Universidad Zaragoza
1542

Trabajo Fin de Grado

Estimación del riesgo de caídas usando acelerómetros y técnicas de Aprendizaje Automático

Autor

Iván Gómez Pascual

Director

Carlos Medrano Sánchez

Escuela Universitaria Politécnica de Teruel

2022

Estimación del riesgo de caídas usando acelerómetros y técnicas de Aprendizaje Automático

Resumen

En este Trabajo Fin de Grado se pretende estimar el riesgo de caídas en ancianos usando acelerómetros para evaluar dicho riesgo gracias a técnicas de Aprendizaje Automático en lenguaje Python.

Para ello se cuenta con una base de datos de personas mayores que llevaron el teléfono en su vida diaria, recogiendo datos del acelerómetro de forma casi continua, así como de un código previo de procesado, realizado en el grupo EduQTech. También se realizaron pruebas controladas caminando. De estos datos en bruto se puede obtener una serie de características reducidas que sirvan como entrada a los clasificadores. Además, se realizaron una serie de pruebas clínicas (prueba Tinetti, test up and go) y entrevistas sobre las caídas que tuvieron recientemente. Estas pruebas permiten determinar si una persona tiene o no riesgo de sufrir una caída. La idea del proyecto es vincular este riesgo clínico con las señales de aceleración utilizando técnicas de Aprendizaje Automático.

Palabras clave: Python, Aprendizaje Automático, Riesgo de caída, *Acelerómetros*.

Fall risk estimation using accelerometers and Machine Learning techniques

Abstract

The aim of this Final Degree Project is to estimate the risk of falls in elderly people using accelerometers to assess this risk thanks to Machine Learning techniques in Python language.

For this purpose, we have a database of elderly people who carried the phone in their daily life, collecting accelerometer data almost continuously, as well as a previous processing code, made in the EduQTech group. Controlled walking tests were also performed. From this raw data, a number of reduced features can be obtained to serve as input to the classifiers. In addition, a series of clinical tests (Tinetti test, up and go test) and interviews about recent falls were performed. These tests make it possible to determine whether or not a person is at risk of falling. The idea of the project is to link this clinical risk with acceleration signals using Machine Learning techniques.

Keywords: Python, Machine Learning, Fall Risk, Accelerometers.

ÍNDICE

Listado ilustraciones

Listado tablas

Listado siglas y abreviaturas

AVAD: Años de Vida Ajustados por Discapacidad.
POMA: Performance-Oriented Mobility Assessment.
TUGT: Timed Up and Go Test.
IA: Inteligencia artificial.
ML: Machine Learning.
DL: Deep Learning.
MPL: Multi-Layer Perceptron.
USENET: Users Network.
API: Application Programming Interfaces.

MEMORIA

1 INTRODUCCIÓN

1.1 Motivaciones del proyecto

Actualmente en el campo de la medicina se necesita una revolución en la que se integren mucho más la última tecnología existente, sobre todo la inteligencia artificial, ya que esto podría suponer una cuestión vital en la vida de los pacientes a la hora de la prevención de anomalías en fases tempranas.

En el caso propuesto, la predicción de caídas en ancianos, sería fundamental su combinación. Al año supone 37,3 millones de lesiones cuya gravedad requiere atención médica (OMS, 2021). Por lo que el *Machine Learning* podría ser una técnica adecuada para pronosticar el riesgo de caída mediante el uso de un acelerómetro llevado por los ancianos. La aceleración es medida directamente con un teléfono móvil, por lo que el sistema de medición es relativamente barato y sin apenas influir en la vida diaria de los ancianos que lo usan. La implementación se llevaría a cabo en código Python para conseguir de la aceleración un sistema predictor de caídas. Al haber diferentes técnicas de *Machine Learning* para obtener este sistema predictor se evaluará cual sería la mejor. Dicha evaluación se realizará con la vinculación de las pruebas clínicas actuales que se utilizan para medir este riesgo (prueba Tinetti, *Test Up and Go*) y entrevistas sobre las caídas que han tenido recientemente.

1.2 Objetivos del proyecto

La idea principal del proyecto es vincular el riesgo de caídas clínico con las señales de aceleración para así determinar si una persona tiene o no riego de sufrir una caída mediante técnicas de Aprendizaje Automático.

Por lo que para este proyecto han sido necesario cumplir con las siguientes metas:

- Familiarizarse con las técnicas de Aprendizaje Automático y el uso de librerías para su implementación en Python.
- Conocer los diferentes entonos de trabajo para programar en Python.
- Aplicación de la base de datos de los ancianos realizada por el grupo EduQTech.
- Estimar las prestaciones de los algoritmos para la predicción del riesgo de caídas en función de varios factores
- Comprobar la diferencia entre las prestaciones obtenidas cuando se usan señales de la vida diaria o señales de pruebas físicas controladas.
- Presentar los resultados de forma compacta y comprensible.

2 FUNDAMENTOS TEÓRICOS

2.1 Caídas en la población mundial

Las caídas suponen un problema real, ya que pueden significar un peligro en la vida de los individuos. Pero antes continuar analizándolas tenemos que saber a lo que se conoce por caída: "Las caídas son sucesos involuntarios que hacen perder el equilibrio y dar con el cuerpo en el suelo o en otra superficie firme que lo detenga" (OMS, 2021).

A continuación, se muestran unos datos claves para entender la magnitud que proporciona la OMS (2021):

- 37,3 millones de caídas revisten suficiente gravedad como para requerir atención médica.
- Anualmente fallecen en todo el mundo unas 684 000 personas debido a caídas.
- Las mayores tasas de mortalidad por esta causa corresponden a los mayores de 60 años.

Las caídas no afectan a todas las personas por igual, según su edad o sexo varía.

- **Sexo:** A priori todos tienen las mismas posibilidades de sufrir caídas, pero las mujeres de avanzada edad están más predispuestas a padecerlas, aunque la las tasas de mortalidad y los AVAD perdidos son más altas en varones. Esto se podría deber a los hombres realizan trabajos de más de riesgo.
- **Edad:** Las personas de una edad avanzada son los que tiene una probabilidad mayor de sufrir lesiones de una mayor gravedad. Para entender mejor los datos se puede observar el ejemplo de EEUU, entorno a un 20/30% de las personas que sufren una caída tienen lesiones de moderadas a graves.

El costo económico según las estimaciones de diferentes países varía mucho, pero el importe del tratamiento de un traumatismo en una persona de 65 años varía según el país. Por ejemplo, 1049$ en EEUU a 3611$ en Finlandia, lo que supondría, sin lugar a dudas, un gran desembolso.

2.2 Base de datos suministrada

Se cuenta con una base de datos de personas mayores que llevaron el teléfono en su vida diaria, recogiendo datos del acelerómetro de forma casi continua, así como de un código previo de procesado, realizado en el grupo EduQTech. De estos datos en bruto se puede obtener una serie de características reducidas que sirvan como entrada a los clasificadores.

Además de esta base de datos se cuenta con otra base de datos similar, pero a diferencia de la primera el acelerómetro se llevaba cuando el sujeto únicamente andaba para así no tener que filtrar el código e introducir menos error (pruebas controladas caminando). Un ejemplo gráfico de la aceleración en la prueba controlada de caminar sería el siguiente:

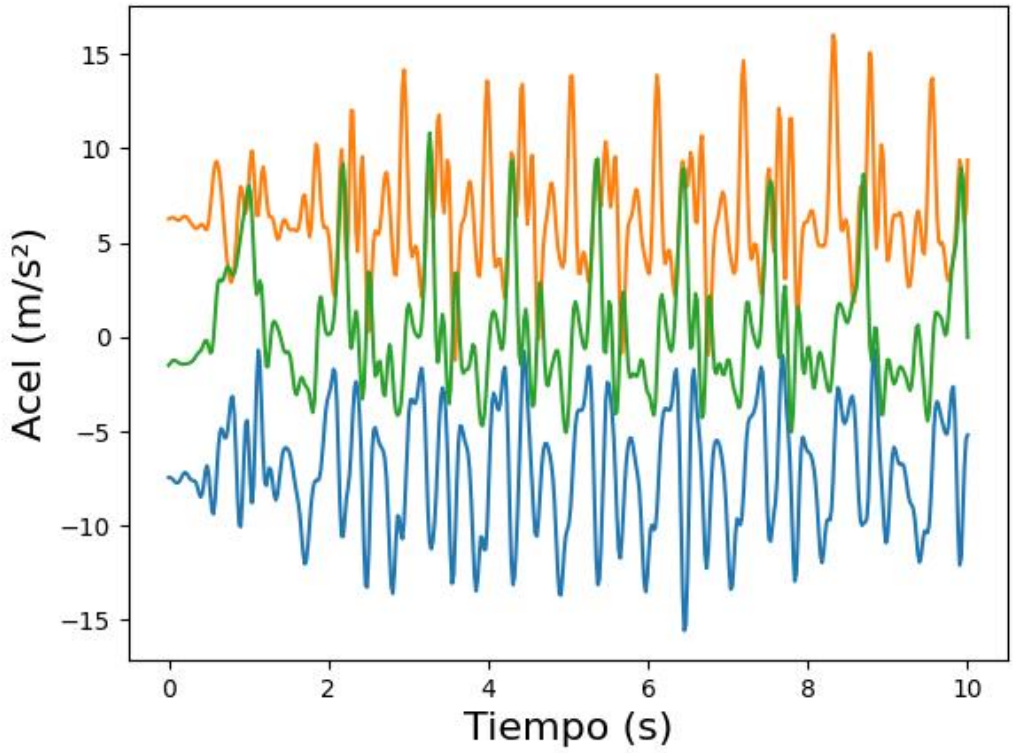


Ilustración 1: Aceleración VS tiempo en prueba controlada andada.

A dichos sujetos se les realizaron una serie de pruebas clínicas: test de Tinetti o evaluación de la movilidad orientada al rendimiento (POMA por sus siglas en inglés) o el *Timed Up and Go* Test (TUGT) y entrevistas sobre las caídas que han tenido recientemente para posteriormente vincularlas con los datos obtenidos del acelerómetro. Estas pruebas son vitales a la hora de predecir las caídas, ya que son los indicadores del riesgo.

De todas las ventanas que se extraen de la vida diaria y que se supone que corresponden a periodos caminando, se extraen las características y luego se hace un *clustering* en 2, 3, hasta 6 *clusters,* dando a lugar a los ficheros llamados por el tutor features_1, features_2 etc. Después al clasificador se pasan sólo los centros de los *clusters.* El objetivo es intentar reducir un poco la falta de fiabilidad de las señales, ya que el detector de la acción "caminar" no es muy bueno. La representación gráfica de esta idea sería la siguiente:

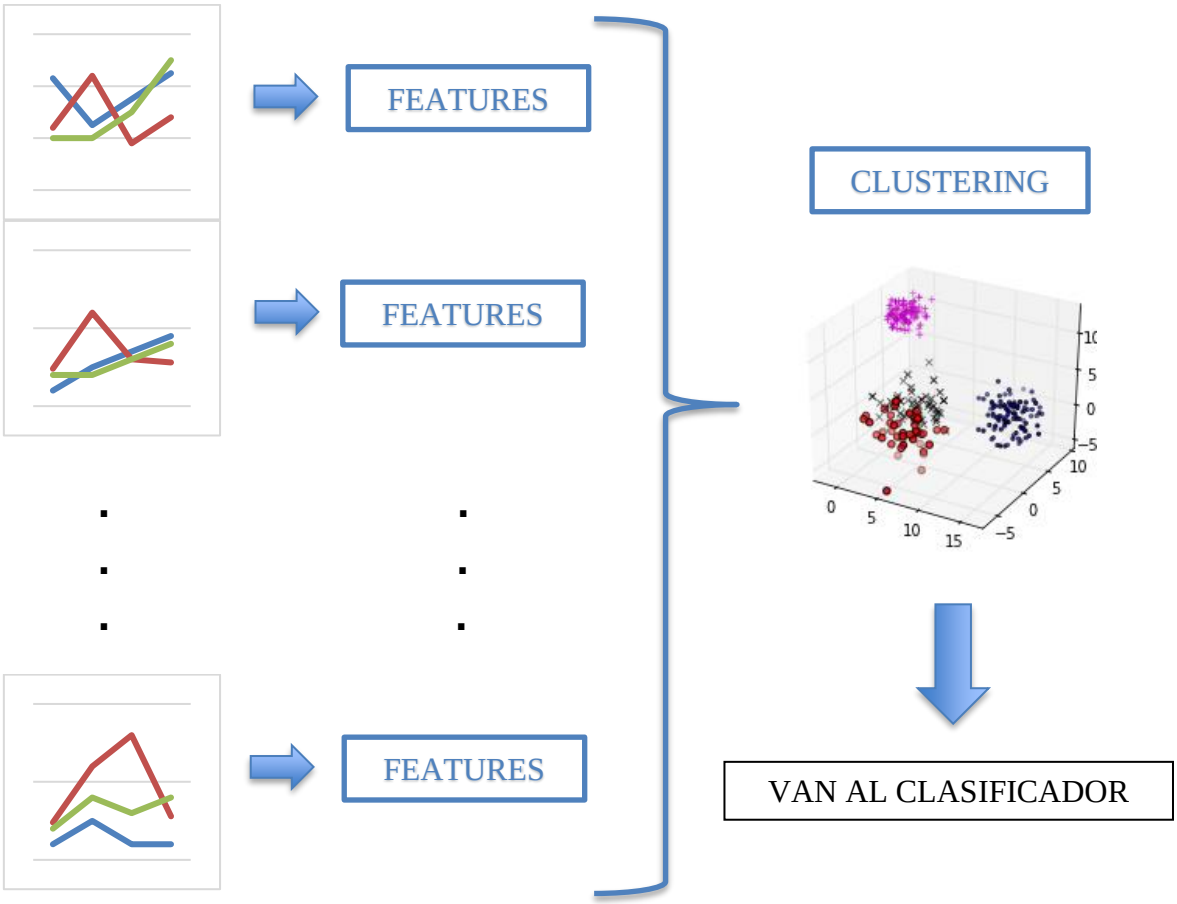


Ilustración 2: Proceso desde que se obtienen los datos en las gráficas hasta el clasificador.

2.2.1 POMA

El POMA o test de Tinetti fue publicado por Mary Tinetti para evaluar la marcha y el equilibrio en ancianos.

La prueba requiere de una silla dura sin brazos y un cronómetro. Tiene dos partes: una evalúa habilidades de equilibrio en una silla y la otra evalúa el equilibrio dinámico durante la marcha. Al realizar dicho test, el evaluador observará varios puntos clave, entre ellos: cómo se levanta y se sienta el paciente en su silla, qué sucede cuando los ojos del paciente están cerrados en la silla, si el paciente se mantiene erguido o no mientras camina, además de observar la velocidad de la marcha, así como la simetría y longitud de los pasos.

La prueba de Tinetti tiene una puntuación de marcha y una puntuación de equilibrio. Utiliza una escala ordinal de 3 puntos de 0, 1 y 2. La marcha se puntúa sobre 12 y el equilibrio se puntúa sobre 16, con un total de 28. Cuanto más bajo sea el puntaje en la prueba de Tinetti, mayor será el riesgo de caída. Aunque esta descripción de la puntación es correcta, cada autor usa un umbral distinto, por ejemplo, EduQTech uso que para una nota menor a 25 tiene riesgo de caída, por lo tanto, se puede decir que es se han juntado el riesgo bajo y medio conforme a la descripción anterior. El grupo de investigación se basó en el trabajo de M.E Tinetti (Tinetti, 1986).

TEST TINNETI PUNTUACIÓN	RIESGO DE CAIDA
<25	Sí
≥25	No

Tabla 1: Riesgo de caídas basado en POMA.

2.2.2 TUGT

El TUGT es una prueba especialmente indicada para medir movilidad y valorar el riesgo de caídas en personas mayores.

Solo se necesita una silla y un cronómetro. La prueba consiste en medir el tiempo necesario para levantarse de la silla (preferiblemente sin utilizar los brazos), caminar hasta la marca situada a tres metros (ambos pies deben rebasar la marca), darse la vuelta y sentarse nuevamente en la silla.

El test tiene un diferente riesgo dependiendo del número de segundos que tarda el sujeto en hacer la prueba. EduQtech establece el umbral en 14 segundos (M. Jacobs y T.Fox, 2008) cómo se puede ver en la tabla.

TEST TUGT PUNTUACIÓN	RIESGO DE CAIDA
< 14	No
≥14	Sí

Tabla 2: Riesgo de caídas basado en TUGT.

2.2.3 RETRO (Caídas anteriores)

RETRO es la base de datos obtenida de las entrevistas sobre las caídas que han tenido recientemente, denominándose RETRO_A a la contestación binaria de los ancianos a la pregunta "¿Has tenido dos o más caídas en los últimos seis meses?". RETRO_B es la respuesta a la misma pregunta, pero con una caída en el mismo periodo de tiempo.

RETRO responde a un nombre corto de la palabra retrospectivo, en el sentido en el que el riesgo de caídas se evalúa por lo que ha pasado anteriormente. En los estudios prospectivos se deduce de las caídas futuras, haciendo un seguimiento de los voluntarios durante un período de tiempo.

La valoración del riesgo es muy sencilla, siendo la misma entre los dos subconjuntos como se puede apreciar en la tabla.

CONTESTACIÓN RETRO_A	RIESGO DE CAIDA
Sí	Con Riesgo
No	Sin riego

Tabla 3: RETRO_A.

CONTESTACIÓN RETRO_B	RIESGO DE CAIDA
Sí	Con Riesgo
No	Sin riego

Tabla 4: RETRO_B.

2.3 Descripción del sistema previo

Existen multitud de trabajos que analizan el riesgo de caída en función de la salida de sensores vestibles, especialmente inerciales. Los trabajos se basan en datos de pruebas controladas (la mayoría) o bien de datos tomados en la vida diaria. Dada la amplitud y la complejidad del problema, el TFG se ha centrado en el uso de diferentes clasificadores y en el análisis de sus prestaciones.

Para la parte correspondiente a la extracción de características, que casi todos los problemas de clasificación necesitan, el director del TFG ha proporcionado un código que extrae una serie de características a partir de ventanas temporales de aceleración de los datos de la vida diaria. En total hay 8 características. También ha indicado un clasificador de partida, *Random Forest*. Tanto las características elegidas como el clasificador aparecen en recientes trabajos (T. E. Lockart et al, 2021). Una diferencia con los trabajos previos, es que los ejes del acelerómetro son conocidos. El sitio más habitual es en la zona lumbar. Por contra, en la base de datos que manejamos, la colocación del móvil en el bolsillo es variable y depende de cada usuario. Por ello antes de analizar cualquier señal se realiza un análisis de componentes principales, incluido en el código, que transforma los ejes a tres ejes ordenados por su nivel de variabilidad.

A modo informativo, se incluye una tabla con las características extraídas:

Características	Obtención	No.
Tiempo de paso	Obtenido a partir del pico de frecuencia de la señal de aceleración total	1
Variación del tiempo de paso	Obtenido a partir de la anchura del pico en frecuencia de la señal de aceleración total	1
RMS	Root Mean Square por eje	3
Relación armónica	Relación entre las sumas de armónicos pares e impares de la frecuencia fundamental de la transformada de Fourier de la señal. Se hace para cada eje	3

Tabla 5: Características extraídas del sistema previo.

3 ASPECTOS GENERALES DE LA INTELIGENCIA ARTIFICIAL

Antes de hablar de conceptos técnicos y adentrarnos profundamente, es necesario tener una idea superficial sencilla de los temas con los que se van a tratar.

El termino inteligencia artificial (IA) se refiere a sistemas o máquinas que imitan la inteligencia humana para realizar tareas y que pueden mejorar iterativamente a partir de la información que recopilan (Oracle, s.f.). La diferencia de la IA con un *software* como los que conocemos es que no está programado para una determinada tarea.

En cuanto a la diferencia entre *Machine Learning* y *Deep Learning* radica en el tipo de algoritmos que se usan en cada caso. Aunque el DL se parece más al aprendizaje humano por su funcionamiento como neuronas. El ML acostumbra a usar técnicas más sencillas como árboles de decisión o redes neuronales de una o unas pocas capas ocultas y el DL redes neuronales, que están más evolucionadas y que tienen más profundidad (capas) (Silva, 2021).

El último término que debemos conocer es el *Big Data*, que es el almacenamiento y procesamiento de cantidades masivas de datos estructurados, semiestructurados y no estructurados con gran potencial para ser extraídos y organizados de forma que proporcionen información valiosa para las organizaciones y empresas. (nexusintegra, 2022)

La relación entre la IA y el *Big Data* es muy fácil de entender, ambos necesitan uno del otro, siendo simple de comprender con el símil de un coche, el *Big Data* sería el combustible para que el motor (IA) funcionará, ya que la inteligencia artificial necesita datos para construir su inteligencia.

Todo lo explicado se ha plasmado en la siguiente figura para una mayor comprensión.

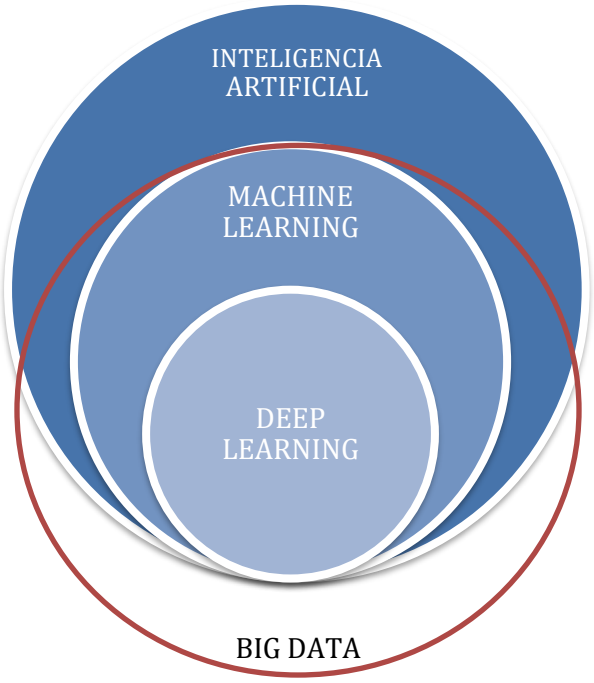


Ilustración 3: Relación entre IA y Big Data.

3.1 Inteligencia Artificial

Una vez conocido lo que es inteligencia artificial, se debe echar una mirada hacia el pasado para tener una perspectiva global.

Aunque hay debate de desde cuándo se puede considerar el punto de partida de la inteligencia artificial, fue en 1956 cuando se habló formalmente en una conferencia por primera vez sobre esta. Desde este año se empezó a explorarla, destacando los trabajos de la Agencia de Proyectos de Investigación Avanzada de Defensa de EEUU.

Un hecho que bastante gente conoce es el que se llevó a cabo por el supercomputador de IBM, ganando al campeón del mundo de ajedrez. Pero no fue realmente a partir de 2012 cuando los expertos consideraron que se estaba produciendo una explosión en la Inteligencia Artificial, presentándose productos comerciales al público en general. Hoy en día nombres como Siri, Alexa o Cortana completamente conocidos.

Actualmente la inteligencia artificial está en todos campos, destacando cuatro de ellos:

- **Medicina:** es el campo que ataña al TFG presentado. Proporciona predicciones de enfermedades, lecturas de pruebas y asistencia personalizada del paciente. Incluso es utilizado en el ámbito de la investigación.
- **Retail:** ayuda a personalizar la compra del consumidor. Además de su utilización en la gestión del inventario.
- **Industria:** la inteligencia artificial ha supuesto en la industria 4.0 una gran revolución ya que se produce dinámicamente. También se puede aplicar para hacer un mantenimiento predictivo e incluso en los controles de calidad.

- **Finanzas:** gracias a la IA se pueden identificar transacciones que tienen probabilidad de ser fraudulentas, así como también automatizar tareas de gestión de grandes datos.

3.2 Machine Learning

Como hemos comentado anteriormente el *Machine Learning* se encarga de analizar datos e identifica patrones para posteriormente realizar la tarea encomendada, donde el entrenamiento es llevado a cabo por las personas para que esta pueda identificar patrones en datos y predecir posteriormente

Según la intervención humana se puede clasificar en:

- **Aprendizaje supervisado:** se suministra un grupo etiquetado de datos. Este es el modelo es el menos complejo.
- **Aprendizaje no supervisado**: los datos están sin etiquetar y extrae de ellos conocimientos o patrones desconocidos anteriormente.
- **Aprendizaje semisupervisado:** se tienen datos parcialmente etiquetados. Comienza con los datos etiquetados para comprender los no etiquetados.
- **Aprendizaje de refuerzo:** el algoritmo aprende observando el mundo que le rodea y retroalimentados conforme obtiene respuestas. Por lo tanto, es un ensayo prueba y error.

También hay una clasificación alternativa según el tipo de resultado:

- **Clasificación:** el resultado es una clase, entre un número limitado de clases. Siendo clase la categoría arbitraria según el problema a resolver.
- **Regresión:** el resultado es un valor numérico.

3.3 Deep Learning

El *Deep Learning* es un conjunto del *Machine Learning* en el que la máquina aprende por ella sola, razonando y sacando sus propias conclusiones, donde el entrenamiento lo realiza la computadora (reconocimiento del habla, la identificación de imágenes o hacer predicciones).

En lugar de organizar datos para que se ejecuten a través de ecuaciones predefinidas, el *Deep Learning* configura parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento. (SAS, s.f.)

3.4 Big data

Big data es un término que describe el gran volumen de datos (estructurados y no estructurados), pero no es la cantidad de datos lo importante. Puede ser analizado para obtener *insights* que conlleven a mejores decisiones y acciones de negocios estratégicas. (SAS, s.f.)

4 ANÁLISIS DE LAS HERRAMIENTAS SOFTWARE

Ante las distintas opciones que se tenían para comenzar a programar se decidió realizarlo en *Jupyter Lab*, ya que se cursó la asignatura de "Introducción a Inteligencia Artificial" en el periodo de Erasmus, en el cual se utilizó dicho *software*.

Para utilizar *Jupyter Lab* se requería previamente el programa "Ubuntu on Windows", el cual se puede obtener de la *Windows Store* de forma gratuita. Antes de seguir profundizando en los programas, tenemos que conocer el lenguaje en el que se va programar y el porqué.

4.1 Python

Python es un lenguaje de programación flexible y diseñado para ser fácil de leer. Además, es orientado a objetos y de alto nivel. Gracias a su sintaxis sencilla es un muy buen lenguaje para aprender a programar. Python utiliza módulos y paquetes, lo cual fomenta el modularidad y la reutilización de código. (Datademia, 2022)

4.1.1 Origen

Python es una creación del informático Guido van Rossum. Rossum trabajaba con el lenguaje ABC, pero no le gustaba, ya que era muy difícil de comprenderlo y difundirlo. Tras estos pensamientos, Rossum creo su propio lenguaje en 1991, subiendo la primera versión en febrero de 1991 a USENET.

El origen del nombre es curioso, ya que como le gustaba leer los episodios de "El circo volador de Monty Python" de la famosa compañía británica de comedia. Le puso a este nuevo lenguaje Python, ya que buscaba que fuera "corto, único y ligeramente misterioso".

4.1.2 Usos

Gracias a su sencillez y sus abundantes posibilidades ha ganado muchos seguidores. Los principales campos donde se utilizan son:

- **Inteligencia artificial y Big data**: su simplicidad y sus numerosas bibliotecas de procesamiento de datos hacen que Python funcione extremadamente bien al analizar y gestionar grandes cantidades de datos en tiempo real.
- **Blockchain**: conocida por ser la base sobre la que se han creado las criptomonedas y los NTFs. Se utiliza porque es seguro y rápido, siendo extremadamente útil al realizar cadenas de bloques, ya que lo simplifica.
- **Desarrollo web**: al utilizar Python en el desarrollo web se consigue webs complejas en menos líneas de código por lo que se optimizan más. Por ejemplo, el *framework* de Python, Django, es utilizado para crear webs dinámicas y extremadamente seguras.
- **Juegos y gráficos 3D**: la creación de gráficos y el manejo de estos en Python es bastante bueno, ya que hay multitud de herramientas y *frameworks.*

4.2 Ubuntu on Windows

Ubuntu on Windows permite usar *Ubuntu Terminal* y ejecutar en línea los comandos de Ubuntu. Esta aplicación instala la versión *Ubuntu 20.04 LTS* en la que se pueden desarrollar aplicaciones multiplataforma, mejorando la ciencia de los datos y flujos de trabajo de desarrollo web dentro de Windows.

Una vez descargado y nada más inicializarlo se abrirá una ventana de la consola y le pedirá que espere uno o dos minutos para que los archivos se descompriman y se almacenen en su ordenador. Posteriormente solo tardar un par de segundos en iniciarse la consola. A continuación, se debe crear una cuenta de usuario y una contraseña para Linux.

Al tener el terminal Linux funcionando dentro de Windows. Lo último es ver si hay actualizaciones e instalarlas, escribiendo el siguiente comando:

```
In [ ]: sudo apt update && sudo apt upgrade
```

Por último, habría que instalar `pip`, el cuál es un sistema de administración de paquetes para instalar y administrar módulos en Python.

```
In [ ]: sudo apt update
        sudo apt install python3-pip
```

Ambos comandos comienzan con `sudo`, porque necesitamos más privilegios para realizar instalaciones. `Apt` es un administrador de paquetes para instalar y administrar software en sistemas basados en Debian GNU.

El comando de actualización actualiza la información del paquete desde el servidor y el de instalación es justo lo que parece, es para instalar software, siendo `python3-pip` es el nombre del paquete que queremos instalar.

4.3 Jupyter Lab

Jupyter Lab es un editor online que te permite trabajar con documentos como los Jupyter notebooks, editores de textos, terminales y componentes *custom*, de forma flexible, integrada y extensible, teniendo una estructura modular.

```
In [ ]: pip3 install --user jupyterlab
```

Después de la instalación de *Jupyter*, hay que asegurarse que se encuentre el nuevo paquete, una forma fácil de hacerlo es simplemente cerrar *Ubuntu* y volver a iniciarlo:

```
In [ ]: jupyter lab
```

Como es normal en los sistemas operativos basados en Unix, al presionar las teclas ctrl y c se detiene el servidor *Jupyter*. Para iniciarlo sin *tokens* y contraseñas (ahorra bastante tiempo) se puede escribir el siguiente comando:

```
In [ ]: jupyter lab --LabApp.token=''
```

Suponiendo que está ejecutando como se ha explicado, sin *token*s, simplemente quedaría ir al sistema y abrir el navegador, yendo a la siguiente página:

```
In [ ]: Localhost:8888
```

Una vez conectado al servidor Jupyter Lab ya disponible para codificar en *Python3*.

4.4 Librerías

Hay una seria de librerías fundamentales para poder trabajar en el proyecto:

4.4.1 Scikit learn

Scikit-Learn es una de estas librerías gratuitas para Python que cuenta con algoritmos de clasificación, regresión, *clustering* y reducción de dimensionalidad. Además, es de código abierto y unifica bajo un único marco los principales algoritmos y funciones, facilitando en gran medida todas las etapas de preprocesado, entrenamiento, optimización y validación de modelos predictivos. (Rodrigo, 2020)

```
In [ ]: pip3 install -U scikit-learn
```

4.4.2 Numpy

NumPy es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. (Wikipedia, s.f.)

```
import numpy as np
```

La parte del código `as np` le dice a Python que asigne `NumPy` al alias de np

4.4.3 Pandas

Pandas es una biblioteca de *software* escrita como extensión de *NumPy* para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales, o leer datos de ficheros como hojas de cálculo. (Wikipedia, s.f.)

```
import panda as pd
```

Igual que ocurre en la biblioteca *NumPy*, la parte del código `as pd` le dice a Python que asigne Pandas al alias de `pd`

4.4.4 Tensorflow y Keras

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. (Wikipedia, s.f.)

A su vez, *Keras*, es una biblioteca de Redes Neuronales de Código Abierto escrita en *Python*. Es capaz de ejecutarse sobre *TensorFlow, Microsoft Cognitive Toolkit o Theano*. Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible. (Wikipedia, s.f.)

5 CLASIFICADORES

5.1 Random Forest

Un *Random Forest* es un conjunto (*ensemble*) de árboles de decisión combinados con *bagging*. Al usar *bagging,* lo que en realidad está pasando, es que distintos árboles ven distintas porciones de los datos. Ningún árbol ve todos los datos de entrenamiento. Esto hace que cada árbol se entrene con distintas muestras de datos para un mismo problema. De esta forma, al combinar sus resultados, unos errores se compensan con otros y tenemos una predicción que generaliza mejor. (Heras, 2020)

5.1.1 Aspectos positivos

- Funciona aceptablemente sin el ajuste de sus parámetros para todos los casos.
- Correcto funcionamiento para problemas de clasificación y regresión.
- Al utilizar múltiples árboles se reduce considerablemente el riesgo de *overfiting* (causa en la que se obtiene malos resultados, ya que aprende los casos particulares**,** pero no es capaz reconocer datos nuevos).
- Con nuevas muestras funciona bien porque utiliza multitud árboles ponderados.

5.1.2 Aspectos negativos

- El tiempo en el que se realiza el entrenamiento puede ser alto.
- Funcionamiento eficiente con base de datos pequeñas.
- Opciones muy limitadas para ajustar los parámetros.

5.2 Multi-Layer Perceptron

Aunque *Scikit-learn* es una librería de aprendizaje automático también puede desarrollar un modelo de aprendizaje profundo. *MLP* está formado por tres capas como mínimo, la de entrada**,** una de salida y *n* capas ocultas entre ambas.

5.2.1 Aspectos positivos

- Puede crear modelos no lineales.
- Con `partial_fit`, se puede entrenar en tiempo real.

5.2.2 Aspectos negativos

- En MLP, si no se entrena bien, las salidas pueden ser muy imprecisas.
- No es posibilite utilizar la GPU.
- No se puede ajustar los parámetros de cada capa.

5.3 Keras

Como ya hemos comentado, *Keras* es un *framework* para hacer Deep Learning. Una de sus opciones es un MLP, por lo que se ha implementado dicha alternativa. Esta cuenta con más capas ocultas que en el MLP de *scikit-learn*. La librería *keras* está más adaptada para este tipo de estructuras que tienden a ser "profundas" (*Deep*).

5.3.1 Aspectos positivos

- Simplificación e interfaz diseñado para un fácil entendimiento.
- Soporte para diferentes GPU.
- Grandes empresas mantienen y desarrollan *Keras*.
- Extraordinaria compatibilidad entre las diversas plataformas.
- Considerable número de usuarios.
- Modelos predeterminados previamente entrenados.

5.3.2 Aspectos negativos

- Limitaciones a cuanto el interfaz es de aplicaciones de bajo nivel.
- Es menos flexible y más estandarizado que otras opciones.

6 IMPLEMENTACIÓN EN JUPYTER LAB

6.1 Ancianos en la vida diaria

Lo primero es cargar las librerías que vamos a utilizar a la hora de programar. Estas han tenido de ser instaladas previamente en *Ubuntu on Windows* con el módulo `pip` descrito anteriormente.

```
# Using numpy to convert to arrays
import numpy as np
# Using pandas to management and analysis of data structures
import pandas as pd
#Using keras as High-level TensorFlow API for building and training deep learning models
import tensorflow as tf
from tensorflow import keras
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Using Skicit-learn to ensemble of Decision Trees
from sklearn.ensemble import RandomForestClassifier
# Using Scikit-learn to obtain sensitivity and specificity values
from sklearn.metrics import precision_score, recall_score
# Using Scikit-learn to obtain a better classification metric for unbalanced data
from sklearn.metrics import f1_score
# Using Scikit-learn to classifier with neural networks
from sklearn.neural_network import MLPClassifier
# Using Scikit-learn to obtain accuracy values
from sklearn.metrics import accuracy_score
```

Ilustración 4: Primera parte del código "Ancianos vida diaria": librerías.

Como podemos ver se encuentra las librerías *Numpy* y *Pandas*, descritas anteriormente, así como *Tensorflow* y *Keras*. Además, se encuentran diversos módulos *Sklearn* para entrenar los datos, para el clasificador *MLP* y el *Random Forest*, así como para las métricas para la evaluación.

Lo siguiente es cargar la base de datos suministrada por el tutor. Lo primero es subir los archivos a *Jupyter Lab* y después cargarlos uno a uno, para eso utilizamos el bucle de la siguiente ilustración. A continuación, se dividen los datos en dos subconjuntos, para entrenar y testear. Siendo el tamaño de testear de un 33%. Además, se ha introducido el parámetro `random_state` en 42, lo que significa que los resultados serán los mismos cada vez que ejecute la división del conjunto para obtener resultados reproducibles.

```
#loop to work with all features
for nn in range (1,6):
    dirname = "features/features_"+ str(nn)+"/"
    xname = dirname + "X_"+ str(nn)+ ".npy"
    for name in ["poma", "retro","retroB" , "tugt"]:
        yname = dirname + "y"+ name + "_"+ str(nn)+ ".npy"
        print ("Feature "+ str(nn)+ ": " + name)
        X = np.load(xname)
        y = np.load(yname)

        #Splitting the data into training and testing sets, being the size of test part 33%.
        #Setting the random state to 42 which means the results will be the same each time
        #I run the split for reproducible results.
        (X_train, X_test, y_train, y_test) = train_test_split(X, y,test_size=0.33,random_state=42)
```

Ilustración 5: Segunda parte del código "Ancianos vida diaria": loop y entrenamiento.

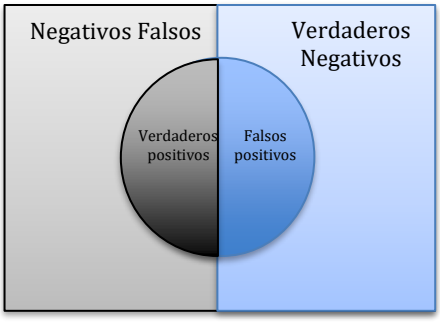
A partir de aquí, se programa los diferentes clasificadores, empezando por ejemplo por *Random Forest*.

Lo primero es entrenar dicho algoritmo. Se ha elegido 300 árboles, aunque contra más árboles suele ser mejor, el tiempo de entrenamiento es mayor. Además, se ha utilizado el mismo `random_state` y un `class_weight = "balanced"` para hacer que los pesos de cada clase no valgan uno, sino que automáticamente los pesos sean inversamente proporcionales a las frecuencias de clase en los datos de entrada:

```
n_samples / (n_classes * np.bincount(y)
```

Ahora que el modelo ya ha sido entrenado, se tiene que predecir sobre el conjunto de datos de testeo, ya que nunca el algoritmo tiene que conocer las respuestas y saber cuánto de bueno es haciéndolo. Para ello se utilizan cinco métricas:

- **Accuracy:** Es básicamente el número total de predicciones correctas dividido por el número total de predicciones.
- **Sensivity o tasa de Verdaderos Positivos:** es la proporción de casos positivos que fueron correctamente identificadas por el algoritmo.
- **Specifity o tasa de Verdaderos Negativos:** son los casos negativos que el algoritmo ha clasificado correctamente.
- **F1score:** Se basa en la *precision* y el *recall*, por lo que las definiremos previamente: Precisión proporciona la calidad de la predicción, siendo el porcentaje de la clase positiva que realmente son; y el r*ecall* muestra la cantidad de la clase positiva se ha identificado correctamente. (Herás, 2020)



PRECISIÓN: 

RECALLL: 

Conociendo esto, *F1Score* es dada por la media entre precisión y *recall,* se puede decir que combina precisión y *recall* en una sola métrica.

```
(2 x precision x recall / (precision+recall))
```

- **Ponderada:** propia métrica que utiliza tanto la *sensivity* como la *specificity* con el porcentaje que hay de ceros y unos. El resultado es muy parecido a la función *f1_score*, dando así una visión global muy acertada.

```
print ("   ---- RAMDOM FOREST ----")
# Instantiate model with 300 decision trees
rf = RandomForestClassifier(random_state=42, n_estimators=300, class_weight = "balanced")
# Trainning the model on training data
rf.fit(X_train, y_train)
#Using the forest's predict method on the test data
y_pred = rf.predict(X_test)
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {y_pred}")
#Using different methods of evaluation
Sensivity = recall_score(y_test, y_pred)
Specifity = recall_score(y_test, y_pred, pos_label=0)
#Printing the results obtained on the screen
print(f"   Accuracy: {rf.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity}")
print(f"   Specifity: {Specifity}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada: {(y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity}")
print()
```

Ilustración 6: Tercera parte del código "Ancianos vida diaria": Random Forest.

El siguiente clasificador sería el *MLP (Stickt learn)*. En este caso el proceder es igual al modelo anterior por la salvedad que hemos utilizado como máximo 25000 interacciones. (valor que se ha ido ajustando conforme las métricas salían mejores, teniendo en cuenta también el tiempo de entrenamiento).

```
print ("   ---- MPL ----")
 # Instantiate model with 2500 max interactions
cla = MLPClassifier(max_iter=25000)
cla.fit(X_train, y_train)
#Using the MLPClassifier on the test data.
y_pred=cla.predict( X_test)
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {y_pred}")
#Using different methods of evaluation
Sensivity_1=recall_score(y_test, y_pred)
Specifity_1=recall_score(y_test, y_pred, pos_label=0)
#Printing the results obtained on the screen
print(f"   Accuracy: {cla.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity_1}")
print(f"   Specifity: {Specifity_1}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada: {(y==0).sum()/y.shape[0]*Sensivity_1+(y==1).sum()/y.shape[0]*Specifity_1}")
print()
```

Ilustración 7: Cuarta parte del código "Ancianos vida diaria": MLP.

Respecto al clasificador *Keras*, se puede decir que el procedimiento es diferente a los anteriores.

Primero creamos un modelo vació de tipo `Secuential`, se ha seleccionado este tipo porque queremos crear un modelo con varias capas de neuronales en orden.

Creamos cuatro capas `Dense` con `model.add()`. Utilizaremos `relu` para activarlo. Además, se agregará una función `sigmoid` para para una última capa únicamente una neurona de salida. Se ha elegido esta cantidad de neuronas, capas y sus funciones de activación por las recomendaciones del libro de Aurélien Géron (Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019). Además de probar constantemente y obtener mejores resultados con estos parámetros.

Para mejorar los resultados se ha añadido una pérdida *(`loss`)*, un `optimizer` de los pesos de las conexiones de las neuronas y las métricas que se quiere obtener. (Na8, 2018)

Tras esto solo quedaría entrenar el modelo como en el resto de clasificadores incluyendo 100 *epochs* (interacciones de aprendizaje). Cuanta más cantidad, mejor será el resultado, pero mayor tiempo tardará, teniendo cuidado obviamente con no pasarnos, ya que se puede producir *overfitting*, por lo que para comprobar que no se esté produciendo habría que revisar las métricas para realmente saber si estamos sobreentrenando.

```
print ("   ---- KERAS ----")
# Instantiate model with 4 layers
model = keras.models.Sequential()
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))
#Improvement of the model
model.compile(loss="mean_squared_error",
optimizer="adam",
metrics=["binary_accuracy"])
#Trainning data with 100 epochs
model.fit(X_train, y_train,epochs=100)
#Using Keras on the test data.
y_pred = model.predict(X_test).round()
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {np.transpose(y_pred) }")
#Using different methods of evaluation
Sensivity_2=recall_score(y_test, y_pred)
Specifity_2=recall_score(y_test, y_pred, pos_label=0)
Accuracy_2=accuracy_score(y_test, y_pred)
#Printing the results obtained on the screen
print(f"   Accuracy: {Accuracy_2"}")
print(f"   Sensivity: {Sensivity_2}")
print(f"   Specifity: {Specifity_2}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada: {(y==0).sum()/y.shape[0]*Sensivity_2+(y==1).sum()/y.shape[0]*Specifity_2}")
print()
```

Ilustración 8: Quinta parte del código "Ancianos día a día": Keras.

6.1.1 Comparación entre los tres métodos.

Primero y tras no apreciar a simple vista unas diferencias significativas entre clasificadores y *features*, se decide en una tabla en las que se pueda considerar los datos de una manera mucho más clara. Para ello se hace uso de tres tablas, una para cada clasificador con sus cinco correspondientes *features* y métricas como se pueden apreciar en la siguiente página.

Tras analizar dichas tablas, se precisará únicamente de las *feutures_1*, ya los resultados son parejos entre todas ellas, siendo incluso mejor y con un tiempo requerido menor porque el *cluster* es de dos. Además, se va a analizar solo con la métrica "ponderada" porque representa una visión general la calidad de la predicción.

RANDOM FOREST																					
		POMA					TUGT					RETRO					RETRO_B				
Datos	No. clust	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado
features_1	2	0,417	0,679	0,875	0,526	0,596	0,500	0,786	0,900	0,571	0,617	0,000	0,929	1,000	0,000	0,171	0,286	0,571	0,667	0,250	0,434
features_2	4	0,714	0,745	0,765	0,682	0,734	0,353	0,709	0,868	0,429	0,504	0,000	0,800	1,000	0,000	0,171	0,286	0,600	0,794	0,353	0,484
features_3	8	0,381	0,615	0,761	0,432	0,529	0,219	0,697	0,896	0,238	0,417	0,000	0,798	1,000	0,000	0,171	0,289	0,578	0,781	0,361	0,481
features_4	16	0,533	0,733	0,838	0,580	0,652	0,344	0,760	0,923	0,447	0,514	0,023	0,788	0,977	0,042	0,186	0,352	0,641	0,837	0,443	0,542
features_5	32	0,402	0,661	0,834	0,488	0,571	0,350	0,727	0,908	0,454	0,513	0,073	0,811	0,983	0,128	0,228	0,356	0,621	0,805	0,434	0,531

Tabla 8: Resultados Random Forest.

MPL																					
		POMA					TUGT					RETRO					RETRO_B				
Datos	No. clust	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado
features_1	2	0,500	0,607	0,688	0,522	0,573	0,625	0,750	0,800	0,588	0,676	0,000	0,750	1,000	0,000	0,390	0,000	0,750	1,000	0,000	0,390
features_2	4	0,429	0,618	0,735	0,462	0,548	0,471	0,673	0,763	0,471	0,556	0,000	0,800	1,000	0,000	0,171	0,429	0,600	0,706	0,450	0,537
features_3	8	0,381	0,624	0,776	0,438	0,535	0,000	0,706	1,000	0,000	0,293	0,000	0,800	1,000	0,000	0,171	0,000	0,587	1,000	0,000	0,390
features_4	16	0,347	0,696	0,880	0,441	0,555	0,131	0,737	0,974	0,219	0,378	0,000	0,802	1,000	0,000	0,171	0,045	0,599	0,977	0,084	0,409
features_5	32	0,333	0,628	0,826	0,419	0,526	0,129	0,672	0,932	0,202	0,364	0,000	0,811	1,000	0,000	0,171	0,068	0,587	0,945	0,118	0,410

Tabla 6: Resultados MLP (scikit-learn).

KERAS																					
		POMA					TUGT					RETRO					RETRO_B				
Datos	No. clust	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado	Sensibilidad	Exactitud	Especificidad	F1score	Ponderado
features_1	2	0,500	0,607	0,750	0,545	0,598	0,500	0,750	0,850	0,533	0,602	0,000	0,750	0,714	0,000	0,279	0,000	0,750	0,714	0,000	0,279
features_2	4	0,429	0,618	0,676	0,439	0,525	0,059	0,673	0,868	0,087	0,296	0,000	0,800	1,000	0,000	0,171	0,190	0,600	0,676	0,222	0,380
features_3	8	0,571	0,624	0,522	0,490	0,552	0,219	0,706	0,935	0,318	0,428	0,045	0,798	0,920	0,920	0,195	0,200	0,587	0,859	0,286	0,457
features_4	16	0,490	0,696	0,768	0,500	0,592	0,213	0,737	0,936	0,310	0,425	0,000	0,802	0,943	0,000	0,161	0,159	0,599	0,863	0,231	0,430
features_5	32	0,351	0,651	0,819	0,433	0,533	0,307	0,672	0,870	0,389	0,472	0,073	0,811	0,946	0,112	0,222	0,294	0,587	0,715	0,344	0,458

Tabla 7: Resultados MLP (Keras)

ANCIANOS EN LA VIDA DIARIA (FEATURES 1 + PONDERADA)				
CLASIFICADOR	POMA	TUGT	RETRO	RETRO_B
RANDOM FOREST	0,596	0,617	0,171	0,434
MLP (scikit-learn)	0,573	0,676	0,390	0,390
KERAS	0,598	0,602	0,279	0,279

Tabla 9: Features 1 + Ponderada.

Tras examinar la tabla se puede concluir que no hay diferencias significativas entre ninguno de los tres clasificadores, pero sí entre las pruebas. POMA y TUGT son muy superiores a las dos definiciones basadas en la entrevista.

Podemos concluir que para la predicción de caídas según POMA y TUGT funciona bastante bien, en el sentido de que las señales de aceleración pueden predecir aceptablemente su resultado.

6.1.2 Generación de gráficas para comprobar el movimiento en las ventanas seleccionadas como entrada al sistema.

Uno de los puntos críticos en la utilización de datos de la vida diaria es la detección previa que se realiza de partes de la señal que correspondan al movimiento de caminar, ya que muchas técnicas de clasificación se basan en este tipo de movimiento. Sin embargo, en el código proporcionado por el tutor esta detección es bastante básica y se basa simplemente en un nivel de actividad de la señal de aceleración. Por ello, se ha querido comprobar de forma visual cómo de acertado es esta selección previa de las ventanas de aceleración, dónde sólo debería haber movimientos correspondientes a una persona caminando. Por ello, se ha introducido en el código suministrado por el tutor una serie de líneas para saber si se está correctamente filtrando que el sujeto se encuentra andando, para ello hay que guardar unas cuantas gráficas por voluntario para después dibujarlas y comprobar si tiene visualmente cierta periodicidad como en las pruebas controladas.

La parte del código introducida se encuentra en la función `allFeature(n_clusters, subjects = None`. En primer lugar, se crea un bucle que genera 5 números aleatorios entre 12000 (número aproximados de ventanas generadas por cada sujeto), dicho número se utilizará para guardar graficas azarosamente, por lo que cada vez que ejecutemos el código tendremos 5 gráficas del sujeto.

```
def allFeatures(n_clusters, subjects = None):
    nyq = FNYQ # For smartphone 50Hz/2
    voluntDic = getVoluntDic2('data/datosUsuarios.csv')
    rdir = 'data/'
    print('OJO features low pass 8 Hz')
    print('OJO que las features se igualen en cuanto a valores (normalizar)')
    print('Esto por el tema de kmeans')
    print('OJO tb a la longitud de la segnal. Si disminuyo wsize')
    print('y quiero llegar a 0.25 Hz en los analisis en freq hay problemas')
    print('porque casi no tengo resolucion y salen indices negativos al buscar anchura')
    coefs = scipy.signal.butter(5, 8.0/nyq, 'lowpass')
    b1 = coefs[0]
    a1 = coefs[1]
    wsize = 512 # 512/50 s, 512 = 2**9
    if(subjects is None):
        subjects = sorted(voluntDic.keys())
    #import ipdb;ipdb.set_trace()
    X = []
    ypoma = []
    ytugt = []
    yretro = []
    yretroB = []
    listanumeros = []
    i=0
    for subject in subjects:
        print(subject)
        nwalk = 0.0
        nwin = 0.0
        path=rdir+subject+'/'
        segs, wtimes, dum1, dum2=getSegmentsPath(path)
        Xkm = [] # For kmeans
        for ac in segs:
            acf = scipy.signal.filtfilt(b1, a1, ac, axis = 0, padtype='constant')
            acft = np.sqrt( (acf**2).sum(1))
            ##
            wints = np.split( acft , range(wsize, acft.shape[0], wsize))
            wints.pop(-1)
            wins = np.split( acf , range(wsize, acf.shape[0], wsize))
            wins.pop(-1)
            ##
            nwin += len(wins)
```

Ilustración 9: Generación de gráficas: 1º parte.

```
            for win, wint in zip(wins, wints):
                if(isWalking(wint)):
                    nwalk += 1.0

                    #SABER CUANTAS HAY
                    if i == 0:
                        for nn in range (1,6):
                            numero = random.randint(1, 12000)
                            listanumeros.extend([numero])

                    for element in listanumeros:
                        if i == element:
                            plt.figure(1)
                            plt.plot(win)
                            plt.xlabel("Tiempo(s)")
                            plt.ylabel("Ace(m/s^2)")
                            plt.savefig("PlotGeneratedUsingMatplotlib"+ str(i)+".png")
                            plt.close(1)

                    i+=1

                    pca = PCA()
                    win2 = pca.fit_transform(win)
                    feat = extractFeatures(win2, wsize)
                    if(feat is not None):
                        Xkm.append(feat.reshape(1, feat.shape[0]))
                    # A este nivel win es una array 512x3, 512 muestras
                    # de los tres ejes, mientras que wint es la aceleracion
                    # total (modulo del vector) de 512x3
                    # Se supone que esta andando. win2 tras el PCA

        # Esto seria el porcentaje de tiempo andando dentro de la actividad
        print(nwalk/nwin)
        Xkm = np.vstack(Xkm)
        # I have decided to scale features
        scaler = StandardScaler()
        X2 = scaler.fit_transform(Xkm)
        # Then cluster
        km = KMeans(n_clusters=n_clusters, n_init=100, max_iter=20000)
        km.fit(X2)
        # Then add but undo scale to recover original units
        X.append(scaler.inverse_transform(km.cluster_centers_))
        yretro.extend(n_clusters*[ int( faller(voluntDic, subject, 'retro'))])
        yretroB.extend(n_clusters*[ int( faller(voluntDic, subject, 'retroB'))])
        ytugt.extend(n_clusters*[ int( faller(voluntDic, subject, 'tugt14'))])
        ypoma.extend(n_clusters*[ int( faller(voluntDic, subject, 'poma25'))])
    X = np.vstack(X)
    yretro = np.array(yretro)
    yretroB = np.array(yretroB)
    ytugt = np.array(ytugt)
    ypoma = np.array(ypoma)
    return X, yretro, yretroB, ytugt, ypoma
```

Ilustración 10: Generación de gráficas: 2º parte.

Tras haber generado 10 gráficas (2 ejecuciones) se puede observar que algunas veces se identifica un movimiento periódico. Esto es típico del movimiento de andar, por lo que el filtro funciona correctamente. Sin embargo, en otras ocasiones no, ya que no se aprecia periodicidad alguna. La introducción de los datos incorrectos no ayudará al clasificador.

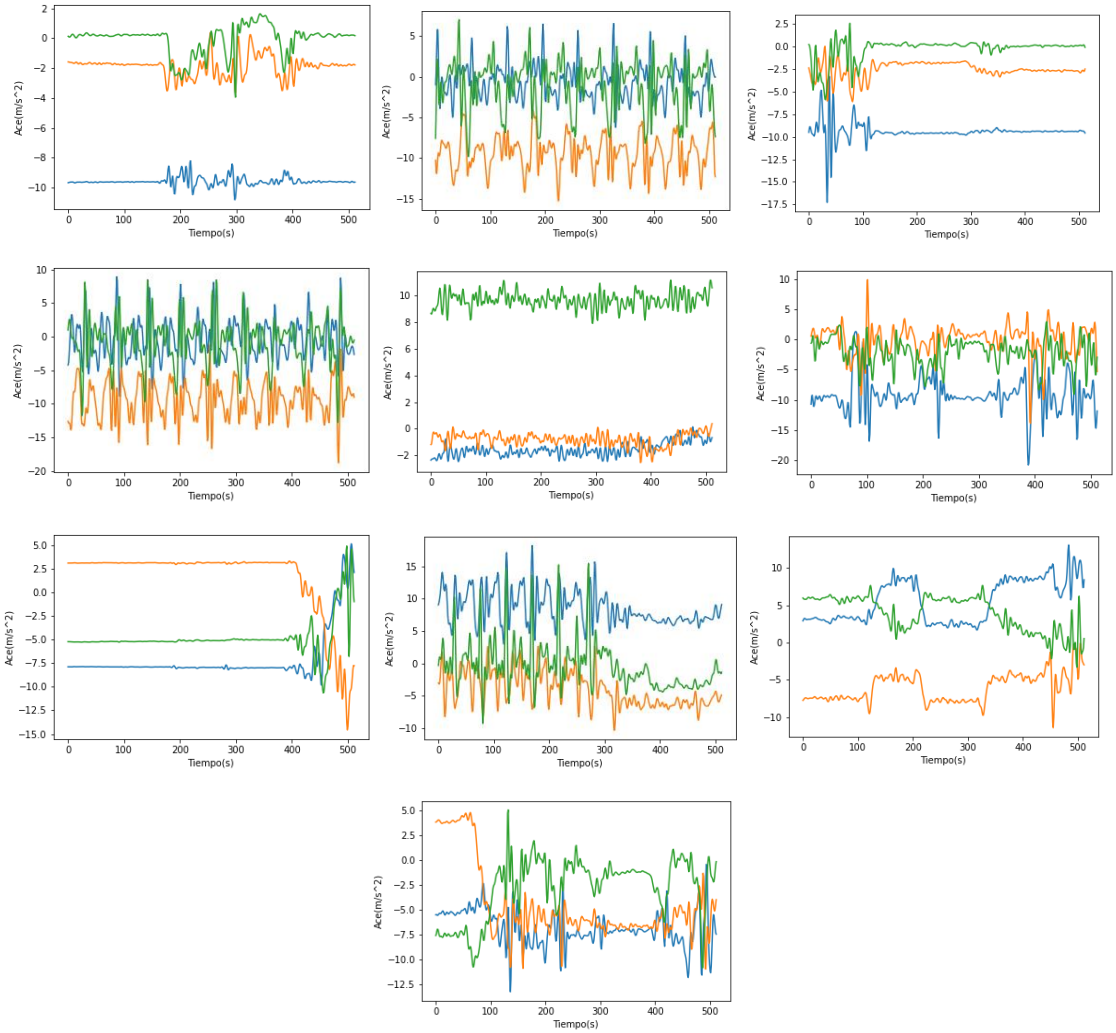


Ilustración 11: Gráficas generadas

6.2 Ancianos andando: prueba controlada

Para este apartado se ha realizado varias funciones del código que se utiliza para obtener los datos que introducimos a los clasificadores a partir de los que están en bruto de los acelerómetros. En este caso utilizaremos los datos de las pruebas controladas, es decir estamos seguros de que el movimiento correspondiente a la aceleración es caminar.

Lo primero que se hace es obtener de la hoja de cálculo "datos usuarios" la lista de sujetos para posteriormente inicializar las listas que queremos conseguir como objetivo final.

```
def allFeatures(subjects = None):
    """
    Si subjects es None (no se le pasa parametro) entonces saca el la
    lista de voluntarios
    Tambien se le puede pasar subjects = ["T001", "T002"], es decir
    una lista de voluntarios. Entonces solo analizara los voluntarios de
    la lista.
    """
    # IVAN GÓMEZ
    nyq = FNYQ # For smartphone 50Hz/2
    voluntDic = getVoluntDic2('data/datosUsuarios.csv')
    rdir = 'data/'
    if(subjects is None):
        subjects = sorted(voluntDic.keys())
    #import ipdb;ipdb.set_trace()
    X = []
    ypoma = []
    ytugt = []
    yretro = []
    yretroB = []
```

Ilustración 12: Primera parte "Ancianos andando": Datos hoja de cálculo + inicializaciones listas.

Tras esto se obtiene cada uno de los archivos de cada individuo donde se encuentra andando. Dependiendo del individuo hay un diferente número de archivos, llegando a no haber ninguno o incluso a ser este defectuoso. Para su implementación se ha creado con una estructura *if else* como se puede apreciar en las siguientes ilustraciones.

```
for subject in subjects:

    if subject == "T003":
        continue
    # Aqui: para cada individuo, y cada archivo andando. OJO algunos no tienen, nombre mal.
    # Leer archivo getLabelledAcelFilter
    if subject == "T006" or  subject == "T024" or  subject == "T026" or  subject == "T027":
        fname=rdir+subject+"/"+subject +"andar1"+".log"
        print(fname)
        final= allFeaturesFINAl(fname)
        X.append(final)
        yretro.append(int(faller(voluntDic, subject, "retro")))
        yretroB.append(int(faller(voluntDic, subject, "retroB")))
        ypoma.append(int(faller(voluntDic, subject, "poma25")))
        ytugt.append(int(faller(voluntDic, subject, "tugt14")))
        continue
```

Ilustración 13: Segunda parte "Ancianos andando": Archivos vacíos + únicamente un archivo.

```
if subject >= "T009" and subject <="T013" or subject == "T017" or subject == "T018" \
or  subject == "Z002" or subject == "Z001" or subject == "T008":
    for nn in range (1,3):
        fname=rdir+subject+"/"+subject +"andar"+str(nn)+".log"
        print(fname)
        final= allFeaturesFINAl(fname)
        X.append(final)
        yretro.append(int(faller(voluntDic, subject, "retro")))
        yretroB.append(int(faller(voluntDic, subject, "retroB")))
        ypoma.append(int(faller(voluntDic, subject, "poma25")))
        ytugt.append(int(faller(voluntDic, subject, "tugt14")))
        continue
```

Ilustración 14: Ilustración 15: Tercera parte "Ancianos andando": Dos archivos.

```
else:
    for nn in range (1,4) :
        fname=rdir+subject+"/"+subject +"andar"+str(nn)+".log"
        print(fname)
        final= allFeaturesFINAl(fname)
        X.append(final)
        yretro.append(int(faller(voluntDic, subject, "retro")))
        yretroB.append(int(faller(voluntDic, subject, "retroB")))
        ypoma.append(int(faller(voluntDic, subject, "poma25")))
        ytugt.append(int(faller(voluntDic, subject, "tugt14")))
```

Ilustración 16:Ilustración 10: Tercera parte "Ancianos andando": Tres archivos

Dentro de estas estructuras secuenciales condicionales se encuentra la segunda función desarrollada, `allFeaturesFinal(fname)`, la cual simplemente se ha implementado para tener una menor cantidad de código y que este de forma más clara. En esta función se lee el archivo, se obtiene unos ejes determinados y finalmente se logra una señal filtrada y transformada.

```
def allFeaturesFINAl(fname):

    acf = getLabelledAcelFilter(fname)
    pca = PCA()
    acf2 = pca.fit_transform(acf)
    X1=extractFeatures(acf2)

    return X1
```

Ilustración 17: Cuarta parte "Ancianos andando": Función en la que reduce el código.

Lo último que se tendría que hacer es pasar de las listas generadas a *array* y devolverlas en dicha función cuando es utilizada.

```
# Pasamos de lista a array y las devolvemos
X = np.vstack(X)
yretro_A = np.array(yretro)
yretroB = np.array(yretroB)
ytugt = np.array(ytugt)
ypoma = np.array(ypoma)

# AQUI: guardar en ficheros si no se quiere repetir
# cada vez
return  X,yretro_A, yretroB, ytugt, ypoma
```

Ilustración 18: Quinta parte "Ancianos andando": Transformación a array y devolución.

Tras realizar esta parte habría que programar el mismo código que en el apartado anterior, pero con las modificaciones necesarias.

Lo primero que habría que hacer es almacenar en una variable los *arrays* obtenidos.

```
DATOS=allFeatures()
```

Ilustración 19: Sexta parte "Ancianos andando": Almacenamiento de los arrays obtenidos.

A continuación, se realiza un bucle que escriba el nombre del tipo de prueba realizado con su correspondiente elemento del *array* (donde ha sido almacenado dicho test anteriormente) y se usan de nuevo los diferentes clasificadores.

```
X = DATOS[0]

for nn in range (1,5):
    if nn == 1:
        print('Retro_A')
    elif nn == 2:
        print('Retro_B')
    if nn == 3:
        print('TUGT')
    elif nn == 4:
        print('POMA')

    y = DATOS[nn]

    #Splitting the data into training and testing sets, being the size of test part 33%.
    #Setting the random state to 42 which means the results will be the same each time
    #I run the split for reproducible results.
    (X_train, X_test, y_train, y_test) = train_test_split(X, y,test_size=0.33,random_state=42)

    print ("   ---- RAMDOM FOREST ----")
    # Instantiate model with 300 decision trees.
    rf = RandomForestClassifier(random_state=42, n_estimators=300, class_weight = "balanced")
    # Trainning the model on training data.
    rf.fit(X_train, y_train)
    #Using the forest's predict method on the test data.
    y_pred = rf.predict(X_test)
    print (f"   TEST      : {y_test}")
    print(f"   PREDICCION: {y_pred}")
    #Using different methods of evaluation
    Sensivity = recall_score(y_test, y_pred)
    Specifity = recall_score(y_test, y_pred, pos_label=0)
    #Printing the results obtained on the screen.
    print(f"   Accuracy: {rf.score(X_test, y_test)}")
    print(f"   Sensivity: {Sensivity}")
    print(f"   Specifity: {Specifity}")
    print(f"   F1score: {f1_score(y_test, y_pred)}")
    print(f"   Ponderada: {(y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity}")
    print()
```

Ilustración 20: Séptima parte "Ancianos andando": Inicio del bucle + Random forest.

```
print ("   ---- MPL ----")
 # Instantiate model with 3000 max interactions
cla = MLPClassifier(max_iter=30000)
cla.fit(X_train, y_train)
#Using the MLPClassifier on the test data.
y_pred=cla.predict( X_test)
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {y_pred}")
#Using different methods of evaluation
Sensivity_1=recall_score(y_test, y_pred)
Specifity_1=recall_score(y_test, y_pred, pos_label=0)
#Printing the results obtained on the screen
print(f"   Accuracy: {cla.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity_1}")
print(f"   Specifity: {Specifity_1}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada: {(y==0).sum()/y.shape[0]*Sensivity_1+(y==1).sum()/y.shape[0]*Specifity_1}")
print()

print ("   ---- KERAS ----")
# Instantiate model with 4 layers
model = keras.models.Sequential()
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))
#Improvement of the model
model.compile(loss="mean_squared_error",
optimizer="adam",
metrics=["binary_accuracy"])
#Trainning data with 500 epochs
model.fit(X_train, y_train,epochs=500)
#Using Keras on the test data.
y_pred = model.predict(X_test).round()
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {np.transpose(y_pred) }")
#Using different methods of evaluation
Sensivity_2=recall_score(y_test, y_pred)
Specifity_2=recall_score(y_test, y_pred, pos_label=0)
Accuracy_2 =accuracy_score(y_test, y_pred)
#Printing the results obtained on the screen
print(f"   Accuracy: {Accuracy_2}")
print(f"   Sensivity: {Sensivity_2}")
print(f"   Specifity: {Specifity_2}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada: {(y==0).sum()/y.shape[0]*Sensivity_2+(y==1).sum()/y.shape[0]*Specifity_2}")
print()
```

Ilustración 21: Octava parte "Ancianos andando": MLP (scikit-learn) + MLP(Keras).

6.2.1 Comparación entre los tres métodos

Una vez ejecutado el código y habiendo obtenido los resultados correspondientes, se introducen los datos en la siguiente tabla para una mayor compresión de los mismos.

ANCIANOS ANDANDO				
CALSIFICADOR	POMA	TUGT	RETRO_A	RETRO_B
RANDOM FOREST	0,795	0,768	0,249	0,662
MLP (Scikit-learn)	0,824	0,759	0,324	0,404
MLP (Keras)	0,735	0,777	0,308	0,562

Tabla 10: Resultados (métrica ponderada) del sistema predictor en ancianos únicamente andando.

Teniendo en cuenta que un clasificador ideal tiene una figura de mérito de 1.0 y tras examinar la tabla se puede concluir que no hay diferencias significativas entre ninguno de los tres clasificadores, pero sí entre las pruebas. POMA y TUGT son muy superiores a las dos definiciones basadas en la entrevista, sobre todo a la llamada Retro_A.

Podemos concluir que para la predicción de caídas según POMA y TUGT funciona extremadamente bien, de manera correcta para Retro_B y mal para su homóloga con dos o más caídas, en el sentido de que las señales de aceleración pueden predecir aceptablemente su resultado.

También se ha realizado un clasificador al azar para su comparación, implementándose con un bucle para todos los casos de test y generando un número aleatorio entre 0 y 1 (`np.random.rand`), siendo >= 0.5 con riesgo y <0.5 sin este. El resultado debía ser de 0.5 para la métrica ponderada, para ellos se ha procedido a realizar la media con todos los datos de dicha métrica.

```
0.4997725138968975
```

Ilustración 22: Resultado de la media ponderada del casificador al azar.

El código que se ha utilizado es el siguiente:

```
# Using numpy to convert to arrays
import numpy as np
# Using pandas to management and analysis of data structures
import pandas as pd
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Using Skicit-learn to ensemble of Decision Trees
from sklearn.metrics import precision_score, recall_score
# Using Scikit-learn to obtain a better classification metric for unbalanced data
from sklearn.metrics import f1_score
# Using Scikit-learn to obtain accuracy values
from sklearn.metrics import accuracy_score
```

Ilustración 23: Predicción al azar. 1ª parte.

```
#loop to work with all features
lista = []
for nn in range (1,6):
    dirname = "features/features_"+ str(nn)+"/"
    xname = dirname + "X_"+ str(nn)+ ".npy"
    for name in ["poma", "retro","retroB" , "tugt"]:
        yname = dirname + "y"+ name + "_"+ str(nn)+ ".npy"
        print ("Feature "+ str(nn)+ ": " + name)
        X = np.load(xname)
        y = np.load(yname)

        #Splitting the data into training and testing sets, being the size of test part 33%.
        #Setting the random state to 42 which means the results will be the same each time
        #I run the split for reproducible results.
        (X_train, X_test, y_train, y_test) = train_test_split(X, y,test_size=0.33,random_state=42)
        print ("   ---- RANDOM ----")
        N = y_test.shape[0]
        y_pred=np.random.rand(N)>0.5
        #Using different methods of evaluation
        Sensivity=recall_score(y_test, y_pred)
        Specifity=recall_score(y_test, y_pred, pos_label=0)
        Accuracy =accuracy_score(y_test, y_pred)
        Ponderada = ((y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity)
        lista.extend([Ponderada])
        #Printing the results obtained on the screen
        print(f"   Accuracy: {Accuracy}")
        print(f"   Sensivity: {Sensivity}")
        print(f"   Specifity: {Specifity}")
        print(f"   F1score: {f1_score(y_test, y_pred)}")
        print(f"   Ponderada: {Ponderada}")
        print()

print(lista)
mean = sum(lista)/len(lista)
print()
print(mean)
```

Ilustración 24: Predicción al azar. 2º parte.

En la siguiente tabla, se ve que son casi siempre cercanas a 0,5. Se desvía alguna métrica, pero puede deberse por casualidad de los números al azar y porque realmente tampoco hay tantos casos de una clase, sobre todo en RETRO, lo que puede hacer más fácil que salgan valores muy diferentes a 0.5.

		PREDICCIÓN AL AZAR			
		POMA	TUGT	RETRO_A	RETRO_B
Datos	No, clusters	Ponderada	Ponderada	Ponderada	Ponderada
features_1	2	0,447	0,441	0,493	0,777
features_2	4	0,619	0,422	0,439	0,445
features_3	8	0,430	0,551	0,477	0,599
features_4	16	0,474	0,396	0,535	0,479
features_5	32	0,481	0,544	0,531	0,472

Tabla 11: Predicción el azar.

Las prestaciones del clasificador original mejoran respecto a este, sobre todo en POMA y TUGT, que siempre sale > 0.5, pero respecto a las entrevistas (RETRO) es, al contrario, es decir, no se tiene mucha capacidad predictiva si intentamos asociar el resultado de la entrevista con las señales de aceleración.

7 COMPARACIÓN ENTRE EL ACELERÓMETRO EN LA VIDA DIARIA Y EN LA PRUEBA CONTROLADA

Viendo los resultados no se puede afirmar cual es el mejor con una rotunda respuesta. Dependerá de las circunstancias. Obviamente los resultados son mejores cuando los ancianos llevan el acelerómetro mientras andan. Ahora bien, ¿Es esta una diferencia significativa? ¿Supone el llevarlo en el día a día una mayor adherencia (al llevar siempre el móvil con uno mismo) que en la prueba controlada? ¿Tiene la capacidad los sujetos de realizar una prueba controlada habitualmente para un autodiagnóstico y sin introducir error? Si lo lleva continuamente, ¿podremos estimar en tiempo real del riesgo de caída ante cambios repentinos en el sujeto? ¿Se podría utilizar el acelerómetro del móvil o de pulseras inteligentes para la obtención de estos datos o tendrían que ser dispositivos especializados?

Todas estas preguntas tienen respuestas que dependen de las circunstancias y de los sujetos.

Si analizamos los datos de manera detenida calculando el porcentaje de diferencia entre los datos de los ancianos andando con los del día a día, se puede observar que el porcentaje medio de diferencia (resta de ambos datos entre el mayor de ellos por cien) es del 33%, por lo que supone una tercera parte, siendo un porcentaje a tener en cuenta.

PORCENTAJE DE DIFERENCIA				
CLASIFICADOR	POMA	TUGT	RETRO_A	RETRO_B
RANDOM FOREST	33%	24%	46%	52%
MLP(scikit-learn)	44%	12%	17%	3%
KERAS	23%	29%	11%	101%

Tabla 12:Porcentaje de diferencia en la comparación

No obstante, está claro que, si nos aseguramos de que los datos de aceleración se han obtenido en condiciones de andar, las prestaciones de los algoritmos mejoran considerablemente.

8 MEJORAS FUTURAS

Este proyecto lo podríamos denominar abierto, lo que significa que al tener código siempre se puede desarrollar nuevas ideas que lo mejoren. Se podría empezar por el perfeccionamiento de la parte del código que identifica que el sujeto está andando, pudiéndose resolver muchas de las preguntas del apartado anterior y llegando a reducir considerablemente el porcentaje de diferencia entre el acelerómetro en la vida diaria y el de solamente andando.

Otra parte sería la mejora en los clasificadores. Esta se podría llevar a cabo hilando más fino en los diferentes parámetros de cada uno, para eso se podría implementar con bucles que encuentre le parámetro que saca una métrica mejor.

Otra mejora se refiere al aumento del número de características implementadas. Se ha escogido un número reducido para acotar el tiempo del proyecto, pero el estudio de Lockart et al (2021) contiene muchas otras características extraídas de las ventanas de aceleración. También se puede intentar introducir los datos en bruto a los clasificadores de tipo Deep Learning, ya que muchas veces son capaces de detectar las características como salida de las capas ocultas, evitando la labor manual de elegirlas previamente. Esta es precisamente una de las ventajas principales de las técnicas de Deep Learning.

9 CONCLUSIONES

Tras la realización de dicho TFG y resolución de todos los problemas que han ido surgiendo, se considera que se ha conseguido el objetivo del mismo: estimar del riesgo de caídas usando acelerómetros y técnicas de Aprendizaje Automático.

Las conclusiones a las que se ha llegado con el desarrollo del proyecto han sido:

- El coste del desarrollo de dicho sistema predictor de caídas es nulo, por lo que la implementación y su puesta en marcha a nivel comercial sería mínimo.
- El software utilizado también es gratuito y con una amplia comunidad de usuarios utilizándolo por lo que hay una cantidad inmensa de información.
- Si se llevará a cabo la comercialización habría que trabajar en el entorno y en la visualización de los datos. Incluso con dichos resultados se podría trabajar mano a mano con personal médico para su posterior interpretación y prescripción médica.
- Si el acelerómetro es un smartwatch o smartphone, este recolectaría datos en un continuo del tiempo, sin generar modificaciones en los hábitos de los sujetos.
- La predicción del riesgo de caída es aceptable respecto de las pruebas clínicas, no así a las basadas en la entrevista.

Con todo lo comentado anteriormente, se podría decir que hay una alternativa real tecnológica a la predicción de riesgo de caída, ya que anteriormente estos test y pruebas clínicas eran laboriosas en cuanto a tiempo y personal, además de su inviabilidad para llevarlas a cabo de una forma continuada en el tiempo.

Personalmente, la realización de este proyecto me ha abierto un nuevo horizonte, ya que esta rama de la ingeniería casi la desconocía por completo. Además, ha sido un reto personal muy parecido a lo que puede ser la vida laboral, ya que te enfrentas a un proyecto que tienes que entregar en un tiempo concreto y con casi total libertad.

Además de rama de la inteligencia artificial, el proyecto te permite asentar todos los conocimientos que has ido adquiriendo durante la carrera, así como competencias a la hora de redactar y maquetar un proyecto.

10 REFERENCIAS

Datademia. (2022). Obtenido de "¿Qué es Python?": https://datademia.es/blog/que-es-python#:~:text=Python%20es%20un%20lenguaje%20de,y%20la%20reutilizaci%C3%B3n%20de%20c%C3%B3digo.

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media, Inc.,.

Heras, J. M. (18 de septiembre de 2020). *IA Artificial*. Obtenido de "Random Forest (Bosque Aleatorio): combinando árboles".

Herás, J. M. (9 de octubre de 2020). *IA Artificial*. Obtenido de "Precision, Recall, F1, Accuracy en clasificación": https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/#:~:text=Precision%20nos%20da%20la%20calidad,Recall%20en%20una%20sola%20medida

Lockart et al. (2021). Prediction of fall risk among community-dwelling older adults using a wearable system. *Scientific Reports*. doi:10.1038/s41598-021-00458-5

M. Jacobs y T.Fox. (2008). Using the Timed Up and Go Test/TUG Test to Predict Risk of Falls. *Assisted Living Consult*.

Na8. (29 de Mayo de 2018). *Aprende machine learning*. Obtenido de "Una sencilla Red Neuronal en Python con Keras y Tensorflow": https://www.aprendemachinelearning.com/una-sencilla-red-neuronal-en-python-con-keras-y-tensorflow/

nexusintegra. (2022). Obtenido de "big data vs inteligencia artificial": https://nexusintegra.io/es/big-data-vs-inteligencia-artificial/

OMS. (26 de Abril de 2021). Obtenido de "Caídas": https://www.who.int/es/news-room/fact-sheets/detail/falls#:~:text=Las%20ca%C3%ADdas%20son%20sucesos%20involuntarios,superficie%20firme%20que%20lo%20detenga.

Oracle. (s.f.). Obtenido de "¿Qué es la inteligencia artificial IA?": https://www.oracle.com/es/artificial-intelligence/what-is-ai/#:~:text=En%20t%C3%A9rminos%20simples%2C%20inteligencia%20artificial,de%20la%20informaci%C3%B3n%20que%20recopilan.

Rodrigo, J. A. (Agosto de 2020). *Ciencia de datos*. Obtenido de "Machine learning con Python y Scikit-learn": https://www.cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.html

SAS. (s.f.). Obtenido de "Deep Learning": https://www.sas.com/es_es/insights/analytics/deep-learning.html

SAS. (s.f.). Obtenido de "Big data": https://www.sas.com/es_es/insights/big-data/what-is-big-data.html

Silva, D. d. (31 de marzo de 2021). *Zendesk*. Obtenido de "Diferencia entre Machine Learning y Deep Learning": https://www.zendesk.com.mx/blog/machine-learning-deep-learning-diferencias/

Tinetti, M. (1986). Performance-oriented Assessment Mobility Problems in Elderly Patients. *JAGS*, 119-126.

Wikipedia. (s.f.). Obtenido de "Pandas (software)": https://es.wikipedia.org/wiki/Pandas_(software)

Wikipedia. (s.f.). Obtenido de "NumPy": https://es.wikipedia.org/wiki/NumPy#:~:text=NumPy%20(pronunciado%20%2F%CB%88n%CA%8C,nivel%20para%20operar%20con%20ellas.
Wikipedia. (s.f.). Obtenido de "TensorFlow": https://es.wikipedia.org/wiki/TensorFlow
Wikipedia. (s.f.). Obtenido de "Keras": https://es.wikipedia.org/wiki/Keras

ANEXO

ANEXO: CÓDIGO

1.1. Ancianos en la vida diaria

```
# In[1]:


# Using numpy to convert to arrays
import numpy as np
# Using pandas to management and analysis of data structures
import pandas as pd
#Using keras as High-level TensorFlow API for building and training deep
learning models
import tensorflow as tf
from tensorflow import keras
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Using Skicit-learn to ensemble of Decision Trees
from sklearn.ensemble import RandomForestClassifier
# Using Scikit-learn to obtain sensitivity and specificity values
from sklearn.metrics import precision_score, recall_score
# Using Scikit-learn to obtain a better classification metric for unbalanced
data
from sklearn.metrics import f1_score
# Using Scikit-learn to classifier with neural networks
from sklearn.neural_network import MLPClassifier
# Using Scikit-learn to obtain accuracy values
from sklearn.metrics import accuracy_score


# In[2]:


#loop to work with all features
for nn in range (1,6):
    dirname = "features/features_"+ str(nn)+"/"
    xname = dirname + "X_"+ str(nn)+ ".npy"
    for name in ["poma", "retro","retroB" , "tugt"]:
        yname = dirname + "y"+ name + "_"+ str(nn)+ ".npy"
        print ("Feature "+ str(nn)+ ": " + name)
        X = np.load(xname)
        y = np.load(yname)
        #Splitting the data into training and testing sets, being the size of
        test part 33%.
        #Setting the random state to 42 which means the results will be the
        same each time I run the split for reproducible results.
        (X_train, X_test, y_train, y_test) = train_test_split(X,
        y,test_size=0.33,random_state=42)

        print ("   ---- RAMDOM FOREST ----")
        # Instantiate model with 300 decision trees.
        Rf = RandomForestClassifier(random_state=42, n_estimators=300,
        class_weight = "balanced")
        # Trainning the model on training data.
        Rf.fit(X_train, y_train)
        #Using the forest's predict method on the test data.
        Y_pred = rf.predict(X_test)
        print (f"   TEST      : {y_test}")
        print(f"   PREDICCION: {y_pred}")
        #Using different methods of evaluation
        Sensivity = recall_score(y_test, y_pred)
        Specifity = recall_score(y_test, y_pred, pos_label=0)
```

```
#Printing the results obtained on the screen.
Print(f"   Accuracy: {rf.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity}")
print(f"   Specifity: {Specifity}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada:
{(y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity}")
print()

print ("   ---- MPL ----")
 # Instantiate model with 2500 max interactions
cla = MLPClassifier(max_iter=25000)
cla.fit(X_train, y_train)
#Using the MLPClassifier on the test data.
Y_pred=cla.predict( X_test)
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {y_pred}")
#Using different methods of evaluation
Sensivity_1=recall_score(y_test, y_pred)
Specifity_1=recall_score(y_test, y_pred, pos_label=0)
#Printing the results obtained on the screen
print(f"   Accuracy: {cla.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity_1}")
print(f"   Specifity: {Specifity_1}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada:
{(y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity}")
print()

print ("   ---- KERAS ----")
# Instantiate model with 4 layers
model = keras.models.Sequential()
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))
#Improvement of the model
model.compile(loss="mean_squared_error",
optimizer="adam",
metrics=["binary_accuracy"])
#Trainning data with 100 epochs
model.fit(X_train, y_train,epochs=100)
#Using Keras on the test data.
Y_pred = model.predict(X_test).round()
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {np.transpose(y_pred) }")
#Using different methods of evaluation
Sensivity_2=recall_score(y_test, y_pred)
Specifity_2=recall_score(y_test, y_pred, pos_label=0)
# Accuracy_2 =accuracy_score(y_test, y_pred)
#Printing the results obtained on the screen
print(f"   Accuracy: {Accuracy_2}")
print(f"   Sensivity: {Sensivity_2}")
print(f"   Specifity: {Specifity_2}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada:
{(y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity}")
print()
```

1.2. Ancianos en pruebas físicas controladas

```
# In[13]:


def allFeaturesFINAl(fname):

    acf = getLabelledAcelFilter(fname)
    pca = PCA()
    acf2 = pca.fit_transform(acf)
    X1=extractFeatures(acf2)

    return X1


# In[14]:


def allFeatures(subjects = None):
    """
    Si subjects es None (no se le pasa parametro) entonces saca el la
    lista de voluntarios
    Tambien se le puede pasar subjects = ["T001", "T002"], es decir
    una lista de voluntarios. Entonces solo analizara los voluntarios de
    la lista.
    """
    # IVAN
    nyq = FNYQ # For smartphone 50Hz/2
    voluntDic = getVoluntDic2('data/datosUsuarios.csv')
    rdir = 'data/'
    if(subjects is None):
        subjects = sorted(voluntDic.keys())
    #import ipdb;ipdb.set_trace()
    X = []
    ypoma = []
    ytugt = []
    yretro = []
    yretroB = []
    for subject in subjects:

        if subject == "T003":
            continue
        # Aqui: para cada individuo, y cada archivo andando. OJO algunos no
        tienen, nombre mal.
        # Leer archivo getLabelledAcelFilter
        if subject == "T006" or  subject == "T024" or  subject == "T026" or
        subject == "T027":
            fname=rdir+subject+"/"+subject +"andar1"+".log"
            print(fname)
            final= allFeaturesFINAl(fname)
            X.append(final)
            yretro.append(int(faller(voluntDic, subject, "retro")))
            yretroB.append(int(faller(voluntDic, subject, "retroB")))
            ypoma.append(int(faller(voluntDic, subject, "poma25")))
            ytugt.append(int(faller(voluntDic, subject, "tugt14")))
            continue

            if subject >= "T009" and  subject <="T013" or  subject == "T017"
            or  subject == "T018"  or  subject == "Z002" or subject == "Z001"
            or  subject == "T008"  :
            for nn in range (1,3):
                fname=rdir+subject+"/"+subject +"andar"+str(nn)+".log"
                print(fname)
                final= allFeaturesFINAl(fname)
                X.append(final)
```

```
                yretro.append(int(faller(voluntDic, subject, "retro")))
                yretroB.append(int(faller(voluntDic, subject, "retroB")))
                ypoma.append(int(faller(voluntDic, subject, "poma25")))
                ytugt.append(int(faller(voluntDic, subject, "tugt14")))
                continue
        else:
            for nn in range (1,4) :
                fname=rdir+subject+"/"+subject +"andar"+str(nn)+".log"
                print(fname)
                final= allFeaturesFINAl(fname)
                X.append(final)
                yretro.append(int(faller(voluntDic, subject, "retro")))
                yretroB.append(int(faller(voluntDic, subject, "retroB")))
                ypoma.append(int(faller(voluntDic, subject, "poma25")))
                ytugt.append(int(faller(voluntDic, subject, "tugt14")))

    # Pasamos de lista a array y las devolvemos

    X = np.vstack(X)
    yretro = np.array(yretro)
    yretroB = np.array(yretroB)
    ytugt = np.array(ytugt)
    ypoma = np.array(ypoma)
    # AQUI: guardar en ficheros si no se quiere repetir
    # cada vez
    return  X,yretro, yretroB, ytugt, ypoma


# In[15]:


allFeatures()


# In[40]:


DATOS=allFeatures()


# In[41]:


X = DATOS[0]

for nn in range (1,5):
    if nn == 1:
        print('Retro_A')
    elif nn == 2:
        print('Retro_B')
    if nn == 3:
        print('TUGT')
    elif nn == 4:
        print('POMA')

    y = DATOS[nn]

    #Splitting the data into training and testing sets, being the size of test
    part 33%.
    #Setting the random state to 42 which means the results will be the same
    each time I run the split for reproducible results.
    (X_train, X_test, y_train, y_test) = train_test_split(X,
    y,test_size=0.33,random_state=42)

    print ("   ---- RAMDOM FOREST ----")
```

```
# Instantiate model with 300 decision trees.
rf = RandomForestClassifier(random_state=42, n_estimators=300,
class_weight = "balanced")
# Trainning the model on training data.
rf.fit(X_train, y_train)
#Using the forest's predict method on the test data.
y_pred = rf.predict(X_test)
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {y_pred}")
#Using different methods of evaluation
Sensivity = recall_score(y_test, y_pred)
Specifity = recall_score(y_test, y_pred, pos_label=0)
#Printing the results obtained on the screen.
print(f"   Accuracy: {rf.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity}")
print(f"   Specifity: {Specifity}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada:
{(y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity}")
print()

print ("   ---- MPL ----")
# Instantiate model with 3000 max interactions
cla = MLPClassifier(max_iter=30000)
cla.fit(X_train, y_train)
#Using the MLPClassifier on the test data.
y_pred=cla.predict( X_test)
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {y_pred}")
#Using different methods of evaluation
Sensivity_1=recall_score(y_test, y_pred)
Specifity_1=recall_score(y_test, y_pred, pos_label=0)
#Printing the results obtained on the screen
print(f"   Accuracy: {cla.score(X_test, y_test)}")
print(f"   Sensivity: {Sensivity_1}")
print(f"   Specifity: {Specifity_1}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada:
{(y==0).sum()/y.shape[0]*Sensivity_1+(y==1).sum()/y.shape[0]*Specifity_1}"
)
print()

print ("   ---- KERAS ----")
# Instantiate model with 4 layers
model = keras.models.Sequential()
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))
#Improvement of the model
model.compile(loss="mean_squared_error",
optimizer="adam",
metrics=["binary_accuracy"])
#Trainning data with 500 epochs
model.fit(X_train, y_train,epochs=500)
#Using Keras on the test data.
y_pred = model.predict(X_test).round()
print (f"   TEST      : {y_test}")
print(f"   PREDICCION: {np.transpose(y_pred) }")
#Using different methods of evaluation
Sensivity_2=recall_score(y_test, y_pred)
Specifity_2=recall_score(y_test, y_pred, pos_label=0)
Accuracy_2 =accuracy_score(y_test, y_pred)
#Printing the results obtained on the screen
print(f"   Accuracy: {Accuracy_2}")
```

```
print(f"   Sensivity:{Sensivity_2}")
print(f"   Specifity: {Specifity_2}")
print(f"   F1score: {f1_score(y_test, y_pred)}")
print(f"   Ponderada:
{(y==0).sum()/y.shape[0]*Sensivity_2+(y==1).sum()/y.shape[0]*Specifity_2}"
)
print()
```

1.3. Clasificador aleatorio

```
# In[1]:


# Using numpy to convert to arrays
import numpy as np
# Using pandas to management and analysis of data structures
import pandas as pd
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Using Skicit-learn to ensemble of Decision Trees
from sklearn.metrics import precision_score, recall_score
# Using Scikit-learn to obtain a better classification metric for unbalanced
data
from sklearn.metrics import f1_score
# Using Scikit-learn to obtain accuracy values
from sklearn.metrics import accuracy_score


# In[6]:


#loop to work with all features
lista = []
for nn in range (1,6):
    dirname = "features/features_"+ str(nn)+"/"
    xname = dirname + "X_"+ str(nn)+ ".npy"
    for name in ["poma", "retro","retroB" , "tugt"]:
        yname = dirname + "y"+ name + "_"+ str(nn)+ ".npy"
        print ("Feature "+ str(nn)+ ": " + name)
        X = np.load(xname)
        y = np.load(yname)

        #Splitting the data into training and testing sets, being the size of
        test part 33%.
        #Setting the random state to 42 which means the results will be the
        same each time
        #I run the split for reproducible results.
        (X_train, X_test, y_train, y_test) = train_test_split(X,
        y,test_size=0.33,random_state=42)
        print ("   ---- RANDOM ----")
        N = y_test.shape[0]
        y_pred=np.random.rand(N)>0.5
        #Using different methods of evaluation
        Sensivity=recall_score(y_test, y_pred)
        Specifity=recall_score(y_test, y_pred, pos_label=0)
        Accuracy =accuracy_score(y_test, y_pred)
        Ponderada =
        ((y==0).sum()/y.shape[0]*Sensivity+(y==1).sum()/y.shape[0]*Specifity)
        lista.extend([Ponderada])
        #Printing the results obtained on the screen
        print(f"   Accuracy: {Accuracy}")
        print(f"   Sensivity: {Sensivity}")
        print(f"   Specifity: {Specifity}")
        print(f"   F1score: {f1_score(y_test, y_pred)}")
        print(f"   Ponderada: {Ponderada}")
        print()

print(lista)
mean = sum(lista)/len(lista)
print()
print(mean)
```

1.4. Generación de gráficas

```
def allFeatures(n_clusters, subjects = None):
    nyq = FNYQ # For smartphone 50Hz/2
    voluntDic = getVoluntDic2('data/datosUsuarios.csv')
    rdir = 'data/'
    print('OJO features low pass 8 Hz')
    print('OJO que las features se igualen en cuanto a valores (normalizar)')
    print('Esto por el tema de kmeans')
    print('OJO tb a la longitud de la segnal. Si disminuyo wsize')
    print('y quiero llegar a 0.25 Hz en los analisis en freq hay problemas')
    print('porque casi no tengo resolucion y salen indices negativos al buscar
    anchura')
    coefs = scipy.signal.butter(5, 8.0/nyq, 'lowpass')
    b1 = coefs[0]
    a1 = coefs[1]
    wsize = 512 # 512/50 s, 512 = 2**9
    if(subjects is None):
        subjects = sorted(voluntDic.keys())
    #import ipdb;ipdb.set_trace()
    X = []
    ypoma = []
    ytugt = []
    yretro = []
    yretroB = []
    listanumeros = []
    i=0
    for subject in subjects:
        print(subject)
        nwalk = 0.0
        nwin = 0.0
        path=rdir+subject+'/'
        segs, wtimes, dum1, dum2=getSegmentsPath(path)
        Xkm = [] # For kmeans
        for ac in segs:
            acf = scipy.signal.filtfilt(b1, a1, ac, axis = 0,
            padtype='constant')
            acft = np.sqrt( (acf**2).sum(1))
            ##
            wints = np.split( acft , range(wsize, acft.shape[0], wsize))
            wints.pop(-1)
            wins = np.split( acf , range(wsize, acf.shape[0], wsize))
            wins.pop(-1)
            ##
            nwin += len(wins)

            for win, wint in zip(wins, wints):
                if(isWalking(wint)):
                    nwalk += 1.0

                    #SABER CUANTAS HAY
                    if i == 0:
                        for nn in range (1,6):
                            numero = random.randint(1, 12000)
                            listanumeros.extend([numero])

                    for element in listanumeros:
                        if i == element:
                            plt.figure(1)
```

```
                        plt.plot(win)
                        plt.xlabel("Tiempo(s)")
                        plt.ylabel("Ace(m/s^2)")
                        plt.savefig("PlotGeneratedUsingMatplotlib"+
                        str(i)+".png")
                        plt.close(1)

                    i+=1

                    pca = PCA()
                    win2 = pca.fit_transform(win)
                    feat = extractFeatures(win2, wsize)
                    if(feat is not None):
                        Xkm.append(feat.reshape(1, feat.shape[0]))
                    # A este nivel win es una array 512x3, 512 muestras
                    # de los tres ejes, mientras que wint es la aceleracion
                    # total (modulo del vector) de 512x3
                    # Se supone que esta andando. win2 tras el PCA

        # Esto seria el porcentaje de tiempo andando dentro de la actividad
        print(nwalk/nwin)
        Xkm = np.vstack(Xkm)
        # I have decided to scale features
        scaler = StandardScaler()
        X2 = scaler.fit_transform(Xkm)
        # Then cluster
        km = KMeans(n_clusters=n_clusters, n_init=100, max_iter=20000)
        km.fit(X2)
        # Then add but undo scale to recover original units
        X.append(scaler.inverse_transform(km.cluster_centers_))
        yretro.extend(n_clusters*[ int( faller(voluntDic, subject, 'retro'))])
        yretroB.extend(n_clusters*[ int( faller(voluntDic, subject,
        'retroB'))])
        ytugt.extend(n_clusters*[ int( faller(voluntDic, subject, 'tugt14'))])
        ypoma.extend(n_clusters*[ int( faller(voluntDic, subject, 'poma25'))])

X = np.vstack(X)
yretro = np.array(yretro)
yretroB = np.array(yretroB)
ytugt = np.array(ytugt)
ypoma = np.array(ypoma)
return X, yretro, yretroB, ytugt, ypoma
```