MÁSTER EN TECNOLOGÍAS Y APLICACIONES EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

UNIVERSIDAD DE ALMERÍA

ESCUELA SUPERIOR DE INGENIERÍA

Detección de actividades en tiempo real con sensores multimodales en una *Smart Home*

Curso 2019/2020

Alumno/a:
Marcos Lupión Lorente

Director/es:
Pilar Martínez Ortigosa
Javier Medina Quero

Agradecimientos

En primer lugar, agradezco a mi directora, Pilar Martínez Ortigosa, por haberme guiado a lo largo de todos estos años. Gracias por toda la ayuda que me ha proporcionado, sin ella todo esto no hubiera sido posible. También, agradecer a la Universidad de Jaén por habernos proporcionado toda la ayuda que hemos necesitado y guiarnos en nuestra incorporación al mundo del IoT. Especialmente a Javier Medina Quero, mi cotutor, que es un experto en el tema que trata el presente trabajo, por todo su esfuerzo, ayuda y comprensión a lo largo de la realización del mismo.

Por otra parte, gracias a mi familia, por estar en todo momento pendientes de cualquier cosa que me ocurría e intentando ayudar en la medida de lo posible. Son un apoyo fundamental y gran parte de lo que soy hoy en día, es gracias a ellos.

A mi pareja, Ana, por confiar siempre en mí, apoyarme en cada paso que doy y hacerme disfrutar de la vida siempre. Por muchos más momentos inolvidables juntos. También agradecer el apoyo a mis amigos de siempre, por poder contar con vosotros para todo y ser una parte fundamental de mi vida. Por estar en las buenas y sobre todo, en las malas.

Agradezco a mis compañeros de máster todo el trabajo y esfuerzo que han realizado para sacar las asignaturas adelante. Hemos sido un equipo y así todo es más fácil.

En último lugar y no menos importante, gracias a mis compañeros de Grado que se han convertido en muy buenos amigos. Un año después de acabar, cada uno con su vida, pero más unidos que nunca. Por mantener esta amistad durante muchos años más y seguir disfrutando como hacemos cada vez que nos vemos.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

1.1 Motivación

En la sociedad actual, se está produciendo un incremento de las personas mayores, por lo que se estima que, en el año 2050 estará duplicado el número de personas con una edad superior a los 60 años (OMS, 2020). Sin embargo, junto con ese aumento, también existirá un alto porcentaje de personas mayores que requerirán mayores necesidades de atención, debido principalmente a una pérdida de la independencia (OMS, 2020).

Debido al envejecimiento poblacional, se están implementando medidas para dar respuesta a dicho envejecimiento (Pinto y cols., 2017). En este caso, resulta esencial destacar el concepto de envejecimiento en el lugar, el cual es definido como *"permanecer viviendo en la comunidad, con cierto nivel de independencia, en lugar de en la atención residencial"* (Pinto y cols., 2017) (Peek y cols., 2019). Por tanto, con dicho envejecimiento, se pretende que las personas mayores mantengan su independencia durante un mayor período de tiempo en su hogar. Además, de esta forma se disminuirían los costes a nivel institucional (Turjamaa y cols., 2019).

En este sentido, las tecnologías están cobrando un papel importante para favorecer dicho envejecimiento en el lugar (Peek y cols., 2019). Cada vez se están utilizando más las tecnologías para aumentar y mejorar la calidad de vida de las personas y su independencia (Grgurić y cols., 2019). En concreto, las Smart Homes proporcionan a las personas mayores una monitorización en su propio hogar, ofreciendo además, cierta seguridad para desempeñar de manera más independiente las actividades de su vida diaria (Turjamaa y cols., 2019).

En la actualidad, se están desarrollando numerosos laboratorios (Espinilla y cols., 2018) (Nugent y cols., 2009) (Chan y cols., 2009) que simulan una *smart home* con el objetivo de desarrollar y probar herramientas que proporcionen a las personas mayores, con enfermedades o en situación de dependencia, un ambiente seguro para que puedan vivir con tranquilidad. Así, para mejorar la monitorización de estas personas, se están implementando aplicaciones que pretenden realizar un reconocimiento de las actividades (AR) de las personas en la vivienda (Kim y cols., 2010).

Así, con el objetivo de reconocer las actividades de los usuarios de la vivienda, se ha apostado por el uso de sensores binarios no intrusivos que permiten conocer la interacción del usuario con los diferentes elementos de la *smart home* (Peetoom y cols., 2015) manteniendo la privacidad. Sin embargo, estos sensores a veces no proporcionan toda la información necesaria para el reconocimiento correcto de actividades, por lo que es necesario la incorporación de nuevos dispositivos como por ejemplo los dispositivos *wearables* (*smartwatch*, *smartphone*) o incluso dispositivos de

localización.

Además, en numerosas ocasiones dos o más personas conviven en la misma vivienda (multi-ocupación), generando situaciones que los sistemas de reconocimiento de actividades deberían gestionar para monitorizar correctamente a los diferentes individuos de la casa (Mohamed y cols., 2017).

En este proyecto se propone un sistema que permite el reconocimiento de las actividades de varios usuarios en una *smart home* de forma concurrente y en tiempo real. De esta forma, se permite la monitorización constante de estos y la actuación rápida en caso de emergencia. Además, para poder realizar tal supervisión, se incluyen diferentes tipos de sensores de última generación que no afectan a la vida que realiza el usuario en la vivienda.

1.2 Objetivos del proyecto

El objetivo principal de este proyecto consiste en la inclusión en la *smart home* de la UAL de nuevos tipos de sensores con el objetivo de recopilar más información del usuario en la vivienda y poder realizar una mejor detección de las actividades existentes. Además, se busca obtener nuevas actividades que con los sensores situados anteriormente no se podían detectar, además de permitir el reconocimiento de las actividades que realizan varios usuarios simultáneamente.

Para poder alcanzar el objetivo principal, se divide el trabajo en subobjetivos:

- Estudio de las mejoras en la infraestructura y clasificación de actividades realizada en el TFG.
- Estudio del funcionamiento de los nuevos sensores (localización, aceleración y giroscopio) a incorporar en el sistema, qué tipo de datos proporcionan y cómo interactúan con el sistema.
- Recogida de datos y análisis de los mismos en tiempo real para poder detectar las actividades que realiza el usuario en el momento que se producen. Para esto, es necesario usar algunas técnicas de análisis de datos.
- Estudio y selección del algoritmo de *machine learning* apropiado para clasificar los datos de los nuevos sensores junto con los sensores instalados en la vivienda previamente.
- Adaptación del algoritmo de reconocimiento de actividades para que permita la obtención de actividades de varios usuarios en la vivienda de forma independiente.
- Incorporación de los nuevos resultados y conclusiones en el sistema web desplegado al hacer el TFG.

1.3 Estructura del documento

En este capítulo se ha realizado una introducción al tema del trabajo que se ha llevado a cabo, así como la motivación del estudiante para desarrollar un proyecto de este tipo. Además, se especifica el proyecto implementado junto con los objetivos que se persiguen al realizar este.

A continuación, en el capítulo 2 se recoge un estudio sobre los trabajos anteriores que se han realizado en ciertos aspectos del ámbito del presente trabajo.

En cuanto al capítulo 3, se hace una descripción de las fases en las cuales se ha dividido el proyecto. Además, se muestran varios diagramas en los que se aprecia la diferencia entre la planificación del proyecto y la ejecución de este.

En el capítulo 4, se describen las herramientas que se han usado para realizar el proyecto. Dentro de estas herramientas se engloban los lenguajes de programación, las herramientas de comunicación, los entornos de desarrollo y herramientas de apoyo al desarrollo de software, las herramientas que han servido para realizar la documentación del proyecto y por último los servicios que se han usado para montar la infraestructura del proyecto.

A continuación, en el capítulo 5 se muestra la infraestructura completa del sistema desarrollado especificando la función de cada elemento del mismo y la tecnología con la cual se lleva a cabo.

El siguiente capítulo, el capítulo 6, se trata de la parte central del documento. En este se explica cómo se han desarrollado las diferentes partes del proyecto hasta conseguir una solución final. Se abordan temas como la instalación del hardware y la recogida de datos, la base de datos, el análisis de datos en tiempo real haciendo uso de machine learning y la creación de un dataset para almacenar las actividades que efectúa el usuario en la vivienda.

El capítulo 7 presenta las diferentes configuraciones en los modelos de machine learning que se han probado y los resultados de cada una de estas. Finalmente, se indica la configuración seleccionada y los resultados en una ejecución final que engloba todas las actividades que se pueden detectar en la smart home.

En el último capítulo, el capítulo 8, se exponen las conclusiones obtenidas al desarrollar el sistema, así como el trabajo futuro en este proyecto.

Posteriormente, se referencia la bibliografía de la que se ha hecho uso a lo largo de la redacción de la documentación.

Finalmente se recogen varios anexos que complementan la información de varios capítulos. En el Anexo I se muestran los diagramas de Gantt usados para la planificación y ejecución de fases; en el Anexo II se exponen las ventanas móviles del sistema web y en el Anexo III se puede visualizar la detección de actividades por parte del sistema al ejecutar la secuencia final.

2 Trabajos relacionados

Hacia el año 1993, se introdujo el término de computación ubicua (Weiser, 1993), mediante el cual, se establecen nuevos dispositivos cotidianos como fuentes de información al usuario cuyo comportamiento es similar al de un ordenador. A partir de este hecho, se comenzó a investigar sobre la utilidad de obtener información de los dispositivos que usa el usuario a lo largo de su día, llegando a la conclusión que esto puede ser una fuente de información del entorno muy precisa, pudiendo mejorar la vida de las personas. De esta forma, entre los años 2002 y 2004, se introduce el concepto de Internet de las Cosas (IoT) (Schoenberger, 2002) (Gershenfeld y cols., 2004). A partir de la definición de este, cada vez fueron aumentando las aplicaciones e investigaciones en este ámbito, debido a que se trata de un campo emergente y por lo tanto, con mucho margen de beneficios. Así, se trata de un dominio en el cual el principal objetivo consiste en mejorar la vida de las personas a través de dispositivos que se incorporan en el medio, tales como sensores, smartphones, dispositivos cotidianos, etc. Sin embargo, dentro de este mundo de IoT se destacan las aplicaciones en ciudades (Caragliu y cols., 2011), generando ciudades inteligentes adaptadas a los ciudadanos; industria (Jeong y cols., 2013) favoreciendo un incremento de la producción y salud (Farahani y cols., 2018).

Dentro de este último campo de la salud, como se ha comentado en el apartado anterior, se está apostando por la monitorización de las personas mayores, con enfermedades o discapacidad en su propio hogar (Peek y cols., 2019). Así, para poder monitorizar de forma correcta y exhaustiva a esta población, se está investigando en el ámbito del reconocimiento de actividades en entornos inteligentes como *smart homes*, formadas por una gran cantidad de sensores y dispositivos que permiten obtener información del entorno para extraer las actividades que realizan los usuarios en esta.

2.1 Dispositivos a incorporar en la *Smart Home*

Entre los diferentes tipos de sensores usados para realizar el reconocimiento de actividades (De-La-Hoz-Franco y cols., 2018) se encuentran los sensores binarios. Estos sensores tienen como objetivo mostrar el estado de ciertos elementos de la vivienda, es decir, si están activos o no. Además, dentro de este grupo de sensores se encuentran los sensores de presencia, que actúan de forma parecida a los sensores anteriormente mencionados, ya que su activación es cuando una persona se encuentra cerca. Así, este tipo de sensores no intrusivos han sido utilizados desde el inicio del reconocimiento de actividades (Medina-Quero y cols., 2018) hasta la actualidad (Hamad y cols., 2020) obteniendo buenos resultados en cuanto a la detección de actividades en *smart homes*.

Además, actualmente se están incorporando dispositivos *wearables* como fuentes de datos para IoT (Fiorini y cols., 2018). Los sensores inerciales que incorporan estos dispositivos hacen que sean muy útiles a la hora de conocer la actividad que hace el usuario en un momento concreto. Se han desarrollado multitud de aplicaciones en el ámbito de la salud con estos tipos de sensores integrados en un *smartwatch*: sistemas de detección de caídas (Gjoreski y cols., 2016) a través del uso de acelerómetros (Ngu y cols., 2017), control del consumo de medicación (Kalantarian y cols., 2016) o control de la demencia (Boletsis y cols., 2015).

Además, dado que el *smartphone* también dispone de sensores inerciales, algunas aplicaciones de IoT basadas en el *fog computing* (Bonomi y cols., 2012) buscan la interacción entre dispositivos de IoT con algún objetivo común. Algunos casos de este enfoque se pueden apreciar en aplicaciones que detectan los malos hábitos de las personas mediante un *smartphone* y un *smartwatch* (Shoaib y cols., 2015).

Por otra parte, en la actualidad se están incorporando sistemas de localización del usuario en la vivienda para determinar con mayor seguridad las actividades que realiza este en la vivienda. Se han estudiado una gran cantidad de enfoques (Ahvar y cols., 2016); sistemas GPS que pueden provocar un riesgo en la seguridad de las personas, sistemas que controlan la ubicación haciendo uso de sensores de ultrasonidos (Sainjeon y cols., 2016), asistentes de voz ((Nath y cols., 2018)), a través del uso del wi-fi de los *smartphones* (Ghourchian y cols., 2017) o usando la tecnología zigbee (Alvarez y Las Heras, 2016).

En cuanto a los sistemas de localización que se están usando actualmente para el reconocimiento de actividades destacan los sistemas RTLS. Se tratan de sistemas que establecen la posición de un objeto o persona dentro de un área a través del uso de *tags* o *chips* situadas en este. Estos componentes referencian al usuario dentro del sistema y se comunican con los componentes que delimitan la red para establecer la posicion relativa del usuario en esta. Dentro de este campo hay una gran cantidad de tecnologías (RFID, IR, WPS, Bluetooth, Zigbee, UWB, etc.) (Arnold, 2020), cada una con sus ventajas e inconvenientes. Dentro de estas, destaca la tecnología UWB (Jimenez Ruiz y Seco Granja, 2017). UWB es una tecnología de radio que hace uso de una banda mayor de 500MHz abarcando un gran rango de frecuencia. Además, tiene una alta capacidad de penetración en los materiales y un margen de error de 10 cm (Pérez, 2020).

En definitiva, para el reconocimiento de actividades se tienen en cuenta datos de diferentes tipos de sensores. Estos sensores son binarios situados en la *smart home*, acelerómetro y giroscopio situados en un *smartwatch* que llevan los usuarios y sensores UWB que portan de igual forma los usuarios cuando están en la vivienda.

2.2 Metodología de reconocimiento de actividades

A lo largo de los años, con la aparición del Internet de las Cosas, los avances en el campo de detección de actividades no han parado de suceder. Estos avances se han debido a los nuevos sensores que se han usado para realizar esta detección y a los diferentes enfoques en el modo de detección de actividades.

Debido a la gran cantidad de datos que aportan los sensores en un ambiente como una *smart home*, es necesario dividir el conjunto de datos para procesar correctamente estos. De esta forma, para realizar la segmentación en los datos se siguen diferentes enfoques (Banos y cols., 2014).

- *Ventana definida por la actividad*: Constituyen ventanas de tiempo definidas cuando ocurre el inicio y fin de ciertas actividades almacenadas. Así, se realiza una "segmentación implícita" del conjunto de datos generados, por lo que en un entorno de tiempo real no se utiliza ya que no se puede conocer a priori el comienzo y fin de la actividad.

- *Ventana definida por eventos*: En este enfoque, se establecen ciertas secuencias ordenadas de acciones o movimientos para definir una actividad. La secuencia puede prolongarse en el tiempo de forma diferente cada vez que se realiza, pero en este caso, no influye en la detección de la actividad.

- *Ventanas de tiempo deslizantes*: En esta tercera aproximación, definida en (Espinilla y cols., 2018), se tratan ventanas de tiempo de tamaño fijo en las cuales se engloban los datos que se reciben de los sensores. Así, el trabajo del algoritmo solamente consiste en definir la actividad que ha ocurrido en los intervalos de tiempo definidos. Esto hace que el preprocesamiento de los datos no sea tan elevado como en los casos anteriores y se pueda analizar en tiempo real los datos procedentes de los sensores (Medina-Quero y cols., 2018), ya que la información que se usa en este caso no depende de la información que viene a continuación en sucesivas ventanas.

Entre los tres enfoques de particionamiento de datos procedentes de los sensores, el enfoque más empleado en la detección de actividades en tiempo real es el tercero (ventanas de tiempo deslizantes). Por lo tanto, en este trabajo se emplea este enfoque, adaptado a los sensores disponibles y a las capacidades del sistema.

Concretamente, la detección de actividades usando ventanas de tiempo deslizantes sigue una serie de etapas según se muestra en la Figura 2.1 definida en (Wang y cols., 2018). En primer lugar, se realiza la obtención de datos y un preprocesamiento para cerciorarse de la corrección de los datos que se usan. En segundo lugar, se dividen los datos en las ventanas de tiempo definidas en el algoritmo. Posteriormente, una vez divididos los datos en las ventanas correspondientes, se obtienen las características principales de estos datos que son usadas por el algoritmo (media, desviación, mínimo, número de veces que se repite un dato, etc...). Una vez se tienen las características de las ventanas de tiempo que mejor representan los datos, se clasifican haciendo uso en este caso de algoritmos de *machine learning*, muy usados en este ámbito por su baja carga computacional y su alto grado de acierto (De-La-Hoz-Franco y cols., 2018).

De esta forma, los modelos de *machine learning* se llevan usando desde la aparición de las *smart homes* para diversas tareas como ahorro en el consumo de energía (Lai y cols., 2013) (Lin y Chen, 2017) y reconocimiento de actividades. En este trabajo, se indaga en el reconocimiento de actividades en *smart homes*. Para realizar este reconocimiento, en la mayor parte de las ocasiones se ha hecho uso de modelos de machine learning, debido a sus altos grados de acierto y su facilidad de entrenamiento y uso (Casale y cols., 2011) (Feng y cols., 2015).

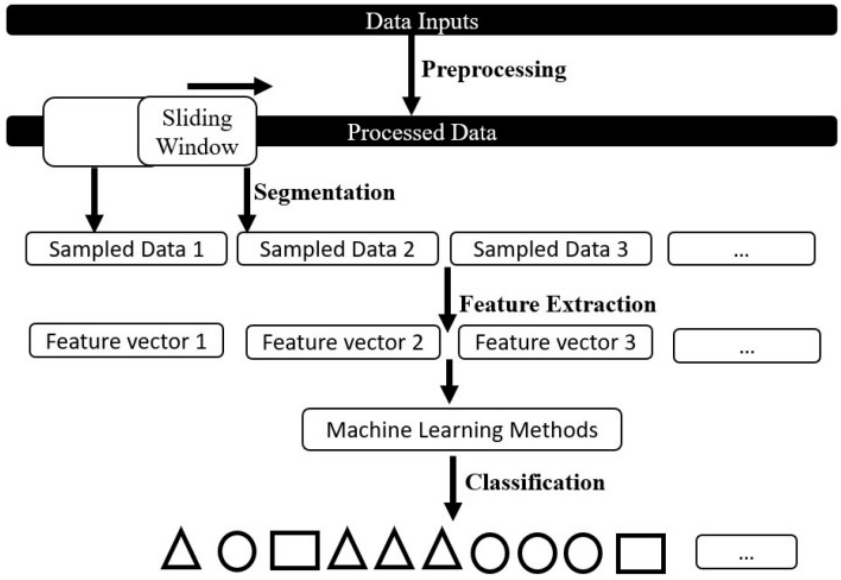


Figura 2.1: Fases Detección Actividades

3 Fases del proyecto

Para la realización del proyecto y el cumplimiento de los objetivos anteriormente mencionados, en primer lugar se dividió el proyecto en diferentes fases con objetivos muy definidos en cada una de ellas para fraccionar el trabajo e ir comprobando el estado de su ejecución. Así, a continuación se mostrará la planificación inicial de las distintas fases y posteriormente las fases que se siguieron para completar el proyecto, pudiendo comparar la diferencia entre ellas.

3.1 Planificación

En primer lugar, una vez definido el proyecto y estimado su alcance, se realizó una lista de tareas a cumplir a lo largo de este proyecto. Además, se dividieron estas tareas en diferentes fases estimando un número de horas a cada fase teniendo en cuenta la experiencia en el desarrollo del Trabajo Fin de Grado. En la Figura 3.1 se pueden observar las fases en las que se planificó el proyecto junto con el número estimado de horas de cada una de ellas. Una vez se establecieron las

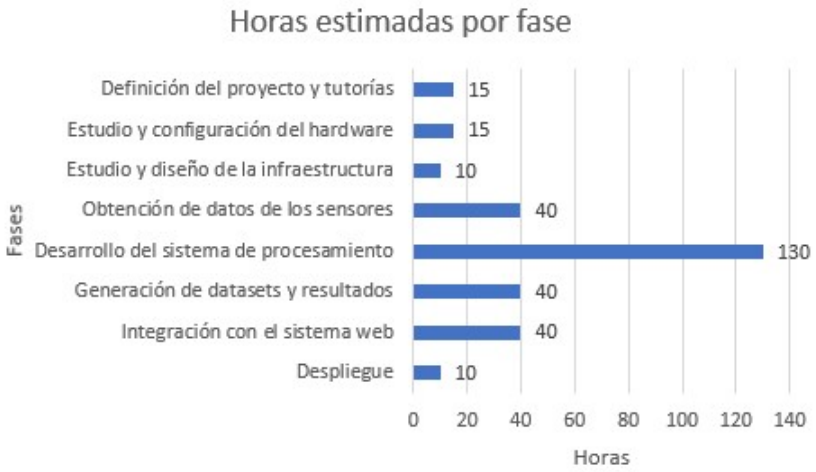


Figura 3.1: Duración en horas de cada fase - Planificación

diferentes fases con las horas estimadas correspondientes, se distribuyeron a lo largo del tiempo con el objetivo de distribuirlas de la forma más eficiente, como se puede observar en el diagrama de Gantt del Anexo 9.1.

La estimación sirvió para tener una visión global de las tareas a realizar y distribuirlas en el tiempo para ejecutar el proyecto correctamente. Sin embargo, a lo largo de la ejecución del proyecto, surgieron subtareas y reajustes en la planificación inicial.

3.2 Ejecución

En esta sección se muestran las fases en las que se dividió el proyecto en su ejecución real. Para cada fase, se especifica el número de horas empleadas en las tareas que conlleva su realización. Como se ha tratado de reflejar de forma precisa las horas empleadas en cada fase, rellenando a lo largo de la realización del proyecto, un archivo de todas las horas empleadas en este y las tareas que se llevaron a cabo en cada una de ellas. Así, las fases fueron las siguientes:

Asistencia a curso sobre IoT y diversas tutorías con la tutora y cotutor (18 horas) En esta primera fase, se asiste a un curso sobre IoT en la actualidad y se presentan diferentes tecnologías que posteriormente se usan en el presente trabajo, como es el caso de MQTT. Además, a raíz de la asistencia a este curso, se establece el tema del trabajo y se efectúan diversas reuniones para comentar las fases del proyecto y el alcance del mismo.

Obtención, estudio y configuración del hardware a incluir (15 horas) En esta fase se estudia y se requiere el hardware necesario para realizar el proyecto. Una vez recibido, se estudia la documentación asociada al mismo y se configura este, efectuando diversas pruebas con el objetivo de comprobar su correcto funcionamiento y la adecuación de los datos con el objetivo del proyecto.

Prueba del hardware y realización de pequeños ejemplos para comprobar su funcionamiento (15 horas) En esta fase se ejecutan pequeños scripts que recogen los datos de los nuevos sensores, se almacenan y se procesan para clasificar un conjunto de actividades de prueba. Se hace esta fase para asegurar que el sistema mejora los resultados respecto al Trabajo Fin de Grado correspondiente.

Estudio y diseño de la nueva infraestructura del sistema (15 horas) Dado que se incorporan nuevos sensores al sistema y proporcionan una gran cantidad de datos por segundo, es necesario diseñar una infraestructura que soporte esto. Así, resulta imprescindible aprender y aplicar técnicas de big data con el objetivo de poder analizar toda la cantidad de datos necesarios para detectar las nuevas actividades.

Desarrollo del sistema (90 horas) Se trata de la fase principal del proyecto. En esta fase se desarrollan de los scripts que recogen los datos en tiempo real, los procesan generando las

actividades que ha llevado a cabo el usuario y los almacenan en la base de datos para que estos datos puedan ser analizados y mostrados posteriormente.

Generación de datasets y comprobación de resultados (45 horas) En esta fase se desarrollan los scripts necesarios para la creación de datasets por parte del usuario que recogen las actividades que se van a detectar en la vivienda inteligente. Además, se implementa la visualización de los resultados con el fin de comprobar que estos resultados son correctos y acordes a las actividades que se realizan en la vivienda.

Despliegue del sistema (10 horas) En esta fase se lleva a cabo la configuración y despliegue del sistema implementado. Se realiza la puesta a punto de algunos elementos que forman parte del sistema como máquinas virtuales y base de datos. Sin embargo, debido a la situación de excepcionalidad, no fue posible desplegar los sensores y los elementos controladores en la *smart home*.

Incorporación de resultados en el sistema web (22 horas) Una vez efectuado el análisis de los datos en tiempo real y obtenidas las actividades del usuario en la vivienda, estos datos se incluyen en el sistema web desplegado en el Trabajo Fin de Grado para incorporar esta nueva funcionalidad. De esta forma, estos datos son accesibles desde cualquier lugar con conexión a Internet.

Interacción con el sistema implementado (20 horas) En esta fase, se interactúa con el sistema con el propósito de comprobar que el sistema funciona correctamente y se obtienen las actividades reales de forma correcta. Dada la situación de excepcionalidad no ha sido posible instalar el sistema en un entorno real de *smart home*, pero se trató de implementar una simulación del sistema lo más parecido posible a una *smart home*.

Redacción de la memoria (50 horas) A lo largo del proyecto se fueron recogiendo los pasos que se iban realizando para completar el proyecto. Así, una vez finalizada la implementación del sistema, se comenzó a redactar la memoria final.

En la Figura 3.2 se puede observar a modo de resumen las horas que se han empleado en cada fase. Como se puede apreciar en esta, la fase en la cual se ha desarrollado el sistema ha sido la que más tiempo ha llevado, con 90 horas. En total, la duración del proyecto es de 300 horas.

La distribución de las tareas en el tiempo se puede observar en el diagrama de Gantt del Anexo 9.2.

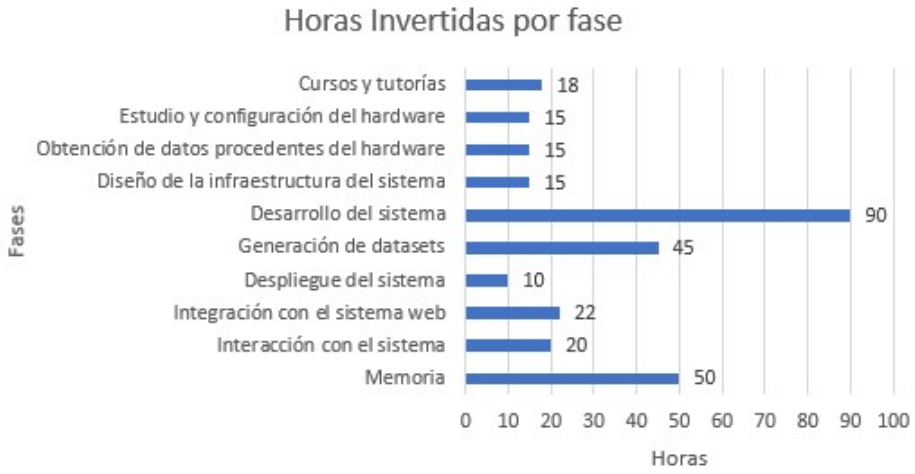


Figura 3.2: Duración en horas de cada fase - Ejecución

4 Herramientas

En este capítulo se va a realizar una descripción de las herramientas que se han usado para desarrollar el trabajo. Además de la descripción de estas herramientas, se va a explicar por qué se ha decidido usar cada una de ellas en lugar de cualquier otra. Para organizar la descripción de las herramientas, estas se han dividido en cuatro categorías.

4.1 Lenguajes de programación

En esta categoría se describen los lenguajes de programación que se han usado para la ejecución el proyecto.

4.1.1 JavaScript

JavaScript se trata de un lenguaje de programación orientado al desarrollo web. Este lenguaje se basa en *C*, y es un lenguaje orientado a objetos. Al principio, se usaba solamente en el lado del cliente para dinamizar las vistas que se muestran a este; sin embargo, también permite actuar en el lado del servidor, como se puede apreciar en el *framework* NodeJS.

El sistema web que se ha desarrollado está basado en NodeJS, ReactJS y Express. Estos *frameworks* están basados en JavaScript, por lo que ha sido el lenguaje predominante en el desarrollo del sistema web. Se ha decidido hacer uso de este lenguaje ya que es de un lenguaje optimizado para el desarrollo de sistemas web, el cual puede ser utilizado para desarrollo *back-end* y *front-end*.

4.1.2 HTML

Se trata de un lenguaje de programación basado en etiquetas cuyo principal objetivo es la creación de páginas web. Este es un estándar dentro del desarrollo web, ya que fue establecido por *W3C*[1]. Todos los navegadores soportan este lenguaje, ya que se usa para mostrar las ventanas al usuario.

Dado que se ha desarrollado un sistema web, se ha usado este lenguaje de programación para

[1] World-Wide-Web Consorcium

realizar el *front-end* del mismo. Se ha decidido usar este lenguaje porque no hay otra opción de cara a implementar la estructura de etiquetas de un sistema web.

4.1.3 CSS

Se trata del lenguaje de programación usado para personalizar y diseñar el estilo de las ventanas que se incluyen en el sistema web. Este lenguaje surgió con la necesidad de separar el contenido de una página web o documento de su visualización, por lo que con este lenguaje se permite personalizar la visualización tanto de documentos como de páginas web. Actualmente, es el lenguaje más usado para establecer los estilos de las páginas web.

Dada la gran cantidad de ventanas a desarrollar, ha sido necesario hacer uso del *framework* MaterializeCSS. Este *framework* incluye un conjunto de clases CSS que se pueden aplicar a los elementos incluidos en formato HTML. Todas estas clases se llevan a cabo siguiendo el diseño *Material Design* de Google tan usado en la actualidad. Además, este *framework* incluye funciones descritas en JavaScript que favorecen el dinamismo de estos elementos.

4.1.4 LaTeX

LaTeX se trata de un lenguaje de tipografía de alta calidad. Con este lenguaje se pueden crear documentos con bastante grado de personalización y calidad, como por ejemplo documentos científicos y publicaciones. Este lenguaje está formado por órdenes construidas a partir de comandos *TeX*, por lo que es un lenguaje de bajo nivel. Además, es un software de código abierto, por lo que está en constante evolución y mejora.

Dada su gran aceptación en la redacción de trabajos de investigación y memorias de proyectos, se ha hecho uso de una plantilla para redactar la memoria final del proyecto. Además, se ha decidido usar este lenguaje por la facilidad con la que se pueden personalizar los elementos que se incluyen en el documento.

4.1.5 Python

Python se trata de un lenguaje de programación *Open Source*, cuya sintaxis es bastante sencilla, ya que se trata de un lenguaje definido a muy alto nivel. Además, soporta la orientación a objetos y es un lenguaje interpretado, es decir, no hace falta compilarlo para ejecutarlo; por ello se usa en una gran cantidad de *scripts*. Además, a parte de su fácil comprensión y gran uso, contiene un conjunto muy amplio de librerías, lo cual facilita al usuario las tareas de programación.

Se ha decidido hacer uso de este lenguaje de programación para la realización del procesamiento de los datos procedentes de los sensores. Gracias al uso de este lenguaje, se han podido

usar librerías como *scikit-learn*, que han ayudado en la creación del algoritmo de reconocimiento de actividades y *pyMongo*, capaz de efectuar una monitorización en tiempo real de datos almacenados en una base de datos MongoDB.

4.2 Comunicación

En esta categoría se engloban los medios a través de los cuales se ha llevado a cabo la comunicación entre los tutores y el alumno a lo largo del desarrollo del proyecto.

4.2.1 Correo electrónico

Se ha hecho uso de esta herramienta para el intercambio de mensajes entre alumno y los tutores. Gracias a esta herramienta, se tiene un historial de los mensajes intercambiados, pudiendo acceder a la diferente información que se ha tratado en ellos a lo largo del tiempo y siendo fácilmente accesible por ambas partes. También se ha usado esta herramienta para fijar reuniones y tutorías en lugar de usar alguna otra plataforma de mensajería instantánea.

4.2.2 Informes

A lo largo del desarrollo del trabajo, se han redactado varios informes para exponer ideas sobre el sistema a desarrollar, así como para el resumen y envío de bibliografía. Gracias a la realización de estos informes, se ha tenido una gran cantidad de información documentada lista para incluir en la memoria final.

4.2.3 Reuniones

Han sido necesarias algunas reuniones con los tutores para definir el ámbito del proyecto y resolver dudas que han ido surgiendo a lo largo del mismo. Se trata de la forma más efectiva de comunicación posible entre ambas partes, ya que la interacción se realiza en persona.

4.3 Desarrollo

En esta categoría se hace una pequeña descripción de los entornos de desarrollo y herramientas usados para la programación del trabajo. Además, se justificará su uso en lugar de otro software con funcionalidad similar.

4.3.1 Visual Studio Code

Se trata de un *IDE* de código abierto, enfocado a la programación en JavaScript dada su rapidez y capacidad de personalización. La principal característica de este software consiste en instalar y configurar complementos para personalizar la experiencia del usuario cuando se programa. A pesar de estar enfocado para el desarrollo de código web (en JavaScript), se puede usar para cualquier lenguaje de programación, pudiendo instalar complementos de todos los lenguajes y adaptándose a estos.

Se ha decidido hacer uso de Visual Studio Code para la codificación del sistema web completo. Dado que este está basado en JavaScript, se ha usado Visual Studio Code ya que se trata de un *IDE* optimizado para este lenguaje de programación. Este incluye numerosos *plugins* que permiten personalizar al usuario los diferentes aspectos del código como su visualización y formato, lo cual ha facilitado la programación. Una funcionalidad importante ha sido poder crear áreas de trabajo, permitiendo tener varios proyectos abiertos a la vez, favoreciendo el intercambio de archivos entre ellos.

4.3.2 Android Studio

Android Studio se trata del IDE oficial para el desearrollo de aplicaciones en Android. El entorno de desearrollo está desarrollado por Google y la primera versión estable vio la luz hacia diciembre del año 2014. Dado que para la programación de APPs nativas en Android es necesario probar la APPs, Android Studio permite la creación de simuladores de algunos smartphones con las distintas versiones de Android y tamaños. Además, permite el testeo de las APPs en dispositivos reales. Otra característica importante se trata de que permite el diseño de la interfaz gráfica haciendo uso de un editor gráfico, facilitando la creación de las distintas ventanas de las APPs y reduciendo la cantidad de código que se tiene que programar. Por otra parte, permite comparar la APK, detectar y eliminar contenido superfluo reduciendo el tamaño de las APPs. Además, muestra diferentes gráficos sobre el consumo de recursos (CPU, memoria y actividad de red) cuando se está ejecutando la aplicación en un dispositivo real.

Se ha hecho uso de Android Studio ya que proporciona todas las herramientas necesarias para la programación de APPs nativas en Android. En el caso de querer programar APPs haciendo uso de otros entornos de desarrollo, habría que instalar algunos componentes que se instalan al instalar Android Studio, como el Android SDK (herramientas que son necesarias para ejecutar APPs en Android).

4.3.3 Git

Git se trata de un sistema de control de versiones de código abierto. Además, se trata de un sistema de control de versiones distribuído, ya que no solamente se tiene un repositorio central en el cual van a estar todos los cambios, sino que se tiene un repositorio central y otro repositorio para cada usuario que participe en este; de esta forma, se tiene una copia de los datos para cada

usuario y solamente se compartirán los datos que cada usuario quiera, siendo esta subida/bajada de datos una operación bastante rápida ya que no se tiene que hacer sobre todo el contenido, sino sobre estas modificaciones que se hagan.

Se ha hecho uso de esta herramienta para realizar un control de código fuente sobre el sistema que se ha desarrollado. Concretamente, se ha usado *GitHub*, que permite tener repositorios privados de forma gratuita y actualmente se trata de la herramienta más usada para control de versiones Git. Gracias a este sistema se ha podido volver a versiones anteriores del código cuando ha sido necesario. Además, se ha usado ya que los *IDEs* que se han comentado anteriormente tienen una integración directa con este software, por lo que se han podido subir, actualizar y revertir los cambios de forma sencilla.

4.3.4 Postman

Se trata de una herramienta desarrollada para ejecutar peticiones *HTTP* a cualquier *API*[2]. De esta forma, se puede testear fácilmente una *API* que se esté desarrollando o cualquier *API* disponible en la red. Esta herramienta está bastante optimizada y en este momento, numerosos *IDEs* están intentando incorporarla de forma nativa.

Aunque algún *IDE* incluye esta herramienta de forma nativa, se ha hecho uso de este *software* porque permite la creación de peticiones a APIs internas y de terceros, además de la creación de entornos de trabajo con variables globales y locales. Además, gracias a la facilidad de uso de esta herramienta, se ha ahorrado tiempo testeando la *API* que se ha desarrollado, al no ser necesaria la realización de consultas a esta desde el sistema, sino directamente desde Postman.

4.3.5 MQTTfx

Esta herramienta consiste en un cliente de MQTT gratuito que permite conectarse a un bróker de MQTT instalado en la máquina local o en una máquina externa y visualizar el estado del bróker. Además, se permite la suscripción a topics, el envío de datos al bróker y la visualización de logs. Está disponible para Windows, macOSX y Linux.

4.3.6 Balena Etcher

Se trata de una herramienta Open Source desarrollada por Balena que permite escribir archivos de imagen (archivos .img o.iso por ejemplo) en unidades de almacenamiento como pen-drives, tarjetas SD o discos duros. Además, dado que está permitida para los sistemas operativos Linux, macOSX y Windows, se trata de una herramienta universal para esta tarea. Además, la operación de grabación de imágenes está optimizada, reduciendo hasta un 50% el tiempo que se trata en llevar a cabo la escritura respecto a otras plataformas.

[2]Interfaz de programación de aplicaciones

4.4 Documentación y productividad

En esta sección se muestran las herramientas que se han usado para realizar la documentación del trabajo desarrollado.

4.4.1 TexStudio

Se trata de un editor de código LaTeX , con todas las ventajas que aportan los editores de textos. Además, permite una visualización de los cambios del documento en tiempo real, y la corrección y muestra de errores de sintaxis a medida que se va generando el documento.

Esta herramienta se ha usado porque se permite su uso en Windows, sistema operativo en el cual se ha trabajado a lo largo de la redacción del proyecto. Además, está especialmente diseñado para la creación de documentos en lenguaje LaTeX , usado para la redacción de la memoria final.

4.4.2 GanttProject

El diagrama de Gantt se trata de una de las herramientas más usadas para la gestión de proyectos. Con este diagrama se planifican las tareas a lo largo del tiempo con el objetivo de desarrollar el proyecto de forma correcta. Esta herramienta fue inventada por Henry L. Gantt entre los años 1910 y 1915. Gracias a esta herramienta, se puede ver de una manera general el tiempo que se ha invertido en cada una de las tareas así como el orden en el que se han realizado y las precedencias de estas entre sí.

El software Gantt Project se trata de un software de código abierto disponible para Windows, OSX y Linux que permite la realización de estimaciones en forma de diagramas de Gantt. Se ha decidido hacer uso de esta herramienta por su facilidad de uso y su gratuidad. Además, se ha encontrado de bastante utilidad la opción de comparar la estimación con la ejecución del proyecto, permitiendo ver las diferencias entre ambas.

4.4.3 Google Drive Docs

Actualmente, Google Drive, servicio de Google en la nube para el almacenamiento y gestión de archivos creado en 2012, permite la creación de archivos en la nube con un editor similar al proporcionado por herramientas como Word. La herramienta con la que se puede llevar a cabo esto es Google Drive Docs. En esta aplicación en la nube se permite la creación, modificación, compartición y almacenamiento de documentos, con todas las ventajas que ello conlleva. Entre algunas de las ventajas se encuentran por ejemplo, el trabajo en grupo sobre un documento o el guardado automático con cada cambio que se realice en el archivo.

Se ha hecho uso de esta herramienta ya que se trata de una utilidad que se encuentra dispo-

nible de forma gratuita en la nube. Así, no requiere instalación y se mantienen los documentos actualizados en todo momento, no siendo necesaria la copia de seguridad de estos documentos. Además, permite tener un historial de los cambios y volver atrás cuando se requiera. Se ha usado para la redacción preliminar de la memoria final y la redacción de distintas anotaciones a medida que se iba desarrollando el proyecto.

4.4.4 Google Drive Sheets

Esta herramienta está integrada también en Google Drive. Así, permite la creación de hojas de cálculo tal y como hace Excel, pero con todas las ventajas de estar en la nube y en el mismo paquete que Google Drive Docs. De esta forma, los archivos que se generan en las herramientas de Google Drive tienen un alto grado de acoplamiento entre ellos.

Esta utilidad está incluida dentro del mismo paquete que la herramienta anterior, con todas las ventajas mencionadas anteriormente. En este caso, se ha usado esta herramienta para calcular el total de horas de cada fase, apuntar las horas de las tareas que se han llevado a cabo en el proyecto y el diseño de gráficos incluidos en la presente memoria.

4.4.5 Trello

Trello se trata de una aplicación web/móvil cuyo objetivo es proporcionar al usuario una herramienta en la cual gestionar los proyecto mediante paneles KANBAN. Además, permite editar y usar diferentes tableros KANBAN de forma colaborativa entre varios usuarios. Así, se trata de un software gratuito con las principales funciones disponibles pero se puede ampliar a una funcionalidad premium para conseguir todas las ventajas que proporciona.

En este proyecto, se ha usado Trello para gestionar todas las tareas del proyecto, así como en la documentación del mismo. Además, se han planificado las fechas límite para tener un control de la temporalización del proyecto.

4.5 Infraestructura

A continuación, se realiza una descripción de los servicios que se han usado para configurar la estructura IoT.

4.5.1 OpenStack

Se trata de un sistema cloud computing de software libre, permitiendo la generación de IaaS[3]. Esta plataforma permite la creación de un entorno cloud que permita la creación y uso de máquinas virtuales o espacios de almacenamiento por parte del usuario.

Dado que al ser alumno de la Universidad de Almería se tiene acceso al servicio de máquinas virtuales desplegado en OpenStack, se ha hecho uso de máquinas virtuales alojadas en este sistema ya que se requiere la ejecución continua de scripts de obtención y procesamiento de datos. Estos scripts tienen que estar siempre en ejecución, por lo que requiere instalarse en un sistema que nunca se apague, siendo inviable su ejecución en el PC portátil o de casa ya que se consumirían muchos recursos.

4.5.2 Heroku

Se trata de un servicio PaaS[4], que permite el despliegue de aplicaciones de forma sencilla, y casi automática, liberando al administrador de aplicaciones de las tareas de configuración y entorno de las diferentes aplicaciones. De esta forma, se permite un rápido despliegue de aplicaciones indicando únicamente la tecnología de back-end que se va a usar.

Ha sido necesario usar Heroku para desplegar el sistema web con la finalidad de ser accesible desde cualquier lugar, no solamente desde la universidad o desde la red local de casa. Así, se ha accedido desde fuera de casa al sistema para poder llevar a cabo el entrenamiento del algoritmo mediante la corrección e introducción de datos derivados de acciones realizadas en la vivienda. Además, se trata de una herramienta gratuita siempre que no se superen los 500mb de almacenamiento.

4.5.3 MongoDB Atlas

Se trata de un servicio DaS[5], que permite la creación de *clusters* de bases de datos en la nube (con todas las ventajas que esto conlleva), pudiendo acceder a ellas desde cualquier aplicación que se desarrolle. Este servicio se encarga de la configuración e instalación de las base de datos en los clusters, por lo que las tareas del administrador solamente consisten en usar estas bases de datos.

Se ha usado este servicio de base de datos en la nube debido a que ha sido necesario crear una base de datos común a varias aplicaciones y *scripts*, por lo que la única forma de conseguir esto ha sido haciendo uso de este servicio, que proporciona bases de datos MongoDB (el tipo de bases de datos usadas en el proyecto). Además, dado que esta plataforma tiene varios planes de

[3]Intraestructura como servicio
[4]Plataforma como servicio
[5]Base de datos como servicio

pago, se ha usado el plan gratuito, que se caracteriza principalmente por tener una cantidad de memoria de 512 MB y la memoria RAM compartida con otros clusters.

4.5.4 Apache Kafka

Apache Kafka se trata de una plataforma Open Source encargada de llevar a cabo un streaming de datos de forma distribuida. De esta forma, está diseñada para recibir datos de diversas fuentes y repartirlos entre procesos para hacer un procesamiento de los mismos de forma eficiente.

Actualmente, se trata de una plataforma que se usa con bastante frecuencia en el campo de Internet de las Cosas. Según Gartner, en el año 2020 aproximadamente estarán conectados alrededor de 20 mil millones de dispositivos (Gartner, 2020), por lo que el procesamiento de los datos que estos generan deben ser procesados de forma eficiente y Kafka permite este procesamiento ya que distribuye la carga entre diferentes procesos.

4.5.5 Eclipse Mosquitto

MQTT (Message Queue Telemetry Transport) se trata de un protocolo de comunicación Open Source encargado de la publicación/suscripción de mensajes. Este protocolo fue diseñado para proporcionar al Internet de las Cosas un protocolo capaz de consumir muy pocos recursos y permitir el paso de mensajes entre dispositivos. De esta forma, se establece un bróker en el cual se almacenan los mensajes procedentes de varias fuentes y organizados en ”topics”, y al mismo tiempo se tienen varios procesos o dispositivos (consumidores) encargados de recibir los datos que se incluyen en los ”topics” a los que está suscritos. De esta forma, se permite el paso de mensajes entre los diferentes dispositivos o procesos y cada uno de ellos recibe los datos que realmente necesita.

Concretamente, se hace uso de Eclipse Mosquitto, software Open Source desarrollado por la fundación Eclipse, programado en C y multiplataforma, que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT. Se ha decidido usar este bróker por su bajo consumo de recursos y su adecuación a servidores de baja potencia, en este caso, los dispositivos Raspberry.

4.5.6 Redis

Redis se trata de un almacén de datos en memoria principal cuyo objetivo es proporcionar al usuario una base de datos cuyo acceso de lectura y escritura es muy rápido, ya que está programado en C y almacena los datos en memoria principal, a diferencia de otras bases de datos que se almacenan en disco. De esta forma, en la mayoría de las ocasiones se usa como memoria caché o bróker de mensajes. En este caso, se requieren almacenar valores calculados para que posteriormente estos datos sean usados por varios procesos de forma independiente, por lo que una vez calculados, se almacenan en el servicio Redis simulando una caché y el acceso a estos por parte de otros procesos es muy eficiente.

5 Metodología

En este capítulo se analizan los conceptos básicos del sistema que se ha desarrollado en el actual trabajo. En primer lugar, se indica la infraestructura que se ha diseñado y los diferentes elementos que forman esta. Posteriormente, se analizan los tipos de dispositivos que actualmente se incorporan en las *smart homes* para mejorar la vida del usuario. Finalmente, se define la metodología para realizar la detección de las actividades en la vivienda. De esta forma, en este capítulo se recogen las bases del sistema desarrollado y por qué se hace uso de cada tecnología.

5.1 Infraestructura del sistema

El proyecto consiste en un sistema de Internet de las Cosas que procesa los datos de diversas fuentes de sensores para detectar las actividades que hacen varios usuarios en una vivienda. Debido a la gran cantidad de dispositivos y datos que engloban la detección de las actividades, se ha propuesto una infraestructura capaz de soportar todas las tareas de recopilación, procesamiento, almacenamiento y visualización de datos. En la Figura 5.1 se muestra todo el flujo de la información desde que se originan los datos hasta que se reflejan los resultados. Así, en los sistemas IoT se pueden apreciar diferentes capas o subsistemas con diferente funcionalidad. En este caso, se tienen principalmente dos subsistemas que interactúan entre sí para que la información fluya de forma bidireccional y el sistema funcione correctamente.

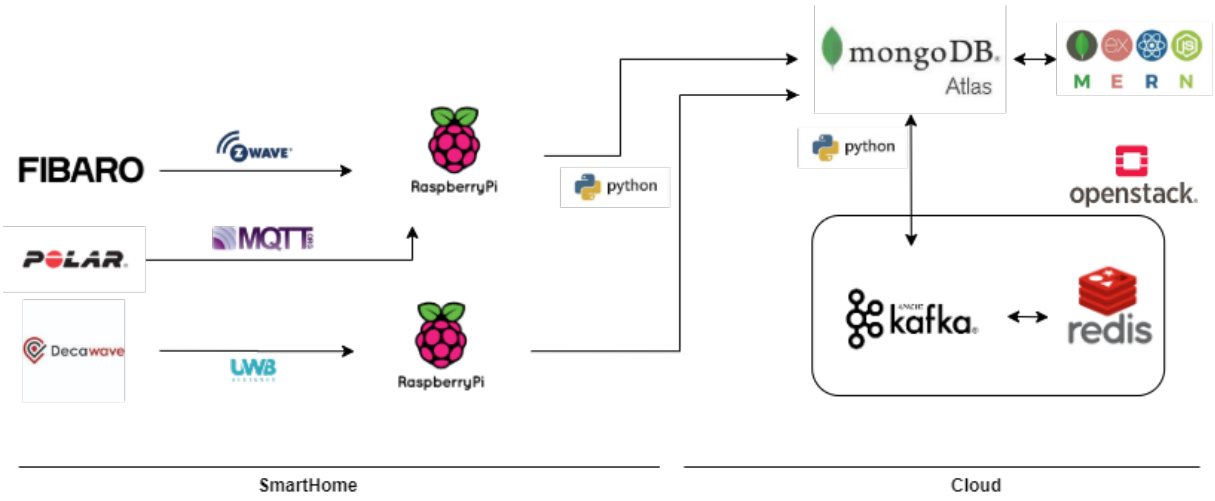


Figura 5.1: Infraestructura del sistema

5.1.1 Sistema en la niebla neblina - *smart home*

En primer lugar se tiene el sistema en la niebla o neblina. De forma general, constituye el sistema que realiza la recogida de datos por parte de los sensores y estos datos son enviados al dispositivo controlador para que este los procese en caso de ser necesario, y los almacene. Los componentes que forman parte de este subsistema son los siguientes:

Sensores Son de los dispositivos que transforman magnitudes físicas en señales eléctricas. Por lo tanto, son los encargados de recoger los cambios que se producen en el medio. En este sistema, se incorporan tres tipos de sensores. En primer lugar, los sensores inerciales incorporados en un *smartwatch*, que se encargan de obtener los gestos y acciones que realiza el usuario cuando este lleva el *smartwatch* puesto. En segundo lugar, sensores de localización que ubican al usuario dentro de la vivienda en todo momento. En último lugar, se incluyen sensores binarios incorporados de forma estática en la *smart home*; siendo los sensores que formaban el sistema IoT desarrollado en el Trabajo Fin de Grado.

Controladores Son los dispositivos encargados de recibir los datos de los sensores y realizar diversas acciones con ellos. En este caso, los controladores son dos dispositivos Raspberry (ordenadores a pequeña escala) cuya función es recibir los datos de los sensores, comprobar que son correctos y mediante varios scripts, incorporarlos en una base de datos alojada en la nube. Por tanto, estos dispositivos actúan de intermediarios entre los sensores y los servicios en la nube para el procesamiento de grandes cantidades de datos.

Protocolos de comunicación Los sensores, para poder enviar datos a los controladores tienen que soportar un protocolo de comunicación mediante el cual realizar esta conexión. En la infraestructura desplegada en este trabajo, se hace uso del protocolo de comunicación ZWave en el caso de sensores binarios, MQTT en el caso de sensores inerciales y UWB en caso del sistema de localización.

El sistema en la niebla o neblina se encuentra en el entorno físico de la *smart home*.

5.1.2 Sistema en la nube - Cloud

El IoT no tiene como objetivo recoger datos y almacenarlos, sino darle una utilidad a estos datos para mejorar la vida de las personas. Por lo tanto, una vez obtenidos los datos de los sensores en el sistema en la niebla o neblina explicado anteriormente, es necesario procesar estos datos y proporcionar valor final. Para realizar las tareas de procesamiento de grandes cantidades de datos, los dispositivos como Raspberry no tienen recursos suficientes, por tanto, son necesarias infraestructuras que soporten este procesamiento. Estas infraestructuras en la actualidad se proporcionan como servicios en la nube ya que se pueden usar los recursos que sean necesarios en cada momento y escalar las aplicaciones cuando se requiera. Por tanto, los servicios que se usan en la nube para procesar los datos del sistema en la niebla o neblina son los siguientes:

MongoDB Atlas Es un servicio de base de datos MongoDB que permite alojar bases de datos no relacionales en la nube (DaaS[1]). De esta forma, para acceder a la base de datos solamente se necesita acceso a internet, por lo que cualquier máquina o servicio con conexión puede hacer uso de los datos de esta.

Sistema web Constituye el servicio web que muestra información acerca de la *smart home* y los sensores que están incorporados en esta. Además, recoge la nueva funcionalidad de detección de actividades desarrollada en el presente trabajo. El sistema web está optimizado para PC, móvil y tablet, permitiendo al usuario visualizar la información sobre la *smart home* desde dentro de esta o deste el exterior, ya que se encuentra desplegada de forma pública en internet.

Sistema de máquinas virtuales en OpenStack OpenStack se trata de un servicio IaaS[2] que usa la Universidad de Almería para tener un entorno de máquinas virtuales en la nube accesibles por los alumnos. En este sistema, se ha hecho uso de una máquina virtual en la cual se han incorporado las herramientas Apache Kafka y RedisDB. En esta máquina virtual se ha creado un algoritmo de reconocimiento de actividades que hace uso de los servicios que proporcionan Apache Kafka y RedisDB. El primero de ellos permite distribuir la carga entre varios procesos y el segundo de ellos permite almacenar resultados parciales en memoria principal para acceder a ellos desde los diferentes procesos de forma muy rápida. Por lo tanto, en la máquina virtual desplegada en OpenStack, se ejecuta de forma indefinida el algoritmo de detección de actividades.

5.2 Análisis de las actividades

En el sistema desarrollado en este trabajo, se ha construido un modelo de reconocimiento de actividades. El modelo recibe como entradas las características extraídas de diferentes sensores haciendo uso de ventanas de tiempo cambiantes en tiempo real para realizar el reconocimiento de actividades. Concretamente, se usan varios tamaños de ventanas para procesar los diferentes datos que proporcionan los sensores incorporados en el sistema.

De esta forma, la ventana base se establece en 0.5 segundos, ya que se ha demostrado que para sensores inerciales recoge correctamente la actividad que realiza el usuario debido a la gran cantidad de datos que se proporcionan por instante de tiempo. En el caso del *tag* de localización que lleva el usuario, la ventana se establece en 1.5 segundos, ya que la frecuencia de las actualizaciones es menor y en el caso de los sensores binarios incorporados en la *smart home*, es de 5 segundos.

Estableciendo las ventanas base de los diferentes tipos de sensores, se tiene una ”imagen” del estado actual de la persona en el momento concreto sin atender a las actividades o movimientos que ha realizado esta persona con anterioridad. Para paliar esto, se crean nuevas ventanas de tiempo anteriores a la ventana base. De esta forma, se puede determinar de mejor forma la

[1] Database as a service
[2] Infraestructure as a Service

actividad que se realiza en un momento dado.

Las ventanas que se han definido para cada tipo de sensor son las que se visualizan en la Figura 5.2. Por lo tanto, cada medio segundo es necesario recoger los datos correspondientes de los sensores en cada ventana de tiempo definida.

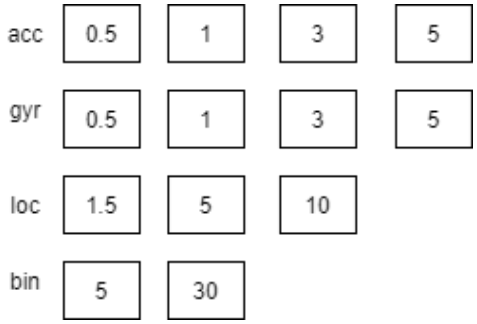


Figura 5.2: Ventanas de tiempo definidas por tipo de sensor

Por otra parte, como se establece en la tercera fase de la detección de actividades mencionada anteriormente, una vez que se tienen los datos de los sensores divididos en ventanas de tiempo, es necesario extraer las características más importantes de estos. En el caso de los sensores inerciales, se usan valores como el mínimo, máximo, media y desviación típica (Figo y cols., 2010). En este caso, se hace uso del máximo, mínimo y media. Para el caso de los sensores de localización se hacen uso de las mismas métricas. Sin embargo, en el caso de los sensores binarios, se tiene en cuenta el estado de los sensores en cada ventana. Si el estado es diferente a lo largo de la ventana, se hace una media de los valores, de esta forma se tiene en cuenta la cantidad de tiempo que ha estado en cada estado.

En definitiva, en la Figura 5.3 se puede apreciar la cantidad de datos que hay que procesar para clasificar la actividad que ocurre cada medio segundo. Dado que el objetivo del proyecto es procesar en tiempo real los datos de los sensores y obtener las actividades que se realizan, es necesario adoptar una infraestructura que permita esto.

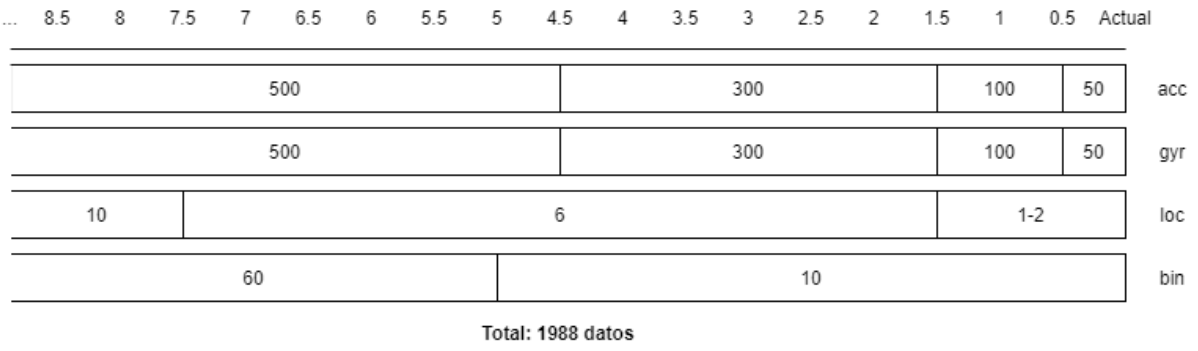


Figura 5.3: Cantidad de datos a procesar cada 0.5s

6 Desarrollo del sistema

En este capítulo se recogen todos los aspectos de la creación y despliegue del sistema de procesamiento de datos en tiempo real. Así, se divide en 5 subsecciones que recogen los conceptos fundamentales del sistema. En primer lugar, se analiza la infraestructura con el objetivo de recoger los datos del entorno. En segundo lugar, se muestra la base de datos en la cual se almacena toda la información del sistema. En tercer lugar, se analizan los conceptos de machine learning que se incluyen en el sistema para reconocer las actividades con un alto grado de certeza. Posteriormente, se muestra la infraestructura del procesamiento de los datos y finalmente la generación de un conjunto de datos de entrenamiento que identifican las actividades que el sistema puede detectar.

Así, este capítulo expone la parte importante del trabajo, ya que se describe la infraestructura, tecnologías, scripts y funcionalidad del sistema.

6.1 Obtención de datos

El sistema de detección de actividades incorporado en la vivienda hace uso de una gran cantidad de datos. Todos estos datos no proceden de un solo dispositivo, sino que son varios los dispositivos que se encuentran en la *smart home* y que dotan de ”inteligencia” a esta. En esta sección se analiza en primer lugar el protocolo de comunicación entre los dispositivos y posteriormente los dispositivos que se han incorporado, así como su configuración, instalación e inclusión en el sistema.

6.1.1 MQTT

Como se comentó en la Sección 4.5.5, MQTT (Message Queue Telemetry Transport) consiste en un protocolo de comunicación Open Source encargado de la publicación/suscripción de mensajes. La gran relevancia que ha adoptado es gracias a su integración en diversas tecnologías como lenguajes de programación (incluyendo librerías), sensores y dispositivos controladores que interactúan con estos sensores. Por esta razón, los dispositivos que se han incorporado en el sistema IoT actual respecto al sistema implementado en el Trabajo Fin de Grado permiten realizan la comunicación con los dispositivos de tipo controlador haciendo uso de este protocolo de publicación/suscripción.

MQTT fue diseñado hacia el año 1999, pero se convirtió en un protocolo de comunicación

abierto en el año 2014. Sin embargo, la importancia del proyecto ha venido acompañada del Internet de las Cosas, ya que facilita la conexión de dispositivos como Raspberry Pi y Arduino con una gran cantidad de sensores.

En cuanto al funcionamiento del protocolo, las comunicaciones se realizan sobre el puerto 1883 en caso de comunicaciones sin cifrar y sobre el puerto 8883 para comunicaciones cifradas mediante SSL[1] o TLS[2]. En la mayoría de los casos las comunicaciones son a través del puerto 1883 sin cifrar, ya que no se envía información sensible a través de este protocolo y los recursos de los dispositivos que lo usan son muy limitados, por lo que no merece la pena cifrar el contenido.

MQTT está formado por diferentes elementos que se describen a continuación:

- *Broker*: Se trata del servidor que almacena la información que recibe por parte de un cliente. Es el encargado de recibir la información y almacenarla en su *topic* correspondiente. En caso de tener suscripciones a dicho *topic*, es el encargado de enviar la información a los clientes suscritos a este.
- *Topic*: *Tema* en el cual se encuadra un mensaje enviado por MQTT.
- *Cliente publicador*: Proceso, aplicación o sensor encargado de proporcionar información al broker. Esta información se envía a un *topic* concreto del broker.
- *Cliente suscriptor*: Proceso, aplicación o sensor que está suscrito a los cambios de uno o más *topics* por lo que cada vez que el *broker* incluya información en este o estos, esta información es recibida por el cliente suscrito a dicho o dichos *topics*.

En la Figura 6.1 se muestra un ejemplo de comunicación a través de MQTT en la que se puede apreciar el flujo de la información en el sistema y los diferentes componentes que forman el mismo. En primer lugar, se observa un *broker* MQTT con dos *topics*, cada uno de ellos almacena la localización de una persona. Por otra parte, se tienen dos personas que incluyen sensores de localización. Cada sensor envía los datos al *topic* de la persona correspondiente. Por otro lado, varios servicios consumidores observan los cambios que se producen en el *broker*. El primero de ellos es un servicio web que muestra la localización de ambas personas. El segundo de ellos se trata de un script que almacena en una base de datos la localización solamente de la persona 1. Como se puede observar, cada suscriptor recibe la información a la que está suscrito.

6.1.2 Sensores ubicados en la *smart home*

El uso de sensores no intrusivos está cada vez más normalizado en las viviendas, permitiendo tener un control de ciertos aspectos de estas y además permitiendo realizar pequeñas tareas de automatización. Estos sensores, como no tienen que ser portados por el usuario ya que están situados en la vivienda, hacen que la inclusión de una gran cantidad de estos no sea apreciable

[1] Secure Sockets Layer
[2] Transport Layer Security

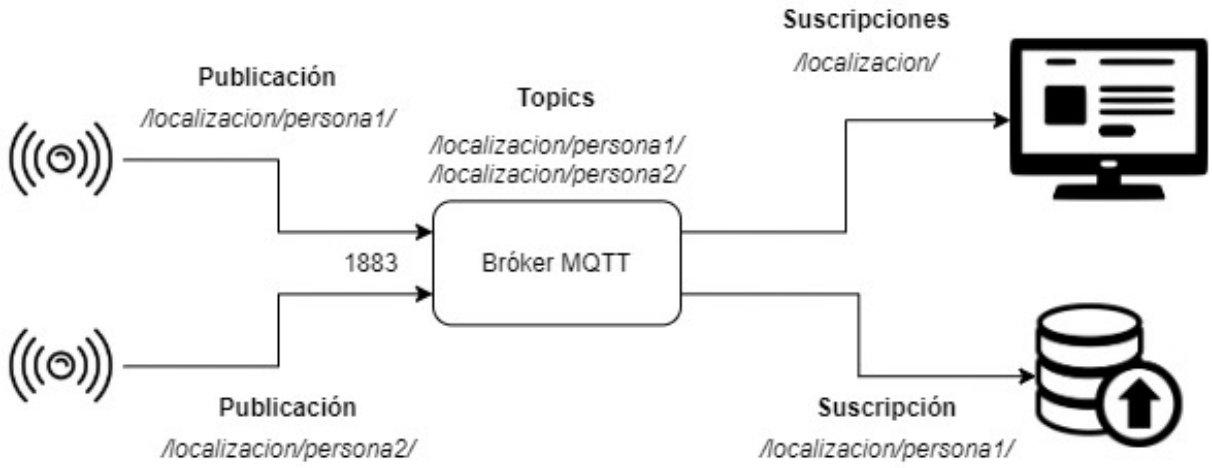


Figura 6.1: Ejemplo comunicación MQTT

para el usuario final. Así, conforme aumenta la cantidad de sensores usados en la vivienda, se recogen más datos sobre el estado de esta y por lo tanto, la detección de actividades se realiza de una forma más precisa.

En la vivienda se ha incorporado una gran cantidad de este tipo de sensores, ya que su precio no es muy elevado y se puede realizar una monitorización constante del estado de la vivienda. Además, en el anterior Trabajo Fin de Grado, se implementó un sistema de reconocimiento de actividades del usuario en la vivienda haciendo solamente uso de estos sensores.

6.1.2.1 Dispositivos utilizados

Los sensores incorporados se comunican con el elemento controlador haciendo uso del protocolo ZWave. Este consiste en un protocolo desarrollado con el fin de permitir comunicaciones inalámbricas entre dispositivos. Este estándar fue diseñado con el objetivo de conectar diferentes dispositivos que facilitan la automatización del hogar, es decir, fue creado especialmente para el uso en domótica, por lo que se adapta a las necesidades de este ámbito. Este estándar realiza la conexión de los dispositivos formando una malla entre ellos y el *hub* o controlador. De esta forma, con el propósito de realizar la conexión entre un dispositivo y el *hub*, se tiene que transmitir esta información a otros dispositivos de la red para intentar llegar a este. Sin embargo, el número máximo de saltos en este caso es de 4. Esto hace que los dispositivos tengan que estar a una distancia no muy lejana con respecto al *hub*. Además, tiene un consumo de energía muy bajo, por lo que lo hace muy interesante para realizar tareas de monitorización continua, como es el caso.

Como se ha comentado, ZWave es usado principalmente en domótica, por lo que muchos fabricantes han creado multitud de sensores que implementan el protocolo y permiten formar redes de sensores que se interconectan entre sí. De esta forma, los sensores que se han usado en este caso para realizar la comunicación y recogida de datos son en su mayor parte sensores de

la marca Fibaro, que se muestran a continuación.

6.1.2.1.1 Sensores

Fibaro Motion Sensor Se trata de un dispositivo multisensor, que incorpora un sensor de presencia, de temperatura, de luminosidad y acelerómetro. En cuanto al sensor de presencia, este dispositivo usa un sensor *IR pasivo*. Este sensor de infrarrojos pasivo se encarga de diferenciar el calor emitido por fuentes de energía como el calor humano por el emitido por objetos estáticos e inertes. Así, de esta forma se puede incorporar el sensor en cualquier espacio interior, aunque haya objetos en el rango de detección, ya que estos no emiten el calor que emite el humano, por lo que no detectaría la presencia de estos objetos.

Este sensor (Figura 6.2) se usa fundamentalmente con la finalidad de detectar la presencia de un usuario en una habitación. Para colocar correctamente el sensor, hay que realizar un estudio de la habitación y localizar el punto en el que se detecta al usuario la mayor parte del tiempo que este está en la habitación. Para ello, se realizan varias pruebas en las que se comprueba la actividad del usuario en la habitación y el funcionamiento del sensor.

Figura 6.2: Fibaro Motion Sensor

Fibaro Door/Window Sensor 2 Fibaro Door/Window Sensor 2 es un dispositivo multisensor (Figura 6.3) que incorpora un sensor de temperatura, de contacto y acelerómetro. El sensor de contacto constituye de la principal funcionalidad del dispositivo ya que permite conocer el estado de un objeto o elemento de la vivienda que se puede abrir/cerrar, como puede ser una puerta, un cajón o una ventana. Para detectar el estado de apertura/cierre, el sensor usa dos partes; la parte principal y una parte secundaria más pequeña que forman un imán. De esta forma, cuando están juntos el estado del sensor es ”cerrado” y cuando están separados es ”abierto”.

Arun PM3 Es un dispositivo (Figura 6.4) diferente a los anteriormente mencionados, ya que la compañía que lo ha creado no es de Fibaro; así, no se trata de un multisensor, sino que consiste en un sensor de presión binario, con solo dos estados posibles, encargado de detectar presión sobre su superficie.

Figura 6.3: Fibaro Door/Window Sensor 2

Figura 6.4: Sensor de presión

Este sensor de presión se ha usado con el fin de comprobar cuándo una persona se sienta sobre una superficie. Dado que la conexión de este sensor se realiza a través de cables, para comunicarse con el controlador habría que situar un controlador cerca de cada sensor de presión. Para evitar esta conexión directa, se hace uso de un dispositivo intermediario. Este dispositivo es del sensor de contacto Door/Window Sensor (Figura 6.5).

Figura 6.5: Door/Window Sensor

De esta forma, el sensor de contacto recibe los cambios de estado del sensor de presión a través del cableado que se muestra en la Figura 6.6, y desde este sensor de contacto se envía la señal

al controlador haciendo uso del protocolo ZWave.

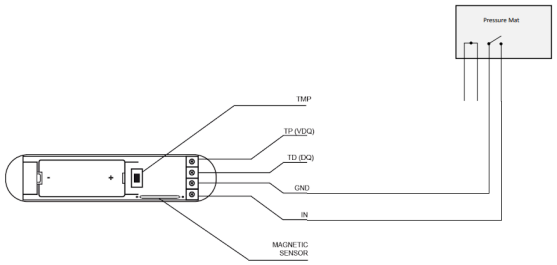


Figura 6.6: Conexión Arun PM3 - Door/Window Sensor

6.1.2.1.2 Controlador Como se ha comentado anteriormente, los sensores que se han usado en la *smart home* envían los datos haciendo uso del protocolo ZWave. El dipositivo que recibe los datos se trata de Raspberry Pi 3 B+ (Figura 6.7). Es un producto que forma parte de la familia de dispositivos Raspberry. Estos dispositivos son especialmente conocidos gracias a su amplia funcionalidad (tiene las mismas funciones que un *PC*), a muy bajo precio (sobre 35 euros). Además, gracias a los pines *GPIO* de los que dispone, se permite la conexión de sensores o elementos electrónicos, lo cual hacen que este producto pueda usarse para la programación de sistemas de igual modo a como se hace con Arduino.

Figura 6.7: Raspberry Pi 3 B+

El dispositivo permite conexiones inalámbricas Bluetooth y Wi-Fi, por lo que para poder recibir datos de sensores que usan el protocolo ZWave es necesario un módulo complementario denominado Razberry (Figura 6.8). Este módulo se conecta a Raspberry Pi haciendo uso de los pines GPIO de la misma. Además, el módulo Razberry incluye un software accesible desde la Raspberry que permite la adición de nuevos sensores en la red que forma, la eliminación de estos, la programación de pequeñas tareas de automatización, etc...

Figura 6.8: Módulo Razberry

6.1.2.2 Adquisición de datos

Sin embargo, para realizar el algoritmo de detección de actividades, es necesario recoger los datos de los sensores y guardarlos de forma externa al software del módulo Razberry. Esto se hace mediante una API denominada *vDev API* para obtener toda la información proporcionada por los sensores instalados. Esta API fue desarrollada y es mantenida por *ZWave.me*, por lo que consiste en una API oficial, garantizándose el soporte de la misma. La estructura de la API se define en (ZWave.me, 2019). En esta API se pueden observar consultas referentes al historial de los dispositivos, notificaciones, ubicaciones, perfiles, etc.

Por lo tanto, para recoger los datos proporcionados por los sensores, es necesaria la programación de un *script* que se ejecute cada cierto tiempo (en este caso, cada segundo). Así, este script recopila los datos y los incluye en la base de datos propia. Así, para obtener los nuevos datos de los sensores, se hace uso del *endpoint* de la *API* que se muestra en el Código 6.1, especificando como parámetro en la URL el instante de tiempo a partir del cual se quieren obtener nuevas notificaciones por parte de los sensores.

Código 6.1: Consulta de notificaciones a la API

```
var options2 = {
   url: 'http://' + direccion + ':8083/ZAutomation/api/v1/notifications?since=' +
       ↪ timestamp_reciente,
   method: 'GET',
   dataType: 'json',
   headers: {
   'Content-Type': 'application/json',
   Cookie: cookie,
};
```

Como resultado a esa consulta HTTP, se obtiene una respuesta (en el caso de haber notificaciones más recientes) como se puede observar en el Código 6.2. Una vez obtenida la respuesta, es

necesario procesarla con el fin de obtener los datos correspondientes a los sensores y guardarlos en la base de datos.

Código 6.2: Respuesta Notificaciones

```
{
   "data": {
      "updateTime": 1557157779,
      "notifications": [
            {
               "id": 1557075066034,
               "timestamp": "2019-05-05T16:51:06.034Z",
               "level": "device-info",
               "message": {
                  "dev": "Baño - Presencia",
                  "l": "22.8 °C",
                  "location": ""
               },
               "type": "device-temperature",
               "source": "ZWayVDev_zway_2-0-49-1",
               "redeemed": false
            },
            {
               "id": 1557075066161,
               "timestamp": "2019-05-05T16:51:06.161Z",
               "level": "device-info",
               "message": {
                  "dev": "Baño - Ducha - Luminosidad",
                  "l": "0 Lux",
                  "location": ""
               },
               "type": "device-luminiscence",
               "source": "ZWayVDev_zway_2-0-49-3",
               "redeemed": false
            }
      ]
}
```

Este proceso se produce de manera indefinida, repitiéndose cada segundo para monitorizar de la forma más precisa posible el estado de la vivienda y realizar el reconocimiento de actividades de forma correcta, ya que hay actividades que tienen una duración muy corta. Para hacer esto, se hace uso de la función de Javascript *setInterval(función,frecuencia en ms)*.

Por otra parte, una vez se encuentran los datos en la base de datos del sistema, como el algoritmo de reconocimiento de actividades trata ventanas de 500ms, para agilizar el proceso de obtención de datos y facilitar el procesamiento, se optó por crear un script que almacena el estado de los sensores de la *smart home* y guardan un registro en la base de datos cada 500ms. De esta forma, en cada ventana de tiempo de medio segundo se tiene un registro con el estado de los sensores permitiendo saber cuándo cambia el estado de cada sensor de una forma exacta.

Por lo tanto, la infraestructura que corresponde a la recogida de los datos de los sensores es la que se muestra en la Figura 6.9.

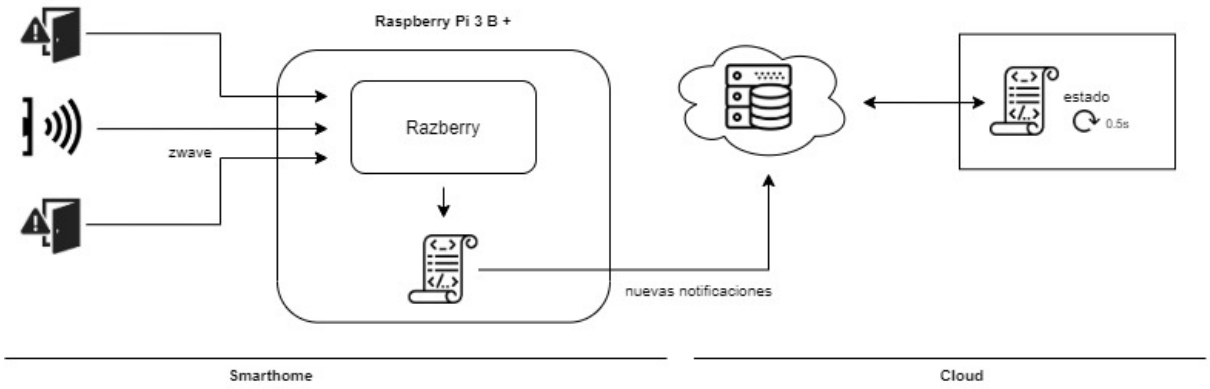


Figura 6.9: Obtención datos *smart home*

6.1.3 Sensores localizados en un *smartwatch*

En esta sección se analiza el *smartwatch* que se ha seleccionado con el fin de obtener los datos de aceleración y giroscopio necesarios para realizar la detección de actividades.

6.1.3.1 Polar M600

Dado que en el proyecto se necesitan datos de aceleración y giroscopio, de forma nativa los fabricantes no proporcionan APIs para acceder a ellos haciendo uso de alguna aplicación o servicio externo al smartphone. Por ello es necesario acceder a estos datos creando software propio que se ejecute en la pulsera o reloj con el propósito de acceder a los datos de los sensores integrados en este. Por lo tanto, para poder incorporar software al *smartwatch* por lo tanto, este debe permitir la instalación de software de terceros. Esto es posible gracias a Android Wear OS, un sistema operativo creado por Google para los *smartwatch*es que permite la instalación de aplicaciones tal y como se hace en un smartphone. De esta forma, se ”simula” el comportamiento de un smartphone en un reloj. Este sistema operativo permitirá instalar aplicaciones propias instalando el fichero *.apk* correspondiente.

Concretamente, en este proyecto se usa un *smartwatch* de la marca Polar, conocida por crear el primer pulsómetro inalámbrico. Actualmente, Polar se encarga de fabricar equipos deportivos como pulseras deportivas, *smartwatch*es y GPS. El dispositivo se trata de Polar M600 (Figura 6.10). Se ha escogido este producto por su alta popularidad en relación a la calidad/precio y a demás a la incorporación de sensores de aceleración y giroscopio y sistema operativo Android Wear OS.

La especificaciones técnicas se muestran en la Tabla 6.1.

Figura 6.10: Polar M600

Característica	Valor
Sistema Operativo	Wear OS by Google
Procesador	MediaTek MT2601, doble núcleo 1,2 GHz - ARM Cortex-A7
Memoria	4 GB de almacenamiento interno + 512 MB de RAM
Batería	500 mAh
Duración de la batería	2 días
Peso	63 g
Dimensiones	45 x 36 x 13
Sensores	Acelerómetro, Giroscopio, GPS, Luz ambiental, Motor de vibración.

Tabla 6.1: Especificaciones técnicas Polar M600

6.1.3.2 Conexión con controlador y almacenamiento de datos

Como se ha comentado anteriormente, para hacer uso de los datos de los sensores de aceleración y giroscopio, es necesario implementar una aplicación propia haciendo uso de Android Studio. El objetivo de esta aplicación consiste en acceder a los datos de aceleración y giroscopio a medida que se generan en el *smartwatch* (cada 10ms) y enviarlos al dispositivo Raspberry que actúa como controlador, haciendo uso de MQTT.

Dado que MQTT se trata de un protocolo de publicación/suscripción de datos, es necesario establecer los diferentes "topics" a los cuales se envían los datos generados por los sensores de aceleración y giroscopio. Los topics siguen la estructura que se muestra en la Tabla 6.2.

Persona	Sensor	Topic
Persona 1	Acelerómetro	/case/inertial/wearA/acc
Persona 1	Giroscopio	/case/inertial/wearA/gyr
Persona 2	Acelerómetro	/case/inertial/wearB/acc
Persona 2	Giroscopio	/case/inertial/wearB/gyr
Persona 3	Acelerómetro	/case/inertial/wearC/acc
Persona 3	Giroscopio	/case/inertial/wearC/gyr

Tabla 6.2: Topics MQTT

Aplicación smartwatch El funcionamiento de la aplicación es bastante sencillo. En primer lugar, se realiza la conexión al controlador (Raspberry Pi) especificando las propiedades del broker MQTT situado en este de la forma que se muestra en el Código 6.3. Para poder acceder al broker MQTT, es necesario que la Raspberry se encuentre en la misma red que el *smartwatch*.

Código 6.3: Conexión con Raspberry

```
static public String WiFI_SSID="MiFibra-40BA-2.4G";
static public String WiFI_PASS="XXXXXX";
static public String URL_BROKER="tcp://192.168.1.127:1883";

static public String TOPIC_WEAR="/case/inertial/wearA";
public static final String TAG = "WearableActivity";


WifiConfiguration wifiConfig = new WifiConfiguration();

wifiConfig.SSID = String.format("\"%s\"", WiFI_SSID);
wifiConfig.preSharedKey = String.format("\"%s\"", WiFI_PASS);
WifiManager wifiManager = (WifiManager)getApplicationContext().getSystemService(↩
    ↪ WIFI_SERVICE);

Log.d("wear","checkWifi!"+getLocalIpAddress());
```

Una vez realizada la conexión con el broker de MQTT situado en la Raspberry, se accede a los valores de los sensores de aceleración y giroscopio como se muestra en el Código 6.4.

Código 6.4: Acceso a sensores

```
public void onSensorChanged(SensorEvent se) {
   t0=System.currentTimeMillis();
   float x = se.values[0];
   float y = se.values[1];
   float z = se.values[2];
   if(se.sensor.getType()==TYPE_GYROSCOPE)
      senderA.add("gyr",new AccData(t0,x,y,z));
   else if(se.sensor.getType()==TYPE_ACCELEROMETER)
      senderR.add("acc", new AccData(t0, x, y, z));
}
```

Para realizar el envío de los datos al broker de MQTT se realiza como sigue en el Código 6.5.

Código 6.5: Envío de datos por MQTT

```
private boolean publish(String topic, String payload){
      try {
         BlockingConnection connection=mqtt.blockingConnection();
         connection.connect();
         connection.publish(topic, payload.getBytes(), QoS.AT_LEAST_ONCE,false);
      } catch (Exception e) {
         Log.d("MQTT","Error sending to topic "+topic);
         e.printStackTrace();
```

```
9           return false;
10      }
11   return true;
12   }
```

A continuación, en la Figura 6.11 se puede apreciar la interfaz gráfica de la aplicación que envía los datos de aceleración y giroscopio al broker especificado.

Figura 6.11: Aplicación Polar M600

Recepción de datos y almacenamiento Una vez enviados los datos a través del protocolo MQTT desde el Polar M600 hacia el controlador Raspberry situado en la *smart home*, se deben recibir los datos y almacenar en la base de datos del sistema. Para hacer esto, se hace uso de un *script* Python usando la librería *paho.mqtt.client*.

En primer lugar, el *script* realiza la conexión con el broker MQTT situado en el puerto 1883. Una vez realizada la conexión, se realiza la suscripción a los topics correspondientes al usuario actual con la finalidad de recibir los datos correspondientes de aceleración y giroscopio (Código 6.6). En la función *on_connect* se realiza la suscripción a los topics.

Código 6.6: Conexión con broker MQTT y suscripción a topics

```
import paho.mqtt.client as mqtt
client = mqtt.Client()

#Obtencion de datos por parametro
smartwatch = str(sys.argv[1])
persona = int(sys.argv[2])

def on_connect(client, userdata, flags, rc):
    client.subscribe("/case/inertial/"+smartwatch+"/gyr")
    client.subscribe("/case/inertial/"+smartwatch+"/acc")
    print("Conexión correcta")

client.on_connect = on_connect
client.on_message = on_message
client.connect("localhost", 1883, 60)
```

```
client.loop_forever()
```

El *script* se ejecuta continuamente, al establecer el bucle infinito *client.loop_forever()*. Así, cuando se recibe un nuevo dato se ejecuta la función que se muestra en el Código 6.7. En ella, se obtienen los datos en formato de cadena de texto, se parsean en formato de diccionario haciendo uso de la librería *ast* y se almacenan en la base de datos.

Código 6.7: Recepción de datos a través de topics

```
def on_message(client,userdata,msg):
    lista = ast.literal_eval(msg.topic)
    array = []
    for dato in lista:
        timestamp = int(dato["t"])
        insertar = {"topic" :msg.topic,"tipo" : msg.split("/")[4], "persona" : ↩
            ↪ persona, "x" : dato["x"] , "y" : dato["y"] , "z" : dato["z"], "↩
            ↪ timestamp" :timestamp }
        array.append(insertar)
        sensor_data.insert_many(array)
```

De esta forma, se ejecuta tantas veces el *script* como usuarios estén usando los diferentes *smartwatch* del sistema. Para la ejecución del *script*, se inserta *python getAccGyr.py nombre smartwatch persona*. En la Figura 6.12 se muestra la forma en la que se transmiten los datos por MQTT desde los *smartwatch*es hasta la base de datos del sistema.

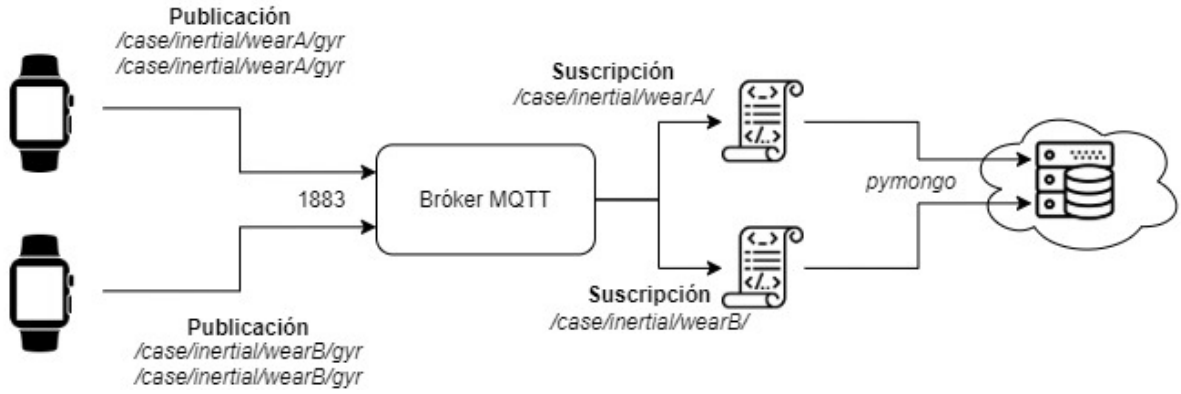


Figura 6.12: Flujo de datos procedentes de smartwatch

6.1.4 Sensores de localización

En el reconocimiento de actividades, los sensores situados en la *smart home* pueden proporcionar información de la localización del usuario en cierto momento dado, al tener situados sensores de detección de presencia. Sin embargo, estos sensores ofrecen valores de presencia/no presencia, sin especificar la distancia entre el usuario y el sensor. Por lo tanto, para solucionar este problema se hace uso de RTLS (Real time location systems), es decir, sistemas que monitorizan

la posición del usuario en cada momento cuando este está presente en la vivienda.

Así, en el mercado actualmente hay muchos proveedores de sistemas RTLS, sin embargo, las tecnologías que usa cada sistema RTLS pueden variar dependiendo de la ubicación en la que se van a establecer los sistemas, el coste, el tamaño, etc. En este caso, se hace uso de un sistema RTLS con UWB (Ultra-Width-Band).

Concretamente, se hace uso del kit MDEK1001 Development Kit de la compañía Decawave, que forma parte de Qorvo. Esta compañía se dedica al diseño, fabricación y comercialización de sistemas de radiofrecuencia inalámbricos y de banda ancha.

6.1.4.1 Decawave MDEK1001 Development Kit

Este kit proporciona al usuario la capacidad de generar sistemas RTLS de forma sencilla, proporcionando una solución escalable y fácil de integrar con los propios servicios o aplicaciones del usuario. Además, permite la visualización del sistema a través de una aplicación de móvil.

La red RTLS está formada por dispositivos DWM1001 (Figura 6.13), en la cual cada dispositivo actúa de forma diferente dependiendo de su función. El módulo está formado por un sensor de movimiento y diversas antenas integradas. Además, proporciona encriptación y hace de *tag* en una red RTLS con un margen de error muy bajo. Además, incorpora Bluetooth para realizar la comunicación con otros dispositivos para su configuración o visualización.

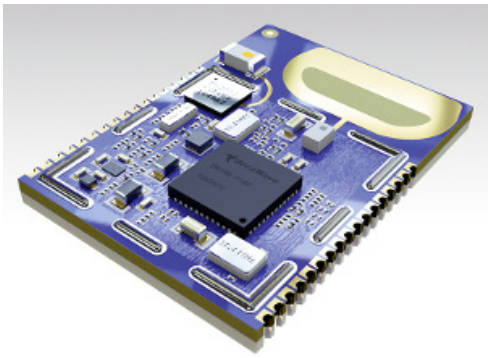


Figura 6.13: Módulo DWM1001

Con el fin de formar una red RTLS, se incorporan diferentes módulos DWM1001 en una distancia no superior a 30 metros del módulo que actúa como gateway. Los diferentes modos que pueden adoptar estos módulos son los siguientes:

- *Gateway*: Elemento que recibe los datos procedentes de los demás módulos para realizar el tratamiento de la información de estos y mostrar la ubicación del resto de módulos.
- *Anchor*: Módulos que se ubican de forma estática en la localización en la que se instala la red RTLS. Estos módulos son los encargados de establecer las ”referencias” en la red y

delimitar su rango y tamaño. Estos módulos proporcionan la ubicación de los módulos *tag* que se mueven dentro del área que forman los módulos *anchor*.

- *Tag*: Módulos que se mueven dentro de la red RTLS. El objetivo de la red por tanto, consiste en determinar la posición relativa de los módulos *tag* en referencia a los módulos *anchor*.

Un ejemplo de red se puede visualizar en la Figura 6.14. En ella se pueden observar los diferentes módulos DWM1001 ubicados, cada uno con su rol. En este caso, se pueden observar dos módulos de tipo *tag*, ubicados dentro de la red definida por los 8 módulos tipo *anchor*. La comunicación entre los módulos de tipo *tag* y *anchor* se realiza a través del protocolo de comunicación UWB. Por otra parte, la información de la ubicación de los diferentes módulos *anchor* y *tag* se comunica al módulo o módulos *gateway*. Este módulo *gateway* puede estar conectado a un dispositivo como Raspberry Pi o similar para enviar y almacenar los datos en este.

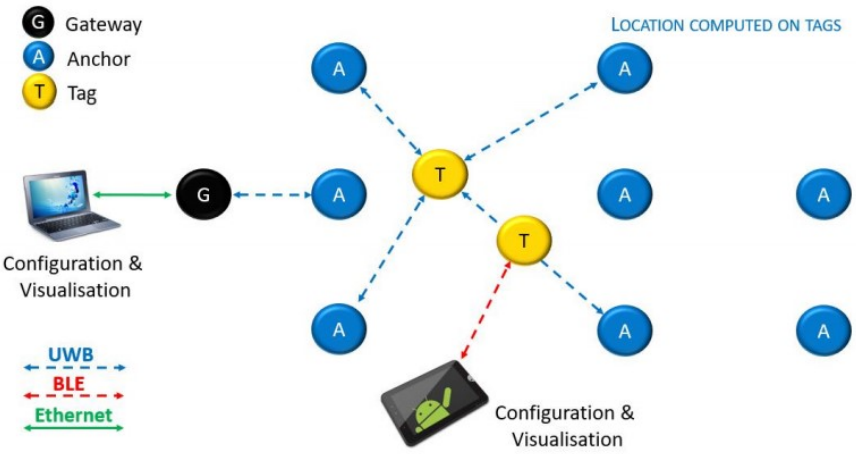


Figura 6.14: Red RTLS - DWM1001

El kit que se usa para formar la red RTLS está formado por 12 módulos DWM1001-DEV que pueden formar una nueva red o integrarse en otra red existente. Esta placa integra el módulo DWM1001 en una placa con el objetivo de poder alimentar este módulo para ubicarlo en la vivienda, cargar el firmware o conectar con un dispositivo Raspberry. En la Figura 6.15 se muestra el módulo DWM1001 junto con todos los componentes que lo forman.

Haciendo uso del kit, se crea una red DRTLS (Two-Way Ranging Real Time Location System Network). Para ello, es necesario tanto el kit hardware como componentes software. Los componentes software son los siguientes:

- *Decawave Positioning and Networking Stack*: Software del módulo DWM1001. Este software proporciona las funciones para que el módulo pueda comportarse como cualquiera de los tres tipos anteriormente mencionados.

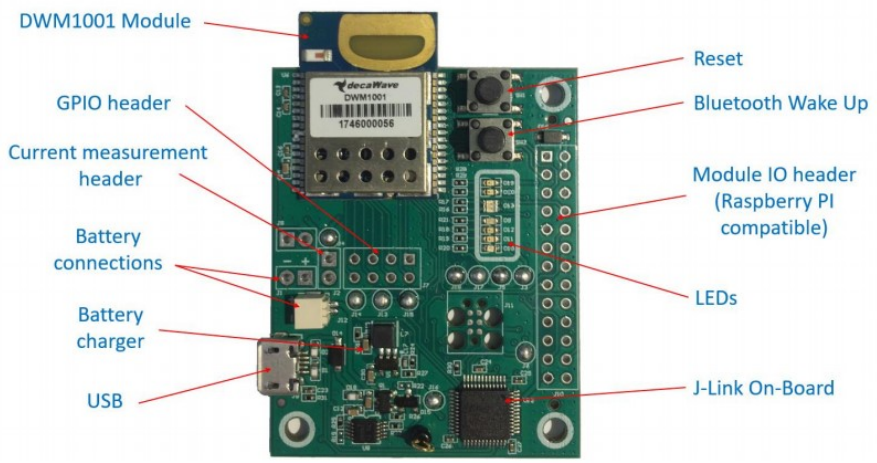


Figura 6.15: Módulo DWM1001-DEV

- *Decawave DRTLS Manager*: Aplicación móvil en la cual se configura la red mediante Bluetooth,
- *Decawave DRTLS Gateway Application*: Software que se instala en el dispositivo en el cual se conecta el módulo "gateway". En este caso, este dispositivo se trata de una Raspberry Pi Model 3. Por lo tanto, el software se debe instalar en la Raspberry para poder realizar la comunicación con el módulo gateway y recibir los datos de este.

6.1.4.2 Configuración de la red DRTLS

Instalación de firmware en DWM1001 En primer lugar, a través de la herramienta SEGGER J-Flash, se carga el firmware correspondiente en el módulo DMW1001 como se muestra en la Figura 6.16. La conexión entre el módulo y el PC se realiza a través del puerto USB.

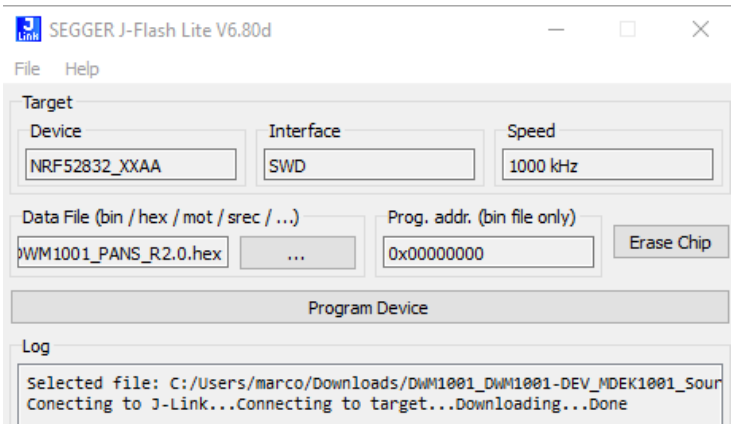


Figura 6.16: Instalación firmware en DWM1001 mediante J-Flash

Creación de la red RTLS A continuación, se instala la aplicación DRTLS Android Manager y se realiza la configuración de la red (Figura 6.17).

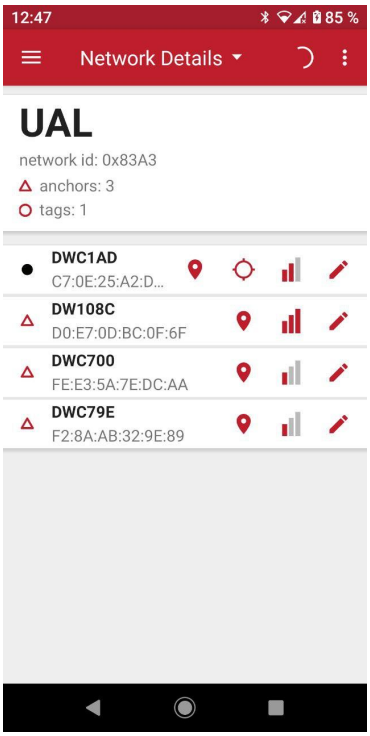


Figura 6.17: Configuración de red RTLS

Puesta a punto de Raspberry Pi 3 Para poder tener el software de Decawave en un dispositivo Raspberry, es necesario cargar la imagen que proporciona el *kit*. Para realizar la carga de la imagen, se usa el programa "Balena Etcher" seleccionando la imagen correspondiente y la tarjeta SD en la cual se va a grabar.

Configuración del gateway Por otra parte, se realiza la conexión de un módulo DWM1001-DEV con los pines GPIO de la Raspberry. La conexión se puede observar en la Figura 6.18.

Una vez conectado con la Raspberry, se inicia el sistema operativo y se accede al dispositivo Raspberry desde el PC local a través de SSH. Para establecer la red RTLS de la cual el dispositivo va a actuar de gateway, es necesario modificar el archivo */etc/dwm1001/dwm1001.conf*. En el archivo, se establece el *id* de la red denominado *panid* (Figura 6.19), que se puede observar en la aplicación Android instalada previamente.

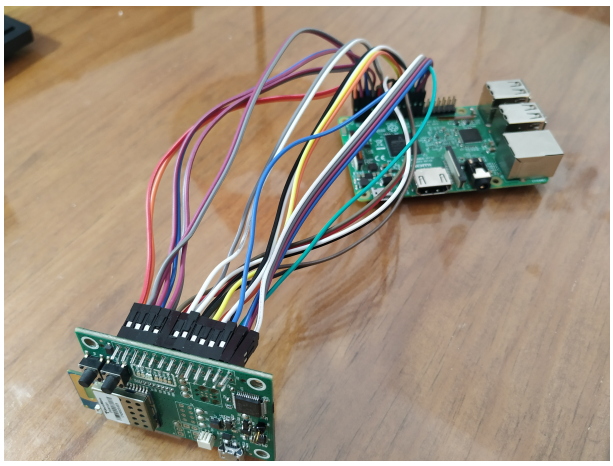


Figura 6.18: Conexión Gateway con Raspberry Pi 3

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ cat /etc/dwm1001/dwm1001.config
########################################################################
# DWM Kernel Module settings
#

# UWB network PANID
# Allowed values: 16-bit number in decimal, hexadecimal or octal format
panid = 0x83A3

# UWB encryption
# Allowed values: true/false
enc_en = false

# UWB encryption key
# Allowed values: 128-bit number in hexadecimal format
enc_key = "11111111222222223333333344444444"

# UWB firmware update
# Allowed values: true/false
uwb_fwup = true

# UWB mode
# Allowed values: 0=Off 1=Passive 2=Active
uwb_mode = 2

# daemon settings
proxy_server_host = "localhost"
proxy_server_port = 1885
mqtt_user = "dwmuser"
mqtt_password = "dwmpass"
mqtt_topic_prefix = "dwm"
pi@raspberrypi:~ $
```

Figura 6.19: Configuración Gateway

6.1.4.3 Red DRTLS del sistema

En un principio, la idea era montar la red DRTLS en la *smart home* de la UAL. Lamentablemente, la situación de excepcionalidad debido al COVID-19 ha hecho que esto no sea posible. Sin embargo, sí se disponían de 5 componentes DWM1001-DEV junto con un dispositivo Raspberry, por lo que se ha podido crear una red local de prueba, posteriormente exportable a la *smart home*. La red se puede visualizar en la Figura 6.20. En esta red, se tiene un dispositivo configurado como gateway, tres de ellos como *anchor* y uno de ellos como *tag*.

Figura 6.20: Red DRTLS local

6.1.4.4 Envío de datos por MQTT y almacenamiento

La imagen que se ha cargado a la Raspberry y que contiene el software *Decawave DRTLS Gateway Application* incluye un broker MQTT cuyo objetivo es recibir los datos que contienen la ubicación de los *tag*s del sistema. En este caso, como solamente se tiene un "*tag*", se tiene un topic denominado *dwm/node/c1ad/uplink/location*. En este, se publican los sucesivos estados de la localización del módulo que actúa como *tag*. En el sistema implementado, al ser un sistema multiusuario, se tendrán varios topics, cada uno de ellos correspondiente a un usuario de la vivienda.

Los datos que se reciben de forma predeterminada se pueden visualizar a través del software MQTTfx, que se trata de un cliente de MQTT que permite recibir datos por MQTT. En la Figura 6.21 se muestra una secuencia de datos recibidos procedentes del sensor de localización.

En consecuencia, para obtener los datos de los *topics*, se crea un *script* en Python (Código 6.8) que recoge los datos y los almacena en la base de datos del sistema. El script es muy parecido al Código 6.6 en el caso de recepción de datos de aceleración y giroscopio. En este caso, se tiene solamente un script que recoge los datos de todos los *tags* de los usuarios de la vivienda. Dado que cada persona genera un dato de localización por segundo, la frecuencia de los datos es menor a la de aceleración y giroscopio y se pueden recoger y procesar en un solo script todos los datos de los usuarios de la vivienda. En este caso, la función *on_message* asocia el *id* del *tag* de cada usuario con el *id* del usuario y almacena en el registro en la base de datos. Concretamente, en

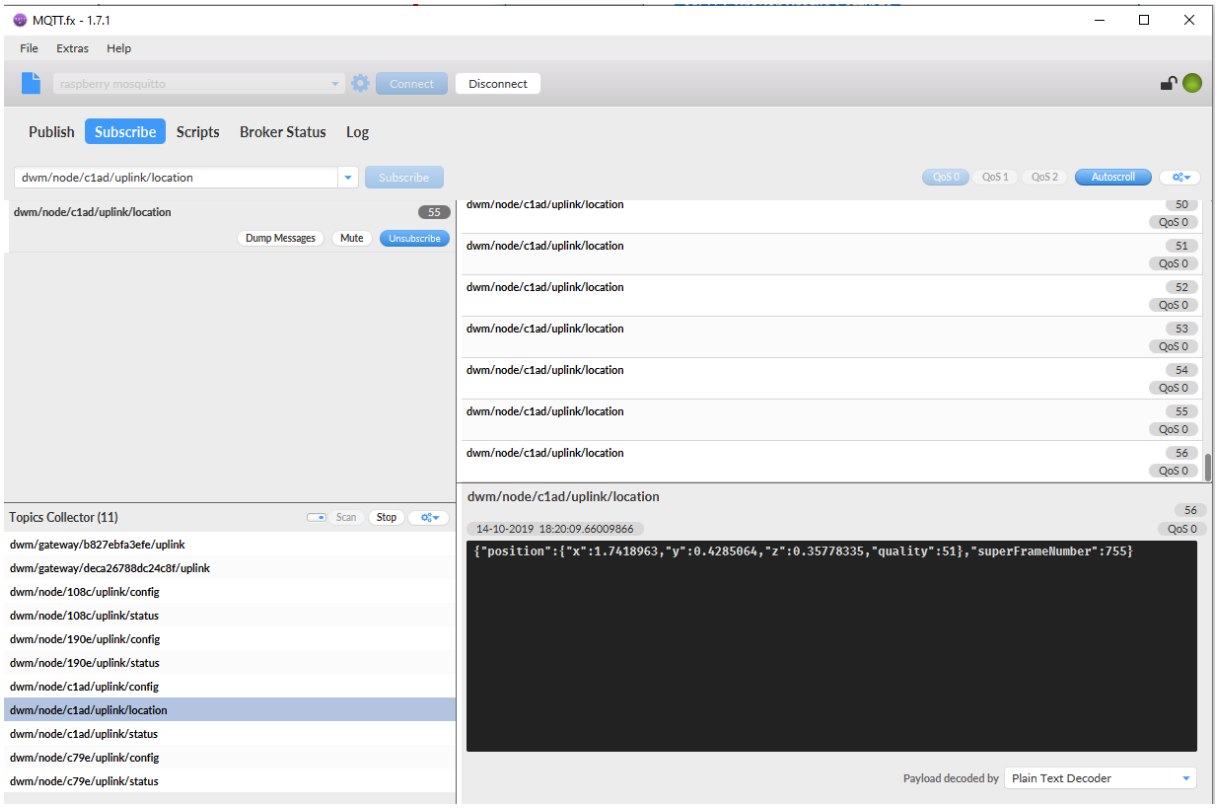


Figura 6.21: Datos que envía el sensor de localización

la colección *sensor_data*, al igual que el resto de datos generados por los sensores.

Código 6.8: Conexión con broker MQTT y suscripción a topics

```
import paho.mqtt.client as mqtt
client = mqtt.Client()

def on_connect(client, userdata, flags, rc):
    #localización del usuario 1
    client.subscribe("dwm/node/c1ad/uplink/location")
    #localización del usuario 2
    client.subscribe("dwm/node/e5ju/uplink/location")
    print("Conexión correcta")

client.on_connect = on_connect
client.on_message = on_message
client.connect("localhost", 1883, 60)
client.loop_forever()
```

En la Figura 6.22 se muestra la transmisión de los datos procedentes de los *tag*s de localización hacia el dispositivo Raspberry y su posterior inserción en la base de datos del sistema.

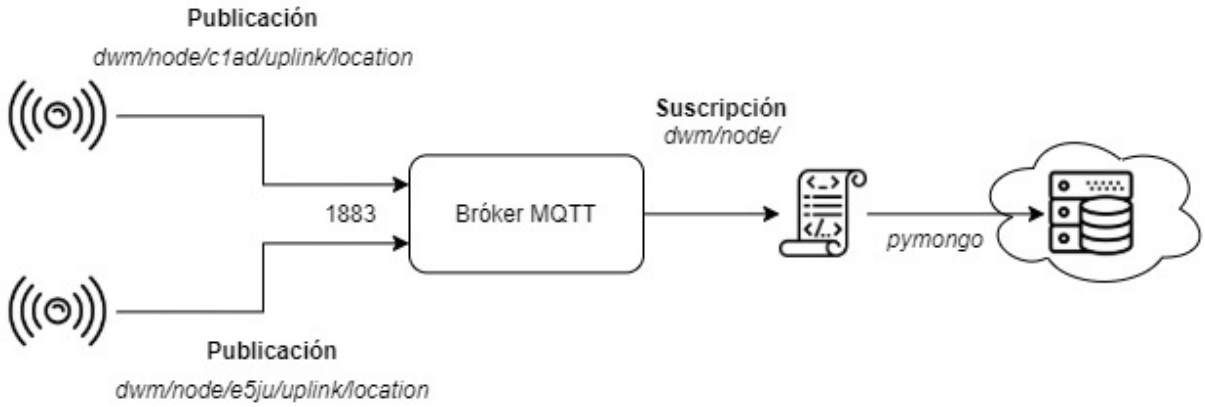


Figura 6.22: Obtención datos localización y procesamiento

6.2 Base de datos

La base de datos constituye el elemento central del sistema. En esta se almacena la información de los sensores y esta información es usada por los scripts de procesamiento de datos y detección de actividades y por otra parte, por el sistema web con la finalidad de mostrar los resultados. Por lo tanto, la base de datos debe estar accesible desde Internet para que todos los elementos del sistema puedan acceder a ella. En este caso, se trata de una base de datos NoSQL, concretamente, una base de datos MongoDB. Esta base de datos almacena colecciones de documentos (tablas y registros en caso de bases de datos SQL). Además, los documentos están en formato JSON, por lo que permiten su fácil manipulación en lenguajes como JavaScript y Python, utilizados en el sistema actual. En este caso, para alojar la base de datos se ha hecho uso de un servicio en la nube denominado MongoDB Atlas. En este apartado se comentan las diferentes colecciones que se han incluido en el sistema y los registros que contienen los documentos almacenados en ellas.

6.2.1 registros_actividad

El sistema de detección de actividades tanto para crear un dataset con datos definidos por el usuario como para analizar la corrección las actividades que ha detectado el algoritmo, necesita conocer cuáles son las actividades que en realidad el usuario ha realizado. Para ello, como se ha definido en la sección 6.5.3, se introducen las actividades que el usuario realiza en tiempo real, indicando cada vez que se inicia o finaliza . De esta forma, se tiene un *log* de estos cambios de actividad por parte del usuario. A continuación se muestran los campos de los documentos pertenecientes a esta colección.

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *l*: Hace referencia al *label* que se recibe por MQTT. Este indica el comienzo de un registro de actividades (start), el número de una actividad o el fin del registro de actividades (end).

- *persona*: Se almacena el identificador de la persona que realiza el registro de actividad.
- *numero_secuencia*: Es un valor que indica el número de dataset en el que se engloba la actividad que se recibe. Es necesario establecer este valor ya que posteriormente, los datasets se generan a partir de los registros de la colección registros_actividad.
- *t*: Se trata de un campo de valor entero que indica el instante de tiempo en el que se genera el registro.

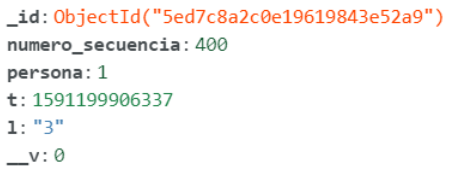


Figura 6.23: Base de datos - Colección registros_actividad

6.2.2 sensor_data

En esta colección se almacenan los datos que van a ser usados para realizar la detección de actividades. Concretamente, se almacenan los datos de los sensores de aceleración y giroscopio del *smartwatch*, de los sensores de localización y del resto de sensores ubicados en la *smart home*. Los documentos de esta colección no tienen siempre los mismos campos, cada tipo de sensor establece los campos para almacenar los datos que necesite. Esta es uno de los motivos por los que se escogió una base de datos no relacional. Los campos que puede contener esta colección son los siguientes:

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *topic*: Hace referencia al topic de MQTT del cual se consumen o reciben los datos.
- *tipo*: Se almacena el identificador del tipo de sensor. Pueden ser varios los tipos de estos sensores:
  - acc: Datos del sensor de acelerómetro.
  - gyr: Datos del sensor de giroscopio.
  - loc: Datos del sensor de localización.
  - bin: Datos de los sensores ubicados en la *smart home*.
- *persona*: Establece la persona que genera los datos.
- *timestamp*: Se trata de un campo de valor entero que indica el instante de tiempo en el

que se genera el registro.

- *estado*: En el caso de sensores de tipo ”bin”, se almacena un documento JSON con el estado de los sensores ubicados en la *smart home* en el momento indicado.

- x,y,z: Campos que almacenan las coordenadas de los sensores de aceleración, giroscopio y localización.


```
_id: ObjectId("5ede6702c271ed5d8eac1f95")
topic: "/case/inertial/wearA/gyr"
tipo: "gyr"
persona: 1
x: 1.1383474
y: -0.2943149
z: -0.7959574
datetime: 2020-06-08T18:27:44.000+00:00
tiempo: "08/06/2020 18:27:44"
timestamp: 1591633664750
```


Figura 6.24: Base de datos - Colección sensor_data

6.2.2.1 Índices

Esta colección almacena los datos que van a ser procesados por el algoritmo de reconocimiento de actividades. Dado que se tiene una gran cantidad de datos por segundo y es necesario procesarlos en tiempo real, es necesario que las consultas a estos datos se realicen de la forma más rápida posible. En este caso, se hizo uso de una base de datos MongoDB por su almacenamiento de los datos en formato JSON y por la flexibidad que proporciona. Sin embargo, una de las características más importantes de esta base de datos consiste en la posibilidad de incorporar lo que se conoce como *índices*.

De forma predeterminada, MongoDB y otros motores de bases de datos almacenan los documentos sin seguir un orden predeterminado, por lo que cada vez que se quiere realizar una consulta, es necesario recorrer todos los elementos de dicha colección y seleccionar los que cumplan los requisitos del usuario. Sin embargo, MongoDB permite la incorporación de *índices*, es decir, estructuras de datos que almacenan una porción del conjunto de datos (MongoDB, 2020). El índice ordena el valor de un campo específico o conjunto de campos, ordenados por el valor del campo. De esta forma, la colección se encuentra ”ordenada” y una vez el usuario quiera realizar una consulta en esta, se hará de forma mucho más rápida. Esto es posible gracias a que los índices de MongoDB se generan en forma de árbol-B, es decir, en forma de árbol pero manteniendo un balanceado entre estos (GenBeta, 2020).

Teniendo en cuenta el funcionamiento de los índices en la base de datos usada, se define un índice que hace que las consultas a los valores generados por los sensores sean muy rápidas, permitiendo obtener estos datos y analizarlos en tiempo real. Sin índices se ha comprobado que realizar el procesamiento en tiempo real en el sistema diseñado es imposible.

El índice que se ha incorporado consiste en establecer los campos timestamp, tipo y persona. De esta forma, dado que el algoritmo de reconocimiento de actividades realiza consultas a estos campos, estas consultas son muy eficientes, al tener los documentos ordenados. El índice definido se muestra en la Figura 6.25.

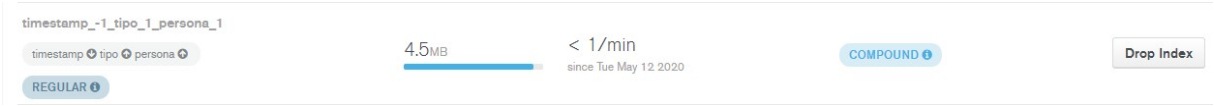


Figura 6.25: Base de datos - Colección sensor_data - Índice

6.2.3 Dataset

Dado que el sistema hace uso de un modelo de *machine learning* para realizar el reconocimiento de actividades, los modelos de *machine learning* contienen registros de entrenamiento para inicializarse y poder clasificar nuevos datos. El conjunto de datos de entrenamiento se denomina dataset. En esta colección se almacena información referente a cada dataset que se genera. Los campos son los siguientes:

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *num_secuencia*: Es un valor que indica el número que identifica el dataset dentro de los diferentes datasets. Este valor es único y se usa en los diferentes scripts como identificador.
- *nombre*: Campo que almacena del nombre del dataset.
- *descripcion*: Almacena una descripción que da el usuario del dataset.
- *actividades*: Se trata del conjunto de actividades que forman parte del dataset. Se almacenan siguiendo la estructura de un diccionario cuya clave es un valor entero del 1 al 15 y el valor es el nombre de la actividad.
- *fecha*: Se almacena la fecha de creación en formato *Date*.
- *fecha_str*: Se almacena la fecha de creación del dataset en formato de texto.

En la Figura 6.26 se muestra el documento que corresponde al dataset de actividades definitivo del que se hace uso en el sistema.

6.2.4 registros_ml

En los algoritmos de *machine learning*, para poder clasificar nuevos datos es necesario partir de datos correctamente clasificados por el usuario. Esto es lo que se conoce como el *dataset*. A partir de estos datos del dataset, el algoritmo se encarga de clasificar cada nuevo dato. En

Figura 6.26: Base de datos - Colección dataset

esta colección se almacenan los datos que forman parte del dataset, y contienen los siguientes campos:

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *numero_secuencia*: Valor de tipo entero que indica el número que referencia el dataset al que pertenece el dato correctamente clasificado.
- *timestamp*: Campo de valor entero que indica el instante de tiempo en el que se genera el registro.
- *datos*: Array que contiene todos los valores de los campos forman cada registro.
- *actividad*: Actividad a la que corresponde el registro del dataset.
- *columnas*: Estructura de datos de formato array que almacena el nombre de los campos del registros. Todos los documentos de un mismo dataset tienen que tener las mismas columnas para que la clasificación se haga de forma correcta.
- *persona*: Identificador de la persona que ha generado el registro.
- *numero_secuencia_analisis*: Identificador correspondiente al dataset de analisis que se usa con el fin de realizar la obtención de actividades en tiempo real. Los registros originales del dataset no tienen este campo. Sin embargo, al realizar analisis en tiempo real, las actividades bien clasificadas se añaden al dataset para que aprenda de estas.
- *creado_a_partir_analisis*: Se trata de un valor de tipo booleano que indica si el registro ha sido creado en el dataset original o posteriormente con la clasificación de las actividades en tiempo real.

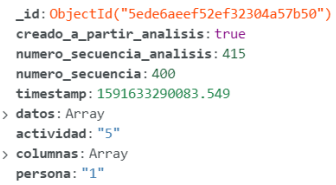


Figura 6.27: Base de datos - Colección registros_ml

6.2.5 actividades_obtenidas_ml

Una vez que se ha realizado la creación de un dataset, se puede comenzar a clasificar nuevos datos. En esta colección se almacenan los registros que se clasifican por el algoritmo o algoritmos de *machine learning* que se usan en la detección de actividades. A continuación se indican los campos de esta colección.

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *numero_secuencia_dataset*: Valor de tipo entero que indica el número que referencia el dataset a partir del cual se ha clasificado este registro.
- *numero_secuencia*: Campo de valor entero que indica el número que referencia al conjunto de registros que se han analizado y clasificado por el algoritmo. Este valor se incorpora para posteriormente, poder calcular el grado de certeza de los diferentes algoritmos.
- *actividad*: Actividad que se ha obtenido al clasificar el registro.
- *inicio*: Campo de valor entero que indica el instante de tiempo en el que comienza la ventana de tiempo a analizar.
- *fin*: Campo de valor entero que indica el instante de tiempo en el que finaliza la ventana de tiempo a analizar.
- *algoritmo*: Se trata del campo que establece el identificador del algoritmo de *machine learning* que ha clasificado el registro actual.
- *persona*: Identificador de la persona que ha generado el dato clasificado.

6.2.6 actividades_realizadas

Una vez que se ha generado un dataset y se quieren clasificar nuevas actividades en base a este para comprobar que los modelos de *machine learning* se han comportado correctamente, es necesario realizar datasets de prueba. Estos datasets estarán formados por los datos clasificados en tiempo real por los algoritmos de *machine learning* pero además, tienen que tener asociadas las actividades reales que se están haciendo en cada momento. De esta forma, se puede relacionar

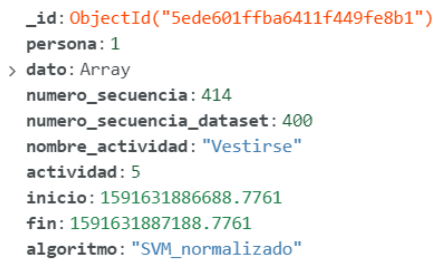


Figura 6.28: Base de datos - Colección actividades_obtenidas_ml

lo que está pasando en la vida real con los datos que están generando los algoritmos y poder medir el grado de acierto de estos. En esta colección se almacenan las actividades reales que el usuario realiza en la vivienda.

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *timestamp_inicio*: Campo de valor entero que indica el instante de tiempo en el que comienza la actividad el usuario.
- *timestamp_inicio*: Campo de valor entero que indica el instante de tiempo en el que finaliza la actividad el usuario.
- *persona*: Se almacena el identificador de la persona que realiza el registro de actividad.
- *numero_secuencia*: Valor que indica el número que identifica el conjunto de actividades que el usuario realiza en un momento dado para comprobar la certeza de los algoritmos de *machine learning*.
- *persona*: Se trata del identificador de la persona que realiza la actividad

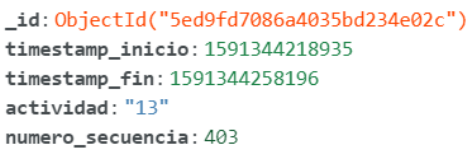


Figura 6.29: Base de datos - Colección actividades_realizadas

6.2.7 actividades_resultado

Cuando se produce la clasificación de la actividad que ha ocurrido cada ventana de tiempo, es necesario que la información sea mostrada en el sistema web. Sin embargo, cuando se produce una actividad durante una gran cantidad de tiempo de forma seguida, si se muestran todos los

registros correspondientes, al final el usuario no visualiza cuánto tiempo ha durado. Por esto, es necesario ”agrupar” los registros que proporciona el modelo de *machine learning* y mostrar al usuario información resumida. En esta colección se almacena esta información, que contiene los siguientes campos:

- *_id*: Todos los documentos por defecto contienen un campo _id único en la base de datos.
- *actividad*: Valor de tipo entero que referencia de forma única la actividad que se ha realizado.
- *inicio*: Campo de valor entero que indica el instante de tiempo en el que comienza la actividad el usuario.
- *fin*: Campo de valor entero que indica el instante de tiempo en el que finaliza la actividad el usuario.
- *persona*: Se almacena el identificador de la persona que realiza la actividad.
- *duracion*: Se almacena la duración de la actividad en segundos.
- *duracion_str*: Este campo almacena la información sobre la duración de la actividad en formato hh:MM:ss
- *inicio_str_fecha*: Este campo almacena la información sobre la fecha en la que se realiza la actividad en formato dd/mm/yyyy.
- *inicio_str_hora*: Este campo almacena la información sobre la hora en la que se realiza la actividad en formato hh:MM:ss.
- *fin_str*: Este campo almacena la información sobre la fecha en la que acaba la actividad en formato dd/mm/yyyy hh:MM:ss.
- *nombre_actividad*: Se trata del campo que almacena el nombre de la actividad asociado al campo de valor entero *actividad*.

En la Figura 6.30 se muestra la información referente a la actividad ”Ordenador” que ha realizado un usuario en un momento dado.


```
_id: ObjectId("5edfc0af53fc9f85fa161c34")
actividad: 4
inicio: 1591722139145.782
fin: 1591722146145.782
persona: 1
duracion: 7
duracion_str: "0:00:07"
inicio_str_fecha: "09/06/2020"
inicio_str_hora: "19:02:19"
fin_str: "09/06/2020 19:02:26"
nombre_actividad: "Ordenador"
```


Figura 6.30: Base de datos - Colección actividades_resultado

6.3 Machine learning

Hasta hace unos años, la mayoría de programas hacían uso de programación tradicional, en la cual a partir de una serie de condiciones, se ejecutan ciertas acciones y se puede controlar el flujo de los programas con las diferentes estructuras de control. Sin embargo, dado que existen multitud de problemas que no pueden ser resueltos mediante la programación tradicional ya que no se abarcan todos los casos posibles de estos problemas, surge la necesidad de cambiar la forma de abordarlos. Aquí surge la Inteligencia Artificial (IA), que trata de resolver los problemas sin programar de forma explícita una solución al mismo, sino proporcionando información a la máquina para que esta pueda resolver el problema y extraer conocimiento de él con el objetivo de resolver futuros problemas (Moroney, 2021).

En el caso del problema actual, se tiene una secuencia de datos proporcionados por sensores que se encuentran en una *smart home* y en un *smartwatch*. En el caso de querer obtener las actividades que realiza el usuario, para cada nueva actividad a detectar, se debería modificar el programa de detección de actividades e incorporar las condiciones que cumplen cada actividad nueva. Sin embargo, cuando se tiene en cuenta una gran cantidad de datos o condiciones para cada actividad y en algunos casos tratándose de actividades muy similares entre sí, sería muy complicado realizar esto mediante programación tradicional. Por lo tanto, aquí entra en juego lo que se conoce como *machine learning*.

El *machine learning* o aprendizaje automático es una rama de la Inteligencia Artificial (IA). Concretamente, consiste en la extracción de patrones en un conjunto de datos a través de ciertos algoritmos o modelos con el objetivo de obtener conocimiento sobre estos datos y poder realizar predicciones futuras sobre nuevos datos. Además, con el tiempo, los algoritmos de *machine learning* aprenden de los nuevos datos y se mejora el grado de conocimiento de estos datos, proporcionando mejores predicciones. Por lo tanto, en el caso del reconocimiento de actividades, el uso del aprendizaje automático está muy establecido ya que proporciona sistemas que se adaptan correctamente a variaciones dentro de una misma actividad y a nuevas actividades.

Además, dentro del aprendizaje automático se encuentran varios enfoques (Iberdrola, 2019). En primer lugar, se tiene el **aprendizaje supervisado**, en el cual existe un conjunto de datos (dataset) etiquetados correctamente por el usuario en base a la experiencia u observación. Por lo tanto, el algoritmo "aprende" de estos datos correctos y realiza clasificaciones de nuevos datos en base a estos. El objetivo de este tipo de algoritmos consiste en definir la nueva "clase" de cada secuencia de datos que recibe. Por otra parte se encuentra el **aprendizaje no supervisado**. En este aprendizaje, a diferencia del anterior, los datos de los que se hace uso no están etiquetados, de modo que es tarea del algoritmo realizar este etiquetado, encontrando patrones entre ellos y organizándolos de la forma que más oportuno estime. Por lo tanto, el usuario en este caso no tiene repercusión en las decisiones que toma el algoritmo. En último lugar, se encuentra el aprendizaje por refuerzo, en el cual el algoritmo aprende mediante el método de prueba/error, recompensando las buenas decisiones para en un futuro generar acciones similares a las acciones que dieron mejor resultado. De esta forma, el algoritmo aprende de forma autónoma.

Por lo tanto, una vez considerados los diferentes enfoques dentro del campo del *machine*

learning, el enfoque que más se adecua al problema actual se trata del aprendizaje automático supervisado en el cual el usuario proporciona un conjunto de actividades junto con un conjunto de datos que corresponden a cada actividad y el algoritmo clasifica nuevas actividades en base a estas. De igual forma, cuando el usuario desee incorporar nuevas actividades a reconocer, bastará con proporcionar una secuencia de las nuevas actividades a detectar y el algoritmo automáticamente tiene en cuenta estas nuevas actividades en futuras clasificaciones.

6.3.1 Algoritmos de machine learning

Dentro del campo del aprendizaje supervisado, existe una gran cantidad de algoritmos con distinto comportamiento que se adaptan mejor a unos problemas u otros. Por lo tanto, en esta sección se estudian diferentes algoritmos que pueden arrojan buenos resultados en el problema de la detección de actividades en *smart home*s. Así, los algoritmos que se describen a continuación se encuentran en la librería *scikit-learn*, por lo que su inclusión en el código python es muy sencilla y se pueden comparar los modelos entre sí de forma sencilla.

6.3.1.1 kNN

El algoritmo *kNN* se trata de un algoritmo de *machine learning* supervisado. Este algoritmo no genera un modelo para la clasificación de los nuevos datos, sino que con el fin de realizar esta clasificación se basa en las instancias clasificadas previamente. Para clasificar un nuevo registro, el algoritmo calcula la distancia entre este y cada registro almacenado en la base de datos del algoritmo, clasificándose según sean las características de los K registros más similares a este.

Dado que el algoritmo se basa en las instancias clasificadas para clasificar los nuevos datos procedentes de los sensores, se tiene que comprobar que las instancias han sido correctamente clasificadas con el objetivo de no contaminar la base de datos y que el algoritmo funcione correctamente. En otras palabras, hay que *guiar* al algoritmo principalmente en sus primeras ejecuciones para que pueda realizar buenas clasificaciones posteriormente.

El funcionamiento del algoritmo sigue las siguientes etapas:

- Obtención de un registro a clasificar: Se proporciona al algoritmo un registro con el objetivo de ser clasificado con alguno de los valores posibles definidos por el usuario.

- Cálculo de distancia entre el registro a clasificar y el *dataset*: Una vez que se ha proporcionado un elemento para clasificar, se calcula la distancia entre este elemento y los registros que se encuentran almacenados en el *dataset*, clasificados con anterioridad. Con el propóstivo calcular esta distancia, se suele hacer uso de las distancias *Euclídea*, *Manhattan*, *Hamming* o *Minkowsky*. A menor valor de distancia entre dos elementos, más próximos se encuentran, es decir, más similares son.

- Selección de los K vecinos más cercanos: El usuario o el sistema define un valor para K. Este

parámetro se establece para seleccionar los K registros que menos distancia tengan respecto al registro a clasificar. Una vez que se tienen estos registros, la clase que predomine en estos será la clase que se asignará al registro a clasificar. Este parámetro puede ser obtenido mediante diferentes métodos para optimizar el resultado del algoritmo *kNN* aplicado a cada tipo de problema.

- Almacenamiento: El registro que se acaba de clasificar se almacena para poder ser usado en futuras clasificaciones de nuevos registros.

En la Figura 6.31 se puede observar la clasificación de un nuevo registro en una de las dos clases, *claseA* y *claseB*. En primer lugar, se calcula la distancia entre el nuevo registro y los demás registros del dataset siguiendo una función de distancia. Una vez se han calculado las distancias, se seleccionan los k registros más próximos. En este caso, se tratan de dos elementos de la *claseB* y un elemento de la *claseA*. Una vez obtenidos los k registros más parecidos al registro a clasificar, se selecciona la clase que más presente está en los registros seleccionados; en este caso, la *claseB*.

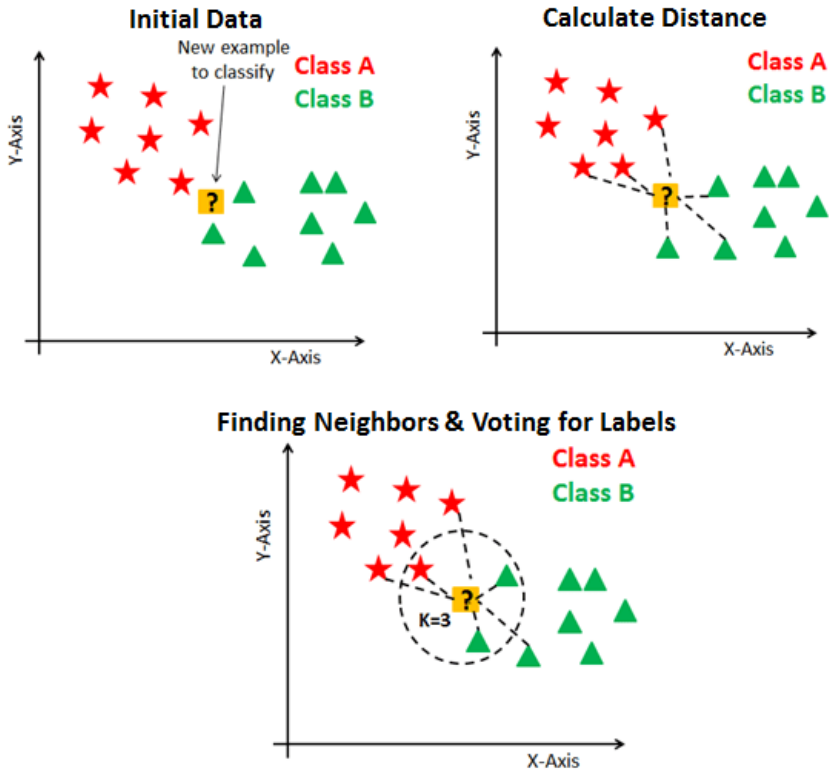


Figura 6.31: kNN - Funcionamiento

Las ventajas del algoritmo son las siguientes (Alsheikh y cols., 2014):

- Tiene una alta tolerancia al ruido, ya que se basa en instancias similares, por lo que el

ruido no afecta.

- Clasifica generalmente bien en muestras de tamaño no superior a 100.000 registros, al menos en la librería que se ha usado para ejecutar el algoritmo.

Los inconvenientes del algoritmo son las siguientes:

- A medida que el conjunto de datos almacenado crece, el tiempo de cómputo para la clasificación es mayor.
- La predicción del valor óptimo de K no se puede realizar con facilidad.
- El coste del algoritmos es *O(n x d)*, siendo n el número total de datos en los que se basa el algoritmo con la finalidad de realizar la clasificación y d la dimensionalidad de estos datos, es decir, el número de campos que forma cada dato (Deng y cols., 2017).
- El sistema debe tener alta capacidad de almacenamiento para guardar todos los datos clasificados.

6.3.1.2 SVM

Las máquinas de vectores de soporte (SVM) es otro de los algoritmos más usados en cuanto a la detección de actividades. El objetivo principal de *SVM* consiste en la generación de una limitación (hiperplano) entre las diferentes clases (actividades en este caso) de tal forma que esta limitación esté lo más separada posible de estas clases, diferenciando de esta forma el conjunto de datos y acotando la "región" correspondiente a cada clase. Por lo tanto, cuanto más separado esté el hiperplano de los elementos de las clases, más acotadas están estas clases y mejor clasificación se hará posteriormente (IArtificial, 2019b).

En la Figura 6.32, obtenida de (Wikipedia, 2019b), se puede apreciar un conjunto de datos con dos tipos de clases, una blanca y otra negra. El algoritmo define tres hiperplanos, *h1*, *h2* y *h3*. Como se puede observar, el hiperplano *h1* no separa correctamente las clases, por lo que la clasificación no se realizará de forma correcta, ya que hay elementos de las dos clases en ambos lados del hiperplano. Por otro lado, el hiperplano *h2*, aunque separa correctamente las clases, no lo hace con suficiente distancia, por lo que cuando se quiera clasificar un nuevo dato, la fiabilidad no será muy grande ya que si por ejemplo, este nuevo dato corresponde a la clase blanca y está situado en la parte inferior izquierda, lo clasificará como clase negra, cuando a simple vista se ve que corresponde a la clase blanca. Finalmente, el hiperplano *h3* define separación entre los elementos de las dos clases de forma que la distancia a los datos más próximos de las clases es la más elevada posible. De esta forma, la separación permitirá clasificar correctamente nuevos datos.

Así, el hiperplano del *SVM* tendrá tantas dimensiones como elementos tenga el vector de características del problema. En el caso de la Figura 6.32 el hiperplano es de dos dimensiones, pero en la mayoría de los problemas (y en el caso actual), los registros tienen una gran canti-

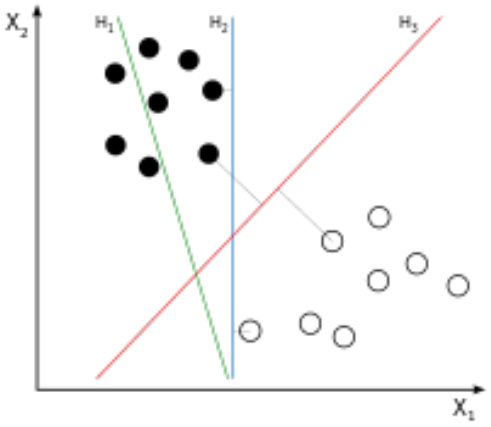


Figura 6.32: SVM - Hiperplanos

dad de características y por tanto el hiperplano no se podrá representar gráficamente, pero el funcionamiento es el mismo. Así, para calcular el hiperplano se usan funciones de minimización de tal forma que el hiperplano resultante minimice esta función. Dependiendo del algoritmo de *SVM* concreto que se use, se tiene una función u otra (Learn, 2019c).

En algunos casos, es necesario incluir una nueva dimensión en los datos, porque con las dimensiones actuales no se pueden delimitar correctamente las clases. Para hacer esto, se hace lo que se conoce como *kernel*. Gracias a esta nueva dimensión, el algoritmo podría diferenciar correctamente las clases.

En definitiva, *SVM* es muy efectivo cuando se tiene un dataset con muchas dimensiones, además, al hacer uso de una función que define el hiperplano que separa las clases, no es necesario en cada nueva clasificación recorrer el dataset, sino que se comprueba con la función y se clasifica como determine.

6.3.1.3 Random Forest

Los árboles de decisión son una técnica de *machine learning* de aprendizaje supervisado que se encarga de formar un árbol de diferentes niveles con decisiones que dan lugar a un resultado final. El resultado final consiste en nodos hoja que se corresponden con la clase que se asocia al dato que se está clasificando.

El funcionamiento de los árboles de decisión se puede observar en la Figura 6.33, obtenida de (Wikipedia, 2019a). En ella, se puede observar un ejemplo de árbol de decisión. En este caso, se quiere predecir si una persona a bordo del *Titanic* vivió o falleció en base a las caraterísticas de esta. Para decidir, el árbol tiene tres niveles. Este define que si una persona es hombre, no sobrevive. En el caso de ser mujer, si se tiene menos de 9.5 años, la mujer muere. En el caso de ser mayor, si esta tiene más de 2.5 familiares a bordo, sobrevive, en caso contrario no. Siguiendo este árbol, se clasificarían los pasajeros para conocer si sobrevivieron o no. Como se puede apreciar, este ejemplo de árbol no es muy correcto ya que, aunque sobrevivieron más mujeres que hombres,

hubo bastantes hombres que no fallecieron.

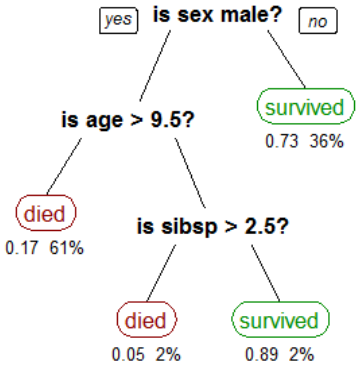


Figura 6.33: Árbol de Clasificación

Una gran cantidad de aplicaciones hace uso de árboles de decisión para realizar clasificaciones de nuevos datos, ya que generalmente se comportan bien, aunque sobreajusten. El sobreajuste consiste en que los algoritmos aprenden muy bien de los datos de entrenamiento pero cuando se quieren comprobar nuevos datos, puede que no los clasifique correctamente porque no se ha generalizado bien.

Por esta razón surge la necesidad de crear lo que se conoce como *Random Forest Classifier*. Esto consiste en crear un conjunto de árboles de decisión creados a partir de un conjunto de datos de entrenamiento y el resultado del algoritmo es la combinación del resultado de los sub-árboles que lo conforman (IArtificial, 2019a). De esta forma, como cada árbol de decisión ha sido entrenado con unos datos diferentes, cuando se desea clasificar un nuevo dato, cada árbol proporcionará un resultado, siendo el resultado final la combinación de los resultados de cada sub-árbol. Por lo tanto, se realiza una mejor generalización del conjunto de datos de entrenamiento.

En cuanto a la parametrización del *Random Forest*, normalmente hay que tener en cuenta el número de sub-árboles que se quieren generar. Normalmente, a mayor número de sub-árboles, mejor generalización del problema se tiene y por tanto, menos sobreajuste y mejor clasificación. Sin embargo, hay que tener en cuenta que con un número muy elevado de sub-árboles, el tiempo es mayor.

En general, como se ha comentado, generaliza bien ante nuevos registros al tener en cuenta el resultado de los sub-árboles. Sin embargo, al no seguir un modelo matemático como en otros algoritmos de *machine learning*, no se ajusta bien a algunos problemas. Además, hay que tener cuidado con el número de sub-árboles que se generan porque un número elevado de árboles hará que no sea eficiente el algoritmo.

6.3.2 Preparación de datos

El proceso de *machine learning* está fundamentado en los datos de los que se disponen para ejecutar los diferentes modelos. Sin embargo, los datos de los que se hacen uso en los modelos pueden provenir de varias fuentes, tener diferentes formatos, estar incompletos, etc. Por esto, es necesario un proceso de preprocesamiento y preparación de los datos para poder hacer uso de los modelos y que estos den el resultado esperado.

6.3.2.1 Gestión de datos incompletos

Cuando se hace uso de un dataset o se genera uno manualmente, son muchos los casos en lo que los registros que forman parte del dataset no contienen todos los campos necesarios para que el modelo haga uso de ellos. Por lo tanto, es necesario gestionar esta incompatibilidad y hay varios enfoques.

- Eliminación: Puede optarse por eliminar los registros que contienen campos vacíos. Sin embargo, dado que hay datasets con una gran cantidad de registros incompletos, no es aplicable a todos los datasets.
- Imputación de nuevos valores: En el caso que no se quiera eliminar el registro completo y sea necesario que todos los campos tengan un valor, se puede optar por rellenar este campo de forma automática. Puede determinarse que siempre se rellene con el mismo valor o hacer uso de técnicas como media, mediana, etc.

Concretamente, en este proyecto se rellenan los valores que faltan haciendo uso de una función de la librería *pandas*, rellenando el valor del campo a 0.

6.3.2.2 Normalización de datos

Dentro de un mismo registro, los valores de sus campos pueden ser muy dispares entre sí. Dado que los principales modelos de clasificación usan funciones de distancia para calcular la similitud o relación entre los registros, cuando existen campos con valores en una escala mucho mayor al resto, estos campos tienen una mayor importancia en la clasificación (Learn, 2019b).

En este caso, es necesario hacer uso de la normalización de los campos, ya que los datos de los sensores binarios son valores entre 0 y 1, los valores de los sensores de localización entre 0 y 15 y los valores de la aceleración y giroscopio entre -15 y 15. En la Figura 6.34 se puede apreciar esta diferencia que se ha comentado.

Por lo tanto, con el fin de hacer una correcta clasificación de nuevos datos y que las actividades con grandes valores de aceleración, giroscopio y localización no ”alteren” la clasificación, se hace uso de la normalización de estos datos. Para realizar esto, hay varios enfoques.

[5.9150577, -4.0007997, 0.7053083708, 5.814559, -0.148355, 2.771845769948, 3.3044882, -0.2727818, 1.3880896672307271, -1.6223338, -14.684752, -11.022489548, -0.8997013, -15.081961, -9.4762476656, -7.422536, -10.425529, -8.633622, 1.8894674, -2.1385782, -0.033207672044145306, 0.0, 0.0, 0.0, 0.8636413, -0.9378003, -0.3558412150400000 5, 0.10128146, -1.5474439, -0.16519375539179076, 0.94061035, -1.0746342, 0.009689158453247866, 0.0, 0.0, 0.0, 1.5131133, -0.657291, 0.71615397, 0.17654005, -0.26389378, -0.011224696869199998, 1.7959439, -2.2162192, -0.007953261323726535, 1.2, 0.0, 0.5625, 1.2, 0.0, 0.6666666666666666, 1.2, 0.0, 0.5684210526315788, 3.5, 0.0, 1.640625, 3.5, 0.0, 1.9444444444444444, 3.5, 0.0, 1.6578947368421053, 1.0, 0.0, 0.46875, 1.0, 0.0, 0.5555555555555556, 1.0, 0.0, 0.47368421052631576, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]

Figura 6.34: Valores de un registro

6.3.2.2.1 Estandarización En algunos casos, los modelos de *machine learning* que se usan para clasificar nuevos datos necesitan que los datos procedentes del dataset tengan atributos que sigan una *distribución normal*, por ejemplo Gaussiana, es decir con una media de cero y una varianza unidad. De esta forma, se distribuyen en torno a un valor central y no se tiene mucha dispersión en los datos.

En este caso, para realizar la estandarización de los datos, se hace uso de la Fórmula 6.1.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{6.1}$$

donde x es un campo del dataset, *min(x)* es el valor mínimo del campo y *max(x)* el valor máximo.

6.3.2.2.2 Escalado de características Algunas veces, se quiere que los diferentes atributos de los registros tengan la misma importancia a pesar de tener valores muy dispares, como es el caso actual. Para ello, se realiza un escalado de los valores a un rango, por ejemplo entre 0 y 1. Este es el caso del escalado máximo-mínimo, en el cual el valor mínimo de un atributo tendrá el valor de 0 y el máximo de 1. El resto se distribuye en el intervalo acorde a su valor inicial. De esta forma, se mantiene la distribución de los datos y todos los atributos tendrán valores comprendidos entre 0 y 1.

Para hacer esto, se sigue la Fórmula 6.2.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{6.2}$$

donde x es un campo del dataset, $\bar{x}$ es la media de los valores del campo y σ es la desviación típica de los valores.

6.3.2.3 Selección de atributos

A veces, se piensa que cuantas más características tengan los registros de un dataset (dimensiones), mejor clasificaciones se harán porque tiene más información el algoritmo de *machine learning* para clasificar nuevos datos. Aunque en la mayoría de las ocasiones esto es así, hay veces en las que algunas características "entorpecen" las decisiones del algoritmo porque no son relevantes y el algoritmo piensa que sí. Por esto es necesario realizar una selección de los atributos o características más importantes de los datos a la hora de generar el modelo y ajustarlo con los datos (Quesada y cols., 2015).

Sin embargo, esta tarea no sigue unas reglas establecidas que se aplican a todos los problemas y se seleccionan las mejores características de los datos, sino que dependen de cada dataset. Así, en el desarrollo del proyecto se proponen varios métodos de selección de atributos, que se exponen a continuación (Learn, 2019a).

6.3.2.3.1 Eliminación de atributos con poca varianza Un primer enfoque para proceder a la eliminación de los atributos consiste en eliminar los atributos cuya varianza sea cero (todos son iguales) o cualquier regla que establezca el usuario, como por ejemplo el atributo que tenga un mismo valor en el 80% de los casos. Así, se consigue eliminar las características que a priori entorpecerán la clasificación, permitiendo al modelo centrarse en las características que más información pueden proporcionar.

6.3.2.3.2 Selección de los mejores *k* atributos Puede darse el caso en que el usuario quiera quedarse con los k mejores atributos de cada registro en base a algún método de importancia de atributos, como puede ser la distribución de Pearson, el análisis de la varianza o la correlación entre variables.

6.3.2.3.3 Selección de atributos con meta-transformadores Una de las técnicas más usadas para la selección de atributos consiste en la utilización de modelos de *machine learning* para realizar esta selección. Este es el caso de los meta-transformadores. En la librería *scikit-learn* que se usa incorpora para hacer uso de los modelos de *machine learning*, se incorpora esta funcionalidad. En ella, se puede hacer uso de modelos lineales para eliminar los atributos cuya importancia sea baja y no importe mucho al realizar el ajuste del modelo.

6.4 Análisis de datos en tiempo real

En el sistema desarrollado, un requisito fundamental es la detección de actividades en tiempo real. Esto es así porque se tiene como objetivo poder incorporar este sistema en entornos reales en los cuales convivan personas mayores o personas con problemas y es necesario la monitorización constante de estos. De esta forma, se pueden detectar situaciones de emergencia muy

rápidamente.

Sin embargo, en este caso, se tiene un gran conjunto de datos por lo que es necesario incorporar un sistema de procesamiento de datos en tiempo real haciendo uso de diferentes tecnologías. En este apartado se aborda toda esta problemática y las soluciones que se implementan para cumplir con el objetivo principal.

6.4.1 Problemática

El sistema IoT desarrollado tiene como objetivo obtener las actividades que ocurren en la *smart home* en tiempo real. Como se desarrolló en la sección 5.2, el sistema procesa los datos siguiendo una estructura de ventanas de tiempo. La cantidad de datos que hay que procesar para determinar la actividad que ocurre en una ventana de medio segundo es muy elevada teniendo en cuenta los sensores instalados en el sistema, tal y como se puede observar en la Figura 5.3.

En primer lugar, se implementó un algoritmo de procesamiento de datos que recogía los datos de la ventanas de tiempo correspondientes y clasificaba la actividad. Por lo tanto, para obtener la actividad que ocurre en medio segundo el algoritmo tenía que realizar las siguientes acciones:

- Obtener todos los datos de los sensores que corresponden a cada ventana de tiempo. Es decir, si se tienen 4 tipos de sensores y cada uno de ellos tiene definidas 3 ventanas de tiempo de diferentes tamaños, se tienen que hacer 12 consultas a la base de datos para obtener toda la información. Hay que tener en cuenta que conforme la ventana de tiempo es mayor, la cantidad de datos que se obtienen también es mayor.

- Calcular las métricas correspondientes a cada ventana. El siguiente paso después de acotar las ventanas y obtener los datos consiste en obtener los valores mínimo, máximo y media de cada ventana según corresponda. En el caso de sensores de localización e inerciales, se obtienen las métricas correspondientes a cada una de las coordenadas (x, y, z).

- Comprobar con un algoritmo de *machine learning*. Una vez obtenidas las métricas de cada sensor en cada ventana, se comprueban los resultados en el algoritmo de *machine learning* correspondiente.

Como se puede suponer, todas estas acciones no se podían realizar en medio segundo, provocando un retraso cada vez mayor ya que para analizar los datos de una ventana se tardaba alrededor de 6 segundos. De esta forma, en 1 minuto se procesaban datos correspondientes a 5 segundos, por lo que esta opción tuvo que ser desechada.

En la Figura 6.35 se muestran los datos y operaciones que se hacían en el algoritmo secuencial. Como se puede apreciar, para calcular las métricas de una ventana de tiempo, se tenía que realizar la consulta a la base de datos correspondiente y calcular las diferentes métricas. En la siguiente "iteración" del algoritmo (el siguiente medio segundo, ya que se trata de una ventana deslizante), volvía a repetirse el proceso, volviendo a consultar prácticamente los mismos datos (algunos de ellos cambiaban de ventana) y teniendo que volver a calcular las métricas. Esto

como se puede observar es un comportamiento muy ineficiente ya que se realizan consultas a la base de datos muy parecidas entre iteraciones, pudiendo evitar esto creando un sistema de ”almacenamiento en caché” que almacenase estos datos. Este sistema posteriormente se creó haciendo uso de **Redis Database**, como se comenta a continuación.

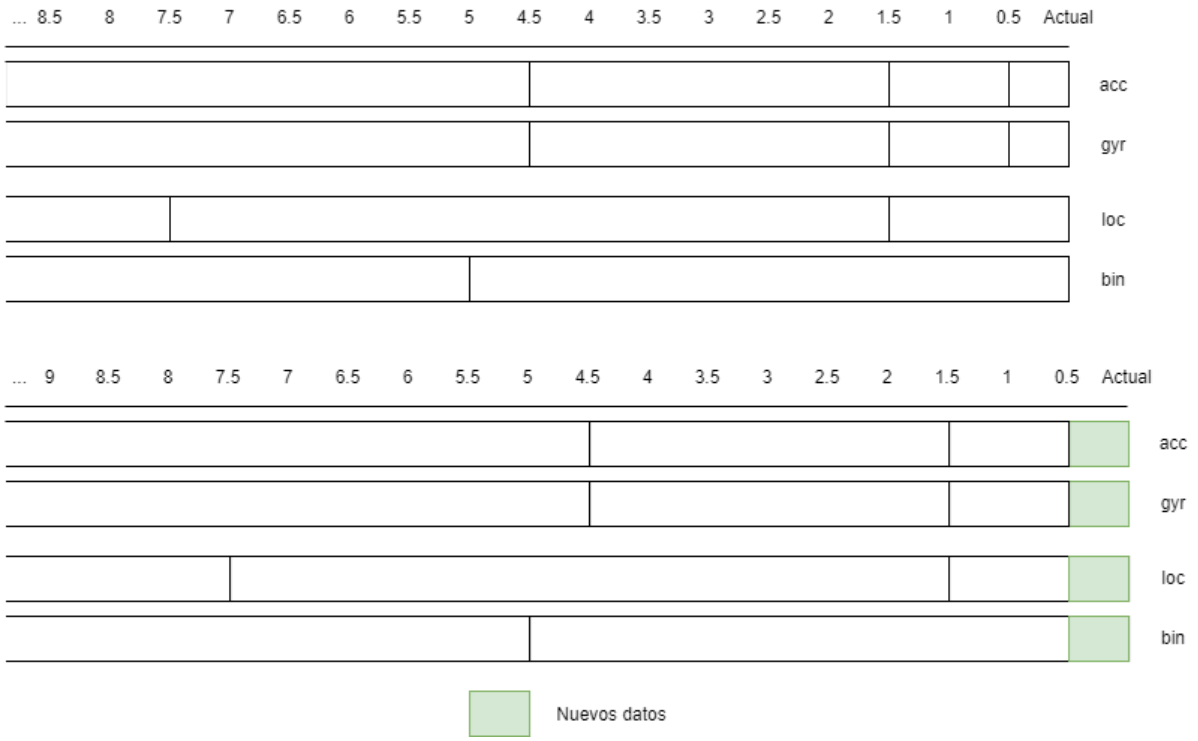


Figura 6.35: Consulta de datos cada medio segundo

Por otra parte, aún guardando los datos en un servicio de memoria caché, recibir los datos de la nueva ventana de tiempo, acceder a los datos de las ventanas antiguas, recalcular las métricas de las ventanas y clasificar la actividad en un solo proceso es una tarea que con los recursos del sistema no fue posible conseguir. Por esto se optó por incorporar una herramienta de procesamiento distribuido de datos entre diferentes procesos **Apache Kafka**. De esta forma, la carga se distribuye entre procesos y se permite procesar en tiempo real los datos de los sensores.

6.4.1.1 Redis Database

Como se ha comentado en la sección anterior, dado que cada medio segundo se obtiene la actividad que realiza el usuario y esta actividad tiene en cuenta datos de diferentes ventanas de tiempo anteriores, se necesitan una herramienta que actúe de caché que evite la consulta repetitiva a la base de datos. Esta herramienta se trata de Redis Database, ya que proporciona un sistema de base de datos noSQL en memoria principal de fácil y rápido acceso.

Redis almacena los datos siguiendo una estructura de clave-valor, por lo que no puede haber elementos con la misma clave. De esta forma, el acceso a un elemento dada su clave tiene complejidad O(1), no importa los elementos que haya, al ser único el valor, siempre va a tardar lo mismo. Analizando los datos que se acceden y procesan en la Figura 6.35, se pudieron observar numerosas operaciones redundantes.

En primer lugar, en cada nueva ventana de tiempo a analizar por el algoritmo, se realizaba una consulta muy parecida a la base de datos, estando los límites de tiempo desplazados 500ms. Esto se podía evitar guardando los datos de los sensores de cada ventana de tiempo en un nuevo registro en Redis. Sin embargo, como el desplazamiento de las ventanas de tiempo es de 500ms y en este desplazamiento se añaden datos, los datos de la ventana quedan obsoletos, porque tienen que incluir valores 500 ms más recientes y eliminar los datos de los 500ms más antiguos. En la Figura 6.36 se muestra el desfase entre las ventanas en el caso del sensor de aceleración. Cuando en la segunda ”iteracion” o ventana, se quiere acceder a los datos de la ventana de tres segundos, no se puede hacer uso de los datos almacenados en Redis con la etiqueta ”acc_1.5”, ya que están obsoletos porque incluyen datos antiguos y no recogen datos nuevos procedentes de la ventana de 1 segundo. Por esto, el enfoque de almacenar ventanas completas queda descartado.

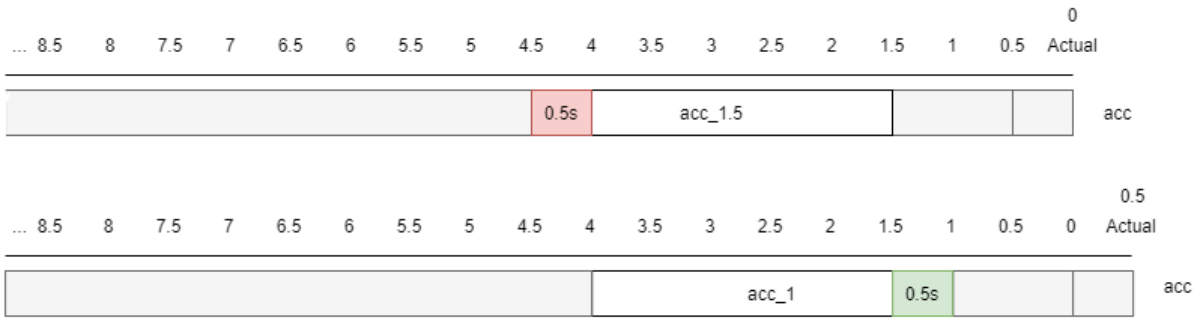


Figura 6.36: Ventanas cambiantes

Por lo tanto, se decidió fracccionar las ventanas de diferente tamaño de cada tipo de sensor en ventanas comunes equivalentes a la ventana de menor tamaño (500ms) como se muestra en la Figura 6.37, correspondiente a la ventana más actual de los sensores de aceleración y giroscopio. De esta forma, como el algoritmo procesa los datos cada 500ms, las ventanas almacenan los datos extraídos de la base de datos la primera vez y no se tienen que hacer más consultas de estos datos a la base de datos.

En segundo lugar, se observó que al dividir las ventanas grandes en un conjunto de ventanas pequeñas de 500ms, no se tenía que acceder a la base de datos tantas veces, pero se realizaba el cálculo de todas las métricas cada nuevo segundo, realizando trabajo repetido ya que muchas ventanas de tiempo no contienen valores importantes. Por esto, se decidió **en lugar de almacenar en la base de datos Redis las notificaciones de los sensores de cada ventana de tiempo, almacenar las métricas de estos valores** tal y como se muestra en la Figura 6.38.

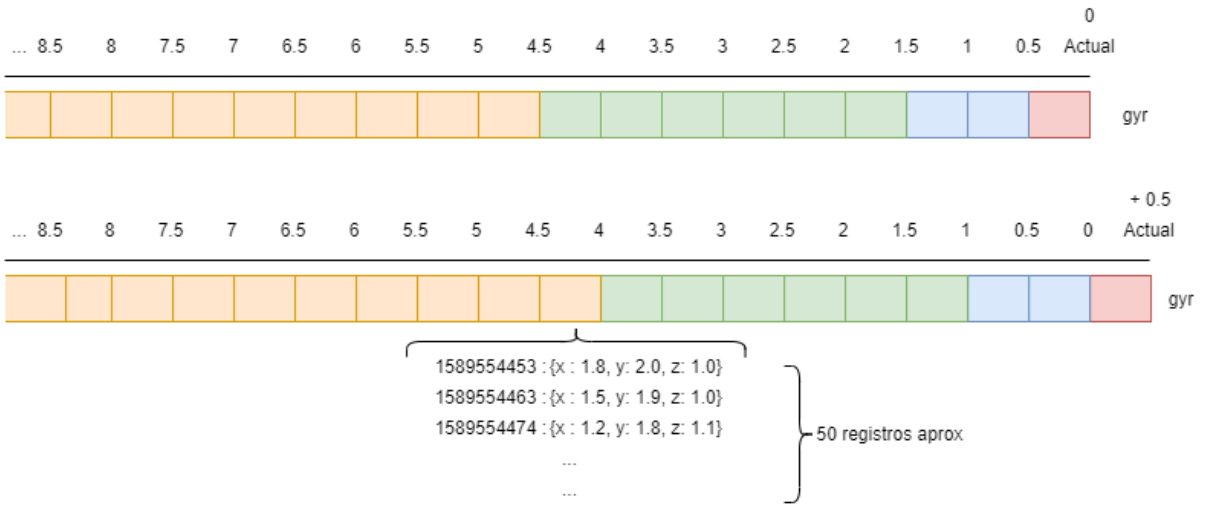


Figura 6.37: Primera optimización - Fraccionamiento en ventanas de 500ms

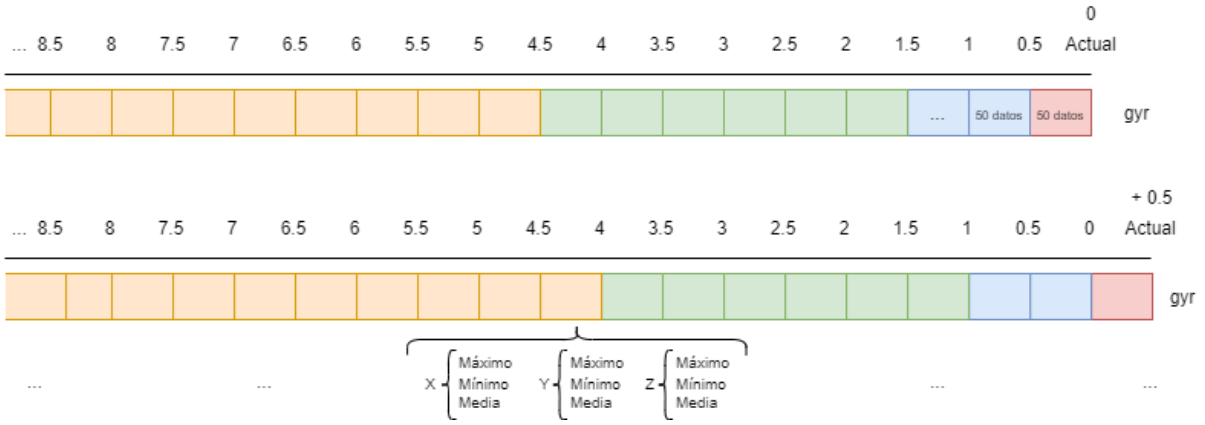


Figura 6.38: Segunda optimización - Almacenamiento metricas de ventanas de 500ms

De esta forma, para procesar los datos de medio segundo solamente se tienen que volver a calcular las métricas corespondientes a las ventanas que engloban a las ventanas de 500ms.

Por lo tanto, los datos que se almacenan en Redis siguen la estructura que se muestra en el Código 6.9.

Código 6.9: Almacenamiento en Redis métricas de ventanas de 500ms

```
{persona}_{sensor}_{timestamp} = {
    "x" :{
        "max" : 3.4,
        "min": -1.2,
        "avg" : 0.6
    },
    "y" :{
        "max" : 3.4,
```

```
        "min": -1.2,
        "avg" : 0.6
    },
    "z" :{
        "max" : 3.4,
        "min": -1.2,
        "avg" : 0.6
    }
}
```

6.4.1.2 Apache Kafka

Como se puede suponer, la obtención de toda la información de los sensores, segmentación, procesado y clasificación si se hace de forma secuencial es muy difícil que esta pueda realizarse en tiempo real, ya que las máquinas de las que se disponen no tienen los suficientes recursos para hacerlo posible. Por lo tanto, es necesario distribuir la carga de trabajo entre varios procesos dentro de la máquina.

El software que permite este reparto de la carga en tiempo real es Apache Kafka. Esta herramienta fue creada por Apache hacia el año 2014 y tiene como principal objetivo proporcionar una plataforma de almacenamiento y distribución de streams de datos entre varios procesos. De forma general, se puede afirmar que se trata de un servicio de publicación/suscripción que permite el procesamiento de streams de datos en tiempo real.

Los componentes principales de Apache Kafka son los siguientes:

- *broker*: Núcleo de Apache Kafka. Se trata del elemento que alberga diferentes ”topics” en los cuales los clientes productores insertan registros y los clientes consumidores reciben los registros de los topics a los que están suscritos. En definitiva, constituye el componente que redistribuye los mensajes entre procesos.

- *topic*: ”Tema” en el cual se insertan mensajes y posteriormente se distribuyen a los clientes que estén suscritos a este topic.

- *particion*: División que se realiza en cada topic. Cada partición almacena mensajes que se insertan en el topic. Por defecto un topic tiene una partición, almacenando todos los mensajes en una cola. Sin embargo, el valor de particiones puede cambiarse para tener varias colas de mensajes dentro de un topic y procesar los datos de forma distribuida.

- *productor*: Componente cliente que inserta nuevos valores en los topics del broker.

- *consumidor*: Componente cliente que se suscribe a ciertos topics del broker y cuando se producen nuevas inserciones en estos, recibe los nuevos cambios.

- *grupo de consumidores*: Conjunto de consumidores de un topic. El grupo de consumidores puede estar formado por uno o varios componentes consumidor. Si un topic tiene más de

un grupo de consumidores, el mensaje se reenvía a cada grupo de consumidores de forma separada.

En la Figura 6.39 se puede observar un caso de ejemplo del funcionamiento de Kafka. En ella se puede apreciar un broker que contiene un topic con tres particiones. Los nuevos mensajes que llegan por parte de los productores, se almacenan en las diferentes particiones de forma ordenada. Así, se tienen dos grupos de consumidores de tres consumidores cada uno. Como el número de particiones en el topic es el mismo que el número de consumidores de cada grupo, cada consumidor recibe los mensajes de una partición. De esta forma, a diferencia de MQTT, en el cual los consumidores reciben el mensaje que se almacena en el topic correspondiente, los consumidores de un mismo grupo de consumidores reciben diferentes mensajes. Además, como se puede observar, ambos grupos de consumidores reciben todos los mensajes almacenados en el topic.

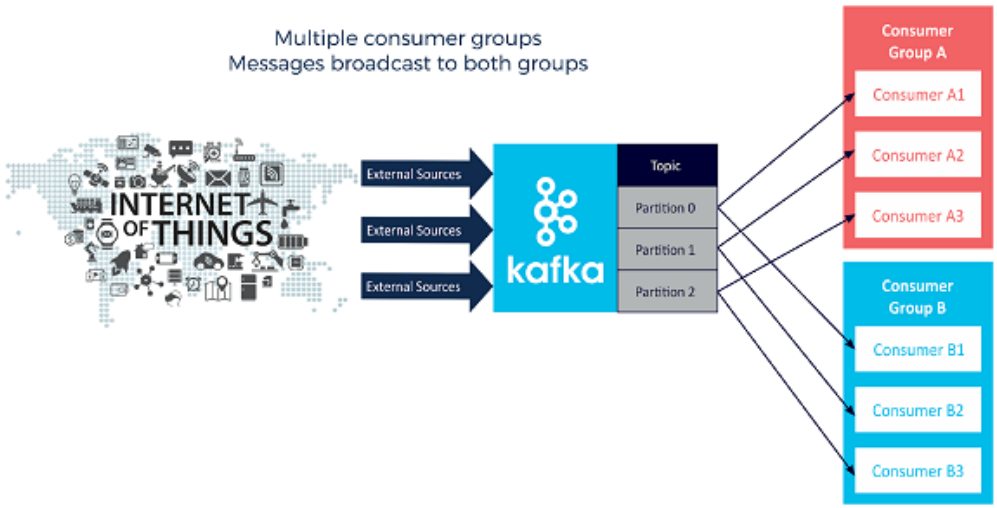


Figura 6.39: Particiones y grupos de consumidores

En el sistema desarrollado se ha explotado la funcionalidad de Kafka para realizar procesamiento de datos distribuido. A continuación, en la Figura 6.40 se muestran los diferentes componentes del sistema.

El sistema para realizar el procesamiento de los datos está formado por un broker que contiene varios topics: ”analisis” y ”dataset”. En topic de ”analisis” se usa para realizar la distribución de la carga cuando se realiza el análisis de los datos en tiempo real y el topic de ”dataset” se usa para crear un nuevo dataset. En este apartado se presenta la funcionalidad de análisis de datos en tiempo real, por lo que se usa el topic ”analisis”.

En primer lugar, se tiene un elemento productor que se encarga de incorporar los nuevos mensajes al topic ”analisis”. Este elemento en el sistema implementado se encarga de generar cada medio segundo un nuevo dato que corresponde al timestamp del instante actual. Este

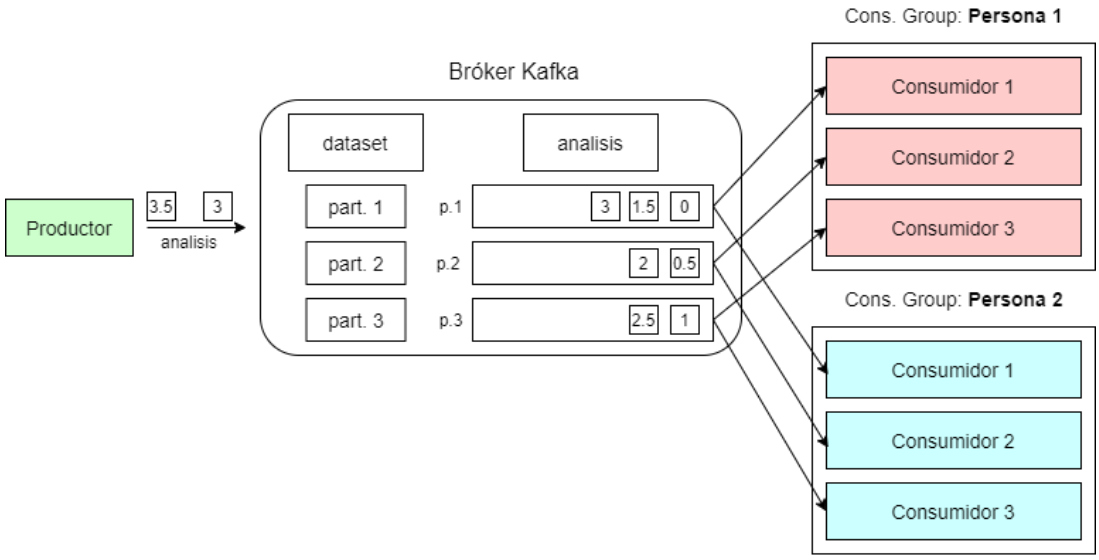


Figura 6.40: Estructura Kafka en el sistema actual

valor se inserta en el topic correspondiente para que un elemento consumidor procese los datos correspondientes a esa ventana de tiempo y clasifique la actividad correspondiente.

El topic "analisis" tiene tres particiones en las cuales se distribuyen los mensajes que llegan del productor, por lo que para optimizar la distribución de la carga son necesarios tres consumidores. De esta forma, se tienen varios grupos de consumidores con tres elementos consumidor cada uno. Cada grupo de consumidores corresponde al análisis de las actividades de un usuario de la vivienda, por lo que se tienen tantos grupos de consumidores como usuarios haya en la vivienda. Cada consumidor recibe una ventana de tiempo de 0.5ms para que clasifique la actividad correspondiente a esta.

De esta forma, un elemento productor va generando las sucesivas ventanas de análisis de datos y los elementos consumidor obtienen las actividades que se han realizado en cada ventana de tiempo. De esta forma, el análisis se realiza entre varios procesos de forma paralela y es posible su realización en tiempo real.

6.4.2 Librerías utilizadas

El núcleo del sistema de reconocimiento de actividades consiste en varios scripts programados en Python gracias a la enorme cantidad de librerías de las que dispone para integrar diferentes servicios y tecnologías. En esta sección se comentan las librerías que se han usado en el caso de la detección de actividades.

- *pymongo*: Proporciona las herramientas para conectar con la base de datos MongoDB así como para realizar las consultas CRUD sobre las colecciones de esta y las configuraciones de la misma.

- *kafka*: Permite establecer un script python como cliente suscriptor o productor de un broker kafka. Además, se pueden establecer algunas configuraciones sobre los topics del broker.
- *redis*: Proporciona las herramientas para interactuar con la base de datos en memoria RedisDB.
- *pandas*: Permite la manipulación y análisis de datos en Python. Proporciona las herramientas para estructurar y analizar datos en Python. Es una extensión de la tradicional librería numpy.
- *scikit-learn*: Ofrece al usuario modelos y técnicas de machine-learning en python.

6.4.3 Procesamiento de datos

El algoritmo de reconocimiento de actividades, como se ha comentado anteriormente, hace uso de varios scripts con roles diferentes. Debido a la gran cantidad de datos que hay que procesar, ha sido necesario el uso de un almacén de datos en memoria (RedisDB) y distribuir el procesamiento de las diferentes ventanas de tiempo entre varios procesos (Apache Kafka). Por lo tanto, el algoritmo está dividido en dos scripts con diferente funcionalidad. En esta sección se analiza el comportamiento de cada algoritmo y los aspectos más destacados de la implementación de los mismos.

6.4.3.1 Script inicialización del entorno

En primer lugar y de forma previa al procesamiento en tiempo real de los datos, es necesario iniciar los servicios de RedisDB y Apache Kafka. La inicialización de estos servicios en la máquina desplegada en la nube se realiza a través de un script que se ejecuta en el CMD. El script se muestra en el Código 6.10.

En este, en primer lugar se inicia el servidor de *zookeeper*, que consiste en un servicio centralizado cuyo objetivo es mantener la información sobre la configuración de Kafka y la gestión de los grupos de servicios. Además, se encarga de corregir errores y condiciones de carrera que no se pueden evitar (Foundation, 2020).

Una vez iniciado Zookeeper, se inicia el servicio de RedisDB y a continuación, se inicia Kafka.

Código 6.10: Script inicialización servicios

```
start cmd /k zkserver
start cmd /k C:\Tools\Redis-x64-3.2.100\redis-server.exe
start cmd /k C:\Tools\kafka_nuevo\kafka_2.12-2.4.1\bin\windows\kafka-server-start
    ↪ .bat C:\Tools\kafka_nuevo\kafka_2.12-2.4.1\config\server.properties
```

Cuando se ejecuta el script están todos los servicios iniciados para su uso por parte de los diferentes scripts de procesamiento y análisis de datos programados en Python. Sin embargo,

antes de que estos puedan ejecutarse, es necesario crear los topics correspondientes. En el Código 6.11 se muestra la creación de los topics *analizar* y *dataset*. En ambos, se crea un número de particiones igual a 3, correspondiendo una partición a cada instancia del script que se genere, en este caso, 3.

Código 6.11: Script creación topics

```
C:\Tools\kafka_nuevo\kafka_2.12-2.4.1\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --topic analizar --replication-factor 1 --partitions 3
C:\Tools\kafka_nuevo\kafka_2.12-2.4.1\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --topic dataset --replication-factor 1 --partitions 3
```

6.4.3.2 Script productor

Este script se encarga fundamentalmente de generar valores de tiempo cada 500 milisegundos e insertarlos en el broker kafka para que los diferentes procesos consuman estos valores y analicen la ventana correspondiente. De esta forma, el script se encarga del proceso que determina qué instantes de tiempo se analizan en cada momento, sin llegar a superar el tiempo actual. Así, para que todos los valores de los sensores estén almacenados correctamente en el momento del análisis de la ventana de tiempo correspondiente, se programa que este script genere ventanas de tiempo siempre menores al tiempo actual y teniendo en cuenta el tiempo que se tarda en obtener todos los datos procedentes de los sensores.

Por lo tanto, en primer lugar se realiza la configuración de la conexión al broker Kafka como sigue en el Código 6.12. En la conexión, se hace uso de la clase KafkaProducer indicando la dirección del broker Kafka.

Código 6.12: Creación productor Kafka

```
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],value_serializer=lambda x:dumps(x).encode('utf-8'))
```

A continuación, la funcionalidad principal del script consiste en ir generando ventanas de tiempo siempre 4 segundos anteriores al instante actual e ir insertándolas en el broker Kafka. Esta funcionalidad se recoge de manera simplificada en el Código 6.13.

Código 6.13: Generación de ventanas de tiempo de 500ms

```
ventana_base = 500
actual = datetime.timestamp(datetime.now())*1000
time.sleep(4)
while True:
ahora = datetime.timestamp(datetime.now())*1000.0
if ahora - actual < delay:
    time.sleep(int((delay - ahora + actual)/1000))
rellenar_cache(actual)
```

```
producer.send("analisis", value=actual)
actual = actual + ventana_base
```

Por lo tanto, la funcionalidad del script está clara en este punto. Sin embargo, dado que el script va generando las ventanas de tiempo cada 500 milisegundos y la carga de trabajo no es muy elevada, se pensó en que este **script pudiese realizar otras tareas para que posteriormente, los scripts que procesan las ventanas tengan menos carga de trabajo**. Como se puede observar en la Línea 8 del Código 6.13, se realiza una llamada a la función *rellenar_cache(actual)*. En esta función, aprovechando que se tiene el valor del timestamp que almacena el instante de la ventana de tiempo a analizar, se recogen de la base de datos los nuevos datos que se han producido en la ventana correspondiente y se almacenan en Redis. En consecuencia, los procesos que procesan las ventanas de tiempo, se encuentran con todos los datos a procesar en caché y no tienen que acceder a la base de datos, haciendo que el procesamiento sea muy eficiente.

En el Código 6.14 se muestra la obtención de los datos de un sensor de tipo inercial (acelerómetro, giroscopio), la obtención de sus métricas principales haciendo uso de pandas y su almacenamiento en Redis con la clave única que referencia al usuario, tipo de sensor e instante de tiempo.

Código 6.14: Cálculo de métricas y almacenamiento en caché

```
def rellenar_cache_inercial(sensor,persona, timestamp_inicio, timestamp_fin):
    datos = sensor_data.find({"persona" : persona, "tipo": sensor, "timestamp": { ↩
        ↪ '$gt': timestamp_inicio, '$lte': timestamp_fin}}, {'x': 1, "y": 1, 'z':↩
        ↪  1}).sort("timestamp", -1)
    datos_acc = [e for e in datos]
    df = pd.DataFrame(datos_acc)
    dicc = {}
    for c in coordenadas_acelerometros:
        if c in df:
            maximo = df[c].max()
            minimo = df[c].min()
            media = df[c].mean()
        else:
            maximo = 0.0
            minimo = 0.0
            media = 0.0
        aux = {"max": maximo, "min": minimo, "avg": media}
        dicc[c] = aux
        redis.set(persona+"_"sensor+"_"+str(timestamp_fin), json.dumps(dicc))
        return dicc
```

El script productor se ejecuta de forma indefinida, por lo que mientras que este introduzca nuevos datos en el broker kafka, los scripts que consumen estos datos siguen funcionando. En la siguiente sección se muestra el funcionamiento del script que consume los datos.

6.4.3.3 Script consumidor

El script consumidor se trata de la funcionalidad principal del sistema desarrollado. En este script, se realiza la agrupación de las pequeñas ventanas de tiempo de 500 milisegundos en ventanas de mayor tamaño, se calculan las métricas de estas ventanas y se comprueban los datos en los algoritmos de *machine learning*. A continuación, se desarrollan las diferentes secciones del script y se resalta la funcionalidad más importante en cuanto al procesamiento de los datos.

Inicialización En primer lugar, se realiza la inicialización de todas las estructuras de datos y variables necesarias para el funcionamiento del script. En el Código 6.15 se muestran las conexiones a los diferentes servicios, las variables que almacenan la información de los datos de entrada y las estructuras de datos que almacenan las ventanas de tiempo.

Código 6.15: Configuración script consumidor

```
#Conexiones a servicios de bases de datos y kafka
redis = redis.Redis(host='localhost', port=6379, db=0)
client = MongoClient("mongodb+srv://usuario:contraseña@tfg-obcxa.gcp.mongodb.net/bd?retryWrites=true")
consumer = KafkaConsumer(brokerID,bootstrap_servers=['localhost:9092'], auto_offset_reset='earliest',enable_auto_commit=True,group_id='my-group', value_deserializer=lambda x: loads(x.decode('utf-8')))

#Variables sensores
ventana_base = 500
metricas = ["max" , "min" , "avg"]
sensores = ["acc", "gyr","loc"]
binarios = ["bin"]
coordenadas_acelerometros = ["x","y","z"]
sensores_binarios = ["Cama", "Baño-Presencia", "Cocina-Presencia",...]

#Ventanas
ventanas_acelerometros = [0.5,1,3,5]
ventanas_localizacion = [1.5,5,10]
ventanas_binarios = [5,30]
```

Recepción de ventanas a procesar Una vez se han inicializado todas las variables y estructuras de datos necesarias para procesar los datos, se comienza la recepción de las ventanas de tiempo procedentes del script productor (Código 6.16). Esta recepción se realiza a través del bucle *for message in consummer*. De esta forma, *message* contiene el instante de la ventana a procesar. Cuando se tiene este valor, se llama a la función *calcular_actividad(inicio,fin,False)*, que se comenta en el siguiente apartado.

Código 6.16: Obtención de ventanas a procesar mediante Kafka

```
for message in consumer:
    timestamp_inicio = message.value
```

```
3    resultado = calcular_actividad(timestamp_inicio-ventana_base,timestamp_inicio,↩
         ↪ False)
```

Obtención de métricas La función *calcular_actividad(inicio,fin)* se trata del núcleo principal del script. En esta, se realiza todo el procesamiento asociado al análisis de las actividades. Así, tiene dos funcionalidades principales: en primer lugar, obtener las métricas almacenadas en la memoria RedisDB correspondientes a las ventanas de tiempo que se tienen en cuenta para la ventana actual y en segundo lugar, calcular las métricas generales y clasificar los datos en los algoritmos de *machine learning*.

En la primera parte de la función, como se ha comentado, se obtiene para cada tipo de sensor, los valores asociados a sus ventanas. Por lo tanto, para cada tipo de sensor se tiene la información que se muestra en la Figura 6.38. En la figura se aprecia la información almacenada para el sensor de giroscopio. En el caso de los sensores de aceleración y localización la información almacenada es la misma.

Sin embargo, en el caso de los sensores binarios, en lugar de almacenar para cada ventana los valores de máximo, mínimo y media, se guarda el estado de cada uno de ellos, que viene representado por la media del estado de estos sensores en la ventana de tiempo correspondiente. Por ejemplo, si el sensor *Cama* en los últimos tres segundos ha estado activo 1.5 segundos, el valor de la ventana es de 0.5. De esta forma, la ventana almacena el estado de los sensores a lo largo de esta, permitiendo tener un mayor conocimiento del estado de estos. La otra alternativa es tener en cuenta solamente el estado inicial o final de la ventana, pero este dato sería menos representativo de la interacción del usuario con este tipo de sensores.

Una vez se tienen los datos de las ventanas de 500 ms, es necesario procesar estos datos para obtener las métricas de las ventanas principales, diferentes en cada tipo de sensor. En la Figura 6.41 se muestran los datos que se obtendrían al calcular las métricas del sensor de giroscopio.

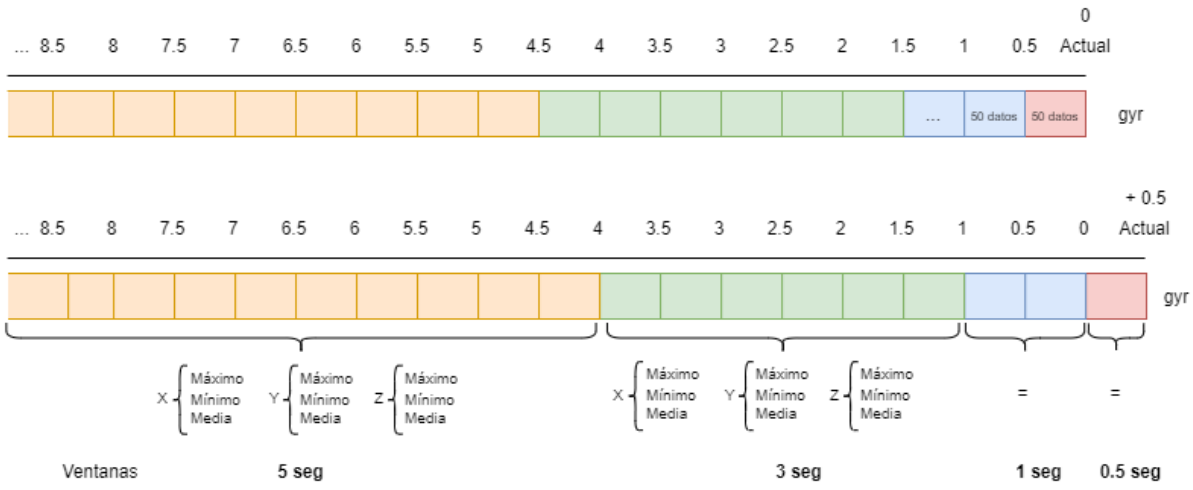


Figura 6.41: Métricas de ventanas principales

Por lo tanto, como resultado del procesamiento, los datos de las ventanas de los diferentes tipos de sensores, se insertan en un array unidimensional para procesarlos en los algoritmos de *machine learning*.

6.4.3.4 Inicialización de scripts

Para la ejecución de los diferentes scripts productor y consumidor, se deben iniciar en total 4 terminales. Esta tarea puede ser tediosa, por lo que se decide automatizarla incluyendo un script de inicialización. En el Código 6.17 se muestran los parámetros de entrada de los scripts y la inicialización de estos. Para iniciar los scripts se indican lo parámetros de la parte superior del script.

Código 6.17: Script inicialización

```
set topic=analizar
set /A numero_secuencia = 400
set /A numero_secuencia_analisis = 418
set tipo=analisis
set /A persona = 1

start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\optimizacion_ventanas\producer.py %topic%
start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\optimizacion_ventanas\consumer.py %tipo% %topic% %numero_secuencia% %numero_secuencia_analisis% %persona%
start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\optimizacion_ventanas\consumer.py %tipo% %topic% %numero_secuencia% %numero_secuencia_analisis% %persona%
start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\optimizacion_ventanas\consumer.py %tipo% %topic% %numero_secuencia% %numero_secuencia_analisis% %persona%
```

En primer lugar, se indica el topic correspondiente al análisis de las actividades, a continuación el número de secuencia (dataset) junto con el número del dataset de análisis, posteriormente el tipo de script (en este caso "analisis", el otro posible caso es "dataset" para generar el dataset). Finalmente, la persona de la cual se analizan los datos.

Una vez se han indicado los parámetros de los diferentes scripts, se produce la inicialización de estos, cada uno en un nuevo terminal (*start cmd /k*). En primer lugar, se inicia el script productor y posteriormente los 3 scripts consumidores.

6.4.4 Aspectos de machine learning

En la sección anterior se ha comentado del proceso de procesamiento de todos los datos que forman parte de las ventanas de tiempo para poder analizarlos en los diferentes algoritmos de

machine learning. En esta sección se destacan todos los aspectos del *machine learning* que se han incluido en el script consumidor.

Campos de los registros El objetivo de la clasificación en *machine learning* consiste en obtener el valor de un campo ”clase” de un registro concreto a partir de los registros con valores de ”clase” correctos. Por lo tanto, el valor de ”clase” quedará definido dependiendo de la similitud del registro a clasificar con el resto de registros almacenados y que nutren el algoritmo de *machine learning*.

De esta forma, en el sistema actual se ha creado toda la estructura de ventanas de tiempo para tener información acerca del desarrollo de las actividades que realiza el usuario. De esta forma se analiza la actividad que realiza en el estado de tiempo teniendo en cuenta cómo el usuario ha llegado a ella.

Siguiendo la estructura de sensores, los datos que aportan y las ventanas que se han definido en cada uno de ellos, los campos que tienen los registros del dataset son los que se muestran en la Figura 6.42. En la esta se hace uso solamente de seis sensores binarios. Dependiendo del dataset que se genere, pueden incluirse más o menos.

```
['AccX0.5Max', 'AccX0.5Min', 'AccX0.5Avg', 'AccX1Max', 'AccX1Min', 'AccX1Avg', 'AccX3Max', 'AccX3Min', 'AccX3Avg'
cY0.5Max', 'AccY0.5Min', 'AccY0.5Avg', 'AccY1Max', 'AccY1Min', 'AccY1Avg', 'AccY3Max', 'AccY3Min', 'AccY3Avg', 'A
5Max', 'AccZ0.5Min', 'AccZ0.5Avg', 'AccZ1Max', 'AccZ1Min', 'AccZ1Avg', 'AccZ3Max', 'AccZ3Min', 'AccZ3Avg', 'AccZ5
', 'GyrX0.5Min', 'GyrX0.5Avg', 'GyrX1Max', 'GyrX1Min', 'GyrX1Avg', 'GyrX3Max', 'GyrX3Min', 'GyrX3Avg', 'GyrX5Max'
GyrY0.5Min', 'GyrY0.5Avg', 'GyrY1Max', 'GyrY1Min', 'GyrY1Avg', 'GyrY3Max', 'GyrY3Min', 'GyrY3Avg', 'GyrY5Max', 'G
0.5Min', 'GyrZ0.5Avg', 'GyrZ1Max', 'GyrZ1Min', 'GyrZ1Avg', 'GyrZ3Max', 'GyrZ3Min', 'GyrZ3Avg', 'GyrZ5Max', 'GyrZ5
in', 'LocX1.5Avg', 'LocX5Max', 'LocX5Min', 'LocX5Avg', 'LocX10Max', 'LocX10Min', 'LocX10Avg', 'LocY1.5Max', 'LocY
Min', 'LocY5Avg', 'LocY10Max', 'LocY10Min', 'LocY10Avg', 'LocZ1.5Max', 'LocZ1.5Min', 'LocZ1.5Avg', 'LocZ5Max', 'L
5Max', 'AccZ0.5Min', 'AccZ0.5Avg', 'AccZ1Max', 'AccZ1Min', 'AccZ1Avg', 'AccZ3Max', 'AccZ3Min', 'AccZ3Avg', 'AccZ5
', 'GyrX0.5Min', 'GyrX0.5Avg', 'GyrX1Max', 'GyrX1Min', 'GyrX1Avg', 'GyrX3Max', 'GyrX3Min', 'GyrX3Avg', 'GyrX5Max'
GyrY0.5Min', 'GyrY0.5Avg', 'GyrY1Max', 'GyrY1Min', 'GyrY1Avg', 'GyrY3Max', 'GyrY3Min', 'GyrY3Avg', 'GyrY5Max', 'G
0.5Min', 'GyrZ0.5Avg', 'GyrZ1Max', 'GyrZ1Min', 'GyrZ1Avg', 'GyrZ3Max', 'GyrZ3Min', 'GyrZ3Avg', 'GyrZ5Max', 'GyrZ5
in', 'LocX1.5Avg', 'LocX5Max', 'LocX5Min', 'LocX5Avg', 'LocX10Max', 'LocX10Min', 'LocX10Avg', 'LocY1.5Max', 'LocY
Min', 'LocY5Avg', 'LocY10Max', 'LocY10Min', 'LocY10Avg', 'LocZ1.5Max', 'LocZ1.5Min', 'LocZ1.5Avg', 'LocZ5Max', 'L
0Min', 'LocZ10Avg', 'BinCama5', 'BinCama30', 'BinBaño-presencia5', 'BinBaño-presencia30', 'BinBaño-wc5', 'BinBaño
0', 'BinCocina-presencia5', 'BinCocina-presencia30', 'BinSofa5', 'BinSofa30']
```

Figura 6.42: Campos de los registros del dataset

Por lo tanto, el primer paso a incluir cuando se implementa un algoritmo de *machine learning* consiste en obtener los campos de los datos que se van a comparar. Para hacer esto, haciendo uso de las variables definidas en el Código 6.15 se almacenan estos campos.

Obtención de los registros del dataset Una vez se conocen los campos de los registros que forman parte del dataset usado para reconocer las actividades, es necesario obtenerlos parar poder iniciar el modelo. En este caso, se almacenan en la base de datos del sistema, concretamente en la colección *registros_ml*. En el Código 6.18 se puede observar la consulta para obtener los datos. Como en la base de datos pueden existir varios datasets, se obtienen los datos correspondientes al dataset que determina el usuario.

Código 6.18: Obtención de registros del dataset

```
datos = registros_ml.find({"numero_secuencia" : numero_secuencia})
```

Inicialización de los modelos El siguiente paso consiste en crear los diferentes modelos con la configuración que se estima conveniente en cada caso. En el Código 6.19 se pueden apreciar la creación de varios modelos. Para realizar la instanciación de cada modelo, se crea una instancia de la clase correspondiente a cada modelo, estableciendo como parámetros la configuración adecuada en cada caso. En el sistema, se inicializan varios modelos KNeighborsClassifier con un valor de número de vecinos (k) igual a 10 y a $\sqrt{longitud\_dataset}$. Una vez creados, se inicializan con los registros de la base de datos haciendo uso de la función *fit(campos,clase)*. En esta función se relacionan los campos de los diferentes registros con la clase correspondiente (en este caso, la clase es la actividad realizada por el usuario).

Código 6.19: Inicialización modelo

```
model1 = KNeighborsClassifier(n_neighbors=int(np.sqrt(len(data)))).fit(datos,y)
model2 = RandomForestClassifier(n_estimators=100).fit(datos,y)
model3 = KNeighborsClassifier(n_neighbors=int(10)).fit(datos,y)
model4 = SVC().fit(datos,y)
```

Normalización En el caso de la normalización o escalado de los datos, se hace uso de las clases correspondientes en cada caso para cambiar los valores de los campos del dataset. En el Código 6.20 se puede observar el procedimiento para escalar los datos; en primer lugar se genera una instancia de la clase de escalado (MinMaxScaler o StandardScaler) y a continuación se realiza la sobreescritura de los datos de dataset con los datos escalados.

Código 6.20: Escalado de datos del dataset

```
scalerMinMax = MinMaxScaler()
datos[columnasDatoAClasificar] = scalerMinMax.fit_transform(datos)

scalerStd = StandardScaler()
datos[columnasDatoAClasificar] = scalerStd.fit_transform(datos)
```

Selección de atributos En el caso de la selección de atributos, se han probado dos técnicas para reducir el número de atributos de la clasificación (Código 6.21). En primer lugar, se hace uso de la clase VarianceThresold para eliminar atributos con varianza 0 o 1. En segundo lugar, se hace uso de un meta-transformador LinearSVC, que se corresponde con el modelo *SVM* (máquinas de vectores de soporte) con una función de *kernel* linear, para realizar la selección de atributos.

Código 6.21: Selección de atributos

```
selVarianceAt = VarianceThreshold()
datos = selVarianceAt.fit_transform(datos)
```

```
lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(datos, y)
model_lsvc = SelectFromModel(lsvc, prefit=True)
datos = model_lsvc.transform(datos)
```

Clasificación de registros El último paso en el uso de *machine learning* consiste en la clasificación del registro. En el Código 6.22 se muestra la clasificación de un registro en los diferentes modelos que se tratan en el sistema. Para realizar esto, en primer lugar, se almacena el registro de un DataFrame de Pandas para cerciorarse que tiene todas los campos necesarios. Posteriormente, se realiza la predicción del registro correspondiente en cada modelo y se almacena el resultado. En el caso de tener que escalar los datos porque el clasificador tiene los datos escalados, se hace uso de la función *transform* correspondiente al escalador usado previamente. En este caso, se hace uso de un escalado de máximo/mínimo.

Código 6.22: Clasificación de registro

```
data = pd.DataFrame({"datos" : registro} , columns=columnasDatoAClasificar)
resultado = {}
for model in diccionario_modelos:
    preds = diccionario_modelos[model].predict([scalerMinMax.transform([data]).↩
        ↪ tolist()[0]])
    act = int(preds[0])
    resultado[model] = act
```

Inserción en la base de datos Una vez clasificado el registro que corresponde a la ventana de tiempo recibida, se incorpora en la colección *actividades_obtenidas_ml* de la base de datos.

Código 6.23: Inserción en base de datos actividad obtenida por machine learning

```

for elemento in resultado:
    actividades_obtenidas_ml.insert_one({
        "dato" :resultado[elemento][1] ,
        "numero_secuencia" : numero_secuencia_analisis,
        "numero_secuencia_dataset" : numero_secuencia,
        "actividad" : resultado[elemento][0] ,
        "inicio" :timestamp_inicio ,
        "fin" : timestamp_inicio + ventana_base,
        "algoritmo":elemento })
```

6.5 Generación del dataset

Como se ha comentado en secciones anteriores, en el campo de *machine learning* los modelos son inicializados con un conjunto de datos correctamente clasificados. De esta forma, cuando se

quiere clasificar un nuevo registro, los modelos comparan este nuevo registro con los datos con los que fueron inicializados. En este caso, al tratarse de modelos de *machine learning* supervisados, los datos correctamente clasificados son definidos por el usuario. Así, antes de poder analizar en tiempo real los datos de los sensores es necesario crear un conjunto de datos correctamente clasificados, que es lo que se conoce como *dataset*.

6.5.1 Metodología

El dataset en este caso es el conjunto de registros que recogen las actividades que realiza el usuario en la *smart home*. Por lo tanto, para poder crear un dataset correcto en el sistema actual, hay que seguir los siguientes pasos.

1. El usuario porta consigo un *smartwatch* Polar junto con un *tag* Decawave. Además, los sensores de la *smart home* se encuentran activos.

2. El usuario abre el sistema web y accede a la ventana de "Insertar dataset". El usuario rellena los diferentes campos del dataset, junto con las actividades que se van a detectar posteriormente.

3. El usuario abre el sistema web y accede a la ventana de "Insertar actividades".

4. Cuando el usuario se encuentra preparado para comenzar a "grabar" el dataset, comienza a realizar las actividades que quiere recoger en este. Así, antes de comenzar a realizar la actividad, recoge el inicio de esta y cuando acaba de realizarla, recoge el fin. El usuario puede recoger tantas actividades como desee, de la duración que estime oportuna. Sin embargo, debe tener en cuenta el orden "normal" en el que se realizan las actividades, ya que el algoritmo da mucha importancia a este factor. Cuando el usuario no desea registrar más actividades, finaliza el proceso de registro de actividades.

5. Una vez el usuario ha recogido las actividades que ha realizado en la *smart home*, se lanza el algoritmo de generación del dataset, que se encuentra en una máquina virtual en la nube. Cuando son procesados los datos y creado el dataset, el usuario puede comenzar a monitorizar las actividades que hace en base a las actividades recogidas previamente en la generación del dataset.

6.5.2 Interfaz web inserción de dataset

En el sistema desarrollado, cada dataset debe ser registrado en el sistema previamente a ser completado con registros de entrenamiento. Esta tarea debe ser realizada por el usuario antes de comenzar a grabar la secuencia de datos de entrenamiento. Se hace con el objetivo de almacenar información descriptiva del dataset como el nombre, una pequeña descripción y el conjunto de actividades que se detecta en él.

Para realizar esta funcionalidad, se ha realizado en la página web desplegada como resultado del Trabajo Fin de Grado asociado a este TFM.La interfaz web para PC de la ventana de inserción del dataset es la que se muestra en la Figura 6.43. En el Anexo 10 se muestra la versión para móvil.

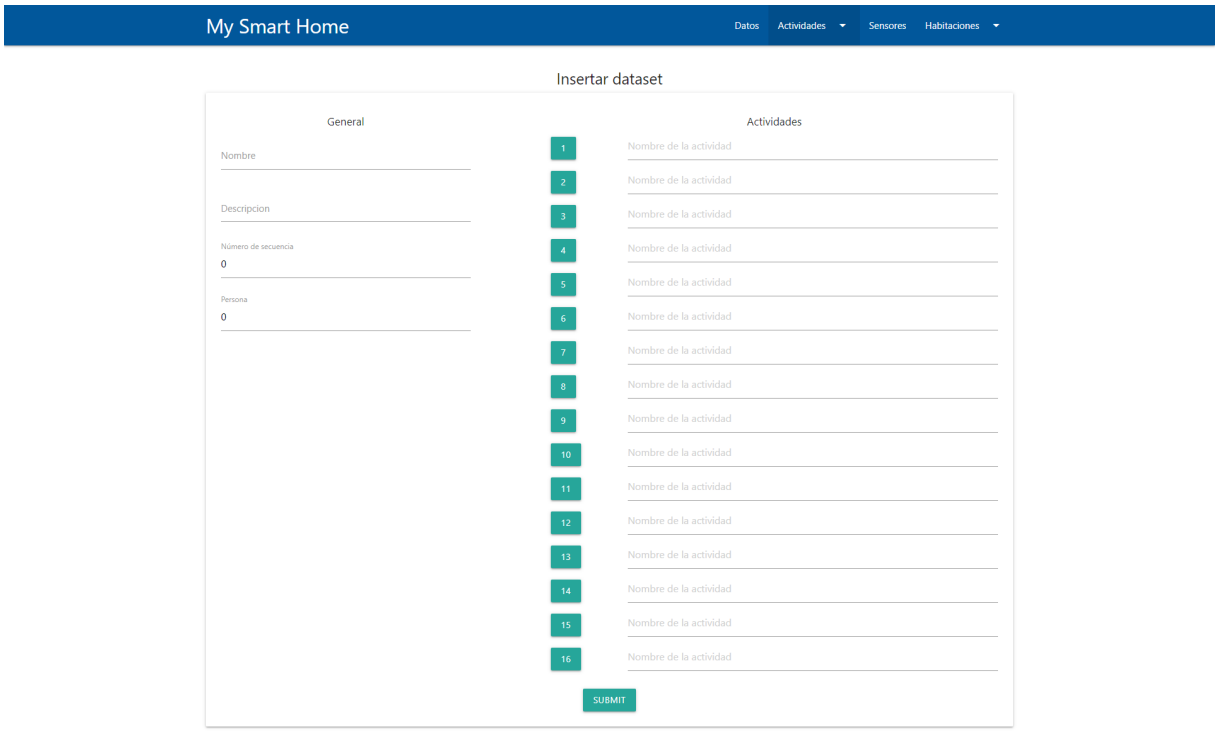


Figura 6.43: Sistema web - Registro nuevo dataset

Como se puede observar, el usuario puede registrar las actividades que posteriormente se van a detectar para que en la ventana de "Registro de actividad", aparezcan y pueda registrarlas a medida que las realiza. Cada actividad está asociada a un número para identificarla de forma única.

6.5.3 Interfaz web registro de actividades

Cuando se trabaja con algoritmos de *machine learning*, se parte de un conjunto de datos correctamente clasificados para producir nuevas clasificaciones en base a estos. En este caso, a partir de un conjunto de datos generados por los sensores mencionados anteriormente, se debe establecer la actividad real que realiza el usuario en cada momento. De esta forma, se logra establecer la relación entre los datos recogidos y la actividad realizada.

Por lo tanto, es necesario crear una interfaz en la cual se indica la actividad que realiza el usuario en cada momento de la generación del dataset y almacenar esta información para posteriormente procesarla y generar el conjunto de datos correspondiente.

Para ello, el usuario hace uso de la ventana "Registrar actividad". En esta ventana de forma predeterminada muestra el aspecto de la figura 6.44. En esta, el usuario establece en primer lugar un número que indica el dataset al que corresponde la actividad que se realiza, ya sea para crear un nuevo dataset o para analizar la certeza de los algoritmos. En segundo lugar, establece el número de secuencia que almacena el registro de actividades. Si el número de secuencia es el mismo que el número establecido en el campo anterior, se añaden las actividades actuales al dataset. En caso de ser diferente, se realiza un análisis en tiempo real de las actividades.

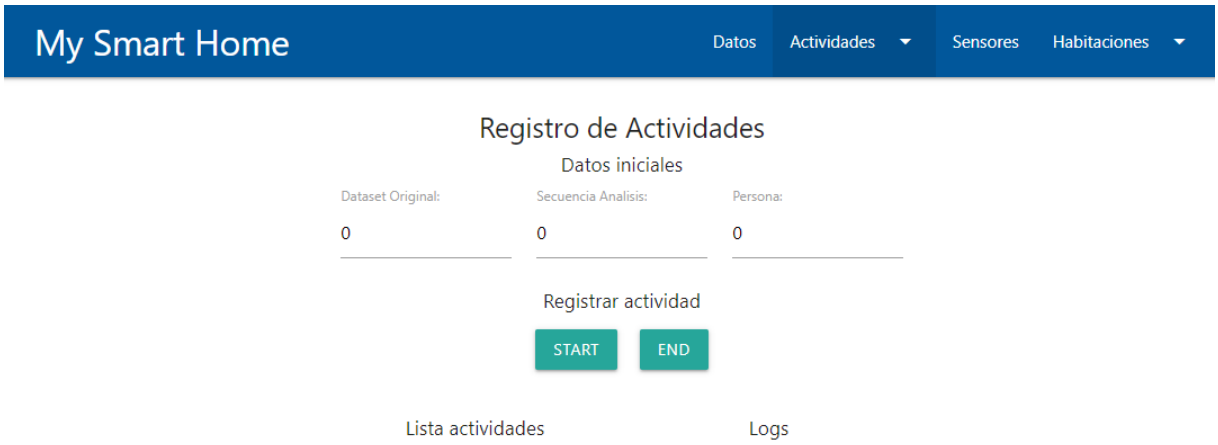


Figura 6.44: Sistema web - Registro de actividad inicial

A continuación, se incluyen dos botones que indican el inicio o el fin de la secuencia de actividades que recoge el usuario. Para comenzar a recoger los datos, el usuario pulsa sobre el botón "Start" y cuando desea finalizar, hace lo mismo sobre el botón "End".

Una vez que el usuario selecciona el número de secuencia del dataset que creó anteriormente, se despliegan las actividades que se registraron en dicho dataset. Por lo tanto, una vez aparecen las actividades, el usuario puede comenzar con el registro de su actividad.

En cuanto a la programación de dicha funcionalidad, cada vez que se pulsa un botón, se envía una petición POST a la API recogiendo los datos de la interfaz y la API envia su contenido a la base de datos. La colección que almacena estos datos se es *registros_actividad*, definida en la Sección 6.2.

Esta funcionalidad se ha incorporado en el sistema web *UAL Smart Home*. De esta forma, el usuario puede recoger las actividades que realiza desde cualquier dispositivo con acceso a Internet, sin importar que se encuentre o no en el entorno local.

En la Figura 6.45 se puede observar una secuencia de actividades realizada por el usuario y registrada en el dataset para que se incluyan registros de entrenamiento en el algoritmo.

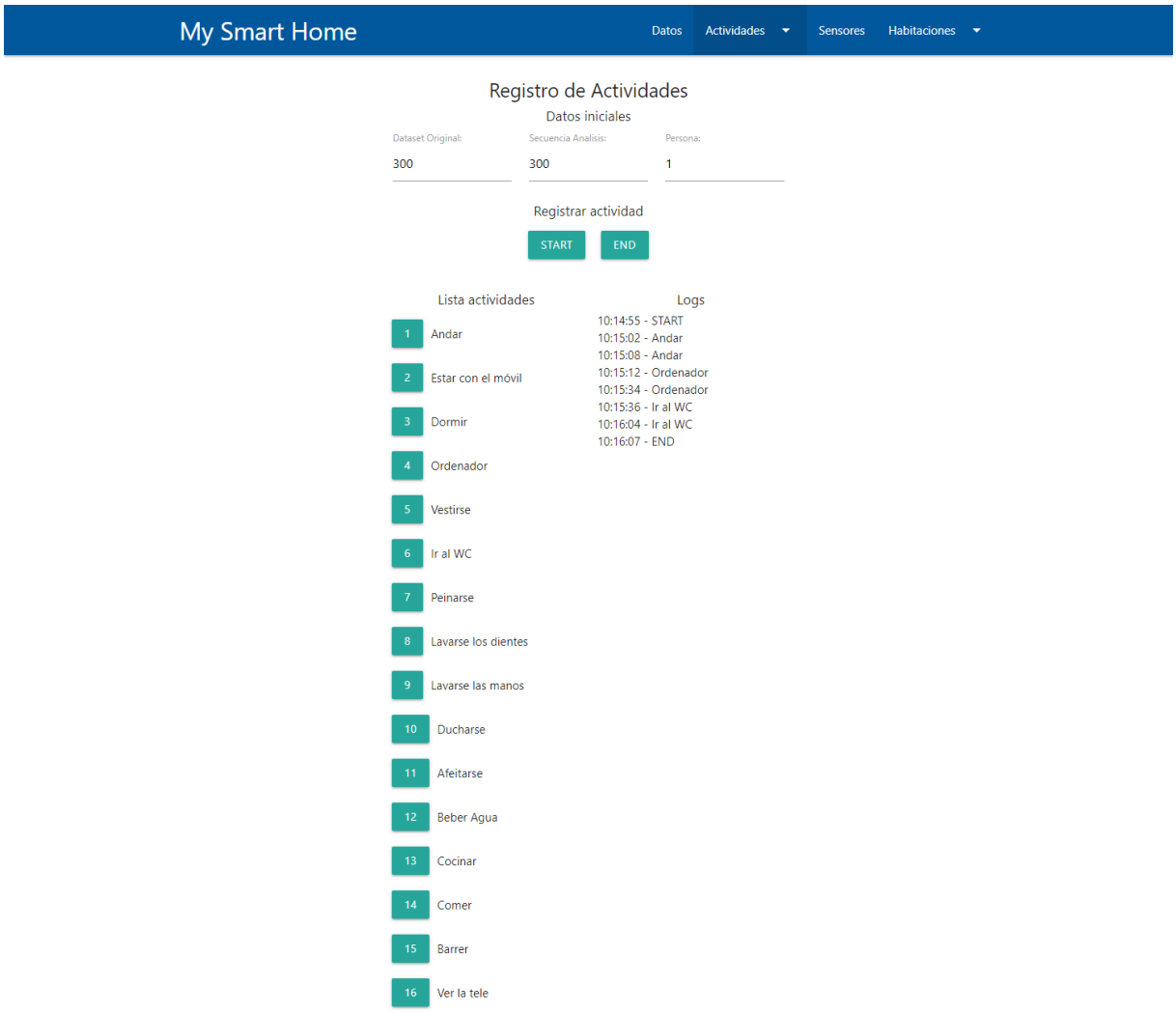


Figura 6.45: Sistema web - Registro de Actividad

6.5.4 Scripts generación del dataset

Una vez el usuario finaliza el registro de las actividades que realiza en la *smart home*, se tiene un registro del comienzo y fin de estas como se muestra en la Tabla 6.3.

Persona	Tiempo	Actividad
1	10:14:55	START
1	10:15:02	Andar
1	10:15:08	Andar
1	10:15:12	Ordenador
1	10:15:34	Ordenador
1	10:15:36	Ir al WC

Persona	Tiempo	Actividad
1	10:16:04	Ir al WC
1	10:16:07	END

Tabla 6.3: Registros de actividades realizadas

Por lo tanto, la tarea principal del algoritmo de generación del dataset consiste en obtener los datos correspondientes a los sensores para cada ventana de medio segundo y asociar la actividad correspondiente. Para obtener los datos de las diferentes ventanas de tiempo, se realiza el mismo proceso que se realizó en el análisis de los datos en tiempo real. De esta forma, se tienen dos scripts que hacen uso de Apache Kafka y RedisDB para realizar las tareas correspondientes a la generación del dataset. A continuación se comenta la funcionalidad de cada uno de ellos.

6.5.4.1 Script productor

El script productor tiene como principal función procesar los datos de la Tabla 6.3 y dividir cada actividad registrada en ventanas pequeñas de 500ms. Una vez se tiene el instante de inicio y fin de la ventana y la actividad asociada, se incorpora el registro en el broker de kafka. Concretamente, se inserta en el topic "dataset".

En la Figura 6.46 se muestran una secuencia de actividades registradas por el usuario en la smarthome y el fraccionamiento de estas que realiza el script productor. De esta forma, cada ventana de 500ms se envía al broker kafka para que el script consumidor recoja la información asociada a esta ventana e inserte un nuevo registro de dataset en la base de datos.

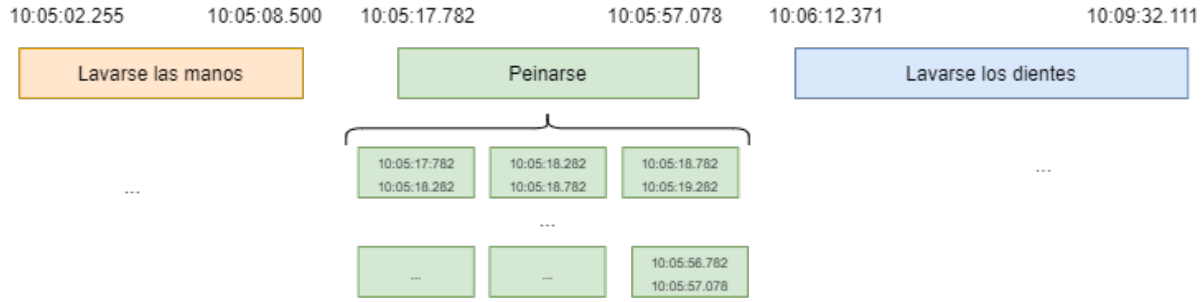


Figura 6.46: Métricas de ventanas principales

Al igual que el script productor del análisis de actividades, en este caso para cada ventana de tiempo de 500ms, se recogen los datos de los sensores incluidos en la base de datos y los almacena en la memoria RedisDB. De esta forma, el script consumidor tendrá los datos de la ventana que recibe en memoria caché y no tendrá que acceder a la base de datos, incrementando la velocidad con la que se genera el dataset.

6.5.4.2 Script consumidor

El script consumidor está suscrito al topic ”dataset” en el cual el script productor introduce los registros. El funcionamiento general es muy parecido al que se describe en la Sección 6.4.3.3, ya que se trata del mismo script pero con ligeras variaciones. En este caso, la recepción de datos del broker Kafka es la misma, pero en lugar de llamar a la función *calcular_actividad(inicio,fin,False)*, se realiza la llamada *calcular_actividad(inicio,fin,True)*.

Al realizar la llamada a esta función, se obtienen los registros de la caché y se accede a la base de datos en caso de que algún registro no esté presente aún en la caché de RedisDB. Una vez dispone de todos los registros de las ventanas de tiempo correspondientes, se realiza el procesamiento de estas englobando las ventanas de 500ms en las ventanas más grandes correspondientes a cada tipo de sensor. Cuando se tienen las ventanas grandes de cada tipo de sensor, se calculan las métricas correspondientes y los resultados se añaden a una estructura de datos de tipo *array* unidimensional.

Una vez todos los datos recogidos en el *array*, se insertan en la base de datos de registros de entrenamiento haciendo uso de la sentencia que se puede visualizar en el Código 6.24.

Código 6.24: Inserción en base de datos dato de entrenamiento

```
registros_ml.insert_one({
    "persona" : persona,
    "numero_secuencia" : message.value["numero_secuencia"],
    "timestamp" : timestamp_inicio ,
    "datos" : dato ,
    "actividad" : actividad,
    "columnas" : message.value["columnas"]
})
```

6.5.4.3 Inicialización de scripts

Para la ejecución de los diferentes scripts productor y consumidor, se deben iniciar en total 4 terminales, al igual que en el caso del análisis de las secuencias en tiempo real. Esta tarea puede ser tediosa, por lo que se decide automatizarla incluyendo un script de inicialización. En el Código 6.25 se muestran los parámetros de entrada de los scripts y la inicialización de estos. Para iniciar los scripts se indican lo parámetros de la parte superior del script.

Código 6.25: Script inicialización generación del dataset

```
set topic=dataset
set /A numero_secuencia = 400
set tipo=dataset

start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\↩
    ↪ creacion_dataset\producer.py %numero_secuencia% %topic%
start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\↩
```

```
    ↪ optimizacion_ventanas\consumer.py %tipo% %topic% %numero_secuencia% %↩
    ↪ numero_secuencia_analisis% %persona%
start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\↩
    ↪ optimizacion_ventanas\consumer.py %tipo% %topic% %numero_secuencia% %↩
    ↪ numero_secuencia_analisis% %persona%
start cmd /k python -W ignore C:\Users\Administrator\Documents\Repos\TFM\↩
    ↪ optimizacion_ventanas\consumer.py %tipo% %topic% %numero_secuencia% %↩
    ↪ numero_secuencia_analisis% %persona%
```

En primer lugar, se indica el topic correspondiente a la generación del dataset; a continuación, el número de secuencia (dataset) y finalmente, el tipo de script (en este caso "dataset" ya que se genera un nuevo dataset).

Una vez se han indicado los parámetros de los diferentes scripts, se produce la inicialización de estos, cada uno en un nuevo terminal (*start cmd /k*). En primer lugar, se inicia el script productor y posteriormente los 3 scripts consumidores.

6.6 Despliegue

Los diferentes scripts de procesamiento de datos y obtención de actividades mediante *machine learning* se ejecutan en una máquina virtual alojada en OpenStack. Las características de la máquina virtual están definidas en la Tabla 6.4.

Elemento	Valor
Sistema Operativo	Windows Server 2012
Memoria RAM	16 GB
Virtual CPU	8
Disco	160 GB

Tabla 6.4: Características Máquina Virtual usada.

De esta forma, al ejecutarse los scripts de procesamiento de datos y reconocimiento de actividades en una máquina virtual alojada en la nube, se permite la ejecución constante de los mismos.

Por otra parte, dado que el sistema web tiene que ser accesible desde la red, se usa Heroku para hacer el despliegue del mismo. Heroku se trata de un PaaS en el cual se despliegan aplicaciones de forma rápida y sencilla, proporcionando el repositorio en el que se encuentra la misma y especificando la tecnología que usa.

Para realizar el despliegue de la aplicación, se realiza de forma automática (Figura 6.47) cada vez que se realiza una acción de *push*[3] sobre el repositorio, por lo que no hay que realizar un despliegue manual de la misma, y los cambios se aprecian rápidamente en la aplicación desplegada.

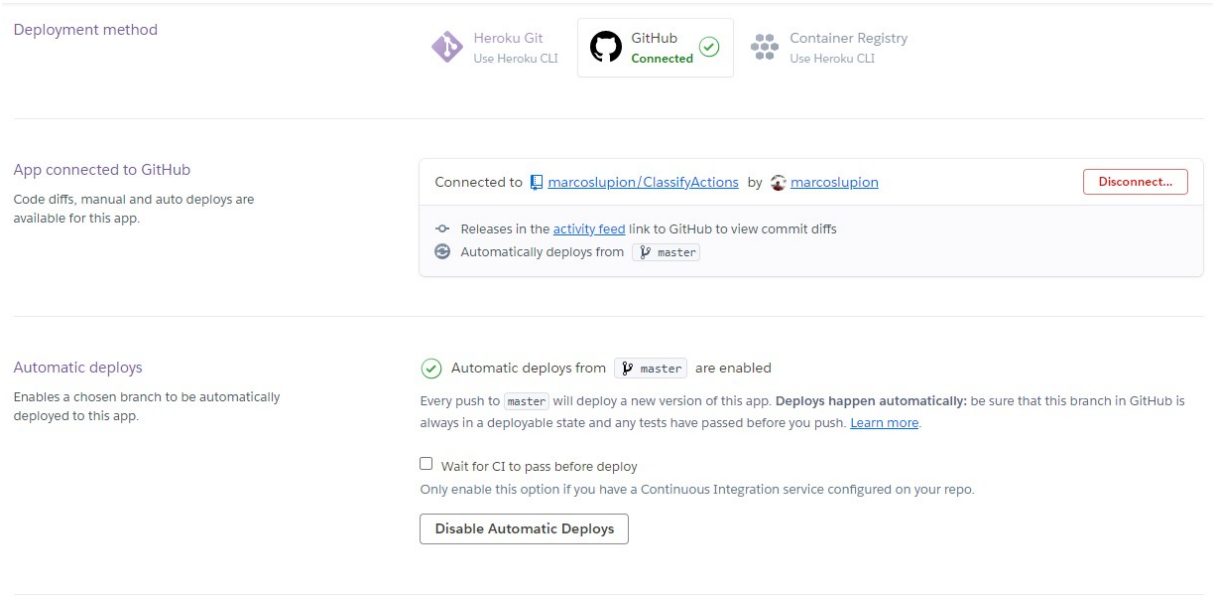


Figura 6.47: Despliegue automático

El sitio web se puede encontrar en la dirección `https://marcoslupiontfg.herokuapp.com`

[3]Corresponde con la acción de subir los cambios al repositorio

7 Resultados

7.1 Algoritmo de reconocimiento de actividades

El sistema que se ha desarrollado permite el procesamiento de los datos generados por los diferentes tipos de sensores de una *smart home* en tiempo real. Además, a partir de estos datos se extraen las actividades que realizan los usuarios de la vivienda. Para ello, como se ha desarrollado en secciones anteriores, es necesario incluir técnicas de *machine learning* basadas en el aprendizaje a partir de datos que se proporcionan de actividades que realiza el usuario en la vivienda. En esta sección se recogen los resultados de aplicar las técnicas de *machine learning* al problema actual.

7.1.1 Actividades a reconocer - Dataset

En primer lugar, los algoritmos de *machine learning* que se utilizan en el proyecto se tratan de algoritmos de aprendizaje supervisado. Estos algoritmos se basan en datos proporcionados por el usuario obtenidos en base a la experiencia u observación para que el algoritmo obtenga relaciones y patrones entre ellos y sea capaz de clasificar nuevos datos en base a estos.

En este caso, los datos que se proporcionan a los algoritmos de *machine learning* han sido creados por un usuario realizando las diferentes actividades a reconocer en la vivienda y etiquetando correctamente estos datos. Las actividades que se han definido en este caso son las siguientes:

1. Andar
2. Estar con el móvil
3. Dormir
4. Ordenador
5. Vestirse
6. Ir al WC
7. Vestirse

8. Peinarse
9. Lavarse los dientes
10. Lavarse las manos
11. Ducharse
12. Afeitarse
13. Beber agua
14. Comer
15. Barrer
16. Ver la televisión

Estas actividades son indicadoras del estado de una persona en la vivienda. Así, en el caso de personas mayores, puede ser indicativo que la persona tiene alguna dolencia o lesión cuando por ejemplo deja de barrer, tarda más de lo normal en ducharse o deja de comer. Por otra parte, se permite controlar la vida sedentaria de los usuarios de la vivienda detectando cuando una persona se pasa la mayor parte durmiendo, con el móvil o con el ordenador. De igual forma, se puede controlar la alimentación de los usuarios de la vivienda controlando la cantidad de veces que se come, bebe agua y va al cuarto de baño a lo largo del día.

En definitiva, las actividades que se han incluido permiten realizar un estudio de la vida de las personas que viven en la *smart home* y detectar posibles deficiencias en estas para corregirlas y mejorar la vida de las personas que residen en esta.

Así, para detectar correctamente las actividades, se incluye una gran cantidad de registros correctamente clasificados de estas actividades. Así, las actividades con mayor facilidad para reconocer requieren menos registros en el dataset ya que siguen patrones fácilmente reconocibles y repetitivos. Sin embargo, algunas de las actividades pueden ser muy similares entre sí y por tanto, más difícilmente diferenciables, siendo necesarios más registros en el dataset de ellas.

En total, se han incluido 8.480 registros de actividades en el dataset. En la Figura 7.1 se presenta el porcentaje de registros que contiene cada actividad dentro del dataset. El gráfico se ha programado haciendo uso de la librería *matplotlib* de *Python*.

Como se puede observar, las actividades con mayor número de registros son *Estar con el móvil*, *Lavarse las manos* y *Afeitarse*. En el caso de la actividad *Estar con el móvil*, ha sido necesario incluir una gran cantidad de registros porque la actividad se puede reconocer en todas las habitaciones de la *smart home* y por tanto, los datos de localización varían cuando esta se realiza. En el caso de *Lavarse las manos*, al realizarse en el cuarto de baño al igual que *Afeitarse* y *Lavarse los dientes*, es necesario incluir un gran conjunto de datos de entrenamiento de esta para que se diferencie bien del resto. En el caso de *Afeitarse* también ha sido necesario incluir

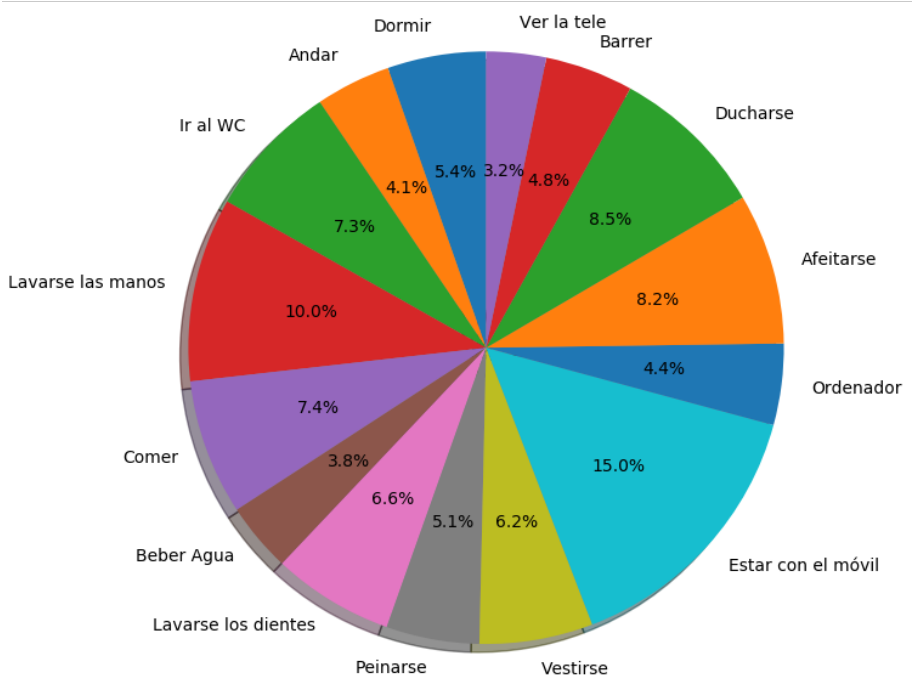


Figura 7.1: Actividades del dataset

una gran cantidad de registros porque puede ser confundido con el hecho de tocarse la cara sin querer, por lo que hay que definir bien el movimiento de la acción de afeitar.

Por otra parte, las actividades mejor definidas son las de *Peinarse*, *Ver la televisión*, *Ordenador* y *Dormir*. En este caso, se tratan de actividades que no dan lugar a confusión ya que en el caso de peinarse forma de un movimiento muy bien definido, al igual que ver las tres restantes, ya que además de realizarse siempre en el mismo lugar, los valores de aceleración y giroscopio son muy pequeños a diferencia del resto de actividades.

En definitiva, la proporción de registros de cada actividad está balanceada en su mayor parte, facilitando el aprendizaje de las actividades por parte de los modelos de *machine learning*.

7.1.2 Configuración de modelos

Una vez que se obtuvo el dataset con las actividades correctamente clasificadas, se diseñaron diferentes modelos que se estimaron convenientes para el reconocimiento de actividades. Los modelos, proporcionados por la librería *scikit-learn* de *Python* son los siguientes:

- kNN (k-vecinos más próximos). k=10

- kNN (k-vecinos más próximos). k=$\sqrt{longitud\_dataset}$
- *SVM* (Máquinas de vector de soporte) Configuración por defecto. Función de kernel = *rbf* (Radial Basis Function).
- Random Forest Classifier (Bosque de árboles de clasificación). Número de árboles = 100.

Además, cada modelo se ejecuta con diferentes configuraciones de selección, normalización y escalado de atributos del vector de características. Las configuraciones son las siguientes:

- **def**: Se corresponde con la configuración por defecto. El modelo obtiene los datos puros del vector de características que se genera en el procesamiento de los datos. En este caso, los datos tendrán valores muy dispares entre sí, incluyendo diferentes rangos y algunos modelos como *SVM* y *kNN* son muy sensibles a los vectores con datos en distintos rangos. Además, no se produce selección de atributos, pudiendo contener atributos que realmente no sean necesarios para la clasificación de nuevos datos y que lo único que hagan sea sobrerepresentar el modelo. La sobrerepresentación (overfit) consiste en que el modelo se adapta completamente a los datos del dataset, pudiendo no generalizar bien y por tanto no clasificar correctamente nuevos datos.
- **escalado**: El vector de características se escala en el rango 0-1 para que todos los atributos tengan la misma importancia al realizar la clasificación. La clase que se usa es *MinMaxScaler*.
- **normalizado**: En este caso, se normalizan los datos del vector de características, teniendo media y desviación típica 1. De esta forma, se tiene una distribución normal y generalmente los modelos como el *SVM* se comportan bastante bien con esta configuración. La clase que se usa se trata de *StandardScaler*.
- **ats_var**: Se realiza la selección de atributos seleccionando aquellos atributos que tengan una varianza mayor de la establecida. En este caso, se hace uso de la clase *VarianceThreshold* con una varianza de 0.2.
- **ats_model**: Se realiza selección de atributos escogiendo aquellos atributos que según el modelo *SVM* son más importantes para la clasificación. Se hace uso de la clase *LinearSVC* para determinar los atributos con mayor importancia y la clase *SelectFromModel* para generar el vector con los atributos resultantes.
- **esc_acts_var**: Se realiza un escalado de los atributos y posteriormente una selección de estos mediante el uso de la varianza.
- **norm_ats_model**: Se normalizan los atributos y posteriormente se seleccionan los más importantes siguiendo el modelo *SVM*.

Por lo tanto, se tienen 7 configuraciones diferentes para cada modelo definido anteriormente. Una vez se definen los modelos y las configuraciones posibles, se inicializan con el conjunto de datos del dataset. Esta inicialización se hace mediante la función *fit*.

La función de inicialización permite construir los diferentes árboles de decisión en el caso del modelo *RandomForest* y construir los hiperplanos en el caso de *SVM*, sin embargo, en el caso del *kNN* no se construye un modelo. Con el fin de comprobar que los diferentes modelos se construyen correctamente y pueden clasificar nuevos datos a partir del conjunto de datos del dataset, es necesario comprobar su "validez" o grado de "certeza" en la clasificación de actividades.

Para hacer esto, se inicializan siguiendo la técnica de *cross-validation*. Esto consiste en dividir el conjunto de datos de entrada en diferentes conjuntos de datos, por ejemplo, 10. De estos conjuntos de datos, los modelos hacen uso de una cantidad de ellos para entrenar, por ejemplo 8, y otro conjunto de ellos para validar el modelo, en este caso, 2. De esta forma, se permite visualizar en un primer momento si los modelos clasificarán correctamente las nuevas actividades, huyendo del overfitting.

En este caso, se aplica la técnica de *cross-validation* dos veces. En la primera ocasión el conjunto de entrenamiento es el 70% y el conjunto de validación el 30%, y en la segunda ocasión el conjunto de entrenamiento es del 80% y el conjunto de validación del 20%.

Los resultados de aplicar la técnica de *cross-validation* en los diferentes modelos con las configuraciones establecidas son los que se recogen en la Tabla 7.1. En ella, se indican los diferentes modelos con las configuraciones establecidas, además de los resultados de certeza (número de instancias correctamente clasificadas/número total de instancias) de dos iteraciones junto con la media de estas.

Modelo	**Tipo**	**Certeza 1**	**Certeza 2**	**Certeza Media**
kNN_raiz	def	80.62	80.25	80.44
kNN_raiz	escalado	89.54	90.15	89.84
kNN_raiz	normalizado	86.91	87.97	87.44
kNN_raiz	ats_var	80.62	80.25	80.44
kNN_raiz	ats_model	80.74	80.42	80.58
kNN_raiz	esc_ats_var	28.11	30.25	29.18
kNN_raiz	norm_ats_model	87.74	88.92	88.33
kNN_10	def	92.45	91.8	92.12
kNN_10	escalado	96.78	96.82	96.8
kNN_10	normalizado	95.6	96.34	95.97
kNN_10	ats_var	92.45	91.8	92.12
kNN_10	ats_model	92.77	92.22	92.5
kNN_10	esc_ats_var	26.02	20.7	23.36
kNN_10	norm_ats_model	95.91	96.4	96.16
RandomForest	def	99.1	99.41	99.25
RandomForest	escalado	99.02	99.53	99.28
RandomForest	normalizado	98.98	99.71	99.34
RandomForest	ats_var	98.98	99.41	99.19
RandomForest	ats_model	98.94	99.47	99.2
RandomForest	esc_ats_var	29.21	28.66	28.94
RandomForest	norm_ats_model	99.02	99.41	99.22

Modelo	Tipo	Certeza 1	Certeza 2	Certeza Media
SVM	def	88.44	90.68	89.56
SVM	escalado	91.27	91.75	91.51
SVM	normalizado	96.89	97.41	97.15
SVM	ats_var	87.54	89.21	88.38
SVM	ats_model	82.39	85.73	84.06
SVM	esc_ats_var	26.26	26.12	26.19
SVM	norm_ats_model	97.17	97.46	97.31

Tabla 7.1: Certeza de las diferentes configuraciones de los modelos

A la vista de los resultados, la conclusión general es que el dataset que se ha generado permite reconocer correctamente las actividades que ha definido el usuario. En la mayor parte de los modelos y las diferentes configuraciones, el porcentaje de acierto es muy elevado, en torno al 90%.

Además, se puede observar que generalmente, cuando se realiza la versión de *cross-validation* con más registros de entrenamiento (versión 2), se obtiene mejor porcentaje de acierto. Esto se debe a que los algoritmos tienen más registros y por lo tanto, tienen más información de las actividades. En ambos caso, el porcentaje de actividades de entrenamiento es del 70% y 80% del dataset. En el modelo que resulte finalmente el porcentaje será del 100%, por lo que la certeza será incluso mayor que lo que se muestra en la tabla.

En cuanto a los diferentes modelos, se puede observar que el modelo **kNN_raiz** no obtiene tan buenos resultados como el resto de modelos. Esto se debe a que se ha establecido el parámetro k del modelo como k=$\sqrt{longitud\_dataset}$. En este caso, el dataset no tiene el mismo número de registros de cada actividad, habiendo actividades que tienen muchos más registros que otras. Por lo tanto, al tener que tener una gran cantidad de vecinos para clasificarse una actividad, las actividades con menos registros difícilmente se clasifican correctamente.

En el caso de **kNN_10**, el valor de k es 10 y por lo tanto, todas las actividades pueden reconocerse sin problema. En este, la configuración por defecto no se comporta correctamente porque los atributos tienen diferentes escalas y se ven sobrerepresentadas las actividades con valores más altos (que se hagan con más rapidez o que se encuentren en la parte de la vivienda con mayores valorse en las coordenadas) en detrimento de las actividades con valores más bajos. Además, se puede apreciar que al seleccionar atributos, se mejora el grado de acierto de las actividades. En definitiva, produce un 96% aproximadamente de certeza siendo un clasificador muy sencillo que no hace uso de un modelo, por lo que puede ser útil en este caso en detrimento de otros clasificadores más complejos.

Por otra parte, el clasificador **RandomForest** se comporta de forma similar con todas las configuraciones, teniendo un porcentaje de acierto de entorno al 99%. El modelo *RandomForest* huye del overfitting al hacer uso de 100 árboles de decisión en este caso, por lo que los atributos menos importantes de forma general afectan menos en la clasificación. Este clasificador a priori parece el más indicado con el objetivo de realizar el reconocimiento de las actividades, ya que

es modelo que mejor resultados obtiene.

Finalmente, el clasificador **SVM** como se puede observar, no obtiene buenos resultados cuando los datos no están normalizados. Por lo tanto, para tener en cuenta el modelo, se normalizan los datos. Además el hecho de seleccionar los atributos normalizados también tiene buenos resultados. La certeza es en torno al 97% en estos casos, bastante similar al clasificador **kNN**, ambos más ligeros que el clasificador *RandomForest*.

Una vez se han obtenido los modelos con las diferentes configuraciones, se hace uso de varios de ellos para hacer sucesivas pruebas con datos en tiempo real diferentes a los datos del dataset. De esta forma, se puede comprobar el grado de acierto con nuevos registros con la finalidad de discernir qué modelos están sobrerepresentados y cuales de ellos se comportan mejor en la detección de actividades en tiempo real.

7.1.3 Análisis de secuencias de actividades

Una vez definidos los modelos y las diferentes configuraciones, con el propósito de probar la certeza de las diferentes actividades que se incluyen en el reconocimiento de actividades, se define una serie de secuencias de actividades. En la Figura 7.2 se indican las tres secuencias definidas. Estas se corresponden con diferentes actividades que el usuario normalmente realiza en la vivienda, en el orden que este las realiza.

Figura 7.2: Secuencias de actividades

Una vez definidas las diferentes secuencias de actividades, se realiza un análisis de los resultados de los diferentes algoritmos para cada una de las secuencias. De esta forma, se puede comprobar cómo se comportan los algoritmos con los nuevos registros y las diferencias de certeza entre ellos.

Para realizar los gráficos de barras se ha hecho uso de la librería *matplotlib* y en el caso de las matrices de confusión, de la librería *seaborn*. En los siguientes subapartados se realiza el análisis de los resultados procedentes de la realización de las diferentes secuencias.

La matriz de confusión se usa en el campo de *machine learning* para comprender en qué clases está fallando la clasificación de los registros. Así, el valor de "certeza" indica el porcentaje de registros correctamente clasificados del total de registros clasificados pero no ofrece ninguna información de las clases que más han fallado al clasificarse.

Así, la matriz de confusión muestra las clases en las que más veces se ha equivocado el modelo cuando realiza la predicción. En ella se representa en las filas las clases (actividades en este caso) correctas y en las columnas, las clases que el modelo predice. Por lo tanto, en la diagonal se encuentran los registros que se han clasificado correctamente. De esta forma se detecta las actividades que los modelos confunden y se pueden entrenar para que dejen de ser confundidas.

A continuación se presentan los resultados para cada una de las secuencias. Una vez se ejecuta una secuencia y se guardan los resultados, los modelos de *machine learning* se actualizan con los resultados de la secuencia, de esta forma se aprende en cada nueva ejecución. Esta es una de las principales características del aprendizaje automático.

7.1.3.1 Secuencia 1

En primer lugar se recogen los resultados de la Secuencia 1. El total de tiempo que ha durado la ejecución de la secuencia es de 8 minutos. En la Figura 7.3 se indica el porcentaje de certeza de cada uno de los modelos con las diferentes configuraciones.

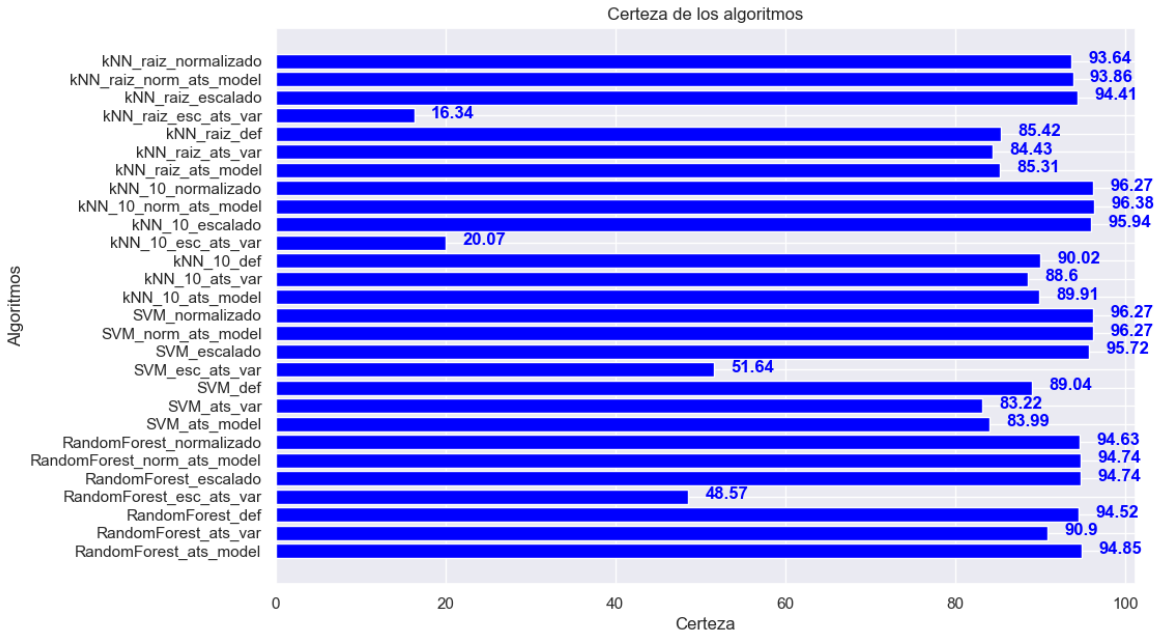


Figura 7.3: Secuencia 1 - Modelos

Como se puede apreciar, los modelos se comportan bastante bien de forma general, aunque en este caso la selección de atributos haciendo uso de la varianza no arroja buenos resultados. Sin embargo, el resto de configuraciones se comporta de forma parecida entre sí.

De esta forma, la tasa de certeza en las mejores configuraciones de los diferentes algoritmos es entorno al 95%, por lo que es un porcentaje bastante elevado y por lo tanto, el reconocimiento

de actividades se realiza de forma correcta a priori.

Sin embargo, con esta gráfica no se puede apreciar el grado de certeza de cada actividad. En la Tabla 7.2 se señala la certeza para cada una de las actividades que se ha realizado teniendo en cuenta la mejor configuración de cada modelo. Como se puede observar en la Figura, la detección de las actividades es correcta en la mayor parte de estas con un grado de certeza del 100%.

Actividad	**Hora de inicio**	**Modelo**	**Accuracy**
Dormir	10:40:42	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Andar	10:41:13	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Estar con el móvil	10:41:35	kNN_raiz_escalado	97.97
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Ir al WC	10:42:38	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Lavarse las manos	10:43:02	kNN_raiz_escalado	97.67
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Comer	10:43:32	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Lavarse las manos	10:44:05	kNN_raiz_escalado	90.32
		kNN_10_norm_ats_model	74.19
		SVM_normalizado	90.32
		RandomForest_ats_model	90.32
Lavarse los dientes	10:44:23	kNN_raiz_escalado	47.36
		kNN_10_norm_ats_model	59.64
		SVM_normalizado	47.36
		RandomForest_ats_model	26.31
Peinarse	10:45:05	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Vestirse	10:45:39	kNN_raiz_escalado	0

		kNN_10_norm_ats_model	85.71
		SVM_normalizado	92.85
		RandomForest_ats_model	85.71
Estar con el móvil	10:46:00	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100
Ordenador	10:47:06	kNN_raiz_escalado	100
		kNN_10_norm_ats_model	100
		SVM_normalizado	100
		RandomForest_ats_model	100

Tabla 7.2: Tasa de acierto de las diferentes actividades realizadas en la secuencia 1

Sin embargo, las actividades de *Lavarse las manos*, *Lavarse los dientes* y *Vestirse* tienen unos grados de acierto menor. Esto se debe a que en el caso de lavarse las manos y lavarse los dientes, el movimiento de la muñeca del usuario es muy parecido y además la localización es la misma, ya que ambas se localizan en el lavabo del cuarto de baño. En el caso de vestirse, al poder realizarse la actividad en el cuarto de baño y en el dormitorio, tiene diferentes valores y no queda muy definida.

Por lo tanto, para mejorar la certeza de estas actividades es necesario continuar con el entrenamiento del algoritmo hasta que finalmente realmente "aprenda" estas actividades.

En la Figura 7.4 se muestra la matriz de confusión referente al modelo *kNN* haciendo uso de *k=10*. En esta, se puede observar que la mayor parte de las actividades se realiza de forma correcta. Sin embargo, en el caso de *Lavarse los dientes*, se clasifica 15 veces como *Lavarse las manos*, 7 veces como *Afeitarse* y 8 veces como *Peinarse*. Por lo tanto, haciendo uso de la matriz de confusión se puede apreciar las actividades que se clasifican incorrectamente y ver con qué clases se han confundido para analizarlas y entrenarlas. Por otra parte, en el caso de *Vestirse*, no se clasifica correctamente ninguna vez en este caso.

En definitiva, la primera secuencia arroja resultados bastante buenos en cuanto a la detección de actividades, además de tratarse de una secuencia larga de actividades en varias habitaciones. En la siguiente secuencia se intentará corregir las actividades que más problemas han dado en la secuencia actual.

7.1.3.2 Secuencia 2

Posteriormente, se analizan los resultados para la secuencia 2. En este caso, la duración de la simulación ha sido de 3 minutos. En ella, se realizan un total de 6 actividades localizadas en el cuarto de baño. Además, los modelos se han actualizado con los resultados de la secuencia anterior. Así, al definir el usuario manualmente las actividades para comprobarlas posteriormente, se incluyen en el dataset las actividades mal clasificadas corregidas.

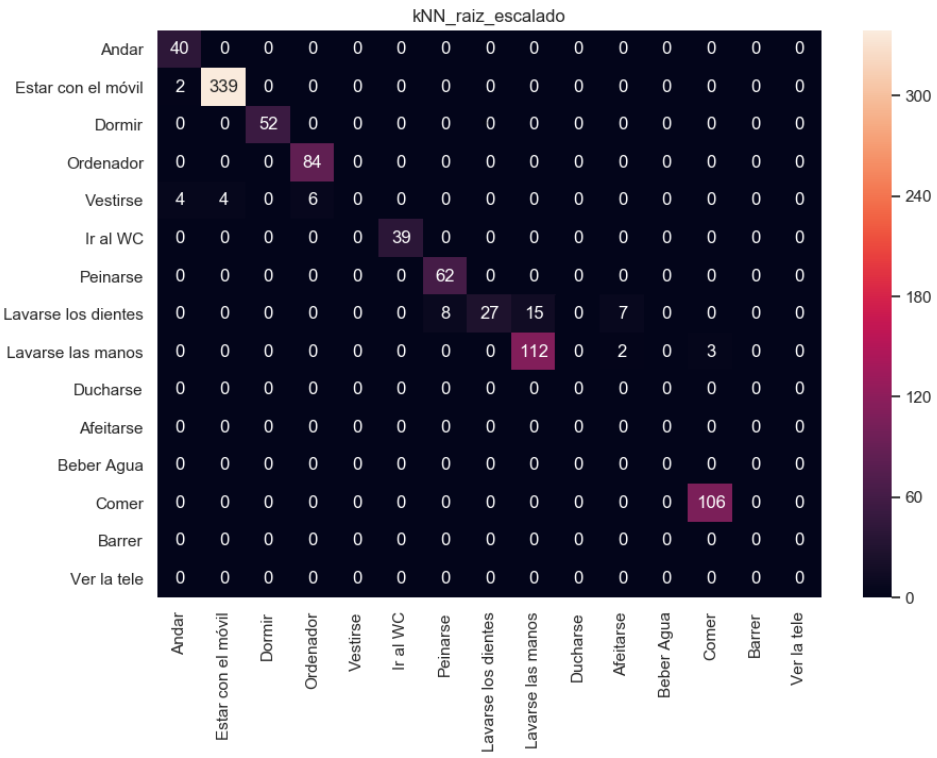


Figura 7.4: Secuencia 1 - Matriz de confusión - kNN_10 escalado

En la Figura 7.5 se pueden observar los porcentajes de certeza de cada uno de los modelos. En este caso, los modelos se acercan al 100% de certeza, diferenciando correctamente las actividades que se realizan en el cuarto de baño. Al igual que en el caso anterior, la configuración de escalado del vector de características y selección de atributos mediante varianza no funciona adecuadamente, por lo que se descarta de cara a la solución final.

Al igual que en la sección anterior, la actividad *Vestirse* no alcanza el 100% en cuanto a su clasificación (Figura 7.6), pero mejor considerablemente, superando el 80% en todos los modelos la segunda vez que se realiza dicha actividad. Esto se debe a que la secuencia tiene en cuenta los resultados de la secuencia anterior y por tanto, el porcentaje de acierto mejora al estar mejor definida la actividad.

En la matriz de confusión definida en la Figura 7.7 se puede observar que la actividad de *Vestirse* se clasifica en ciertas ocasiones como *Afeitarse*, al realizarse también en el cuarto de baño e incluir movimientos por encima de la cabeza; sin embargo, el porcentaje de acierto es mucho más elevado, quedando más acotada que anteriormente.

Por otra parte, se muestra en la Figura 7.8 la matriz de confusión del modelo *RandomForest* en el cual se alcanza el 100% de acierto en la clasificación de todas las actividades realizadas. En este caso, se observa que la matriz de confusión solamente tiene valores en la diagonal, señal de que todos los registros han sido bien clasificados.

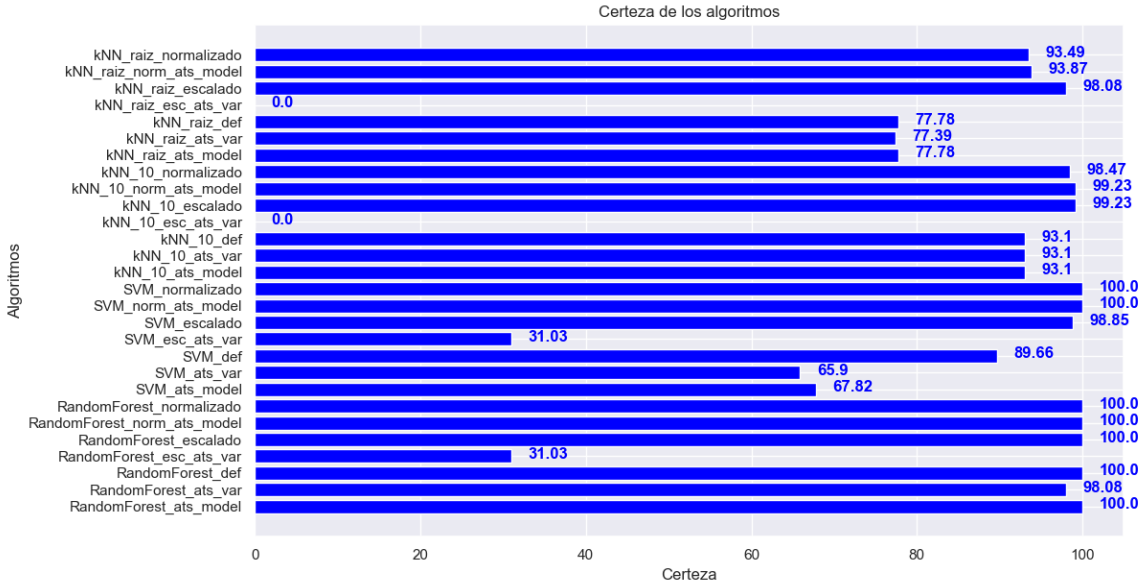


Figura 7.5: Secuencia 2 - Modelos

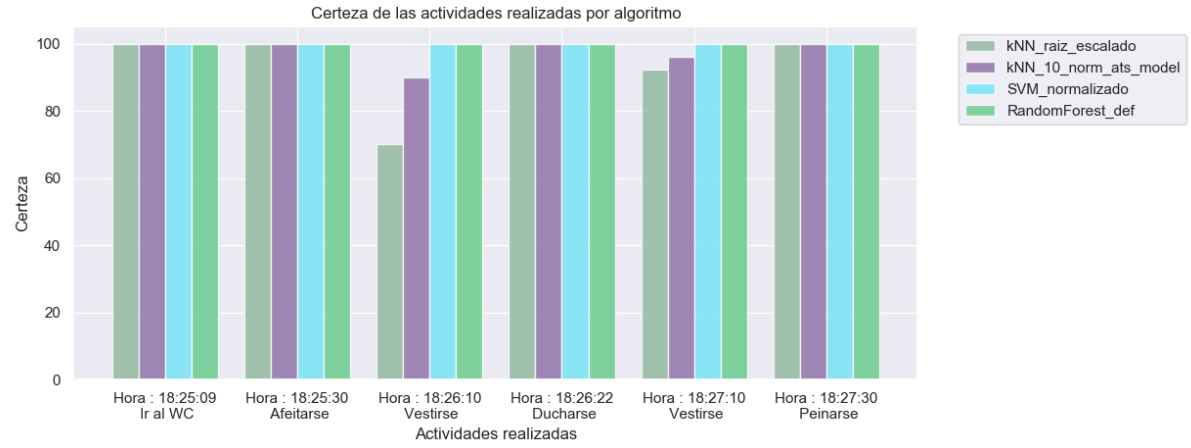


Figura 7.6: Secuencia 2 - Actividades

7.1.3.3 Secuencia 3

Finalmente, se realiza la tercera secuencia que incluye actividades en la cocina, cuarto de baño y salón. El total de tiempo que ha durado la ejecución de la secuencia es de 4 minutos.

En la Figura 7.9 se indican los diferentes modelos con todas las configuraciones posibles. Al igual que en las dos secuencias anteriores, la configuración de valores escalados con selección de atributos mediante la varianza no funciona para detectar las actividades. Sin embargo, el resto de configuraciones se sitúan en torno al 90% de media. En este caso, el modelo que mejor se comporta es *RandomForest*, seguido del algoritmo *kNN* con 10 atributos y posteriormente, *SVM*

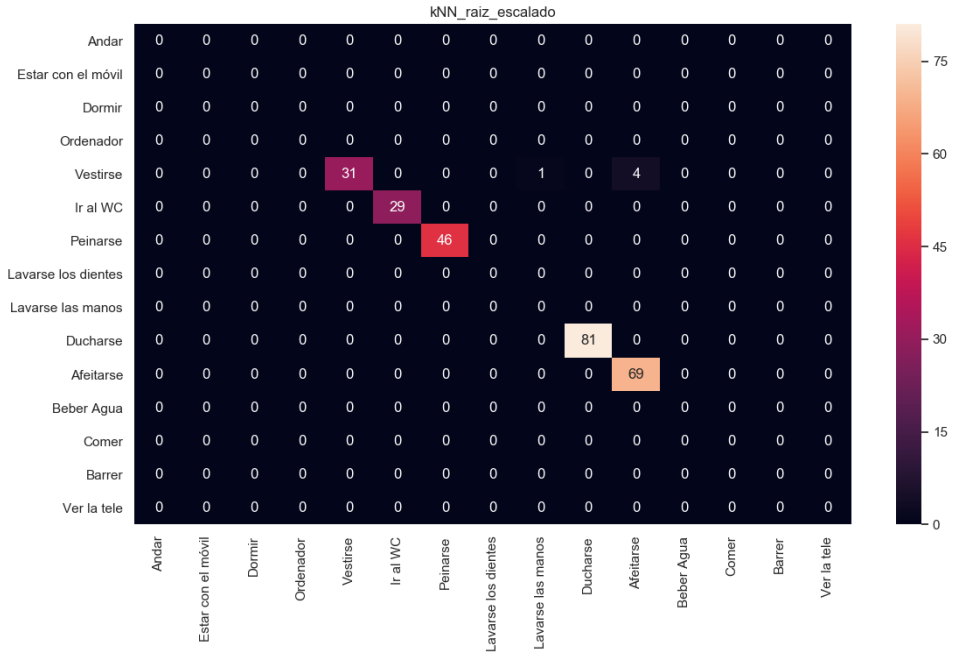


Figura 7.7: Secuencia 2 - Matriz de confusión - kNN raiz escalado

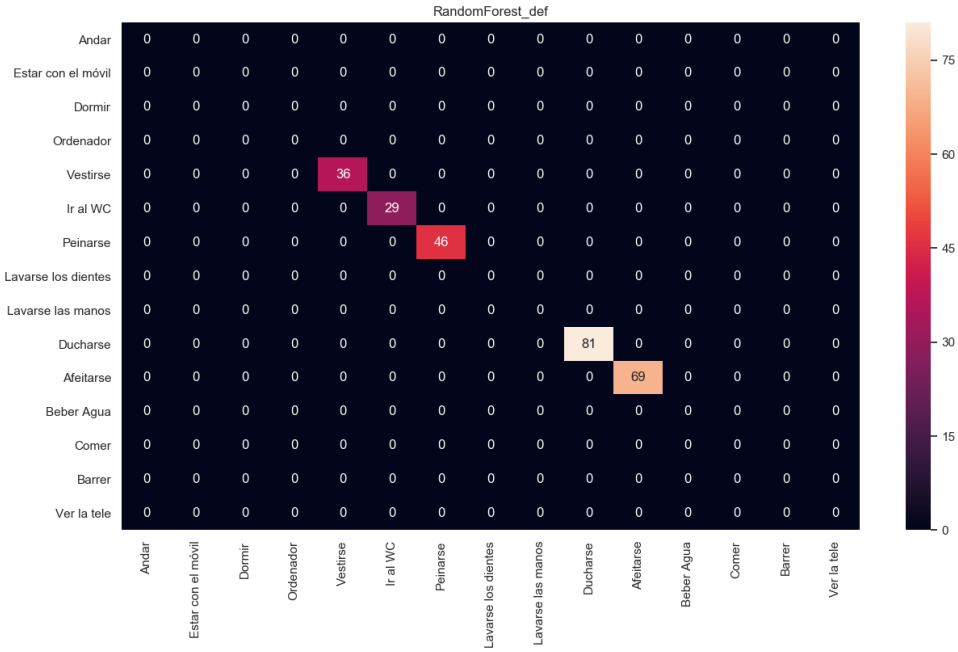


Figura 7.8: Secuencia 2 - Matriz de confusión - RandomForest por defecto

y *kNN* con *k*=raiz.

Para localizar las actividades que peor se han clasificado, se hace uso de la gráfica que se

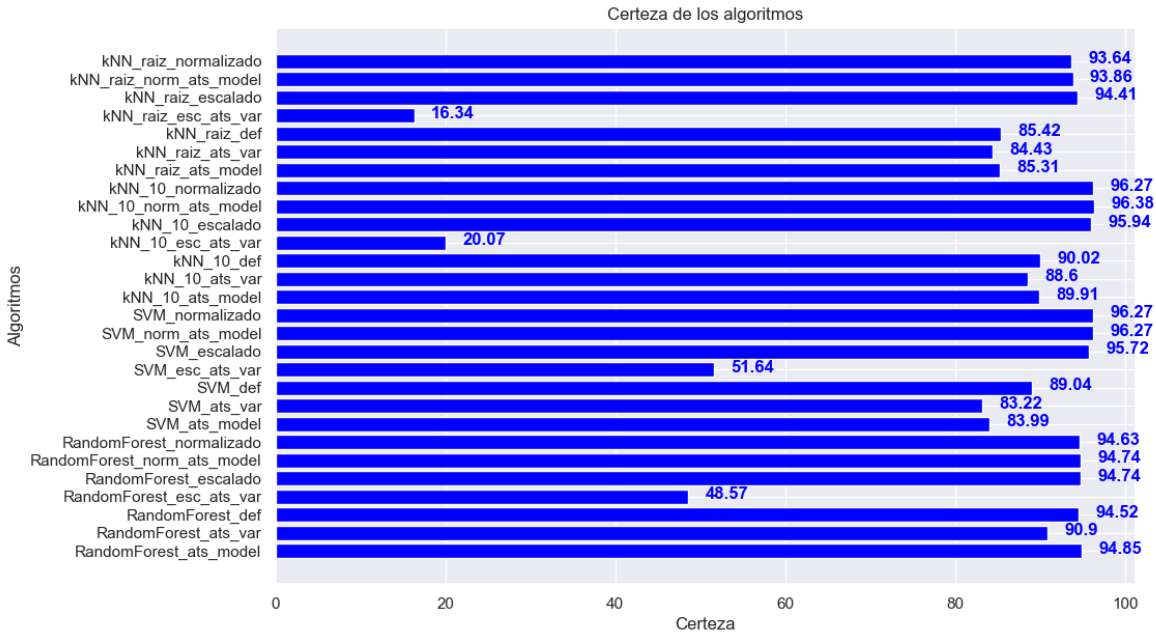


Figura 7.9: Secuencia 3 - Algoritmos

muestra en la Figura 7.10. En ella se puede observar que la actividad *Lavarse los dientes* se detecta correctamente en esta ocasión, favorecida por los registros incluidos a partir de las secuencias anteriores. Además, las actividades de la cocina se detectan con una certeza del 100% en casi todas las configuraciones. Sin embargo, las actividades del salón, al tener menos registros en el dataset, tienen una tasa de acierto menor. Sin embargo, en los mejores modelos se mantiene superior al 85%, por lo que basta con un poco de entrenamiento para que la certeza se acerque al 100%.

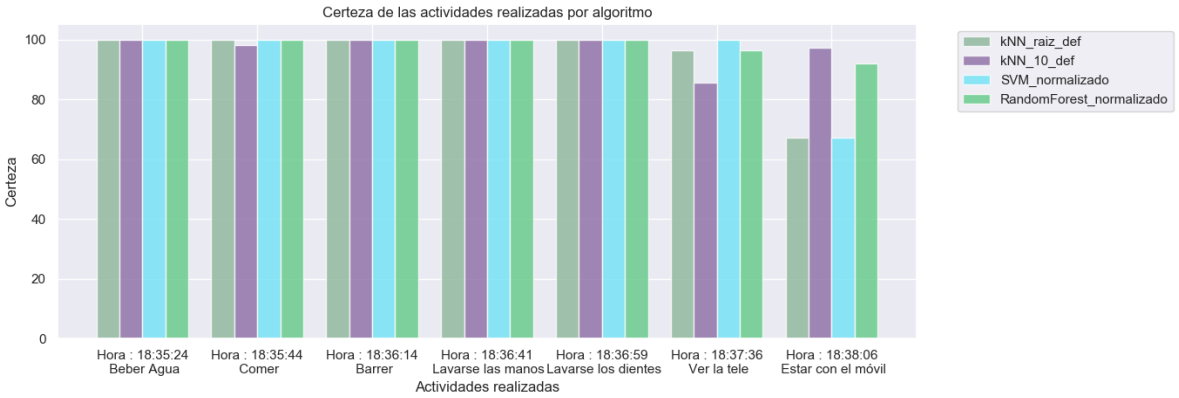


Figura 7.10: Secuencia 3 - Actividades

Para localizar las actividades que se han clasificado erróneamente y comprobar con qué clases se han confundido, como se ha realizado anteriormente, se ha hecho uso de la matriz de confusión.

En la Figura 7.11 se presenta la matriz de confusión del modelo *RandomForest* haciendo uso de la configuración por defecto. En el caso de la actividad *Estar con el móvil*, se confunde con *Ver la tele*, al estar el usuario sentado en el sofá del salón en lugar del dormitorio como anteriormente. Esto se puede corregir incluyendo más registros de la actividad *Estar con el móvil* cuando el usuario está sentado en el sofá, para que diferencie cuando el usuario tenga algo en la mano (estar con el móvil) a cuando está quieto viendo la televisión.

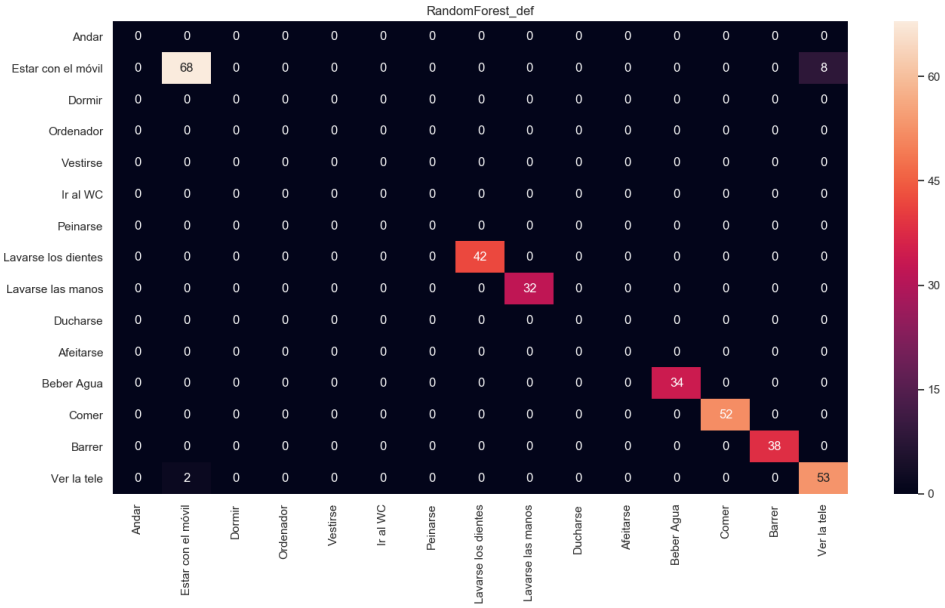


Figura 7.11: Secuencia 3 - Matriz de confusión - RandomForest por defecto

7.1.4 Mejor configuración

Una vez analizadas las secuencias de las diferentes actividades, se constata que las actividades se detectan correctamente por los diferentes algoritmos, con un porcentaje superior al 90% en el caso de las mejores configuraciones de cada algoritmo. En la Tabla 7.3 se recogen las tasas de acierto para cada configuración de los diferentes modelos.

Modelo	Secuencia 1	Secuencia 2	Secuencia 3	Media
RandomForest_ats_model	94.85	100.0	96.05	96.97
RandomForest_ats_var	90.9	98.08	96.35	95.11
RandomForest_def	94.52	100.0	96.96	97.16
RandomForest_esc_ats_var	48.57	31.03	33.13	37.58
RandomForest_escalado	94.74	100.0	97.57	97.44
RandomForest_norm_ats_model	94.74	100.0	96.96	97.23
RandomForest_normalizado	94.63	100.0	97.57	97.4
SVM_ats_model	83.99	67.82	81.76	77.86
SVM_ats_var	83.22	65.9	81.46	76.86

Modelo	Secuencia 1	Secuencia 2	Secuencia 3	Media
SVM_def	89.04	89.66	88.15	88.95
SVM_esc_ats_var	51.64	31.03	32.83	38.5
SVM_escalado	95.72	98.85	90.27	94.95
SVM_norm_ats_model	96.27	100.0	92.4	96.22
SVM_normalizado	96.27	100.0	92.4	96.22
kNN_10_ats_model	89.91	93.1	96.35	93.12
kNN_10_ats_var	88.6	93.1	96.35	92.68
kNN_10_def	90.02	93.1	96.66	93.26
kNN_10_esc_ats_var	20.07	0.0	7.29	9.12
kNN_10_escalado	95.94	99.23	92.4	95.86
kNN_10_norm_ats_model	96.38	99.23	91.79	95.8
kNN_10_normalizado	96.27	98.47	92.1	95.61
kNN_raiz_ats_model	85.31	77.78	91.19	84.76
kNN_raiz_ats_var	84.43	77.39	91.19	84.34
kNN_raiz_def	85.42	77.78	91.79	85.0
kNN_raiz_esc_ats_var	16.34	0.0	10.03	8.79
kNN_raiz_escalado	94.41	98.08	84.5	92.33
kNN_raiz_norm_ats_model	93.86	93.87	89.36	92.36
kNN_raiz_normalizado	93.64	93.49	89.06	92.06

Tabla 7.3: Tasa de acierto de las diferentes configuraciones de los modelos en las secuencias

A la vista de los resultados, el modelo **RandomForest** cosecha mejores resultados que el resto, ya que los valores de certeza se sitúan en torno al 97% en los mejores casos. Así, dentro de este, cuando se realiza una normalización o escalado previo, mejora el rendimiento respecto al vector de características original, pero la diferencia no es muy elevada (medio punto). Por lo tanto, en caso de seleccionar este modelo, se tendría en cuenta el tiempo de procesamiento que se lleva a cabo en la normalización/escalado de los vectores de características. En cuanto a la selección de atributos, esta no produce una mejora en la tasa de acierto, por lo que se comprueba que los atributos que se han incluido en el vector de características son adecuados para realizar la detección de actividades.

En cuanto al modelo **SVM**, se obtienen resultados muy parecidos al *RandomForest*, aunque un punto por debajo, en torno al 96%. Se comporta especialmente bien cuando se tiene un vector de características normalizado y en menor medida, escalado. Con el vector de características normal no se comporta adecuadamente, pudiendo decir que el modelo *SVM* no se comporta bien con vectores desnormalizados o con valores de diferentes rangos.

En último lugar, se comparan los modelos **kNN**. En ambos casos, se obtienen unas tasas de acierto superiores al 90%, siendo un modelo válido para el reconocimiento de actividades. En el caso de *k=10*, la certeza es mayor, en torno al 95%, muy parecido al *SVM*. Sin embargo, el porcentaje de acierto de la configuración con la raíz cuadrada es ligeramente inferior (92%). En ambos casos, se comportan mejor con valores normalizados y escalados que en la configu-

ración por defecto. Además, la selección de atributos no es un factor especialmente relevante, confirmando lo que se dedujo con los resultados del *RandomForest*.

Por lo tanto, una vez analizada la tabla de los resultados de las diferentes secuencias, se obtiene la conclusión que la mejor configuración general para los algoritmos consiste en la normalización de los valores del vector de características y posteriormente, la selección de atributos mediante un modelo *SVM*. Esta conclusión se corresponde con los resultados al aplicar la *cross-validation* en los registros del dataset que se pueden visualizar en la Tabla 7.1. Por lo tanto, se comprueba que al realizar la cross-validation para la evaluación de los modelos, se obtienen resultados que se corresponden con la certeza en la detección de nuevas actividades.

Esto es fundamental porque en un futuro, para probar nuevos modelos y configuraciones, no es necesario generar nuevas clasificaciones, sino que se puede aplicar la técnica de *cross-validation* al generar el modelo correspondiente y observar a priori si el modelo se va a comportar bien con nuevas actividades.

7.1.5 Configuración final

En este apartado se realiza una secuencia final en la que se engloban todas las actividades y se compara el rendimiento de la mejor configuración de cada algoritmo. En este caso, la secuencia final tiene una duración de 10 minutos, realizando un total de 15 actividades. El orden de realización de las actividades ha sido en primer lugar la secuencia 1, en segundo lugar la secuencia 3 y en último lugar la secuencia 2. En el Anexo 11 se recoge la secuencia de las actividades que se han realizado y que han sido detectadas por el algoritmo de detección de actividades.

Como resultado, se observa en primer lugar (Figura 7.12) la certeza obtenida por cada uno de los modelos con las diferentes configuraciones.

Se puede observar que, de acuerdo con los resultados obtenidos al aplicar la técnica de *cross-validation*, el modelo que mejor se comporta en este caso es *RandomForest*, seguido del SMV. Después se encuentra el modelo *kNN* haciendo uso de *k=10* y en último lugar el modelo *kNN* haciendo uso de *k=raiz*. Además, hay que destacar que los 4 modelos generan resultados de certeza en un porcentaje superior al 90%, por lo que la clasificación de las actividades en todos ellos se realiza de forma muy precisa.

Por otra parte, en cuanto a las actividades a clasificar, a continuación se muestran las diferentes matrices de confusión de los 4 modelos que se analizan en esta sección.

En primer lugar, se aprecia la matriz de confusión referente al algoritmo *kNN* con *k=raiz* (Figura 7.13). En esta se puede observar que la actividad *vestirse* no se reconoce de forma correcta, obteniendo un porcentaje de certeza inferior al 50%. Además, se producen ciertos errores cuando el usuario está con el móvil, clasificando como la actividad *ver la tele*. En el resto de actividades el acierto es bastante elevado.

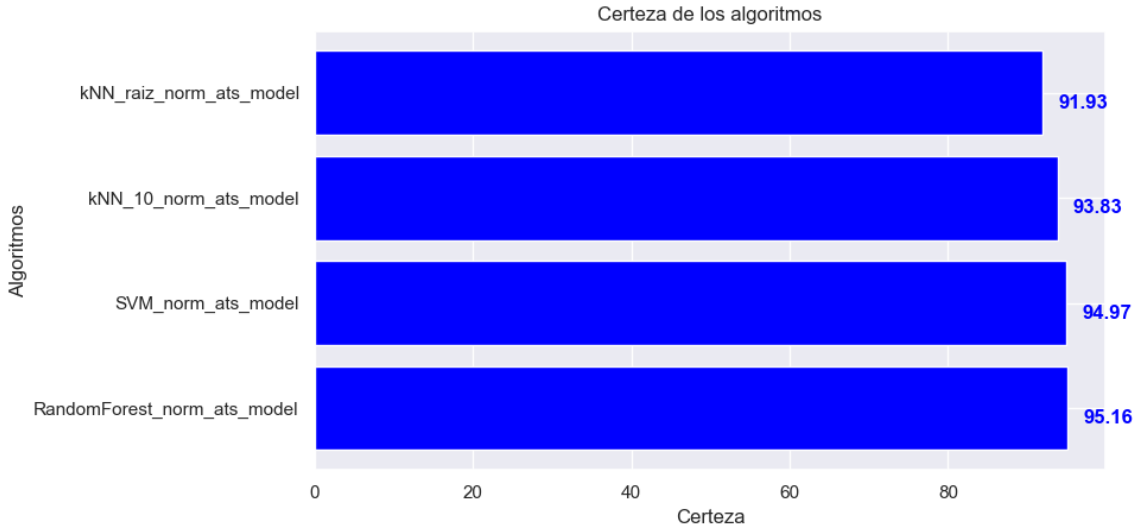


Figura 7.12: Secuencia Final - Algoritmos

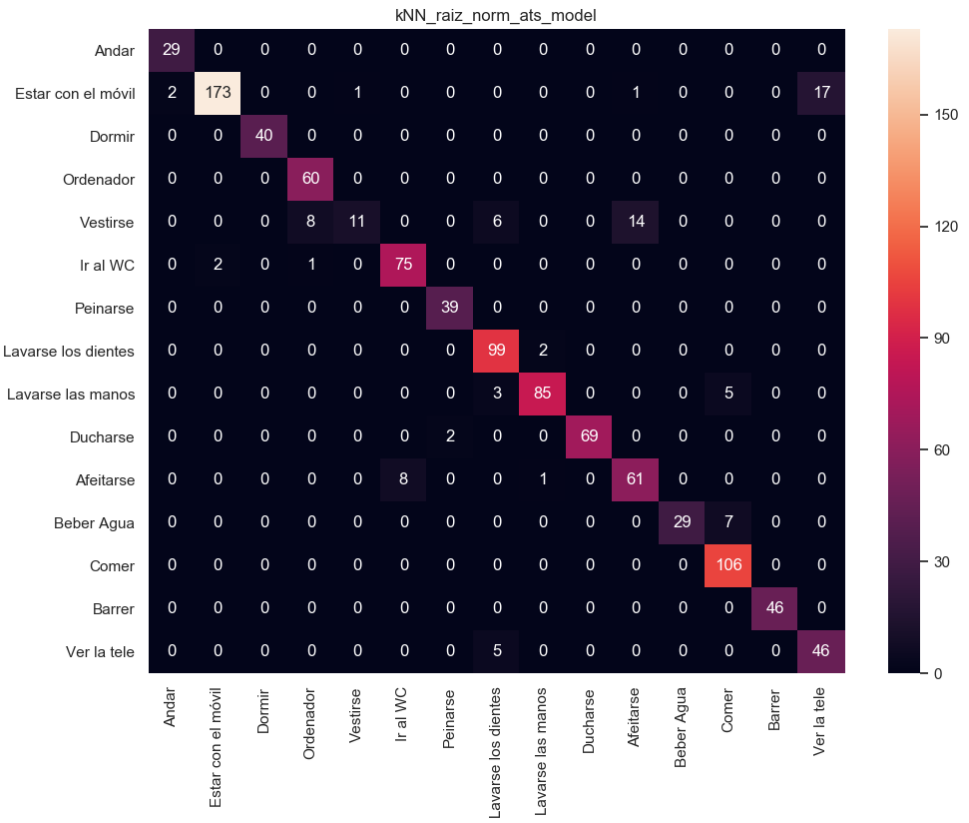


Figura 7.13: Secuencia Final - Matriz de confusión - kNN k=raiz

En segundo lugar, se visualiza la matriz de confusión referente al algoritmo kNN con *k=10*

(Figura 7.14). Al igual que en el caso anterior, se producen errores en el reconocimiento de la actividad *vestirse* y en el caso de *ver la tele*, obteniendo resultados bastante parecidos en el resto de actividades.

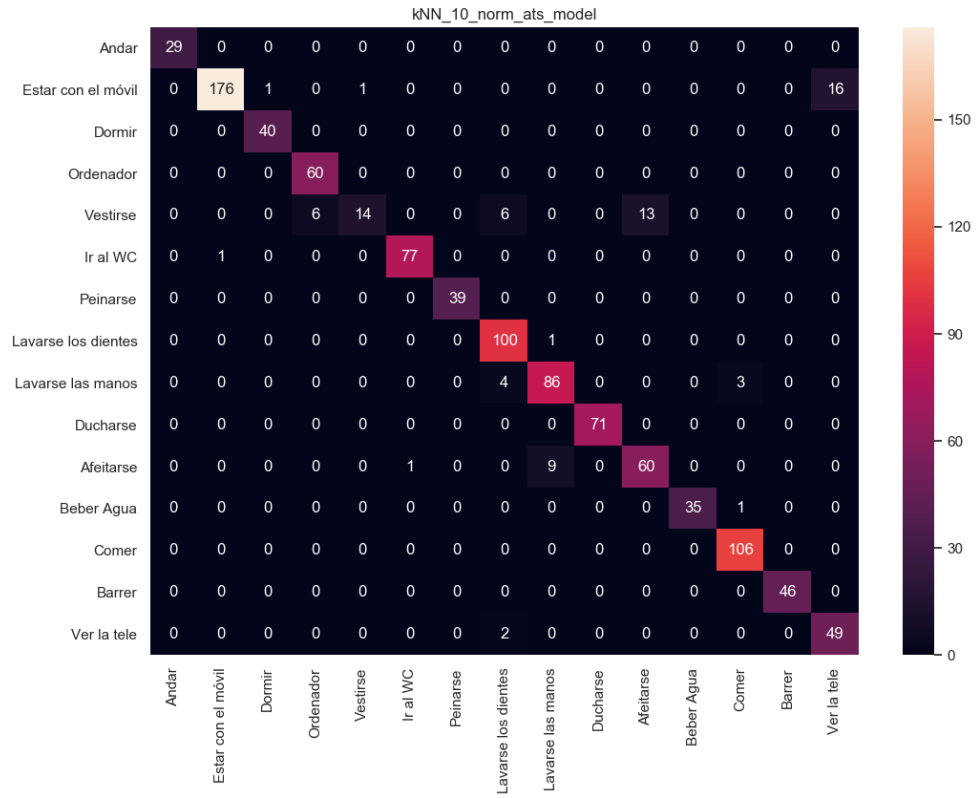


Figura 7.14: Secuencia Final - Matriz de confusión - kNN k=10

A continuación, se presenta la matriz de confusión del algoritmo *SVM* (Figura 7.15). En este caso, la actividad de *vestirse* se reconoce algo mejor pero sigue con un porcentaje de acierto inferior al 50%, confundiéndose con las actividades *lavarse los dientes* y *afeitarse*. Al igual que en los casos anteriores, también se confunde al clasificar la actividad de *estar con el móvil* cuando el usuario está sentado en el sofá.

Finalmente, en la Figura 7.16 se muestra la matriz de confusión del modelo RandomForest con mejores resultados. En este caso, la actividad *vestirse* logra reconocerse con un mayor grado de certeza que en casos anteriores, superando el 50%. Por otra parte, la actividad de *estar con el móvil* sigue siendo clasificada como *ver la tele* cuando el usuario está sentado en el sofá.

Por lo tanto, el modelo que mejor se comporta se trata del **RandomForest**. En la Figura 7.17 se indica el grado de certeza de las diferentes actividades realizadas en la secuencia final haciendo uso del modelo *RandomForest*. Como ya se ha comentado anteriormente, la actividad que no se reconoce correctamente es *vestirse*, sin embargo, el resto de actividades se identifican con un alto grado de certeza, teniendo éxito el sistema diseñado y el modelo escogido.

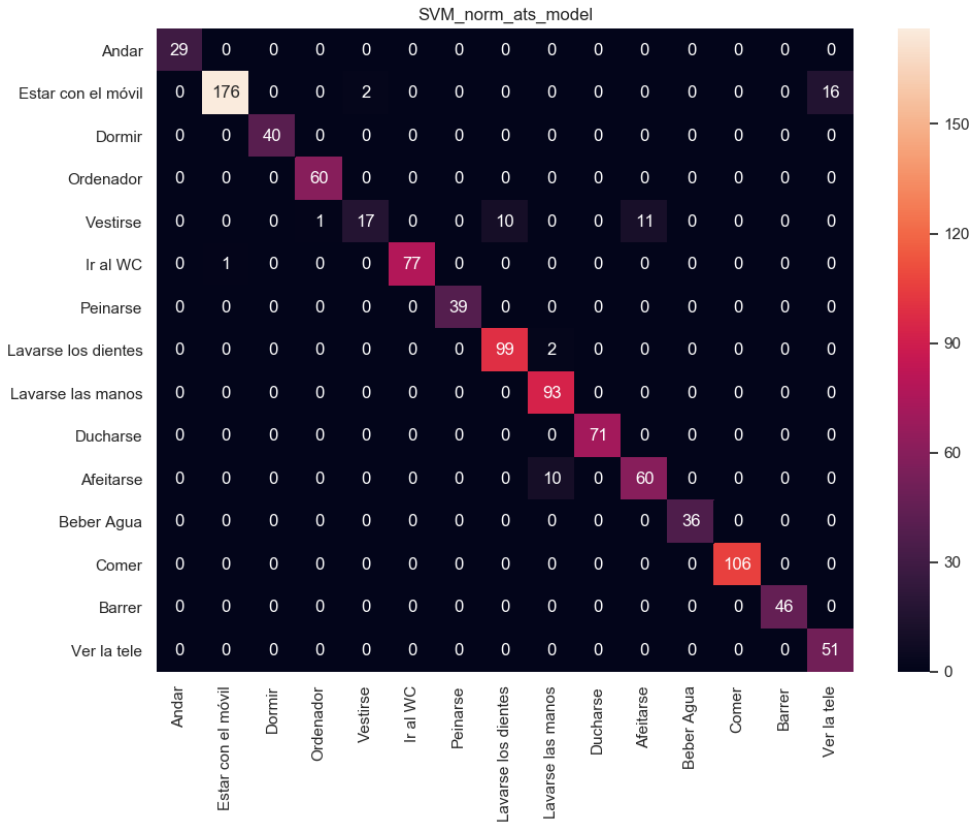


Figura 7.15: Secuencia Final - Matriz de confusión - SVM

En definitiva, los cuatro modelos se comportan correctamente con el dataset actual, teniendo unas tasas de acierto superiores al 90%. Sin embargo, hay varias actividades que requieren un mejor entrenamiento para conseguir una mayor diferenciación entre ellas.

Como resultado final, para realizar el reconocimiento de actividades en los scripts que procesan los datos, se incluye el modelo **RandomForest** con una configuración de normalización de datos y selección de atributos mediante *SVM*. En este caso, la inicialización de los scripts es cada vez mayor, ya que se tiene un conjunto mayor de datos de entrenamiento, pero el porcentaje de acierto es muy elevado, por lo que merece la pena optar por esta configuración.

A modo de mejora, se podrían incluir tres modelos y combinar los resultados de estos para clasificar la actividad ya que la clasificación del dato correspondiente a cada ventana de tiempo se realiza de forma muy rápida permitiendo clasificar el registro usando varios modelos en tiempo real.

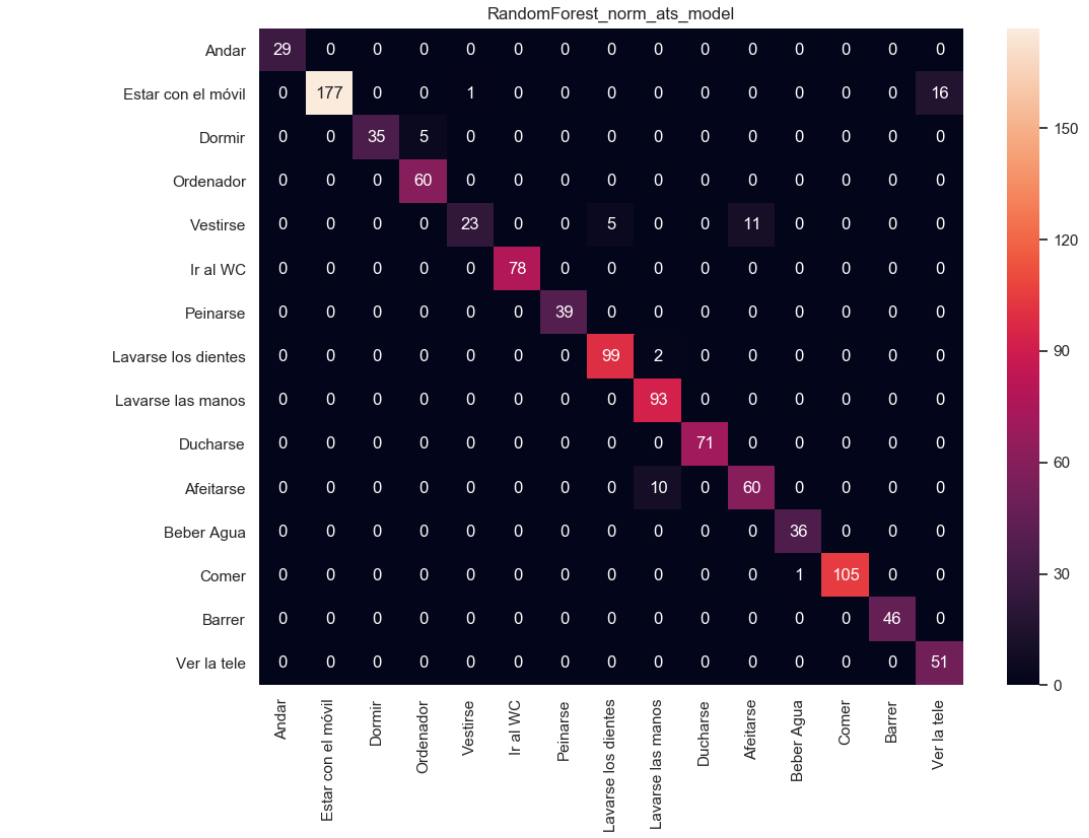


Figura 7.16: Secuencia Final - Matriz de confusión - RandomForest

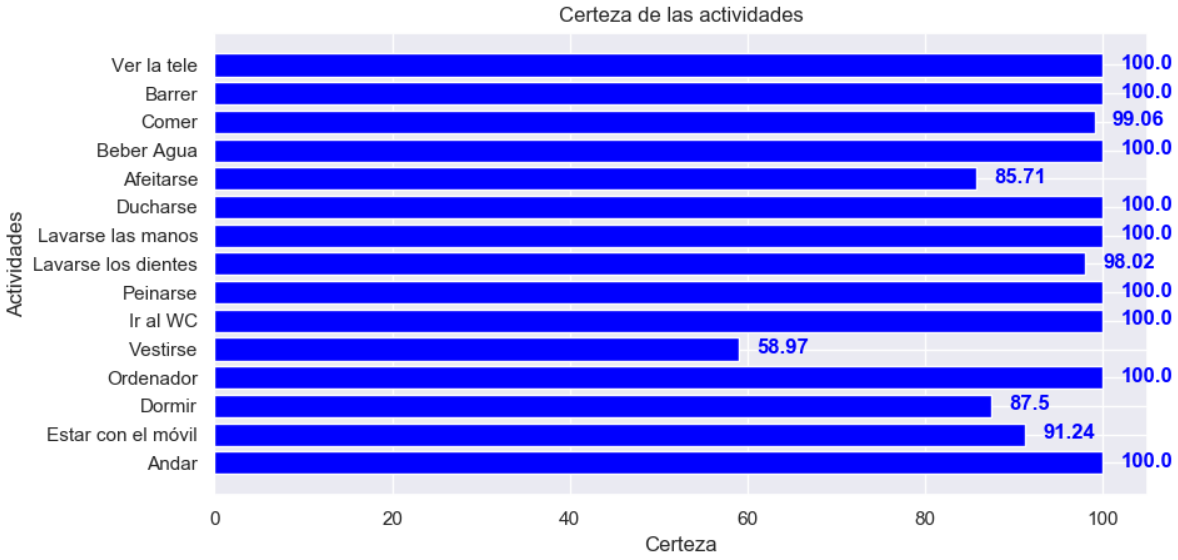


Figura 7.17: RandomForest - Secuencia final - Certeza actividades

7.2 Visualización de actividades

El objetivo general del sistema desarrollado es la visualización de las actividades que realizan los usuarios en una *smart home* a lo largo del día. Dentro de la *smart home* pueden habitar varios usuarios y cada uno de ellos realiza diferentes actividades. Por lo tanto, es necesario proporcionar una herramienta para visualizar las actividades que han ocurrido en la vivienda. Esta herramienta se proporciona a través de una interfaz web incluida en la página web desplegada como resultado del Trabajo Fin de Grado. En esta sección se recoge la funcionalidad del sistema que permite al usuario visualizar las actividades en el sistema web.

7.2.1 Extracción de actividades generales

El algoritmo de reconocimiento de actividades desarrollado, como resultado, obtiene la actividad que realiza el usuario cada medio segundo. Por lo tanto, si posteriormente se quieren visualizar las actividades que ha realizado el usuario, se tiene una gran cantidad de registros de medio segundo cada uno, concretamente, 172.800 registros al día por usuario. Tal cantidad de datos es demasiado elevada para poder entender las actividades que han ocurrido a lo largo del día, por lo que es necesario agrupar los registros correspondientes a cada actividad y conocer su duración real. En la Tabla 7.4 se puede observar un ejemplo de secuencia de actividades obtenidas por el algortimo de *machine learning*.

Hora	**Actividad**	**Hora**	**Actividad**
10:55:13.336	Ducharse	10:55:17.836	Vestirse
10:55:13.836	Ducharse	10:55:18.336	Vestirse
10:55:14.336	Ducharse	10:55:18.836	Vestirse
10:55:14.836	Ducharse	10:55:19.336	Lavarse las manos
10:55:15.336	Ducharse	10:55:19.836	Lavarse las manos
10:55:15.836	Ducharse	10:55:20.336	Lavarse las manos
10:55:16.336	Ducharse	10:55:20.836	Lavarse las manos
10:55:16.836	Vestirse	10:55:21.336	Lavarse las manos
10:55:17.336	Vestirse	10:55:21.836	Lavarse las manos

Tabla 7.4: Secuencia de actividades obtenidas por el algoritmo

Como se puede observar, se tienen las actividades que ha hecho el usuario a lo largo de 9 segundos y hay como resultado 18 registros que incluyen tres actividades. Para poder mostrar las actividades agrupadas es necesario ejecutar un script cuyo objetivo es formar actividades en tiempo real a partir de la secuencia de registros como se observa en la Tabla 7.4.

El funcionamiento del script es el que procede. Cada 10 segundos, se recogen las actividades que se han realizado en la vivienda en ese intervalo de tiempo. Las actividades se obtienen de forma ordenada, siendo la primera actividad la actividad más antigua. Una vez obtenidas

las actividades, se almacena la hora de inicio de la primera actividad y dicha actividad. A continuación, se obtiene la siguiente actividad. Si es la misma actividad, se continúa la ejecución hasta que la actividad obtenida sea diferente. Cuando el modelo predice una actividad diferente, se establece el instante de fin de la actividad anterior y se inserta en la base de datos tal y como se indica en el Código 7.1.

Código 7.1: Inserción en base de datos dato de entrenamiento

```
insertar = {
    "actividad" : actividad_anterior["actividad"],
    "inicio" : actividad_anterior["inicio"],
    "fin" : actividad_anterior["fin"],
    "persona" : actividad_anterior["persona"],
    "duracion" : actividad_anterior["duracion"],
    "duracion_str" : actividad_anterior["duracion_str"],
    "inicio_str_fecha" : actividad_anterior["inicio_str_fecha"],
    "inicio_str_hora" : actividad_anterior["inicio_str_hora"],
    "fin_str" : actividad_anterior["fin_str"],
    "nombre_actividad" : nombre_actividades[str(actividad_anterior["actividad"])] ↩
        ↪ if actividad_anterior["actividad"] != 0 else ""
}
actividades_resultado.insert_one(insertar) if actividad_anterior["actividad"] != ↩
    ↪ 0 and actividad_anterior["duracion"] > 4 else ""
```

En la parte inferior, se muestra la condición de inserción en la base de datos. Solamente se insertan actividades con una duración de más de 4 segundos. Esto se hace porque hay casos en los que el algoritmo de reconocimiento de actividades, cuando se produce un cambio de actividad, puede ser un poco impreciso.

Finalmente, cuando se produce la inserción en la base de datos, se actualiza la actividad ”actual” junto con su hora de inicio y se continúa el procesamiento de las siguientes actividades. De esta forma, se van agrupando las actividades y se tiene la información que importa sobre la actividad que realiza el usuario, que consiste en la hora de inicio, la duración y la hora de fin.

El resultado es una secuencia como se muestra en la Tabla 7.5.

Hora	Actividad	Duración
10:55:13.336	Ducharse	00:00:03.000
10:55:16.836	Vestirse	00:00:02.500
10:55:19.336	Lavarse los dientes	00.00:03.000

Tabla 7.5: Registros de actividades realizadas agrupadas

7.2.2 Interfaz web actividades realizadas

Se pueden visualizar las actividades que han realizado los usuarios en la vivienda a lo largo del tiempo en la página "Lista actividades" del sistema web (Figura 7.18). Esta funcionalidad ha sido añadida al sistema web desarrollado en el Trabajo Fin de Grado. Por lo tanto, aunque el algoritmo y el procesamiento de los datos se ejecuten con scripts en *python*, el resultado se visualiza en una interfaz web para móvil y PC accesible desde la *smart home* y el exterior.

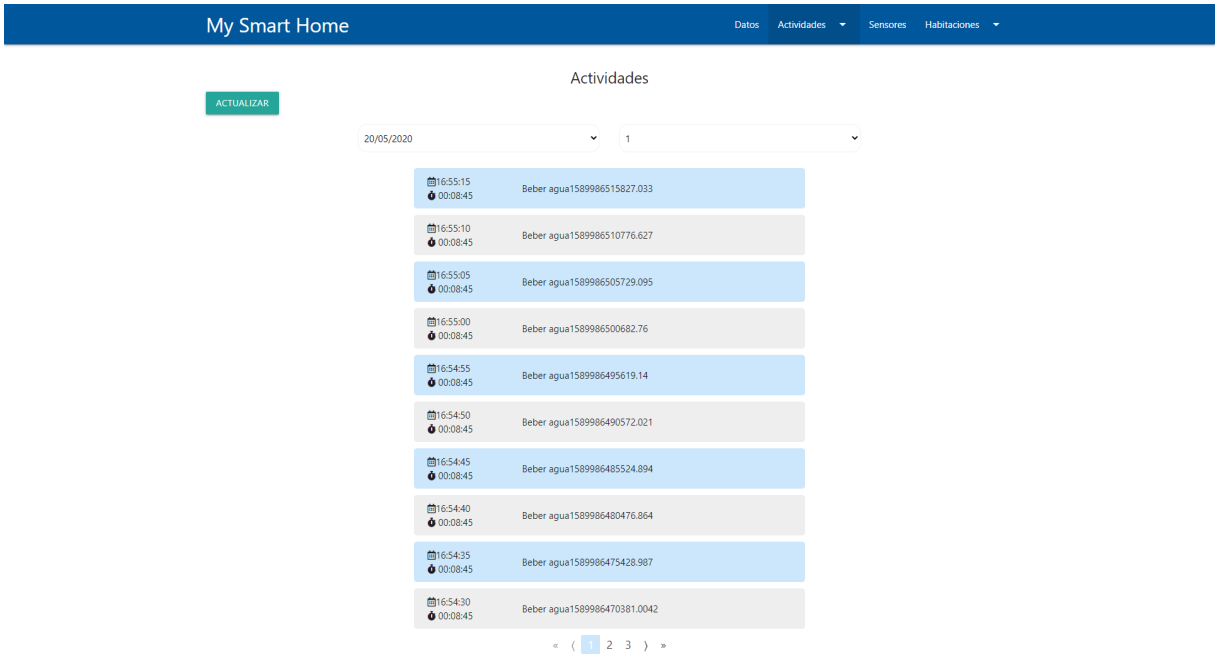


Figura 7.18: Sistema web - Lista de actividades

Concretamente, para visualizar las actividades que se realizan en la vivienda, se disponen de dos filtros u opciones. En primer lugar, el usuario selecciona el día del cual quiere visualizar las actividades y en segundo lugar, el usuario que realiza dichas actividades. De esta forma, se permite navegar en todo el histórico de actividades de una forma rápida y accesible. Por otra parte, se incorpora una paginación para organizar mejor la información que se ofrece al usuario y que los resultados carguen de forma más rápida.

8 Conclusiones y trabajo futuro

8.1 Conclusiones

El objetivo principal del presente Trabajo fin de Master consistía en realizar un reconocimiento de las actividades que realizan varios usuarios en una *smart home* de forma concurrente. Además, este reconocimiento de actividades debía realizarse en tiempo real, permitiendo al personal que monitoriza la vivienda, la rápida actuación en caso de emergencia.

Para realizar todo esto, fue necesario incorporar nuevos tipos de sensores y dispositivos en la *smart home*, tales como sensores binarios (presencia y apertura), de localización (mediante la tecnología UWB[1]) y *smartwatches*. De esta forma, se obtienen los datos de sensores de diferentes fuentes que describen de forma apropiada la actividad del habitante de la vivienda en todo momento.

Sin embargo, la incorporación de estos nuevos dispositivos proporciona al sistema una gran cantidad de datos por segundo que es necesario gestionar y procesar para extraer la información importante de estos. Además, dado que el objetivo principal del sistema es realizar el reconocimiento de actividades en tiempo real, es necesario la inclusión de tecnologías de caché (RedisDB) y procesamiento distribuido (Apache Kafka) para acometer esta tarea.

Por lo tanto, al hacer uso de las herramientas mencionadas anteriormente, se permite que el sistema obtenga la información del entorno en tiempo real. No obstante, dado que las actividades que realiza el usuario no siempre se realizan de la misma forma, es necesario la incorporación de modelos de *machine learning* que gestionen toda esta información referente a las actividades. Así, a lo largo del desarrollo del sistema, se probaron diferentes modelos y configuraciones de *machine learning*. Finalmente, el modelo *RandomForest* haciendo uso de datos normalizados y seleccionando los mejores atributos mediante un modelo *SVM* linear, obtiene una certeza de en torno al 95% en el reconocimiento de 15 actividades que realiza el usuario en la vivienda.

Así, los resultados del reconocimiento de actividades que se realiza en una máquina virtual alojada en el sistema *OpenStack* de la Universidad de Almería, se incorporan a un sistema web desplegado como resultado del Trabajo Fin de Grado. De esta forma, ha sido necesario modificar este sistema para incluir la nueva información que se obtiene en el presente trabajo.

[1] Ultra Wide Band

8.2 Trabajo futuro

Debido a la situación generada por la aparición del virus SRAS-CoV-2 (Novel y cols., 2020), no se pudieron instalar los sensores de localización en la *smart home* ni realizar pruebas en el sistema desarrollado con varios usuarios. Por lo tanto, una vez que la situación mejore, el objetivo, siguiendo con el objetivo principal del proyecto, se implantarán todos los sensores y dispositivos descritos en la presente memoria y se comprobará el funcionamiento del sistema una vez desplegado en el entorno real.

Por otra parte, en el campo del reconocimiento de actividades también se usan modelos de *deep learning* (Ronao y Cho, 2016), concretamente, redes neuronales. En Song-Mi Lee y cols. (2017) se propone el uso de una red neuronal para reconocer las actividades que realiza un usuario en base a los sensores de aceleración y giroscopio del *smartphone*, obteniendo un porcentaje elevado de acierto. Así, debido a la similitud con los datos que se generan en la *smart home*, se podrían testear diferentes modelos de redes neuronales para comprobar si mejora el grado de certeza del reconocimiento de actividades desarrollado en el presente sistema.

Además, se podrían incluir nuevos dispositivos en la *smart home* para incrementar aún más el grado de certeza en la detección de actividades. Así, el uso de cámaras termográficas (Ju Han y Bhanu, 2005) se han usado para realizar la detección de actividades, preservando la intimidad de las personas de la vivienda y proporcionando información acerca de la actividad o el estado de la persona o personas en la vivienda. Así, además de hacer uso de modelos de machine learning con las secuencias de datos procedentes de los dispositivos y sensores incorporados en el presente trabajo, se podría complementar el sistema realizando análisis de imágenes proporcionadas por estas cámaras haciendo uso de modelos de redes neuronales convolucionales, como se ha definido en el párrafo anterior.

Bibliografía

Ahvar, E., Lee, G., Han, S., Crespi, N., y Khan, I. (2016, jun). Sensor Network-Based and User-Friendly User Location Discovery for Future Smart Homes. *Sensors*, *16*(7), 969. Descargado 2020-05-14, de http://www.mdpi.com/1424-8220/16/7/969 doi: 10.3390/s16070969

Alsheikh, M. A., Lin, S., Niyato, D., y Tan, H.-P. (2014). Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys Tutorials*, *16*(4), 1996-2018.

Alvarez, Y., y Las Heras, F. (2016). Zigbee-based sensor network for indoor location and tracking applications. *IEEE Latin America Transactions*, *14*(7), 3208-3214.

Arnold, S. (2020). *Ultimate 2019 Real Time Location System (RTLS) Tech Guide.* <https://www.realtimenetworks.com/blog/ultimate-2019-real-time-location-system-rtls-tech-guide>. (Accedido 14 de Mayo de 2020)

Banos, O., Galvez, J.-M., Damas, M., Pomares, H., y Rojas, I. (2014, apr). Window Size Impact in Human Activity Recognition. *Sensors (Basel, Switzerland)*, *14*(4), 6474–6499. Descargado 2020-05-14, de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4029702/ doi: 10.3390/s140406474

Boletsis, C., McCallum, S., y Landmark, B. F. (2015). The Use of Smartwatches for Health Monitoring in Home-Based Dementia Care. En J. Zhou y G. Salvendy (Eds.), *Human Aspects of IT for the Aged Population. Design for Everyday Life* (pp. 15–26). Cham: Springer International Publishing. doi: 10.1007/978-3-319-20913-5_2

Bonomi, F., Milito, R., Zhu, J., y Addepalli, S. (2012). Fog computing and its role in the internet of things. En *Proceedings of the first edition of the mcc workshop on mobile cloud computing* (p. 13–16). New York, NY, USA: Association for Computing Machinery. Descargado de https://doi.org/10.1145/2342509.2342513 doi: 10.1145/2342509.2342513

Caragliu, A., Bo, C. D., y Nijkamp, P. (2011). Smart cities in europe. *Journal of Urban Technology*, *18*(2), 65-82. Descargado de https://doi.org/10.1080/10630732.2011.601117 doi: 10.1080/10630732.2011.601117

Casale, P., Pujol, O., y Radeva, P. (2011). Human activity recognition from accelerometer data using a wearable device. En J. Vitrià, J. M. Sanches, y M. Hernández (Eds.), *Pattern recognition and image analysis* (pp. 289–296). Berlin, Heidelberg: Springer Berlin Heidelberg.

Chan, M., Campo, E., Estève, D., y Fourniols, J.-Y. (2009). Smart homes — current features and future perspectives. *Maturitas*, *64*(2), 90 - 97. Descargado de http://www.sciencedirect

.com/science/article/pii/S0378512209002606 doi: https://doi.org/10.1016/j.maturitas.2009.07.014

De-La-Hoz-Franco, E., Ariza-Colpas, P., Quero, J. M., y Espinilla, M. (2018). Sensor-based datasets for human activity recognition – a systematic review of literature. *IEEE Access*, *6*, 59192-59210.

Deng, Z., Zhu, X., Cheng, D., Zong, M., y Zhang, S. (2017). Efficient knn classification algorithm for big data. *Neurocomputing*, *195*(3), 143-148.

Espinilla, M., Martínez, L., Medina, J., y Nugent, C. (2018). The experience of developing the ujami smart lab. *IEEE Access*, *6*, 34631-34642.

Espinilla, M., Medina, J., Hallberg, J., y Nugent, C. (2018, 03). A new approach based on temporal sub-windows for online sensor-based activity recognition. *Journal of Ambient Intelligence and Humanized Computing*. doi: 10.1007/s12652-018-0746-y

Farahani, B., Firouzi, F., Chang, V., Badaroglu, M., Constant, N., y Mankodiya, K. (2018). Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare. *Future Generation Computer Systems*, *78*, 659–676.

Feng, Z., Mo, L., y Li, M. (2015). A random forest-based ensemble method for activity recognition. En *2015 37th annual international conference of the ieee engineering in medicine and biology society (embc)* (p. 5074-5077).

Figo, D., Diniz, P. C., Ferreira, D. R., y Cardoso, J. M. P. (2010, oct). Preprocessing techniques for context recognition from accelerometer data. *Personal and Ubiquitous Computing*, *14*(7), 645–662. Descargado 2020-05-14, de https://doi.org/10.1007/s00779-010-0293-9 doi: 10.1007/s00779-010-0293-9

Fiorini, L., Bonaccorsi, M., Betti, S., Esposito, D., y Cavallo, F. (2018, Oct). Combining wearable physiological and inertial sensors with indoor user localization network to enhance activity recognition. *Journal of Ambient Intelligence and Smart Environments*, *10*(4), 345–357. doi: 10.3233/ais-180493

Foundation, A. (2020). *Zookeeper.* <https://zookeeper.apache.org/>. (Accedido 10 de junio de 2020)

Gartner. (2020). *2020 connected devices.* <https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016>. (Accedido 12 de junio de 2020)

GenBeta. (2020). *Índices.* <https://www.genbeta.com/desarrollo/mongodb-creacion-y-utilizacion-de-indices>. (Accedido 10 de junio de 2020)

Gershenfeld, N., Krikorian, R., y Cohen, D. (2004, 11). The internet of things. *Scientific American*, *291*, 76-81. doi: 10.1038/scientificamerican1004-76

Ghourchian, N., Allegue-Martínez, M., y Precup, D. (2017, 02). Real-time indoor localization in smart homes using semi-supervised learning..

Gjoreski, H., Bizjak, J., y Gams, M. (2016). Using smartwatch as telecare and fall detection device. En *2016 12th international conference on intelligent environments (ie)* (p. 242-245).

Grgurić, A., Mošmondor, M., y Huljenić, D. (2019). The smarthabits: An intelligent privacy-aware home care assistance system. *Sensors*, *19*(4). Descargado de https://www.mdpi.com/1424-8220/19/4/907 doi: 10.3390/s19040907

Hamad, R. A., Hidalgo, A. S., Bouguelia, M., Estevez, M. E., y Quero, J. M. (2020, Feb). Efficient activity recognition in smart homes using delayed fuzzy temporal windows on binary sensors. *IEEE Journal of Biomedical and Health Informatics*, *24*(2), 387-395. doi: 10.1109/JBHI.2019.2918412

IArtificial. (2019a). *Random forest.* <https://iartificial.net/random-forest-bosque-aleatorio/>. (Accedido 1 de junio de 2020)

IArtificial. (2019b). *Svm.* <https://iartificial.net/maquinas-de-vectores-de-soporte-svm/>. (Accedido 1 de junio de 2020)

Iberdrola. (2019). *Machine learning.* <https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico>. (Accedido 1 de junio de 2020)

Jeong, W.-S., Kim, S.-H., y Min, K.-S. (2013). An analysis of the economic effects for the iot industry. *Journal of Internet Computing and Services*, *14*(5), 119–128.

Jimenez Ruiz, A. R., y Seco Granja, F. (2017, aug). Comparing Ubisense, BeSpoon, and DecaWave UWB Location Systems: Indoor Performance Analysis. *IEEE Transactions on Instrumentation and Measurement*, *66*(8), 2106–2117. Descargado 2020-05-14, de http://ieeexplore.ieee.org/document/7891540/ doi: 10.1109/TIM.2017.2681398

Ju Han, y Bhanu, B. (2005). Human activity recognition in thermal infrared imagery. En *2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05) - workshops* (p. 17-17).

Kalantarian, H., Alshurafa, N., y Sarrafzadeh, M. (2016, feb). Detection of Gestures Associated With Medication Adherence Using Smartwatch-Based Inertial Sensors. *IEEE Sensors Journal*, *16*(4), 1054–1061. doi: 10.1109/JSEN.2015.2497279

Kim, E., Helal, S., y Cook, D. (2010, 01). Human activity recognition and pattern discovery. *IEEE pervasive computing / IEEE Computer Society [and] IEEE Communications Society*, *9*, 48. doi: 10.1109/MPRV.2010.7

Lai, Y.-X., Lai, C.-F., Huang, Y.-M., y Chao, H.-C. (2013). Multi-appliance recognition system with hybrid svm/gmm classifier in ubiquitous smart home. *Information Sciences*, *230*, 39 - 55. Descargado de http://www.sciencedirect.com/science/article/pii/S0020025512006561 (Mobile and Internet Services in Ubiquitous and Pervasive Computing Environments) doi: https://doi.org/10.1016/j.ins.2012.10.002

Learn, S. (2019a). *Feature selection.* <https://scikit-learn.org/stable/modules/feature_selection.html>. (Accedido 1 de junio de 2020)

Learn, S. (2019b). *Preprocesamiento de datos.* <https://scikit-learn.org/stable/modules/preprocessing.html>. (Accedido 1 de junio de 2020)

Learn, S. (2019c). *Svm.* <https://scikit-learn.org/stable/modules/svm.html>. (Accedido 1 de junio de 2020)

Lin, C. M., y Chen, M. T. (2017). Design and implementation of a smart home energy saving system with active loading feature identification and power management. En *2017 ieee 3rd international future energy electronics conference and ecce asia (ifeec 2017 - ecce asia)* (p. 739-742).

Mastery, M. L. (2016). *Confusion matrix.* <https://machinelearningmastery.com/confusion-matrix-machine-learning/>. (Accedido 8 de junio de 2020)

Medina-Quero, J., Zhang, S., Nugent, C., y Espinilla, M. (2018). Ensemble classifier of long short-term memory with fuzzy temporal windows on binary sensors for activity recognition. *Expert Systems with Applications*, *114*, 441 - 453. Descargado de http://www.sciencedirect.com/science/article/pii/S0957417418304937 doi: https://doi.org/10.1016/j.eswa.2018.07.068

Mohamed, R., Perumal, T., Sulaiman, M. N., y Mustapha, N. (2017). Multi resident complex activity recognition in smart home: A literature review. *Int. J. Smart Home*, *11*(6), 21–32.

MongoDB. (2020). *Índices.* <https://docs.mongodb.com/manual/indexes/>. (Accedido 10 de junio de 2020)

Moroney, L. (2021). *Ai and machine learning for coders.* O'Reilly Media, Inc.

Nath, R. K., Bajpai, R., y Thapliyal, H. (2018). Iot based indoor location detection system for smart home environment. En *2018 ieee international conference on consumer electronics (icce)* (p. 1-3).

Ngu, A., Wu, Y., Zare, H., Polican, A., Yarbrough, B., y Yao, L. (2017). Fall Detection Using Smartwatch Sensor Data with Accessor Architecture. En H. Chen, D. D. Zeng, E. Karahanna, y I. Bardhan (Eds.), *Smart Health* (pp. 81–93). Cham: Springer International Publishing. doi: 10.1007/978-3-319-67964-8_8

Novel, C. P. E. R. E., y cols. (2020). The epidemiological characteristics of an outbreak of 2019 novel coronavirus diseases (covid-19) in china. *Zhonghua liu xing bing xue za zhi= Zhonghua liuxingbingxue zazhi*, *41*(2), 145.

Nugent, C., Mulvenna, M., Hong, X., y Devlin, S. (2009, 08). Experiences in the development of a smart lab. *International Journal of Biomedical Engineering and Technology*, *2*. doi: 10.1504/IJBET.2009.027796

OMS. (2020). *Envejecimiento y ciclo de vida.* <https://www.who.int/ageing/about/facts/es/>. (Accedido 25 de junio de 2020)

Peek, S., Luijkx, K., Vrijhoef, H., Nieboer, M., Aarts, S., van der Voort, C., … Wouters, E. (2019). Understanding changes and stability in the long-term use of technologies by seniors who are aging in place: A dynamical framework. *BMC Geriatrics*, *19*. doi: 10.1186/s12877-019-1241-9

Peetoom, K. K. B., Lexis, M. A. S., Joore, M., Dirksen, C. D., y Witte, L. P. D. (2015). Literature review on monitoring technologies and their outcomes in independently living elderly people. *Disability and Rehabilitation: Assistive Technology*, *10*(4), 271-294. Descargado de `https://doi.org/10.3109/17483107.2014.961179` doi: 10.3109/17483107.2014.961179

Pinto, n., García-Casal, J. A., Csipke, E., Jenaro, C., y Franco, M. (2017, 01). Ict-based applications to improve social health and social participation in older adults with dementia. a systematic literature review. *Aging and mental health*, *21*, 58-65. doi: 10.1080/13607863.2016.1262818

Pérez, E. (2020). *Qué ventajas tiene la tecnología de banda ultraancha (UWB): una vieja alternativa al Bluetooth que Apple y Samsung están revitalizando.* <`https://www.xataka.com/servicios/que-ventajas-tiene-tecnologia-banda-ultrancha-uwb-vieja-alternativa-al-bluetooth-que-apple-samsung-estan-revitalizando`>. (Accedido 14 de mayo de 2020)

Quesada, F. J., Moya, F., Espinilla, M., Martínez, L., y Nugent, C. D. (2015). Feature sub-set selection for activity recognition. En A. Geissbühler, J. Demongeot, M. Mokhtari, B. Abdulrazak, y H. Aloulou (Eds.), *Inclusive smart cities and e-health* (pp. 307–312). Cham: Springer International Publishing.

Ronao, C. A., y Cho, S.-B. (2016). Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications*, *59*, 235 - 244. doi: https://doi.org/10.1016/j.eswa.2016.04.032

Sainjeon, F., Gaboury, S., y Bouchard, B. (2016). Real-time indoor localization in smart homes using ultrasound technology. En *Proceedings of the 9th acm international conference on pervasive technologies related to assistive environments.* New York, NY, USA: Association for Computing Machinery. Descargado de `https://doi.org/10.1145/2910674.2910718` doi: 10.1145/2910674.2910718

Schoenberger, C. R. (2002). The internet of things chips at the checkout counter. *Forbes*, *169*, 155-161.

Shoaib, M., Bosch, S., Scholten, H., Havinga, P. J. M., y Incel, O. D. (2015). Towards detection of bad habits by fusing smartphone and smartwatch sensors. En *2015 ieee international conference on pervasive computing and communication workshops (percom workshops)* (p. 591-596).

Song-Mi Lee, Sang Min Yoon, y Heeryon Cho. (2017). Human activity recognition from accelerometer data using convolutional neural network. En *2017 ieee international conference on big data and smart computing (bigcomp)* (p. 131-134).

Turjamaa, R., Pehkonen, A., y Kangasniemi, M. (2019, 07). How smart homes are used to support older people: An integrative review. *International Journal of Older People Nursing*, *14*. doi: 10.1111/opn.12260

Wang, G., Li, Q., Wang, L., Wang, W., Wu, M., y Liu, T. (2018, jun). Impact of Sliding Window Length in Indoor Human Motion Modes and Pose Pattern Recognition Based on Smartphone

Sensors. *Sensors (Basel, Switzerland)*, *18*(6). Descargado 2020-05-14, de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6021910/ doi: 10.3390/s18061965

Weiser, M. (1993). Hot topics-ubiquitous computing. *Computer*, *26*(10), 71-72.

Wikipedia. (2019a). *Árboles de decisión.* <https://es.wikipedia.org/wiki/Aprendizaje_basado_en_%C3%A1rboles_de_decisi%C3%B3n>. (Accedido 1 de junio de 2020)

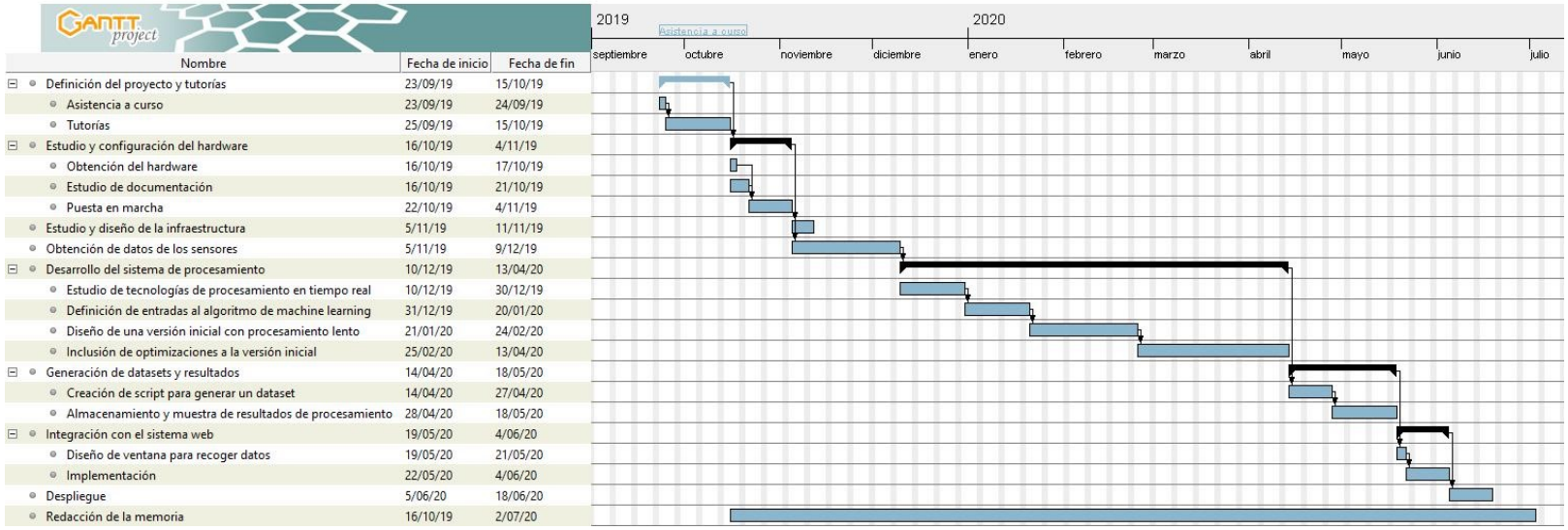
Wikipedia. (2019b). *Svm.* <https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte>. (Accedido 1 de junio de 2020)

ZWave.me. (2019). *Z-way api.* <https://zwayhomeautomation.docs.apiary.io/#>. (Accedido 17 de mayo de 2020)

9 Anexo I. Diagramas de Gantt

En este anexo se muestran los diagramas de Gantt asociados al proyecto. En primer lugar, se muestra el diagrama que muestra la planificación inicial de tareas y tiempos de duración. En segundo y último lugar se visualiza el diagrama de Gantt que contiene la ejecución de las tareas a lo largo del tiempo y su duración asociada.

9.1 Planificación



Nombre	Fecha de inicio	Fecha de fin
Definición del proyecto y tutorías	23/09/19	15/10/19
Asistencia a curso	23/09/19	24/09/19
Tutorías	25/09/19	15/10/19
Estudio y configuración del hardware	16/10/19	4/11/19
Obtención del hardware	16/10/19	17/10/19
Estudio de documentación	16/10/19	21/10/19
Puesta en marcha	22/10/19	4/11/19
Estudio y diseño de la infraestructura	5/11/19	11/11/19
Obtención de datos de los sensores	5/11/19	9/12/19
Desarrollo del sistema de procesamiento	10/12/19	13/04/20
Estudio de tecnologías de procesamiento en tiempo real	10/12/19	30/12/19
Definición de entradas al algoritmo de machine learning	31/12/19	20/01/20
Diseño de una versión inicial con procesamiento lento	21/01/20	24/02/20
Inclusión de optimizaciones a la versión inicial	25/02/20	13/04/20
Generación de datasets y resultados	14/04/20	18/05/20
Creación de script para generar un dataset	14/04/20	27/04/20
Almacenamiento y muestra de resultados de procesamiento	28/04/20	18/05/20
Integración con el sistema web	19/05/20	4/06/20
Diseño de ventana para recoger datos	19/05/20	21/05/20
Implementación	22/05/20	4/06/20
Despliegue	5/06/20	18/06/20
Redacción de la memoria	16/10/19	2/07/20

Figura 9.1: Planificación - Diagrama de Gantt

9.2 Ejecución

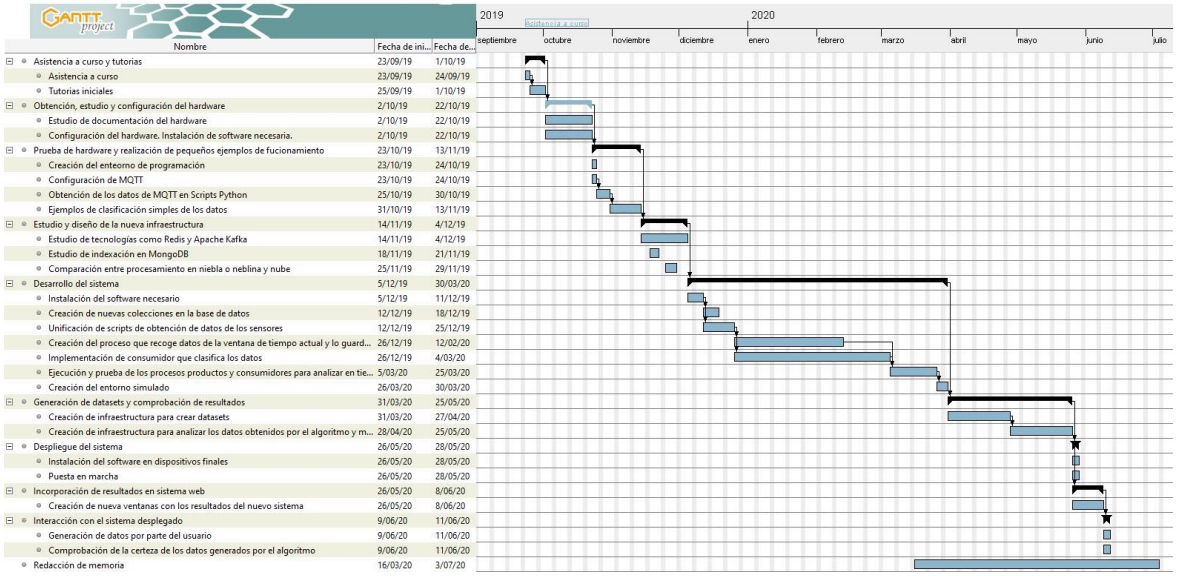


Figura 9.2: Ejecución - Diagrama de Gantt

10 Anexo II. Sistema web. Ventanas móviles.

En este anexo se muestra la versión móvil de las diferentes ventanas del sistema web.

10.1 Insertar dataset

Figura 10.1: Sistema Web. Versión móvil. Insertar dataset

10.2 Insertar actividad

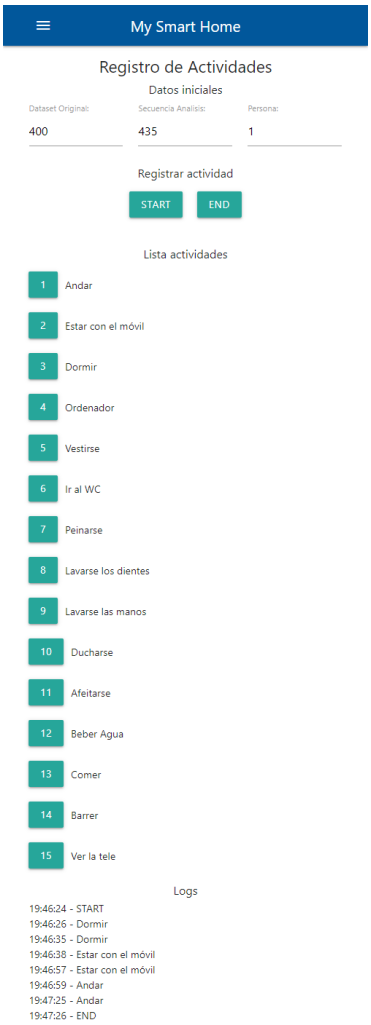


Figura 10.2: Sistema Web. Versión móvil. Insertar actividad

10.3 Lista actividades

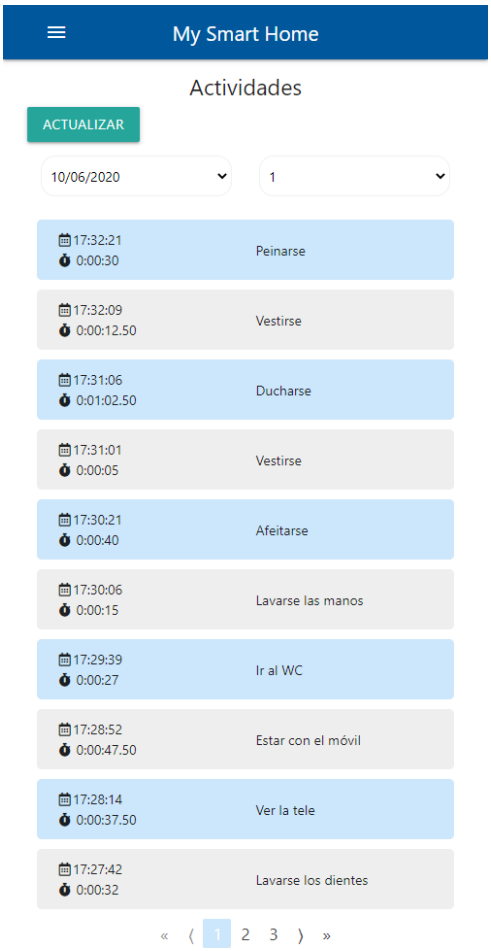


Figura 10.3: Sistema Web. Versión móvil. Lista actividades

11 Anexo III. Lista de actividades detectadas.

En este anexo se muestra la lista de las diferentes actividades que se han detectado en la realización de la secuencia final.

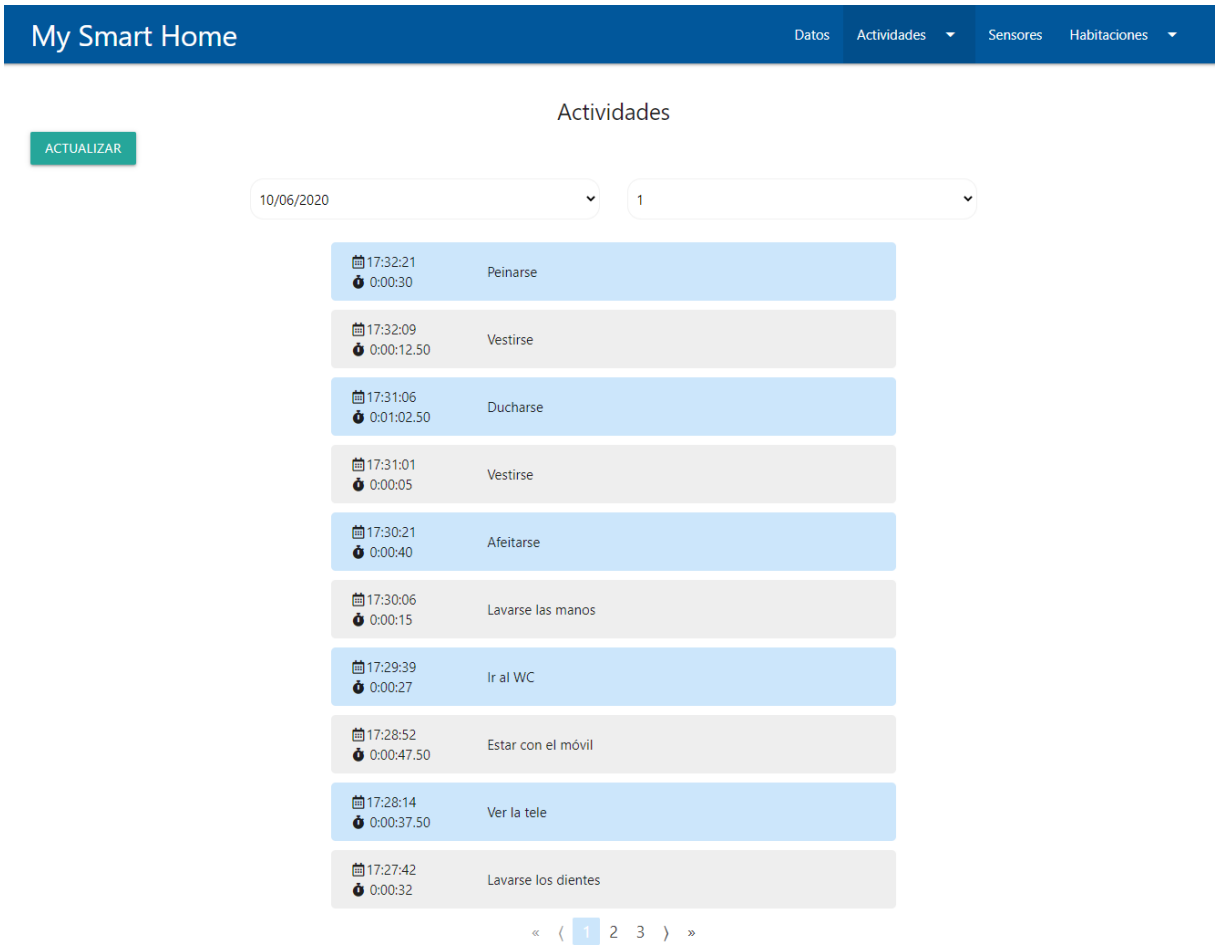


Figura 11.1: Lista actividades Página 1

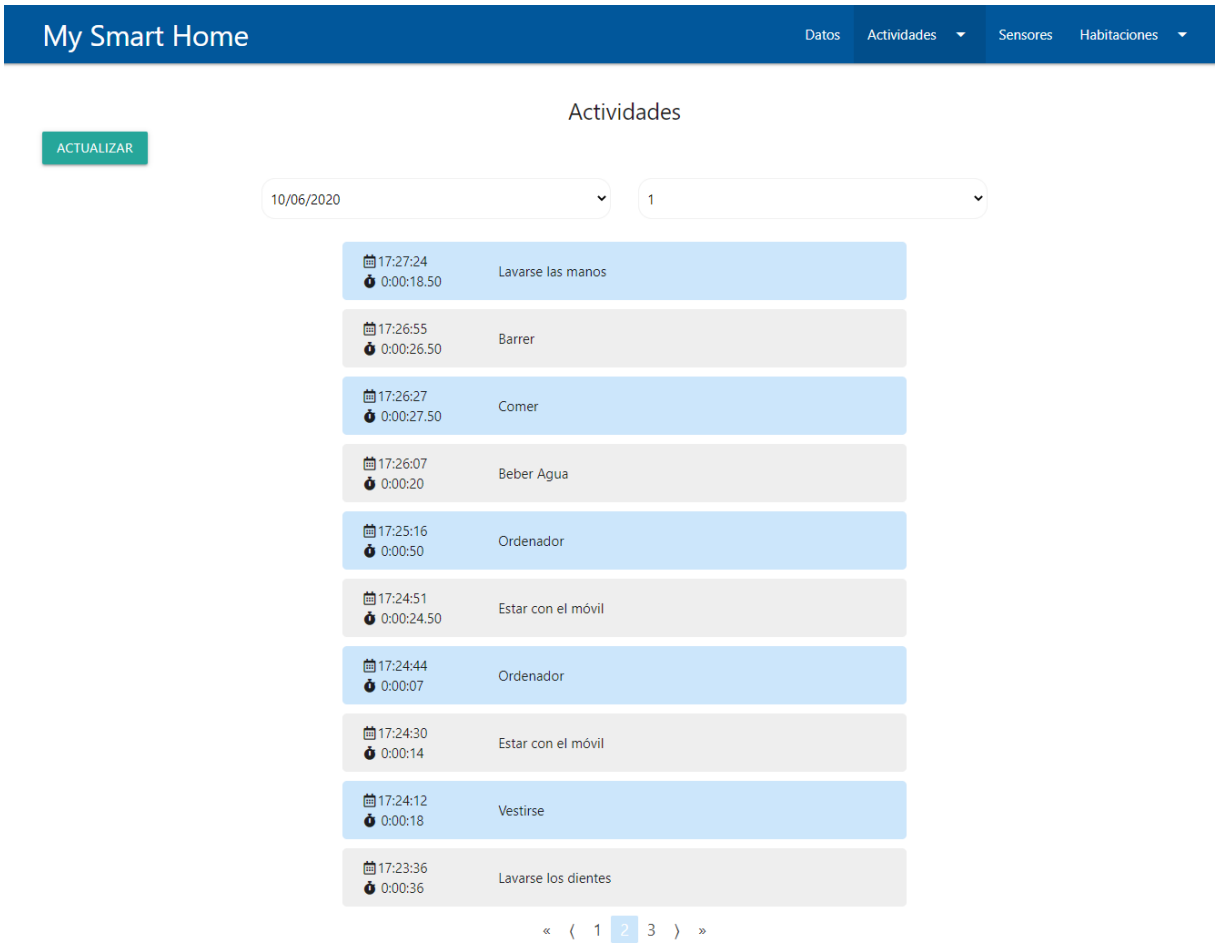


Figura 11.2: Lista actividades Página 2

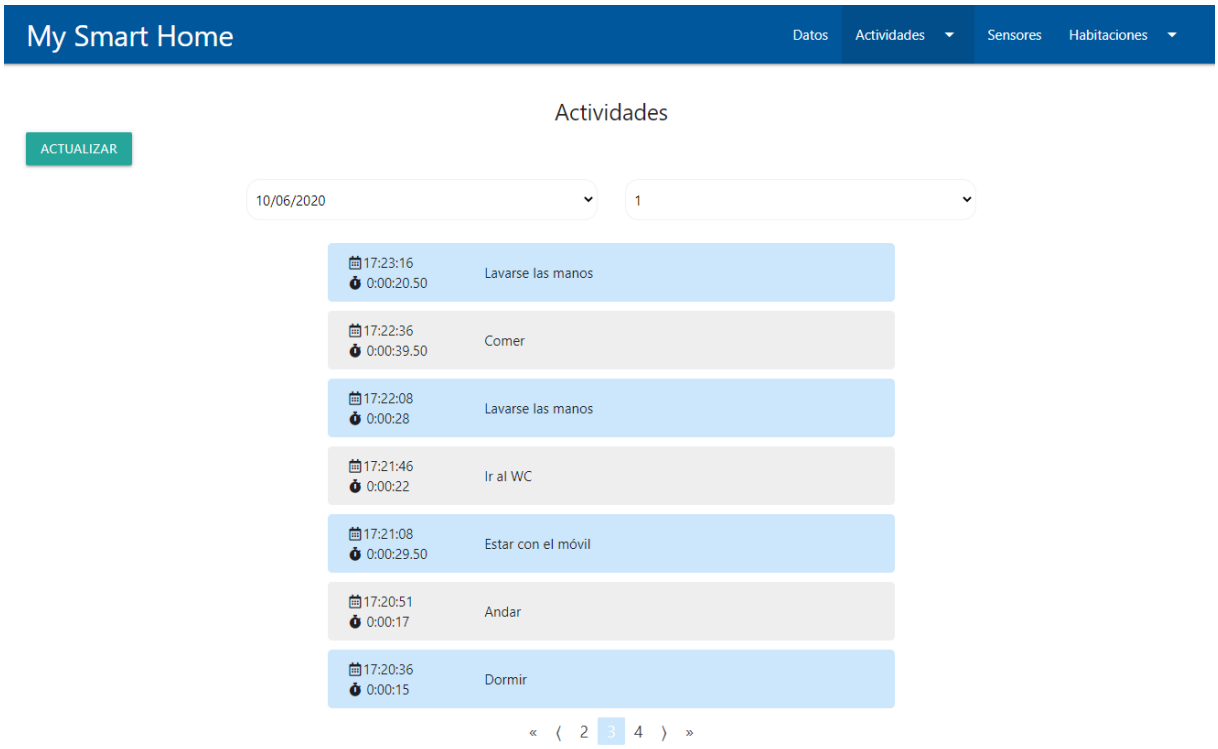


Figura 11.3: Lista actividades Página 3

Resumen/Abstract

A medida que pasan los años, la esperanza de vida va aumentando, creciendo también el número de personas mayores con necesidades de atención que viven solas. De esta forma, las tecnologías proporcionan una herramienta para permitir la independencia de estas personas de forma segura. Así, mediante el reconocimiento de actividades, se permite la monitorización de los usuarios en la vivienda, detectando posibles anomalías y cambios en el comportamiento.

En este proyecto, se propone un sistema de reconocimiento de actividades en tiempo real de los usuarios de una vivienda. Este sistema está formado por dispositivos como sensores binarios, de localización y smartwatches. Los datos que proporcionan estos, se incluyen en un modelo de *machine learning* para obtener las actividades que realizan los usuarios en la vivienda.

As the years go by, life expectancy is increasing, and the number of older people with care needs living alone is also growing. In this way, technologies provide a tool to enable these people to be independent in a safe manner. Thus, by means of activity recognition, it is possible to monitor users in the home, detecting possible anomalies and changes in behavior.

In this project, a real-time activity recognition system of the users of a house is proposed. This system is made up of devices such as binary and location sensors and smartwatches. The data provided by these are included in a machine learning model to obtain the activities carried out by the users in the house.

UNIVERSIDAD DE ALMERÍA

MÁSTER EN TECNOLOGÍAS Y APLICACIONES EN INGENIERÍA INFORMÁTICA