ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DE COMPUTADORES

Identificación de personas por su forma de andar usando sensores inerciales

Gait recognition using inertial sensors

Realizado por

Rubén Delgado Escaño

Tutorizado por

Julián Ramos Cózar

Departamento

ARQUITECTURA DE COMPUTADORES

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Julio de 2017

Fecha defensa:

El Secretario del Tribunal

Resumen: El estudio analítico sobre la forma de caminar de cada persona y de cómo puede utilizarse esto para identificar de manera inequívoca a cada individuo es lo que se tratará en el presente Trabajo de Fin de Grado (TFG). Se usarán un conjunto de sensores inerciales comúnes en el mercado, como puede ser el de un sistema IMU o un terminal móvil (Smartphone). El fin de este proyecto es el desarrollo de un algoritmo que pueda implementarse en múltiples plataformas o sistemas (con fines de seguridad), así como plantar las bases para futuros proyectos derivados de este, con fines como pueden ser la medicina, los deportes de élite u otros campos de investigación. Para la creación de este algoritmo se elegirá y trabajará en un sistema de desarrollo completo de modelos de Machine Learning y se experimentará con múltiples técnicas ya utilizadas en otros casos de uso o en el mismo. El objetivo final será el de hallar el modelo con mayor porcentaje de precisión en aciertos. A su vez, para la obtención de los datos, se ha desarrollado una aplicación móvil que recoge los mensurandos necesarios de los sensores y los sube a una base de datos para su posterior análisis.

Palabras claves: aprendizaje automático, forma de andar, sensores inerciales, identificación de personas

Abstract: The analytical study on how each person walks and how this can be used to unambiguously identify each individual is what will be studied in this final degree project. A common set of inertial sensors in the market will be used, such as an IMU system or a mobile terminal (Smartphone). The purpose of this project is the development of an algorithm that can be implemented in multiple platforms or systems (with security purposes), as well as base to the foundations for future projects derived from it, for purposes such as medicine, elite sports or other fields of research. For the creation of this algorithm a complete development system of models of Machine Learning will be chosen and worked on, as well as experimenting with multiple techniques already used in other use cases or in the same one. The final objective will be to find the model with the highest percentage accuracy in hits. In turn, to obtain the data, a mobile application has been developed that collects the necessary measurands from the sensors and uploads them to a database for further analysis.

Keywords: machine learning, gait, inertial sensors, person identification

Índice

Capítulos **Página**

1. Introducción

1.1. Estado del Arte

El aprendizaje automático tiene una amplia gama de aplicaciones, incluyendo motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, juegos y robótica.

El estudio del uso de clasificadores que se nutren de sensores inerciales directamente situados en los usuarios es algo ya ampliamente realizado en campos relacionados con la medicina, como pueden ser el reconocimiento de la actividad humana (Gadaleta y Rossi, 2006) (Laerhoven y Cakmakci, 2000), detección de caídas (Turan y Barshan, 2014), detección del Parkinson (Monsonís, Pérez, Romagosa, y Rodríguez-Molinero, 2012) o la telemonitorización de pacientes con esta enfermedad (Herrlich y cols., 2011).

En el desarrollo de modelos que permiten la identificación de personas pueden utilizarse sensores como los ya usados en las aplicaciones anteriormente nombradas. Ya existen estudios similares, cuyos resultados se comentarán en el capítulo 5:

- **IDNet: Smartphone-based Gait Recognition with Convolutional Neural Networks** (Gadaleta y Rossi, 2016). Se usa una red neuronal convolucional en el preprocesado para seleccionar los mejores predictores para el modelo de aprendizaje automático.
- **Gait Identification Using Accelerometer on Mobile Phone** (Thang, Viet, Nguyen, y Choi, 2012). Se presentan dos métodos de clasificación. En el primero se usa la *Dynamic Time Warping* (Silva y Batista, 2016), o la deformación dinámica del tiempo, para evaluar la similitud del paso entre dos secuencias que pueden variar de velocidad. El segundo método consiste en una SVM encargada de clasificar en el dominio de la frecuencia.
- **Authentication of Smartphone Users Based on the Way They Walk Using k-**

NN Algorithm (Nickel, Wirtl, y Busch, 2012). Se basa en modelos ocultos de Markov (Wikipedia, 2017) junto a clasificadores K-vecinos más cercanos.

- **Influence of Holding Smart Phone for Acceleration-Based Gait Authentication** (Watanabe, 2014). Uso de técnicas para conocer la posición del teléfono, permitiendo reconocer la identidad del usuario incluso cuando se utiliza el teléfono como dispositivo de llamada.

- **Biometric gait recognition based on wireless acceleration sensor using k-nearest neighbor classification** (Choi, Youn, LeMay, Burns, y Youn, 2014). Se proponen una serie de nuevas métricas basadas en la variación de velocidad en el tirón de la pierna al caminar.

- **Smartphone based user verification leveraging gait recognition for mobile healthcare systems** (Ren, Chen, Chuah, y Yang, 2013). Se propone un sistema de identificación dividido en tres etapas: la identificación del ciclo de pasos, la interpolación del ciclo de pasos y el cálculo de la similitud en las puntuaciones.

- **An Efficient HOS-Based Gait Authentication of Accelerometer Data** (Sprager y Juric, 2015). Los patrones de los pasos se obtienen mediante el análisis de señales utilizando estadísticos de orden superior.

1.2. Objetivos

Los objetivos que se plantearon previos a la realización del presente TFG fueron los siguientes:

- Implementación de un sistema de sensores que se fijaría al sujeto y tomaría datos inerciales de su movimiento.

- Creación de un dataset a partir de estos datos inerciales.

- Probar diversas técnicas de filtrado y de obtención de características sobre el dataset.

- Construcción de un modelo de aprendizaje automático que intentará identificar de manera inequívoca la identidad del sujeto al que pertenecen los datos introducidos en él. A diferencia de los modelos construidos en los artículos antes mencionados, se buscará un enfoque más simple en cuanto a las técnicas de preprocesamiento y de extracción de características, con el objetivo de probar una mayor variedad de técnicas y que estas sean aplicables en una implementación para dispositivos móviles.
- Comprobar la viabilidad del modelo, estudiando distintos factores que puedan influir en él (como pueden ser el tipo de calzado, pesos adicionales, uso de distintas rutas en la toma de los datos, etc), y aplicando técnicas de validación.

1.3. Estructura de la Memoria

Esta memoria se ha estructurado en capítulos que cubren distintos aspectos que han sido desarrollados o estudiados durante la realización del actual TFG. A continuación se introducen los contenidos de cada capítulo:

Capítulo 2: Tecnologías y Herramientas utilizadas

Explicación de las tecnologías y herramientas utilizadas, así como un análisis de los motivos que han llevado a su elección.

Capítulo 3: Aplicación Android

Exposición de los casos de uso y el diagrama UML de actividad de la aplicación implementada para la construcción del dataset usado para desarrollar este TFG. Adicionalmente se han incluido algunas consideraciones sobre su uso y compatibilidad.

Capítulo 4: Creación del Clasificador

Explicación del procedimiento seguido para la creación del modelo, pasando por todas las fases del proceso: creación del dataset, preprocesado, elección de predictores, entrenamiento y validación.

Capítulo 5: Análisis de Resultados

Se muestran y analizan los resultados obtenidos por los distintos modelos entrenados en distintas condiciones, y se llegan a conclusiones sobre estos. También se mostrarán los modelos finales propuestos.

Capítulo 6: Conclusiones y Líneas Futuras

Se finaliza la memoria extrayendo unas conclusiones sobre el trabajo llevado a cabo y proponiendo nuevas líneas de estudio sobre el dataset creado.

2. Tecnologías y Herramientas utilizadas

En este capítulo se analizarán las tecnologías utilizadas en el proyecto, así como una breve explicación de por qué se han usado estas herramientas y no otras de carácter similar.

2.1. Sensores

Se necesita un sistema de recolección de datos con sensores que capturen la información inercial necesaria para el propósito de este trabajo. Los sensores inerciales más comúnmente utilizados son el acelerómetro, el giroscopio y el magnetómetro.

Acelerómetro

Los acelerómetros son dispositivos electromecánicos que detectan fuerzas estáticas o dinámicas de aceleración. Dentro de las fuerzas estáticas se pueden incluir la gravedad (Figura 1), mientras que en las fuerzas dinámicas pueden incluirse vibraciones o movimiento.

Generalmente, los acelerómetros contienen placas capacitivas internas, algunas fijas y otras unidas a muelles minúsculos que se mueven internamente cuando las fuerzas de aceleración actúan sobre el sensor. A medida que estas placas se mueven entre sí, la capacitancia entre ellas cambia, permitiendo con ello determinar la aceleración. Otros acelerómetros pueden estar centrados alrededor de materiales piezoeléctricos: estas diminutas estructuras de cristal dan como salida carga eléctrica cuando se colocan bajo tensión mecánica, por ejemplo, aceleraciones.

La mayoría de los acelerómetros tendrán una gama seleccionable de fuerzas que pueden medir. Estos rangos pueden variar desde $\pm$ 1 g hasta $\pm$ 250 g. Típicamente, cuanto menor es el rango, más sensibles serán las lecturas del acelerómetro. Por ejemplo, para medir pequeñas vibraciones en una mesa, usar un acelerómetro de pequeño rango proporcionará datos más detallados que usar un rango de $\pm$ 250 g, que sería más adecuado para un cohete o algo similar.

Adicionalmente, algunos acelerómetros incluyen características tales como detección de

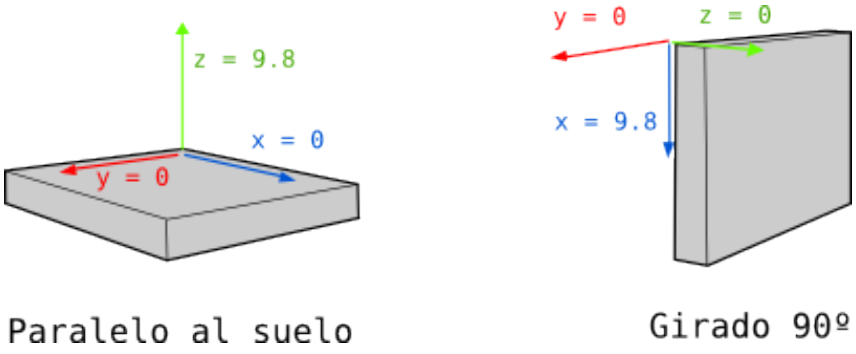


Figura 1: Acelerómetro con sus ejes

golpes, detección de caída libre (utilizada para la protección de discos duros o smartphones), la detección de golpes dobles o la corrección de temperatura.

En este caso de estudio, se necesita un acelerómetro de 3 ejes con una alta precisión y con el rango suficiente para medir los cambios en la velocidad del movimiento de las piernas al andar. Un ejemplo de sensor podría ser el acelerómetro situado en el BMI160 (*BMI160 Datasheet*, 2015), con un rango de $\pm$ 16 g y con una resolución de 0,002. Este es un modelo relativamente común en smartphones de marcas como Apple, Nexus, Sony o Xiaomi, además de en los smartphones utilizados durante el desarrollo del presente TFG.

Giroscopio

Los giroscopios son sensores pequeños y económicos que miden la velocidad angular, que es simplemente una medida de la velocidad de rotación. Estas se miden en grados por segundo (°/s) o revoluciones por segundo (RPS).

El sensor del giroscopio dentro del encapsulado es de entre 1 y 100 micrómetros. Cuando se gira el giroscopio, una pequeña masa resonante se desplaza a medida que cambia la velocidad angular (Figura 2). Este movimiento se convierte en señales eléctricas de muy baja corriente que pueden ser amplificadas y leídas.

Este tipo de sensores se pueden utilizar para determinar la orientación y se encuentran en

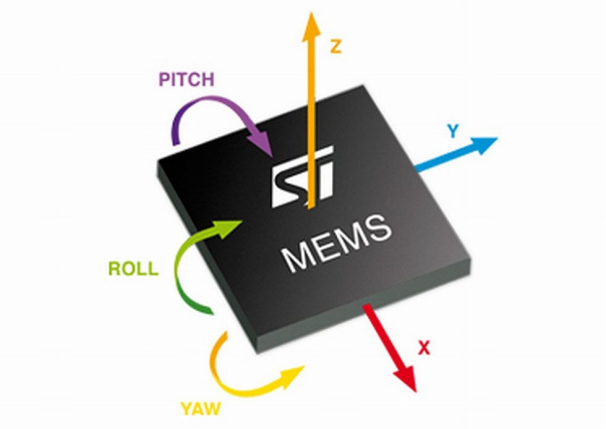


Figura 2: Giroscopio con sus ejes

la mayoría de los sistemas de navegación autónoma. Por ejemplo, si se desea equilibrar un robot se puede utilizar un giroscopio para medir la rotación desde la posición equilibrada y enviar correcciones a un motor.

Hay muchas especificaciones a considerar a la hora de elegir un tipo de giroscopio para su uso. Las más importantes vuelven a ser el rango, que es la velocidad angular máxima que el giroscopio puede leer, y la sensibilidad, que se mide en mV por grado por segundo (mV/°/s).

Igual que en el caso del acelerómetro, un giroscopio de 3 ejes con una alta precisión y un bajo rango conformará parte del sistema de recolección de datos, midiendo los cambios en la rotación de las piernas al andar. Volviendo a poner de ejemplo el BMI160 (*BMI160 Datasheet*, 2015), su giroscopio tiene un rango de $\pm$ 2000 °/s y una resolución de 0,001.

Magnetómetro

El magnetómetro es un sensor de efecto Hall en miniatura que detecta el campo magnético de la Tierra a lo largo de tres ejes perpendiculares. Este sensor produce voltaje proporcional a la fuerza y la polaridad del campo magnético a lo largo de los ejes del sensor. El voltaje detectado se convierte en señal digital que representa la intensidad del campo magnético. Otras tecnologías usadas para el magnetómetro pueden incluir dispositivos magneto-resistivos que cambian la resistencia medida al cambiar el campo magnético.

Además de la información general de rotación, el magnetómetro es crucial para detectar la orientación relativa del sensor en relación con el norte magnético de la Tierra. Es por esto que debemos descartar este sensor del sistema de recolección de datos, pues sus tomas de datos dependerán de la dirección azimutal en la que ande el sujeto y no de su forma de caminar, con lo cual esta información resultaría inútil, tanto para la identificación del mismo como para conocer la posición del móvil con respecto a este.

IMUs

Una unidad de medida inercial, o IMU, es un dispositivo electrónico que mide e indica la aceleración específica de un cuerpo, su velocidad angular y, a veces, el campo magnético que rodea al cuerpo, utilizando una combinación de acelerómetros, giroscopios y magnetómetros. Los IMU se usan típicamente en aviones, incluyendo vehículos aéreos no tripulados (UAVs), entre muchos otros, y naves espaciales, incluyendo satélites y transbordadores.

Se pueden utilizar los datos recolectados de los sensores del IMU para rastrear la posición del sistema usando navegación por estima, procedimiento matemático para calcular la ubicación actual de un dispositivo haciendo cálculos basados en el rumbo y la velocidad de movimiento a lo largo de un período, y conociéndose el punto inicial. (Furuno, s.f.). La gran desventaja al aplicar este tipo de navegación es que normalmente son afectadas por un error acumulativo debido a que el sistema de guía está continuamente agregando los cambios detectados a las posiciones previamente calculadas, provocando que cualquier error se vaya acumulando de punto a punto. Esto lleva a una 'deriva', que es la diferencia entre el punto donde el sistema piensa que se encuentra localizado y la posición real, y que crece a cada iteración.

Tiene interés para nuestro proyecto al ser un conjunto de dispositivos que contiene físicamente todos los sensores necesarios para la toma completa de los datos necesarios, pero eso sólo nos permitiría tomar capturas controladas en laboratorio, aumentando el tiempo de construcción del dataset y reduciendo el número potencial de candidatos. Esto ya fue propuesto

anteriormente, con un diseño propio de un sistema hardware centrado en el reconocimiento de actividades (Choudhury y cols., 2008).

Con el fin de facilitar el uso de un sistema de estas características, en vez de utilizar un IMU convencional se hará uso de un smartphone, que comúnmente tienen los tres sensores anteriormente nombrados, permitiendo que cualquiera pueda utilizar su dispositivo para contribuir a la construcción del dataset.

2.2. Android

Con casi un 88 % de cuota de mercado mundial a fecha de Noviembre de 2016, Android es el principal sistema operativo para teléfonos móviles. Además, una de sus mejores características es que el desarrollo en esta plataforma carece de costes adicionales por licencia (a excepción de que se quiera posicionar la app desarrollada en la tienda oficial). Esto lo hace muy popular entre desarrolladores, ya que los costes para lanzar una aplicación son muy bajos.

Por tanto, se utilizará una aplicación Android (*Documentación Android*, s.f.) de creación propia para capturar los datos. Esta decisión viene de la necesidad de hacer la aplicación accesible y universal para poder construir un dataset lo más grande posible.

Por desgracia utilizar Android también implica aceptar una serie de puntos negativos, entre ellos la gran fragmentación en especificaciones de los dispositivos móviles y la incompatibilidad de aplicaciones producidas por las actualizaciones del sistema.

Como entorno de desarrollo integrado (IDE) se utilizará Android Studio (*Documentación Android Studio*, s.f.), IDE oficial para el desarrollo de aplicaciones para Android y que se basa en IntelliJ IDEA, IDE de Java, (*Documentación IntelliJ IDEA*, s.f.). Éste incluye un emulador de dispositivos Android y un depurador USB para cargar directamente la aplicación en desarrollo a un dispositivo real.

2.3. Machine Learning

Parte del objetivo final de este TFG es la creación de un modelo de Machine Learning capaz de identificar de manera inequívoca la identidad del sujeto al que pertenecen los datos analizados.

El Machine Learning, o aprendizaje automático, es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras identificar patrones complejos en millones de datos, usando un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros.

La principal diferencia de esta técnica con respecto a otras es que, al igual que los modelos estadísticos, el objetivo es entender la estructura de los datos y ajustar las distribuciones teóricas a estos. Por lo tanto, con los modelos estadísticos hay una teoría detrás del modelo que está demostrado matemáticamente, pero esto requiere que los datos cumplan ciertos supuestos fuertes también. El aprendizaje automático se ha desarrollado sobre la base de la capacidad de utilizar computadoras para investigar la estructura de los datos, incluso si no tenemos una teoría de cómo se ve esa estructura. Debido además a que el aprendizaje automático a menudo utiliza un enfoque iterativo para aprender de los datos, el aprendizaje puede automatizarse fácilmente.

El aprendizaje automático se utiliza en motores de búsqueda, diagnósticos médicos, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, robótica, etc.

Hay múltiples herramientas para el desarrollo con Machine Learning, pensadas para distintos lenguajes de programación, como son:

- C: **Shark** (*Documentación Shark*, s.f.) o **LibSVM** (*Documentación LibSVM*, s.f.).
- Python: **scikit-learn** (*Documentación scikit-learn*, s.f.) o **TensorFlow** (*Documentación TensorFlow*, s.f.).
- Java: **Weka** (*Documentación Weka*, s.f.).

2.4. MATLAB

MATLAB es un entorno de desarrollo y un lenguaje de programación de cuarta generación pensados para el desarrollo de software matemático, ampliamente utilizado y con una gran cantidad de librerías y toolboxes (conjunto de funciones y herramientas con una temática común). MATLAB es muy popular en el ambiente académico debido a su habilidad para operacionalizar expresiones matemáticas complejas, poseer un soporte rico para álgebra y cálculo, y permitir la computación simbólica. A menudo se utiliza para la creación de prototipos de nuevos modelos de Machine Learning y, en algunos casos, para producir soluciones completas.

Como librería de desarrollo para el modelo se utiliza la toolbox de MATLAB (*Documentación MATLAB*, s.f.) de Machine Learning **Statistics and Machine Learning Toolbox** (*Documentación Statistics and Machine Learning Toolbox*, s.f.), más concretamente sus funciones y herramientas destinadas a la creación y utilización de los modelos de clasificación, como la aplicación *Classification Learner* que proporciona una interfaz gráfica en la que entrenar modelos con distintos clasificadores personalizables.

Aunque MATLAB es de por si un lenguaje de programación propio, este puede exportarse a C gracias a MATLAB Code, una herramienta perteneciente al propio IDE de MATLAB. Gracias a esto podría desplegarse el modelo final en otro sistema, sin la necesidad de tener en éste instalado ni MATLAB ni MATLAB Runtime (conjunto independiente de bibliotecas que permite la ejecución de aplicaciones o componentes de MATLAB en ordenadores que no tienen instalado MATLAB).

3. Aplicación Android

Se ha creado una aplicación Android para recolectar los datos necesarios para la construcción y validación del modelo, aprovechando los sensores que traen los smartphones actuales. Esta aplicación se ha creado con la idea de que el sujeto pueda utilizarla de forma autónoma en un entorno real y sin supervisión, dando la posibilidad adicional de subir los datos capturados a un servidor perteneciente al departamento de Arquitectura de Computadores, junto a un pequeño cuestionario que proporcionará información sobre el sujeto y su experimento.

La toma de datos se hará muestreando 4 sensores a una frecuencia de 50Hz: acelerómetro, acelerómetro lineal, sensor de gravedad y giroscopio. Estos datos se almacenarán en raw, o en 'crudo', en la memoria principal del smartphone sin realizar ningún tipo de filtrado adicional. Esto último se ha decidido para que el dataset construido pueda ser aprovechable por otros proyectos relacionados con la motricidad humana. Se dan más detalles sobre la frecuencia de muestreo y el uso de estos datos capturados en el capítulo siguiente, apartado 4.1.

3.1. Casos de Uso

Se procede a la especificación de los posibles escenarios.

Captura de Datos

Descripción: Realiza una captura de datos de 128 segundos.

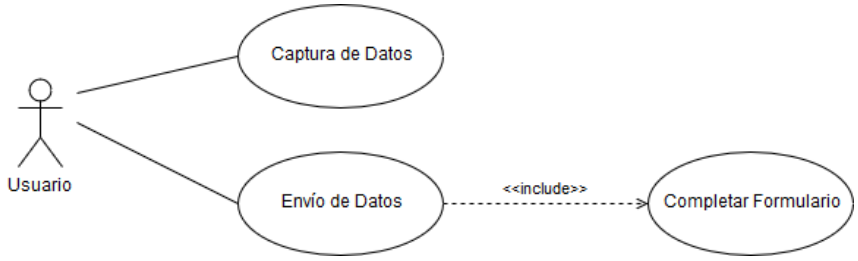


Figura 3: Casos de Uso APP Android

Precondiciones:

- La aplicación debe tener permisos de escritura en almacenamiento.
- El dispositivo debe estar en modo avión.
- La aplicación no debe estar realizando actualmente otra captura de datos.
- El dispositivo debe disponer de todos los sensores necesarios.

Escenario principal de éxito:

1. El usuario abre la aplicación.
2. La aplicación le muestra al usuario la interfaz de captura de datos.
3. El usuario pulsa el botón de inicio de la captura.
4. La aplicación verifica que posee permisos de escritura en almacenamiento.
5. La aplicación verifica que el dispositivo tiene el modo avión activado.
6. La aplicación notifica al usuario que comenzará la captura tras una cuenta regresiva.
7. La aplicación lanza una notificación al finalizar la cuenta regresiva avisando del comienzo de la captura de datos.
8. La aplicación realiza la captura de datos. Para que esta sea válida con respecto al conjunto de datos que se busca, el usuario debe estar andando de forma constante.
9. La aplicación lanza una notificación avisando del final de la captura de datos tras tomar todas las muestras.

Envío de Datos

Descripción: Enviar las capturas de datos almacenadas.
Precondiciones:

- El dispositivo debe no estar en modo avión.

- La aplicación no debe estar realizando actualmente una captura de datos.
- El dispositivo tiene actualmente almacenados 4 o más capturas de datos.

Escenario principal de éxito:

1. El usuario abre la aplicación.
2. La aplicación le muestra al usuario la interfaz de captura de datos.
3. El usuario pulsa el botón de inicio de envío de las capturas.
4. La aplicación le muestra al usuario la interfaz de envío de datos.
5. El usuario rellena todos los campos obligatorios del formulario.
6. El usuario pulsa el botón de envío.
7. La aplicación inicia el envío de los datos.
8. La aplicación notifica al usuario el fin del proceso de envío.
9. La aplicación le muestra al usuario la interfaz de captura de datos.

3.2. Diagrama de Actividad

El siguiente gráfico, Figura 4, representa el diagrama de actividad de la aplicación desarrollada.

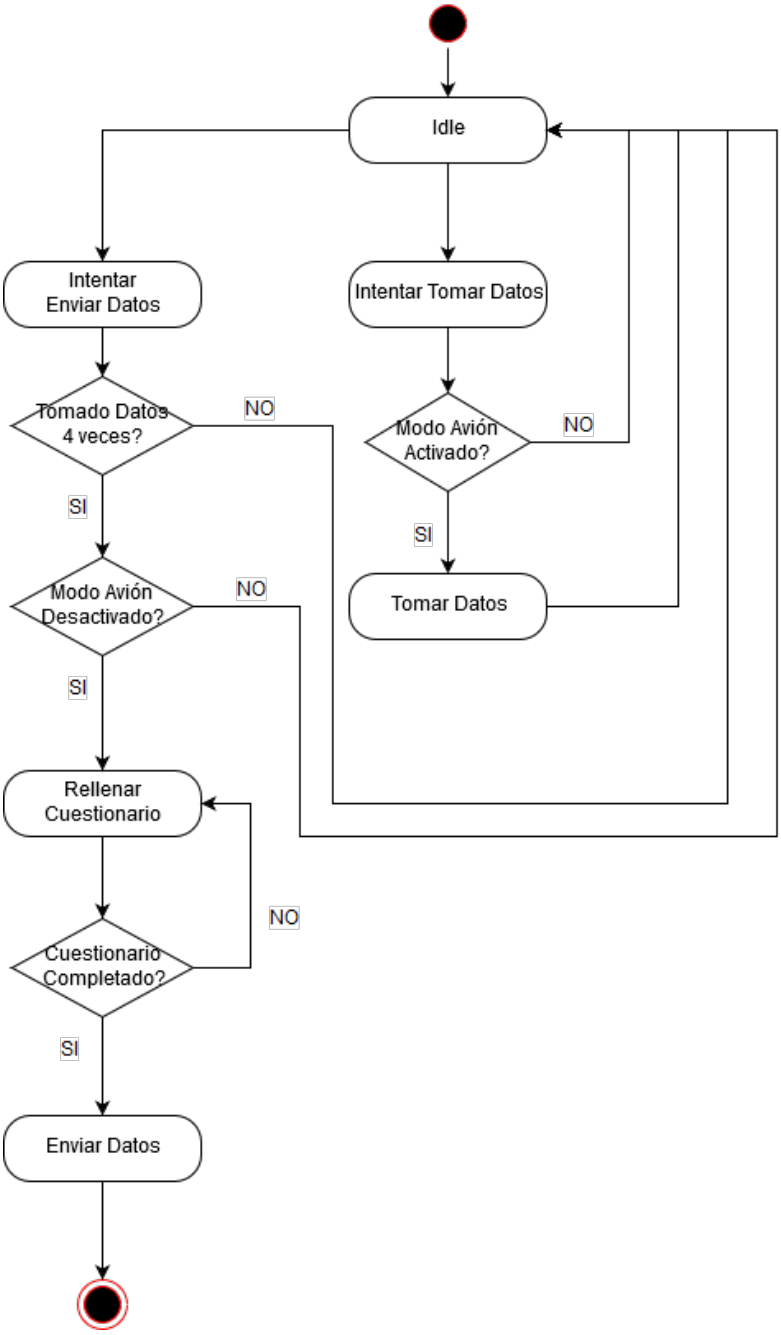


Figura 4: Diagrama de Actividad APP Android

3.3. Implementación

En éste apartado se mostrará la implementación de la aplicación. Esta ha sido dividida en dos actividades diferenciadas según su caso de uso: MainActivity y SendActivity, que se soportan en clases auxiliares para cumplir sus cometidos. La relación entre ellas y otros elementos de la aplicación pueden verse en la Figura 5.

Captura de Datos

Se ha creado una actividad Android (*Documentación Android: Activity*, s.f.) llamada MainActivity para contener el menú principal de la app, que permite iniciar la captura de datos o llamar a una segunda actividad encargada del envío de los datos al servidor, SendActivity.

Para tomar los datos de los sensores se ha implementado un servicio Android que se ejecuta en segundo plano, que se encarga de inicializar y de atender los eventos generados por estos. Para ello se ha usado un objeto *SensorManager* (*Documentación Android: SensorManager*, s.f.) encargado de proporcionar el acceso a los distintos sensores del smartphone. Usando la función *getDefaultSensor* se obtiene una instancia del sensor deseado, objeto *Sensor* (*Documentación Android: Sensor*, s.f.), y éste se utiliza para registrar un *callback* de sensores que será el encargado de atender la lectura de éstos y la escritura en el archivo correspondiente. Adicionalmente, el *callback* genera un thread para que la lectura del sensor y la escritura

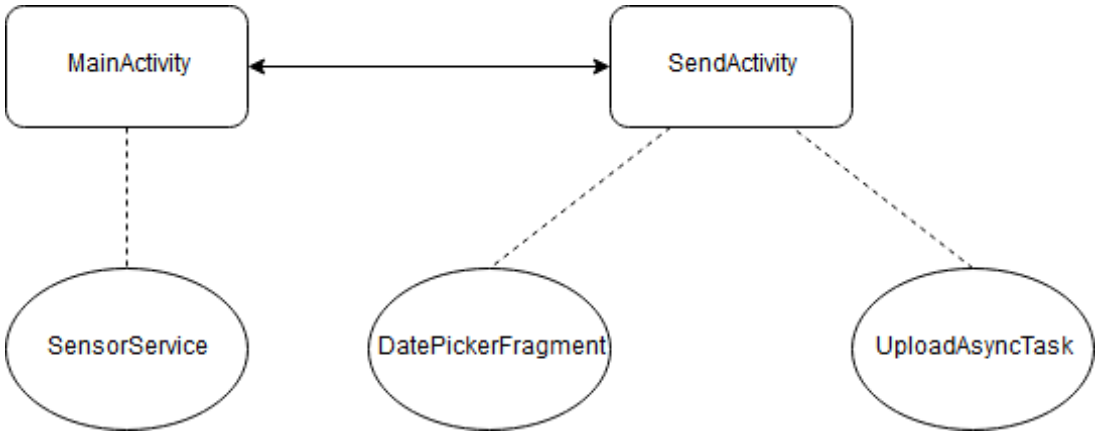


Figura 5: Implementación de la APP Android

del archivo se hagan en él, liberando lo antes posible el propio *callback*. Dentro del thread se ha implementado un bucle de control que da al usuario un tiempo inicial para ajustar su dispositivo a la posición correcta, así como para dar tiempo a los sensores a iniciarse y realizar una calibración interna inicial.

Se tomarán muestras de 4 sensores a una frecuencia de 50Hz (20 milisegundos): *Accelerometer*, *Gyroscope*, *Gravity* y *Linear Acceleration* (*Documentación Android: Motion Sensor*, s.f.). De los sensores anteriores, solo los dos primeros serán los únicos obligatorios a tener por el dispositivo, y solo esos dos serán los usados para comprobar si se han tomado todas las muestras necesarias para finalizar a captura. Se ha tomado esta decisión de diseño debido a que los sensores *Gravity* y *Linear Acceleration* no están implementados en todos los dispositivos Android, al depender estos del fabricante del smartphone y no del sistema operativo.

Estas capturas se almacenan en archivos independientes, junto a una marca de tiempo obtenida con la función *elapsedRealtime*, de la clase del sistema *SystemClock* (*Documentación Android: SystemClock*, s.f.).

El inicio y fin de la toma de capturas hace uso de un sistema de avisos para el usuario, notificación Android, implementado con un objeto de la clase *NotificactionCompact.Builder* y enviado al gestor de servicios del sistema operativo a través de un objeto de la clase *NotificationManager*.

Envío de Datos al Servidor

La actividad SendActivity contiene un formulario a completar por el usuario y que será enviado junto a los datos al servidor. Este formulario hace uso de un DialogFragment (*Documentación Android: DialogFragment*, s.f.) que genera un calendario para la selección de fechas.

Una vez van a enviarse los datos esta actividad crea un AsyncTask (*Documentación An-*

droid: AsynTask, s.f.) que será la que envíe los datos al servidor a través de una conexión segura por SFTP, usando la librería *JSch* (*Documentación JSch*, s.f.). Para ello se inicia una sesión SSH usando la función *getSession*, a la que se hace conexión para, posteriormente, abrir un canal del tipo SFTP con la función *openChannel*, sobre la que se podrá enviar el archivo al servidor.

El archivo es comprimido previamente en extensión *zip* con la librería *java.util.zip* (Mahmoud, 2002), y contendrá los archivos con la información de los sensores (4 por cada captura de datos realizada) y un archivo adicional con la información rellenada por el usuario en la encuesta.

3.4. Consideraciones Adicionales

La aplicación no es compatible con Android Lollipop o anteriores debido a la gestión del ahorro energético y de servicios de esas versiones de Android. El problema principal radica en que el sistema operativo protege la autonomía del dispositivo apagando servicios que no considera necesarios cuando se apaga la pantalla del smartphone y este no está conectado a una fuente de alimentación.

Adicionalmente, se han detectado problemas relacionados con la frecuencia de muestreo y la sincronización de los sensores dependientes del dispositivos, debidos a la implementación de los drivers de los sensores que implemente el fabricante del dispositivo.

4. Creación del Clasificador

En esta capítulo se expondrá todo el proceso de creación (Figura 6) de los distintos clasificadores estudiados, desde la definición del dataset hasta los métodos de validación para los modelos finales.

4.1. Creación del Dataset

El dataset utilizado para este TFG contiene muestras de 20 personas, tomadas con 2 móviles distintos, con una edad comprendida entre los 18 y los 24 años. Ninguna de estas personas tiene alguna disfunción o problema motriz, por lo que podemos considerar que es una población estándar apta para generalizar un modelo de identificación a partir de la forma de andar.

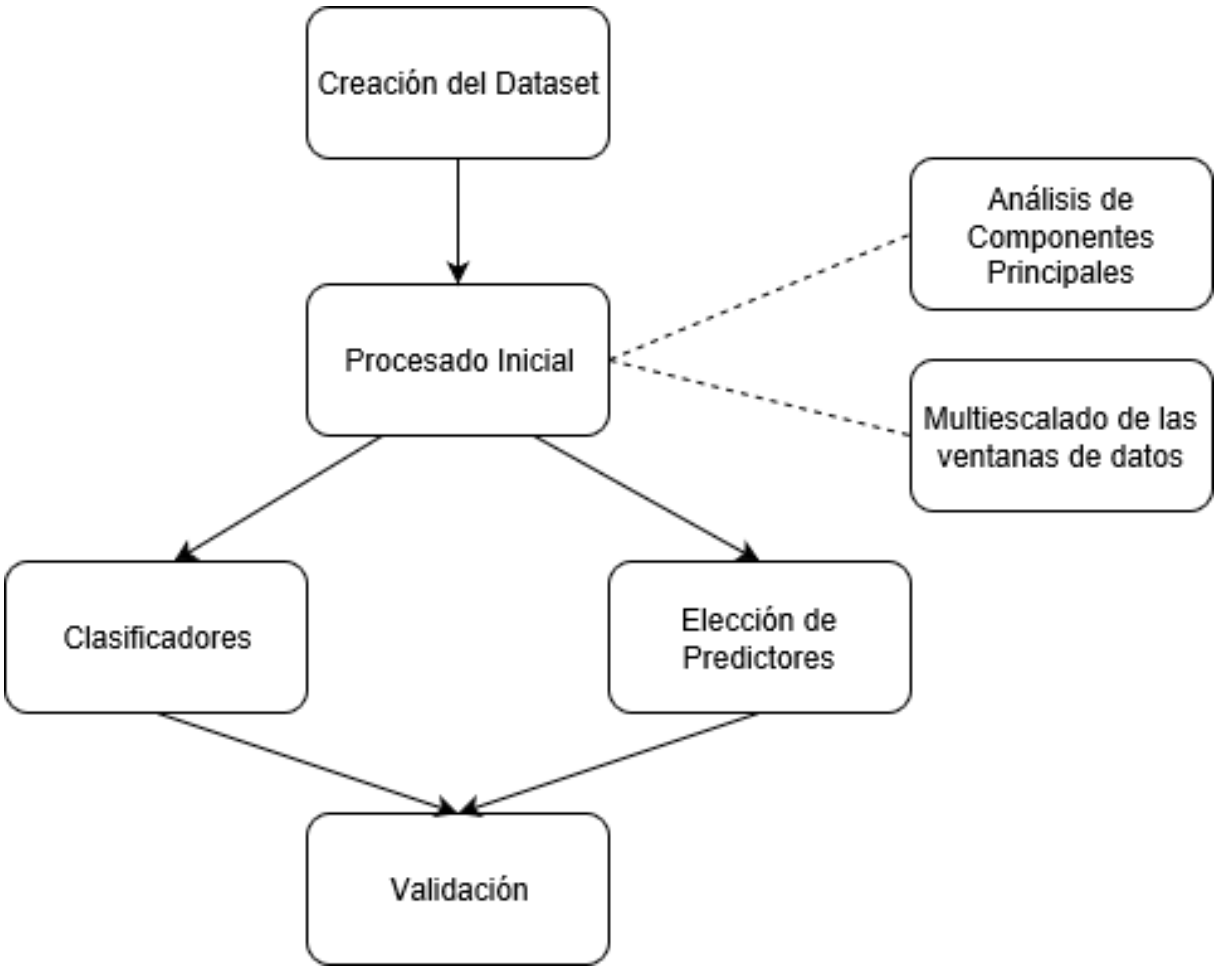


Figura 6: Proceso de creación del clasificador

Para cada sujeto se han tomado 4 capturas de muestras con la aplicación Android creada para este cometido. Cada una de estas 4 capturas han sido tomadas con el sujeto andando de forma constante, en una zona medianamente llana (sin grandes desniveles ni escalones o similares) y con el dispositivo situado en el bolsillo delantero izquierdo del pantalón, con la pantalla pegada hacia la pierna y con la zona superior del dispositivo hacia arriba. Existen estudios que proponen técnicas para evitar que la posición del dispositivo no sea relevante de cara a los resultados finales, pero aquí no se aplicarán con el objetivo de simplificar el problema planteado (Khan, Lee, Lee, y Kim, 2010). Para introducir algunas variaciones en los experimentos, las localizaciones y recorridos de estos han ido variando. Adicionalmente, algunos sujetos no han realizado todas sus capturas el mismo día, creando una variación de calzado, vestimenta, peso adicional, etc.

La duración de cada captura es de 128 segundos, y contiene 6400 muestras por cada eje de cada sensor, 50 ventanas de 128 datos de las que, al aplicarle el solapamiento del 50 %, se obtienen 99 ventanas completas. Estas muestras son tomadas a una frecuencia de 50Hz, suficiente para capturar el movimiento en el cuerpo humano (Karantonis, Narayanan, Mathie, Lovell, y Celler, 2006).

Tres de las cuatro capturas tomadas para cada sujeto se utilizarán durante la fase de entrenamiento con validación. La cuarta muestra se usará en un test posterior para medir la precisión predictiva de los modelos.

4.2. Procesado inicial

Aunque para la creación del modelo solo se van a utilizar dos sensores (Aceleración Lineal y Giroscopio), en estos experimentos se capturan cuatro sensores (añadiendo la Aceleración con el componente de gravedad y la Gravedad como un vector independiente). La explicación para esto es que no todos los dispositivos móviles disponen de la Aceleración Lineal, al depender esto del fabricante del dispositivo y no de Android. Para solucionarlo se puede utilizar la Ace-

leración con el componente de gravedad y el vector de Gravedad independiente para obtenerlo de una manera sencilla. Si no, utilizando un filtro de paso alto con la Aceleración con el componente de la gravedad y el Giroscopio también podría obtenerse. Esta última es la técnica que realmente se utiliza en el dispositivo móvil para obtener este sensor (*Documentación Android: SensorEvent*, s.f.). Ambos métodos se aplicarían en un procesado posterior a la captura, y no durante ésta. En el dataset utilizado para el desarrollo de este TFG no ha sido necesario aplicar estas técnicas para completar los datos.

Para el procesado de los 6 subconjuntos de datos o señales (obtenidas de los 3 ejes pertenecientes a los sensores de Aceleración Lineal y Giroscopio) éstos se concatenarán, de forma individual para cada señal, en ventanas deslizantes de 128 muestras (2.56 segundos) con un 50 % de solapamiento (los 64 últimos datos de una ventana serán los 64 primeros de la siguiente) (Anguita, Ghio, Oneto, Parra, y Reyes-Ortiz, 2012). Por un lado, y teniendo en cuenta la velocidad media de una persona sana al andar, el tiempo de la ventana es suficiente para analizar un ciclo completo de movimiento (Figura 7), y por otro lado el solapamiento permitirá (desde un punto de vista analítico-matemático) relacionar cada movimiento con el movimiento anterior y el siguiente.

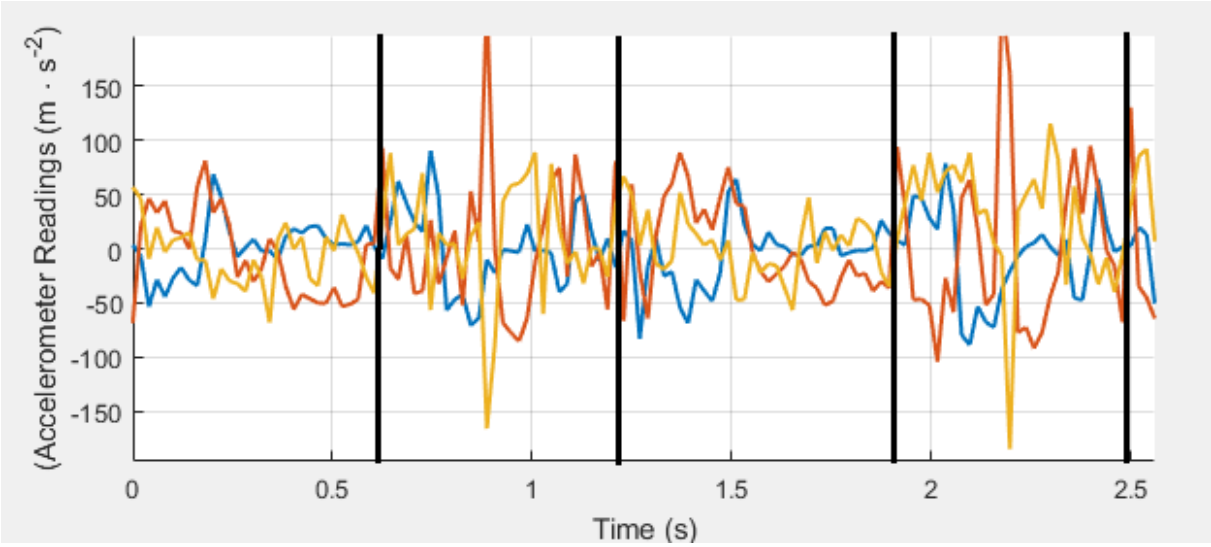


Figura 7: Múltiples ciclos completos de movimiento en una ventana de 128 muestras. Azul: eje X, naranja: eje Y, amarillo: eje Z

4.3. Clasificadores

Se va a construir un sistema de clasificación capaz de predecir una categoría (en este caso de uso, cada categoría será un sujeto). Este es un sistema perteneciente a la rama de aprendizaje supervisado en Machine Learning, lo que quiere decir que realizará predicciones a futuro basadas en comportamientos o características que se han visto en los datos ya almacenados.

Para ello se trabajaran con todos los *learners* ya implementados en la toolbox de MATLAB sobre la que se apoya el TFG, comparando cual responde mejor al conjunto de datos dependiendo de las características utilizadas. Se debe aclarar que, aunque estos son creadores de clasificadores configurados por la propia toolbox, se podrían modificar estos (por ejemplo para variar los criterios de ramificación en los árboles de decisión, o para variar la escala del kernel en las máquinas de soporte vectorial). Aun así se ha decidido usar los clasificadores ya configurados para obtener los modelos más genéricos posibles y evitar el sobreentrenamiento en estos, además de poder usar una mayor variedad de tipos de modelos al no tener que invertir un tiempo adicional en su configuración.

Árboles de Decisión

Los árboles de decisión son sencillos de interpretar, con muy bajos tiempos en entrenamiento y predicción y con requisitos bajos de memoria. Por otro lado tienen una baja precisión predictiva, y sufren sobreentrenamiento fácilmente conforme aumentan el número de ramificaciones.

Durante el desarrollo se han trabajado con tres variedades de este modelo, definidos por la propia toolbox y cuyas diferencias residen solo en el número máximo de 'divisiones' o ramificaciones: **Simple Tree** (máximo 4), **Medium Tree** (máximo 20) y **Complex Tree** (máximo 100).

Análisis Discriminante

Es un algoritmo popular y suele ser de los primeros clasificadores a probar. Es rápido, preciso, fácil de interpretar y bueno para conjuntos de datos amplios. El análisis discriminante asume que las diferentes clases generan datos basados en una distribución gaussiana diferente. Para formar un clasificador, la función de ajuste estima los parámetros de una distribución gaussiana para cada clase.

Se ha probado un modelo denominado por la toolbox como **Lineal Discriminant**, que crea fronteras lineales, y otro denominado **Quadratic Discriminant**, que a costa de un mayor consumo de memoria crea fronteras no lineales (elipses, hipérbolas y parábolas).

Máquinas de Soporte Vectorial

Una Máquina de Soporte Vectorial clasifica los datos encontrando el mejor hiperplano que separa los puntos de datos de una clase de los de la otra clase. El mejor hiperplano para una SVM es el que tiene la mayor anchura mínima entre las dos clases. Son modelos lentos comparados con los anteriores y con necesidades mayores en memoria. Los modelos ya configurados proporcionados por la herramienta son:

- **Linear SVM**: Separa las clases con una función lineal.
- **Quadratic SVM**: Separa las clases con una función polinómica no lineal de segundo orden.
- **Cubic SVM**: Separa las clases con una función polinómico no lineal de tercer orden.
- **Fine Gaussian SVM**: Separa las clases con una distribución gaussiana, usando un kernel con una escala igual a $\frac{\sqrt{P}}{4}$, siendo P el número de predictores.
- **Medium Gaussian SVM**: Separa las clases con una distribución gaussiana, usando un kernel con una escala igual a $\sqrt{P}$, siendo P el número de predictores.
- **Coarse Gaussian SVM**: Separa las clases con una distribución gaussiana, usando un kernel con una escala igual a $\sqrt{P} * 4$, siendo P el número de predictores.

K-Vecinos más Cercanos

Este clasificador se basa en la distancia a puntos (o vecinos) en un conjunto de datos de entrenamiento: dado un conjunto puntos y una función de distancia se hace una búsqueda de K vecinos más cercanos. Puede ser una manera sencilla pero eficaz de clasificar nuevos datos, utilizando distintas métricas para determinar la distancia. Los modelos ya configurados proporcionados por la herramienta son:

- **Fine KNN**: Distancia euclídea y $K = 1$.
- **Medium KNN**: Distancia euclídea y $K = 10$.
- **Coarse KNN**: Distancia euclídea y $K = 100$.
- **Cosine KNN**: La distancia en este caso es uno menos el coseno del ángulo entre las observaciones (tratadas como vectores), y con $K = 10$.
- **Cubic KNN**: Distancia Minkowski de exponente 2 y $K = 10$.
- **Weighted KNN**: Se usa como medida del parecido la inversa de la distancia euclídea al cuadrado, con $K = 10$.

Conjunto de Clasificadores

También se ha propuesto el concepto de combinar clasificadores como una nueva dirección para la mejora del rendimiento de los clasificadores individuales. Estos clasificadores podrían basarse en una variedad de metodologías de clasificación, y podrían alcanzar una tasa diferente de individuos correctamente clasificados. El objetivo de esto es generar resultados más seguros, precisos y certeros.

Existen numerosos métodos para el agrupamiento de clasificadores: usar diferentes subconjuntos de datos de entrenamiento con un único método de aprendizaje, utilizar diferentes parámetros de entrenamiento con un solo método de entrenamiento, o usar diferentes métodos de aprendizaje de forma conjunta. En el trabajo presente se ha utilizado esta última técnica. Los modelos ya configurados proporcionados por la herramienta son:

- **Boosted Trees**: Genera múltiples árboles de decisión de tamaño medio (máximo 20 ramificaciones) combinados con el algoritmo *AdaBoost*, en el que la salida de los árboles de decisión se combina en una suma ponderada que representa la salida final del clasificador mejorado. El algoritmo *AdaBoost* ajusta la salida de los árboles clasificadores de forma que los árboles posteriores son ajustados en favor de aquellas instancias mal clasificadas por los árboles anteriores. Aunque *AdaBoost* es más sensible a datos ruidosos y valores atípicos, en algunos problemas puede ser menos susceptible al sobreentrenamiento comparado con otras técnicas.

- **Bagged Trees**: Genera múltiples árboles de decisión complejos (máximo 100 ramificaciones) combinados con el algoritmo de agregación de *bootstrap* (o *Bagging*). Partiendo de un conjunto de entrenamiento, y mediante la extracción aleatoria con reemplazamiento con el mismo número de elementos que el conjunto original de elementos, se obtienen B muestras *bootstrap* (cada una del mismo tamaño que el conjunto de entrenamiento inicial). Con cada una de estas muestras *bootstrap* se entrenará un árbol de decisión distinto, y estos se combinarán usando la clase predicha más frecuente. Esta técnica es útil cuando el conjunto de entrenamiento presente observaciones ruidosas debido a que en algunas de estas muestras se habrán eliminado, o al menos reducido, la presencia de observaciones ruidosas, por lo que el clasificador construido en ese conjunto presentará un mejor comportamiento que el clasificador construido en el conjunto original.

- **Subspace Discriminant**: Combina clasificadores discriminantes usando el algoritmo de subespacios aleatorios. Este método es similar al *bagging*, a excepción de que las características se muestrean al azar, con reemplazo, para la creación de cada clasificador. Esto provoca que los clasificadores no se centren demasiado en las características que parecen altamente predictivas en el conjunto de entrenamiento, pero no son tan predictivas para los puntos fuera de ese conjunto. Por esta razón, el algoritmo de subespacios aleatorios es una opción atractiva para problemas donde el número de características es mucho mayor que el número de puntos de entrenamiento.

- **Subspace KNN**: Misma mecánica que el modelo *Subspace Discriminant*, pero utili-

zando clasificadores basados en la búsqueda de K vecinos más cercanos en lugar los clasificadores discriminantes.

- **RUSBoosted Trees**: Adición del algoritmo RUS al modelo *Boosted Trees*. El algoritmo RUS (*Random UnderSampling*) simplemente elimina ejemplos de la clase mayoritaria al azar, hasta que se consiga la distribución de clase deseada. Después se aplica el algoritmo *AdaBoost* y se combinan múltiples árboles de decisión de tamaño medio, tal y como se explicó anteriormente.

4.4. Elección de Predictores

Para entrenar y realizar a posteriori predicciones se necesitan un conjunto de características o atributos que describan cada una de las instancias del conjunto de datos. En este caso se necesitan características capaces de describir una señal cuyo origen es un sensor físico. Para ello se han elegido las siguientes operaciones, todas aplicadas a las ventanas de 128 datos construidas anteriormente (Figuras 8, 9, 10 y 11) y de forma individual, y que dan un resultado unitario:

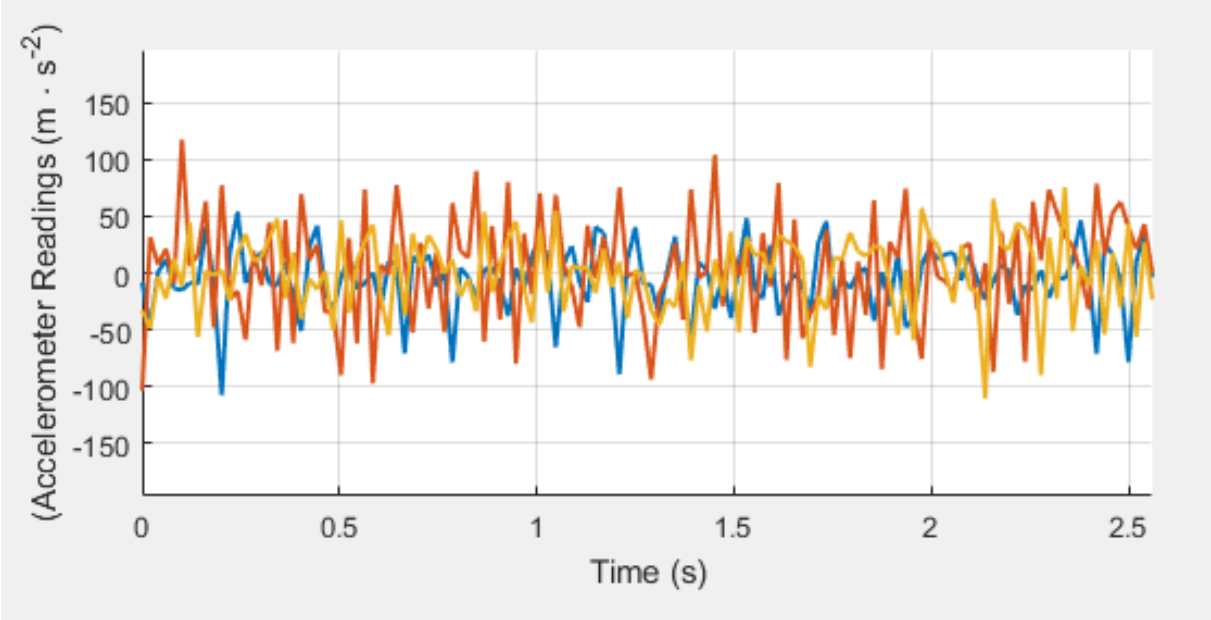


Figura 8: Señales del acelerómetro en el dominio del tiempo de una ventana. Azul: eje X, naranja: eje Y, amarillo: eje Z

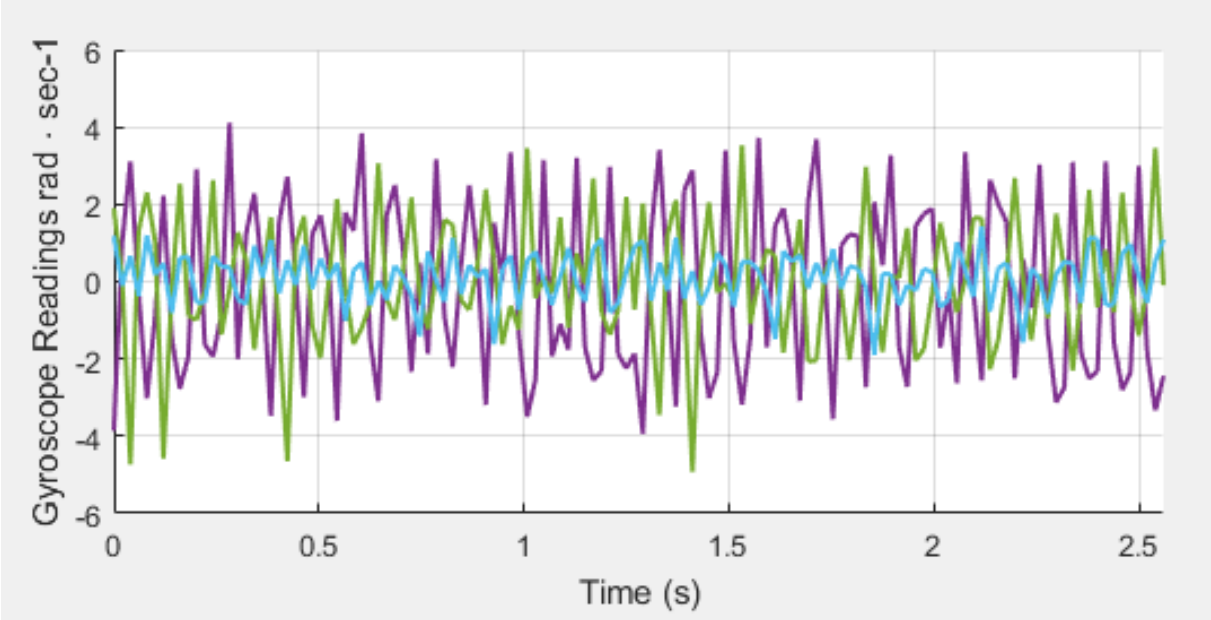


Figura 9: Señales del giroscopio en el dominio del tiempo de una ventana. Morado: eje X, verde: eje Y, azul: eje Z

- **Wmean**: Media de los 128 elementos de la ventana.
- **Wstd**: Desviación estándar de los 128 elementos de la ventana.
- **Wpca1**; Análisis de componentes principales de los 128 elementos de la ventana. Se utilizará únicamente el primer componente encontrado.
- **Wmedian**: Mediana de los 128 elementos de la ventana.
- **Wmode**: Moda de los 128 elementos de la ventana.
- **Wvar**: Varianza de los 128 elementos de la ventana.
- **Wmaximumpeakfreq**: Máximo del espectro de frecuencia de los 128 elementos de la ventana.
- **Wnumberpeaks**: Número de máximos locales del espectro de frecuencia de los 128 valores de la ventana. En nuestro caso, un máximo local es un dato que es mayor que las dos muestras vecinas.
- **Wintegralfreq**: Integral del espectro de frecuencia de los 128 elementos de la ventana.

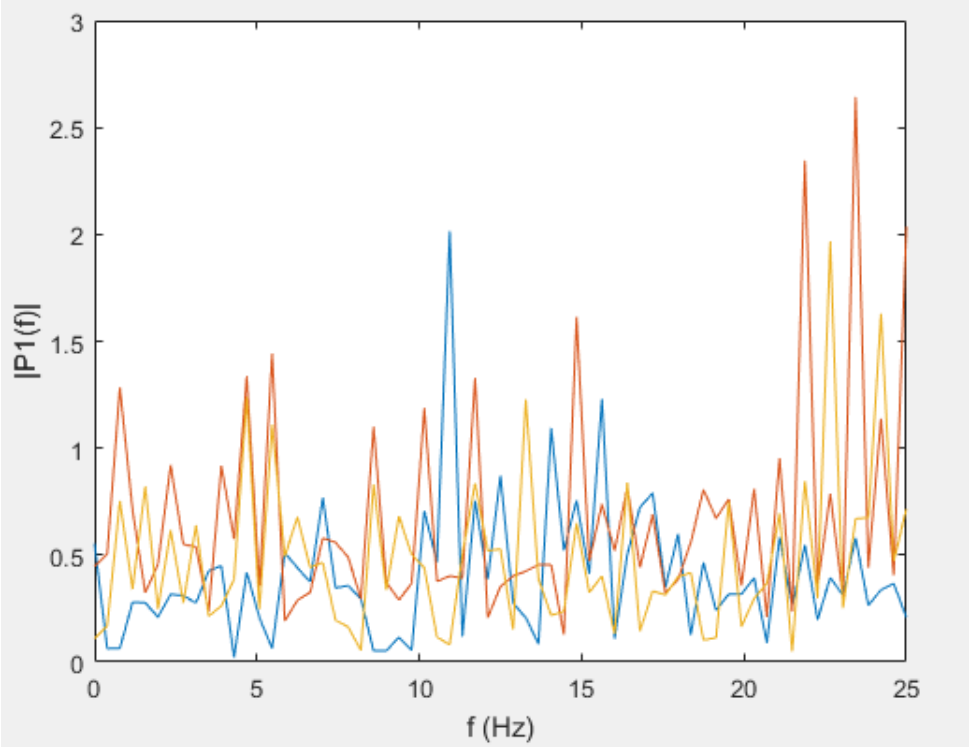


Figura 10: Señales del acelerómetro en el dominio de la frecuencia de una ventana. Azul: eje X, naranja: eje Y, amarillo: eje Z

- **Wmax**: Valor máximo de los 128 elementos de la ventana.
- **Wmin**: Valor mínimo de los 128 elementos de la ventana.
- **Wrange**: Rango, o diferencia entre los elementos máximo y mínimo, de los 128 elementos de la ventana.
- **Wintegralpowerspectrum**: Integral de las potencias del espectro de frecuencia de los 128 elementos de la ventana.
- **Wpeakmean**: Valor medio de los máximos locales del espectro de frecuencia de los 128 elementos de la ventana.
- **Wzerocrossing**: Número de veces que se produce un cruce del valor 0 en los valores de los 128 elementos de la ventana.

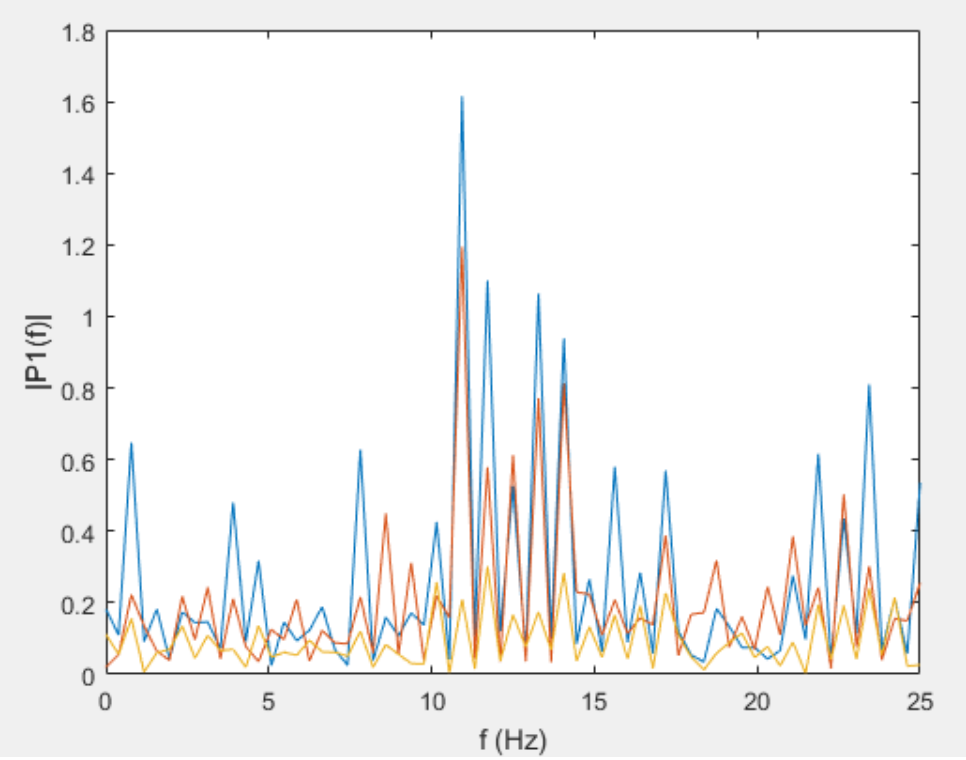


Figura 11: Señales del giroscopio en el dominio de la frecuencia de una ventana. Azul: eje X, naranja: eje Y, amarillo: eje Z

- **Wmeancrossing**: Número de veces en la ventana que se produce un cruce del valor de la media de los 128 elementos de la ventana.

- **Winterquartilerange**: IQR (*interquartile range*), o rango intercuartílico, de los 128 elementos de la ventana. Para ello se calculan los percentiles correspondientes al 75 % y al 25 % , y se restan entre sí.

- **Wmeanabsolutedeviation**: MAD (*mean absolute deviation*), o desviación media absoluta, de los 128 elementos de la ventana. Para ello se sigue la siguiente ecuación:

$$\left| \frac{\sum (vector\_value - vector\_mean)}{vector\_size} \right|$$

4.5. Validación del modelo

La toolbox de MATLAB utilizada dispone de dos algoritmos de validación que nos servirán de contraste de hipótesis para los modelos entrenados:

- **Cross-Validation**: La validación cruzada es una técnica de validación de modelos que se utiliza principalmente en entornos donde la meta es la predicción, y se quiere estimar la precisión con la que un modelo predictivo se comportará en la práctica. Una ronda de validación cruzada implica particionar una muestra de datos en N 'folds' o subconjuntos complementarios, realizar el entrenamiento con todos los subconjuntos menos uno, y validar el análisis con el subconjunto restante. Para reducir la variabilidad, se realizan varias rondas de validación cruzada utilizando diferentes subconjuntos como subconjunto de test, y los resultados de validación se promedian entre iteraciones. De esta forma, por ejemplo, en una validación cruzada de 3 folds implicará el entrenamiento y prueba del modelo 3 veces:

  - Entrenamiento en folds 1 y 2, test en fold 3
  - Entrenamiento en folds 1 y 3, test en fold 2
  - Entrenamiento en folds 2 y 3, test en fold 1

  Una de las principales razones para usar la validación cruzada son los casos en los que no hay suficientes datos disponibles como para dividirlos en un conjunto de entrenamiento y otro de test sin perder cantidades significativas de información.

- **Holdout Validation**: Esta técnica divide el dataset de forma aleatoria en dos subconjuntos complementarios, uno utilizado para el entrenamiento y otro para el test. Esta técnica es más simple que la validación cruzada, pero solo es útil en datasets con un tamaño suficientemente grande.

4.6. Análisis de Componentes Principales

El análisis de componentes principales (PCA, por sus siglas en inglés) es un procedimiento estadístico que utiliza una transformación ortogonal para convertir un conjunto de observa-

ciones de variables posiblemente correlacionadas en un conjunto de valores de variables linealmente no correlacionadas denominadas componentes principales. El número de componentes principales será menor o igual que el menor del número de variables originales o el número de observaciones. Esta transformación se define de tal manera que el primer componente principal tiene la mayor varianza posible (es decir, representa la mayor parte de la variabilidad en los datos como sea posible), y cada componente sucesivo a su vez tiene la varianza más alta posible bajo la restricción de que es ortogonal a los componentes precedentes. Los vectores resultantes son un conjunto de bases ortogonales no correlacionadas.

El *Principal Component Analysis* se utiliza principalmente como una herramienta en el análisis de datos y para hacer modelos predictivos. En el TFG que nos ocupa, se ha realizado el PCA utilizando la herramienta que proporciona para ello la propia toolbox de MATLAB. En ella se seleccionará el porcentaje de información capturada deseado o el número de predictores finales tras aplicar el PCA.

4.7. Multiescalado de las ventanas de datos

El multiescalado o multirresolución es una técnica utilizada en múltiples áreas que consiste en el uso de distintas escalas de los datos originales (o una parte de ellos) para la solución de problemas que tienen características importantes en múltiples escalas de tiempo o espacio. En el caso del actual TFG, esta variación de escalas será en función del tiempo.

En Machine Learning, el multiescalado se utiliza en el procesamiento de imágenes para aplicar un suavizado y un submuestreo repetido a múltiples resoluciones, y en el procesamiento de señales se utiliza para crear una representación multifrecuencial usando kernels como los gaussianos o los laplacianos.

Se han construido por un lado filtros de paso bajo gaussianos (Wikipedia, 2016) similares a las pirámides gaussianas (Anderson, Bergen, Burt, y Ogden, 1984) usadas en el Machine Learning con imágenes, modificando la señal por convolución con una función gaussiana, y ob-

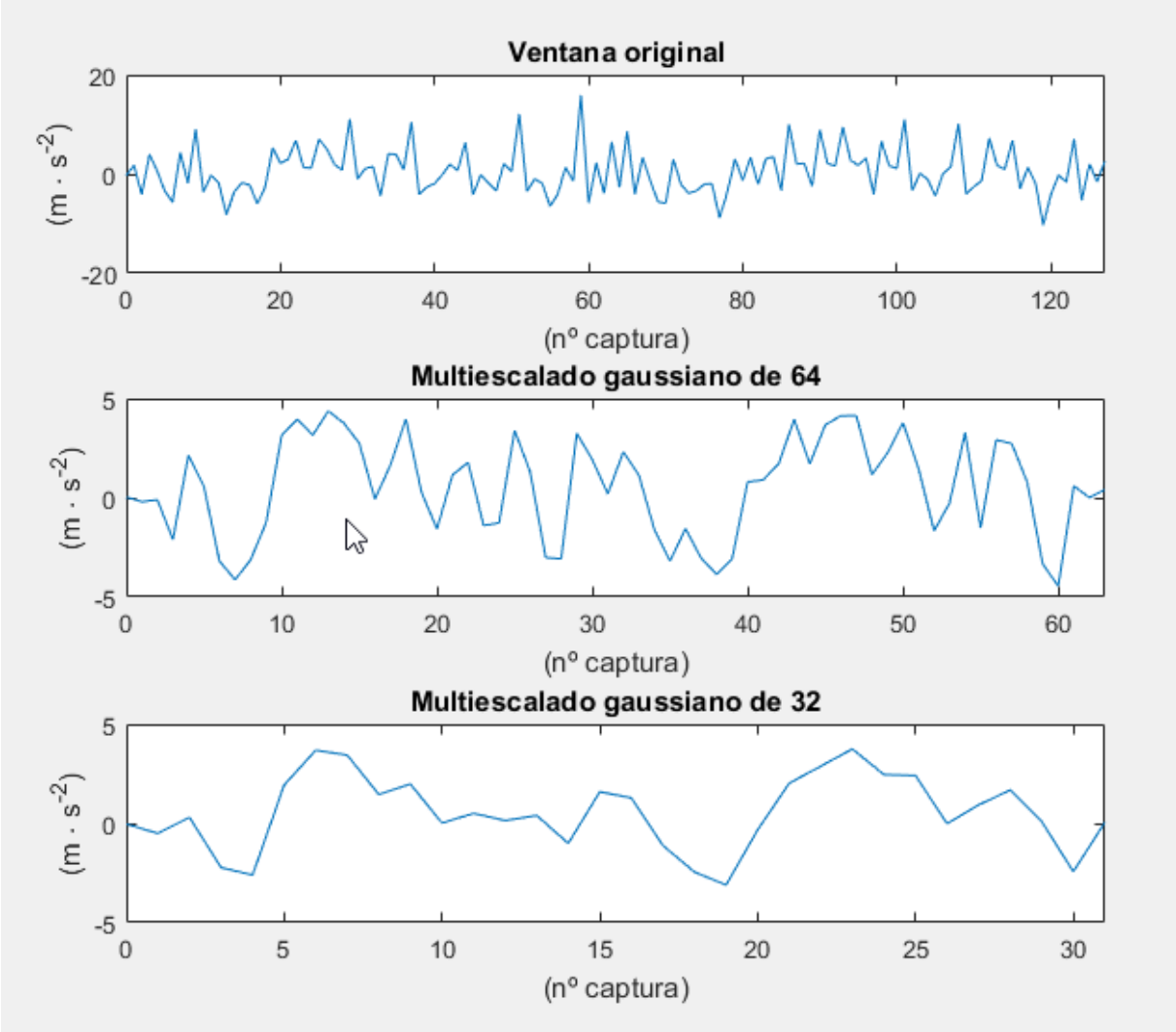


Figura 12: Aplicación de multiescalado con filtro gaussiano

teniendo como resultado una señal suavizada cuya dimensión es la mitad de la original (Figura 12); y por el otro lado se ha seguido una técnica muy simple de multiescalado en lugar de la habitual para señales, en la que se han construido vectores de N elementos, siendo N menor que 128, y construyéndose estos vectores con los N elementos centrales de los vectores de 128 datos (Figura 13). Finalmente se han relacionado estos nuevos vectores con los originales (considerando ambos como parte de la misma instancia) y se les han extraído a todos las mismas características.

Al aplicar estas técnicas se obtendrán nuevas ventanas a las que, aunque tengan un tamaño

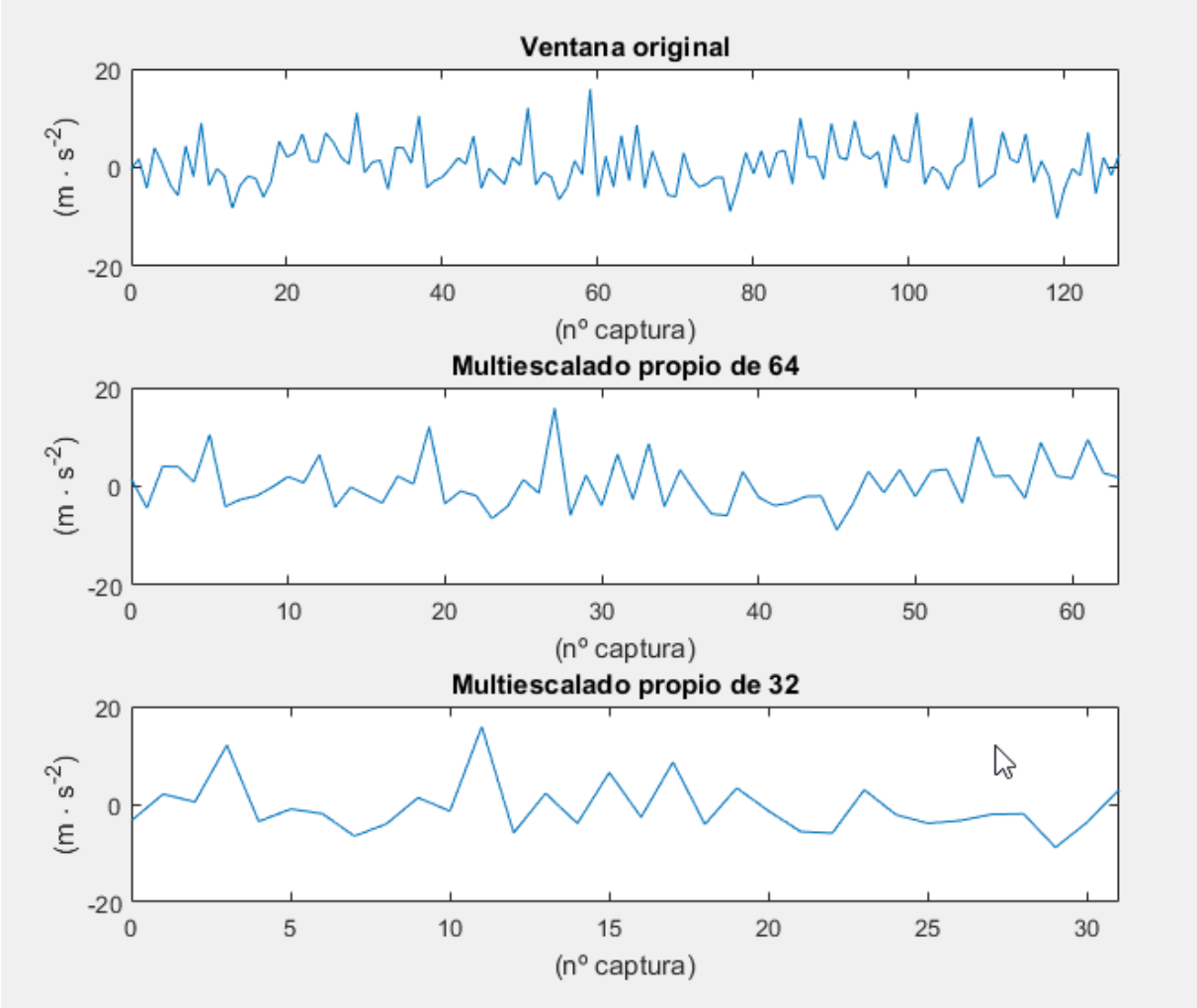


Figura 13: Aplicación de multiescalado propio

reducido con respecto a las originales, habrá que calcular también los predictores anteriormente descritos, doblando o incluso triplicando el número final de predictores como se verá en el capítulo 5, apartado 5.3.

5. Análisis de Resultados

Previamente al proceso de entrenamiento se ha dividido el dataset en dos subconjuntos complementarios: uno con el 75 % de los datos que hará de subconjunto de entrenamiento y validación, y otro con el 25 % de los datos que hará de subconjunto de test (*holdout validation*). Estos han sido creados de forma que, para todas las clases, la división se ha realizado de forma equitativa, manteniendo así el mismo número de ventanas para cada clase perteneciente al mismo subconjunto. Al porcentaje de aciertos del subconjunto de tests con respecto al modelo entrenado se le nombrará en las tablas pertenecientes a estos experimentos como 'Aciertos Test'.

Además, el subconjunto de entrenamiento será entrenado utilizando una validación cruzada, o *cross-validation*, de 5 subconjuntos complementarios aleatorios. El porcentaje de aciertos calculado por la validación cruzada en el entrenamiento del modelo se le nombrará en las tablas pertenecientes a este apartado como 'Aciertos Val.'.

A estas tablas también se les han añadido dos columnas que proporcionan métricas de tiempo: 'Tiempos Entr.' indica el tiempo que ha sido necesario para entrenar el modelo y 'Vel. Pred.' indica la cantidad de instancias que es capaz de predecir el modelo por segundo.

5.1. Resultados iniciales

A continuación se muestran los resultados (calculados a partir de la predicción individual de cada ventana) obtenidos en los entrenamientos de los múltiples 'learners' proporcionados de base por la app *Classification Learner*. Estos han utilizado las 18 características explicadas en el capítulo anterior, apartado 4.4, generando un total de 108 predictores tras calcular todas estas características de forma individual para cada ventana creada a partir de las 6 señales obtenidos del dataset. Tras esto se ha construido dos tablas que unen todos estos predictores, quedando una tabla de 5940x108 datos para la fase de entrenamiento y otra tabla de 1980x108 datos para la fase de test. La primera dimensión de la tabla se obtiene de: 20 sujetos multiplicado

Tabla 1

Resultados iniciales utilizando todos los predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	95.2 %	5.7129	57000	82.61 %
Medium Tree	57.1 %	3.8433	67000	54.51 %
Simple Tree	24.1 %	2.6989	67000	22.52 %
Linear Discriminant	95.5 %	1.1754	43000	93.97 %
Quadratic Discriminant	99.5 %	1.6014	34000	96.55 %
Linear SVM	100 %	31.25	2700	99.14 %
Quadratic SVM	100 %	34.369	2000	99.09 %
Cubic SVM	100 %	33.404	1500	99.09 %
Fine Gaussian SVM	16 %	96.245	250	6.74 %
Medium Gaussian SVM	100 %	53.351	500	97.87 %
Coarse Gaussian SVM	99.8 %	44.909	630	98.93 %
Fine KNN	99.8 %	3.4078	4300	96.65 %
Medium KNN	99.6 %	3.439	4000	97.31 %
Coarse KNN	97.2 %	4.0135	3600	96.96 %
Cosine KNN	99.1 %	3.3362	4200	95.64 %
Cubic KNN	98.5 %	104.66	110	93.36 %
Weighted KNN	99.8 %	3.484	4000	97.16 %
Boosted Trees	94.5 %	81.809	7500	88.24 %
Bagged Trees	99.8 %	18.493	6700	98.07 %
Subspace Discriminant	99.8 %	16.591	2000	99.54 %
Subspace KNN	99.5 %	43.88	290	92.49 %
RUSBoosted Trees	72.6 %	85.828	8000	59.89 %

por 99 ventanas que tiene cada captura de datos multiplicado por el número de capturas de datos utilizado para cada fase de las pruebas (3 para la fase de entrenamiento y validación y 1 para la fase de test). A ambas tablas se les incluirán una columna indicando la clase a la que pertenece la instancia para comprobar si las predicciones realizadas por los modelos son correctas.

A la vista de los resultados obtenidos podemos obtener las siguientes conclusiones:

- Como se puede observar mejor en la Figura 14, todos los diseños presentan una mayor precisión predictiva cuando se trata con los propios datos de entrenamiento y validación que cuando se trata con los datos de prueba. Esta es una conclusión lógica debido a que los modelos suelen ajustarse a los datos de entrenamiento cuando se van creando. El decremento de la precisión no es alto, lo que indica que los modelos no sufren de

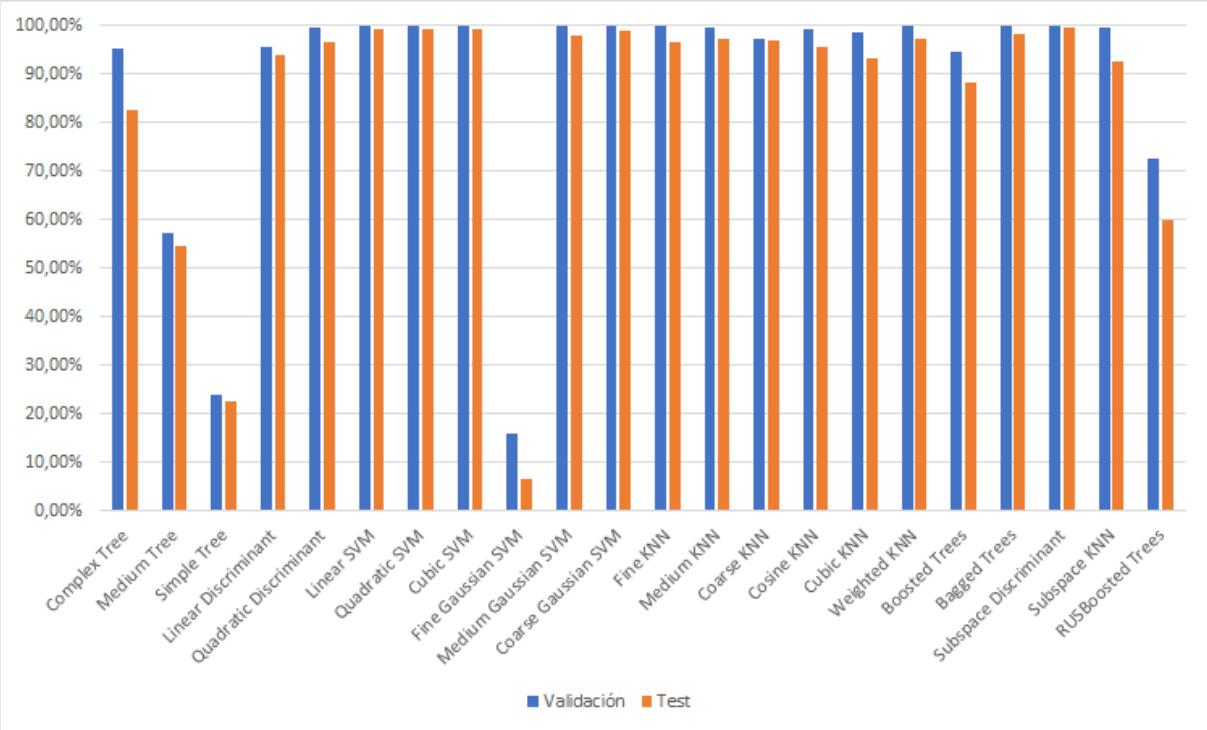


Figura 14: Precisión de los Modelos Originales

sobreentrenamiento.

- Los modelos basados en árboles de decisión dan mejores resultados conforme se aumenta su número de ramificaciones máximas, lo que indica que el caso de uso actual es demasiado complejo para este tipo de modelos. Tal vez podría aumentarse la predicción del modelo *Complex Tree* aún más si se aumentaran su número de ramificaciones máximas, pero sería a costa de provocar un gran sobreentrenamiento como se puede deducir de la diferencia entre el porcentaje de aciertos en la validación y en el entrenamiento. Como prueba de ello, posteriormente se han lanzado árboles de decisión con un mayor número máximo de ramificaciones, obteniéndose que el máximo de estas que se pueden obtener para el conjunto de datos tratados es de 120, número en el cuál se alcanza una tasa de acierto en test de 84.18 %, y observándose que si se coloca un número mayor que ese como número máximo de ramificaciones el *learner* seguirá sin sobrepasar las 120 ya mencionadas debido a que no es capaz de sacar más información de los predictores.

- Lo porcentajes de acierto en test obtenidos en los modelos basados en análisis discrimi-

nantes indican que es válida la suposición de aproximar por distribuciones gaussianas. Según la información proporcionada por la toolbox de MATLAB la memoria necesaria para el modelo *Quadratic Discriminant* es 100 veces mayor que para el modelo *Linear Discriminant*, lo que hace mucho más atrayente a este último.

- Todos los modelos basados en máquinas de soporte vectorial dan muy buenos resultados excepto el *Fine Gaussian SVM*, cuyo kernel es demasiado pequeño para el caso de uso actual. También se observa que los datos siguen mejor una distribución polinomial que una gaussiana. El punto negativo de estos modelos es que todos requieren una gran cantidad de memoria cuando tratan con clasificaciones de más de dos clases distintas, además de ser difícilmente interpretables. Solo puede aconsejarse el uso de uno de estos modelos en sistemas que no tengan limitaciones de capacidad computacional o de memoria, y en casos de uso donde no se tengan requisitos de tiempo real duro.

- La distancia euclídea parece ser la mejor al usar los modelos basados en K-vecinos más cercanos. Además se observa que el punto dulce está en 10 vecinos. Este tipo de modelos son más rápidos y usan menos memoria que los basados en máquinas de soporte vectorial, pero siguen siendo más complejos y lentos que los basados en análisis discriminantes.

- La combinación de múltiples árboles simples (20 ramificaciones) con el algoritmo *AdaBost* supera en predicción al uso de un único árbol complejo (100 ramificaciones), manteniendo aun así un muy bajo uso de la memoria, pero generando un predictor lento si se compara con los árboles originales. Por otro lado la combinación de árboles complejos usando *Bagging* obtiene resultados aún mejores sacrificando una cantidad de memoria más de 100 veces superior al caso anterior.

- El mejor resultado de todos estos modelos iniciales se obtiene con la combinación de múltiples modelos de análisis discriminante (*Subspace Discriminant*).

- La combinación de múltiples modelos basados en k-vecinos más cercanos (*Subspace KNN*) es peor que cualquiera de los anteriormente mostrados basados en esta misma técnica.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	98%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%
2	0%	97%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%
3	0%	0%	73%	0%	9%	5%	4%	0%	0%	0%	0%	0%	0%	0%	9%	0%	0%	0%	0%	0%
4	0%	0%	0%	88%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	2%	0%	0%	7%	0%	0%
5	0%	0%	0%	8%	81%	0%	0%	0%	2%	5%	0%	0%	0%	0%	0%	0%	2%	0%	0%	1%
6	0%	0%	0%	4%	0%	91%	5%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	5%	0%	0%	16%	74%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%
8	0%	0%	1%	0%	0%	3%	1%	76%	0%	1%	0%	0%	5%	0%	0%	1%	2%	0%	7%	0%
9	2%	0%	0%	10%	0%	0%	0%	0%	75%	1%	0%	0%	0%	0%	0%	0%	1%	4%	5%	2%
10	3%	0%	0%	0%	0%	0%	0%	0%	0%	93%	0%	0%	2%	0%	0%	2%	0%	0%	0%	0%
11	0%	0%	0%	13%	0%	0%	0%	0%	0%	0%	84%	0%	0%	0%	0%	0%	0%	3%	0%	0%
12	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	91%	0%	0%	9%	0%	0%	0%	0%	0%
13	0%	0%	0%	0%	0%	2%	0%	0%	0%	6%	0%	0%	89%	0%	0%	0%	2%	1%	0%	0%
14	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%
15	0%	0%	0%	0%	0%	38%	6%	1%	0%	0%	0%	0%	0%	0%	55%	0%	0%	0%	0%	0%
16	0%	0%	0%	11%	0%	0%	0%	0%	0%	6%	0%	0%	0%	0%	0%	82%	1%	0%	0%	0%
17	1%	0%	0%	2%	0%	4%	0%	5%	0%	0%	0%	0%	0%	0%	0%	0%	88%	0%	0%	0%
18	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	23%	0%	0%	0%	2%	0%	0%	74%	0%	0%
19	1%	0%	0%	6%	0%	0%	33%	0%	0%	3%	0%	0%	0%	0%	0%	2%	0%	0%	55%	0%
20	1%	0%	0%	1%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	4%	0%	0%	92%

Figura 15: Matriz de confusión de Tests de ComplexTree

- El modelo *RUSBoosted Trees* da una mala tasa de acierto debido a que la distribución de clases que crea el algoritmo *RUS* contiene demasiada poca información como para construir un clasificador. Esta técnica es útil cuando el dataset sufre de desequilibrio en sus clases, cosa que no ocurre en el utilizado.

- Cabe destacar la alta precisión obtenida por algunos modelos en la fase de test, superando en 16 de los 22 modelos el 90 % de aciertos, y superando además 4 de estos el 99 %.

Como modelo final se recomienda la utilización de *Susbspace Discriminant*, debido a su alta tasa de acierto con el subconjunto de test y a sus bajos requisitos de memoria. Si por el caso contrario se quisiera sacrificar un poco de precisión con el fin de obtener una mayor velocidad de predicción, se recomendaría el uso del modelo *Linear Discriminant*.

A continuación, en las Figuras 15 y 16, se muestran la matriz de confusión de un modelo con una buena, pero no de las más altas, tasa de acierto (*Complex Tree*) y la del mejor modelo obtenido en la tabla 1 (*Subspace Discriminant*). En ellas las filas representan la clase original de la instancia a clasificar y las columnas representan la clase predicha por el modelo.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	96%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
12	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
13	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
14	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%
15	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%
16	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%
17	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%
18	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	97%	0%	0%
19	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	98%	0%
20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%

Figura 16: Matriz de confusión de Tests de Subspace Discriminant

La clave de una buena predicción, y un buen modelo, es obtener una clasificación equitativa: obtener buenos resultados en todas las clases. No deben centrarse en una, ni tampoco cometer demasiados errores intentando clasificar instancias de una clase en concreto. Teniendo esto en cuenta, se puede afirmar que ambos modelos realizan una buena predicción.

5.2. Aplicación del Análisis de Componentes Principales

Se les ha reducido la dimensionalidad a los datos originales utilizando la técnica *Principal Component Analysis* con distintos porcentajes de información retenida (respecto a los predictores originales). Aun así su uso no está mejorando los resultados iniciales (ver Tabla 2), sino empeorándolos. Esta situación podría ser debida a que el dataset utilizado no contiene el número suficiente de datos por instancia y/o a que los resultados ya son suficientemente buenos y no dejan mucho margen de mejora. No obstante, cabe destacar que hay una gran correlación entre los diferentes predictores, como se observa del hecho de la gran cantidad de información retenida por un número pequeño de autovectores.

Tabla 2

Mejores modelos según PCA

Porcentaje PCA	N° Predictores	Mejor Modelo	Aciertos Test
Sin PCA	108	Subspace Discriminant	99.54 %
99.5 %	43	Linear SVM	96.55 %
99 %	36	Linear SVM	97.16 %
98 %	28	Linear SVM	97.06 %
97 %	23	Linear SVM	95.84 %
96 %	20	Quadratic SVM	95.13 %
95 %	18	Medium Gaussian SVM	93.81 %

Merece la pena resaltar que no se debe confundir el número de predictores mostrados en la Tabla 2 con el número total de predictores realmente calculados: en todos los casos se necesitarán utilizar los 108 predictores iniciales y aplicarles un cálculo adicional reducir su número, por lo que se acabará complicando el modelo y aumentando el tiempo computacional, aunque reduciendo la cantidad de memoria necesaria por el modelo.

Los resultados completos de la aplicación del análisis de componentes principales se encuentran en el anexo A, Tablas 6, 7, 8, 9, 10 y 11.

5.3. Aplicación de Técnicas de Multiescalado

Se muestran los mejores resultados (Tabla 3) de entrenamientos aplicando múltiples variaciones de *Multiescalado* con ventanas de 64 y 32 datos, creando estas ventanas con una pirámide gaussiana (Figura 17) o seleccionando los elementos centrales de la venta original (Figura 18) tal como se explicó en el capítulo anterior, apartado 4.7. Al aplicar el multiescalado se doblan, o triplican, el número de ventanas por instancia en los datos, por lo que en constancia se doblan, o triplican, el número de predictores, pasando a ser 216 ó 324 según el caso.

Queda demostrado que, en algunos modelos, el Multiescalado puede suponer una mejora en la precisión, pero esto supone doblar o triplicar la cantidad de predictores utilizados, lo que

Tabla 3

Mejores modelos según Multiescalado

Tipo Multiescalado	Mejor Modelo	Aciertos Test
Gaussiano 64	Subspace Discriminant	99.75 %
Gaussiano 32	Subspace Discriminant	99.54 %
Gaussiano 64+32	**Subspace Discriminant**	**99.85 %**
Propio 64	Subspace Discriminant	99.59 %
Propio 32	Subspace Discriminant	99.54 %
Propio 64+32	Subspace Discriminant	99.65 %

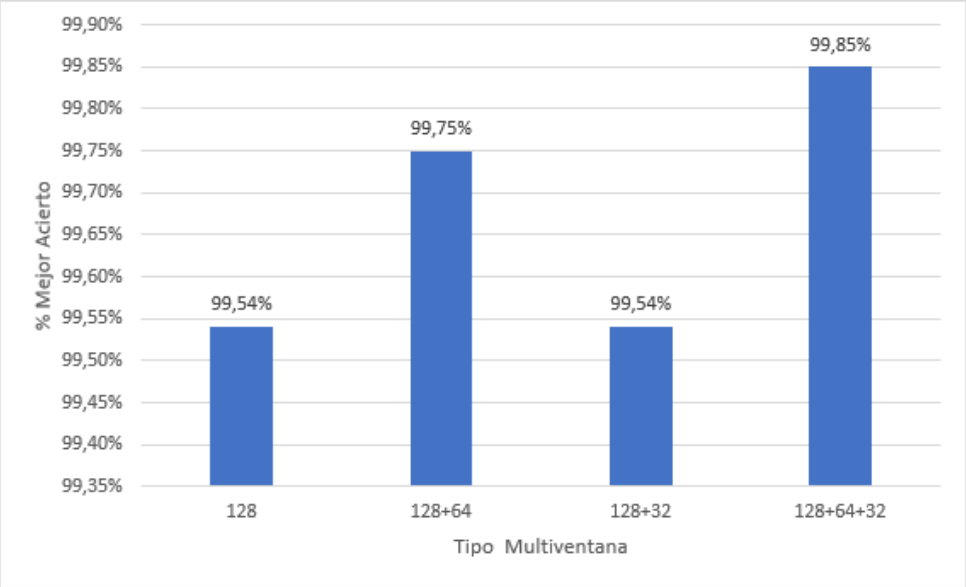


Figura 17: Mejor Porcentaje de Acierto en Tests según Multiescalado Pirámide Gaussiana

puede suponer un problema en sistemas con baja memoria o con baja capacidad computacional.

Los resultados completos de la aplicación de las técnicas de Multiescalado se encuentran en el anexo A, Tablas 12, 13, 14, 15, 16 y 17

5.4. Selección de Mejores Predictores

Finalmente se ha realizado una **Selección de Características** (o *Feature Selection*) para buscar los mejores predictores o combinación de predictores para los modelos trabajados. Esta

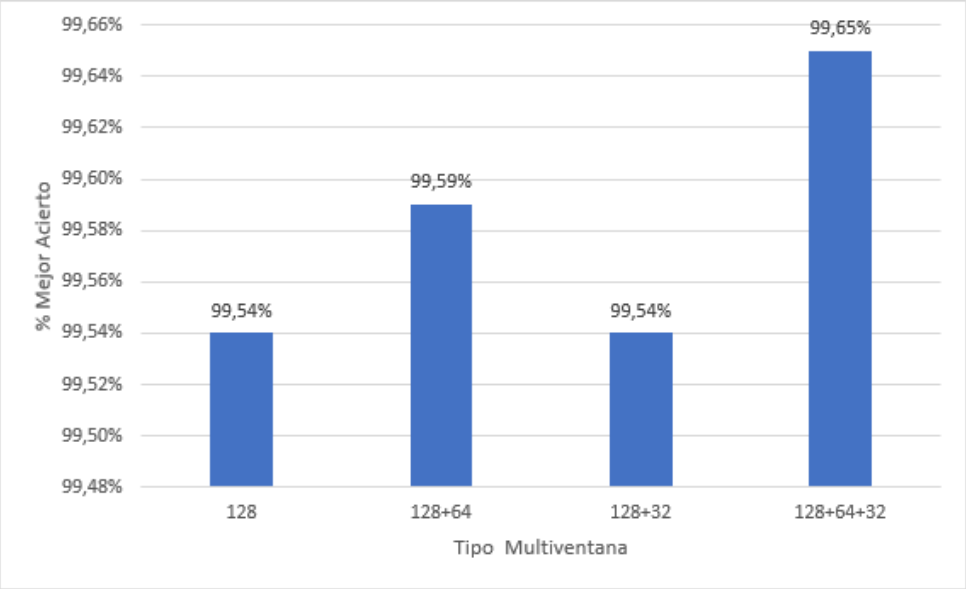


Figura 18: Mejor Porcentaje de Acierto en Tests según Multiescalado Propio

información es útil para reducir el número de predictores de forma óptima en sistemas con limitaciones de memoria o capacidad computacional.

Tabla 4

Mejores modelos según Selección de Mejores Predictores

N° Predictores	Ultimo Predictor añadido	Mejor Modelo	Aciertos Test
1	Wmeanabsolutedeviation	Quadratic SVM	90.6 %
2	Wzerocrossing	Medium Gaussian SVM	95.94 %
3	Wmedian	Medium Gaussian SVM	98.48 %
4	Wmax	Linear SVM	98.83 %
5	Wmode	Quadratic SVM	99.24 %
6	Wmaximumpeakfreq	Quadratic SVM	99.34 %

La Figura 19 muestra la mejor combinación de 6 predictores y como, de izquierda a derecha, va aumentando la precisión del mejor modelo conforme se añaden estos. Se ha parado en 6 pues las mejoras en precisión al añadir nuevos descriptores eran marginales.

Es de destacar cómo con solo 6 descriptores bien elegidos se supera fácilmente los mejores resultados del PCA, incluso para el caso de proyectar en 43 dimensiones en el que se retenía el 99.5 % de la información original.

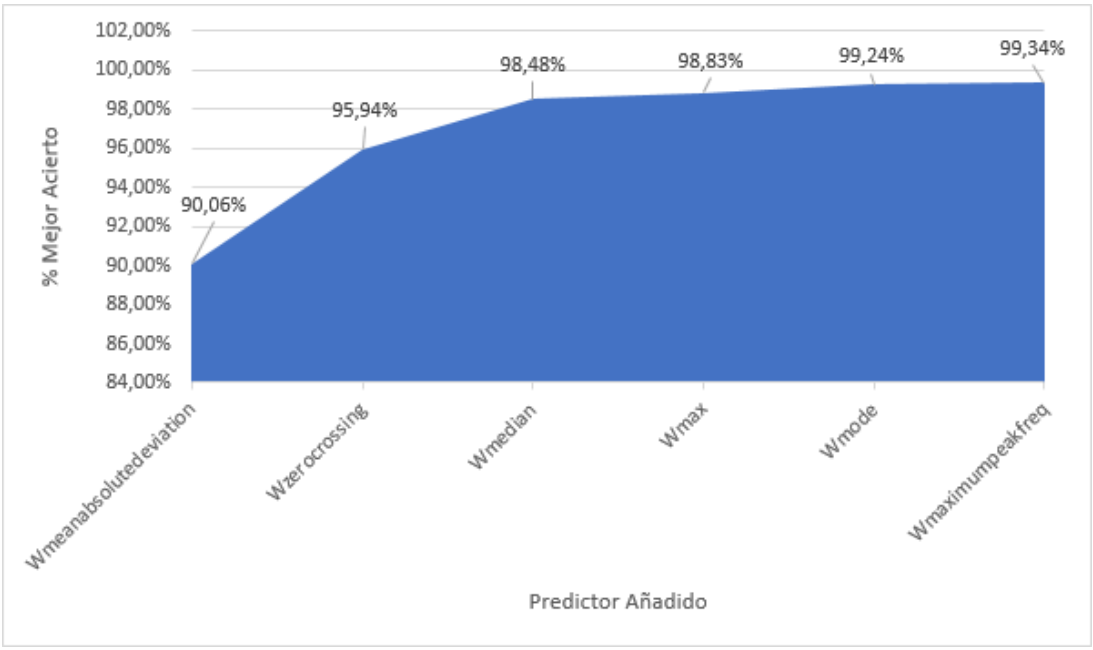


Figura 19: Mejor Porcentaje de Acierto en Tests en mejores Predictores

5.5. Comparativa con trabajos similares

La Tabla 5 presenta una comparativa de los resultados presentados por otros autores que han realizado trabajos similares al nuestro, es decir, la identificación de personas basadas en datos de los acelerómetros, y que fueron comentados brevemente en la introducción.

El único que nos iguala en precisión es el primero, que está basado en *Convolutional Neural Networks* (CNNs), modelo mucho más complejo que el nuestro. En el resto de los casos las precisiones están por debajo.

Tabla 5

Resultados en Trabajos Similares

Artículo	Porcentaje de Predicción
IDNet: Smartphone-based Gait Recognition with Convolutional Neural Networks	99.85 %
Influence of Holding Smart Phone for Acceleration-Based Gait Authentication	96.36 %
Authentication of Smartphone Users Based on the Way They Walk Using k-NN Algorithm	96.33 %
An Efficient HOS-Based Gait Authentication of Accelerometer Data	94 %
Gait Identification Using Accelerometer on Mobile Phone	92.7 %
Smartphone based user verification leveraging gait recognition for mobile healthcare systems	$\sim$90 %

6. Conclusiones y Líneas Futuras

Tras la realización de este TFG se han llegado a unas conclusiones respecto a las cuestiones que se plateaban al principio y se han ido planteando una serie de ideas que no se han podido realizar por falta de tiempo, pero que se consideran interesantes proponer como desarrollos futuros basados en este trabajo.

6.1. Conclusiones

Se han cumplido los objetivos establecidos, construyendo un dataset y desarrollando un conjunto de modelos con tasas de predicción bastante buenas y utilizables en multitud de sistemas con distintos requisitos de aciertos y con distintas limitaciones. Adicionalmente se ha implementado una app Android con la cual el departamento de Arquitectura de Computadores podrá seguir ampliando el dataset actual y desarrollar otros proyectos de investigación haciendo uso de este.

Desarrollar este Trabajo de Fin de Grado ha supuesto una gran oportunidad para investigar en el campo del aprendizaje automático, aprender las técnicas más usuales y trabajar en un caso de uso real, novedoso y con un amplio margen de evolución.

Tras la finalización de este TFG se han aprendido las siguientes lecciones:

- Queda demostrada la viabilidad de la identificación de personas utilizando su forma de andar a partir de técnicas de aprendizaje automático utilizando únicamente los sensores inerciales pertenecientes a un Smartphone.
- Se han construido una serie de modelos que, por sus diferenciadas características, pueden utilizarse en entornos con distintas limitaciones o requisitos de predicción. Por ejemplo, podría exportarse alguno de los modelos más simples de los entrenados (como el *Complex Tree* inicial) para sistemas con pocos recursos, como pueden ser sistemas basados en Microcontroladores o dispositivos Smartphone. Por otro lado, podrían usarse los modelos más complejos (como el *Subspace Discriminant* con multiescalado obtenida por

Piramide Gaussiana) para entornos en los que se necesitan predicciones muy precisas con un gran número de clases, y realizando la parte computacional en sistemas más capaces. En algunos modelos también se observa una muy rápida velocidad de predicción, demostrando que podrían usarse para predecir en tiempo real, mientras el sujeto a clasificar camina.

- Aunque en el caso actual el PCA no supone una mejora, cosa que era difícil con unas tasas de acierto iniciales tan altas, y las técnicas de multiescalado mejoran poco la precisión con respecto a su coste adicional, no debemos descartar estas técnicas para entornos en los que hay un mayor número de clases a predecir.
- En sistemas con bajos recursos computacionales bastaría a con calcular solo 5 ó 6 predictores para superar una tasa de acierto del 99 %.
- Se ha igualado la precisión de los modelos a la de otros sistemas más complejos (basados en CNNs) y se ha superado a otros similares propuestos en trabajos anteriores.
- Los resultados presentados se basan en la detección usando una ventana. Ampliando la identificación a un mínimo de tres los resultados serían aún mejores, próximos al 99.99 %

6.2. Líneas Futuras

Tras la realización del TFG se han recopilado algunas líneas futuras de trabajo:

- Inclusión al dataset de una mayor variabilidad. Aunque se han jugado con elementos como los recorridos, el calzado o el uso de pesos adicionales, podría estudiarse el problema planteado al incluirse al dataset terrenos más irregulares, cuestas pronunciadas, escaleras, muestras tomadas a paso ligero o en carrera, etc.
- Mejora del sistema de captura de datos. Actualmente este se encuentra limitado en número de dispositivos compatibles debido a problemas derivados de la poca cohesión que hay en el ecosistema Android. Por ello podría plantearse su exportación a otras plataformas móviles, o incluso la creación de un sistema hardware propio que pueda distribuirse a bajo coste.

- Estudio de la fusión de distintas técnicas ya utilizadas en este trabajo, así como la inclusión de nuevas. Debido a los requisitos de tiempo, en el actual TFG no han podido probarse cosas como la unión de la **Selección de Mejores Predictores** con las técnicas de **Multiescalado**, o la de este último con las de **PCA**.
- Mayor uso de la personalización de modelos proporcionada por la toolbox de MATLAB, como pueden ser: el número máximo de ramificaciones en los arboles de decisión, variación en la escala de los kernels de los algoritmos de soporte vectorial, el número de vecinos usados en las técnicas de K-vecinos más cercanos, etc. Aun así, al no hacer uso de esto en el presente TFG se han conseguido entrenar modelos más genéricos y menos propensos al sobreentrenamiento.
- Independizar la posición y orientación del móvil para realizar las predicciones.
- Estudiar otros casos de usos aplicados al mismo dataset, como pueden ser: identificación del sexo, problemas físicos, identificación de uno contra todos, etc.

Bibliografía

Anderson, C. H., Bergen, J. R., Burt, P. J., y Ogden, J. M. (1984). *Pyramid methods in image processing.*

Anguita, D., Ghio, A., Oneto, L., Parra, X., y Reyes-Ortiz, J. (2012). Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. En *Ambient assisted living and home care: 4th international workshop, iwaal 2012, vitoria-gasteiz, spain, december 3-5, 2012. proceedings* (pp. 216–223). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: https://doi.org/10.1007/978-3-642-35395-6_30

Bmi160 datasheet (n.º BST-BMI160-DS000-07). (2015, 2).

Choi, S., Youn, I.-H., LeMay, R., Burns, S., y Youn, J.-H. (2014). Biometric gait recognition based on wireless acceleration sensor using k-nearest neighbor classification. En *International conference on computing, networking and communications (icnc).* doi: https://doi.org/10.1109/ICCNC.2014.6785491

Choudhury, T., Consolvo, S., Harrison, B., Hightower, J., LaMarca, A., LeGrand, L., ... Haehnel, D. (2008). The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*. doi: https://doi.org/10.1109/MPRV.2008.39

Documentación android. (s.f.). `https://developer.android.com/index.html`. (Accedido: 8 de Junio de 2017)

Documentación android: Activity. (s.f.). `https://developer.android.com/reference/android/app/Activity.html`. (Accedido: 15 de Junio de 2017)

Documentación android: Asyntask. (s.f.). `https://developer.android.com/reference/android/os/AsyncTask.html`. (Accedido: 15 de Junio de 2017)

Documentación android: Dialogfragment. (s.f.).
hhttps://developer.android.com/reference/android/app/DialogFragment.html.
(Accedido: 15 de Junio de 2017)

Documentación android: Motion sensor. (s.f.).
https://developer.android.com/guide/topics/sensors/sensors_motion.html.
(Accedido: 15 de Junio de 2017)

Documentación android: Sensor. (s.f.).
https://developer.android.com/reference/android/hardware/Sensor.html.
(Accedido: 15 de Junio de 2017)

Documentación android: Sensorevent. (s.f.).
https://developer.android.com/reference/android/hardware/SensorEvent.
html#values. (Accedido: 15 de Junio de 2017)

Documentación android: Sensormanager. (s.f.).
https://developer.android.com/reference/android/hardware/SensorManager.html.
(Accedido: 15 de Junio de 2017)

Documentación android studio. (s.f.).
https://developer.android.com/studio/intro/index.html?hl=es-419. (Accedido: 20 de Junio de 2017)

Documentación android: Systemclock. (s.f.).
https://developer.android.com/reference/android/os/SystemClock.html.
(Accedido: 15 de Junio de 2017)

Documentación intellij idea. (s.f.). https://www.jetbrains.com/idea/documentation/.
(Accedido: 20 de Junio de 2017)

Documentación jsch. (s.f.). http://www.jcraft.com/jsch/. (Accedido: 20 de Junio de 2017)

Documentación libsvm. (s.f.). https://www.csie.ntu.edu.tw/∼cjlin/libsvm/. (Accedido: 20 de Junio de 2017)

Documentación matlab. (s.f.). https://es.mathworks.com/help/matlab/. (Accedido: 13 de Junio de 2017)

Documentación scikit-learn. (s.f.). http://scikit-learn.org/stable/documentation.html. (Accedido: 20 de Junio de 2017)

Documentación shark. (s.f.). http://image.diku.dk/shark/sphinx_pages/build/html/index.html?utm_source=feedly. (Accedido: 20 de Junio de 2017)

Documentación statistics and machine learning toolbox. (s.f.). https://es.mathworks.com/help/stats/. (Accedido: 15 de Junio de 2017)

Documentación tensorflow. (s.f.). https://www.tensorflow.org/api_docs/python/. (Accedido: 20 de Junio de 2017)

Documentación weka. (s.f.). http://www.cs.waikato.ac.nz/ml/weka/documentation.html. (Accedido: 20 de Junio de 2017)

Furuno. (s.f.). *What is dead reckoning (dr)?* Descargado de http://www.furuno.com/en/gnss/technical/tec_dead (Accedido: 20 de Junio de 2017)

Gadaleta, M., y Rossi, M. (2006). Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. *IEEE Transactions on Information Technology in Biomedicine*(10), 156 - 167.

Gadaleta, M., y Rossi, M. (2016). Idnet: Smartphone-based gait recognition with convolutional neural networks. *CoRR*.

Herrlich, S., Spieth, S., Nouna, R., Zengerle, R., Giannola, L. I., Pardo-Ayala, D. E., ... Garino, P. (2011). Ambulatory treatment and telemonitoring of patients with parkinson's disease. En R. Wichert y B. Eberhardt (Eds.), *Ambient assisted living: 4. aal-kongress 2011, berlin, germany, january 25–26, 2011* (pp. 295–305). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: https://doi.org/10.1007/978-3-642-18167-2_20

Karantonis, D., Narayanan, M., Mathie, M., Lovell, N., y Celler, B. (2006). Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. *IEEE Transactions on Information Technology in Biomedicine*. doi: https://doi.org/10.1109/TITB.2005.856864

Khan, A., Lee, Y., Lee, S., y Kim, T. (2010). *Human activity recognition via an accelerometer-enabled-smartphone using kernel discriminant analysis* (Inf. Téc.). Future Information Technology (FutureTech). doi: https://doi.org/10.1109/FUTURETECH.2010.5482729

Laerhoven, K. V., y Cakmakci, O. (2000). What shall we teach our pants? En *Proceedings of the 4th ieee international symposium on wearable computers* (pp. 77–). Washington, DC, USA: IEEE Computer Society. Descargado de `http://dl.acm.org/citation.cfm?id=851037.856531`

Mahmoud, Q. H. (2002). *Compressing and decompressing data using java apis.* Descargado de `http://www.oracle.com/technetwork/articles/java/compress-1565076.html` (Accedido: 20 de Junio de 2017)

Monsonís, A., Pérez, C., Romagosa, J., y Rodríguez-Molinero, A. (2012). Dyskinesia and motor state detection in parkinson's disease patients with a single movement sensor. doi: https://doi.org/10.1109/EMBC.2012.6346150

Nickel, C., Wirtl, T., y Busch, C. (2012). Authentication of smartphone users based on the way they walk using k-nn algorithm. En *Eighth international conference on intelligent information hiding and multimedia signal processing (iih-msp).* doi: https://doi.org/10.1109/IIH-MSP.2012.11

Ren, Y., Chen, Y., Chuah, M. C., y Yang, J. (2013). Smartphone based user verification leveraging gait recognition for mobile healthcare systems. En *Conference on sensor, mesh and ad hoc communications and networks (secon), 2013 10th annual ieee communications society.* doi: https://doi.org/10.1109/SAHCN.2013.6644973

Silva, D., y Batista, G. (2016). *Speeding up all-pairwise dynamic time warping matrix calculation.* doi: https://doi.org/10.1137/1.9781611974348.94

Sprager, S., y Juric, M. B. (2015). An efficient hos-based gait authentication of accelerometer data. *IEEE Transactions on Information Forensics and Security*(10), 1486 - 1498.

Thang, H. M., Viet, V. Q., Nguyen, T. D., y Choi, D. (2012). Gait identification using accelerometer on mobile phone. En *Conference on control, automation and information sciences (iccais), 2012 international.* doi: https://doi.org/10.1109/ICCAIS.2012.6466615

Turan, A., y Barshan, B. (2014). Detecting falls with wearable sensors using machine learning techniques.
doi: http://dx.doi.org/10.3390/s140610691

Watanabe, Y. (2014). Influence of holding smart phone for acceleration-based gait authentication. En *Conference on emerging security technologies (est), 2014 fifth international.* doi: https://doi.org/10.1109/EST.2014.24

Wikipedia. (2016). *Gaussian filter — Wikipedia, the free encyclopedia.* Descargado de `https://en.wikipedia.org/wiki/Gaussian_filter` (Accedido: 20 de Junio de 2017)

Wikipedia. (2017). *Hidden markov model — Wikipedia, the free encyclopedia.* Descargado de `https://en.wikipedia.org/wiki/Hidden_Markov_model` (Accedido: 20 de Junio de 2017)

Anexos

A. Tablas de resultados

En el presente anexo se han unificado todos los resultados procedentes de los experimentos realizados durante el desarrollo del actual TFG y ya explicados en los capítulos anteriores, con el objetivo de facilitar su lectura y posterior análisis.

Tabla 6

Resultados con PCA de 99,5 %, 43 predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	77.1 %	8.687	19000	66.78 %
Medium Tree	54.8 %	3.819	21000	49.49 %
Simple Tree	22.4 %	3.1239	21000	21.86 %
Linear Discriminant	95 %	2.3979	16000	90.67 %
Quadratic Discriminant	96.9 %	2.3601	15000	92.6 %
Linear SVM	99.2 %	24.387	3700	96.5 %
Quadratic SVM	99.3 %	24.195	2700	96.55 %
Cubic SVM	99.3 %	28.01	2200	96.2 %
Fine Gaussian SVM	42 %	73.489	360	18.76 %
Medium Gaussian SVM	99.3 %	30.458	1900	96.5 %
Coarse Gaussian SVM	96.1 %	46.741	560	94.47 %
Fine KNN	93.2 %	3.7262	7300	81.85 %
Medium KNN	92.4 %	2.7546	7100	85.55 %
Coarse KNN	83.9 %	2.9563	6300	80.17 %
Cosine KNN	97.4 %	2.6833	7300	90.52 %
Cubic KNN	89.2 %	53.932	220	80.58 %
Weighted KNN	93.3 %	3.2339	6400	85.75 %
Boosted Trees	69.9 %	66.224	5300	62.98 %
Bagged Trees	96.8 %	24.097	5700	89.96 %
Subspace Discriminant	97.1 %	14.498	2300	95.03 %
Subspace KNN	98.5 %	42.95	290	88.89 %
RUSBoosted Trees	58.4 %	63.611	8000	55.22 %

Tabla 7

Resultados con PCA de 99 %, 36 predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	77.1 %	8.6192	13000	66.78 %
Medium Tree	55 %	4.9897	14000	49.49 %
Simple Tree	22.5 %	4.0926	14000	21.86 %
Linear Discriminant	94.2 %	2.6838	12000	90.62 %
Quadratic Discriminant	96.5 %	2.9484	11000	91.79 %
Linear SVM	99.1 %	35.71	2500	97.16 %
Quadratic SVM	99.4 %	35.345	1900	96.09 %
Cubic SVM	99.3 %	31.517	2000	95.74 %
Fine Gaussian SVM	59.8 %	62.197	460	28.65 %
Medium Gaussian SVM	99.2 %	25.822	2300	96.45 %
Coarse Gaussian SVM	95.3 %	40.527	800	93.97 %
Fine KNN	94.1 %	2.4368	8700	83.37 %
Medium KNN	93.9 %	2.5174	7900	85.85 %
Coarse KNN	86 %	3.3056	5900	83.22 %
Cosine KNN	97.3 %	2.7644	7300	89.71 %
Cubic KNN	91.2 %	42.094	290	83.01 %
Weighted KNN	94.7 %	3.0356	6400	86.41 %
Boosted Trees	68.4 %	52.316	6200	62.98 %
Bagged Trees	96.9 %	20.796	5100	91.43 %
Subspace Discriminant	97.1 %	13.025	2300	94.07 %
Subspace KNN	98.2 %	39.771	310	88.59 %
RUSBoosted Trees	59.1 %	51.778	5700	55.78 %

Tabla 8

Resultados con PCA de 98 %, 28 predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	77.1 %	3.7327	24000	66.83 %
Medium Tree	55 %	2.8495	24000	49.49 %
Simple Tree	22.5 %	2.3446	24000	21.86 %
Linear Discriminant	91.9 %	1.667	21000	89 %
Quadratic Discriminant	95.3 %	1.8841	19000	89.6 %
Linear SVM	98.5 %	21.406	3900	97.06 %
Quadratic SVM	99.1 %	25.58	2800	96.65 %
Cubic SVM	99 %	30.188	2100	95.79 %
Fine Gaussian SVM	79.9 %	64.531	660	50.15 %
Medium Gaussian SVM	98.7 %	26.639	2300	96.91 %
Coarse Gaussian SVM	92.9 %	39.713	820	91.02 %
Fine KNN	94.6 %	2.7482	8400	83.98 %
Medium KNN	94.1 %	2.3508	8500	86 %
Coarse KNN	86.5 %	3.3608	6100	82.56 %
Cosine KNN	95.6 %	2.2612	9400	87.98 %
Cubic KNN	91.4 %	37.382	340	82.91 %
Weighted KNN	95.1 %	2.3196	8800	86.31 %
Boosted Trees	69.4 %	44.667	5600	63.24 %
Bagged Trees	96.6 %	20.192	5000	90.82 %
Subspace Discriminant	95.5 %	12.492	2500	92.95 %
Subspace KNN	97.9 %	32.546	400	88.03 %
RUSBoosted Trees	59.2 %	44.708	5700	54.87 %

Tabla 9

Resultados con PCA de 97 %, 23 predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	77.5 %	3.3894	25000	65.72 %
Medium Tree	55 %	2.5551	25000	49.49 %
Simple Tree	22.5 %	2.1827	25000	21.86 %
Linear Discriminant	90.9 %	1.6127	22000	87.98 %
Quadratic Discriminant	94.1 %	1.7529	19000	86.71 %
Linear SVM	98.1 %	20.461	4000	95.84 %
Quadratic SVM	98.8 %	20.266	3400	95.13 %
Cubic SVM	98.7 %	20.677	3200	94.78 %
Fine Gaussian SVM	89.3 %	42.443	1100	62.83 %
Medium Gaussian SVM	97.8 %	22.077	2800	95.74 %
Coarse Gaussian SVM	90.1 %	33.947	1400	89.5 %
Fine KNN	94.4 %	2.4796	9200	82.3 %
Medium KNN	93.5 %	2.2383	11000	84.38 %
Coarse KNN	87.1 %	2.8747	7500	82.1 %
Cosine KNN	94.7 %	2.0483	11000	83.82 %
Cubic KNN	90.8 %	27.28	470	81.24 %
Weighted KNN	94.3 %	2.7154	8500	84.89 %
Boosted Trees	67.3 %	37.158	6000	63.34 %
Bagged Trees	96.5 %	18.958	5500	89.4 %
Subspace Discriminant	94.2 %	11.236	2600	90.87 %
Subspace KNN	97.6 %	27.815	470	87.27 %
RUSBoosted Trees	59 %	37.06	5400	50.36 %

Tabla 10

Resultados con PCA de 96 %, 20 predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	76.7 %	3.2733	25000	65.82 %
Medium Tree	55 %	2.4379	26000	49.49 %
Simple Tree	22.5 %	2.0904	26000	21.86 %
Linear Discriminant	90.3 %	1.6511	21000	87.12 %
Quadratic Discriminant	93.7 %	1.8116	20000	87.53 %
Linear SVM	97.6 %	19.742	4200	94.93 %
Quadratic SVM	98.5 %	19.869	3500	95.13 %
Cubic SVM	98.6 %	20.325	3300	94.32 %
Fine Gaussian SVM	94.5 %	34.285	1400	80.17 %
Medium Gaussian SVM	97.6 %	19.87	2800	94.93 %
Coarse Gaussian SVM	89.4 %	29.536	1400	87.83 %
Fine KNN	94.8 %	2.4226	9500	83.92 %
Medium KNN	93.6 %	2.2357	10000	84.99 %
Coarse KNN	88 %	2.4112	8700	82.81 %
Cosine KNN	94.7 %	2.4801	9900	85.19 %
Cubic KNN	91.8 %	23.787	580	82 %
Weighted KNN	95.1 %	2.4392	9800	86.26 %
Boosted Trees	69.2 %	32.388	6100	63.34 %
Bagged Trees	96.4 %	17.933	5200	89.6 %
Subspace Discriminant	93.5 %	11.669	2500	89.91 %
Subspace KNN	97.6 %	25.38	540	87.02 %
RUSBoosted Trees	60.8 %	33.047	5200	54.56 %

Tabla 11

Resultados con PCA de 95 %, 18 predictores

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	76.6 %	3.2256	26000	64.47 %
Medium Tree	54.1 %	2.3437	26000	48.38 %
Simple Tree	22.5 %	2.0066	27000	21.86 %
Linear Discriminant	89.6 %	1.5762	24000	86.36 %
Quadratic Discriminant	93.3 %	1.7055	20000	85.9 %
Linear SVM	97.2 %	19.467	4200	93.76 %
Quadratic SVM	98.3 %	19.7	3600	93.61 %
Cubic SVM	98.2 %	20.001	3400	93.51 %
Fine Gaussian SVM	95.7 %	32.967	1500	83.16 %
Medium Gaussian SVM	96.9 %	20.099	2800	93.81 %
Coarse Gaussian SVM	88.3 %	31.398	1400	86.61 %
Fine KNN	95.2 %	2.558	8700	83.32 %
Medium KNN	94 %	2.0246	12000	84.74 %
Coarse KNN	88 %	2.3138	9700	82.61 %
Cosine KNN	94.6 %	2.6172	9300	84.08 %
Cubic KNN	92.4 %	21.6	550	82.4 %
Weighted KNN	95.2 %	2.2444	11000	85.29 %
Boosted Trees	69.4 %	30.966	5700	63.08 %
Bagged Trees	95.8 %	17.559	5600	88.49 %
Subspace Discriminant	92.9 %	11.977	2400	87.98 %
Subspace KNN	97.3 %	26.222	530	86.26 %
RUSBoosted Trees	58.6 %	33.489	5000	51.88 %

Tabla 12

Resultados con Multiescalado de 128+64 con Pirámide Gaussiana

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	94.6 %	15.467	32000	82.15 %
Medium Tree	56.7 %	7.763	39000	54.31 %
Simple Tree	23.9 %	5.3898	41000	22.51 %
Linear Discriminant	94.8 %	3.9461	12000	92.01 %
Quadratic Discriminant	98.9 %	3.8038	13000	93.76 %
Linear SVM	100 %	49.809	950	99.19 %
Quadratic SVM	100 %	63.157	440	99.09 %
Cubic SVM	99.9 %	67.87	380	98.78 %
Fine Gaussian SVM	18.6 %	152.98	150	5.88 %
Medium Gaussian SVM	100 %	82.941	300	97.67 %
Coarse Gaussian SVM	99.8 %	81.047	310	98.88 %
Fine KNN	99.8 %	9.0787	1800	95.74 %
Medium KNN	99.7 %	8.0202	1700	97.26 %
Coarse KNN	97.3 %	7.6139	1700	97.16 %
Cosine KNN	98.8 %	7.3754	1900	93.61 %
Cubic KNN	98.7 %	257.85	46	93.76 %
Weighted KNN	99.8 %	7.3151	1800	97.36 %
Boosted Trees	95 %	188.17	4900	87.17 %
Bagged Trees	99.7 %	23.767	5400	97.77 %
Subspace Discriminant	100 %	31.371	980	99.75 %
Subspace KNN	99.8 %	96.214	130	92.95 %
RUSBoosted Trees	73.5 %	182.6	5500	72.41 %

Tabla 13

Resultados con Multiescalado de 128+32 con Pirámide Gaussiana

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	94.3 %	17.9127	26000	79.2 %
Medium Tree	48 %	11.944	28000	38.89 %
Simple Tree	23.9 %	8.1561	29000	22.41 %
Linear Discriminant	94.8 %	2.9577	11000	91.62 %
Quadratic Discriminant	98.7 %	4.9799	9800	90.06 %
Linear SVM	99.9 %	53.557	940	97.92 %
Quadratic SVM	99.9 %	71.624	390	97.26 %
Cubic SVM	99.8 %	74.753	360	96.65 %
Fine Gaussian SVM	10.8 %	153.8	150	5.48 %
Medium Gaussian SVM	99.9 %	87.107	290	94.02 %
Coarse Gaussian SVM	99.7 %	77.91	320	97.26 %
Fine KNN	99.7 %	7.8392	1800	90.37 %
Medium KNN	99.5 %	7.7273	1800	93.76 %
Coarse KNN	96.6 %	8.085	1700	94.32 %
Cosine KNN	98.2 %	7.857	1800	90.26 %
Cubic KNN	97.8 %	264.74	46	87.83 %
Weighted KNN	99.7 %	6.7574	2100	93.76 %
Boosted Trees	95 %	174.49	5500	86.11 %
Bagged Trees	99.8 %	23.24	5400	97.06 %
Subspace Discriminant	100 %	29.9	930	99.54 %
Subspace KNN	99.7 %	92.655	130	90.97 %
RUSBoosted Trees	76.9 %	179.82	5700	45.18 %

Tabla 14

Resultados con Multiescalado de 128+64+32 con Pirámide Gaussiana

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	94.2 %	33.213	17000	79.61 %
Medium Tree	50.5 %	23.262	18000	41.73 %
Simple Tree	22.8 %	13.327	19000	22.41 %
Linear Discriminant	94.4 %	5.1137	6900	91.28 %
Quadratic Discriminant	98.1 %	7.9168	6300	89.01 %
Linear SVM	99.9 %	77.91	620	97.92 %
Quadratic SVM	99.9 %	102.87	280	97.31 %
Cubic SVM	99.9 %	110.2	250	96.70 %
Fine Gaussian SVM	8.7 %	211.9	110	5.12 %
Medium Gaussian SVM	99.9 %	120.88	210	93.91 %
Coarse Gaussian SVM	99.6 %	108.06	240	96.81 %
Fine KNN	99.8 %	12.33	1100	91.53 %
Medium KNN	99.6 %	12.062	1100	94.98 %
Coarse KNN	96.7 %	11.95	1100	94.17 %
Cosine KNN	98.1 %	10.181	1400	90.16 %
Cubic KNN	98.3 %	359.13	33	89.25 %
Weighted KNN	99.7 %	10.595	1300	94.88 %
Boosted Trees	95.1 %	274.87	4600	87.88 %
Bagged Trees	99.8 %	28.028	3700	97.51 %
Subspace Discriminant	100 %	62.764	340	99.85 %
Subspace KNN	99.9 %	142.79	87	91.58 %
RUSBoosted Trees	64.2 %	280.42	4000	54.82 %

Tabla 15

Resultados con Multiescalado de 128+64 con Método Propio

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	95.1 %	18.571	23000	81.85 %
Medium Tree	58.6 %	12.843	26000	54.51 %
Simple Tree	24 %	8.8027	27000	22.52 %
Linear Discriminant	95.4 %	3.4061	10000	93.56 %
Quadratic Discriminant	99.4 %	5.3344	9400	95.69 %
Linear SVM	100 %	43.902	1200	99.29 %
Quadratic SVM	100 %	64.585	440	99.14 %
Cubic SVM	100 %	72.738	400	99.09 %
Fine Gaussian SVM	15.3 %	156.72	160	5.78 %
Medium Gaussian SVM	100 %	86.711	320	98.07 %
Coarse Gaussian SVM	99.8 %	75.523	360	98.94 %
Fine KNN	99.9 %	9.0266	1700	96.6 %
Medium KNN	99.7 %	8.1905	1700	97.72 %
Coarse KNN	96.9 %	8.479	1600	96.65 %
Cosine KNN	99.1 %	8.107	1700	95.28 %
Cubic KNN	98.4 %	293.01	41	93.41 %
Weighted KNN	99.7 %	8.1277	1700	97.68 %
Boosted Trees	94.9 %	205.93	4300	87.63 %
Bagged Trees	99.8 %	28.169	4000	96.81 %
Subspace Discriminant	99.9 %	38.513	740	99.59 %
Subspace KNN	99.1 %	101.91	120	92.04 %
RUSBoosted Trees	77.2 %	201.33	4400	60.24 %

Tabla 16

Resultados con Multiescalado de 128+32 con Método Propio

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	94.7 %	18.327	23000	82.25 %
Medium Tree	55.6 %	11.992	26000	54.51 %
Simple Tree	24 %	8.324	27000	22.52 %
Linear Discriminant	94.8 %	3.2999	10000	92.85 %
Quadratic Discriminant	99.2 %	5.4831	9500	95.33 %
Linear SVM	99.9 %	57.691	910	98.73 %
Quadratic SVM	99.9 %	76.398	380	98.63 %
Cubic SVM	100 %	78.883	370	98.23 %
Fine Gaussian SVM	8.1 %	157.14	160	5.02 %
Medium Gaussian SVM	99.9 %	93.507	270	96.96 %
Coarse Gaussian SVM	99.7 %	81.089	330	98.33 %
Fine KNN	99.8 %	8.637	1700	94.83 %
Medium KNN	99.5 %	8.2118	1700	97.01 %
Coarse KNN	96.1 %	8.53	1600	96.65 %
Cosine KNN	98.7 %	8.2167	1700	95.23 %
Cubic KNN	98.1 %	290.62	41	92.55 %
Weighted KNN	99.6 %	8.1549	1700	96.75 %
Boosted Trees	94.8 %	174.28	4800	88.24 %
Bagged Trees	99.9 %	23.993	4400	97.01 %
Subspace Discriminant	99.9 %	34.063	790	99.54 %
Subspace KNN	99.3 %	92.514	140	93.15 %
RUSBoosted Trees	74.5 %	184.2	5000	60.09 %

Tabla 17

Resultados con Multiescalado de 128+64+32 con Método Propio

Modelo	Aciertos Val.	Tiempo Entr. (sg)	Vel. Pred. (obs/sg)	Aciertos Test
Complex Tree	94.3 %	24.178	18000	81.49 %
Medium Tree	58.1 %	17.857	19000	54.51 %
Simple Tree	22.8 %	12.184	19000	22.52 %
Linear Discriminant	94.8 %	5.0371	6500	92.95 %
Quadratic Discriminant	99.3 %	7.6173	6000	95.44 %
Linear SVM	99.9 %	84.203	570	98.88 %
Quadratic SVM	99.9 %	99.718	290	98.94 %
Cubic SVM	99.9 %	107.19	270	98.53 %
Fine Gaussian SVM	6.6 %	209.62	110	5.02 %
Medium Gaussian SVM	99.9 %	123.56	210	97.16 %
Coarse Gaussian SVM	99.7 %	109.31	240	98.53 %
Fine KNN	99.9 %	12.701	1100	96.04 %
Medium KNN	99.7 %	12.325	1100	97.21 %
Coarse KNN	96.1 %	12.576	1100	95.44 %
Cosine KNN	98.9 %	12.567	1100	92.7 %
Cubic KNN	98.5 %	433.28	27	93.61 %
Weighted KNN	99.7 %	12.431	1100	97 %
Boosted Trees	94.7 %	310.78	3400	86.11 %
Bagged Trees	99.8 %	34.507	3400	97.16 %
Subspace Discriminant	99.9 %	71.495	330	99.65 %
Subspace KNN	99 %	152.77	81	91.94 %
RUSBoosted Trees	75.7 %	273.47	3800	54.41 %

B. Manual de usuario de la aplicación Android

Instalación

El APK de la aplicación se proporciona en el CD entregado junto a esta memoria. Es necesario copiar el APK en el dispositivo móvil y ejecutar el archivo para proceder con la instalación. Se recomienda su ejecución en dispositivos con Android 6.0 o superior.

Captura de Datos

Una vez instalada y ejecutada la aplicación mostrará la interfaz de captura de datos (Figura 20). Desde esta podrá seleccionarse un tiempo de 'delay' o espera, que será el tiempo que pasará entre que el usuario pulsa el botón para iniciar la captura de datos y el momento en el que el dispositivo la empezará. Con esto el usuario puede elegir un intervalo de tiempo cómodo para ajustar el dispositivo en la posición correcta de captura y que comience a caminar. También se muestra el número de capturas almacenadas actualmente en la memoria del dispositivo.

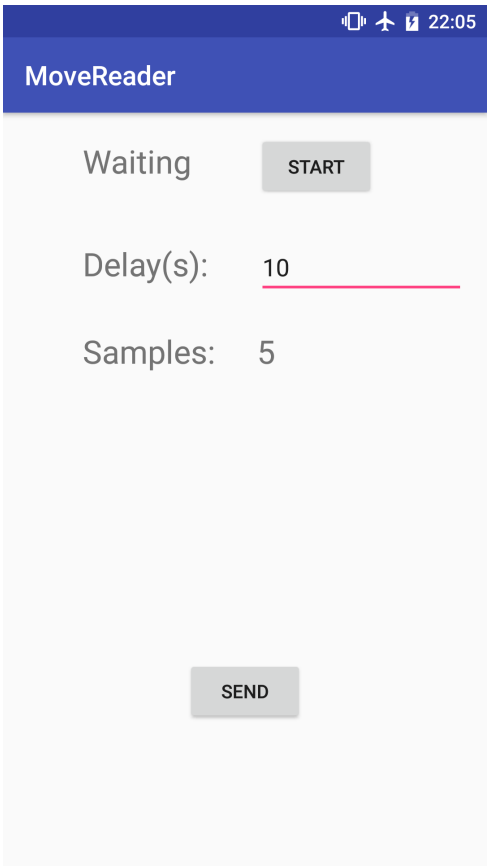
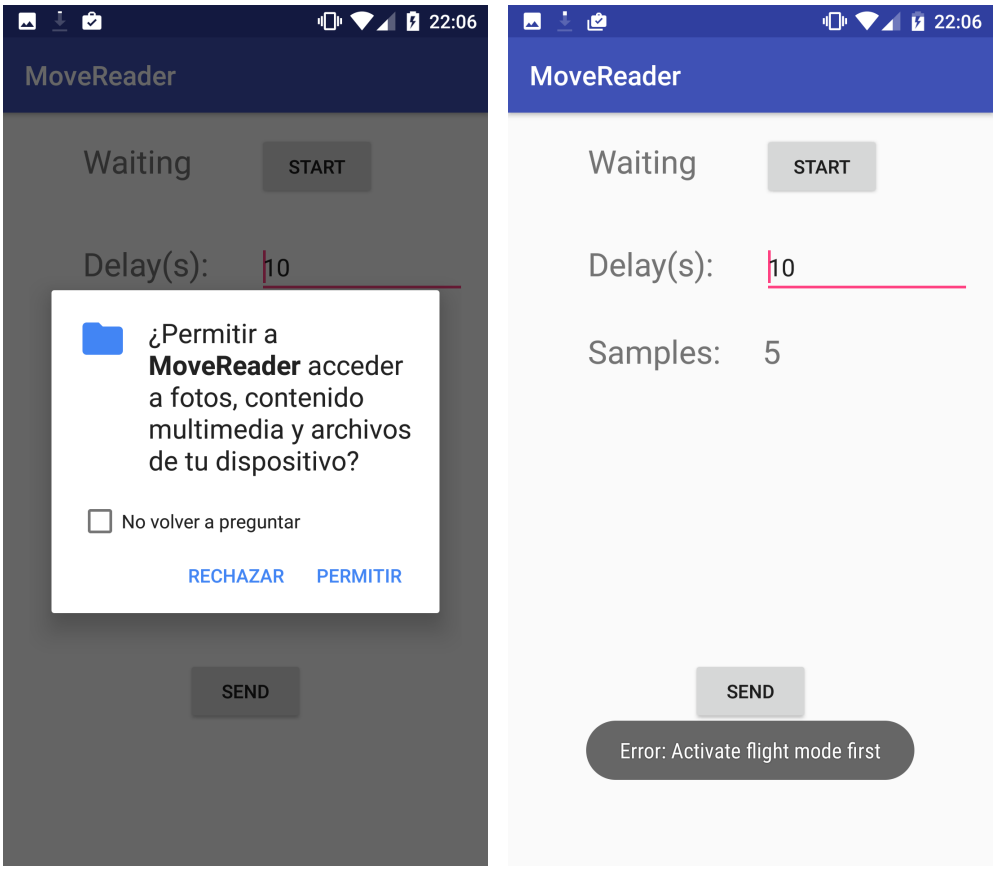


Figura 20: Menú de Captura de Datos de la APP

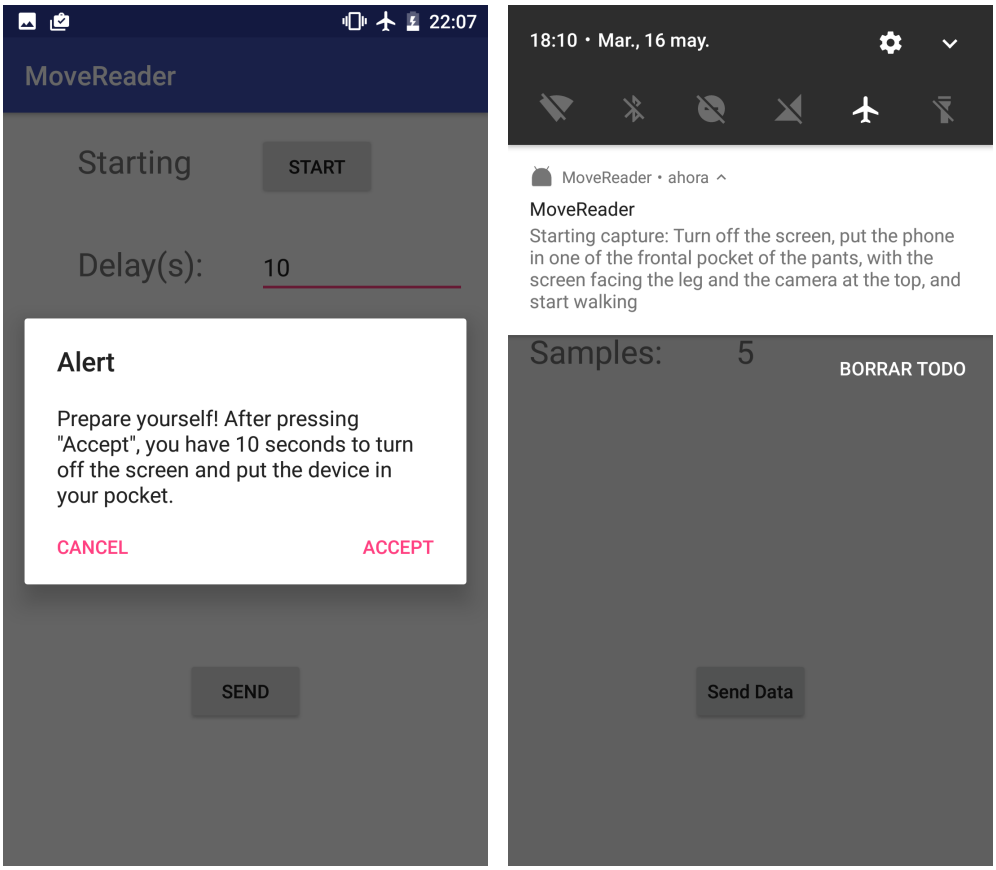
Cuando el usuario pulsa el botón de inicio de capturas (botón 'Start') la aplicación comprobará si se le han dado permisos de escritura y si el móvil tiene activado el modo avión, con el objetivo de poder escribir las capturas en el almacenamiento del dispositivo y que posibles notificaciones u otras fuentes de vibración no interfieran con la prueba. Si lo primero fallara, se le mostraría al usuario una ventana emergente en la que tendría que asignar permisos de almacenamiento antes de poder volver a intentarlo. En cambio, si el problema fuera el modo avión el móvil solo mostraría un pequeño popup avisando al usuario de que él debe activar manualmente el modo avión (Figura 21).



(a) Petición de Permiso de Almacenamiento (b) Aviso Modo Avión no activado

Figura 21: Posibles Errores en Captura de Datos

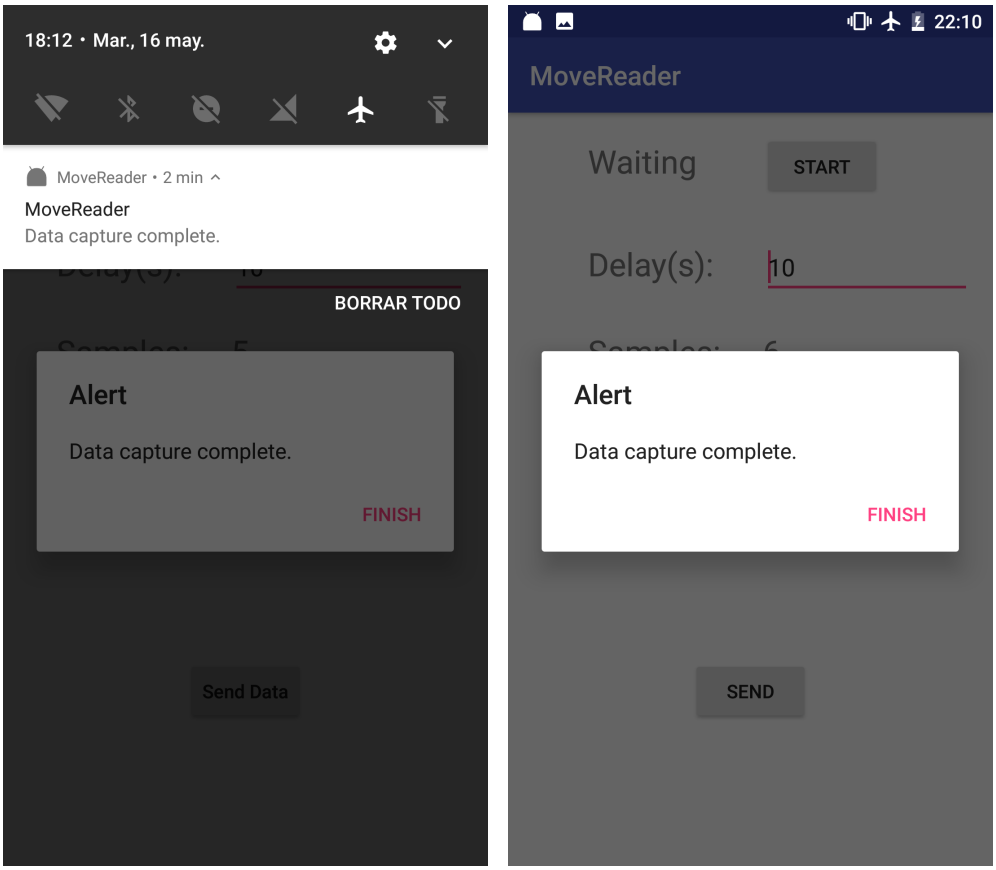
Si la petición de comienzo de captura se realiza correctamente, una ventana emergente avisará al usuario de que debe pulsar el botón 'Accept', apagar la pantalla y colocar el móvil en su bolsillo en la posición indicada antes de que finalice el tiempo indicado (Figura 22). Al finalizar este tiempo una notificación avisará con sonido y vibración al usuario de que la captura ya ha comenzado y debe empezar a caminar.



(a) Ventana emergente de aviso (b) Notificación de aviso

Figura 22: Avisos de comienzo de Captura de Datos

Al finalizar la captura de datos correctamente, la aplicación lanzará una segunda notificación con sonido y vibración junto con una ventana emergente (Figura 23).

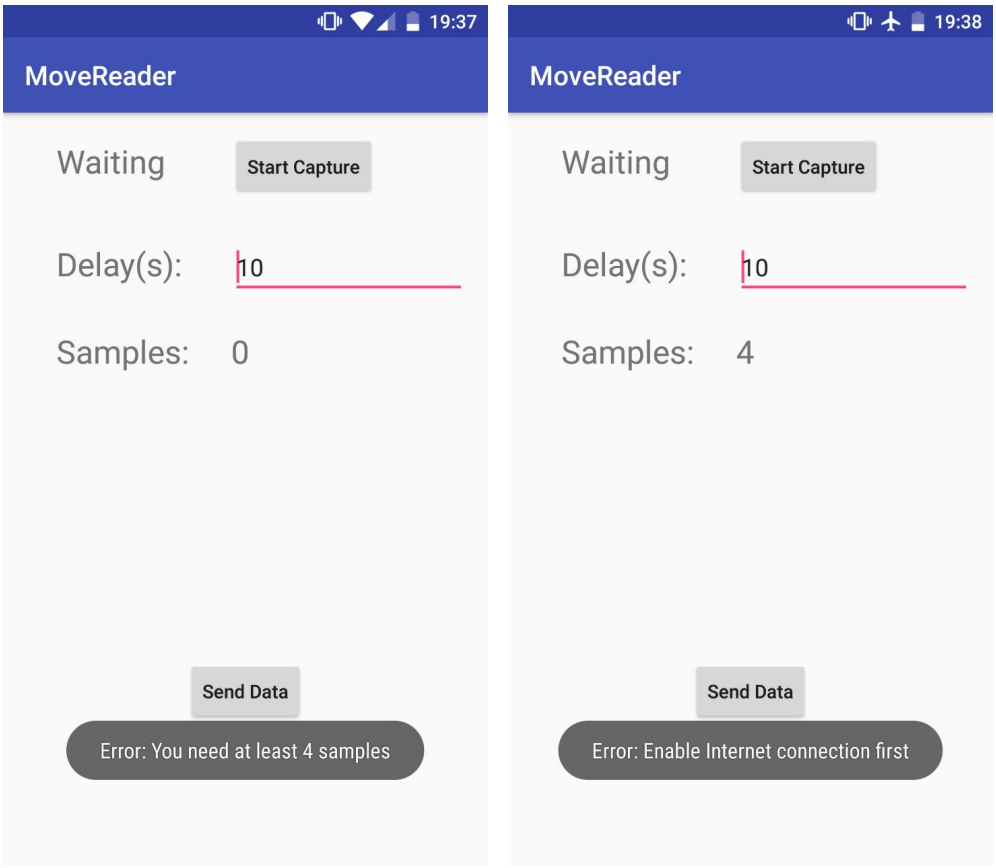


(a) Notificación de aviso (b) Ventana emergente de aviso

Figura 23: Avisos de fin de Captura de Datos

Envío de Datos al Servidor

Una vez se han tomado 4 o más capturas de datos la aplicación permitirá el envío de estas a un servidor para su posterior análisis. A esta función se puede acceder pulsando el botón 'Send' situado en la zona inferior de la interfaz de captura de datos, y su uso también requiere que el dispositivo tenga una conexión a Internet activa (Figura 24).



(a) Aviso capturas mínimas no alcanzadas (b) Aviso falta de conexión a Internet

Figura 24: Posibles Errores en Envío de Datos

Se mostrará una nueva interfaz con un cuestionario (Figura 25) que servirá de información de referencia a futuros investigadores que hagan uso del dataset.

- **Email**: Correo electrónico del usuario. Será usado únicamente como un sistema para identificar la procedencia de los datos, y poder agrupar así todos los datos que procedan del mismo contribuidor.
- **Observations**: Campo para que el usuario pueda dar la información adicional que considere necesaria sobre las capturas entregadas (calzado inusual, carga de peso adicional, etc). Este campo no es obligatorio.
- **Gender**: Sexo al que pertenece el usuario.
- **Birthdate**: Fecha de nacimiento del usuario.

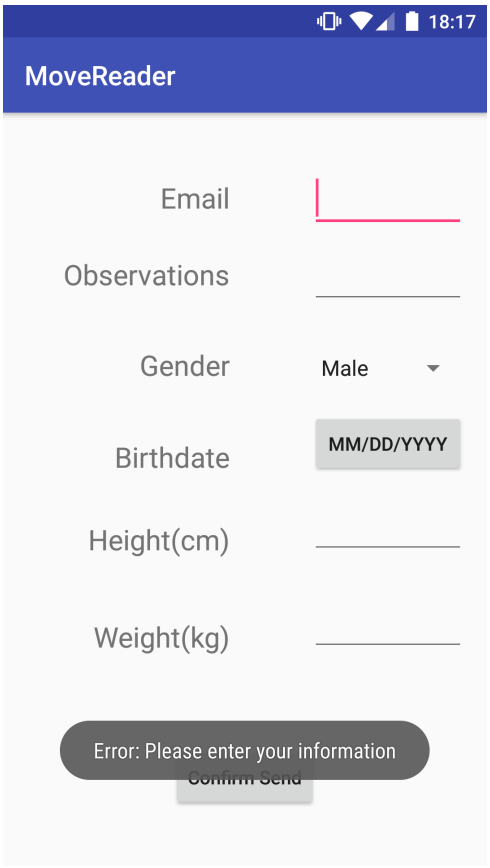


Figura 25: Error en Envío de datos por falta de campos obligatorios

- **Height**: Altura (en centímetros) del usuario.
- **Weight**: Peso (en kilogramos) del usuario.

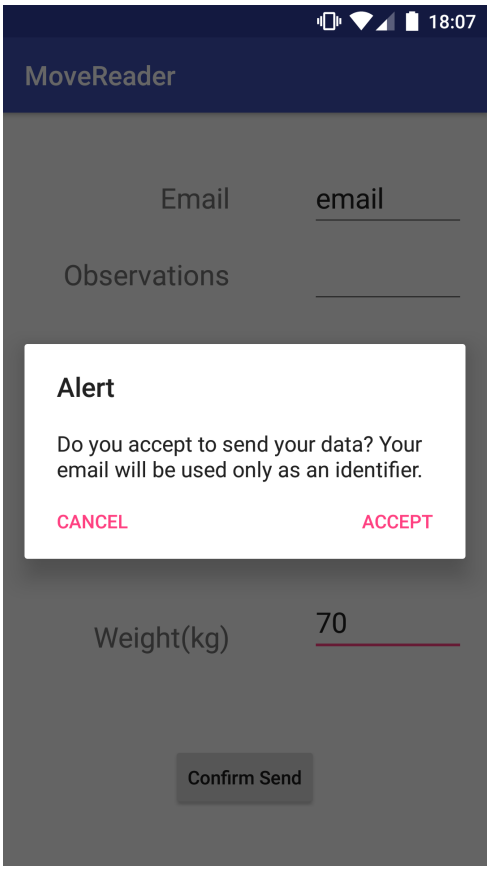
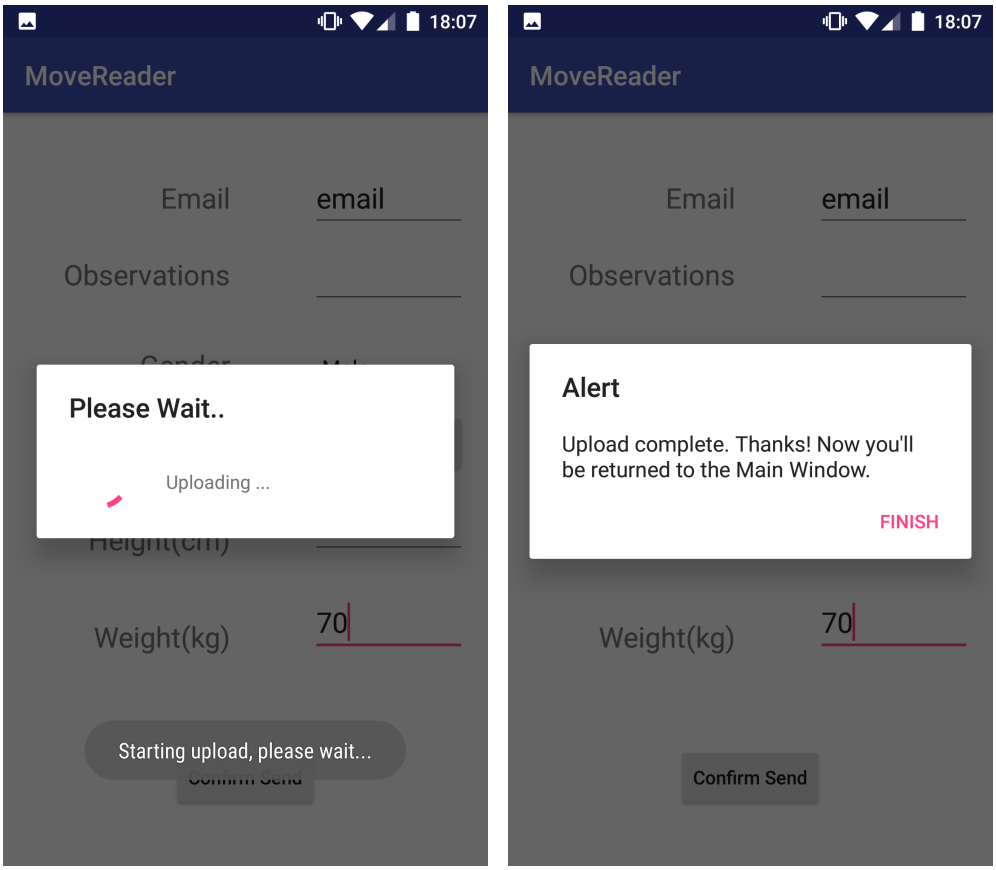


Figura 26: Ventana Emergente de confirmación de Envío de Datos

Tras rellenar todos los campos y confirmar el envío (botón 'Confirm Send') se le pedirá al usuario una última confirmación (Figura 26) en la que el usuario aceptará el envío de sus datos.



(a) Proceso de subida de datos (b) Ventana emergente fin de subida de datos

Figura 27: Proceso de Envío de Datos

Finalmente, una serie de ventanas emergentes (Figura 27) mantendrán informado al usuario del inicio y fin de la subida de datos al servidor.

C. Índice de figuras

D. Índice de tablas