TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

StressDetector

Sistema distribuido basado en tecnologías móviles/wearables y Machine Learning como aproximación al estudio del estrés

Autor

Ismael José Moyano Romero

Directores

Francisco Manuel García Moreno

José Luis Garrido Bullejos

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Julio de 2020

StressDetector:
Sistema distribuido basado en tecnologías móviles/wearables y Machine Learning como aproximación al estudio del estrés

Ismael José Moyano Romero

Palabras clave: sistemas distribuidos, tecnologías móviles/wearables, Machine Learning, estrés, eSalud.

Resumen

Los problemas de salud causados por el estrés están en auge. Cada vez más personas sufren del mismo y les dificulta su día a día sin que lleguen siquiera a darse cuenta, situación ante la cual se plantea este proyecto. StressDetector es un sistema distribuido basado en tecnologías móviles, wearables y Machine Learning, cuyo objetivo es proporcionar una aproximación a detectar y avisar automáticamente al usuario cuando pueda estar sufriendo niveles elevados de estrés.

Utilizando un smartphone, un smartwatch y una pulsera Empatica E4 para tomar datos fisiológicos del usuario, el sistema los envía a un servidor que los procesa en tiempo real con un algoritmo de Machine Learning para aprender y detectar cuándo dichos datos pueden conllevar niveles altos de estrés para el usuario, ante lo cual se alertará al mismo. El sistema de sensores no es intrusivo y la aplicación sólo requiere ser iniciada, permitiendo así al usuario desarrollar su vida de forma normal.

El desarrollo del sistema hace uso de tecnologías actuales y avanzadas, y su aplicación posee un carácter experimental. Su objetivo es explorar la funcionalidad que podría tener un sistema de detección automático de estrés no intrusivo con la tecnología disponible actualmente, además de abrir las puertas a una posible futura aplicación comercial para la población general tras su futura extensión.

StressDetector:
A distributed system based on mobile/wearable and Machine Learning technologies as an approach to the study of stress

Ismael José Moyano Romero

Keywords: distributed systems, mobile/wearable technologies, Machine Learning, stress, eHealth.

Abstract

Stress-caused health issues are on the rise. Everyday, more and more people suffer stress and it cripples their everyday without them even noticing, and that's why this project came to be. StressDetector is a distributed system based on mobile/wearable and Machine Learning technologies, with the objective of detecting and warning the user when they may suffer high stress levels.

Using a smartphone, a smartwatch and the Empatica E4 wristband to take physiological data from the user, the system sends them to a server where they are processed in real time by a Machine Learning algorithm to learn and detect when said data can mean high stress levels for the user, in which case they will be alerted. The use of sensors is non-intrusive and the software application just requires to be initiated, thus allowing the user to live a normal life.

This project is focused on the development of a system that makes use of current and advanced technologies, and also provides an experimental side of its application. The objective is to explore the functionality that an automatic, non-intrusive stress detector would have with the technologies available today, and to set the record for a future commercial product useful for the general public through its extension.

Yo, Ismael José Moyano Romero, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77148120W, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Ismael José Moyano Romero

Granada a 6 de julio de 2020

D. **Francisco Manuel García Moreno**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D. **José Luis Garrido Bullejos**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***StressDetector: Sistema distribuido basado en tecnologías móviles/wearables y Machine Learning como aproximación al estudio del estrés***, ha sido realizado bajo su supervisión por **Ismael José Moyano Romero**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 6 de julio de 2020 .

Los directores:

Francisco Manuel García Moreno **José Luis Garrido Bullejos**

Dedico este trabajo a mi familia, que siempre me ha apoyado incondicionalmente, y sin los que no habría llegado hasta donde estoy ahora.

A Jose, fuente de mi amor por el mundo de la informática.

A Ángeles, que ha estado siempre ahí cuando la he necesitado.

A Amanda, que tiene un corazón que no le cabe en el pecho.

Gracias por todo, os quiero.

Índice

Capítulo 1. Introducción

En este capítulo se proporcionará una introducción al proyecto, abarcando el motivo de su realización, los objetivos a alcanzar, y datos como su planificación temporal y su presupuesto.

1.1 Motivación del proyecto

El estrés es un problema prevalente en la sociedad actual. En mayor o menor medida, todo el mundo lo ha sufrido, y sus consecuencias pueden llegar a ser nefastas. Según el Instituto Nacional de Estadística [1], el ciudadano medio posee un nivel de estrés de 4,18/7, cifra más que alarmante. Pero el mayor peligro del estrés es cuando el sujeto no es consciente del mismo, ante lo cual es bastante probable que no le ponga solución. Es ahí donde entra este proyecto, que se ha denominado StressDetector.

StressDetector busca explorar los límites de la tecnología actual a la hora de desarrollar un detector automático del estrés que pueda funcionar de forma constante y no intrusiva, y con los propios dispositivos que pueden portar los usuarios (teléfono inteligente, reloj inteligente, etc.). Aunque ya existen sistemas para detectar el estrés, están mucho más orientados hacia el ámbito médico y no hacia el uso cotidiano. Y aunque actualmente se requieren aparatos específicos, certificados, y de precio elevado para poder obtener datos fisiológicos de forma precisa, los dispositivos móviles que el usuario lleva consigo todo el tiempo poseen sensores cada vez más precisos que pronto podrán equipararse a los utilizados en medicina, y que seguramente en unos años lleguen a superarlos. StressDetector busca sentar las bases para un posible futuro proyecto que pueda ser utilizado por el usuario corriente para medir su nivel de estrés de forma sencilla y satisfactoria con los dispositivos que tenga ya disponibles.

Este proyecto surgió por mi motivación a la hora de realizar un Trabajo de Fin de Grado relacionado con tecnologías móviles. Contacté con el Profesor José Luis Garrido Bullejos, a quien conocía de haber sido mi profesor en las asignaturas Sistemas Operativos y Desarrollo de Sistemas Distribuidos, ya que sabía que él tenía amplia experiencia en el campo. Aunque todavía no tenía claro exactamente qué tipo de proyecto quería realizar, el profesor me propuso realizar un detector de estrés para personas mayores basado en un sistema distribuido, idea que me entusiasmó. Posteriormente vimos que el proyecto realmente podría ser útil para todo tipo de personas, así que decidimos ampliar su público objetivo. El profesor propuso también contar con el Profesor Francisco Manuel García Moreno en el proyecto como co-tutor, ya que tenía experiencia en el desarrollo de sistemas distribuidos similares y en el desarrollo de aplicaciones de Machine Learning. La propuesta definitiva de cómo se estructuraría el proyecto se decidió en una reunión posterior, junto con ambos tutores.

1.2 Objetivos

Este proyecto tiene como objetivo principal el desarrollo de un sistema distribuido que pueda ser novedoso y de utilidad a la detección automática del estrés. Se plantea inicialmente como una aproximación experimental, a falta de un estudio, análisis y evaluación detallada de los resultados en cuanto a su posible eficacia. Así mismo, durante el desarrollo del proyecto se desea aprender y aplicar técnicas y tecnologías actuales para el desarrollo de sistemas software de interés en el dominio de la eSalud.

Para alcanzar el objetivo principal se plantean los siguientes objetivos específicos:

- Diseñar e implementar sistemas que sean sencillos y fáciles de utilizar abordando las principales fases de desarrollo del software. Aunque el sistema final puede poseer cierta complejidad interna, es importante que el sistema sea fácil de utilizar por el usuario final, de cara a que no haya problemas posteriormente a la hora de validarlo con los usuarios.

- Aprender a desarrollar aplicaciones Android que formen parte de un sistema distribuido mayor, con requisitos de comunicación con otros dispositivos y tecnologías. Aunque se tiene experiencia anterior trabajando con Android, nunca se había desarrollado una aplicación que fuese parte de un sistema distribuido, y eso llamaba mucho la atención.

- Aprender a desarrollar una aplicación Wear OS para que sea parte de un sistema distribuido, también con requisitos de comunicación con otros dispositivos. Nunca se había desarrollado una aplicación para wearables, y era un campo de la tecnología móvil que se quería explorar.

- Conocer y aplicar técnicas de análisis de datos, incluyendo representación, preprocesamiento, visualización, etc. Para que la aplicación pueda ser realmente útil, es extremadamente importante asegurar que los datos son fiables. Es por eso que en la mayoría de los algoritmos (p.ej. Machine Learning), es un aspecto esencial, puesto que se facilita el aprendizaje del mismo asegurando siempre un cierto porcentaje de precisión en los resultados.

- Adquirir conocimientos y aprender a programar algoritmos de Machine Learning, con la intención de conseguir desarrollar sistemas software que tengan la capacidad de autoaprendizaje y mejora. Al haber cursado la mención de Ingeniería del Software, la experiencia en el campo del Machine Learning era prácticamente nula, por lo que sería un buen complemento investigar las posibilidades del Machine Learning y el cómo se podría aplicar al proyecto.

- Asegurar que los datos se procesan y se devuelven al usuario con fluidez y de la forma más aproximada al tiempo real. Dada la naturaleza de la propuesta es esencial que, una vez el sistema detecte que el usuario tiene niveles de estrés por encima de lo recomendado, se notifique al mismo de forma oportuna. La integración de servicios externos en el sistema puede aliviar la carga de procesamiento de los datos en el propio teléfono, y permitir visualizar los mismos sin retraso alguno en el dispositivo del usuario.

- Diseñar un sistema que pueda ser fácilmente escalable. Aunque el sistema está diseñado para funcionar con unos dispositivos wearables concretos, es importante también que el sistema pueda integrar fácilmente futuros dispositivos, mejorando así sus capacidades y funcionalidad.

- Investigar y analizar aplicaciones relacionadas con el campo de la eSalud. Por ejemplo, sobre las señales fisiológicas derivadas del estrés. A pesar de que ya ha habido algunos estudios relacionados, este es un campo en auge, ya que la tecnología disponible para ello todavía es reciente.

1.3 Planificación temporal

Para representar la planificación temporal del proyecto, se incluye un diagrama de Gantt (Figuras 1 y 2). En total, el proyecto ha durado desde su planteamiento a mediados de noviembre hasta que ha sido finalizado a principios de julio. Sin embargo, la mayor parte del trabajo se ha realizado durante el periodo del segundo cuatrimestre del año académico, en concreto desde el comienzo del desarrollo de la aplicación Android el 7 de marzo a la finalización del algoritmo de Machine Learning y el preprocesamiento de los datos el 30 de junio, más 5 días extra para terminar de elaborar la memoria; siendo un total de 121 días, o 17 semanas.

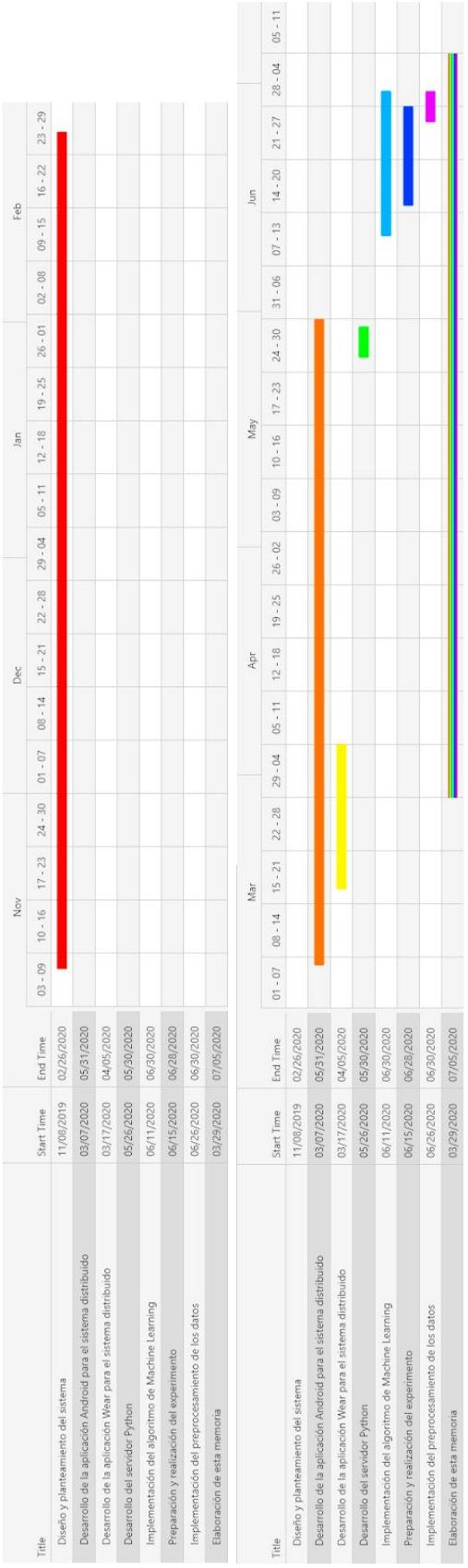


Figura 1. Diagrama de Gantt, parte 1.
Figura 2. Diagrama de Gantt, parte 2.

1.4 Presupuesto

Para el desarrollo del proyecto se supone la contratación de un desarrollador con el título de Ingeniero en Informática, con alguna experiencia previa en el desarrollo en plataformas móviles. Se consideran sueldos brutos (incluye cuotas a la Seguridad Social, etc.) por mes trabajando a tiempo completo. El proyecto requerirá de 6 meses para su desarrollo. Se considera el precio medio de mercado para el ordenador, el smartphone y el smartwatch. Tanto el smartphone como el smartwatch requerirán tener una versión de Android igual o superior a Android 5.1.

Gastos	**Coste**
1 desarrollador con experiencia durante 6 meses	2.700€/mes * 6 meses = 16.200€
1 ordenador	800€
1 smartphone	300€
1 smartwatch	300€
1 Empatica E4	1.700€
Total	**19.300€**

Tabla 1. Cálculo del presupuesto del proyecto.

Capítulo 2. Tecnologías utilizadas

En este capítulo se presentarán las distintas tecnologías utilizadas en el proyecto, incluyendo tecnologías móviles, wearables, el protocolo MQTT y Machine Learning.

2.1 Desarrollo en Android

2.1.1 Android y su historia

A pesar de la diversidad de Sistemas Operativos propietarios de la era móvil, una vez comenzó la era del smartphone o teléfono inteligente, sólo prevalecieron 3: Android (de Google), iOS (de Apple), y Windows Phone (de Microsoft). Mientras que iOS y Windows Phone solo están disponibles en los teléfonos fabricados por su propia compañía (con algunas excepciones por parte de Windows Phone), Android es de código abierto y, por lo tanto, está disponible para usar por cualquiera. Es por esto que, prácticamente desde su lanzamiento, Android ha sido el Sistema Operativo móvil más extendido en el mercado.

Comprado por Google en 2005 y lanzado al mercado oficialmente en 2008, Android tiene ya más de 12 años y 10 versiones a sus espaldas. Android está basado en el Sistema Operativo Linux. A pesar de estar, actualmente, en su versión 10.0 (con la versión 11.0 en beta), esto no es más que la versión del Sistema Operativo base. Al ser de código abierto, un gran número de compañías lo han hecho suyo y han desarrollado sus propias capas sobre el mismo, algunas ya casi tan consolidadas como el sistema base: One UI (de Samsung), MIUI (de Xiaomi) o OxygenOS (de OnePlus), por ejemplo. A pesar de ello, un gran número de características esenciales del Sistema Operativo dependen directamente de Google y de sus servicios. Es por ello que a la hora de desarrollar para Android es extremadamente recomendable utilizar las API y los tutoriales que facilita la compañía.

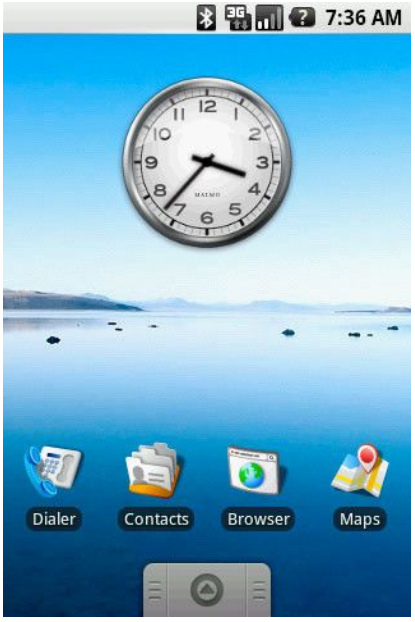


Figura 3. Android 1.0.

2.1.2 Desarrollo con Android Studio

Basado en el software IntelliJ IDEA de JetBrains, Android Studio sustituyó a Eclipse como entorno de desarrollo integrado (IDE, por sus siglas en inglés) oficial de Android en 2013. Android Studio está disponible para los principales Sistemas Operativos de sobremesa: Windows, Linux y macOS.

Android Studio enfoca el desarrollo de aplicaciones Android utilizando un modelo vista-controlador: cada «pantalla» de la aplicación posee una vista (layout) y un controlador (actividad). Para el desarrollo del controlador, Android Studio acepta 2 lenguajes: Java y Kotlin; mientras que el desarrollo de la vista se realiza en XML.

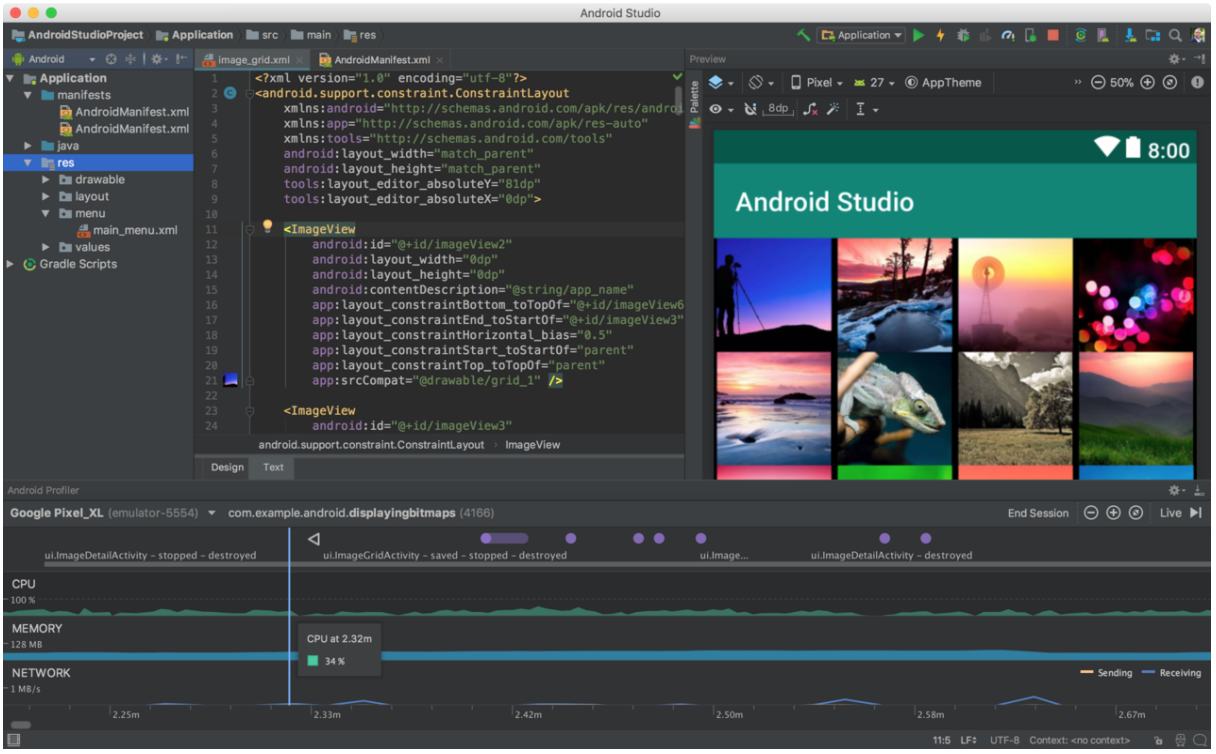


Figura 4. Android Studio.

Android Studio incorpora, además, soporte para el sistema de construcción de aplicaciones Gradle. Éste se encarga de compilar el archivo «.apk» de la aplicación, que luego podrá instalar el sistema Android.

Sin embargo, la mejor baza de este IDE es la gran cantidad de utilidades de las que dispone, todas integradas en el propio entorno. Desde un depurador de aplicaciones, a emuladores de cualquier versión de Android existente en los que poder testear las aplicaciones, incluso se permite la posibilidad de conectar un teléfono al ordenador y poder testear las aplicaciones en el mismo utilizando la herramienta ADB (Android Debug Bridge).

2.2 Desarrollo en Wear OS

2.2.1 Wear OS y su historia

Wear OS (antes conocido como Android Wear) fue anunciado en marzo del 2014, y lanzado al mercado en junio del mismo año. Al estar basado en Android, sigue un sistema de versiones paralelo al mismo, estando su primera versión basada en Android 4.4.

Wear OS es una versión reducida de su hermano mayor, adaptando sus capacidades y su navegabilidad a las características de un smartwatch. Esto hace que posea muchas menos funcionalidades que la versión base, pero sin embargo ayuda a conservar batería y a que el sistema sea mucho más fluído, dadas las especificaciones reducidas de los relojes con respecto a los teléfonos.

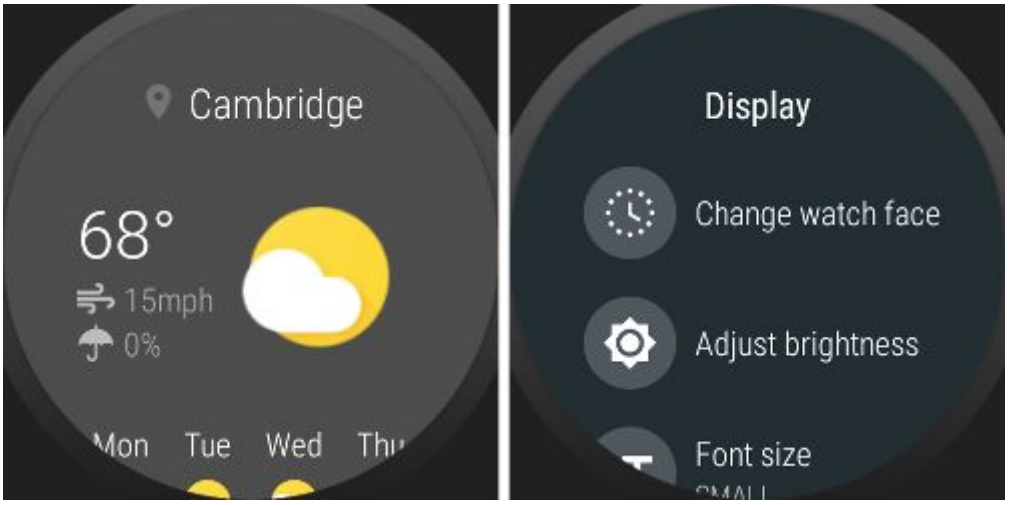


Figura 5. Wear OS.

2.2.2 Diferencias con el desarrollo en Android

El desarrollo para Wear OS es fundamentalmente idéntico al de Android base, ya que comparten gran parte de la arquitectura del Sistema Operativo. Sin embargo, la gran diferencia es que la aplicaciones para Wear deben ser adaptadas para las especificaciones reducidas de sus dispositivos. Aunque técnicamente se podría ejecutar la versión Android de Google Chrome en un dispositivo Wear, el resultado sería prácticamente inutilizable, ya que estos dispositivos poseen procesadores mucho menos potentes, memorias RAM con mucha menos capacidad, y pantallas extremadamente más pequeñas. Una buena aplicación Wear debe ser ligera y sencilla de manejar.

2.3 Pulsera Empatica E4

La Empatica E4 es una pulsera inteligente desarrollada por la empresa Empatica, enfocada a la investigación y monitorización de la epilepsia.

El dispositivo dispone de 4 sensores: acelerómetro, sensor de pulsaciones, sensor de conductividad en la piel (GSR) y sensor de temperatura. Mediante el uso de la tecnología Bluetooth, el dispositivo puede conectarse a un smartphone para enviar los datos de los sensores en tiempo real, los cuales se pueden almacenar posteriormente en la nube o en un ordenador. La pulsera también dispone de un modo independiente, en el cual puede tomar medidas y almacenarlas en el propio dispositivo, para su posterior volcado en un PC.

Sin embargo, lo más interesante de la E4 es la disponibilidad de una API de desarrollo para aplicaciones Android e iOS, que permite incorporar su uso en cualquier aplicación que se desee. Esto expande las posibilidades del dispositivo, y permite su utilización en proyectos de investigación de cualquier tipo, desde proyectos de monitorización de personas con ciertas enfermedades, al posible desarrollo de un polígrafo.

Figura 6. Empatica E4.

2.4 MQTT

2.4.1 MQTT y su historia

Figura 7. Logo de MQTT.org.

MQTT o Message Queuing Telemetry Transport es un estándar ISO basado en un protocolo de suscripción-publicación creado en 1999. Está diseñado para conexiones remotas que requieren una mínima cantidad de identificación a nivel de código. Creada por Andy Stanford-Clark y Arlen Nipper, su primer uso fue para la monitorización de una tubería de aceite a través del desierto. Desde entonces no ha dejado de crecer hasta convertirse en uno de los mayores estándares en la industria a la hora de establecer redes de conexiones entre varios dispositivos, sobre todo en entornos IoT (Internet of Things).

2.4.2 Desarrollo con MQTT

Para la utilización de MQTT se definen dos tipos de entidades distintas: el broker y el cliente. El broker es un servidor que actúa como intermediario entre todos los clientes, recibiendo sus mensajes para posteriormente reenviarlos a través del canal correspondiente. Mientras tanto, el cliente es cualquier programa que implemente MQTT y se conecte al broker.

MQTT implementa un sistema de suscripciones mediante topics o temas, permitiendo que cada dispositivo pueda publicar sus mensajes en un topic determinado, mientras recibe los mensajes de todos los topics a los que esté suscrito. Por ejemplo, una nevera puede enviar su temperatura a través del topic «temperature», mientras que recibe la orden de fabricar hielo a través del topic «ice» y la de activar el modo turbo a través del topic «turbo».

2.5 Machine Learning y análisis de datos

El Machine Learning es el subcampo de la Inteligencia Artificial que estudia los algoritmos que «aprenden» por sí mismos. Los algoritmos de Machine Learning crean un modelo de datos llamado modelo de entrenamiento, a partir del cual pueden realizar predicciones sobre nuevos datos. Dado que utilizan probabilidad en estas predicciones, están fuertemente relacionados con el campo de la estadística.

Los algoritmos de Machine Learning se organizan en diferentes categorías en función de unos criterios concretos. Por ejemplo, si se utilizan datos etiquetados previamente en la fase de entrenamiento (por ejemplo con 2 etiquetas: «estrés» y «no estrés»), es habitual categorizarlos en 2 grupos:

1) Aprendizaje supervisado, en el cual el algoritmo aprende qué salida, a partir de datos etiquetados a priori, debe tener cada entrada, aprendiendo así una relación entre ambas.

2) Aprendizaje no supervisado, en el que el algoritmo descubre (sin utilizar datos etiquetados) resultados por sí solo. Esto puede suponer una meta en sí misma, ya que puede ayudar a descubrir patrones desconocidos en propios datos.

Para la detección del estrés, lo más indicado será utilizar algoritmos de aprendizaje supervisado, ya que se trata de predecir patrones en los datos fisiológicos del usuario, conociendo previamente si esos datos (etiquetados) pertenecen a situaciones de estrés o de relajación. Estos algoritmos se llaman algoritmos de clasificación, o clasificadores. Algunos ejemplos de clasificadores son:

- k-Nearest Neighbors (k-NN), uno de los más utilizados por su simplicidad, clasifica los elementos según la mayoría de elementos cercanos (vecinos) de un tipo que tenga.

- Logistic Regression, estima valores binarios dado un conjunto de variables independientes.

- Naive Bayes, basado en el teorema de Bayes de asunción de independencia entre predictores. Esto significa que se separa la dependencia de cada uno de los datos recibidos.

- Support Vector Machines (SVM), despliega los datos en un hiperplano y utiliza vectores para clasificarlos.

Como se puede apreciar, existe una gran cantidad, y variedad de algoritmos clasificadores, lo que aporta una flexibilidad inmensurable a la hora de aplicar un algoritmo de Machine Learning con aprendizaje supervisado, pudiendo elegir el que mejor convenga en cada situación.

Capítulo 3. Estado del arte en la detección del estrés. Trabajos relacionados.

En este capítulo se presentarán los métodos de detección de estrés en la actualidad y se introducirán algunos trabajos de investigación relacionados con el tema.

3.1 La detección del estrés

El estrés se puede definir como la respuesta del organismo, ya sea biológica o fisiológica, para afrontar una situación que se percibe como amenazante. Dado que el estrés desencadena siempre una reacción fisiológica en el cuerpo, se puede medir dicha respuesta para determinar el estado del individuo.

En el caso de la medición del estrés, el primer paso sería determinar qué respuestas fisiológicas denotan su presencia en el organismo. El estrés está estrechamente relacionado con las emociones negativas tales como la ira, el asco o el miedo, por lo que detectando emociones negativas en el sujeto, estaríamos detectando también a su vez estrés. Existen numerosos estudios al respecto, en la Tabla 2 se aprecian varios ejemplos.

Emotion elicitation method	Emotions elicited	Subjects	Signals measured	Data analysis technique	Results
Imagery	Disgust, anger, pleasure, and joy	50 people (25 males, 25 females)	Self-reports, heart rate, skin conductance, facial EMG	ANOVA	Acceleration of heart rate was greater during disgust, joy, and anger imageries than during pleasant imagery. Disgust could be discriminated from anger using facial EMG
Imagery script development	Neutrality, fear, joy, action, sadness, and anger	27 right-handed males between ages 21–35	Heart rate, skin conductance, finger temperature, blood pressure, electro-oculogram, facial EMG	DFA, ANOVA	99% correct classification was obtained. This indicates that emotion-specific response patterns for fear and anger are accurately differentiable from each other and from the response pattern for neutrality
11 auditory stimuli mixed with some standard and target sounds	Surprise	20 healthy controls (as a control group) and 13 psychotic patients	GSR	Principal component analysis clustered by centroid method	78% for all, 100% for patients

Tabla 2. Ejemplo de estudios sobre la detección de emociones [2].

Se puede observar que existen numerosas formas de medir emociones, tales como el pulso, la conductividad en la piel, los músculos faciales o el movimiento de los ojos. Sin embargo, en numerosos estudios se establece una correlación entre las emociones negativas y las alteraciones fisiológicas en las pulsaciones y en la conductividad en la piel. Será entonces especialmente interesante utilizar ambos parámetros para tratar de medir el nivel de estrés del usuario, a través de sus emociones negativas.

3.2 Stress Recognition using Wearable Sensors and Mobile Phones

Autores: Akane Sano, Rosalind W. Picard [3].

Este estudio realizado en el Massachusetts Institute of Technology (MIT) tenía el objetivo de recabar datos fisiológicos o de comportamiento relacionados con el estrés para luego analizarlos utilizando Machine Learning. Para ello, recolectaron datos de 18 participantes durante 5 días utilizando un wearable con sensores (Affectiva Q-Sensor) y un smartphone, a la vez que varias encuestas realizadas periódicamente.

El proceso que siguió el experimento fue el siguiente:

1) Los participantes visitaron el laboratorio, rellenaron las pre-encuestas y les proporcionaron la pulsera y el software que utilizaría el teléfono.

2) La pulsera tomó datos del acelerómetro y del sensor de conductividad en piel.

3) El teléfono guardó los datos de llamadas, SMS, localización y actividad de la pantalla, además de ofrecer encuestas a los participantes por las mañanas y por las tardes.

4) Los participantes volvieron al laboratorio y completaron una encuesta post-experimento sobre cómo se habían sentido esos últimos 5 días.

El resultado del experimento ofrece un 87,5% de precisión con las encuestas y un 75% con los datos de la pulsera y del teléfono, utilizando un modelo de clasificación binaria de Machine Learning; concluyendo que existía una correlación entre el nivel de estrés y el nivel de actividad y el uso del teléfono.

3.3 A Comparison of Wearable and Stationary Sensors for Stress Detection

Autores: Simon Ollander, Christelle Godin, Aurélie Campagne, Sylvie Charbonnier [4].

En este experimento, se utiliza la Empatica E4 para comparar la precisión de un dispositivo medidor wearable con uno estacionario.

Se seleccionaron 7 participantes, los cuales fueron sometidos a 2 pruebas distintas:

1) La primera prueba consistía en realizar el TSST (Trier Social Stress Test), simulando una entrevista de trabajo enfrente de un jurado durante 5 minutos.

2) La segunda prueba ejercía de control, y consistía en leer un texto en voz alta estando aislado durante 5 minutos. El objetivo de esta prueba era someter al participante a una tarea similar al TSST, pero sin los elementos estresantes.

Los participantes fueron equipados con dos sistemas de medición distintos: la Empatica E4, que medía el volumen de pulso en sangre (BVP), el intervalo entre latidos (IBI) y la conductividad en piel; y el AD Instruments Powerlab, que recogía un electrocardiograma y la conductividad en los dedos.

Una vez obtenidos los resultados, se compararon utilizando diversos factores estadísticos, con la conclusión de que la Empatica E4 funciona tan bien como los dispositivos estacionarios, con algunas puntualizaciones. La Empatica E4 sufre pérdidas ocasionales de datos. En concreto, hubo problemas con pérdidas de datos de intervalos entre latidos, lo cual significa que su utilización en aplicaciones que requieran una medición cardíaca constante y precisa podría no ser la más adecuada. Sin embargo, su medición de la conductividad en la piel no solo se determinó ser igual de precisa que la del dispositivo estacionario, sino que resultó más discriminatoria que la medición de conductividad en dedos proporcionada por el AD Instruments Powerlab.

Esto significa que, a la hora de utilizar la Empatica E4 para la medición del estrés, no solo será tan eficaz como otros dispositivos estacionarios, sino que seguramente los superará.

3.4 Monitoring stress with a wrist device using context

Autores: Martin Gjoreski, Mitja Luštrek, Matjaž Gams, Hristijan Gjoreski [5].

En este proyecto se utiliza la Empatica E4 y Machine Learning para crear un sistema que determine el estrés del usuario de forma contínua e ininterrumpida, en intervalos de 20 minutos.

Para medir el estrés de los participantes se utilizaron 2 métodos distintos:

1) Para la medición del estrés en el laboratorio, se utilizó el Montreal Imaging Stress Task, que consiste en una serie de ecuaciones matemáticas que el sujeto deberá resolver, disponiendo cada vez de un tiempo de respuesta menor.

2) Para la medición del estrés en entornos sin restricciones, se utilizó un sistema de encuestas en el teléfono, que se presentarían al sujeto cada 4-6 horas. Si el resultado de dichas encuestas denotase estrés, se pasaría a grabar las señales fisiológicas del sujeto con la Empatica E4 .

Una vez recopilados los datos, se preprocesaron para el algoritmo de Machine Learning tal y como se muestra en la Figura 8.

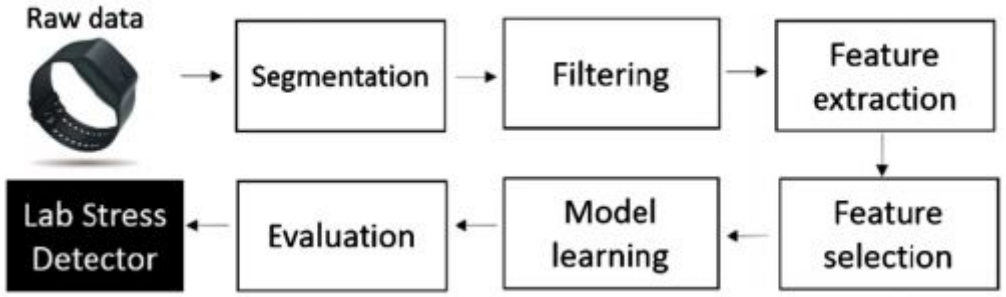


Figura 8. Preprocesamiento de los datos tomados por la Empatica E4 [5].

Una vez analizados los resultados, se observó que los resultados del experimento de laboratorio obtuvieron una precisión del 7%, bastante poco aceptable, mientras que en el experimento en el entorno sin restricciones (fuera del laboratorio), se obtuvo una precisión del 95%, una diferencia enorme; concluyendo que efectivamente, con más investigación, es posible la existencia de un sistema que pueda detectar el estrés en tiempo real utilizando wearables como la Empatica E4.

3.5 Otros proyectos relacionados

Otros ejemplos de proyectos relacionados que se han consultado a la hora de desarrollar StressDetector son:

- A Survey of Affective Computing for Stress Detection: Evaluating technologies in stress detection for better health. Shalom Greene, Himanshu Thapliyal, Allison Caban-Holt.[6]

- Objective measures, sensors and computational techniques for stress recognition and classification: A survey. Nandita Sharma, Tom Gedeon.[7]

- Sensor Placement for Activity Detection Using Wearable Accelerometers. Louis Atallah, Benny Lo, Rachel King, Guang-Zhong Yang.[8]

- A Study About Feature Extraction for Stress Detection. Ionut-Vlad Bornoiu, Ovidiu Grigore.[9]

Capítulo 4. Propuesta y desarrollo del sistema

En este capítulo se explorará la propuesta y desarrollo del sistema StressDetector, presentando su planteamiento a nivel software, para posteriormente tratar cada uno de sus componentes en detalle.

4.1 Descripción general de la propuesta

4.1.1 Introducción

El proyecto consiste en el desarrollo de un sistema distribuido con tecnologías móviles/wearables que detecta si el usuario está estresado o no mediante un algoritmo de Machine Learning. El objetivo de este proyecto consiste en desarrollar el sistema propuesto, para posteriormente estudiar su funcionamiento con un experimento, y finalmente intentar mejorarlo con los datos obtenidos.

La propuesta de sistema es la siguiente:

1) Una parte esencial del sistema se encargará de recibir datos fisiológicos del usuario, y de enviarlos a un servidor para posteriormente recibir una respuesta. También ejercerá la función de interfaz gráfica para todo el sistema, ofreciendo al usuario el estado de los dispositivos del sistema y las respuestas del servidor.

2) Además, otra parte del sistema recibirá y procesará los datos aplicando un algoritmo de Machine Learning, para posteriormente devolver el resultado. Además, dicha parte incorporará también toda la funcionalidad para el procesamiento de los datos recibidos.

Tras llevar a cabo pruebas acerca del funcionamiento correcto del sistema, incluyendo el algoritmo de Machine Learning, se realizará un experimento con varios participantes para validar el sistema, con el objetivo doble de probar el funcionamiento del algoritmo con los datos disponibles en ese momento (aún limitados) y el de obtener datos nuevos para la mejora del mismo.

Finalmente, una vez realizado el experimento y recabados los nuevos datos, se utilizarán varias técnicas de preprocesado sobre los mismos para tratar de mejorar la precisión del algoritmo todo lo posible.

Así se podrá alcanzar el objetivo principal del proyecto: estudiar la viabilidad de un sistema de detección de estrés en tiempo real que aprenda de forma automática con la tecnología disponible actualmente, y crear una base para la posible creación futura de una aplicación apta para su uso en el mercado.

4.1.2 Propuesta de arquitectura del sistema

El sistema distribuido consta de cuatro elementos principales además del usuario: un smartphone Android, un smartwatch Android Wear, una pulsera Empatica E4 y un servidor Python.

Los distintos dispositivos se conectan entre sí tal y como se muestra en la siguiente figura, siendo el smartphone el subsistema principal al cual se conectarán ambos wearables, y que posteriormente enviará los datos al servidor, donde serán procesados devolviendo una respuesta. A continuación, se procederá a explicar el rol individual de cada subsistema y su aportación al sistema global.

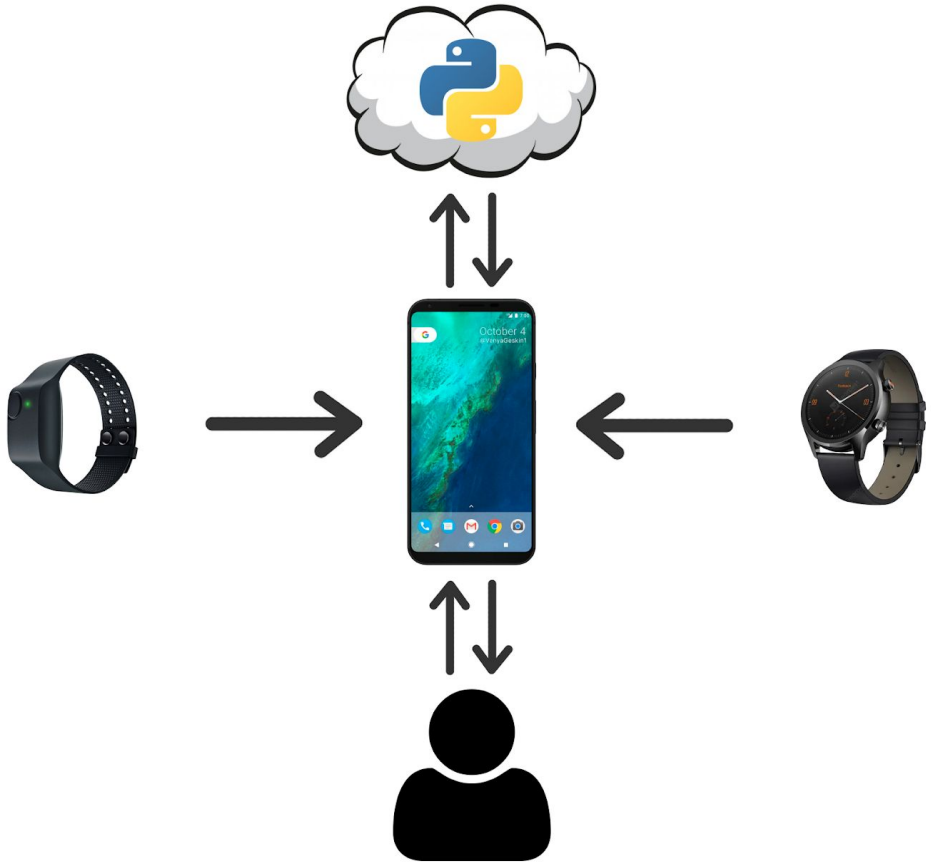


Figura 9. Diagrama de conexiones del sistema

4.1.2.1 Smartphone

El smartphone ejecutará el subsistema principal de StressDetector, al cual se conectarán el resto de subsistemas locales para enviar las lecturas de sus sensores. Este subsistema también recogerá datos de los propios sensores del teléfono, incluyendo: giroscopio, acelerómetro, magnetómetro, sensor de luz y sensor de proximidad. Dichos datos se procesarán en el servidor, para que posteriormente se pueda ofrecer como resultado el nivel de estrés del usuario.

4.1.2.2 Smartwatch

El smartwatch ejecutará el subsistema encargado de obtener los datos de los sensores del reloj y los enviará a la aplicación móvil utilizando la conexión Bluetooth. Los sensores utilizados por el reloj serán: giroscopio, acelerómetro, magnetómetro, sensor de luz y sensor de pulso cardíaco.

4.1.2.3 Empatica E4

La pulsera Empatica E4 se conectará directamente con el subsistema del móvil mediante Bluetooth y enviará las lecturas de sus sensores extremadamente precisos utilizando la API propia de Empatica. Sus sensores serán: acelerómetro, pulso del volumen sanguíneo, actividad electrodérmica, pulsaciones y temperatura.

4.1.2.4 El servidor

El servidor recibirá los datos de los otros 3 dispositivos a través del smartphone, y los procesará utilizando un algoritmo de Machine Learning, devolviendo al teléfono si el usuario está estresado o no.

4.1.2.5 El usuario

El usuario solo necesitará portar el smartwatch y la Empatica E4 en la muñeca, y podrá interactuar con el sistema utilizando el smartphone, el cual proporcionará el estado del resto de dispositivos en todo momento junto con el resultado del análisis del servidor.

4.1.2.6 Comportamiento del sistema

El comportamiento que se requiere del sistema descrito anteriormente de forma general se muestra detalladamente en las dos siguientes figuras, utilizando diagramas de colaboración y de secuencia UML. En particular la Figura 10 muestra la colaboración entre subsistemas, y la Figura 11 muestra cómo interactúan los subsistemas durante la ejecución y en relación al tiempo.

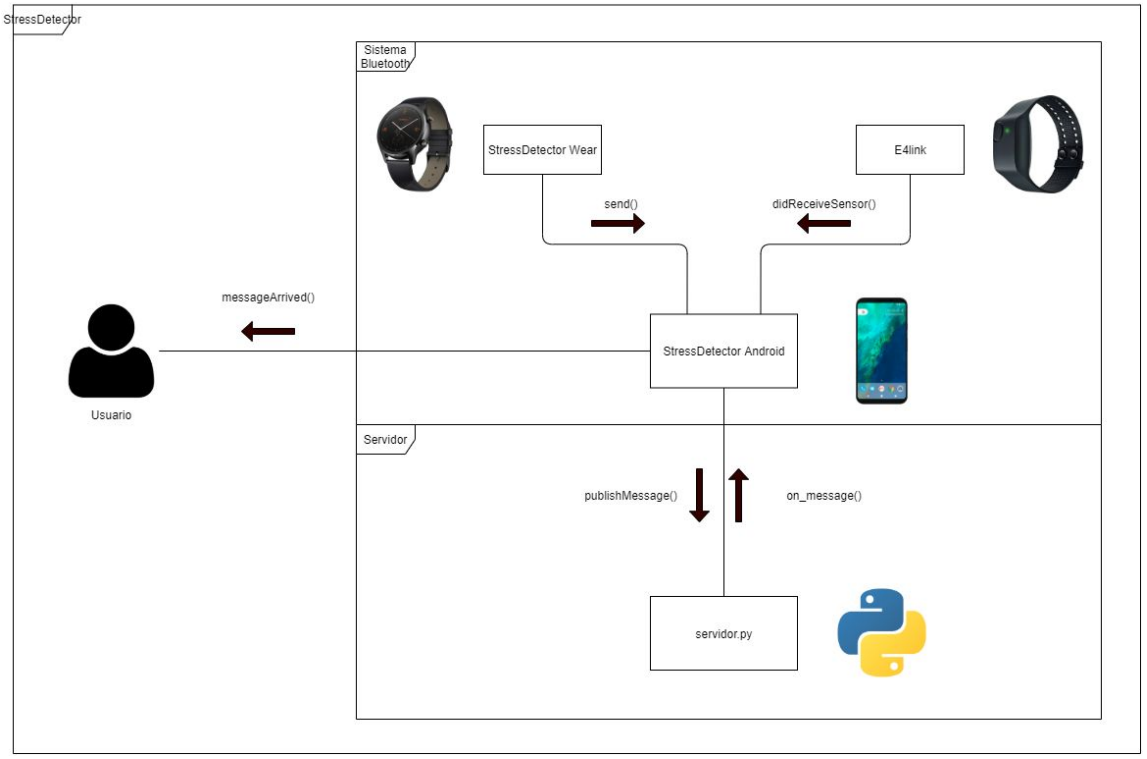


Figura 10. Diagrama de colaboración de subsistemas.

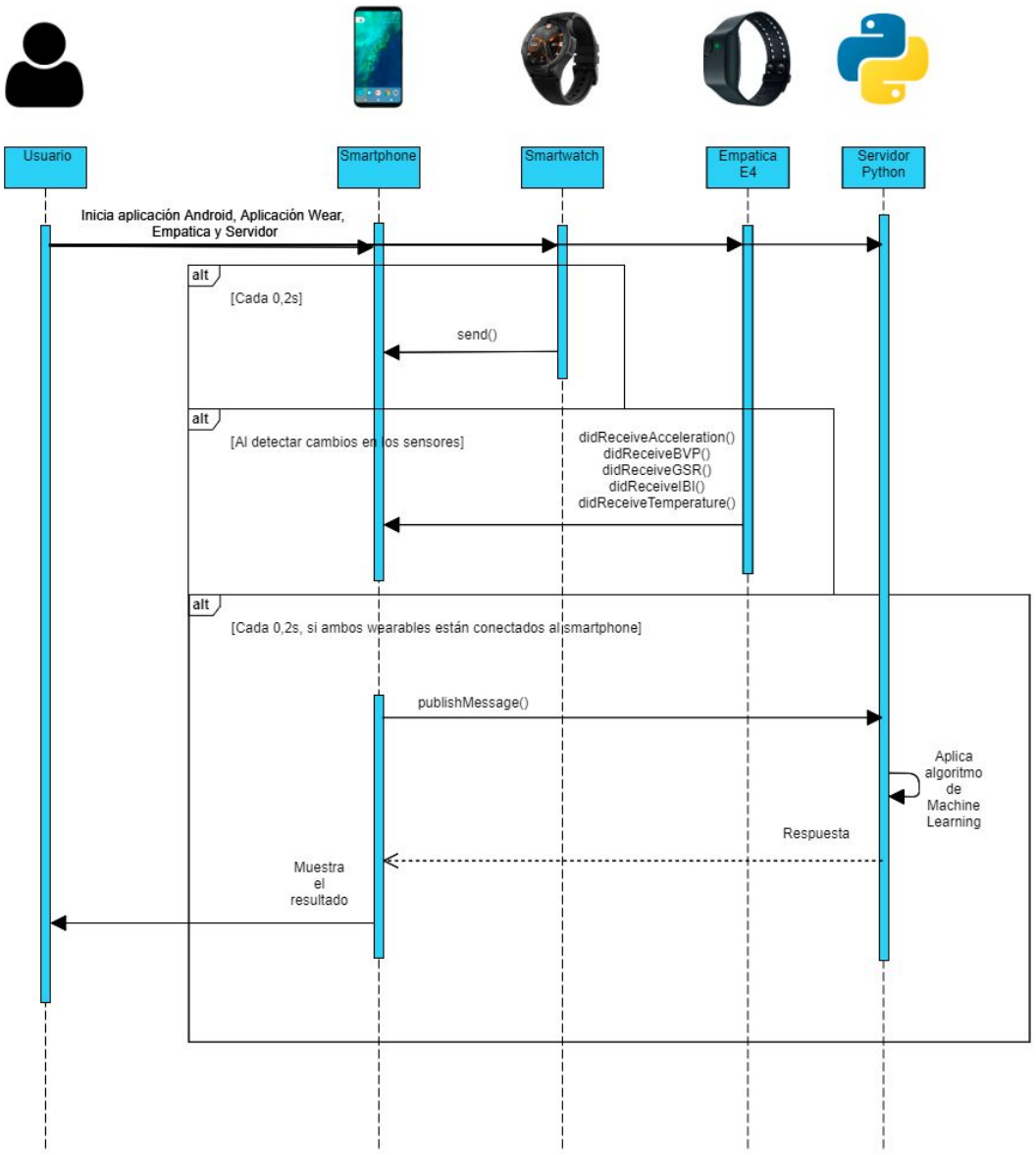


Figura 11. Diagrama de secuencia del sistema.

4.1.3 Requisitos

A continuación se especifican los requisitos del proyecto, dividiéndose en funcionales, no funcionales y de información.

4.1.3.1 Requisitos funcionales

- **RF_1:** El sistema debe poder ser utilizado de forma cotidiana sin que afecte al usuario.
- **RF_2:** El usuario debe poder ver qué dispositivos están conectados.
- **RF_3:** El usuario debe poder ver el estado de los sensores de los dispositivos en tiempo real.
- **RF_4:** El usuario debe poder visualizar el resultado de la medición de su estrés de forma sencilla y clara.

4.1.3.2 Requisitos no funcionales

- **RNF_1:** El sistema debe ser escalable.
- **RNF_2:** Se debe cumplir la restricción de tiempo de que la aplicación Wear debe enviar datos a la aplicación Android con una frecuencia determinada.
- **RNF_3:** La aplicación Android debe enviar datos al servidor Python con una frecuencia determinada.
- **RNF_4:** La aplicación Wear debe funcionar de forma fluida, salvando los problemas que se puedan derivar de las limitaciones de memoria en el reloj.
- **RNF_5:** La aplicación Wear debe ser eficiente, evitando el envío doble de mensajes.
- **RNF_6:** Las conexiones para la comunicación entre los componentes del sistema (p.ej. con el middleware MQTT) tienen que ser fiables, es decir, deben ser estables y sin fallos.
- **RNF_7:** El sistema debe ser sencillo de utilizar
- **RNF_8:** La aplicación Android debe distinguir correctamente de qué dispositivo provienen los datos.
- **RNF_9:** El sistema debe poder ser utilizado con otros dispositivos distintos.
- **RNF_10:** El servidor debe enviar las respuestas con rapidez.
- **RNF_11:** El algoritmo de Machine Learning debe devolver resultados precisos.
- **RNF_12:** La aplicación Android debe enviar los datos al servidor sólo cuando el sistema Bluetooth esté correctamente establecido.

4.1.3.3 Requisitos de información

RI_1: La aplicación Android debe mostrar los datos actualizados de los sensores de los 3 dispositivos en todo momento.

RI_2: La aplicación Android debe mostrar los dispositivos conectados en todo momento.

RI_3: La aplicación Android Wear debe mostrar los datos actualizados de sus sensores en todo momento.

RI_4: El servidor debe mostrar los resultados de las predicciones del algoritmo de Machine Learning con cada toma de datos.

4.1.4 Metodología de desarrollo utilizada

A la hora de organizar el desarrollo del proyecto, era esencial establecer qué metodología de desarrollo se adoptaría. Dadas las características particulares del proyecto, siendo un sistema distribuido para el cual había que desarrollar cada uno de sus componentes por separado para luego comprobar que funcionaban correctamente entre sí, se decidió seguir una metodología de desarrollo mixta.

De cara al sistema como conjunto, se siguió una metodología básica de cascada: primero se desarrolló la aplicación Android, luego la aplicación Wear OS, después el servidor, a continuación el algoritmo de Machine Learning, posteriormente se realizó el experimento y finalmente se hizo el preprocesamiento de los datos.

Sin embargo, la aplicación Android siguió una metodología de desarrollo en espiral, ya que al ser el núcleo del sistema requirió de cambios constantes conforme avanzaba el proyecto: primero se desarrollaron sus funciones básicas, al desarrollarse la aplicación Wear se le introdujeron las conexiones con la misma, posteriormente se le introdujeron las conexiones con la Empatica E4 y se comprobó el buen funcionamiento de todo el sistema Bluetooth, y finalmente al desarrollarse el servidor se le introdujo la funcionalidad de la biblioteca MQTT.

A pesar de ser un planteamiento de desarrollo extremadamente simple, fue muy eficaz a la hora de establecer siempre el próximo objetivo a cumplir, con lo que ayudó mucho a la hora de saber qué hacer siempre en cada momento, aprovechando el tiempo disponible lo máximo posible.

4.2 Aplicación Android

La aplicación Android ha sido desarrollada en el IDE Android Studio utilizando Java [10], y es el núcleo de todo el sistema. Su estructura es la que se puede apreciar en el diagrama de clases de la Figura 12.

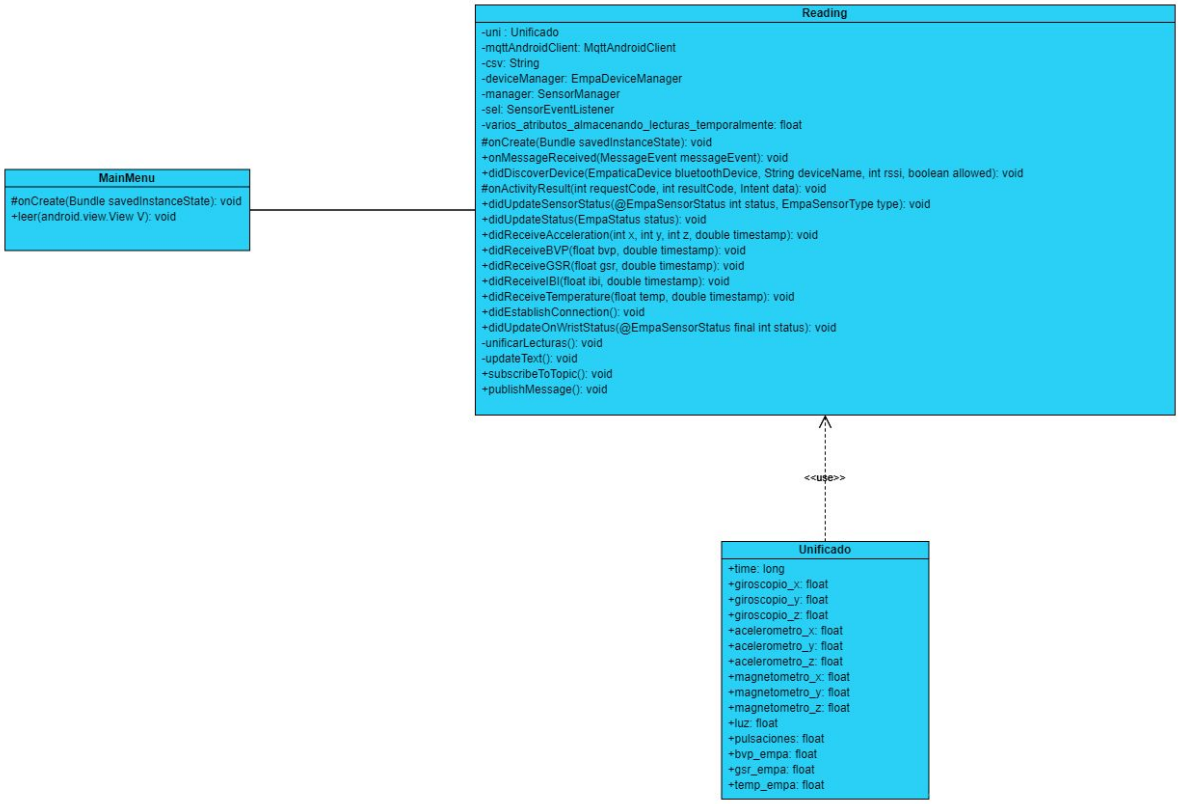


Figura 12. Diagrama de clases de la aplicación Android.

Como se puede observar, la aplicación tendrá una estructura extremadamente sencilla, existiendo solo 2 pantallas con sus clases correspondientes (el menú principal y la pantalla de gestión del sistema), más la clase Unificado, que almacenará los datos unificados de los sensores de los dispositivos locales (smartphone, smartwatch, Empatica E4). Se pasará ahora a explicar su funcionamiento en profundidad.

Una vez conectados el smartwatch y la pulsera Empatica E4 por Bluetooth, la aplicación comienza a recibir datos de los sensores de ambos dispositivos.

La conexión con el smartwatch utiliza la API de paso de mensajes MessageClient, de Google. En un principio se intentó utilizar Google Fit, ya que era mucho más completa, pero estaba más enfocada al almacenamiento de los datos a largo plazo en lugar de enviar mensajes directamente. MessageClient es de una simpleza extrema, aunque tan extrema que a veces daba problemas de condición de carrera. Solamente dispone de un método para enviar y un método listener para recibir. En el caso de la aplicación móvil, solo se utiliza el método de recepción. Una vez se ha recibido el mensaje del reloj, en formato JSON, se almacena temporalmente.

```
@Override
public void onMessageReceived(MessageEvent messageEvent) {

    if (messageEvent.getPath().equals("/sensores")) {
        try {
            JSONObject respuesta = new JSONObject(new String(messageEvent.getData()));
            watch_connected=true;

            time_watch = respuesta.getLong( name: "time");

            giros_watch_x = (float) respuesta.getDouble( name: "giroscopio_x");
            giros_watch_y = (float) respuesta.getDouble( name: "giroscopio_y");
            giros_watch_z = (float) respuesta.getDouble( name: "giroscopio_z");

            acele_watch_x = (float) respuesta.getDouble( name: "acelerometro_x");
            acele_watch_y = (float) respuesta.getDouble( name: "acelerometro_y");
            acele_watch_z = (float) respuesta.getDouble( name: "acelerometro_z");

            magne_watch_x = (float) respuesta.getDouble( name: "magnetico_x");
            magne_watch_y = (float) respuesta.getDouble( name: "magnetico_y");
            magne_watch_z = (float) respuesta.getDouble( name: "magnetico_z");

            luz_watch = (float) respuesta.getDouble( name: "luz");

            pulsa_watch = (float) respuesta.getDouble( name: "pulsaciones");
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```

Figura 13. Función de recepción de mensajes del reloj, onMessageReceived().

La conexión con la Empatica E4 utiliza la API propietaria de Empatica para gestionar conexiones en Android. Ésta requiere que se implementen y sobreescriban todos los métodos de la API, pero por suerte existe un repositorio propio de la empresa con una aplicación de ejemplo [11]. Sin embargo, la integración de la API está muy mal mantenida para las versiones más recientes de Android, lo que dio pie a numerosos problemas que hubo que resolver paso a paso. El sistema dispone de una función para cada tipo de sensor, y en cada una se almacenan dichos datos de forma temporal. Cabe destacar que las medidas del sensor IBI (Inter-Beat Interval) se convierten directamente a pulsaciones por segundo, para poder unificarlas posteriormente con las del smartwatch. Ésto significa que las lecturas del volumen de pulso en sangre (BVP) y del IBI son redundantes, por lo que aunque se reciben ambas, sólo se utiliza el IBI en los cálculos.

```
@Override
public void didReceiveAcceleration(int x, int y, int z, double timestamp) {
    acele_empa_x = (float) (x*9.80665/64);
    acele_empa_y = (float) (y*9.80665/64);
    acele_empa_z = (float) (z*9.80665/64);
}

@Override
public void didReceiveBVP(float bvp, double timestamp) {
    //Blood Volume Pulse
    bvp_empa = bvp;
}

@Override
public void didReceiveGSR(float gsr, double timestamp) {
    //Electrodermal Activity Sensor
    gsr_empa = gsr;
}

@Override
public void didReceiveIBI(float ibi, double timestamp) {
    //Inter-Beat Interval
    ibi_empa = 60/ibi;
}

@Override
public void didReceiveTemperature(float temp, double timestamp) {
    temp_empa = temp;
}
```

Figura 14. Funciones de recepción de datos de los sensores de la Empatica E4.

Igual que con los dos dispositivos wearables, el teléfono también almacena los resultados de sus propios sensores. Para ello, utiliza la API de gestión de sensores de Android, creando un SensorManager. Dicho manager gestiona los datos de todos los sensores del dispositivo. Para intentar que la aplicación funcione correctamente en la mayor cantidad de smartphones del mercado, se seleccionan los sensores a escuchar de forma dinámica, eligiendo cada sensor a partir de la lista de sensores del dispositivo, y según su tipo. Una vez seleccionados, se le asigna un método listener a cada uno, para posteriormente en cada listener almacenar los datos de cada sensor de forma temporal.

```
manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

List giroscopio = manager.getSensorList(Sensor.TYPE_GYROSCOPE);
if(giroscopio.size()>0) {
    manager.registerListener(sel, (Sensor) giroscopio.get(0), SensorManager.SENSOR_DELAY_NORMAL);
}
```

Figura 15. Ejemplo de método listener de SensorManager.

```
public void onSensorChanged(SensorEvent event) {

    time_phone = Calendar.getInstance().getTimeInMillis();

    if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
        float[] values = event.values;
        giros_phone_x = values[0];
        giros_phone_y = values[1];
        giros_phone_z = values[2];
    }
```

Figura 16. Función de recepción de datos de los sensores.

Conciliar la frecuencia de muestreo de todos los dispositivos fue un reto, dada la variedad de tecnologías y bibliotecas utilizadas, por lo que la solución fue utilizar varias hebras: una para la interfaz y otra que se ejecutase cada 0,2 segundos para tomar los datos, como se puede observar en la siguiente figura. Para poder sincronizarse correctamente con la aplicación del smartwatch, ambas aplicaciones empiezan a tomar datos a los 0 milisegundos del siguiente segundo desde su ejecución.

```
class enviar extends TimerTask {
    public void run() {
        runOnUiThread(() -> {
            unificarLecturas();
            updateText();
        });
    }
};

while(Calendar.getInstance().getTimeInMillis()%1000 != 0){
}

Timer timer = new Timer();
timer.schedule(new enviar(),  delay: 0,  period: 200);
```

Figura 17. Función de sincronización de la toma de datos.

Cada vez que se ejecuta la hebra de muestreo se ejecutan dos funciones: unificarLecturas() y updateText(). updateText() simplemente actualiza los datos que muestra la interfaz de la aplicación, mientras que unificarLecturas() unifica los datos leídos por los sensores dependiendo de qué dispositivos estén conectados; de forma que al final resulten unos datos unificados que se almacenan en un JSON con la estructura de la Figura 18. Al terminar de unificar los datos, y si ambos wearables están conectados, se llama a la función de envío de datos al servidor, publishMessage().

```
class Unificado{
    public long time;

    public float giroscopio_x;
    public float giroscopio_y;
    public float giroscopio_z;

    public float acelerometro_x;
    public float acelerometro_y;
    public float acelerometro_z;

    public float magnetometro_x;
    public float magnetometro_y;
    public float magnetometro_z;

    public float luz;
    public float pulsaciones;

    public float bvp_empa;
    public float gsr_empa;
    public float temp_empa;
};
```

Figura 18. Unificación de datos de los dispositivos.

Se profundizará en el uso de la tecnología MQTT en su propio apartado, pero para la conexión con el servidor se ha utilizado la versión del cliente para Android, Paho. La documentación del cliente para Android estaba extremadamente desactualizada, y fue fuente de innumerables quebraderos de cabeza. Cualquier problema con la API devolvía errores internos de Java con manejo de Strings. Para solucionarlo hubo que descubrir, a base de ensayo y error, que existían hasta 3 versiones más recientes que se podían incluir en la compilación (las cuales no se mencionaban en ningún lugar de la documentación, ni de internet en general), y que al menos describían mínimamente los errores cuando ocurrían, permitiendo finalmente solucionarlos. Problemas aparte, la función publishMessage() almacena los datos unificados en un string con formato CSV cada vez que se la llama, y cada 25 muestras se envían al servidor. El motivo de esto es el poder dividir los datos recogidos en ventanas, aparte de aligerar la carga en el broker de MQTT, reduciendo las peticiones de 5 por segundo a 1 cada 5 segundos.

```
public void publishMessage(){

    if(csv_cont==0){
        csv = "pulsaciones, gsr, temperatura";
        csv_cont++;
    }
    else{
        csv += System.lineSeparator() + (double)uni.pulsaciones + "," + (double)uni.gsr_empa + "," + (double)uni.temp_empa + "," + modo;
        csv_cont++;
    }

    if(csv_cont==26){
        csv_cont=0;

        try {
            MqttMessage message = new MqttMessage();

            message.setPayload(csv.getBytes());
            mqttAndroidClient.publish(subscriptionTopic, message);
            csv = "";
        } catch (MqttException e) {
            System.err.println("Error Publishing: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Figura 19. Función de envío de mensajes al servidor, publishMessage().

Finalmente, para recibir los datos del servidor se establece un sencillo listener que recibirá un booleano determinando si el usuario está estresado o no, que pasará a mostrarse en pantalla con un mensaje. Dicho mensaje se irá actualizando conforme se reciban mensajes.

En cuanto a la interfaz de la aplicación, se ha intentado que su estructura sea lo más sencilla y visual posible, dado que su uso será interno. Aparte de la pantalla de título, sólo dispone de una actividad principal, que resume lo que ocurre en el sistema en todo momento. En la parte superior se muestra el resultado recibido del servidor. Posteriormente se muestran los datos ya unificados junto con la marca de tiempo actual y, justo debajo, los datos que devuelve cada dispositivo de forma individual.

Figura 20. Interfaz de la aplicación móvil.

4.3 Aplicación Wear OS

4.3.1 Funcionamiento de la aplicación Wear OS

Dado que enviar los datos a la aplicación Android de forma constante sin disponer de una aplicación dedicada para ello suponía una complicación enorme, se optó por desarrollar una. La versión de StressDetector para Wear OS es extremadamente simple, ejecutando solamente la función de recoger datos de los sensores del reloj para enviarlos al smartphone. Su estructura es la que se muestra en el diagrama de clases de la Figura 21.

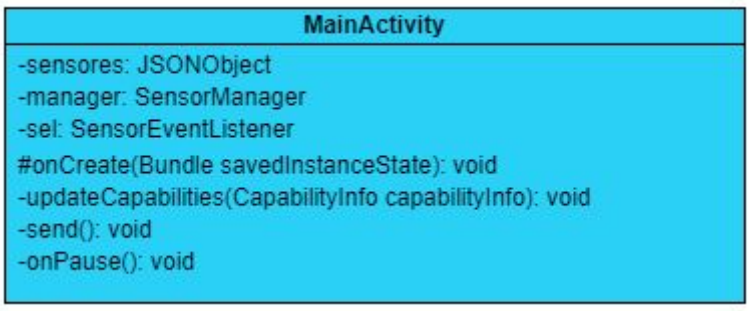


Figura 21. Diagrama de clases de la aplicación Wear OS.

Al igual que la aplicación móvil, la versión Wear utiliza la API de Android para la recogida de datos de sensores, creando un SensorManager para posteriormente recibir cambios en los sensores mediante un SensorEventListener. Dichos datos se almacenan temporalmente en un JSON, para posteriormente enviarse mediante el método send().

```
private void send() {

    if (nodeId != null) {
        byte[] data = sensores.toString().getBytes();
        Task<Integer> sendTask = Wearable.getMessageClient(context).sendMessage(nodeId, MESSAGE_PATH, data);
    }
}
```

Figura 22. Función de envío de mensajes al smartphone, send().

Surgieron bastantes problemas a la hora de lograr un flujo contínuo de datos, ya que la frecuencia de actualización de los sensores era demasiado rápida y se llamaba demasiado a send(), provocando cuellos de botella y problemas de condición de carrera. Alterar el parámetro de frecuencia de refresco de los sensores no dio resultado, ya que el manager de sensores es compartido por todo el sistema operativo, y la frecuencia de refresco siempre será la mayor solicitada por cualquiera de sus componentes. El propio sistema Android utiliza los sensores con frecuencia «SENSOR_DELAY_NORMAL», por lo que cualquier intento de bajar la frecuencia será sobrescrito. Es por esto que se recurrió a una solución similar a la de la aplicación móvil, creando dos hebras: una para la interfaz y otra que enviase los datos cada 0,2 segundos.

```
class enviar extends TimerTask {
    public void run() { send(); }
};


while(Calendar.getInstance().getTimeInMillis()%1000 != 0){
}

Timer timer = new Timer();
timer.schedule(new enviar(),  delay: 0,  period: 200);
```

Figura 23. Sincronización del envío de datos.

Por supuesto, aunque por medio del código se haya intentado sincronizar la lectura de sensores de todos los dispositivos de la red, esta no será nunca completamente exacta, ya que existirán retrasos por parte de la conexión Bluetooth y del propio sistema operativo. Sin embargo, ya que cada dato dispone de su marca temporal propia, y que luego además al enviar los datos al servidor para que sean procesados se recopilan las muestras por conjuntos, estos pequeños retrasos no afectan al sistema, y se puede asegurar que se toman muestras cada 0,2 segundos, o con una frecuencia de 5Hz.

Figura 24. Interfaz de la aplicación Wear OS.

4.4 Servidor y el protocolo MQTT

La conexión entre el smartphone y el servidor Python utiliza el protocolo de conexión MQTT [12]. Dicho protocolo requiere que ambos dispositivos se suscriban a un broker online que gestione los mensajes. Para ello se ha utilizado el broker gratuito HiveMQ [13].

Una vez conectados, el servidor se suscribirá al tema «stress_data» mientras que el smartphone se suscribirá a «stress_results», y comenzarán a enviar mensajes.

El servidor Python sigue un ciclo de vida sencillo. Una vez se ejecuta, se queda en un loop constante esperando mensajes por parte del smartphone, del tema «stress_data». Una vez recibe uno, llama a la función on_message(), que aplica el algoritmo de Machine Learning a los datos, y devuelve el resultado con el tema «stress_results».

```
client.on_message=on_message

client.connect("broker.hivemq.com")

client.subscribe("stress_data")

client.loop_forever()
```

Figura 25. Ciclo de vida del servidor Python.

4.5 Machine Learning y el clasificador k-NN

Una vez establecido el sistema de conexiones con el servidor, es el turno de desarrollar el algoritmo de Machine Learning. Para ello, se empleará la biblioteca scikit-learn, un estándar en Python para el desarrollo con Machine Learning. El proceso que sigue el algoritmo se puede resumir en 3 pasos:

Primero, es necesario cargar el archivo CSV con los datos. Para ello, se almacenan las lecturas recibidas en tandas de 25 a través de MQTT en un fichero CSV, para cargarlo inmediatamente después. Aunque es un comportamiento redundante, la biblioteca falla si no se hace en ese orden. Cabe destacar que inicialmente se utilizó una función propia para cargar el fichero CSV, mientras que después se pasó a utilizar la biblioteca Pandas, que es la más utilizada para análisis de datos en Python. A pesar de que ambas funcionan perfectamente, su forma de cargar los datos es incompatible entre sí. Sin embargo, dicho cambio fue necesario, ya que sería esencial utilizar Pandas más adelante para mejorar el algoritmo.

Una vez cargados los datos, hay que dividirlos en data (X) y target (y). X contendrá todas las filas del archivo CSV excepto la columna objetivo (la etiqueta de estrés/no estrés), mientras que y contendrá dicha columna. En este proyecto, esa columna contendrá 0 y 1, dependiendo de si la muestra denota que el usuario está relajado o estresado, respectivamente.

Ahora es el momento de elegir qué clasificador se utilizará para trabajar con los datos, y en este caso se ha elegido k-NN. El funcionamiento de k-NN es extremadamente simple: para clasificar un elemento, se fija en sus elementos más cercanos, o «vecinos», y determina el resultado (la etiqueta) según el mayor número de vecinos cercanos que tenga con la misma clasificación. El número, «k», será un parámetro a recibir por el clasificador. Por ejemplo, en la figura 26, el nuevo elemento se clasificará con la etiqueta «verde» para un valor de k=2, mientras que se clasificará como «rojo» para un valor de k=3.

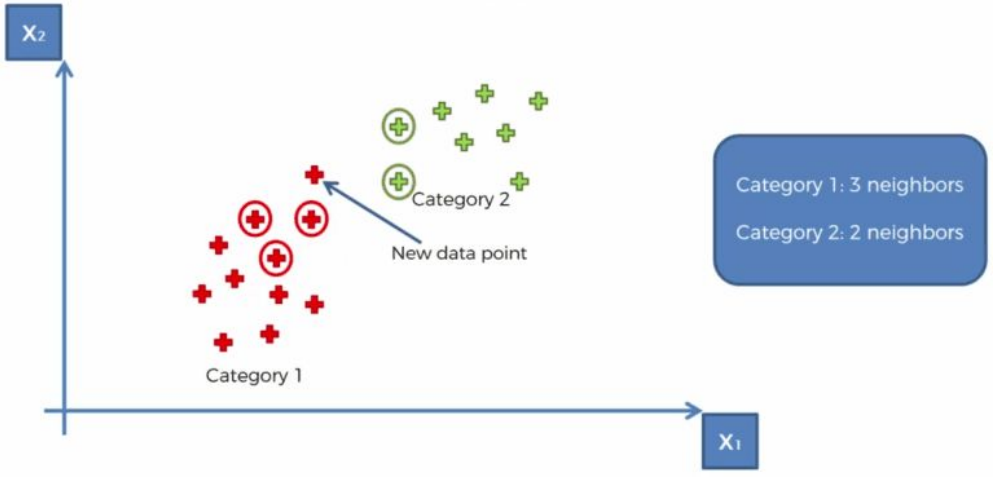


Figura 26. Funcionamiento del clasificador k-NN [18].

El clasificador k-NN ejercerá 2 funciones en este algoritmo: la de entrenar los datos y la de predecir datos nuevos. La primera es esencial: entrenando el algoritmo con nuestros datos ya etiquetados previamente, podremos conseguir que prediga la clasificación de datos nuevos.

Para entrenar los datos con train(), será esencial separarlos primero en datos de entrenamiento (X_train, y_train) y de prueba (X_test, y_test). La idea de esto es que el algoritmo prediga el resultado de un subgrupo de datos ya disponible y aprenda del mismo. Por ejemplo, si entrenamos el clasificador con una separación 20-80, se predecirá el primer 20% de los datos con respecto al 80% restante, luego el siguiente 20% con el otro 80% y así hasta un total de 5 veces.

En este caso, utilizamos además el método conocido como validación cruzada (K-Fold) para separar los datos antes de entrenar el algoritmo. K-Fold aplica la media aritmética a la medida obtenida por cada una de las particiones, mejorando el resultado del entrenamiento.

Una vez terminado el entrenamiento, el clasificador estará listo para predecir datos nuevos. Se llamará a la función predict() del clasificador, pasándole las 25 tomas recibidas por el servidor; y devolverá «y_pred», que será el resultado deseado. Siendo «y_pred» el resultado de predecir las 25 tomas, se elegirá el resultado mayoritario como resultado definitivo, el cual pasará a enviarse a la aplicación Android. En la Figura 27 se muestra el código del algoritmo.

Una vez diseñado el algoritmo, es muy importante utilizar las funciones para medir la precisión disponibles en la biblioteca scikit-learn, para saber cómo de satisfactorio es el funcionamiento del algoritmo y si tiene margen de mejora. En este momento la precisión del algoritmo deja bastante que desear, pero eso mejorará a partir del experimento.

```
"""Datos que se toman como base"""
dataset = load_dataset('csv/base.csv')

X = dataset.data

y = dataset.target


algoritmo = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)

"""KFold"""
kf = KFold(n_splits=5)

for train_index, test_index in kf.split(X):

    X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_index], y[test_index]
    algoritmo.fit(X_train, y_train)

"""Se cargan los datos con las features"""
dataset2 = load_dataset('csv/recibidos.csv')

X_temp = dataset2.data

y_temp = dataset2.target

y_pred = algoritmo.predict(X_temp)

"""Se muestra la predicción para cada muestra y se devuelve la mayoritaria"""
print(y_pred)
c = Counter(y_pred)

value, count = c.most_common()[0]
if value == 0:
    print("RELAJADO")
    client.publish("stress_results", "0")
if value == 1:
    print("ESTRESADO")
    client.publish("stress_results", "1")
```

Figura 27. Algoritmo de Machine Learning, utilizando el clasificador k-NN.

Capítulo 5. Validación del sistema

En este capítulo se procederá a detallar el proceso que se siguió para validar el sistema desarrollado, diseñando y llevando a cabo un experimento con varios sujetos, y tratando luego los datos obtenidos con técnicas de preprocesado para tratar de mejorar la precisión del algoritmo de Machine Learning.

5.1 Objetivo y planteamiento

Para comprobar que el sistema funcionaba correctamente, y para obtener datos para que el algoritmo de Machine Learning pudiera aprender de ellos, se decidió realizar un experimento con varias personas. Dicho experimento consistiría en conectar el sistema al usuario y proceder al visionado de un vídeo preparado especialmente para el mismo. Además, el sujeto tendría que rellenar un formulario antes y después del visionado, para estudiar cómo se sentía en cada momento.

Durante el transcurso del experimento se sometería al sujeto a niveles fluctuantes de estrés, alternando entre fases altamente relajantes y fases altamente estresantes. El objetivo final del experimento era doble:

1) Comprobar cómo funcionaba el algoritmo de Machine Learning a la hora de predecir el estrés con las muestras iniciales de las que se disponía (tomadas conmigo mismo).

2) Almacenar las lecturas recibidas por parte de los sujetos, con el objetivo de utilizarlas posteriormente para refinar las predicciones del algoritmo de Machine Learning.

5.2 Instrumentos

Figura 28. Captura del vídeo del experimento.

Enlace al vídeo: https://photos.app.goo.gl/Qcik7FysQMwejdWy8

El vídeo constaba de 3 partes: una parte inicial relajante, una parte intermedia estresante y una parte final relajante. Con ello se pretendía medir fluctuaciones rápidas en los niveles de estrés del usuario.

En las partes relajantes se utilizaron audiovisuales de naturaleza [14] por su carácter tranquilizador. En concreto se utilizaron un prado, un lago, un río y un bosque de cerezos. También se introdujo una canción del género lo-fi como acompañamiento para las partes relajantes [15]. El objetivo de la canción de fondo, aparte de resultar relajante, era el de crear un hilo conductor. La parte estresante intermedia interrumpe a la canción de golpe, pero luego ésta continúa reproduciéndose por donde iba cuando empieza la parte relajante final, creando la «idea» de que, en efecto, la parte estresante se ha terminado y el usuario se puede volver a relajar.

En la parte estresante se utilizó un vídeo de gente andando en un metro [16], extremadamente alterado en cuanto a velocidad de reproducción, e incorporando movimientos y giros para agobiar al sujeto. A esto le acompañaba una canción de metal extremadamente intensa [17]. La parte final de la canción consistía en un bucle de 3 segundos bastante «desagradable», por lo que se aprovechó para incorporar al vídeo efectos cada vez más estresantes, incluyendo giros de cámara, el sonido de un martillo neumático y el sonido de una ambulancia.

En una versión del vídeo se planteó el incorporar elementos de «miedo» en la parte estresante. A pesar de que el incorporar sustos y visuales perturbadores es algo que se suele hacer habitualmente en experimentos de psicología, siendo ésta una prueba de carácter más ligero y enfocada al ámbito de la informática, personalmente no me parecía ético. Encontrar el equilibrio con estos elementos es complicado, ya que sin emplear visuales realmente desagradables es realmente difícil asustar al sujeto, y si no asusta de verdad puede resultarle gracioso y sacarle del vídeo. De hecho se llegó a desarrollar una versión del vídeo con un par de «caras» que aparecían de forma errática, pero a los sujetos de prueba más que estresante les resultaba algo patético. Es por esto que finalmente se recurrió a utilizar ruidos fuertes y confusos y movimientos bruscos de cámara.

Por otro lado, se creó un formulario simple para que los usuarios lo rellenaran por partes, antes y después de ver el vídeo. En la primera parte se les pedía edad y género, y se les preguntaba si sufrían alguna patología relevante o tomaban algún tipo de medicación. En la segunda, se les pedía que midieran del 1 al 10 cómo de estresados se sentían antes de ver el vídeo, y luego en cada una de las 3 partes del mismo. El objetivo de esto era entender cómo había “experimentado” el vídeo cada persona, para poder posteriormente compararlo con las lecturas recibidas por el sistema.

5.3 Metodología

Inicialmente se buscaba realizar el experimento con al menos unas 15 o 20 personas, pero dada la excepcionalidad de la situación provocada por el COVID-19, sólo se pudieron realizar pruebas a 8 personas, de las cuales 2 aportaron feedback para el planteamiento y mejora del experimento, y 6 se sometieron a la versión final del mismo. Cabe destacar que el sensor de pulsaciones del modelo de Empatica E4 disponible estaba estropeado, por lo que no se tuvo en cuenta para este experimento y se eliminó del cálculo en la aplicación. Sin embargo, el sensor de pulsaciones del smartwatch hizo la función de sustituto perfectamente.

Se dio a los participantes del experimento una vaga idea sobre la temática del mismo, pero no se les aportó detalles sobre la estructura del vídeo, ya que ello arruinaría el efecto deseado.

El experimento se realizó en una habitación aislada, con poca luz y a temperatura media. Se intentó reducir las distracciones todo lo posible. A cada usuario se le pidió que rellenara la primera parte del formulario, y se le conectaron los 2 wearables. Me quedé con el teléfono para comprobar que todo funcionaba correctamente durante el proceso y, ya que no podía abandonar la habitación por el riesgo a perder la conexión Bluetooth con los wearables, me fui a otra parte de la habitación y pedí a los sujetos que ignoraran mi presencia y se concentraran en el vídeo. Finalmente, tras la visualización, se les pidió que rellenaran el formulario y que contaran informalmente cómo se habían sentido. Se muestran los resultados del formulario en la siguiente figura.

Número	Edad	Sexo	¿Consumes algún tipo de sustancia o medicación de forma regular?	¿Posees algún tipo de patología relevante?	Antes del vídeo	1º parte (relajante)	2º parte (estresante)	3º parte (relajante)
1	58	Hombre	Parapres	Hipertensión	7	3	8	3
2	47	Hombre	no	no	1	1	8	3
3	22	Hombre	no	no	4	3	5	4
4	23	Mujer	no	taquicardias	3	7	3	6
5	22	Hombre	no	no	1	8	6	1
6	59	Hombre	no	no	3	1	5	1

Tabla 3. Datos recibidos del formulario.

Por lo que se pudo extraer, hubo 3 reacciones distintas al vídeo:

1) Los sujetos 1 y 2 realmente experimentaron el vídeo como se esperaba: se relajaron mucho en la primera parte, se estresaron mucho en la segunda parte y se volvieron a relajar en la tercera. El sujeto 1 se sentía más estresado al comienzo del experimento, lo que se refleja en que le costó más relajarse en las partes relajantes, mientras que el sujeto 2 tuvo una respuesta de libro.

2) Los sujetos 3 y 6 apenas se sintieron afectados por la parte estresante del vídeo. A pesar de que existe un contraste entre las partes relajantes y estresantes, fue mucho menos pronunciado que con el resto. El sujeto 3 me remarcó que «le había gustado la música» de la parte estresante mientras que el 6, aunque constató que la parte estresante sí que le había estresado comparativamente, se mantuvo bastante relajado durante todo el proceso.

3) Los sujetos 4 y 5 experimentaron el vídeo de una forma completamente inesperada: durante la primera parte relajante se esperaban un susto en cualquier momento, por lo que no bajaron la guardia y se mantuvieron estresados; mientras que en la segunda parte vieron que el video no tenía ese estilo y su estrés bajó comparativamente, y ya en la parte final vieron que definitivamente no iba a pasar y su estrés bajó en picado.

5.4 Resultados

Las siguientes gráficas representan los resultados obtenidos en el experimento:

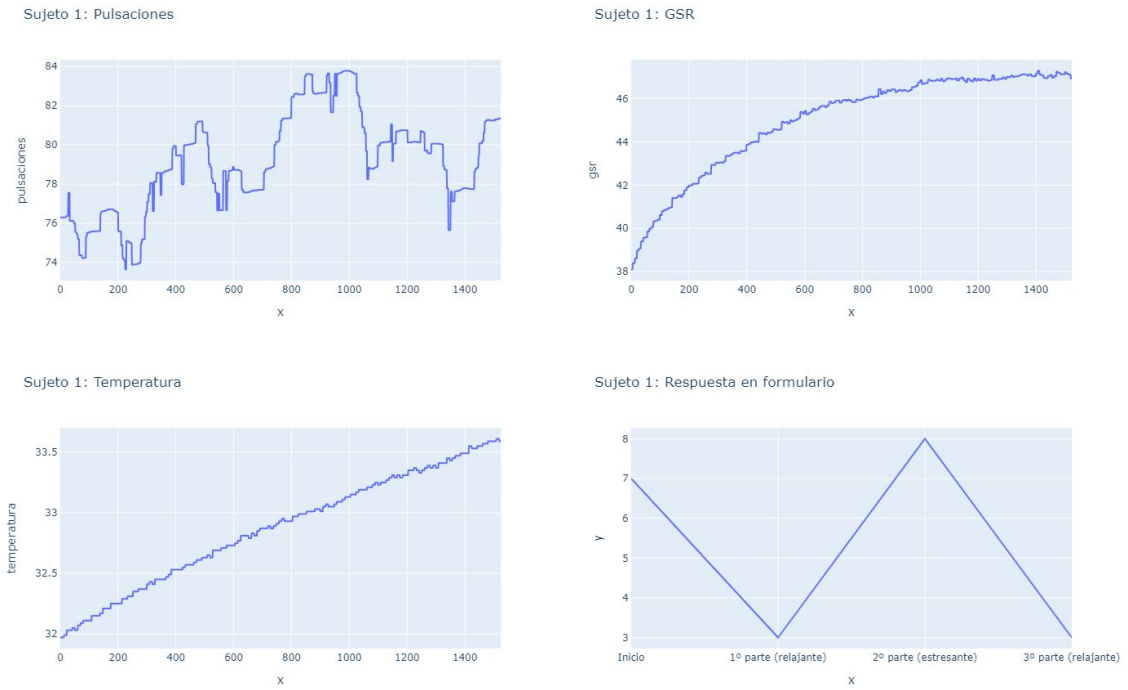


Figura 29. Gráficas del sujeto 1.

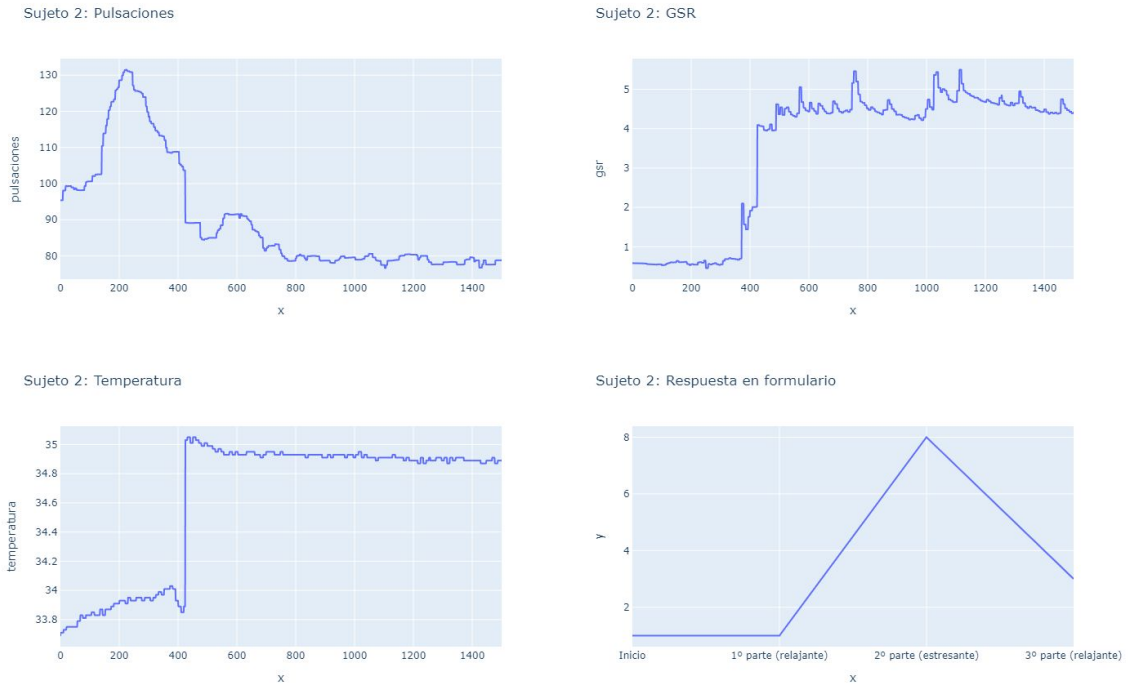


Figura 30. Gráficas del sujeto 2.

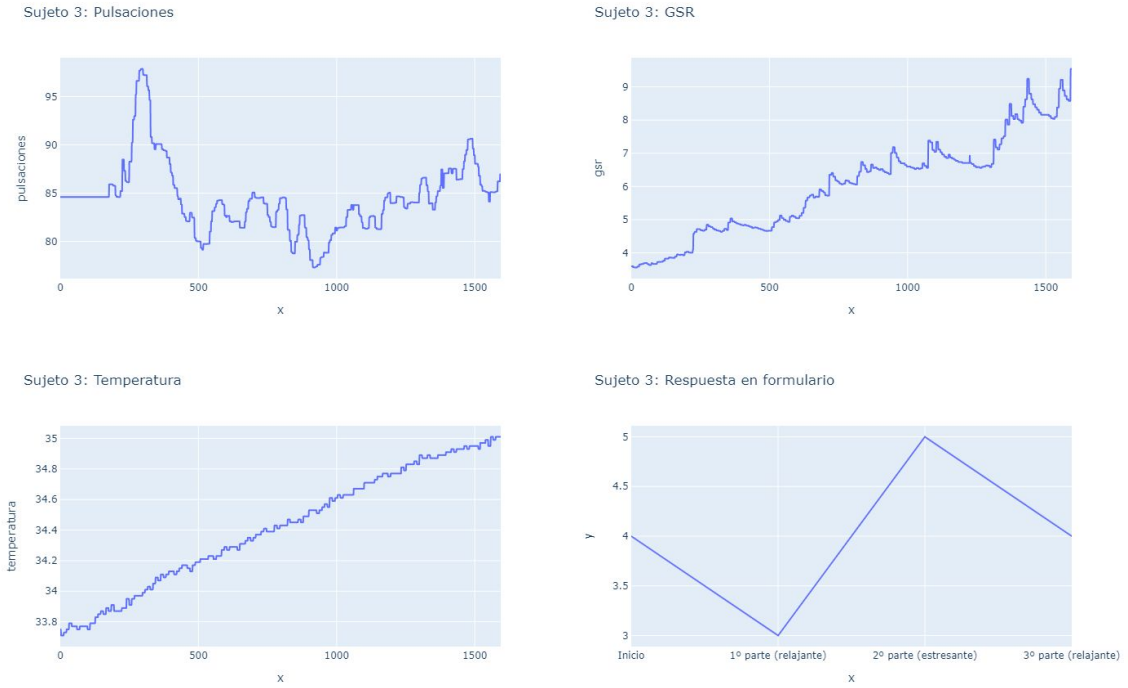


Figura 31. Gráficas del sujeto 3.

Figura 32. Gráficas del sujeto 4.

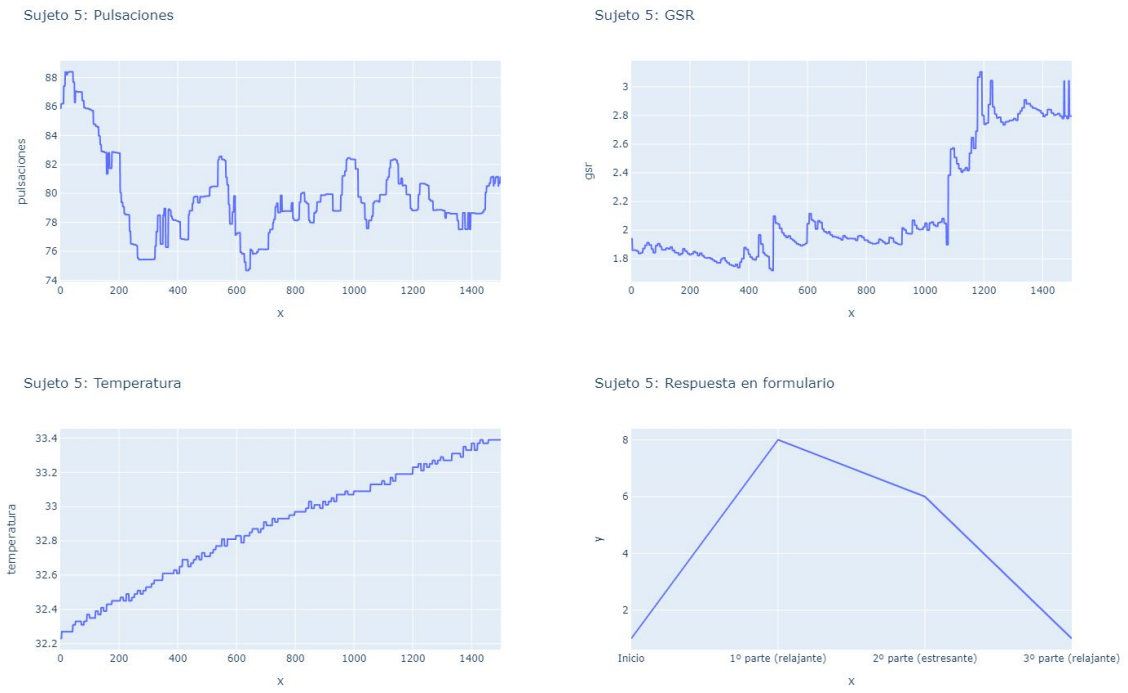


Figura 33. Gráficas del sujeto 5.

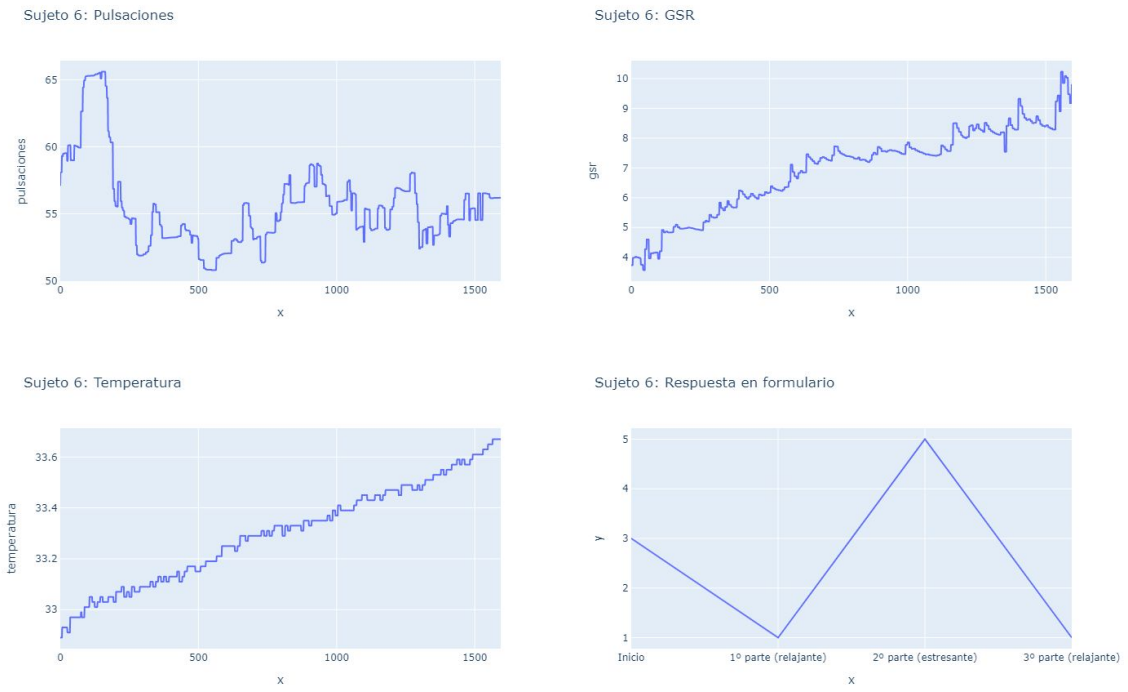


Figura 34. Gráficas del sujeto 6.

Como se puede observar, todos los sujetos siguen unas pautas similares, aunque no sean exactamente las que se deseaban. La conductividad en piel y la temperatura siempre ascienden durante todo el proceso sin importar la fase del vídeo, mientras que las pulsaciones bajan notablemente al poco de comenzar, y luego se mantienen medianamente estables. Ésto indica que seguramente el vídeo era efectivo a la hora de relajar al usuario, pero no tanto a la hora de estresarlo. Anotar que las gráficas del sujeto 4 son extrañamente erráticas, por lo que seguramente la Empatica E4 estuviera fallando a la hora de enviar los datos. Es bastante extraño, ya que es algo que no había pasado nunca, por lo que seguramente no fuese un error de conexión y el sujeto la moviera en la muñeca, provocando el fallo de los sensores.

En cuanto a la prueba con los datos que me tomé de base, no funcionó para nada bien. Básicamente, siempre devolvía que el usuario estaba relajado. Los datos base no estaban preprocesados, y se notaba muchísimo, lo que motivó a trabajar con los nuevos datos recogidos para conseguir que el algoritmo tuviera una fiabilidad mínimamente aceptable, a pesar de que los datos del experimento no fueran especialmente concluyentes.

En conclusión, a pesar de que el experimento no fue todo lo concluyente que habría gustado, se cree que ha sido una gran experiencia de aprendizaje sobre cómo plantear un experimento con el objetivo de recabar datos.

5.5 Proceso de refinamiento del algoritmo. El preprocesamiento de datos.

El proceso de desarrollo del algoritmo fue extremadamente duro. A pesar de que el resultado final es relativamente simple, siendo un completo novato me costó mucho comprender cómo aplicar la teoría del Machine Learning al dominio del problema. Es por ello que el algoritmo comenzó con un funcionamiento realmente pésimo, y pasó por numerosas iteraciones para tratar de mejorar su precisión.

El problema principal aquí fue el nulo preprocesamiento que se hacía sobre los datos recibidos. Tal cual se recibían los datos (giroscopio_x, acelerómetro_y, luz), se metían en la función de entrenamiento, y esto volvía completamente loco al clasificador. Es en este momento cuando los tutores me instaron a realizar una feature extraction sobre los datos, para hacer luego una feature selection.

Una feature extraction consiste en extraer datos nuevos a partir de los datos en bruto que capturan los sensores (tales como la media, la mediana, desviación estándar, kurtosis, etc.), que puedan resultar más concluyentes a la hora de tratar con ellos. Además, a partir de la feature extraction se haría una feature selection, analizando cuáles de esos datos son más relevantes a la hora de predecir el estrés y cuales son completamente inútiles o redundantes.

Fue cuando leí la descripción de la feature selection que me di cuenta de algo: la gran mayoría de los datos que estaba recibiendo eran basura. Siempre había pensado que utilizar, por ejemplo, el magnetómetro del smartphone, era un poco inútil, pero hasta ese momento no descubrí que seguramente estaría dañando activamente los resultados. Decidí realizar una feature selection directamente sobre los datos en bruto de los sensores que recibía para ver si de verdad afectaba tanto a los resultados. La gráfica resultante, en la figura 35, es cuanto menos dolorosa.

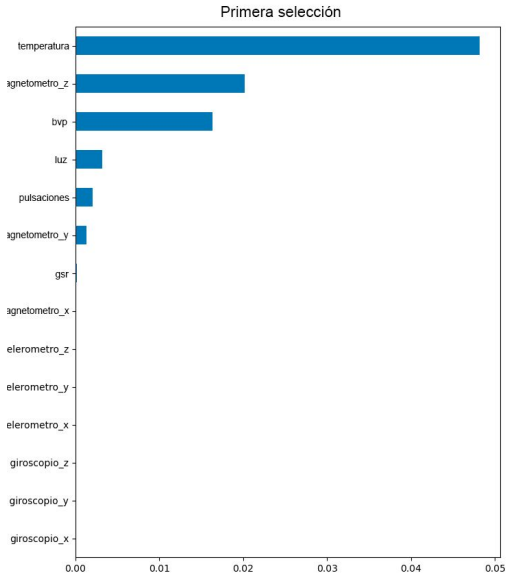


Figura 35. Feature selection con todos los sensores.

Viendo el sinsentido generado en esta tabla, que no solamente pondera menos de la mitad de los datos, sino que da valor a datos completamente aleatorios, decidí cortar directamente la cantidad de datos de sensores recibidos. Reducí la lista a: GSR, pulsaciones, temperatura y acelerómetro. Después de esta purga, se generó la gráfica de la figura 36, que ya parecía ser bastante más lógica. Anotar que fue aquí cuando me di cuenta de que el BVP recibido de la Empatica E4 es redundante con el sensor de pulsaciones, son los mismos datos en otra unidad. Como el sensor de pulsaciones de la Empatica E4 estaba defectuoso, se eliminó también de la lista.

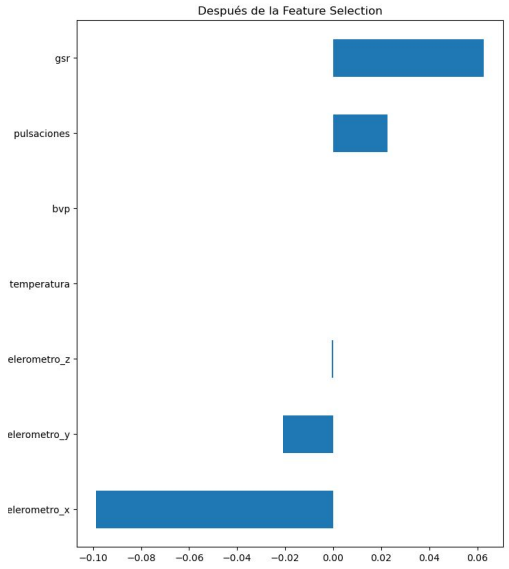


Figura 36. Feature selection con todos los sensores.

A partir de esta gráfica, y observando que el acelerómetro también afectaba negativamente a los datos, surgió la lista definitiva de sensores utilizados: GSR, pulsaciones y temperatura. Ésta sería la lista que pasaría a utilizarse en el experimento.

Una vez realizado el experimento, se decidió intentar utilizar los datos recibidos para mejorar los resultados del clasificador todo lo posible. Tras varias iteraciones, se llegó a una versión con una precisión del 78,71%, resultado más que aceptable teniendo en cuenta que los datos utilizados no eran los mejores.

El primer paso realizado en la nueva versión es separar los datos del experimento en «relajado» y «estresado» (etiquetar los datos). Una vez hecho eso, se almacenan en tandas de 100 tomas, las cuales se dividen en ventanas mediante la técnica de sliding windows, que las «trocea» con un 50% de overlapping. A dicho resultado se le realiza una feature extraction, recibiendo media, mediana, desviación estándar, mínimo, máximo, máximo-mínimo, asimetría (skewness) y kurtosis. Una vez recibidos todos esos datos, y para promediar los datos del *feature extraction*, se realiza la media entre todas las ventanas y se almacena el resultado en el archivo final CSV.

Una vez realizado este proceso con todos los datos, divididos en «relajado» y «estresado», de cada uno de los sujetos del experimento, se puede comprobar la precisión del modelo utilizando el estimador GridSearchCV de scikit-learn.

A pesar de la gran cantidad de cambios que hubo que realizar para llegar a este punto, una precisión del 78,71% es una mejora gigantesca con respecto a cuando se creó el algoritmo, que ni siquiera lograba calcular la precisión correctamente. Lo que esto nos muestra es la extrema importancia que tiene el tratamiento de datos (preprocesamiento) a la hora de trabajar con ellos, ya que sin apenas modificar el algoritmo esto puede suponer la diferencia entre un algoritmo perfectamente fiable y una «escopeta de feria clasificadora».

Capítulo 6. Conclusiones y trabajos futuros

En este capítulo se aportarán las conclusiones sobre la realización del proyecto, junto con los posibles trabajos que se podrán abordar en el futuro, y el punto de vista personal del autor sobre cómo fue su desarrollo.

A continuación se enumeran las conclusiones del proyecto, las cuales están relacionadas y alineadas con el cumplimiento de los objetivos que se fijaron:

- El proyecto ha servido al estudiante para adquirir destrezas y habilidades que han servido para su formación académica y personal, e inicio de su formación profesional. Abordar solo (aunque con la guía de los tutores) una propuesta de proyecto avanzada realista, fuera de la protección de las aulas, y el tener que averiguar cómo sería óptimo resolverla, con qué herramientas y de qué forma, ha sido extremadamente satisfactorio y enriquecedor.

- Se ha adquirido conocimiento sobre campos de la informática siempre se habían querido explorar, como sistemas wearables; e incluso algunos que solo se conocían de nombre, como los algoritmos de clasificación de Machine Learning. Incluso se han adquirido conocimientos tangenciales sobre cómo organizar un experimento, que normalmente pertenecerían a campos como la psicología o la medicina.

- Se ha llevado a cabo el desarrollo de un prototipo de sistema distribuido que es totalmente funcional y operativo.

- La aplicación Android fue un reto de desarrollo, al requerir el uso de diferentes tecnologías de comunicación para su conexión con otros 3 dispositivos simultáneamente. Sin embargo, se ha conseguido una implementación que funciona con una gran fluidez, debido mayormente a la simpleza de su diseño externo. Su interfaz gráfica, aunque simple y no muy llamativa, es extremadamente útil, y proporciona toda la información del sistema a un golpe de vista.

- La aplicación para Wear OS y el servidor son igualmente ligeros, y realizan su trabajo con la fluidez y la fiabilidad que se les requiere.

- El algoritmo de Machine Learning, tras haber implementado sucesivas versiones, ha resultado tener una precisión de 78,7%, la cual se puede considerar bastante aceptable teniendo en cuenta las limitaciones del experimento.

Ha sido muy interesante el tener que organizar solo el desarrollo de un proyecto a lo largo de todo un cuatrimestre. Cuando se eliminan las presiones de las entregas temporales, puede llegar a ser tentador el dejarlo todo para el final. Creo que en este aspecto también se ha realizado un buen trabajo: el proyecto se ha ido realizando periódicamente durante todo el cuatrimestre, y a pesar de que ha habido que desarrollar gran parte del proyecto en el último mes, creo que esto es algo natural dada la carga que hay durante el curso lectivo.

Todo esto no significa que no se hayan encontrado contratiempos durante el desarrollo de este proyecto, al contrario: ha habido bibliotecas mal documentadas, falta de base en aspectos clave, dispositivos defectuosos, e incluso una pandemia global que nos ha dejado a todos sin escuela (o peor, sin biblioteca). Sin embargo creo que se ha sabido afrontar y encontrarle solución a cada uno de los problemas que han ido surgiendo, y el resultado obtenido, bajo mi punto de vista, ha sido más que satisfactorio.

De cara al futuro, creo que se puede profundizar y avanzar aún más en línea con el trabajo realizado, ya sea por mi parte o por la de cualquier persona interesada. A pesar de que el proyecto ha terminado, todavía posee mucho margen de mejora. Se le podrían añadir más dispositivos para obtener aún más medidas, o incluso quitar alguno. Después de la limpieza de los datos, y con una Empatica E4 con el sensor de pulsaciones funcionando correctamente, el reloj resultaría redundante. Por otro lado, el algoritmo todavía tiene muchísimas posibilidades sin aprovechar, una persona avezada en el tema podría hacer maravillas con él. Incluso una persona fuera del campo de la informática, de psicología por ejemplo, podría realizar un nuevo estudio que fuera más concluyente que el realizado aquí utilizando este sistema. Un detector de estrés que esté disponible de forma cotidiana utilizando dispositivos ya existentes es algo que aún no existe, pero que posee un potencial enorme, y sería increíble que alguien lograra llevarlo a la realidad.

Sin embargo, como valoración personal considero que la versión actual de StressDetector ha sido un proyecto tremendamente exitoso tanto a nivel funcional como a nivel académico, que me ha hecho crecer aún estando ya en el ocaso de mi estancia en el grado de Ingeniería Informática.

Capítulo 7. Bibliografía

En este capítulo se aportarán las referencias utilizadas durante el desarrollo del proyecto.

7.1 Documentos

[1] Niveles de estrés según el INE
https://www.ine.es/jaxi/Datos.htm?path=/t15/p419/a2006/p02/&file=02018.px#!tabs-tabla

[2] Using Noninvasive Wearable Computers to Recognize Human Emotions from Physiological Signals. Christine Lætitia Lisetti, Fatma Nasoz.
https://link.springer.com/article/10.1155/S1110865704406192

[3] Stress Recognition Using Wearable Sensors and Mobile Phones. Akane Sano, Rosalind W. Picard.
https://ieeexplore.ieee.org/abstract/document/6681508

[4] A comparison of wearable and stationary sensors for stress detection. Simon Ollander, Christelle Godin, Aurélie Campagne, Sylvie Charbonnier.
https://ieeexplore.ieee.org/abstract/document/7844917

[5] Monitoring stress with a wrist device using context. Martin Gjoreski, Mitja Luštrek, Matjaž Gams, Hristijan Gjoreski.
https://www.sciencedirect.com/science/article/pii/S1532046417301855

[6] A Survey of Affective Computing for Stress Detection: Evaluating technologies in stress detection for better health. Shalom Greene, Himanshu Thapliyal, Allison Caban-Holt.
https://ieeexplore.ieee.org/abstract/document/7574455

[7] Objective measures, sensors and computational techniques for stress recognition and classification: A survey. Nandita Sharma, Tom Gedeon.
https://www.sciencedirect.com/science/article/pii/S0169260712001770

[8] Sensor Placement for Activity Detection Using Wearable Accelerometers. Louis Atallah, Benny Lo, Rachel King, Guang-Zhong Yang.
https://ieeexplore.ieee.org/document/5504808

[9] A Study About Feature Extraction for Stress Detection. Ionut-Vlad Bornoiu, Ovidiu Grigore.
https://ieeexplore.ieee.org/document/6563421/

7.2 Android y servidor

[10] Android documentation.
https://developer.android.com/

[11] E4link 1.0 Sample Project
https://github.com/empatica/e4link-sample-project-android

[12] Beginners Guide To The Paho MQTT Python Client
http://www.steves-internet-guide.com/into-mqtt-python-client/

[13] HiveMQ MQTT Broker
https://www.hivemq.com/public-mqtt-broker/

7.3 Experimento

[14] Visuales relajantes: “Calmed By Nature”
https://www.youtube.com/channel/UCJuMbdKSMThk2RpALASyXVQ

[15] Canción relajante: “lナm SURF - Take Care”
https://www.youtube.com/watch?v=HTNXz6WspCU

[16] Visuales estresantes: “People walking - timelapse”
https://www.youtube.com/watch?v=7PTriE459ko

[17] Canción estresante: “Demoni e Dei (Contropotere) - by TxSxBx”
https://www.youtube.com/watch?v=JqC252t8oOg

7.4 Machine Learning

[18] Funcionamiento del clasificador Knn.
http://brokerstir.com/knn-classifier-algorithm/

[19] Scikit-learn Tutorial: Machine Learning in Python
https://www.dataquest.io/blog/sci-kit-learn-tutorial/

[20] Target Variable
https://www.datarobot.com/wiki/target/

[21] Beginner's Guide to Feature Selection in Python
https://www.datacamp.com/community/tutorials/feature-selection-python

[22] Feature Selection with sklearn and Pandas
https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b

[23] Cross-validation: evaluating estimator performance
https://scikit-learn.org/stable/modules/cross_validation.html#k-fold

[24] Beginner's Guide to Feature Selection in Python
https://www.datacamp.com/community/tutorials/feature-selection-python

ANEXO I: Problemas encontrados y sus soluciones

En la siguiente tabla se procederá a listar algunos de los problemas más relevantes que se tuvieron durante el desarrollo del proyecto, su descripción y la solución que se le aplicó.

El problema	Descripción	Solución encontrada
Coincidencia con la pandemia provocada por el COVID-19	Éste es un problema que seguramente habrá tenido cualquier trabajo desarrollado durante este periodo, pero el confinamiento obligatorio a causa de la pandemia dificultó enormemente el avance del proyecto; desde la tosquedad de la comunicación online con los tutores a la pérdida de sujetos para el experimento.	Dado que era algo inevitable, la solución pasó por trabajar más y mejor.
Falta de potencia del smartwatch	A pesar de que el smartwatch utilizado en el proyecto era nuevo y con unas buenas características, fue bastante complicado trabajar con él, sufriendo ralentizaciones de hasta 30 segundos y parones repentinos al ejecutar aplicaciones.	Se intentó reducir lo máximo posible la carga de la aplicación Wear desarrollada, y se establecieron medidas especiales para evitar problemas si se cerraba de golpe. Por ejemplo, introducirle un botón que matara el proceso, porque si no a veces se quedaba en segundo plano mandando mensajes duplicados y volviendo loco al teléfono.
Problemas con MessageClient, la biblioteca de conexión de Wear	MessageClient posee un funcionamiento extremadamente simple, con un método para enviar y un listener para recibir. Sin embargo no parece estar preparado para el envío	En vez de enviar los datos del reloj cada vez que se actualizaran los sensores, se implementó un sistema para que se sincronizara con el

	continuo de datos, ya que al utilizar un flujo de datos demasiado grande se atascaba y perdía paquetes, o los enviaba de forma desordenada.	teléfono y enviara los datos con la misma frecuencia con la que el teléfono los procesaba, reduciendo así el flujo de datos al mínimo y evitando la sobrecarga de MessageClient.
La Empatica E4 utilizada estaba defectuosa	La Empatica E4 utilizada en el proyecto tenía un sensor de pulsaciones defectuoso cuyas lecturas bailaban constantemente en un rango de +/-20 pulsaciones por minuto, además de fallar su conexión en algunas pruebas.	Se dejó de tener en cuenta el sensor de pulsaciones de la Empatica E4 para los cálculos, ya que por suerte el smartwatch disponía del suyo propio. En cuanto a los fallos durante las pruebas, no se podía hacer otra cosa que estar pendiente durante las mismas por si fallaba.
El repositorio de ejemplo de Empatica está desactualizado	El repositorio que provee Empatica como ejemplo para el desarrollo de aplicaciones Android, a pesar de haberse actualizado hace apenas 5 meses, da problemas al implementarlo en proyectos Android modernos.	Se investigó si otras personas tenían problemas similares y se fue depurando el código proporcionado poco a poco.
La documentación de la biblioteca MQTT para Android está extremadamente desactualizada	La documentación de MQTTAndroid deja muchísimo que desear para ser un estándar que se supone que es bastante utilizado. Posee código completamente obsoleto que hay que corregir paso a paso, pero lo peor es que dispone de versiones más recientes a la que muestran en su página oficial, y en cualquier otra página relacionada en internet. Si no se usan esas versiones más recientes, la biblioteca devuelve errores	En cuanto al código, hubo que corregirlo y adaptarlo poco a poco recabando datos de los problemas que tuvieron otras personas. En cuanto a la existencia de versiones más recientes, la solución fue darme cabezazos contra el muro hasta que se me ocurrió probar a subirle el número a la versión en

	internos de Java totalmente incomprensibles.	Gradle por si funcionaba.
Falta de base de Machine Learning	Como estudiante de la rama de ingeniería del software, y habiendo cursado solo la asignatura de IA, mi conocimiento sobre Machine Learning era completamente nulo al comienzo de este proyecto; lo que sumado a que no pude disponer de toda la asistencia necesaria debido a la pandemia hizo que desarrollar el algoritmo fuese inmensamente desafiante.	Tuve que aprender Machine Learning de forma autodidacta con la ayuda de algunos tutoriales que me enviaron los tutores, pasando de tener conocimiento nulo a la implementación del algoritmo en menos de 2 semanas.
Problemas con el preprocesamiento de los datos	Al igual que con el Machine Learning, mi experiencia con el tratamiento y procesado de datos era casi nula, más allá de algunas lecciones en la asignatura Algorítmica. Esto significaba que, por muy bien desarrollado que estuviera el algoritmo, estaría cojo sin una buena base con la que entrenarse.	Como con el problema anterior, la solución pasó por aprender cómo funcionaba el preprocesamiento de datos de forma express, contando con la ayuda de los tutores a la hora de enfocar el problema.
El experimento no fue tan concluyente como se esperaba	A pesar del amplio trabajo de investigación realizado sobre cómo se podría realizar satisfactoriamente un experimento de este tipo, los datos fisiológicos tomados de los sujetos no fueron tan concluyentes como se esperaba.	Aunque este es un problema que no se puede solucionar, se aprovechó lo mejor posible el resultado obtenido: los datos fisiológicos sirvieron para refinar el algoritmo, y el experimento como experiencia para el futuro.

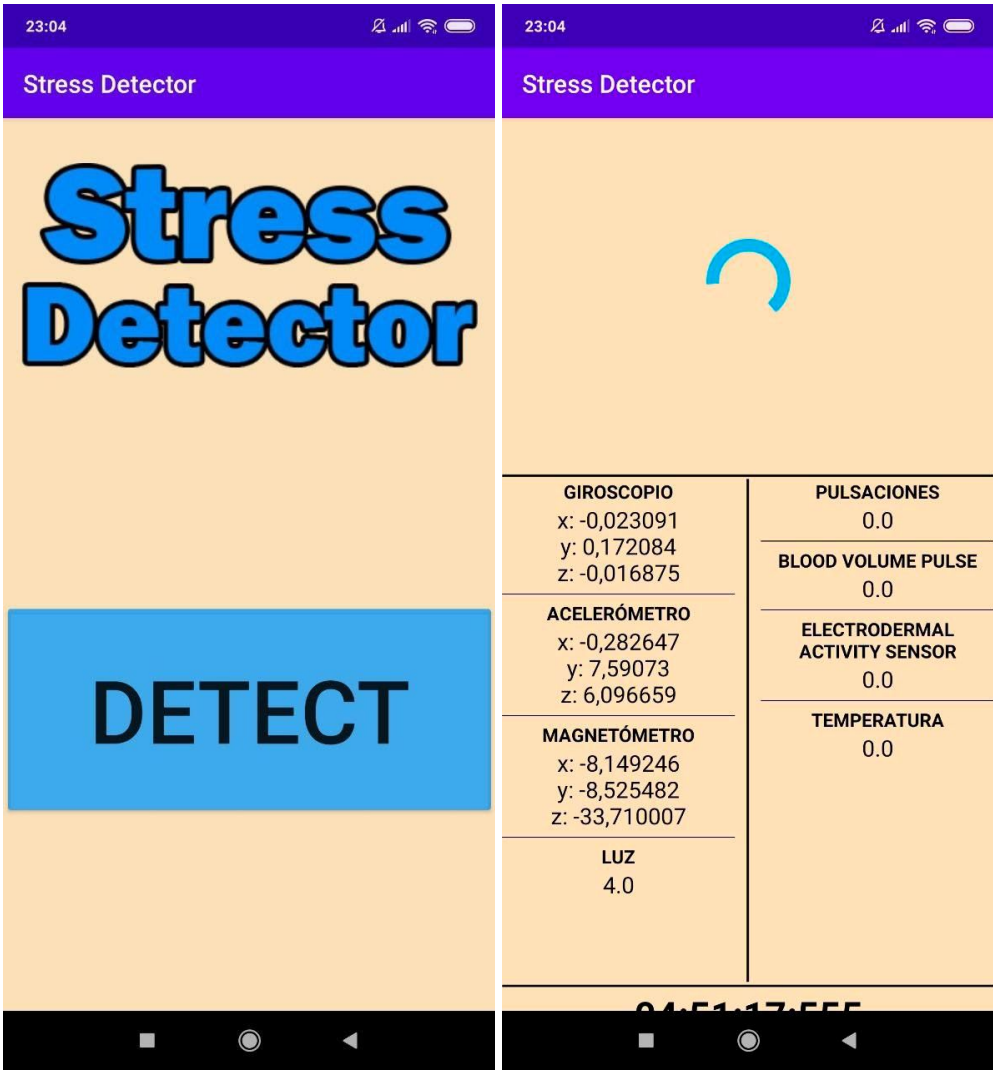
Tabla 4. Problemas que surgieron durante el desarrollo del proyecto.

ANEXO II: Manual de uso del sistema

Se desea que este pequeño manual sirva como guía para desplegar el sistema de StressDetector partiendo de cero.

El primer paso será preparar los 3 dispositivos del sistema.

Una vez instalada la aplicación en el smartphone (con el archivo .apk), simplemente habrá que ejecutarla y darle al botón “Detect” para acceder a la pantalla de conexión. La parte superior muestra el resultado del servidor (por ahora no mostrará nada), la parte intermedia muestra los datos unificados que se mandarán al servidor, y la parte inferior muestra el estado de los sensores de cada dispositivo de forma individual.



Ahora que el smartphone está preparado, habrá que ejecutar la aplicación en el smartwatch (previamente instalada mediante el fichero .apk). La aplicación empezará a enviar datos al smartphone automáticamente, los cuales se reflejarán en la interfaz. Se recomienda comprobar que los datos se envían correctamente.

Anotar que a veces el reloj se ralentiza al abrir la aplicación, lo que provoca que se envíen datos dobles. Si ésto ocurre, se puede solucionar pulsando el botón “EXIT” y volviéndola a abrir.

Una vez conectado el reloj, se podrá conectar la Empatica E4 pulsando su botón durante 2 segundos hasta que su led se vuelva de color azul.

Ahora que ya está listo el sistema Bluetooth, la aplicación móvil comenzará a enviar mensajes al broker MQTT cada 0,2 segundos. El siguiente paso será ejecutar el servidor Python en el PC, ejecutando el fichero servidor.py.

Una vez se muestre el mensaje “Ready”, el servidor estará listo para recibir los mensajes del broker. Cuando reciba los mensajes los procesará y devolverá el resultado al smartphone, que lo mostrará en la parte superior de la interfaz.