, `Cook`, `Eat` and `Clean`. These are the only datasets in which participants perform the activities in a specific given order (the particular order is shown in the enumerated activities above). Thereby OA and OA-E gather data from two controlled environments and the only difference between them lies in that residents in OA-E deliberately commit a mistake when doing the activity.

Figure 4.1 shows the F-score for OA (the blue bar) and OA-E (the orange bar). We can observe the F-score values are almost the same for the two datsets when using the Base configuration and significantly higher for OA-E under the other feature configurations.



(a) Base set of features, window size 10

(b) Base set of features, window size 50

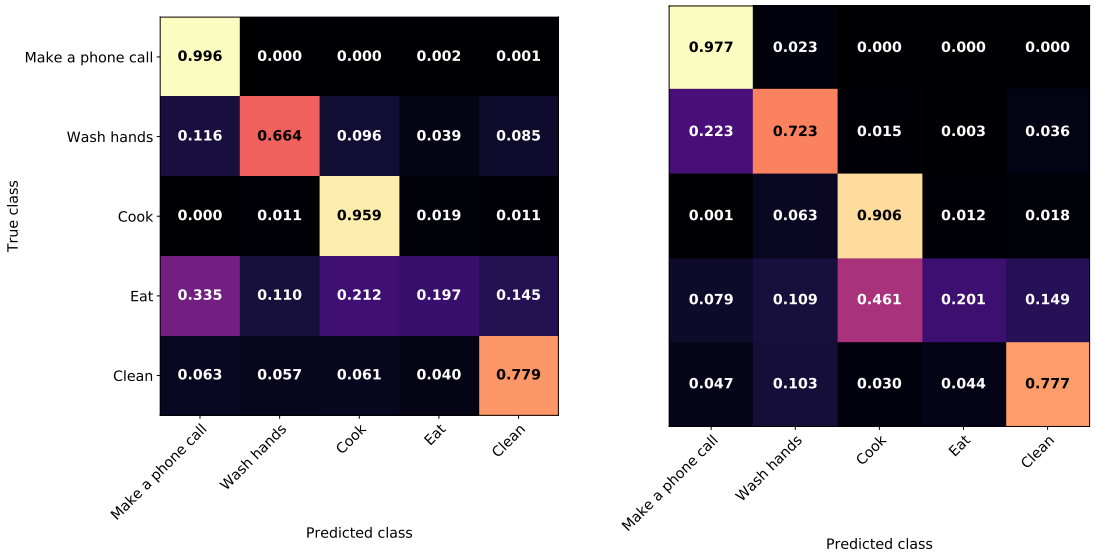
(c) Base + TD set of features, window size 10

(d) Base + SD set of features, window size 10

Figure 4.2: Confusion Matrices of applying naive Bayes to the OA dataset

These values are also consistent with the average accuracy shown in Table 4.1, where the accuracy for the Base configuration is practically the same in both datasets and the accuracy of OA-E is slightly higher for the rest of configurations. The committed mistakes in OA-E may increase the sensor counts of the most discriminative sensors, allowing samples to be classified more easily.

Figures 4.2 and 4.3 show the confusion matrices of OA and OA-E, respectively. Comparing figures 4.2a and 4.2b of OA we can see that the activities `Wash Hands`, `Eat` and `Clean` (second, fourth and fifth row respectively) are better recognized in the window of size 50 and overall the values of the diagonal are higher in 4.2b than in 4.2a. When samples of one class are consistently getting classified as samples of another class, such as it happens with `Wash hands` getting recognized as samples of `Make a phone call`, it means that the likelihood for both classes is very similar and consequently the prior probability of the classes becomes a determinant factor. In both OA and OA-E the prior of the activity `Make a phone call` is higher than `Wash hands` (see Tables 3.1 and 3.2).



(a) Base set of features, window size 10

(b) Base set of features, window size 40

Figure 4.3: Confusion Matrices of applying naive Bayes to the OA-E dataset

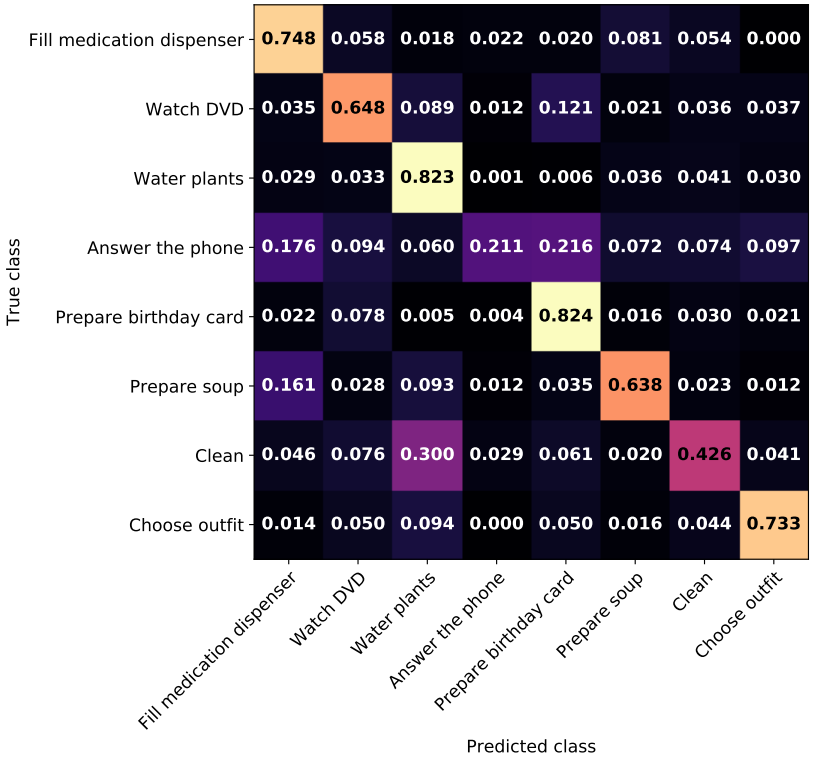
Similarly, the accuracy in OA-E increases with large size windows (see the confusion matrices in Figures 4.3a and 4.3b). Unlike OA, in OA-E the classifier struggles to correctly classify samples of activity `Eat` even using large size windows. We can also observe that the number of correctly predicted samples of classes such as `Make a phone call`, `Cook` and `Clean` is greater in OA-E than in OA, thus justifying the higher values of the overall accuracy and F-score in OA-E.

Finally, figures 4.2c and 4.2d show the confusion matrices of the dataset OA when using using the features TD and SD respectively. Adding dependencies make it more difficult to correctly classify samples from all activities, as the values in the diagonal are lower here compared to the Base configuration (see Figures 4.2a and 4.2b). We also see that when using dependencies, specially SD (Figure 4.2d), a significant number of samples from all classes are classified as `Cook`, shown by the amount of samples present in the third column in the matrix. This shows that the classifier struggles telling samples from each class apart and chooses to tag them as `Cook` as this is the activity with the highest prior probability (35% in the OA dataset, according to Table 3.1).

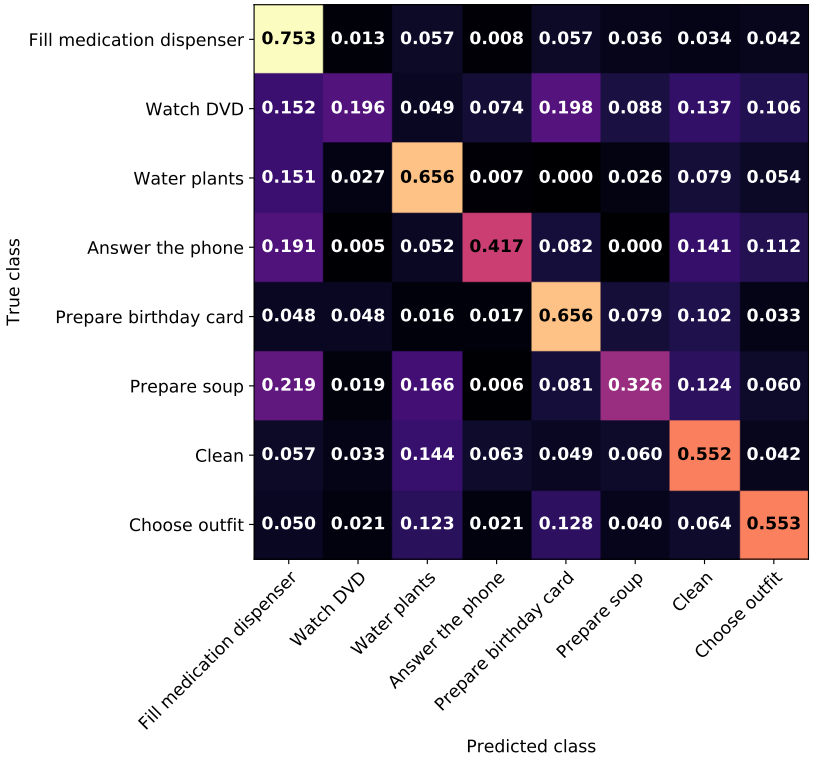
IV. Analysis of the IwA dataset

In this dataset participants were requested to complete the following eight activities in their preferred order: `Fill medication dispenser`, `Watch DVD`, `Water plants`, `Answer the phone`, `Prepare birthday card`, `Prepare soup`, `Clean` and `Choose outfit`. Participants were also encouraged to stop one activity and resume it later.

The worst accuracy values shown in Table 4.1 correspond to the IwA dataset. Particularly, the configurations Base, Base+SD and Base+TD+SD feature the lowest accuracy among all datasets. Figure 4.1 also shows that the F-score of IwA (green bars) is the lowest for all configurations, with the exception of configurations that use TD, which boast the second lowest. As mentioned in the section about the window size, IwA is the dataset with the noisiest windows, and therefore it is expected that the classifier struggles with samples from this dataset.

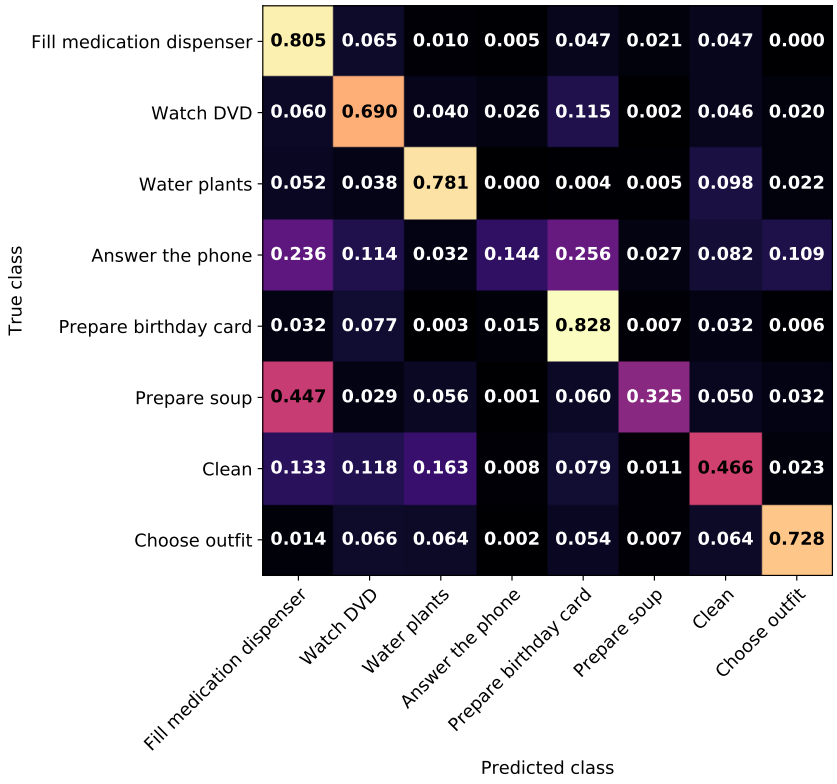


(a) Base set of features, window size 10



(b) Base set of features, window size 50

Figure 4.4: Confusion Matrices of applying naive Bayes to the IwA dataset



(c) Base + TD set of features, window size 50

Figure 4.4: Confusion Matrices of applying naive Bayes to the lwA dataset (cont.)

Figures 4.4a and 4.4b show the confusion matrices of lwA, when using the Base configuration, for window size 10 and 50, respectively. Looking at the values of the diagonals in both matrices, we can generally affirm that increasing the window size worsens the results. This is not the case, however, of the activity `Answer the phone`. At window size 10, `Answer the phone` is the class with the fewest correctly classified samples, as evidenced by the fact that samples are found evenly across the row rather than accumulated in the diagonal. This is consistent with the fact that Naive Bayes, as a generative classifier, struggles with classes that have a low prior probability, which is the case of the activity `Answer the phone` (see Table 3.3).

Interestingly, the proportion of correctly classified samples of `Answer the phone` increases with window size 50. This is explained because the classifier is able to use the additional information in the windows to correctly identify samples of this class. However, this seems to apply only to `Answer the phone` (from value 0.211 in Figure 4.4a to 0.417 in Figure 4.4b) and `Clean` (from 0.426 in Figure 4.4a to 0.552 in Figure 4.4b).

The two worst performing classes with window size 50 are `Watch DVD` (second row with value 0.196) and `Prepare soup` (sixth row with value 0.326). Since these two activities have relatively high prior probabilities, it is unlikely that the cause of their misclassification is due to prior probability alone. However, these two activities can be easily done in the background while performing other activities and so they are the most interweaved of all. This in turn increases the noisiness of their associated windows even further, making them hard to tell apart from other classes.

When we examine the confusion matrix of the configuration Base+TD with window size 50 in Figure 4.4c, we see that overall the values of the diagonal are higher than in 4.4b. We also find several activities in which the majority of their samples are still misclassified, meaning that their values in the diagonal are under under 0.500. These activities are `Prepare soup` (sixth row), `Answer the phone` (fourth row) and `Clean` (seventh row). It is unlikely that these misclassifications are only due to Naive Bayes favoring the classes with highest prior probabilities. For example, most samples of `Prepare soup` are getting classified as samples of `Fill medication dispenser`. This is evidenced as the value in the first cell of the sixth row in Figure in 4.4c is higher than the cell in the diagonal. However `Prepare soup` has a higher prior probability than `Fill medication dispenser` (see Table 3.3). In this case, we believe that adding dependencies is actually altering sensor counts differently and it is mostly affecting sensor counts of the most identifiable sensors of these activities.

In general, we can say that the naive Bayes classifier performs worse in lwA than in OA and OA-E. However, adding the time dependency comparatively lowers the loss of performance. We can thus conclude that adding TD to the feature configuration benefits this dataset, albeit not enough to reach the performance obtained by the other datasets.

V. Analysis of the DLR dataset

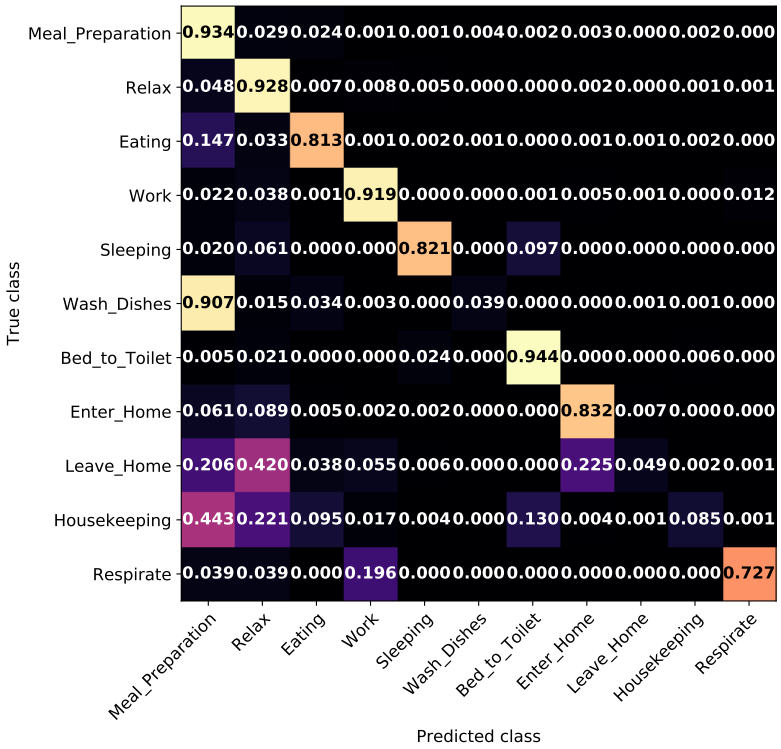


Figure 4.5: Confusion matrix for the DLR dataset using Base features and window size 10

As we detailed in Chapter 3, DLR is a dataset made up of 11 activities, done by a single participant: `MealPreparation`, `Relax`, `Eating`, `Work`, `Sleeping`, `Wash Dishes`, `Bed to Toilet`, `Enter Home`, `Leave Home`, `Housekeeping` and `Resperate`. As in lwA, participants are allowed to perform the activities in any order. DLR is the only dataset with no instructions on how a person must complete an activity.

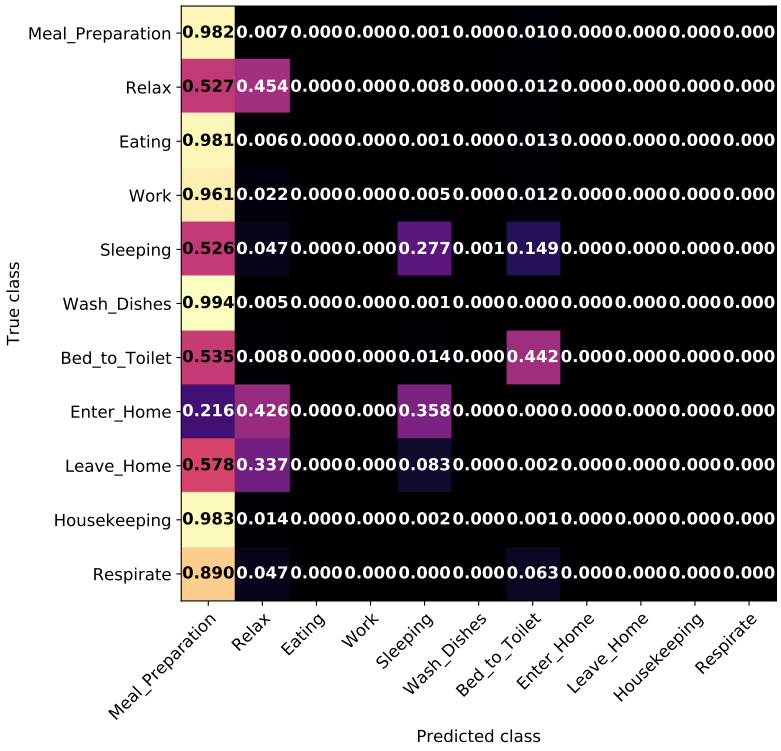


Figure 4.6: Confusion matrix for the DLR dataset using Base + SD features and window size 10

Under the Base feature configuration, DLR scores the highest F-score (Figure 4.1, red bar) and the highest accuracy (a value of 0.8680 in Table 4.1) out of all datasets and set of features configurations. It performs though badly under other configurations with the exception of Base+SD. Although this is globally the most difficult dataset due to its more relaxed environment, higher number of sensors (and therefore larger vector size) and higher number of classes, a thorough inspection of the confusion matrix reveals that the imbalanced data of DLR (when the number of data available for the different classes is different) may explain its abnormally high accuracy values.

The confusion matrix for the Base configuration (Figure 4.5) returns the best scores. The activities `Meal_Preparation` (first row) and `Relax` (second row) are the ones with the highest recall; i.e., the cell in the diagonal has a value close to one while the rest of the cells in the row have values close to zero. However, these are also the activities with the lowest precision as events from other activities are commonly categorized in one of these them, as indicated by the high values of all the cells except the ones in the diagonal in the first two columns.

The lowest classification rate is for the activity `Wash_Dishes` (sixth row) which is completely dominated by `Meal_Preparation` (first column), but it also happens for other activities such as `Leave_Home` (ninth row) and `House_Keeping` (tenth row). An activity is dominated by another one when most, if not all, samples of the such class are getting exclusively classified as belonging to the other one. For example, `Wash_Dishes` is completely dominated by `Meal_Preparation` since 90.7% of `Wash_Dishes` samples are classified as `Meal_Preparation`, while only 3.9% of its samples are correctly classified (see sixth row of Figure 4.5). As a result, several classes have values close to 0 in the diagonal, meaning that most of their samples are misclassified. However, given that the percentage of correctly classified samples of `Meal_Preparation` and `Relax` exceed 90% and that according to Table 3.4 they make up over 80% samples in the dataset, the classifier scores at least an accuracy over 72%.

The question that raises here is why the Naive Bayes classifier tends to classify most samples in these two classes. The answer is once again due to how generative classifiers favor the classes with high prior probability. DLR is a very unbalanced dataset so unless the difference in the likelihood of the classes is large enough, most samples will be classified as either `Meal_Preparation` or `Relax`. For example, it is easy to understand that `Wash_Dishes` can be mistaken with `Meal_Preparation` since both activities probably occur within the same space and trigger similar sensors. In contrast, the activity `Respirate` (last row), has the lowest prior probability of all, over 500 times smaller than `Meal_Preparation` (see Table 3.4), but the singular features of this activity make it to not get misclassified.

Another aspect to consider is why both accuracy and F-score significantly drop when adding dependencies. Figure 4.6 shows the confusion matrix for DLR when using Base+SD configuration. We can observe that that most cells in the first column have values over 0.5, while the majority of cells in the rest of the matrix have values close to zero with, meaning that activities are essentially classified as `Meal_Preparation`. This is likely because the SD makes samples from every class hard to tell apart from each other, and so Naive Bayes classifiers default to tagging them as one of the two most common classes.

4.3 Support Vector Machine

Support Vector Machine, or SVM, is one of the most used techniques both for classification and regression [27]. SVMs do not require many samples to provide adequate results and their performance can also be improved through the use of *soft-margins* and *kernels*.

Imagine we have a set of samples X, made of n features. Each sample $x \in X$ is interpreted as a n-dimensional vector within the space $\mathbb{R}^n$ and it belongs to one of two classes. We assume the samples are linearly separable, that is, the samples of each class are separable through an $\mathbb{R}^{n-1}$ hyperplane. The hyperplane is used to quickly identify the class a sample belongs to. Figure 4.7 shows a two dimensional example with samples from one of two classes, red or blue, and the hyperplane separating them.

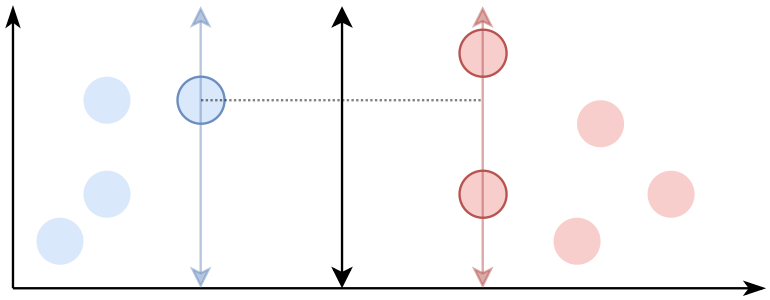


Figure 4.7: Example of an hyperplane separating a set of samples

The theory behind SVMs supports how to obtain the best possible hyperplane [27], which is one that besides correctly separating the samples of both classes maximizes the margin. The margin of the hyperplane is the distance to the closest sample, independently of its class. Therefore, if the margin is maximized, the hyperplane is equally distant to the closest samples of each class. In Figure 4.7 we see that the closest samples to the hyperplane are the ones in bold circle borders. If we move or tilt the hyperplane in any direction, one of those samples will end up closer to the hyperplane than the other two, thus decreasing the margin. As such, the hyperplane of Figure 4.7 is the optimal separation for this example.

A hyperplane is defined by a set of samples named the *support vectors*, which are found on the margin of the hyperplane itself. In Figure 4.7 those samples are the ones with the bold circle borders. The process of obtaining the best hyperplane consists in finding the *support vectors* samples and the weight each one of them will have on the hyperplane. This process is essentially an optimization problem solved through gradient descent. The process of finding the *support vectors* has a pseudo-cubic temporal complexity defined as the product of the number of samples, the number of features and the number of iterations, which may be problematic for large datasets [7].

Unlike Naive Bayes, SVMs do not work with joint, prior and posterior probabilities. Instead SVM attempts to directly predict the class of the sample from its features. This is why SVM is referred to as a discriminative classifier, as opposed to a generative classifier such as naive Bayes. While generative classifiers obtain the joint probability $P(c, x)$, discriminative classifiers base their predictions directly on the likelihood $P(x|c)$ or by directly associating a class to a sample, like SVM does. This is because generative classifiers are based on understanding the similarities between samples of the same class, while discriminative classifiers are based on identifying the differences between samples of different classes. Overall, it has been proven that discriminative classifiers take longer to train but they have a lower error rate than their generative counterparts in the task of classification [21].

So far we have introduced SVMs as linear classifiers and presented a two-class example where data is linearly separable. We can expand this concept of SVM to support multi-class problems and non-linearly separable data.

In order to deal with with multi-class problems we will simply train one SVM per pair of classes in the problem. By the end of training, we will have $C(C-1)/2$ trained SVMs, where C is the number of classes. Classifying a new sample requires applying this process to all our two-class SVMs, and then hold a vote. The class that receives more 'votes' is selected as the predicted class. This is known as a "one-against-one" approach [7]. This process is simple but it has some drawbacks. First, the amount of SVMs to train increases quadratically with the number of classes and no simple solution exists to overcome this limitation. Second, there is a chance of a tie in the vote, but this can be fixed through a simple tiebreak such as selecting one of the most voted classes at random, or through a more complex voting procedure that prevents ties.

When handling non-linearly separable problems we have to define *soft-margins* and *kernels*, although usually both are applied simultaneously. Applying *soft-margins* consists in modifying the optimization formula of the SVM to accommodate a degree of tolerance for incorrectly classified samples [27]. Consequently, these SVMs have one additional parameter $\mathcal{C}$ that regulates how 'tolerant' the SVM should be to misclassified samples. There is no algorithmic way to obtain the ideal value of $\mathcal{C}$ for a given problem, but instead it must be adjusted manually using empiric measurements.

The use of *kernels* consists in applying a transformation to the non-linearly separable data used in the SVM to a new space that, ideally, becomes linearly separable [27]. This method is highly popular because it can be implemented without affecting the temporal cost of the SVM, and because there exists a large variety of kernels. Some examples of kernels are the polynomial kernel or the hyperbolic tangent kernel [27].

4.3.1. Design

For this work we opted for using the implementation of SVM provided by the API Scikit-learn 0.22.2, which in turn is a wrapper for the LIBSVM implementation by Chang and Lin [6, 7] [2]. The implementation used has support for multi-class classification through the "one-against-one" approach described earlier, and the use of both soft margins and kernels. The kernel used for this data is a Gaussian Radial Basis Function (RBF) kernel. We chose the kernel and the parameters empirically after trying out some test runs with different configurations.

We also had to apply some additional modifications to the feature vectors before using them into the model. First of all, for SVM to work correctly, all features must be on the same scale. This is not the case for our samples, as we combine one-hot encoding features, ranging between zero and one, with a feature that measures the number of seconds in a day, which ranges between zero and 86400. Therefore, we applied a feature standarization, named *Z-Score Standarization*, where we assume the samples form a Gaussian distribution and we replace them with their z-score values in such distribution.

In order to do so we first calculate the Gaussian distribution defined by the original values and then we obtain the values of the features across the set of samples. From these values we define a Gaussian distribution and we extract its mean μ and standard deviation σ. Finally, we the z-score of each value is computed as:

$$z\text{-}score_x = \frac{x - \mu}{\sigma}$$

The mean and standard deviation of the features are obtained from the training set of samples, but these two measurements must be applied to all the samples used during the training or evaluation of the classifier.

One more transformation is needed to make sure the feature vector is normalized and so that the magnitude is equal to 1. This is because the formula that the SVM internally optimizes is based on the dot product of the different samples. This normalization does not affect how samples are interpreted by the SVM or the results, it will just reduce the temporal cost of training the model.

Finally, it is also worth mentioning that the impact of the temporal complexity of optimization of the model hinders the ability of SVM to work with large datasets. This is especially relevant when working with the DLR dataset, which comprises over $600,000$ sensor events, while the Scikit-learn documentation indicates that the SVM implementation "may be impractical beyond tens of thousands of samples"[3]. In order to overcome this limitation, if we have more then $30,000$ samples in our training set, we will choose randomly up to $30,000/|C|$ samples of each class, where $|C|$ indicates the number of classes in the dataset. This is the number of samples that we can use to train our models in a reasonable amount of time without significantly affecting our results.

[2] The paper that details the LIBSVM implementation has since been updated as the library was expanded. The updated version may be found at https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf.

[3] Scikit-learn. sklearn.svm.SVC. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2020-05-21

4.3.2. Results

Table 4.2 and Figure 4.8 show the accuracy and F-score, respectively, for the four datasets of our study and all combinations of features and window sizes. Additionally, Table 4.3 showcases a comparative summary of the differences in accuracy between Naive Bayes (NB) and SVM. Similarly to the evaluation of NB, we will start discussing the effect of adding dependencies in the datasets, followed by the evaluation of the impact of the window sizes. Finally, we will comment each dataset individually, utilizing the obtained confusion matrices.

Dataset	Set of Features	Window Size					
		10	20	30	40	50	Average
OA	Base	0.9245	0.9573	0.9566	0.9585	0.9596	0.9513
	Base + TD	0.8731	0.9126	0.9207	0.9212	0.9248	0.9105
	Base + SD	0.8654	0.9220	0.9390	0.9464	0.9539	0.9253
	Base + TD + SD	0.8143	0.8687	0.8819	0.8928	0.8979	0.8711
OA-E	Base	0.9405	0.9642	0.9563	0.9573	0.9627	0.9562
	Base + TD	0.9064	0.9364	0.9373	0.9395	0.9387	0.9317
	Base + SD	0.8954	0.9351	0.9369	0.9477	0.9435	0.9317
	Base + TD + SD	0.8568	0.8954	0.9074	0.9128	0.9179	0.8981
IwA	Base	0.8604	0.9339	0.9485	0.9517	0.9602	0.9309
	Base + TD	0.7983	0.8613	0.8668	0.8644	0.8593	0.8500
	Base + SD	0.6841	0.8346	0.8654	0.8855	0.8954	0.8330
	Base + TD + SD	0.6053	0.6977	0.7275	0.7492	0.7670	0.7093
DLR	Base	0.6948	0.7538	0.7873	0.8106	0.8349	0.7763
	Base + TD	0.6326	0.6379	0.6372	0.6372	0.6382	0.6366
	Base + SD	0.6852	0.7281	0.7569	0.7729	0.7840	0.7454
	Base + TD + SD	0.6476	0.6456	0.6466	0.6470	0.6469	0.6467

Table 4.2: Accuracy (%) obtained with SVM by dataset and set of features

I. Impact of the feature configurations

Similarly to NB, adding dependencies to the feature vectors has a negative impact in the SVM classifier as can be observed in Table 4.2 and Figure 4.8. Likewise, introducing TD and SD may accidentally make the average values of the sensors of each class be closer to each other and this may also be the cause that classes become non-linearly separable.

In the case of OA, OA-E and DLR datasets, the negative impact is actually not very big. For example, in the SVM figures of Table 4.2, the accuracy of the Base configuration in the dataset OA is 95%, while it drops to 87% when using Base+TD+SD, making a gap of about 8%. Exceptionally, the drop of accuracy is more pronounced in the IwA dataset. If we look at Table 4.2, the Base configuration returns the highest accuracy for IwA (93%) and the lowest is just under 71%, thus leading to a gap of 22%.

In conclusion, we can say that SVM performs essentially well with the Base configuration. Also, albeit SVM has proven to be a better classification technique, it is still unable to compensate for the side effects of adding the dependencies TD and SD. This is especially notable in IwA, which has the most noisy windows out of the entire data collection.

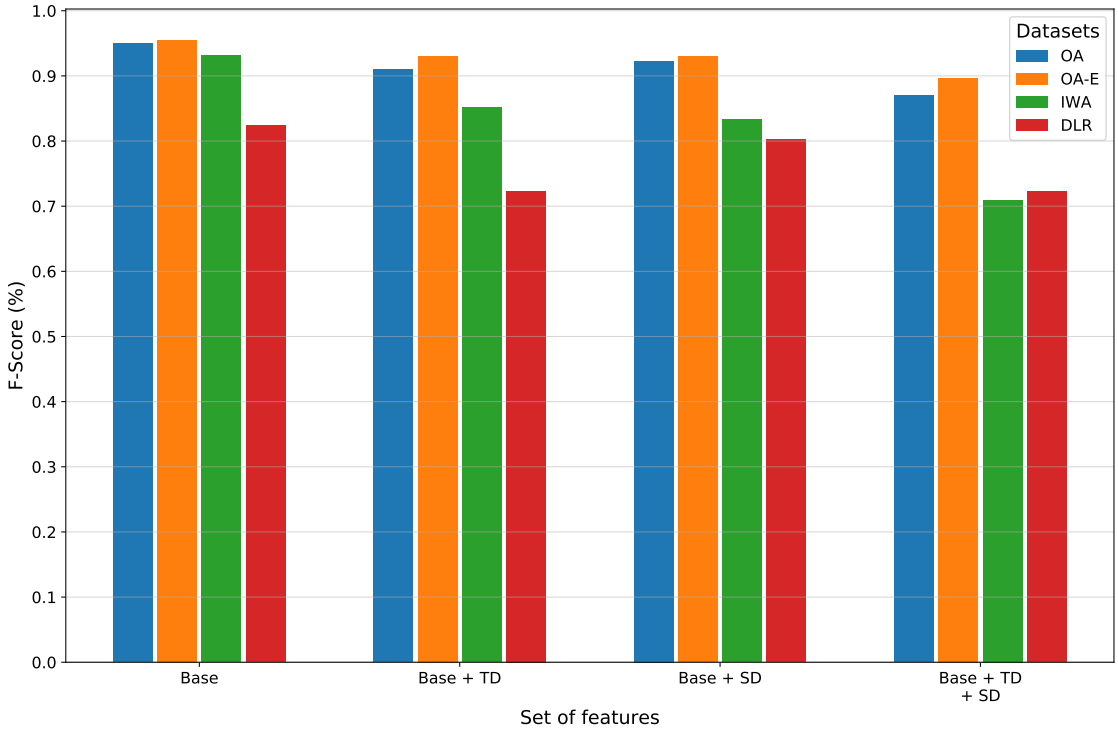


Figure 4.8: F-Score obtained with a SVM classifier by dataset and set of features

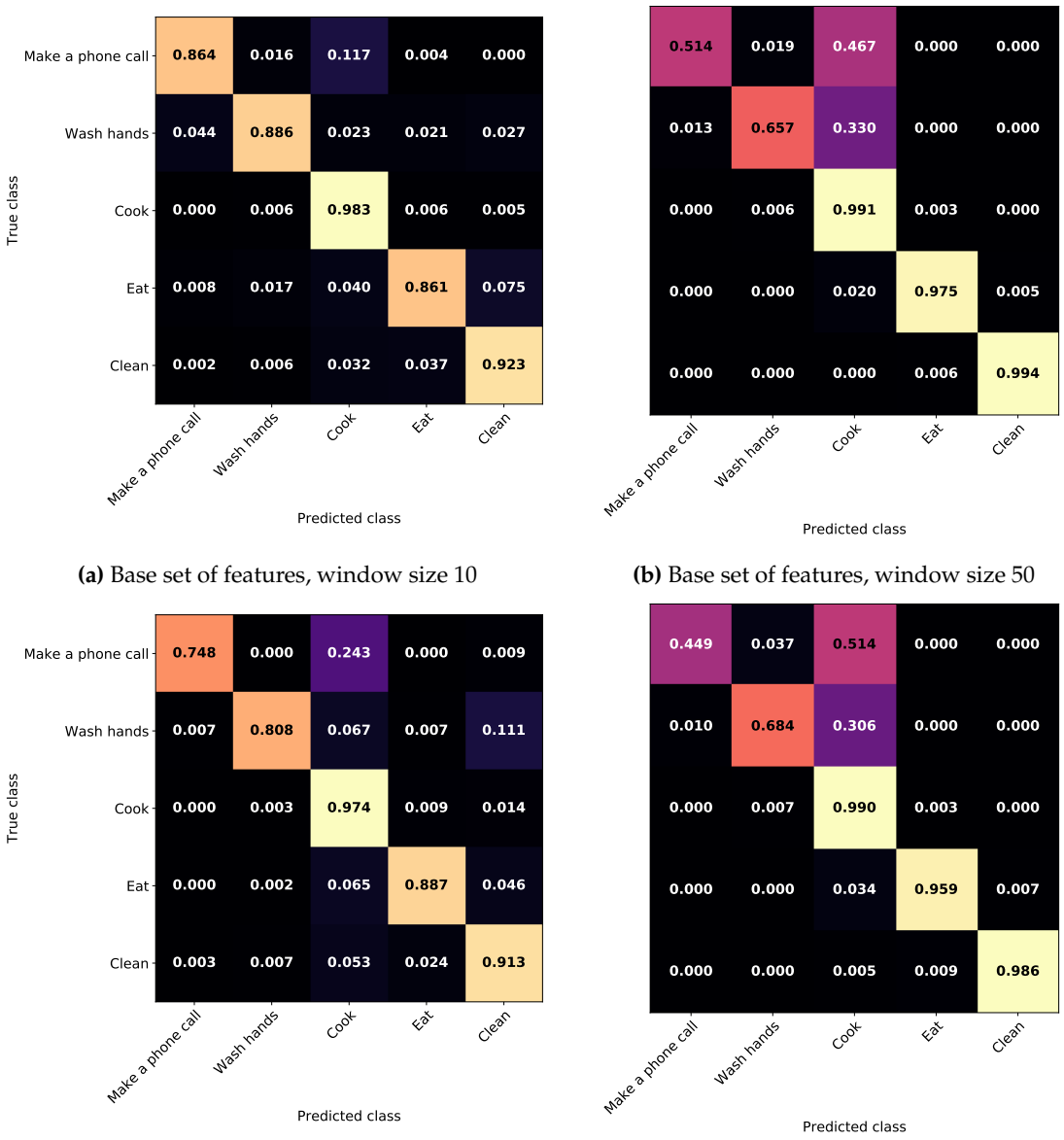
Dataset	Set of Features	Accuracy (%)		
		Naive Bayes	SVM	Improvement
OA	Base	0.8090	0.9513	**0.1423**
	Base + TD	0.6789	0.9105	**0.2316**
	Base + SD	0.6941	0.9253	**0.2312**
	Base + TD + SD	0.5874	0.8711	**0.2837**
OA-E	Base	0.8128	0.9562	**0.1434**
	Base + TD	0.7792	0.9317	**0.1525**
	Base + SD	0.7466	0.9317	**0.1851**
	Base + TD + SD	0.6778	0.8981	**0.2203**
IwA	Base	0.5647	0.9309	**0.3656**
	Base + TD	0.5972	0.8500	**0.2528**
	Base + SD	0.4530	0.8330	**0.3800**
	Base + TD + SD	0.4460	0.7093	**0.2633**
DLR	Base	0.8680	0.7763	**-0.0917**
	Base + TD	0.5541	0.6366	**0.0825**
	Base + SD	0.7173	0.7454	**0.0281**
	Base + TD + SD	0.4448	0.6467	**0.2019**

Table 4.3: Improvement in accuracy (%) from Naive Bayes to SVM

II. Impact of the window size

Unlike Naive Bayes, enlarging the window size positively benefits the SVM outcomes (this is confirmed by looking at the numbers in Table 4.2). In section 4.2.2 we discussed that large size windows increase the amount of noise –sensor events belonging to activities others than the window activity– and that this caused a major impact in IwA and DLR. However, SVM turns out unaffected by the increase of the window size. We will further analyze the confusion matrices to observe in more detail how the window size improves the results.

III. Analysis of the OA and OA-E datasets



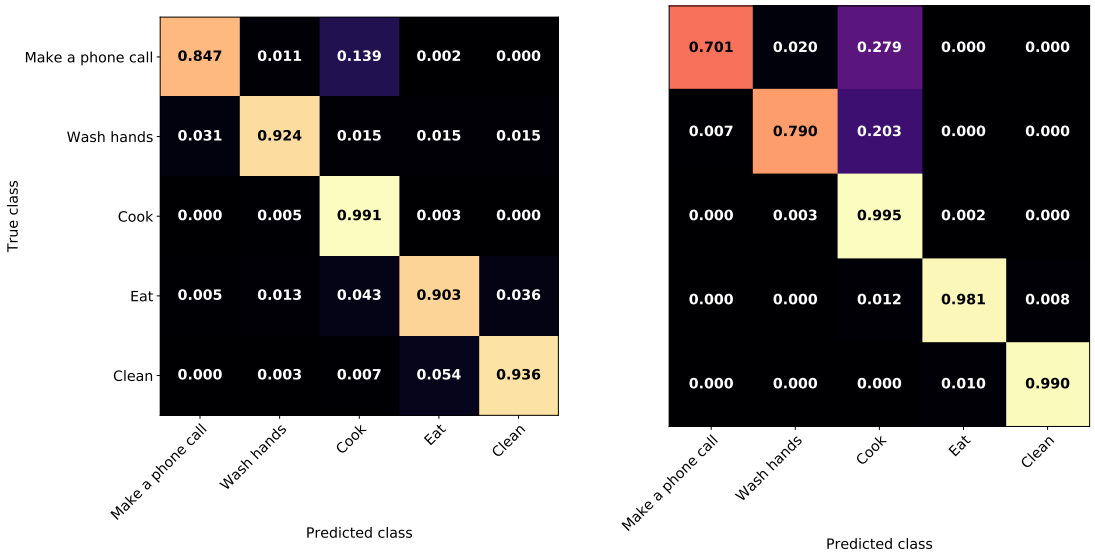
(a) Base set of features, window size 10

(b) Base set of features, window size 50

(c) Base + TD set of features, window size 50

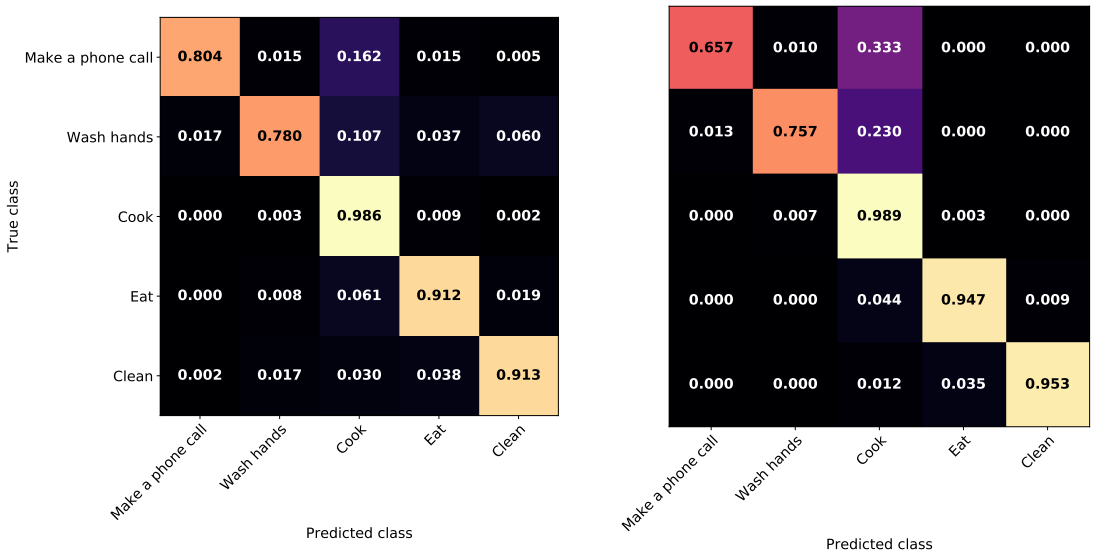
(d) Base + SD set of features, window size 50

Figure 4.9: Confusion Matrices of applying SVM to the OA dataset



(a) Base set of features, window size 10

(b) Base set of features, window size 50



(c) Base + TD set of features, window size 50

(d) Base + SD set of features, window size 50

Figure 4.10: Confusion Matrices of applying SVM to the OA-E dataset

Table 4.2 shows very high values of accuracy for both OA and OA-E datasets, reaching 95% accuracy with the Base configuration. Similarly, the F-scores for both datasets are consistently above 90%, with the exception of the Base+TD+SD configuration (see Figure 4.8, where the blue bars represent the OA dataset and the orange ones the OA-E data). While both datasets have similar scores, OA-E performs marginally better. This is more evident in Figure 4.8, where the orange bars are slightly longer than the blue ones for all feature configurations. As before, our hypothesis is that the better behaviour of OA-E is due to the induced 'mistakes' that participants make on doing the activities, which provides additional sensor events.

The confusion matrices of the OA and OA-E datasets are the following:

- Figures 4.9a and 4.9b show the matrices of applying the Base set of features to the OA dataset with a window size of 10 and 50, respectively. In the matrix with a window size 10 we see the values in the diagonal are overall high, ranging from 0.861 up to 0.983, which indicates a high rate of correctly classified samples. In the matrix with window size 50, the activities fall into one of two groups. The first two activities `Make a phone call` (first row) and `Wash hands` have a mediocre percentage of correctly classified samples, 0.514 and 0.657, respectively, and a large portion of their samples are getting classified as being of the activity `Cook`. The other three classes instead have a high percentage of correctly classified samples, above 0.97.

  We can conclude that, with small windows, SVM identifies better the whole set of activities, and the number of false positives (FP) of `Cook` is significantly lower than with a larger window. In contrast, the accuracy of some classes is higher with window size 50, thus resulting in an overall higher accuracy.

- Figures 4.9c and 4.9d show the matrices of OA when applying the the Base+TD and Base+SD feature configurations, both for a window size 50. Basically, we observe the same behaviour here as with the Base configuration for window size 10 and 50; that is, an increase in the number of FP for class `Cook` with SD as well as a lower accuracy for `Make a phone call` and `Wash hands`.

- Figures 4.10 show the matrices of the OA-E dataset. We can observe a behaviour and trend similar to OA except that in OA-E the values in the diagonal are slightly higher, especially so for the first two classes.

The reason why the SVM classifiers favors the activity `Cook` over the others may stem from activity `Cook` being the tie-braking vote (see section 4.3 for an explainattion of the SVM performance). It may also be the case that in a two-class SVM that has `Cook` as one of its classes, somehow `Cook` secures more votes during the voting procedure, thus giving `Cook` an unfair advantage.

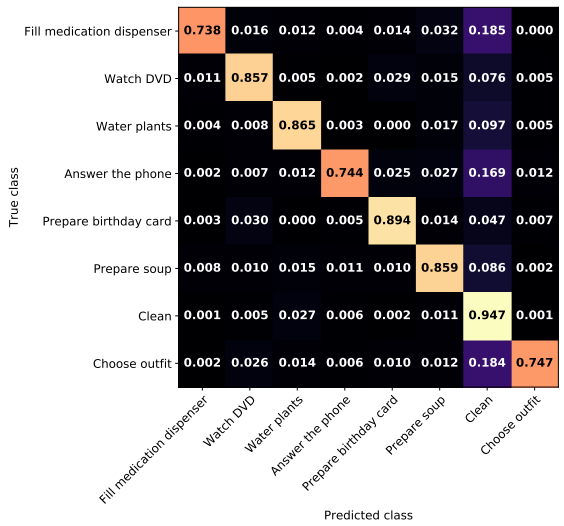
We can conclude that the OA and OA-E perform better with the SVM classifier than Naive Bayes. From Figure 4.3 we may see that both OA and OA-E datasets have a higher accuracy in SVM across all sets of features. Also by comparing the confusion matrices of SVM (figures 4.9 and 4.10) with those of Naive Bayes (figures 4.2 and 4.3), we conclude that SVM gets more samples correctly classified than NB.

IV. Analysis of the IwA dataset

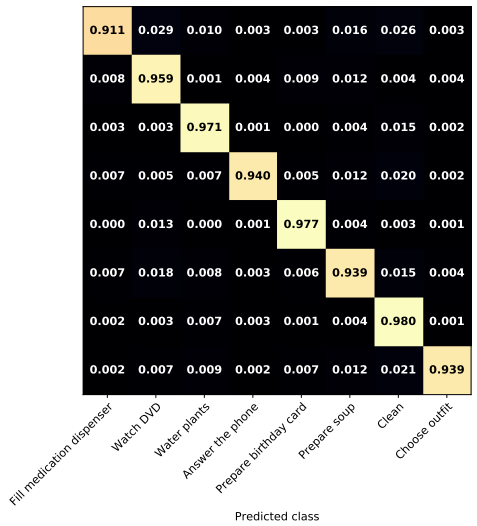
The accuracy in the IwA dataset reaches 93% accuracy with the Base set of features (see Table 4.2). This high value is also reinforced by the F-score in Figure 4.8 (green bars). The confusion matrices are the following:

- Figure 4.11a shows the matrix of applying the Base set of features with a window size 10. The values in the diagonal range from 0.738 to 0.947, while the rest of cells have values close to zero. Exceptionally, the number of FP (false positives) for `Clean` is abnormally high.

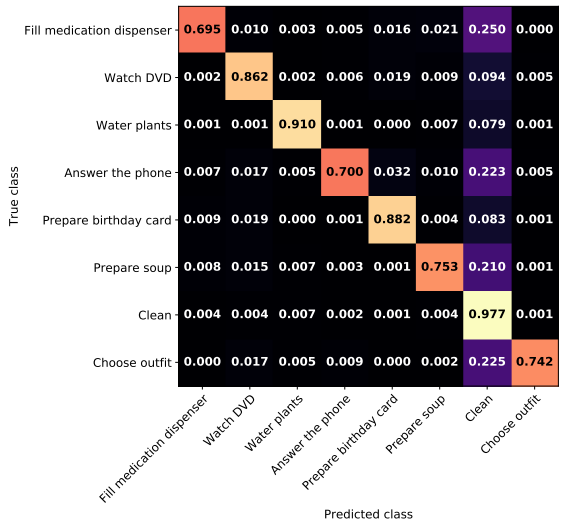
- Figures 4.11c and 4.11c show the matrices when using window size 50 with the Base+TD and Base+SD configurations, respectively. They show a similar distribution to Figure 4.11a with a slightly lower accuracy for almost all the classes and a notable increase of FP for the activity `Clean`.



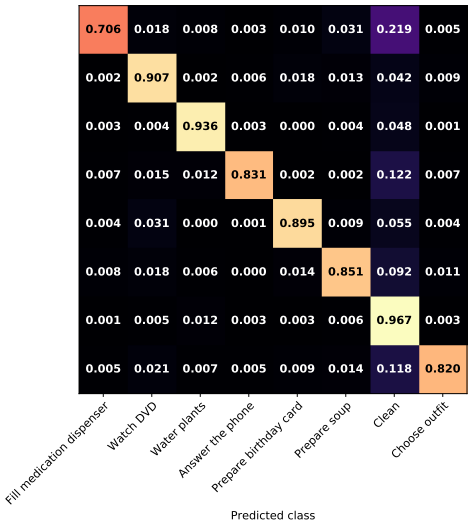
(a) Base set of features, window size 10



(b) Base set of features, window size 50



(c) Base + TD set of features, window size 50



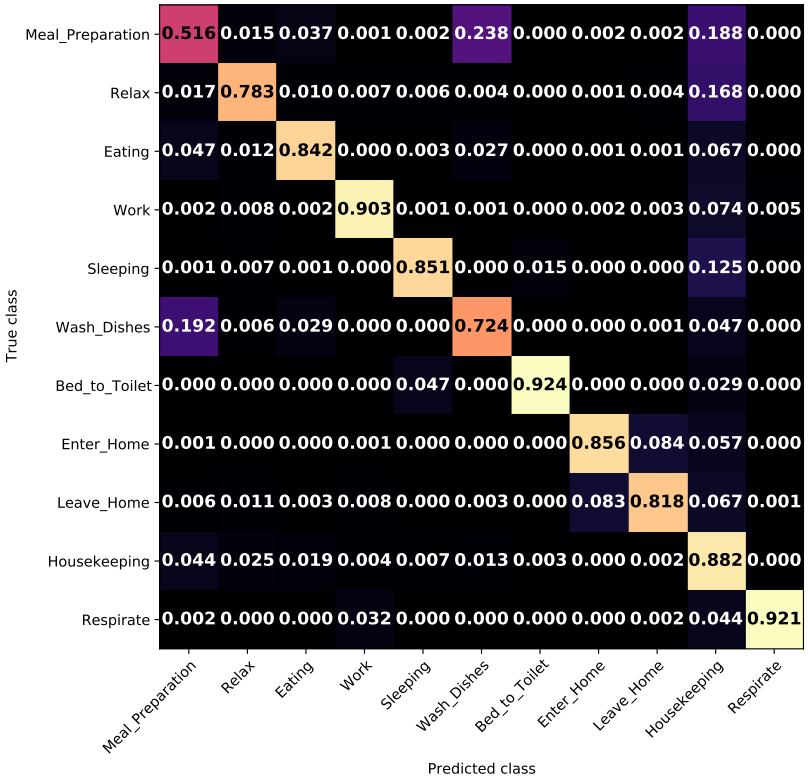
(d) Base + SD set of features, window size 50

Figure 4.11: Confusion Matrices of applying SVM to the IwA dataset

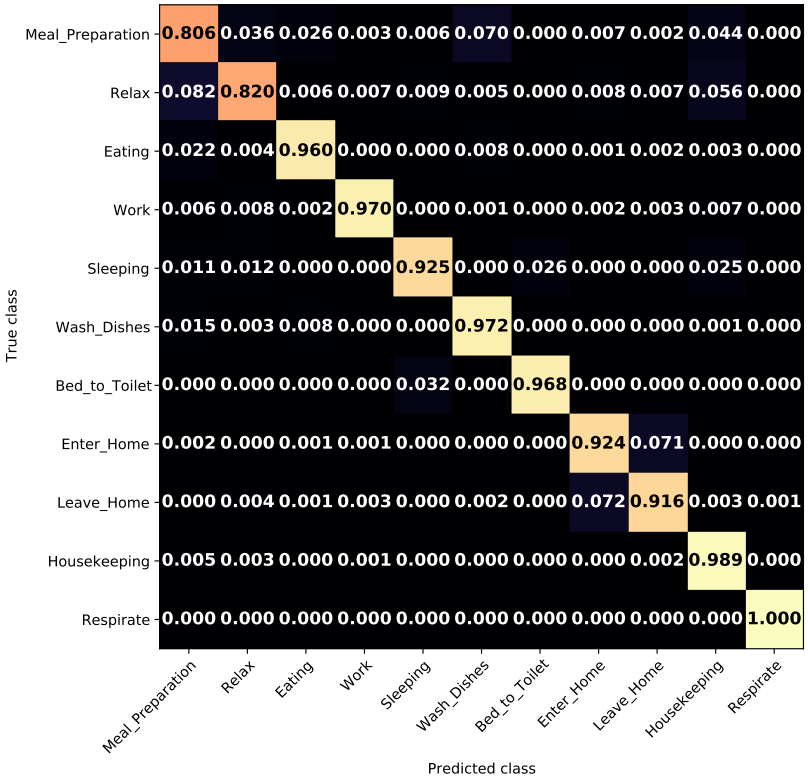
- Interestingly, Figure 4.11b, the confusion matrix when applying the Base configuration with a window size 50, show a significantly higher values in the diagonal and a noteworthy drop in the number of FP of `Clean`.

In conclusion, the SVM has a tendency to misclassify samples as being samples of `Clean`. Overall, it seems that as the number of misclassified samples increases, rather than being classified across all possible activities, SVMs tends to always classify them under the same class, in this case `Clean`. As in OA and OA-E where misclassified samples are categorized as `Cook`, this may be explained by the behaviour of the classifier's voting procedure in multi-class problems.

Observing Table 4.3 we see that IwA is the dataset that benefits the most out of SVM. This is especially true for the Base and Base+SD sets of features, with accuracy increments of 36% and 38%, respectively, from Naive Bayes. The confusion matrices of SVM (Figure 4.11) compared with those of the Naive Bayes (Figure 4.4) also clearly show that the values of the diagonal are much higher in SVM, overall indicating a higher amount of correctly classified samples for each activity.

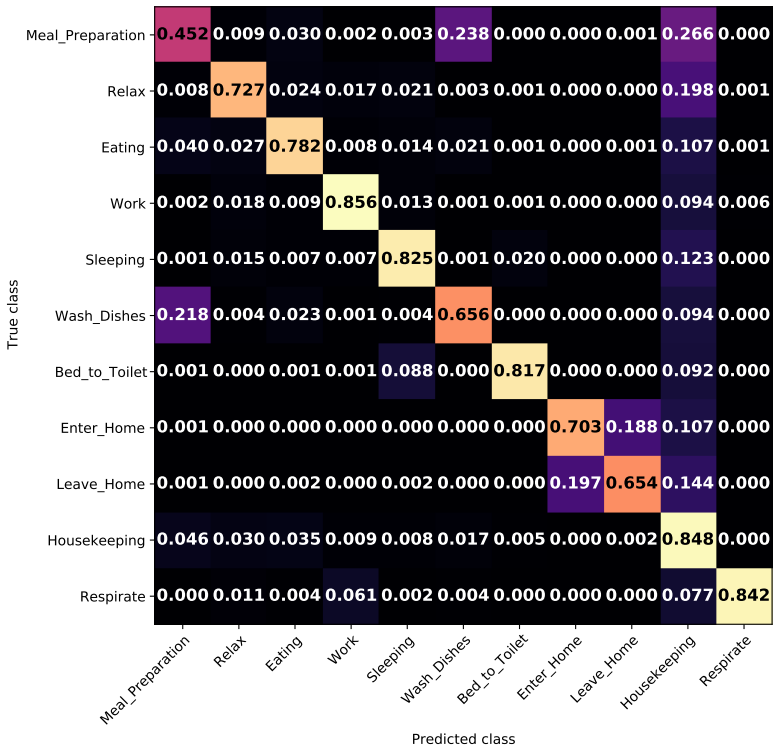


(a) Base set of features, window size 10



(b) Base set of features, window size 50

Figure 4.12: Confusion Matrices of applying SVM to the DLR dataset



(c) Base + TD set of features, window size 50

Figure 4.12: Confusion Matrices of applying SVM to the DLR dataset (cont.)

V. Analysis of the DLR dataset

By observing the figures in Table 4.2 and Figure 4.8 (red bars) we may conclude that the DLR dataset is the worst performing dataset across all feature configurations, the only exception being the Base+TD+SD set of features where it has a marginally better F-score than the IwA dataset. These results are consistent with the more difficult characteristics of this dataset, particularly a significantly higher number of classes and a much less constrained environment compared to the other datasets. Still, the confusion matrices show a more positive result:

- Figure 4.12a shows the matrix of applying the Base set of features with a window size of 10. While most values outside the diagonal are close to zero, we can observe some distinctive characteristics. For example, the percentage of samples `Wash_Dishes` classified as `Meal_Preparation` is of 19.2%, while the other way around is of 23.8%. The are also several values in the column of `Housekeeping` with abnormally high values, indicating a large number of FP for this activity. The values in the diagonal are generally over 0.8 with the notable exceptions for the activities `Meal_Preparation` (first row), `Relax` (second row) and `Wash_dishes` (sixth row).

- Figure 4.12c shows the matrix of applying the Base+TD feature configuration with a window size 10. Compared to Figure 4.12a, the values in the diagonal are smaller, the number of False Positives for activity `Housekeeping` is higher and as before samples `Wash_dishes` are classified as `Meal_Preparation` and vice versa. However, in this matrix there is also a significant percentage of samples `Leave_Home` misclassifed as `Enter_Home` (19.7%) and vice versa (18.8%).

- Figure 4.12b shows the matrix of applying the Base set of features with a window size 50. Compared with the previous two matrices, this matrix has higher values in the diagonal, consistently over 0.8, while the remaining cells have values close to zero. There also is not a class with an abnormal amount of FP, as it was the case with `Housekeeping`.

In a nutshell, when using small windows or dependencies, the SVM classifier struggles identifying samples correctly. This is noticeable in these two aspects:

- A high amount of false positives for the activity `Housekeeping` (this was also observed in OA and OA-E with `Cook` and in IwA with `Clean`)
- The swapping of the classification of activities `Meal_Preparation` and `Wash_Dishes` between each other, and likewise, but to a lesser extent, the swapping of activities `Leave_Home` and `Enter_Home` from each other. This can be explained by the similarity of the samples of the interchanged classes.

From the analysis of the results we have two contrasting pictures. On the one hand, according to Table 4.3, the application of the Base set of features to DLR is the only case that yields a less accuracy in SVM than in NB. On the other hand, comparing the confusion matrices of the SVM classifier (figures 4.12a and 4.12b) with the ones from NB (Figure 4.5), we see the SVM matrix is diagonally dominant, which in Naive Bayes is not, and overall the values outside the diagonal are much smaller in SVM. Therefore, how are these two views able to coexist?

This ends up being a consequence of the imbalanced nature of the DLR dataset. As a remainder, according to Table 3.4, the samples of `Meal_Preparation` and `Relax` make over 84% of the entire dataset. As such, both the F-score and accuracy are heavily influenced by the performance of these two classes. In NB the percentages of correctly classified samples of `Meal_Preparation` and `Relax` are higher than any other class, inflating the global accuracy and F-score. However, in SVM the opposite occurs; i.e., the percentages of correctly classified samples of `Meal_Preparation` and `Relax` are the lowest of the dataset, negatively impacting the global scores.

4.4 Random Forests

Random Forests (RF) is a powerful ensemble model that has found application in the task of human activity recognition [4]. A RF is made up of hundreds of decision trees, which in itself is a type of discriminative model. While SVMs attempt to classify samples directly, a decision tree (and consequently a Random Forest) classifies samples through the likelihood $P(x|c)$ of a sample x for a given class c, and then tagging the sample with the class that returns the highest likelihood, ignoring the prior probability of each class.

In this work, we focus on a type of decision tree also known by Classification And Regression Trees (CART) [5]. Similarly to SVM, a decision tree interprets data vectors within a vector space but unlike SVMs that define a hyperplane dividing the elements of two classes, decision trees define boundaries splitting data into partitions, ideally in a way that samples of different classes are separated from each other [5]. The major strength of a decision tree is its ability to deal with one feature at a time, using only the ones that provide the most discriminative information.

On the left part of Figure 4.13 we can see a set of samples of a two-class problem (red and blue). On the right we can see the boundaries created by a decision tree that partitions the data into four regions.

Boundaries are specified as:

1. the query $x_1 \leq k$ defines two regions, say R1 on the left and R2 on the right
2. on R2, the query $x_2 \leq k''$ defines in turn two regions, one at the top with three blue samples, and one at the bottom with four red samples
3. on R1, the query $x_2 \leq k'$ defines two new regions, the one at the top with three red samples and the one at the bottom with five blue samples

Once we have the partitions, given a sample that satisfies $x_1 < k$, then if it holds $x_2 < k'$ the sample will be classified as blue. Otherwise, it belongs to the red class. On the other hand, a sample in which $x_1 > k$ and $x_2 < k''$ will be classified as belonging to class red; otherwise it belongs to the blue class.

A close look at the samples on the left part of Figure 4.13 reveals that neither Naive Bayes nor SVM would work correctly: the classes are not linearly separable and the features x_1 and x_2 are dependent on each other (samples belonging to the blue class have similar values for both features, while the opposite is true for samples in the red class).

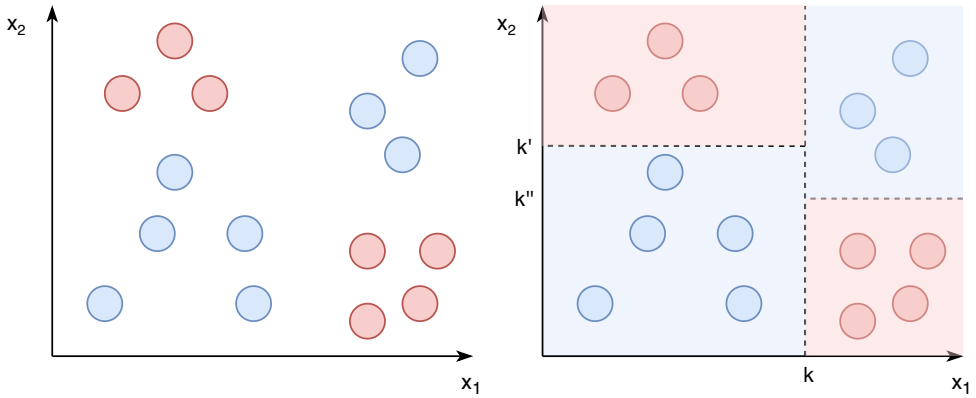


Figure 4.13: Example problem applied to a decision tree

This line of thinking can be modeled into a binary tree, hence the name decision tree, as the one showed at Figure 4.14. A sample to be classified starts at the root node of the tree. Intermediate nodes guide samples along the tree until they reach a leaf node. The path a sample follows all the way down to a leaf node defines the probability of reaching such node.

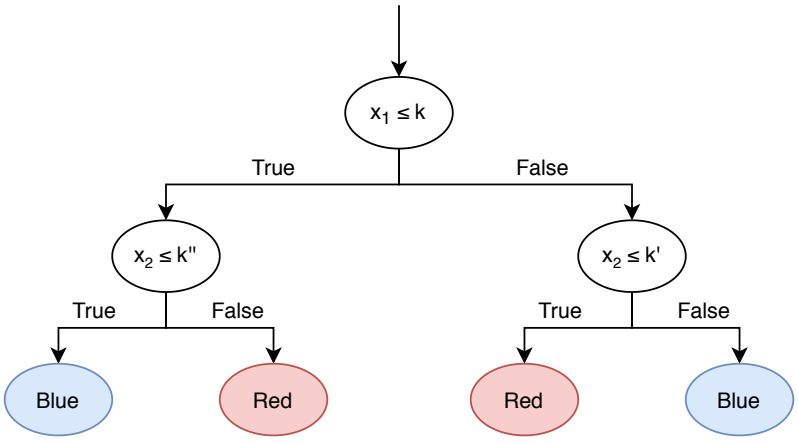


Figure 4.14: Inner workings of decision tree for the example in Figure 4.13

In order to train a decision tree we need a way to measure the *impurity* of a node, which is the uncertainty associated to taking a decision in the node, which in turn is related to the number of samples of each class in the node. Hence, a pure node is one formed only by samples of a single class, while a fully impure node is one that comprises the same number of samples of each possible class. There exist several ways to measure the node impurity, all of them based on the proportion of samples of each class in a given node. Let n be the node of a decision tree, C the number of classes and X_n the number of samples in n; the proportion of samples of class $c \in C$ in node n is equal to:

$$p_{nc} = \frac{X_{nc}}{X_n}$$

where X_{nc} indicates the number of samples of class c in node n. One way to measure impurity is, for example, the Gini criterion [5]:

$$H_{Gini}(X_n) = \sum_{c \in C} p_{nc}(1 - p_{nc})$$

Regardless on how the impurity is measured, when a node with X samples is selected to be partitioned, the data will be split into two partitions X_r (right partition) and X_l (left partition). The partition will be chosen as to minimize the average impurity of the resulting nodes, weighted by the number of samples in each partition:

$$\min_{\substack{\forall X_r, X_l \\ X_r \cup X_l = X \\ X_r \cap X_l = \emptyset}} \frac{|X_r|}{|X|} H(X_r) + \frac{|X_l|}{|X|} H(X_l)$$

It is important to note that it not needed to repeatedly split the partitions of a tree until we get all pure nodes. Instead, we can stop partitioning a node when a threshold of impurity or a minimum number of samples is reached in the node. When the split stops at a node n, n is declared as a leaf node and the probability of each class c in n is calculated as p_{nc}.

The main disadvantage of decision trees is that too many splits may cause lack of generalization and thus overfitting. Additionally, the training process is hard and expensive, as finding the ideal partition at a given node is a NP-Complete problem. As such, decision trees have a set of parameters that limit the growth of the tree such as, but not limited to:

- Minimum number of samples that a node requires to exist;
- Minimum number of samples that a node requires to split;
- Minimum threshold of impurity that a node requires to split;

There also ways to avoid overfitting in an already built tree through pruning, but we will not cover such aspect in this paper.

Random Forests also deal with overfitting, albeit in a different manner. The hundreds of decision trees of a RF are each trained with a subset of the set of training samples so that every tree will be slightly different. This is known as bagging, and the result is that the RF is more stable and less prone to overfitting than any individual decision tree that forms it [4]. RFs have the same parameters that the decision trees, with the addition of a new one that indicates the number of decision trees in the model.

When a samples is fed to be classified, the following steps occur:

1. Each tree classifies the sample individually and outputs the probabilities that the given sample belongs to each of the classes
2. The Random Forest will get the class probabilities from all its trees and average it
3. The Random Forest will classify the sample as the class with the highest average class probability

4.4.1. Design

For this work, we opted for using the implementation of Random Forest provided by the API Scikit-learn 0.22.2 [6]. The trees within the RF are "an optimised version of the CART algorithm"[4], which was explained above. For our models we used the Gini measure of impurity and we used 100 decision trees, the default amount by the library. This number of trees proves to be ideal, as it takes a reasonable time to train and increasing the number of trees does not have any noticeable effect in the accuracy of the model.

We also did not limit the growth of our decision trees in any way, in other words, we did not impose a maximum depth to the tree or a minimum amount of samples in a node to be split besides the fact that a node must have at least one sample. While this configuration may jeopardize the model towards overfitting, our results showed that this was not the case.

4.4.2. Results

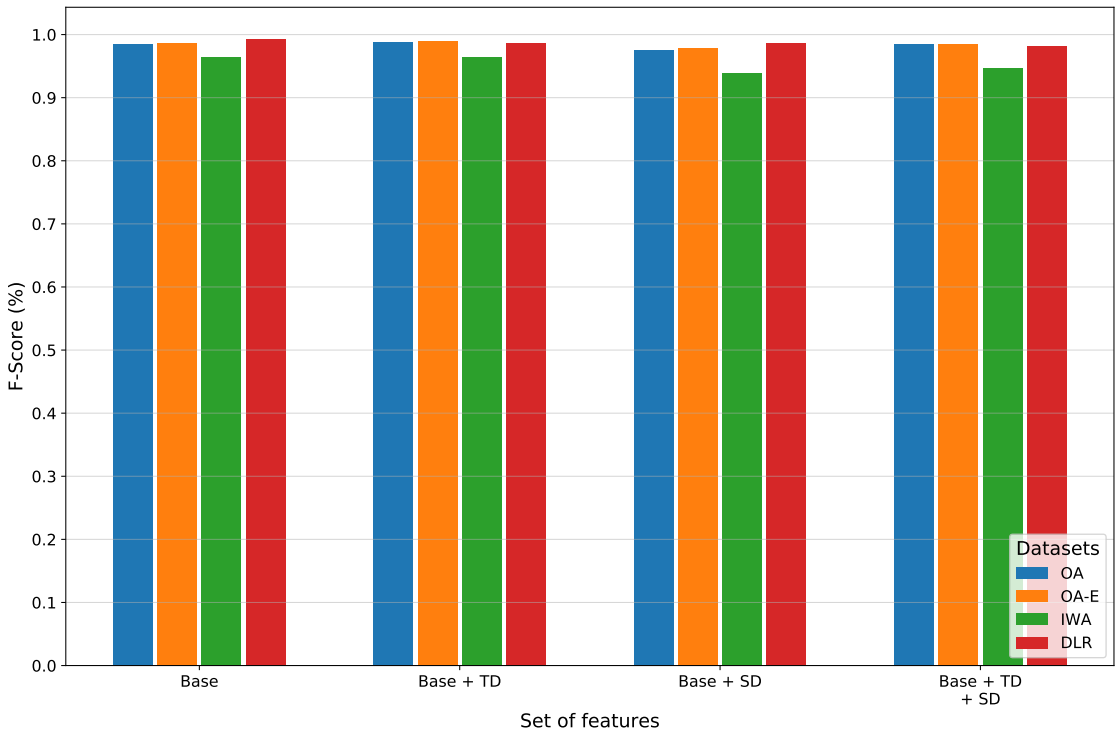


Figure 4.15: F-Score obtained with a RF classifier by dataset and set of features

[4]Scikit-learn. 1.10. Decision Trees. https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart. Accessed: 2020-06-30.

Dataset	Set of Features	Window Sizes					
		10	20	30	40	50	Average
OA	Base	0.9714	0.9859	0.9884	0.9905	0.9903	0.9853
	Base + TD	0.9751	0.9880	0.9918	0.9915	0.9927	0.9878
	Base + SD	0.9528	0.9753	0.9820	0.9845	0.9850	0.9759
	Base + TD + SD	0.9645	0.9852	0.9895	0.9911	0.9929	0.9846
OA-E	Base	0.9773	0.9866	0.9889	0.9895	0.9926	0.9870
	Base + TD	0.9837	0.9885	0.9902	0.9922	0.9938	0.9897
	Base + SD	0.9667	0.9774	0.9812	0.9815	0.9849	0.9783
	Base + TD + SD	0.9759	0.9849	0.9884	0.9881	0.9890	0.9853
IwA	Base	0.9515	0.9693	0.9631	0.9676	0.9703	0.9644
	Base + TD	0.9402	0.9658	0.9705	0.9726	0.9723	0.9643
	Base + SD	0.9087	0.9382	0.9403	0.9518	0.9567	0.9391
	Base + TD + SD	0.9056	0.9506	0.9562	0.9637	0.9614	0.9475
DLR	Base	0.9820	0.9941	0.9970	0.9977	0.9982	0.9938
	Base + TD	0.9744	0.9860	0.9898	0.9913	0.9922	0.9868
	Base + SD	0.9730	0.9853	0.9901	0.9922	0.9934	0.9868
	Base + TD + SD	0.9687	0.9814	0.9863	0.9885	0.9896	0.9829

Table 4.4: Accuracy (%) obtained with RF by dataset and set of features

Dataset	Set of Features	Accuracy (%)		
		Naive Bayes	Random Forests	Improvement
OA	Base	0.8090	0.9853	**0.1763**
	Base + TD	0.6789	0.9878	**0.3089**
	Base + SD	0.6941	0.9759	**0.2818**
	Base + TD + SD	0.5874	0.9846	**0.3972**
OA-E	Base	0.8128	0.9870	**0.1742**
	Base + TD	0.7792	0.9897	**0.2105**
	Base + SD	0.7466	0.9783	**0.2317**
	Base + TD + SD	0.6778	0.9853	**0.3075**
IwA	Base	0.5647	0.9644	**0.3997**
	Base + TD	0.5972	0.9643	**0.3671**
	Base + SD	0.4530	0.9391	**0.4861**
	Base + TD + SD	0.4460	0.9475	**0.5015**
DLR	Base	0.8680	0.9938	**0.1258**
	Base + TD	0.5541	0.9868	**0.4327**
	Base + SD	0.7173	0.9868	**0.2695**
	Base + TD + SD	0.4448	0.9829	**0.5381**

Table 4.5: Improvement in accuracy (%) from Naive Bayes to Random Forests

Dataset	Set of Features	Accuracy (%)		
		SVM	Random Forest	Improvement
OA	Base	0.9513	0.9853	**0.0340**
	Base + TD	0.9105	0.9878	**0.0773**
	Base + SD	0.9253	0.9759	**0.0506**
	Base + TD + SD	0.8711	0.9846	**0.1135**
OA-E	Base	0.9562	0.9870	**0.0308**
	Base + TD	0.9317	0.9897	**0.0580**
	Base + SD	0.9317	0.9783	**0.0466**
	Base + TD + SD	0.8981	0.9853	**0.0872**
IwA	Base	0.9309	0.9644	**0.0335**
	Base + TD	0.8500	0.9643	**0.1143**
	Base + SD	0.8330	0.9391	**0.1061**
	Base + TD + SD	0.7093	0.9475	**0.2382**
DLR	Base	0.7763	0.9938	**0.2175**
	Base + TD	0.6366	0.9868	**0.3502**
	Base + SD	0.7454	0.9868	**0.2414**
	Base + TD + SD	0.6467	0.9829	**0.3362**

Table 4.6: Improvement in accuracy (%) from SVM to Random Forest

Table 4.4 contains the accuracy obtained with Random Forest for every combination of dataset, set of features and window size, and Figure 4.15 shows the average F-score. We present two more tables to compare the accuracy of Random Forests with Naive Bayes (Table 4.5) and SVM (Table 4.6). We will follow the same structure for explaining the results of RF that we used for NB and SVM: impact of the feature configuration, impact of the window size and finally an analysis per dataset.

I. Impact of the feature configurations

In Figure 4.15 we can observe that the four color bars are essentially the same for all feature configurations, indicating that the effect of adding dependencies is minimal. The only dataset with a more visible effect in F-score is the IwA dataset (green bar), as this is the dataset has the noisiest windows.

The numbers in Table 4.4 support the F-score results as there is hardly significant differences in accuracy across the different feature configurations except for the IwA dataset, which shows the lowest values. Still, the difference in accuracy between the best and worst configuration in the IwA is minimal in comparison with the NB and SVM classifiers: only a difference of 2.53%. It is noteworthy that Base+TD does show here a marginally better result, something that did not happen with NB and SVM.

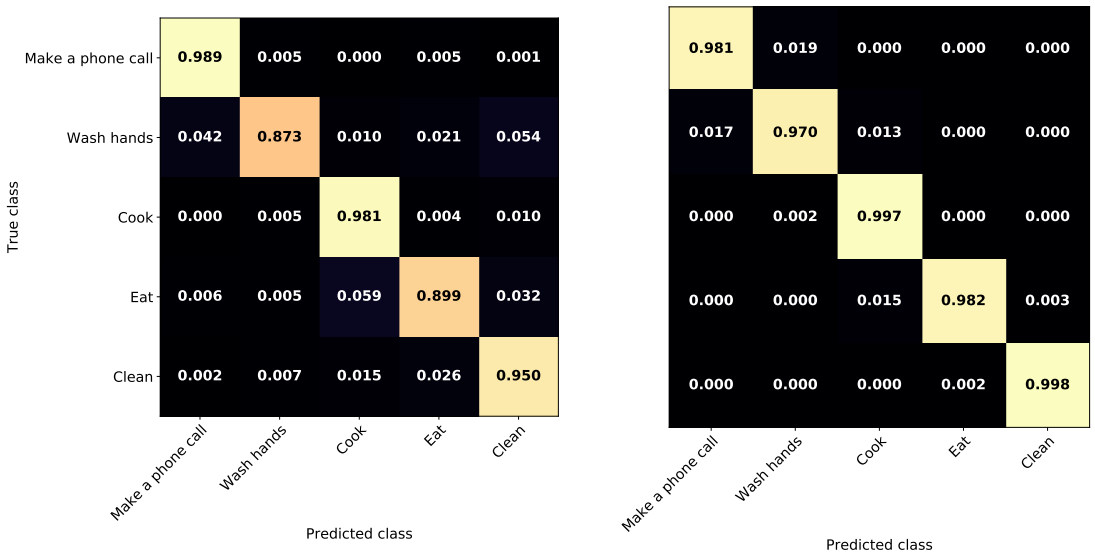
We can pose the question, why Random Forests are not so negatively affected by adding dependencies as NB and SVM classifiers? The reason is that the functioning of RF itself applies a more controlled and individualized treatment of the features and so it avoids the unintended effect of making the sensor counts be very similar across classes. For example, if the samples of one class have an average value of 1.000001 for a given dimension and the samples of another class average a value of 1.000002, a decision tree is able to differentiate between them and choose samples with a value equal or smaller of 1.000001. In addition, the boundary placed by a SVM is conditioned by the other dimensions and a NB classifier would obtain the same likelihood for both samples and so it would need to resort to using the prior probability to make the distinction.

II. Impact of the window sizes

Table 4.4 confirms that the effect of the window size on the accuracy in RF is much smaller than in the NB and SVM classifiers. Even though larger window sizes do lead to higher accuracy, overall the difference in accuracy between the smallest and largest window is equal or smaller to 2%. This contrasts, for example, with the SVM results, where the largest difference in accuracy is 14% for the Base configuration of the DLR dataset (see Table 4.2) or 6% in NB (see Table 4.1).

Consequently, how are Random Forests able to utilize the additional information provided by the windows while minimizing the added noise? The most probable explanation is related to the way individual decision trees are built. At each node split, a query about a feature must be selected and, while choosing the optimal split is costly, the advantage is that decision trees are able to select the features that maximize the ability of the classifier to differentiate among classes. As such, RF benefit from the extra information of wider windows while they tend to skip features that are not helpful in distinguishing classes and avoid the features most affected by the increasing noise.

III. Analysis of the OA and OA-E datasets



(a) Base + SD set of features, window size 10

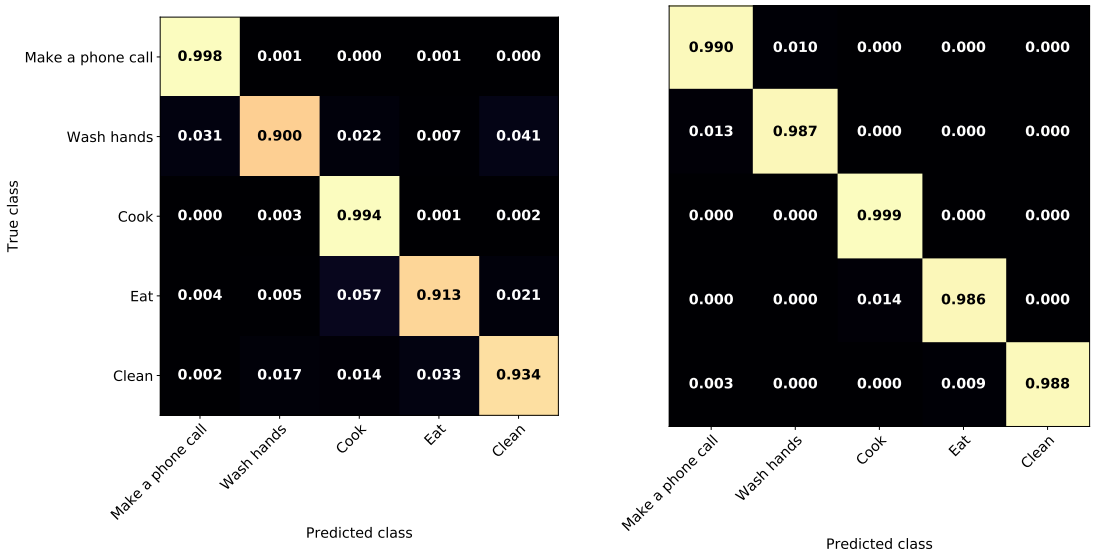
(b) Base + SD + TD set of features, window size 50

Figure 4.16: Confusion Matrices of applying Random Forest to the OA dataset

Table 4.4 and Figure 4.15 show that OA and OA-E score very well in both metrics, accuracy and F-score. F-score is practically the same in both datasets while OA-E slightly outperforms OA in accuracy. The slight edge in scores that OA-E has over OA with the NB and SVM classifiers is not shown here. Random Forest performs admirably well in both datasets and there is little room for improvement.

Regarding the confusion matrices:

- In Figures 4.16a and 4.16b we show the configurations that lead to the lowest (Base+SD) and highest (Base+SD+TD) accuracy, respectively, for the OA dataset (according to Table 4.4).
- In Figures 4.17a and 4.17b we show the configurationa that lead to the lowest (Base+SD) and highest (Base+TD) accuracy, respectively, for the OA-E dataset.



(a) Base + SD set of features, window size 10

(b) Base + TD set of features, window size 50

Figure 4.17: Confusion Matrices of applying Random Forest to the OA-E dataset

Across all confusion matrices we see the same common characteristics. Matrices in Figures 4.16 and Figures 4.17 are almost diagonal, indicating that most samples are classified in their respective class. Also all values outside the diagonal are close to 0, indicating that no class has an abnormally high value of false positives or false negatives.

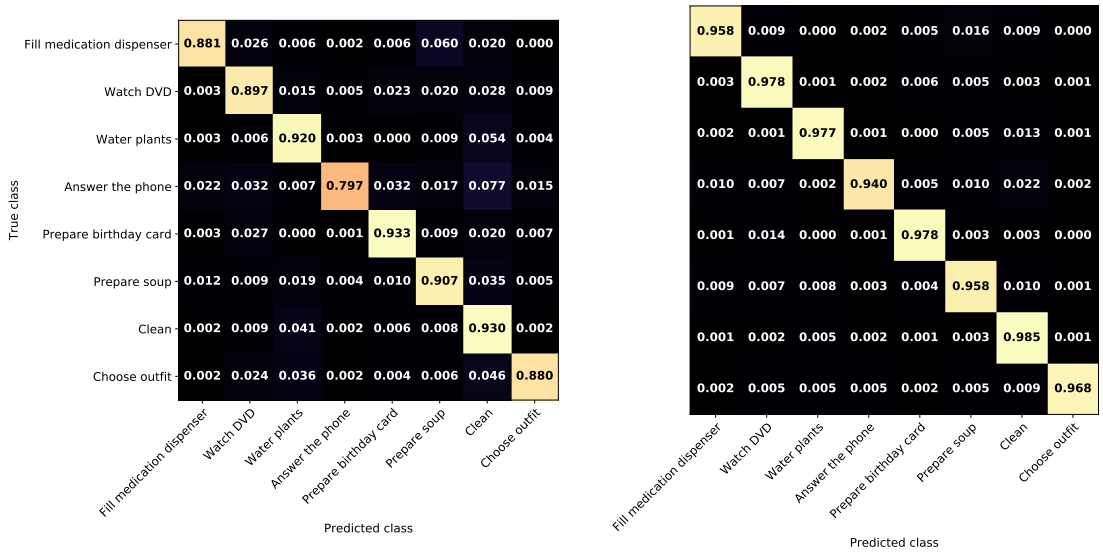
Overall, Random Forests is clearly the best classifier as it is able to identify all classes correctly, independently of the window size and feature configuration. Tables 4.5 and 4.6 show that RFs clearly outperform NB and SVM across all configurations. Also, if we compare the confusion matrices of RF (Figures 4.16 and 4.17) with the matrices of Naive Bayes (Figures 4.2 and 4.3) and SVM (Figures 4.9 and 4.10), we can observe the percentage of samples correctly is significantly higher in RF over the other two classifiers.

IV. Analysis of the IwA dataset

According to Table 4.4, the IwA dataset has the lowest accuracy of all the datasets, and this is also corroborated by the F-score green bar (see Figure 4.15).

As for the confusion matrices, Figure 4.18a shows that the configuration Base+TD+SD reports the lowest accuracy, while the configuration Base+TD in Figure 4.18b reports the highest accuracy. Similarly to the datasets OA and OA-E, the RF classifier is able to correctly classify all the activities in the dataset, independently of the window size and feature configuration.

The values for IwA in Tables 4.5 and 4.6 show that Random Forests outperform NB and SVM in all configurations. If we compare the RF confusion matrices with those of SVM (Figure 4.11), although in both cases the matrices are diagonally dominant, the values in the diagonal are higher for Random Forests. Also the outcomes of Random Forests do not feature an abnormal amount of false positives for any class. If we compare the confusion matrices of RF with the matrices of Naive Bayes (Figure 4.4), we observe the values in the diagonal are all significantly higher and the rest of values are significantly lower, meaning that more samples are correctly classified.



(a) Base + TD + SD set of features, window size 10 **(b)** Base + TD set of features, window size 40

Figure 4.18: Confusion Matrices of applying Random Forest to the lwA dataset

V. Analysis of the DLR dataset

DLR is the dataset that reports the highest accuracy and F-score when using the Base set of features as well as comparable results with those of OA and OA-E for the rest of feature configurations (Table 4.4 and Figure 4.15 (red bar)). We can say the results are impressive considering that DLR is the most difficult dataset. Random Forests is the technique that benefits the most from using more samples during training, and DLR is indeed the dataset that has significantly more sensor events than any other dataset (see the tables from Chapter 3), which gives it a competitive advantage.

The confusion matrices are the following:

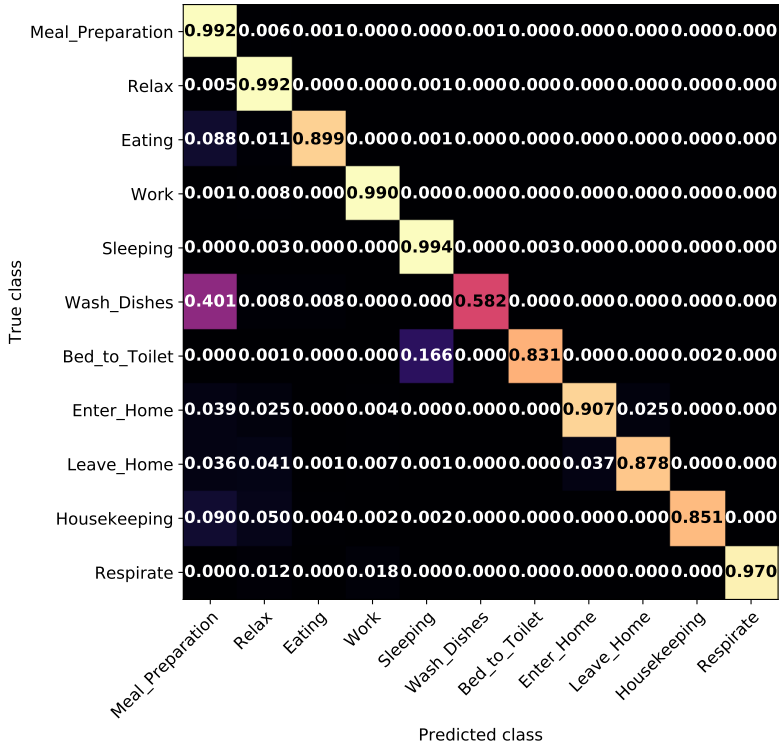
- Figure 4.19a shows the Base set of features with a window size 10. While most of the values in the diagonal are higher than 0.8, indicating a high rate of correctly classified samples, the value in the sixth column (activity `Wash_Dishes`) is abnormally low, only 0.582. Additionally, the percentage of samples of `Wash_Dishes` misclassified as `Meal_Preparation` is of 40.1%.

- Figures 4.19c and 4.19d show the matrices when using the Base+TD and the Base+SD configurations, respectively, with window size 10. Similarly to Figure 4.19a, the values in the diagonal are high with the exception of `Wash_Dishes`. There is also a large number of samples of `Wash_Dishes` misclassified as `Meal_Preparation` (0.622 and 0.599 for Base+TD and Base+SD, respectively) as well as more than 10% of samples of `Eating` and `Housekeeping` misclassified as `Meal_Preparation`.

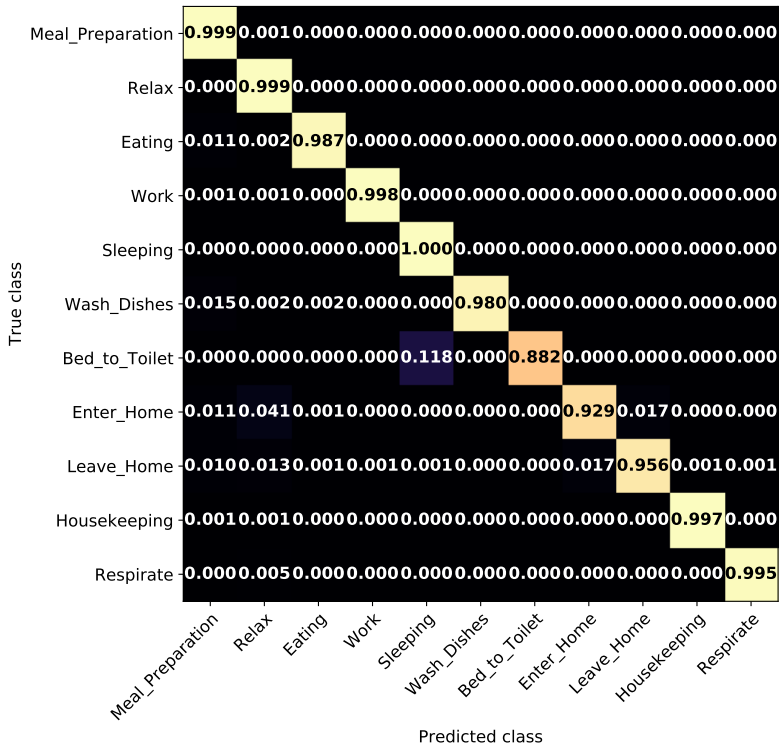
- In Figure 4.19b, the Base set of features with a window size 50, we observe very high values in the diagonal, the smallest one being 0.882. Also the rest of the values outside of the diagonal are all very close to zero. As a result, it is clear that almost all the activities are correctly identified.

The high number of Wash_Dishes samples classified as Meal_Preparation is not a new scenario as we observed the same in both SVM and Naive Bayes (see Figures 4.12 and 4.5). However, unlike SVM, in RF the amount of samples of Meal_Preparation misclassified as Wash_Dishes is very small with (around 0.1%). What happens in RF is that the proportion of samples of each activity in the dataset affects the probabilities of the tree and the leaf nodes, which in turn affects the classification probability of each sample. As a result, the trees of the RF classifier will contain inflated class probabilities for the activities with most samples.

Finally, a comparison of the accuracy of RF with NB (see Table 4.5) and with SVM (see Table 4.3) reveals that RF has the highest accuracy of any of the tested classifiers. Comparing the confusion matrices with those from NB (see Figures 4.5 and 4.6) we can see that although both classifiers tend to misclassify samples of Wash_Dishes as Meal_Preparation, in general the number of correctly classified samples is higher in Random Forests for all the activities. A comparison of the confusion matrices from RF with those from SVM (see Figure 4.12) displays a smaller number of misclassified samples of Wash_Dishes and Bed_to_Toilet. in SVM. However, the percentage of correctly classified samples of the two largest classes, Meal_Preparation and Relax, is much higher in Random Forests.

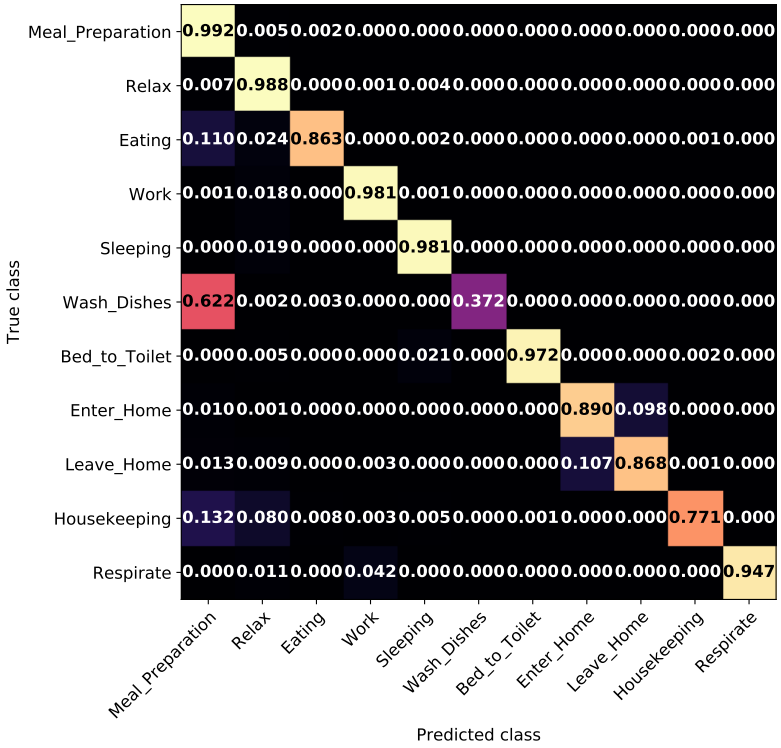


(a) Base set of features, window size 10

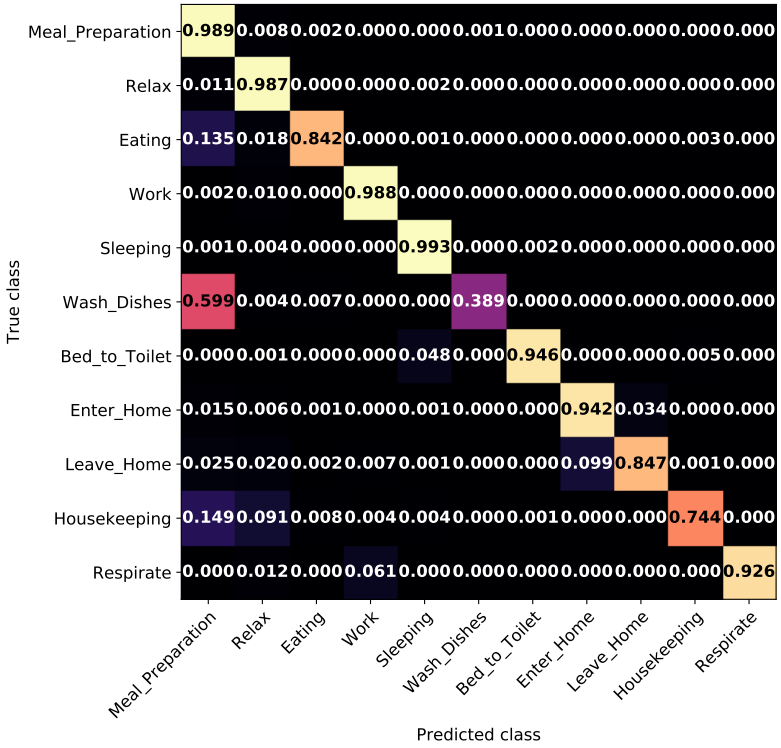


(b) Base set of features, window size 50

Figure 4.19: Confusion Matrices of applying Random Forest to the DLR dataset



(c) Base + TD set of features, window size 10



(d) Base + SD set of features, window size 10

Figure 4.19: Confusion Matrices of applying Random Forest to the DLR dataset (cont.)

CHAPTER 5
Conclusions

In this project we managed to accomplish the objectives we proposed ourselves in Chapter 1. After analyzing the performance of the Naive Bayes, Support Vector Machine and Random Forest classifier, we are ready to summarize our experience with this work as well as highlighting the strengths and weaknesses of the three classifiers.

By using four different datasets we were able to give a complete picture of each classifier. The OA and OA-E datasets played the role of a control group, as the 'easiests' datasets, and also showed how the repetition of a wrongly executed activity of the participant affects classification. Ultimately, the results of OA-E turned out to be positive since participants take overall longer to complete the tasks, generating more information in the process. The performance on the IwA dataset evidences how classifiers are able to handle a significant amount of noise. Finally, the DLR is the more challenging dataset, not only because of an increasing amount of noise, but also for being a very unbalanced dataset, as a result of the lack of restrictions on the participants activities. Additionally, we were able to gauge the amount of information and noise in each sample by changing the window size of the event windows.

Overall Random Forests proved to be the most adequate classification technique thanks to its high flexibility in placing the class boundaries. Specifically, the ability to distinguish activities by choosing the most adequate features enables RF to perform correctly across all the tested datasets. Not only it has performed correctly in all of them, but also no significant drop of accuracy and F-score was reflected in the results. We can thus conclude that Random Forests is the most robust classification method. In addition, the analysis of the confusion matrices enabled us to examine in detail the results of each classifier and thus corroborate our conclusion that RF has proved to be the best classifier.

We can also argue that Naive Bayes falls short in the most difficult datasets. The amount of noise makes NB struggle to correctly identify the classes with fewer sensor events in the DLR dataset. SVM did not either perform as well as Random Forests since the results showed a large number of false positives for certain classes across datasets. Besides taking much longer to train, SVMs also struggle with the DLR dataset due to the large quantity of samples of pairwise classes with exchanged labels, heavily worsening its metrics.

We can also conclude that it is not recommendable to apply the sensor and time dependencies to future classifiers. In the end, the negative side effects made the classifiers worse at differentiating activities from each other. While there are some scenarios in which they proved useful, the results obtained when applying dependencies were ultimately too inconsistent. A proper application of these dependencies would require some further study.

Due to the extensiveness of the field dedicated to the study of Human Activity Recognition, our work may be expanded in several ways:

- Evaluate classifiers other than the ones commented in this work. Looking back at the state of the art, sequential models have been commonly used for the task of HAR. In this project, we decided to focus on non-sequential models as they are able to use the same feature representation, making the comparison between them fairer. Sequential models require a completely different representation that does not rely on data segmentation. A similar analysis as the one presented here is doable focusing only in sequential models such as Hidden Markov Models and Conditional Random Fields.

  Other non-sequential models that are also prevalent but have not been tackled in this work are Deep Neural Networks.

- Evaluate the classifiers using a different data segmentation method. In this work we experienced with the sensor event based windowing but other more complex methods such as "change point detection-based activity segmentation" [22] can be tested. This type of segmentation is based on training an unsupervised model that attempts to detect when there was a change in the participants behaviour in order to try to build a window encompassing all the events of a single activity.

- The same analysis is also doable in more complex scenarios of HAR, beyond the restrictions included in our definition in section 2.1, namely datasets with two or more participants at a time, multi-resident environments, or scenarios with two or more activities being realized at a time, in which case it becomes a multi-label classification problem. Also it is possible to work with data coming from wearable sensors rather than environment sensors, as the MHEALTH and UCI HAR datasets.

- In this project a large amount of information was left unused in each dataset so as to make sure the same set of features are extracted from each dataset. Another interesting research direction lies in extracting more highly specific features which fully utilize the information provided by the data.

Bibliography

[1] Damla Arifoglu and Abdelhamid Bouchachia. Activity recognition and abnormal behaviour detection with recurrent neural networks. *Procedia Computer Science*, 110:86 – 93, 2017. 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops. URL: http://www.sciencedirect.com/science/article/pii/S1877050917313005, doi:https://doi.org/10.1016/j.procs.2017.06.121.

[2] O. Banos, C. Villalonga, R. Garcia, A. Saez, M. Damas, J. A. Holgado, S. Lee, H. Pomares, and I. Rojas. Design, implementation and validation of a novel open framework for agile development of mobile health applications. *Biomedical Engineering Online*, 14(S2:S6):1–20, 2015.

[3] Oresti Banos, Rafael Garcia, Juan Holgado-Terriza, Miguel Damas, Hector Pomares, Ignacio Rojas, Alejandro Saez, and Claudia Villalonga. mhealthdroid: A novel framework for agile development of mobile health applications. In Leandro Pecchia, LimingLuke Chen, Chris Nugent, and Jose Bravo, editors, *Ambient Assisted Living and Daily Activities*, volume 8868 of *Lecture Notes in Computer Science*, pages 91–98. Springer International Publishing, 2014. URL: http://dx.doi.org/10.1007/978-3-319-13105-4_14, doi:10.1007/978-3-319-13105-4_14.

[4] Leo Breiman. *Machine Learning*, 45(1):1573–0565, 10 2001. doi:10.1023/A:1010933404324.

[5] Leo Breiman, Charles J. Stone, Jerome H. Friedman, and Richard A. Olshen. *Classification and regression trees*. Chapman Hall, 1984.

[6] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[7] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, 2011. URL: https://doi.org/10.1145/1961189.1961199.

[8] Guilin Chen, Aiguo Wang, Shenghui Zhao, Li Liu, and Chih-Yung Chang. Latent feature learning for activity recognition using simple sensors in smart homes. *Multimedia Tools and Applications*, 77(12):15201 – 15219, 6 2018. doi:10.1007/s11042-017-5100-4.

[9] Diane J. Cook. Learning Setting-Generalized Activity Models for Smart Spaces. *IEEE Intelligent Systems*, 27(1):32–38, 2012.

[10] Anguita Davide, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz. In *21st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning Bruges*, pages 437–442. ESANN, 6 2013.

[11] Wolfgang Ertel. *Introduction to Artificial Intelligence*. Springer International Publishing AG, Cham, Switzerland, 2017.

[12] L. G. Fahad, S. F. Tahir, and M. Rajarajan. Activity recognition in smart homes using clustering based classification. In *2014 22nd International Conference on Pattern Recognition*, pages 1348–1353, 2014. doi:10.1109/ICPR.2014.241.

[13] Iram Fatima, Muhammad Fahim, Young-Koo Lee, and Sungyoung Lee. Effects of smart home dataset characteristics on classifiers performance for human activity recognition. In Sang-Soo Yeo, Yi Pan, Yang Sun Lee, and Hang Bae Chang, editors, *Computer Science and its Applications*, pages 271–281, Dordrecht, 2012. Springer Netherlands.

[14] Enrique Garcia-Ceja and Ramon F. Brena. An improved three-stage classifier for activity recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 32(01):1860003, 2018. doi:10.1142/S0218001418600030.

[15] H. Gjoreski, M. Gams, and M. Lutrek. Human activity recognition: From controlled lab experiments to competitive live evaluation. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 139–145, 2015.

[16] S. Ha and S. Choi. Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 381–388, 2016. doi:10.1109/IJCNN.2016.7727224.

[17] Yong-Joong Kim, Yonghyun Kim, Juhyun Ahn, and Daijin Kim. Integrating hidden markov models based on mixture-of-templates and k-nn^2 ensemble for activity recognition. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*, pages 1636–1641. IEEE, 2016. doi:10.1109/ICPR.2016.7899871.

[18] Narayanan Chatapuram Krishnan and Diane J. Cook. Activity recognition on streaming sensor data. *Pervasive Mob. Comput.*, 10:138–154, 2014. doi:https://doi.org/10.1016/j.pmcj.2012.07.003.

[19] Douglas L. Weeks, Gina L. Sprint, Virgeen Stilwill, Amy Lou Meisen-Vehrs, and Diane J. Cook. Implementing wearable sensors for continuous assessment of daytime heart rate response in inpatient rehabilitation. *Telemedicine and e-Health*, 24(12):1014–1020, 12 2018. doi:10.1089/tmj.2017.0306.

[20] R. Mohamed, T. Perumal, M. N. Sulaiman, N. Mustapha, and M. N. Razali. Conflict resolution using enhanced label combination method for complex activity recognition in smart home environment. In *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, pages 1–3, 2017.

[21] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2001.

[22] Tinghui Wang Samaneh Aminikhanghahi and Diane J. Cook. Real-Time Change Point Detection with application to Smart Home Time Series Data. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):1010–1023, 2019. doi:https://doi.org/10.1109/tkde.2018.2850347.

[23] M. Schmitter-Edgecombe and D.J. Cook. Assessing the Quality of Activities in a Smart Environment. *Methods of Information in Medicine*, 48(05):480–485, 2009. doi:https://doi.org/10.3414/me0592.

[24] Ahmad Shahi, Brendon Woodford, and Hanhe Lin. Dynamic real-time segmentation and recognition of activities using a multi-feature windowing approach. In *Lecture Notes in Computer Science*, volume 10526, pages 26–38, 10 2017. doi:10.1007/978-3-319-67274-8_3.

[25] Geetika Singla, Diane J. Cook, and Maureen Schmitter-Edgecombe. Tracking Activities in Complex Settings Using Smart Environment Technologies. *Int J Biosci Psychiatr Technol*, 1(1):25–35, 2009. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2794487.

[26] Pavle Skocir, Petar Krivic, Matea Tomeljak, Mario Kusek, and G. Jezic. Activity detection in smart home environment. *Procedia Computer Science*, 96:672–681, 12 2016. doi:10.1016/j.procs.2016.08.249.

[27] Alexander J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Stat. Comput.*, 14(3):199–222, 2004. doi:10.1023/B:STCO.0000035301.49549.88.

[28] Abdulhamit Subasi, Kholoud Khateeb, Tayeb Brahimi, and Akila Sarirete. Chapter 5 - human activity recognition using machine learning methods in a smart healthcare environment. In Miltiadis D. Lytras and Akila Sarirete, editors, *Innovation in Health Informatics*, Next Gen Tech Driven Personalized MedSmart Healthcare, pages 123 – 144. Academic Press, 2020. URL: http://www.sciencedirect.com/science/article/pii/B9780128190432000058, doi:https://doi.org/10.1016/B978-0-12-819043-2.00005-8.

[29] Tim van Kasteren, Gwenn Englebienne, and B. Krose. Activity recognition using semi-markov models on real world smart home datasets. *JAISE*, 2:311–325, 01 2010. doi:10.3233/AIS-2010-0070.

[30] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A review of human activity recognition methods. *Frontiers Robotics AI*, 2:28, 2015. doi:10.3389/frobt.2015.00028.

[31] Garrett Wilson, Christopher Pereyda, Nisha Raghunath, Gabriel de la Cruz, Shivam Goel, Sepehr Nesaei, Bryan Minor, Maureen Schmitter-Edgecombe, Matthew E. Taylor, and Diane J.Cook. Robot-enabled support of daily activities in smart home environments. *Cognitive Systems Research*, 54:258 – 272, 2019. doi:https://doi.org/10.1016/j.cogsys.2018.10.032.

[32] Nawel Yala, Belkacem Fergani, and Anthony Fleury. Feature Extraction for Human Activity Recognition on Streaming Data. In *International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6, 2015. doi:https://doi.org/10.1109/inista.2015.7276759.

[33] G. Michael Youngblood, Edwin O. Heierman, Lawrence B. Holder, and Diane J. Cook. D.j.: Automation intelligence for the smart environment. In *In: Proceedings of IJCAI*, pages 1513–1514, 2005.