Clasificación de actividades físicas basada en mediciones del acelerómetro de un dispositivo móvil

Raúl Arias Lévano
Universidad Nacional de Ingeniería
Lima, Perú
raul.arias.l@uni.pe

Milton Palacin Grijalva
Universidad Nacional de Ingeniería
Lima, Perú
mpalacing@uni.pe

Bernick Salvador Rosas
Universidad Nacional de Ingeniería
Lima, Perú
bsalvador.ro@uni.pe

Miguel Paz Arcelles
Universidad Nacional de Ingeniería
Lima, Perú
mpaza@uni.pe

***Resumen*—En el presente trabajo final del curso se muestra un procedimiento de clasificación de cinco actividades físicas (caminando, sentado, saltando, corriendo, subiendo) utilizando la señal proporcionada por el acelerómetro de un dispositvo móvil (celular portable). Se mostrarán los pasos de recolección de datos, análisis, procesamiento y transformación para la creación de un dataset, entrenamiento de un modelo de clasificación (de redes neuronales), la validación del modelo y, luego de lograr la construcción del mismo, la utilización de dicho modelo en una aplicación móvil. Para lograr este propósito, se recolectarán datos de las cinco actividades mencionadas líneas arriba para 2 rangos de edad (adulto y joven) y dos géneros (varón y mujer), resultando así un total de 1085 muestras originales.**

***Palabras Clave*—Acelerómetro, FFT, Redes Neuronales, Dataset, Machine Learning, WebSocket, Móvil**

I. Introducción

La pandemia ocasionada por la COVID-19 ha hecho que el mundo comience una nueva normalidad y que, como parte de esta, muchas personas tengan que adaptarse al trabajo remoto, y como consecuencia tengan un comportamiento sedentario llevando a que personas con poca actividad física, agraven aun más su situación. Existen diversas aplicaciones móviles que identifican las actividades físicas realizadas por sus portadores haciendo uso de diferentes sensores, como la postura [1]. Con el objetivo de poner en práctica los conocimientos adquiridos en el curso, en este trabajo se desarrollará una aplicación móvil para celulares con sistema operativo Android para clasificar la actividad que esté realizando su portador. Para lograr esto, se desarrollará un modelo de solución (modelo de una arquitectura de solución), que va desde la recopilación inicial de datos para entrenar un modelo de Machine, hasta el uso de este modelo en tiempo real.

Se desarrollará un aplicación que recopile datos de la señal del acelerómetro del dispositivo. Las señales serán tomadas a una frecuencia de **50 Hz** por una ventana de tiempo de **5 s**, que conformarán una muestra. Estas muestras luego serán procesados para armar el dataset que será utilizado para la creación del modelo, pasando desde el el aumento de datos, filtrado, extracción de características, tratamiento de valores atípicos, hasta la normalización de los mismos. Con el dataset preparado, se procederá a entrenar un modelo de clasificación basado en redes neuronales.

Objetivos

Los objetivos principales de este trabajo se pueden listar de la siguiente manera:

- Construcción de una aplicación móvil en Android para la recopilación de datos del acelerómetro (formulario de etiquetado).
- Contrucción de un API[1] de WebSocket[2] con Python para recepcionar los datos recolectados en tiempo real por una aplicación en el dispositivo móvil.
- Aumento y transformación los datos recopilados y crear un dataset.
- Entrenamiento de un modelo de clasificación.
- Lograr un porcentaje de precisión alto tanto en el entrenamiento con en el uso del modelo.
- Construcción de una aplicación móvil en Android para la monitorización de la actividad física realizada por el portador.
- Construcción de un API de WebsSocket para que pueda ser consultado en tiempo real por una aplicación en el dispositivo móvil.

II. Antecedentes

Hayes y colegas llevaron a cabo una revisión sistemática de literatura (Web of Science, PubMed, SPORTDiscus, PsycINFO y CINAHL) acerca de investigaciones publicadas entre el 2000 al 2018 sobre la actividad física (PA) y el comportamiento sedentario (SB) en la transición de la adolescencia a la adultez. Encontraron que solo en 3 de los 16 artículos revisados se utilizaron acelerómetros como medición del PA y/o del SB, en la mayoría de los estudios restantes, estas mediciones fueron llevadas a cabo principalmente mediante cuestionarios o encuestas [2].

Entre otras investigaciones anteriores que se llevaron a a cabo donde emplearon acelerómetros como parte de sus experimentos, estas estuvieron orientadas a determinar los cambios de

[1]API: pos sus siglas en ingles, Interfaz de Progrmación de Aplicaciones

[2]WebSocket: es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. La ventaja de este intercambio es que se accede de forma más rápida a los datos. En concreto, WebSocket permite una comunicación directa entre una aplicación web y un servidor WebSocket. En otras palabras: la web que se solicita se muestra en tiempo real.

comportamiento sedentario a través de las etapas naturales de la vida humana con el objetivo de diseñar intervenciones exitosas de cambio de estilo de vida [3], encontrar patrones y determinantes que midan objetivamente la actividad física de moderada a vigorosa (MVPA) en adultos jóvenes [4], e identificar factores individuales, sociales y ambientales asociados con la MVPA en mujeres jóvenes de entre 14 a 23 años [5]. Ceron y colegas desarrollaron un sistema móvil (Android Mobile App) conformado por celulares, relojes inteligentes y dispositivos de geolocalización BLE (Bluetooth Low Energy) de Estimote, para clasificar 6 comportamientos sedentarios ('viendo TV sentado', 'viendo TV reclinado', 'almorzando/desayunando/cenando', 'utilizando un computador', 'manejando un carro' y 'siendo transportado por un carro') de un grupo de 15 personas (8 varones y 7 mujeres de entre 25 a 87 años) basado en datos de acelerómetros e inclinómetros y utilizando técnicas de Machine Learning de clasificación supervisada [6].

III. Modelo de Solución (Arquitectura de Solución)

El diseño de la arquitectura de la solución se realizó en base a la metodología CRISP-DM. A continuación se detalla los pasos realizados:

- Definición de necesidad del caso de estudio: el caso de estudio abarca el siguiente problema "Debido al trabajo remoto ocasionado por las restricciones de la pandemia COVID-19, las personas tienen un comportamiento más sedentario. Para prevenir enfermedades asociadas al sedentarismo (e.g. cardiovasculares) es recomendable para las personas adultas realizar ciertos tipos de actividades físicas durante por lo menos 150 minutos (2 horas y media) de actividad aeróbica moderada o 75 minutos (1 hora y cuarto) de actividad aeróbica intensa por semana. Para lograr este objetivo, un primer paso es clasificar las actividades físicas de las personas para ello usaremos las señales recolectadas del acelerómetro de los dispositivos móviles". Los objetivos se establecieron en la sección I.
- Estudio y comprensión de los datos: se utilizó la Aplicación de Recolección en los dispositivos móviles para recolectar datos de las señal del acelerómetro. Esta herramienta de recolección permitió almecenar datos de manera "online" y "offline" en archivos CSV. Los datos recolectados corresponden a los valores de las acelaraciones, que incluyen la gravedad, en el sistema de coordenadas X, Y y Z en relación a la posición del del acelerómetro del dispositivo móvil (ver fig. 1).
- Análisis de los datos y selección de características: con la ayuda del software Jupyter se generó un cuaderno de trabajo que permitió realizar el procesamiento de los datos (descrito en la sección V). En general se realiza todas la tareas necesarias para limpiar, seleccionar y formatear la data para generar el dataset de trabajo.
- Modelado: los pesos de modelo entrenado son guardados en un archivos portable. Esto permitirá más adelante utilizarlo en la consulta en línea.
- Evaluación (obtención de resultados): para generar los resultados se utilizó la Aplicación de Monitoreo en los dispositivos móviles.
- Despliegue (puesta en producción): actividad que no es parte de alcance del presente trabajo. Sin embargo, se ha dejado listo todos los librerias y binarios necesarios para su despliegue en servidores de producción.

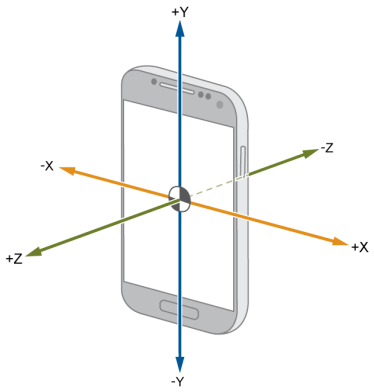

Figura 1. Sistema de coordenadas del acelerómetro de un dispositivo móvil

La arquitectura de la solución (ver fig. 2) está compuesto por los siguientes componentes:

A. Aplicación de Recolección

Se desarrollo un applicativo móvil en Android (ver fig.3) que permite recolectar los datos de la señal emitida por el acelerómetro incorporado en el dispositivo. Se elaboró un "Formulario de Etiquetado" para ingresar información de configuración antes de realizar el experimento (el experimiento consiste en realizar la accción seleccionada). A continuación la explicación de los valores de cada campo:

- Actividad (Estado): lista con los valores *NINGUNO, SENTADO, PARADO, ECHADO, CAMINANDO, SALTANDO, entre otros*.
- Sexo: con las opciones de *VARÓN o MUJER*
- Edad: con las opciones de *JOVEN o ADULTO*
- Tiempo: por defecto tiene el valor de *5 seg.* (ventana de tiempo para todos los experimentos).
- Frecuencia: por defecto tiene el valor de *50Hz* (50 mediciones/captura de señal por segundo).
- Ip/Puerto: IP o host con el puerto del servidor de WebSocket. En caso no existe conexión con el servidor se generá un archivo en el directorio *download* del móvil.
- Nombre: nombre de referencia para generar archivos de los experimentos en el servidor.

Se utiliza la librería *com.neovisionaries:nv-websocket-client:2.13* para la conexión con el servidor de WebSocket.

B. API de WebSocket de Recolección

El API de Websocket se desarrollo con ayuda de las siguientes librerías de python: *asyncio* y *websockets*. El formato de mensaje entre el móvil y el servidor es: "MM,dd,HH,mm,ss,sss,**x,y,z**,actividad,sexo,edad,nombre".

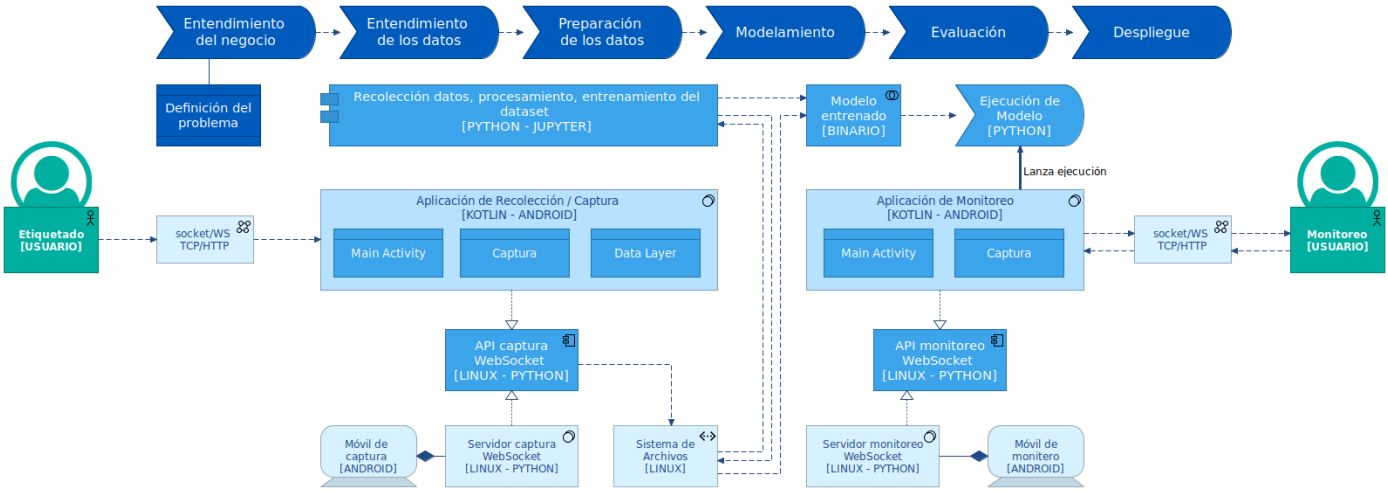

Figura 2. Arquitectura de la Solución

Figura 3. Aplicación de Recolección

Cuando la aplicación móvil inicia la captura, despúes de **3 s**, empieza a enviar mensajes a la frecuencia de muestreo (velocidad) de **50 Hz**, es decir envía un mensaje cada **20 ms**. En el mismo lapso de tiempo se guarda los datos dentro de un archivo CSV con nombre: *"nombre_actividad_sexo_edad_fechahorminseg.csv"*.

C. Modelo Entrenado

Luego de realizar los experimentos y recopilar las muestras se procedió a realizar el procesamiento de los datos (ver las siguientes secciones). Todo el trabajo de codificación del modelo se desarrollo en la plataforma Jupyter-Lab/Collab con base en python. Un componente adicional es el archivo con el modelo optimizado de clasificación. El archivo con los pesos pre-entrenado del modelo *MLPClassifier* de *sklearn.neural_network* tiene el nombre de *modelo_deteccion.uni*.

D. Aplicación de Monitoreo

Se desarrollo un applicativo móvil en Android (ver fig.4) que permita predecir en que estado se encuentra la persona. Los datos de la señal emitida por el acelerómetro, incorporado en el dispositivo, son enviadas de manera continua después de iniciar la monitorización al servidor de WebSocket. Para iniciar el monitoreo se configura el host o el IP con el puerto del servidor de WebSocket. Cuando la conexión con el servidor no se realiza o existe algún tipo de error no se monitorizará el estado. El envío de los mensajes con las coordenas del acelerómetro se realiza a la misma frecuencia de muestreo (velocidad) de recolección **50 Hz**. A diferencia de la aplicación de recolección, en esta aplicación se recibe un mensaje de respuesta del servidor de WebSocket con el estado "predecido", que de manera previa fue evaluado por el modelo pre-entrenado de ML.

E. API de WebSocket de Monitoreo

El formato de mensaje entre el móvil y el servidor es: "**x,y,z**,GUID", donde **GUID** es un identificador único de monitoreo, el cuál permite verificar la pertenencia del mensaje de envío y respuesta del portador del móvil. Cuando la aplicación móvil inicia el monitoreo, despúes de **3 s**, empieza a enviar mensajes a la frecuencia de muestreo (velocidad) de **50 Hz** de manera constantes. En la aplicación del servidor, desarrollado en python, con el mensaje recibido de las coordenas x, y, z

Figura 4. Aplicación de Monitoreo

se genera un arreglo de hasta 100 registros (dos segundos de capturas de señal), que corresponde a la misma cantidad de registros de una muestra utilizada después de la segmentación previo a la generación del dataset, luego se procede a generar los mismos atributos que el modelo pre-entranado para luego realizar la predicción.

F. Hardware y Software

- Las aplicaciones móviles desarrolladas se ejecutan en teléfonos inteligentes con sistema operativo Android $\geq$ 16, API $\geq$ 8.0.0. Como mínimo los dispositivos deben tener integrado un acelerómetro.
- La frecuencia de muestreo es 50 Hz. Los dispositivos móviles, dependiendo de la marca y modelo, tienen en promedio 50 Hz y algunos denominados de "alta gama" tiene la capacidad de variar entre 5 Hz y 500 Hz.
- Para el despliegue de los servidores WebSocket solo se utiliza el protocolo HTTP/TCP en el puerto 5000. El servidor que da soporte a los servicios es un Linux Ubuntu 20.10 con python 3.9.
- Para el desarrollo de las aplicaciones móviles se utilizó el IDE "Android Studio 4.1.2" y el lenguaje de programación "Koltlin".
- Para la construcción de los modelos de ML se utilizó JupyterLab, Collab, Python 3.7-3.9 y librerías conocidas de ML.
- Los equipos físicos utilizados fueron tres (3) Laptops y cinco (5) teléfonos inteligentes Samsumg Galaxy, (2) Motorola y ZTE.

IV. Recopilación de Datos

Para el proceso de recolección se utilizó el aplicativo móvil y el servidor de Websocket descrito en la sección anterior. Cada experimento ejecutado consistió en realizar una actividad (estado) por tipo de género y edad. En total se ejecutarón un promedio 50 experimentos, cada factor configura una muestra. En la tabla I se muestra los experimentos realizados:

Persona	Estado	Sexo	Edad	#Experimentos
Emperatriz	Sentado	Mujer	Adulto	55
Emperatriz	Saltando	Mujer	Adulto	50
Emperatriz	Caminando	Mujer	Adulto	52
Emperatriz	Subiendo	Mujer	Adulto	53
Emperatriz	Corriendo	Mujer	Adulto	71
Milca	Sentado	Mujer	Joven	55
Milca	Saltando	Mujer	Joven	53
Milca	Caminando	Mujer	Joven	51
Milca	Subiendo	Mujer	Joven	50
Milca	Corriendo	Mujer	Joven	63
Milton	Sentado	Varon	Adulto	50
Milton	Saltando	Varon	Adulto	50
Milton	Caminando	Varon	Adulto	53
Milton	Subiendo	Varon	Adulto	51
Milton	Corriendo	Varon	Adulto	64
Raul	Sentado	Varon	Joven	51
Raul	Saltando	Varon	Joven	52
Raul	Caminando	Varon	Joven	50
Raul	Subiendo	Varon	Joven	51
Raul	Corriendo	Varon	Joven	60
			TOTAL	**1085**

Tabla I
Experimentos realizados

De acuerdo a lo expuesto en la sección anterior, la recopilación de los datos fue realizado a 50 Hz (50 ciclos por segundo) duracte cinco segundos, logrando obtener 250 observaciones o registros de las coordenadas de aceleración x, y, z del acelerómetro que se encuentra en el dispositivo móvil utilizado para la captura. Un conjunto representativo de las señales de actividad del acelerómetro se pueden visualizar en las figuras 5, 6 y 7

V. Procesamiento de los Datos

A. Aumento de datos

En el proceso de recopilación de datos se han obtenido 1085 muestras, almacenadas en archivos con formato CSV. Para realizar la generación de más muestras se hizo el siguiente procedimiento: cada muestra contiene datos de 5000 ms (5 segundos) y a partir de cada una de estas se generaron 3 nuevas muestras, cada una de 2000 ms con un traslape de 500 ms. Como la frecuencia de muestreo de es 50 Hz, tenemos que cada muestra contiene 250 registros, las 3 nuevas muestras contienen:

- Segmento 1: registros del 1 al 100 (0 ms - 2000 ms)
- Segmento 2: registros del 75 al 175 (1500 ms - 3500 ms)
- Segmento 3: registros del 150 al 250 (3000 ms - 5000 ms)

Con un overlap de 25 registros (500 ms). Con este procedimiento, el número total de muestras fue aumentado a 3255

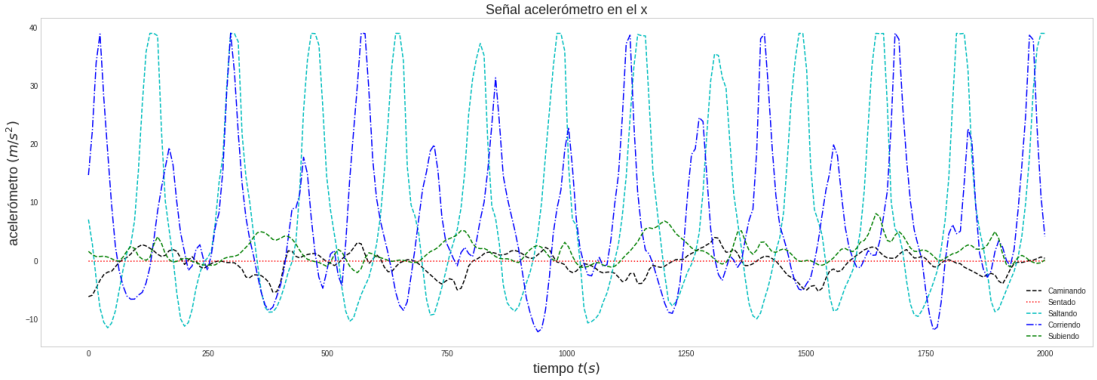

Figura 5. Muestra de la señal de aceleración de los cinco estados en la eje X

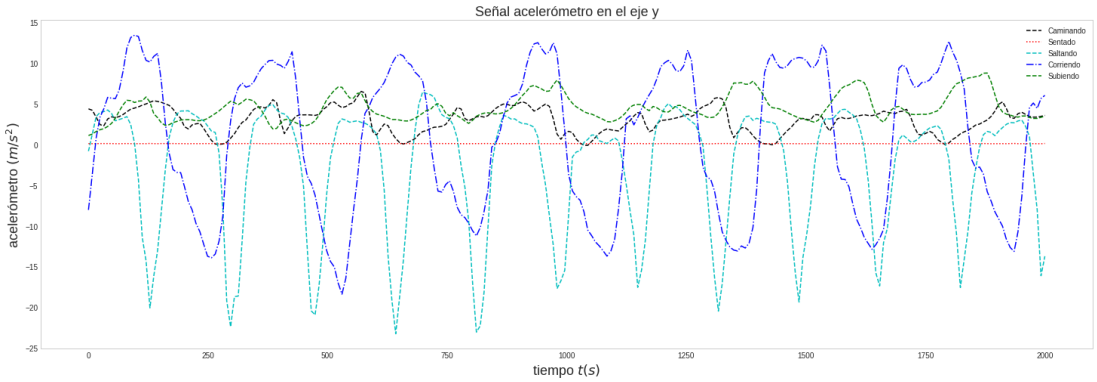

Figura 6. Muestra de la señal de aceleración de los cinco estados en la eje Y

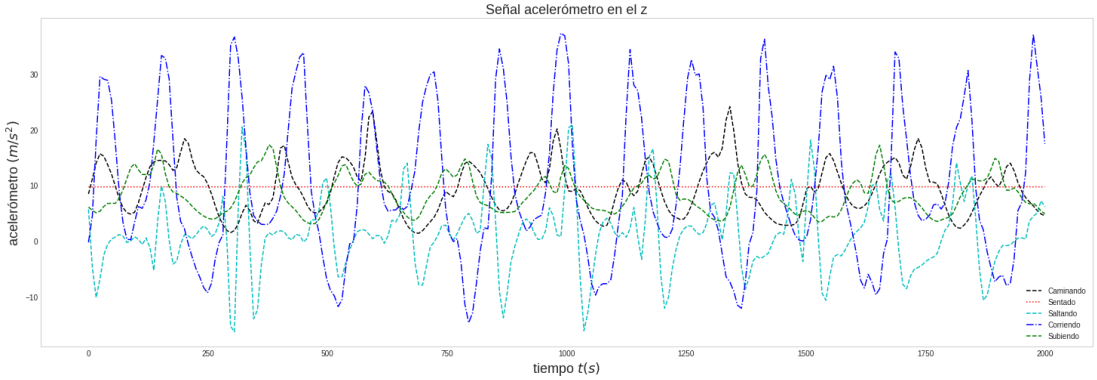

Figura 7. Muestra de la señal de aceleración de los cinco estados en la eje Z

que serán utilizadas para el entrenamiento y prueba del modelo que se utilizará.

Figura 8. Segmentación de las muestras

En el listado 1 se muestran las funciones a utilizar para el segmentado de las muestras originales para el aumento de datos.

```
def sampleSegmentation(data,
                       nSamplesInSegment,
                       nOverlap):
    segments = []
    nSamples = len(data)
    for i in range(0, nSamples, nSamplesInSegment-
        nOverlap):
        if i + nSamplesInSegment > nSamples:
            break
        segments.append(data[slice(i, i+
            nSamplesInSegment)])
    return segments


```

```
def generateSamples(fileName, dir_segmentado):
    numSamples = 100
    numOverlap = 25
    df = pd.read_csv(fileName)
    segments = sampleSegmentation(df, numSamples,
        numOverlap)
    for i in range(len(segments)):
        f_path_parts=fileName.split("/")
        f_name, f_ext=f_path_parts[-1].split(".")
        segment_file_name=dir_segmentado + f_name
            + "_" + str(i) + "." + f_ext
        if os.path.exists(segment_file_name):
            os.remove(segment_file_name)
        segments[i].to_csv(segment_file_name)
```

Listing 1. Funciones de segmentación de muestras

En el listado 2 se realiza la generación de nuevas muestras.

```
dir_original_data = 'data_original/' # carpeta con
     todos las muestras (csv)
dir_segmentos = 'segmentado/' # carpeta donde se
    almacenaran los nuevos csv creados

# Generando toda las muestras (3 por cada una)
data_original = os.listdir(dir_original_data)
for mfile in data_original:
    complete_name = dir_original_data + mfile
    if not os.path.isfile(complete_name):
        continue
    generateSamples(complete_name, dir_segmentos)
```

Listing 2. Segmentación de muestras

B. Filtrado

Para la construcción de un modelo de clasificación debemos contar con un dataset que contenga las principales características de los datos que acabamos de generar. Para tal fin, se deben transformar los datos haciendo uso de herramientas estadísticas, así como de series de tiempo como el centroide espectral (usado en [7] por Singha et al.), muy utilizado en el procesamiento de audios .

Sin embargo, antes de empezar a extraer características, aplicaremos un filtro a todas y cada una de las muestras. Para este trabajo hemos usado un filtro gaussiano para ”suavizar” la serie de tiempo. En la figura 9 se muestra el gráfico de una muestra tomada de una mujer joven realizando la actividad ”caminando”.

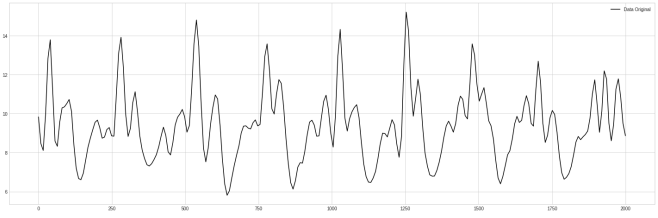

Figura 9. Señal original

A todas las muestras vamos a aplicarle un filtro gaussiando con $\sigma = 1.0$. Al aplicar este filtro a la señal anterior, obtenemos el resultado mostrado en la figura 10.

En la figura 11 se muestra la superposición de la señal original y la señal filtrada de color negro y rojo, respectivamente.

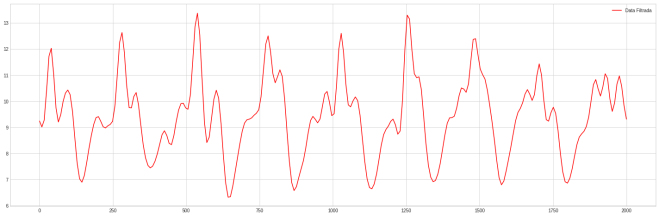

Figura 10. Señal filtrada

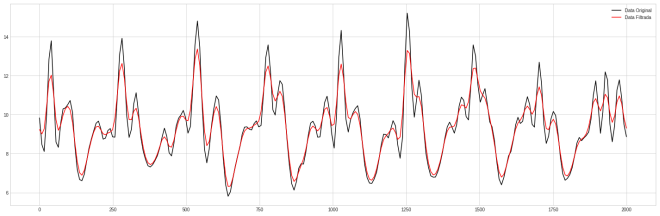

Figura 11. Superposición señal original y señal filtrada

C. Extracción de Características

Ahora que ya se tienen filtrados los datos que se van a utilizar, comenzamos con la extracción de características. En este proceso, se utilizarán los datos de las aceleraciones axiales (x, y, z) leídos del acelerómetro (filtradas) y los datos obtenidos al aplicar la Transformada Rápida de Fourier (FFT) a estos datos axiales.

- Media de datos axiales. Las primeras tres características que se extraen son las medias de las acelraciones axiales en los 3 ejes (x, y, z) que nos proporciona el acelerómetro.
- Media de datos FFT. Las medias los datos obtenidos al aplicar la Transformada Rápida de Fourier a cada una de las aceleraciones axiales (media del módulo, ya que son números complejos).
- Mediana de datos axiales.
- Mediana de datos FFT.
- Centroide Espectral: El centroide espectral es una medida utilizada para caracterizar un espectro e indica dónde está su centro de masa. El centroide espectral de la muestra para los tres ejes usando los valores de FFT como pesos viene dado por [7]:

$$CE = \sum_{i=1}^{n} (x_i * f_i)/n \tag{1}$$

donde:
x_i: señal del acelerómetro
f_i: señal x_i transformada con FFT
n: tamaño de la muestra

En el listado 3 de código se muestra la implemetación en python.

```
def spectral_centroid(x, spect_x):
    magnitudes = np.abs(spect_x)
    length = len(x)
    freqs = np.abs(np.fft.fftfreq(length)
                   [:length//2 + 1])
```

```
    magnitudes = magnitudes[:length//2 + 1]
    return np.sum(magnitudes*freqs) / np.sum(
        magnitudes)
```

Listing 3. Centroide del Espectro

- Magnitud: La magnitud se define como el promedio de la raíz cuadrada de los cuadrados datos triaxiales y se calcula de la siguiente manera [7]:

$$Mag = (\sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2 + z_i^2})/n \quad (2)$$

donde:
x_i, y_i, z_i: señal en el eje correspondiente del acelerómetro
n: tamaño de la muestra
En el listado 4 de código se muestra la implemetación en python.

```
def magnitude(x, y, z):
    l = len(x)
    sum = 0
    for i in range(l):
        sum = sum + math.sqrt(x[i]**2 + y[i]**2
            + z[i]**2)
    return sum/l
```

Listing 4. Magnitude

- Diferencia promedio de la media: La diferencia absoluta de la media de la ventana para cada eje (dominio del tiempo) se calcula de la siguiente manera [7]:

$$ADM = Avg(|x_i - x_{media}|) \quad (3)$$

donde:
x_i: señal del acelerómetro
x_{media}: la magnitud definida en la ecuación 2

VI. Creación del Dataset

Ya se identificaron las características que se van a extraer desde las muestras. A continuación se muestra las 24 columnas que contendrá el dataset:

- Media de la aceleración en el eje X.
- Media de la aceleración en el eje Y.
- Media de la aceleración en el eje Z.
- Media de la FFT en el eje X.
- Media de la FFT en el eje Y.
- Media de la FFT en el eje Z.
- Mediana de la aceleración en el eje X.
- Mediana de la aceleración en el eje Y.
- Mediana de la aceleración en el eje Z.
- Mediana de la FFT en el eje X.
- Mediana de la FFT en el eje Y.
- Mediana de la FFT en el eje Z.
- Centroide en el eje X.
- Centroide en el eje Y.
- Centroide en el eje Z.
- Magnitude de las aceleraciones en los 3 ejes.
- ADM de las aceleraciones en el eje X.
- ADM de las aceleraciones en el eje Y.
- ADM de las aceleraciones en el eje Z.
- Magnitude de las FFT en los 3 ejes.
- ADM de la FFT en el eje X.
- ADM de la FFT en el eje Y.
- ADM de la FFT en el eje Z.
- Clase (Sentado, caminando, saltando, corriendo, subiendo)

```
data_contents = os.listdir(dir_segmentos)
row_id = 0
for mfile in data_contents:
    complete_name = dir_segmentos + mfile

    if not os.path.isfile(complete_name):
        continue
    df_content = pd.read_csv(complete_name)

    lb_x = gaussian_filter1d(df_content.x, 1)
    lb_y = gaussian_filter1d(df_content.y, 1)
    lb_z = gaussian_filter1d(df_content.z, 1)

    spect_x = abs(fft.fft(lb_x))
    spect_y = abs(fft.fft(lb_y))
    spect_z = abs(fft.fft(lb_z))
    data_class = mfile.split("_")
    class_label = data_class[1]

    sc_x = spectral_centroid(lb_x, spect_x)
    sc_y = spectral_centroid(lb_y, spect_y)
    sc_z = spectral_centroid(lb_z, spect_z)

    magnit = magnitude(lb_x, lb_y, lb_z)
    adm_x = np.average(abs(lb_x - magnit))
    adm_y = np.average(abs(lb_y - magnit))
    adm_z = np.average(abs(lb_z - magnit))

    magnitS = magnitude(spect_x, spect_y, spect_z)
    admS_x = np.average(abs(spect_x - magnitS))
    admS_y = np.average(abs(spect_y - magnitS))
    admS_z = np.average(abs(spect_z - magnitS))

    row = []
    row.append(row_id) # id
    row.append(np.mean(lb_x))
    row.append(np.mean(lb_y))
    row.append(np.mean(lb_z))
    row.append(np.mean(spectrum_x))
    row.append(np.mean(spectrum_y))
    row.append(np.mean(spectrum_z))
    row.append(np.median(lb_x))
    row.append(np.median(lb_y))
    row.append(np.median(lb_z))
    row.append(np.median(spect_x))
    row.append(np.median(spect_y))
    row.append(np.median(spect_z))
    row.append(sc_x)
    row.append(sc_y)
    row.append(sc_z)
    row.append(magnit)
    row.append(adm_x)
    row.append(adm_y)
    row.append(adm_z)
    row.append(magnitS)
    row.append(admS_x)
    row.append(admS_y)
    row.append(admS_z)
    row.append(class_label)
    dataset_array.append(row)
    row_id = row_id + 1

cols = ["id", "mean_x", "mean_y", "mean_z",
        "mean_fft_x", "mean_fft_y", "mean_fft_z",
        "median_x", "median_y", "median_z",
        "median_fft_x", "median_fft_y",
```

```
        "median_fft_z", "spec_cent_x",
        "spec_cent_y", "spec_cent_z",
        "magnitude", "adm_x", "adm_y",
        "adm_z", "magnitude_spect", "admS_x",
        "admS_y", "admS_z", "class"]
dataset = pd.DataFrame(data=dataset_array,
                columns=cols)
```

Listing 5. Creación del Dataset

A. Identificación de Outliers

Para identificar los valores extremos se utilizó la técnica gráfica, donde se pudo visualizar (ver fig. 12) los valores extremos de los atributos *mean_y, mean_fft_y, mean_fft_z, media_fft_x, media_fft_y, media_fft_z, adm_x*. A continuación se procedió a verificar las filas afectadas con la función *stats.zscore()*, obteniendo un total 2049 filas que tienen como mínimo un outliers en alguna columna.
Dado que la señal del acelerómetro puede tener valores superiores a los valores máximo o minimos que representa el mismo estado, por ejemplo: en el estado saltar existe un patrón sin embargo puede existir alguién que salta un poco más en cierto intervalo de la muestra, por lo tanto consideramos no eliminar y solo reemplazar valores con la ayuda del IRQ. Después de aplicar la actualización a los valores atípicos mínimos y máximos con $Q1 - 1.5 * IQR$ y $Q3 + 1.5 * IQR$ respectiva se puede visualizar el resultado en la fig. 13.

```
Q1 = ds.iloc[:, :-1].quantile(0.25)
Q3 = ds.iloc[:, :-1].quantile(0.75)
IQR = Q3 - Q1
ds.iloc[:, :-1] = np.where(ds.iloc[:, :-1] < (Q1 -
     1.5 * IQR), Q1, ds.iloc[:, :-1])
ds.iloc[:, :-1] = np.where(ds.iloc[:, :-1] > (Q3 +
     1.5 * IQR), Q3, ds.iloc[:, :-1])
```

Listing 6. Tratamiento de Outliers

B. Normalización

El filtro Gaussiano[3] de pasa baja permite eliminar ruido de la señal de forma ligera con un sigma de uno (1) para la desviación del kernel Gaussiano. Luego se aplicó un tratamiento a los valores extremos. Con el objetivo de mejorar el rendimiento del modelo y disminuir la influencia de valores máximo o mínimos (resago de los procesos anteriores), se aplica una normalización de norma unitaria (L2 Norm) en el rango de $[-1, 1]$ con la librería *sklearn.preprocessing.Normalize*.

```
X = ds.iloc[:,:-1]
y = ds["class"]
data_cols = list(X)
scaler = Normalizer().fit(X)
tmp_scaled = scaler.transform(X)
X = pd.DataFrame(tmp_scaled)
X.columns = data_cols
X.describe()
```

Listing 7. Normalización

[3]Filtro Gaussiano: el filtro gaussiano pasa bajos es un operador bidimensional de convolución que se utiliza para eliminar ruido y suavizar bordes.

VII. Modelo de Clasificación

Con el dataset ya definido, procedemos con la construcción del modelo de clasificación.

A. Construcción del Modelo

El modelo de clasificación que se utilizará es una Red Neuronal. Para definir el número de capas ocultas y el número de nodos por capa a utilizar, se va a realizar. Adicionalmente, también se harán pruebas con diferentes learning rates para obtener los parámetros que nos brindan la mejor precisión.
Estos 3 parámetros podrán tomar los valores mostrados en la tabla II:

Parámetro	Valores
Capas (ocultas)	[2,3]
Nodos	[30, 40, 50, 60, 70, 80]
Learning Rate	[0.01, 0.001, 0.0001, 0.0001]

Tabla II
Parámetros a probar

Solo se realizarán pruebas con combinaciones de 2 y 3 capas ocultas. En todos los casos, las capas tendrán el mismo número de nodos, desde 30 hasta 80, con un aumento de 10 nodos en cada prueba. El tercer parámetro a variar, será el learning rate inicial. En total se realizarán $2x6x4 = 48$ entrenamientos diferentes para obtener la mejor combinación de parámetros.

En el listado de código 8 se muestra el código usado para encontrar los mejores parámetros para una red neuronal de 2 capas. Se realizó lo mismo utilizando 3 capas ocultas.

```
nodes_per_layer= [30, 40, 50, 60, 70, 80]
learn_rate_init = [10e-6, 10e-5,10e-4, 10e-3]
train_par_res = []
for n_nodes in nodes_per_layer:
    for learn_rate in learn_rate_init:
        clf = MLPClassifier(
            hidden_layer_sizes=(n_nodes, n_nodes),
            max_iter=500,
            learning_rate_init=learn_rate,
            solver='adam')
        clf.fit(X_train, y_train)
        train_score = clf.score(X_train, y_train)
        test_score = clf.score(X_test, y_test)

        y_pred = clf.predict(X_test)

        acc_i = accuracy_score(y_test, y_pred)
        vals = []
        vals.append(2) #num hidden layers
        vals.append(n_nodes) #num nodes per layer
        vals.append(clf.n_iter_) #num iterations
        vals.append(learn_rate)
        vals.append(clf.loss_) #loss
        vals.append(acc_i) #accuracy
        vals.append(train_score) #train_score
        vals.append(test_score) #test_score

        train_par_res.append(vals)
```

Listing 8. Búsqueda de parámetros óptimos

En la tabla III se muestran los mejores 10 resultados (respecto a la precisón). Cabe resaltar que la función optmizadora que se utilizará es "Adam", el cual es el valor por defecto de un modelo de clasificación en la libería sklearn de python.

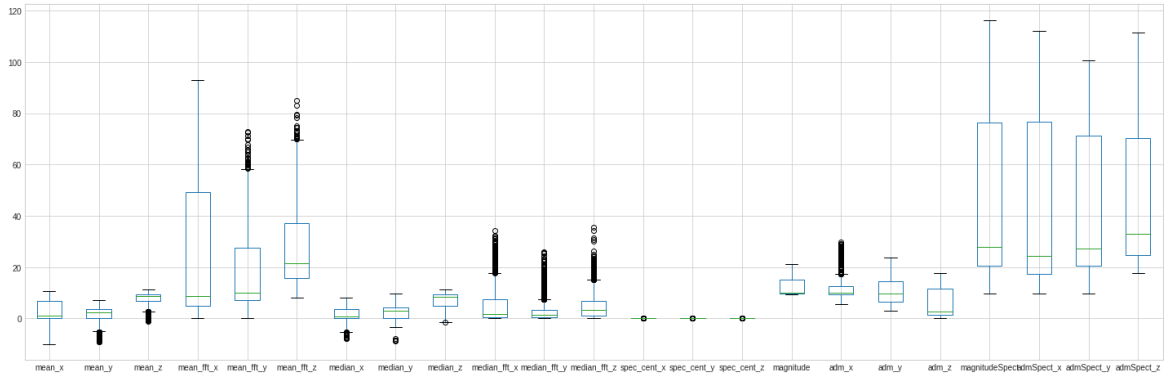

Figura 12. Identificación de outliers

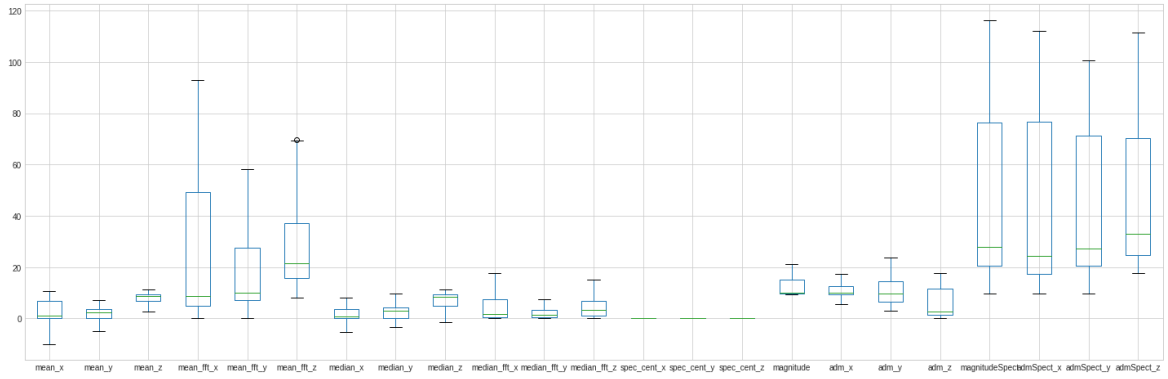

Figura 13. Limpieza de outliers

#Capas	Nodos	#Iter	Lr	Loss	Precisión
2	70	153	0.010	0.108719	0.963152
2	80	118	0.010	0.098859	0.958034
2	60	103	0.010	0.109764	0.958034
2	40	354	0.001	0.123083	0.957011
2	40	102	0.010	0.111774	0.957011
2	80	427	0.001	0.123365	0.954964
2	60	452	0.001	0.111216	0.954964
3	40	423	0.001	0.111395	0.954964
3	70	094	0.010	0.119929	0.954964
3	40	125	0.010	0.112220	0.953940

Tabla III
10 MEJORES RESULTADOS DE LOS PARÁMETROS

En esta tabla III también se ha incluído el número de iteraciones que fueron necesarias para que el modelo convergiera; esto nos servirá para conocer otro parámetro, el número máximo de iteraciones o epochs.

Como se puede observar, el mejor resultado se obtuvo al usar 2 capas ocultas, con 70 neuronas cada uno. El número de iteraciones necesarias fue de 153 por lo que usaremos un número un poco mayor para darle espacio de diferencia cuando se entrene la red neuronal a utilizar. Entonces la combinación de parámetros que se utilizará para el entrenamiento de nuestro modelo será la siguiente:

- Capas ocultas: 2
- Nodos por capa oculta: 70
- Learning Rate: 0.01
- Máximo número de iteraciones (epochs): 160

B. *Entrenamiento*

Ahora, con lo parámetros ya definidos, procedemos a realizar el entrenaimento. Esto se muestra en el código en el listado 9. Primero se declaran las variables que contendran los datos de entrenamiento y los datos de prueba del modelo. Como se había mencionado antes, se usarán el 70% de los datos para el entrenamiento y 30% para las pruebas de precisión del modelo. Luego se procede con el entrenamiento del modelo.

```
X = ds.iloc[:,:-1]
y = ds["class"]

X_train, X_test, y_train, y_test =
    train_test_split(X, y, stratify=y,
    test_size=0.3, random_state=2)

clf = MLPClassifier(
            hidden_layer_sizes=(70, 70),
            max_iter=160, solver='adam',
            learning_rate_init=0.01)

clf.fit(X_train, y_train)
```

Listing 9. Entrenamiento del Modelo

Con la cual obtenemos la curva de pérdida por iteración mostrada en la figura 14.

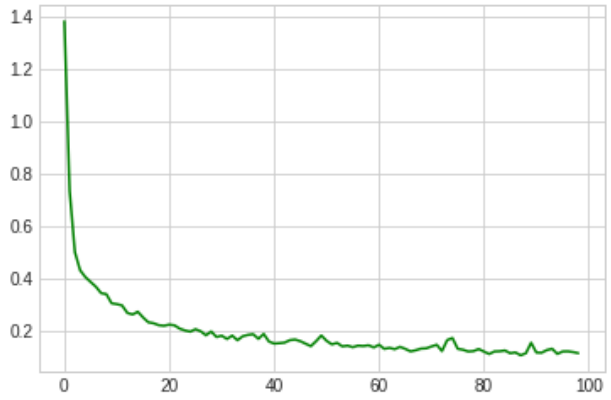

Figura 14. Pérdida vs epochs

Y los resultados de precisión tanto de los datos de entrenamiento como de prueba, como se muestra en el código listado 10, obtenemos que el resultado es:

- Precisión datos de entrenamiento: 0.9661
- Precisión datos de prueba: 0.9621

```
clf.score(X_train, y_train)
clf.score(X_test, y_test)
```

Listing 10. Scores del entrenamiento

Para verificar el rendimiento de nuestra red neuronal, procedemos a hacer un k-fold de nuestro modelo. Para esto hacemos uso de la librería sklearn, como se muestra en el listado de código 11

```
from sklearn.model_selection import
    cross_val_score

scores = cross_val_score(clf, X, y, cv=10)
scores
```

Listing 11. Medición de rendimiento del modelo

Usamos k=10 y obtenemos los siguientes resultados de la tabla IV.

Iteración	Precisión
1	0.95092025
2	0.96625767
3	0.95705521
4	0.95398773
5	0.97852761
6	0.92615385
7	0.94153846
8	0.96615385
9	0.96000000
10	0.95384615
Prom	0.95544407

Tabla IV
RESULTADOS DEL K-FOLD AL MODELO

También calculamos la matriz de confusión de estos resultados de la clasificación para los datos de prueba. Para esto hacemos uso de la librería sklearn. En el listado de código 12 se muestra cómo se obtiene esta matriz.

```
from sklearn.metrics import confusion_matrix

y_pred = clf.predict(X_test)
cm = confusion_matrix(y_test.values, y_pred)
```

Listing 12. Obteniendo la Matriz de confusión

Cuyo resultado se muestra en la fig. 15

```
array([[167,   0,   0,   0,  19],
       [  0, 195,   0,   0,   0],
       [  0,   0, 176,   3,   0],
       [  0,   0,   0, 232,   0],
       [ 15,   0,   0,   0, 170]])
```

Figura 15. Matriz de Confusión

En el listado 13 de código se muestra la construcción de una función auxiliar para mostrar de una mejor manera visual la matriz de confusión del modelo entrenado y su uso.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)
            [:, np.newaxis]

    fig, ax = plt.subplots(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=
        cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90,
        fontsize=20)
    plt.yticks(tick_marks, classes, fontsize=20)

    fmt = '0.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape
        [0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] >
                     thresh else "black", fontsize
                     =20)

    plt.tight_layout()
    plt.ylabel('Etiquetas verdaderas', )
    plt.xlabel('Etiquetas predecidas')
```

Listing 13. Función para la Matriz de Confusión

En el siguiente listado de código 14 se muestra la llamada a esta función, pasando la matriz de confusión antes calculada.

```
labels = ['CAMINANDO', 'SENTADO', 'SALTANDO', '
    CORRIENDO', 'SUBIENDO']
plot_confusion_matrix(cm, classes=labels,
    normalize=True, title='Matriz de confusion
    normalizada', cmap = plt.cm.Greens)
```

Listing 14. Obteniendo la Matriz de Confusión

La matriz de confusión resultante se muestra en la figura 16. En este caso, la matriz de confusión está normalizada para mostrar valores entre 0 y 1.

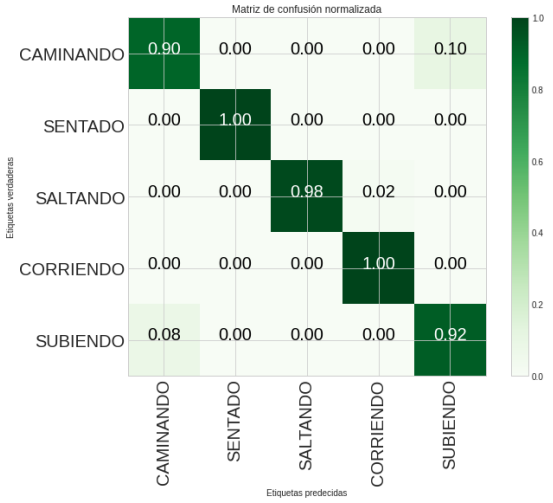

Figura 16. Matriz de Confusión Normalizada

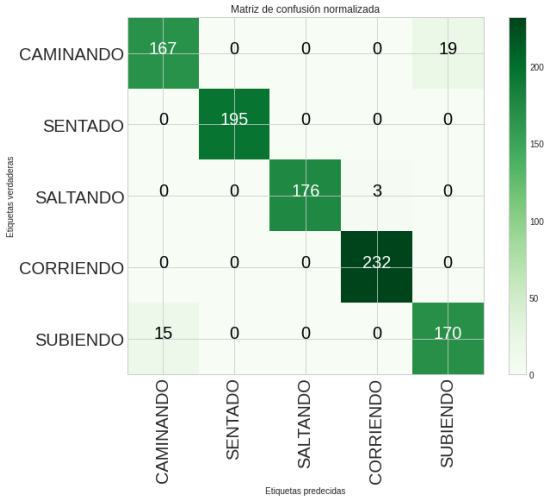

Figura 17. Matriz de Confusión no normalizada

En la figura 17 se muestra la matriz de confusión no normalizada, usando la función antes definida.
Finalmente, guardamos el modelo entrenado para ser consultado por la aplicación móvil (listado 15) haciendo uso de la librería joblib.

```
filename = 'modelo_deteccion.uni'
joblib.dump(clf, filename)
```

Listing 15. Almacenamiento del modelo

C. Consulta del Modelo

En el proceso modelado y entranamiento se generó un archivo *modelo_deteccion.uni* donde se guardó el modelo optimizado y los pesos de red neuronal "MLPClassifier". Todo el proceso de de predicción tiene una duracion de dos (2) segundos, es decir cada dos segundos se actualiza el estado en el móvil. A continuación se detalle el procedimiento especial para la consulta al modelo (ver fig. 18):

1) La persona de prueba inicia en la aplicación móvil la monitorización.
2) El móvil se conecta al servidor por HTTP en el puerto 5000.
3) El servidor establece la conexión por WebSocket.
4) El móvil envía las acelerqaciones por coordenas del acelerómetro mediante mensajes.
5) Cada 100 capturas el servidor de WebSocket envía la matriz a la función extracción y luego al evaluación por el modelo pre-entrenado.
6) El modelo retorna el estado predecido al controlador de eventosl del WebSocket.
7) El servidor retorna el estado predecido al móvil mediante un mensaje.
8) El móvil muestra el pantalla el estado predecido.

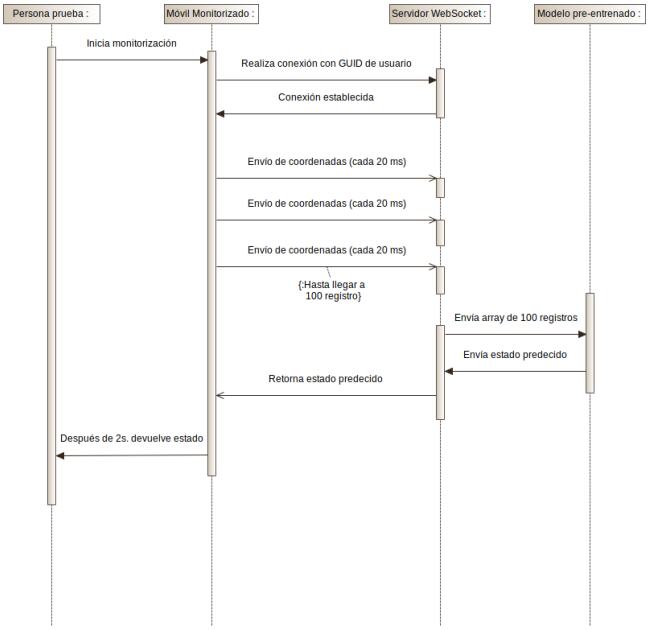

Figura 18. Fllujo de Consulta del Modelo

VIII. Resultados

A. Resultados del Modelo

Todos los resultados obtenidos en el proceso de implementación del modelo se han detallado y descrito en las secciones anteriores.

B. Resultados de la Consulta del Modelo

De manera inicial se realizó cinco pruebas con los estados SENTADO, CAMINANDO y SALTADO a seis segundos cada una, logrando predicción de forma correcta en todos los casos (ver fig.22 y 21).
Luego se procedió a realizar una prueba con todos los estados durante tres minutos (tener presenta que la predicción se da cada 2 s). Logrando obtener 90 registros por cada estado haciendo un total de 450 predicciones (ver fig. 19) con el siguiente resultado ver fig. 20.

	PREDECIDO					
REAL	CAMINANDO	CORRIENDO	SALTANDO	SENTADO	SUBIENDO	**Total**
CAMINANDO	33	2		1	54	**90**
CORRIENDO		77		2	11	**90**
SALTANDO	2	24	61	1	2	**90**
SENTADO				90		**90**
SUBIENDO	19	12			59	**90**
Total	**54**	**115**	**61**	**94**	**126**	**450**

Figura 19. Cuadro de predicciones reales

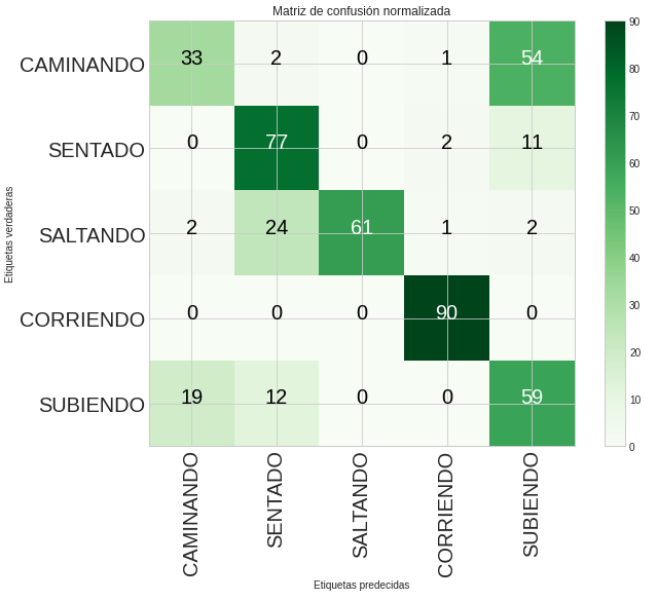

Figura 20. Matriz de confusión de predicciones reales

IX. Conclusiones

Se logró construir un modelo de clasificación de actividades físicas y una aplicación que lo consulta en tiempo real.
Todos los modelos de arquitectura RNA implementados alcanzan un valor de pérdida (Loss) menor a 0.1 a partir de la iteración número 80.
De acuerdo a la matriz de confusión el modelo logra predecir en el dataset de pruebas: 90% el estado CAMINANDO, 100% el estado SENTADO, 98% el estado SALTANDO, 100% el estado CORRIENDO y 92% el estado SUBIENDO.

La efectividad de las 23 características seleccionadas demuestran su efectividad en los resultados xx%

Se demostró que la ventana de tiempo de dos (2) segundos de recolección de datos para la consulta en tiempo real al modelo es suficiente para predecir el estado actual del portador del dispositivo móvil.

X. Recomendaciones

Se recomienda hacer pruebas construyendo la red neuronal utilizando otras librerías (como Pytorch o Keras) en la cual se tenga un mejor control de cada paso de la red , aunque esto requiere un tiempo mucho más prolongado y mayor conocimiento de cada proceso interno de una red neuronal.

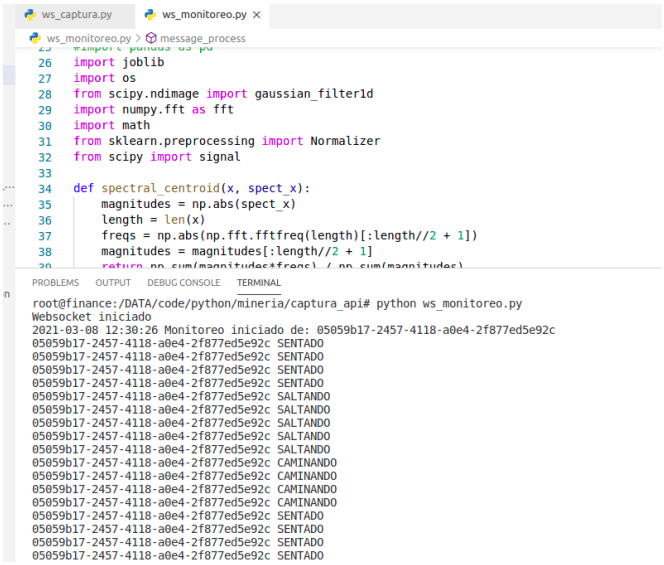

Figura 21. Servidor de Monitoreo

Figura 22. Resultado de Monitoreo

Se recomienda utilizar la Transformada Rápida de Fourier (FFT) para la extracción de características de la señal del acelerómetro.
Es recomendable utilizar el centroide espectral derivado del cálculo del FFT de la señal del acelerómetro como patrón de característica adicional a las entradas del modelo.
El empleo de la magnitud y de la diferencia promedio de la media como características de las señales del acelerómetro es recomendado para la extracción de mayor información contenida en dichas señales.
Se recomienda utilizar un WebSoket como mecanismos de comunicación entre el dispositivo móvil y el servidor con el objetivo de mejorar la latencia en el envío de mensaje, dado que las frecuencias del acelerómetro son altas. Esta recomendación fue implementada, no fue necesario usar APIs.
El uso de esta tecnología ha permitido que la interacción en la recolección y monitoreo de los datos de la señal del acelerómetro sea en tiempo real sin sobrecargar la red,

dispositivo móvil ni el servidor.

BIBLIOGRAFÍA

[1] H.-H. Hsu, K.-C. Tsai, Z. Cheng, and T. Huang, "Posture recognition with g-sensors on smart phones," in *2012 15th International Conference on Network-Based Information Systems*. IEEE, 2012, pp. 588–591.

[2] G. Hayes, K. P. Dowd, C. MacDonncha, and A. E. Donnelly, "Tracking of physical activity and sedentary behavior from adolescence to young adulthood: a systematic literature review," *Journal of Adolescent Health*, vol. 65, no. 4, pp. 446–454, 2019.

[3] F. B. Ortega, K. Konstabel, E. Pasquali, J. R. Ruiz, A. Hurtig-Wennlöf, J. Mäestu, M. Löf, J. Harro, R. Bellocco, I. Labayen *et al.*, "Objectively measured physical activity and sedentary time during childhood, adolescence and young adulthood: a cohort study," *PloS one*, vol. 8, no. 4, p. e60871, 2013.

[4] K. Li, D. Haynie, L. Lipsky, R. J. Iannotti, C. Pratt, and B. Simons-Morton, "Changes in moderate-to-vigorous physical activity among older adolescents," *Pediatrics*, vol. 138, no. 4, 2016.

[5] D. R. Young, D. Cohen, C. Koebnick, Y. Mohan, B. I. Saksvig, M. Sidell, and T. Wu, "Longitudinal associations of physical activity among females from adolescence to young adulthood," *Journal of Adolescent Health*, vol. 63, no. 4, pp. 466–473, 2018.

[6] J. D. Ceron, D. M. Lopez, and G. A. Ramirez, "A mobile system for sedentary behaviors classification based on accelerometer and location data," *Computers in Industry*, vol. 92, pp. 25–31, 2017.

[7] T. B. Singha, R. K. Nath, and A. Narsimhadhan, "Person recognition using smartphones' accelerometer data," *arXiv preprint arXiv:1711.04689*, 2017.

[8] K. R. Rao, D. N. Kim, and J. J. Hwang, *Fast Fourier transform-algorithms and applications*. Springer Science & Business Media, 2011.

ANEXO A: TRANSFORMADA RÁPIDA DE FOURIER

La Transformada Rápida de Fourier (FFT, por sus siglas en inglés) es una implementación eficiente de la Transformada Discreta de Fourier (DFT, por sus siglas en inglés). Entre todas las transformaciones discretas, la DFT es la más ampliamente utilizada en el procesamiento digital de señales. El desarrollo de la DFT original llevado a cabo por Cooley y Tukey, seguido de varias modificaciones y mejoras hechas por investigadores, ha proporcionado el incentivo y el ímpetu para su rápida y amplia utilización en un gran número de diversas disciplinas. La DFT puede ser definida como [8]:

$$X^F(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{4}$$

k=0,1,...,N-1, N coefientes DFT

$$W_N = e^{-j\frac{2\pi}{N}} \tag{5}$$

$$W_N^{kn} = e^{-j\frac{2\pi kn}{N}} \tag{6}$$

Donde x(n), n=0,1,...,N-1 es una secuencia muestreada uniformemente, T es el intervalo de muestreo. $\mathrm{W_N} = \exp(-\mathrm{j}2/\mathrm{N})$ es la N-ésima raíz de la unidad, y $\mathrm{X^F(k)}$, k=0,1,...,N-1 es el k-ésimo coeficiente DFT. $\mathrm{j} = \sqrt{-1}$.

Ejemplo de aplicación:

Se calculará la DFT de una señal de tiempo, que resulta de la superposición (o adición) de 2 señales sinusoidales de frecuencias diferentes (50Hz y 80Hz), empleando el método de la Transformada Rápida de Fourier (FFT). Para ello se utilizarán los métodos 'fft' y 'fftfreq' de la librería SciPy (v.0.16.1) de Python.

En las figuras Fig. 23 y Fig. 24 se muestran las señales sinusoidales independientes, mientras que en la Fig. 25 se observa la señal resultante de la adición de las dos anteriores mencionadas.

En la Fig. 26 se muestra la parte positiva del módulo de la DFT de la señal acoplada.

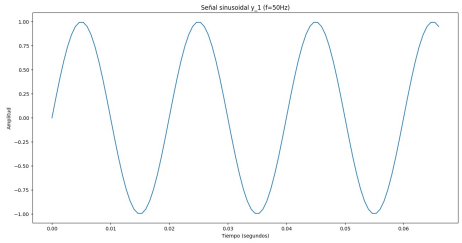

Figura 23. Señal sinusoidal de 50Hz

El código implementado se detalla a continuación:

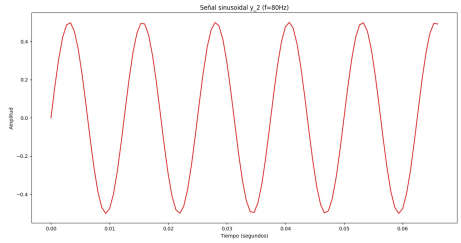

Figura 24. Señal sinusoidal de 80Hz

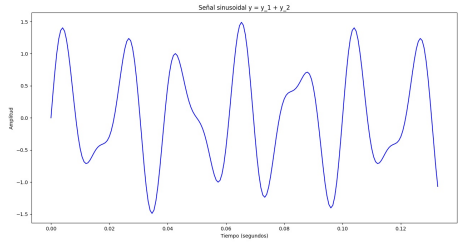

Figura 25. Señal superpuesta resultante

```python
from scipy.fft import fft, fftfreq
import matplotlib.pyplot as plt
import numpy as np

# N mero de muestras
N = 600
# Periodo de muestreo
T = 1.0 / 1500.0

x = np.linspace(0.0, N*T, N, endpoint=False)
y1 = np.sin(50.0 * 2.0*np.pi*x)
y2 = 0.5*np.sin(80.0 * 2.0*np.pi*x)
plt.figure(1)
plt.plot(x[0:100], y1[0:100])
plt.title('Se al sinusoidal y_1 (f=50Hz)')
plt.xlabel('Tiempo (segundos)')
plt.ylabel('Amplitud')
plt.figure(2)
plt.plot(x[0:100], y2[0:100],'-r')
plt.title('Se al sinusoidal y_2 (f=80Hz)')
plt.xlabel('Tiempo (segundos)')
plt.ylabel('Amplitud')
plt.show()

y = y1 + y2
plt.figure(3)
plt.plot(x[0:200], y[0:200], '-b')
plt.title('Se al sinusoidal y = y_1 + y_2')
plt.xlabel('Tiempo (segundos)')
plt.ylabel('Amplitud')
plt.show()

yf = fft(y)
xf = fftfreq(N, T)[:N//2]
plt.figure(4)
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
plt.title('DFT de la se al y')
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Amplitud')
plt.show()
```

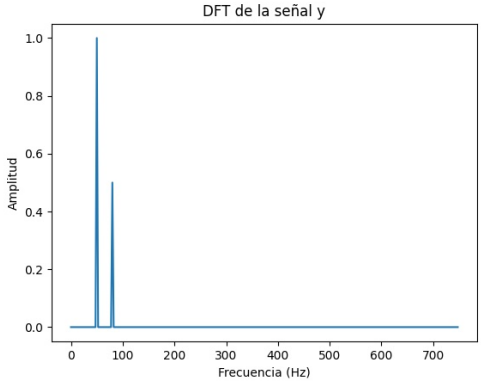

Figura 26. DFT de la señal superpuesta resultante

Anexo B: Código Fuente

```python
#!/usr/bin/env python
# coding: utf-8

# In[2]:


get_ipython().run_line_magic('load_ext', '
    autoreload')
get_ipython().run_line_magic('autoreload', '2')
import os
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy.fft as fft
from scipy.ndimage import gaussian_filter1d
from scipy import signal

# Configuraciones de estilos de Mapplot
# ======================================
plt.style.use('seaborn-whitegrid')
plt.rcParams['image.cmap'] = "bwr"
plt.rcParams['savefig.bbox'] = "tight"

# Omitir los Warnings
# ======================================
import warnings
warnings.filterwarnings('ignore')

# Configuracion de impresi n de pandas
# ======================================
pd.set_option("display.precision", 8)
pd.set_option('display.width', 1000)


# # Preparaci n de la se al (en PC)

# In[102]:


get_ipython().system('rm -rf data_original')
get_ipython().system('rm -rf segmentado')
get_ipython().system('mkdir segmentado')
get_ipython().system('mkdir data_original')
get_ipython().system('cp -r emperatriz*/*
    data_original/')
get_ipython().system('cp -r milca*/* data_original
    /')
get_ipython().system('cp -r milton*/*
    data_original/')
```

```
get_ipython().system('cp -r raul*/* data_original/')
print(" Realizado !")


# In[98]:


len(os.listdir("data_original"))


# In[99]:


# nSamplesInSegment: numero de filas que van en un segmento
# nOverlap: numero de filas que se van a superponer
def sampleSegmentation(data, nSamplesInSegment, nOverlap):
    segments = []
    nSamples = len(data)
    for i in range(0, nSamples, nSamplesInSegment-nOverlap):
        if i + nSamplesInSegment > nSamples:
            break
        segments.append(data[slice(i, i + nSamplesInSegment, 1)])
    return segments


def generateSamples(fileName, dir_segmentado):
    numSamples = 100
    numOverlap = 25
    df = pd.read_csv(fileName)
    segments = sampleSegmentation(df, numSamples, numOverlap)
    for i in range(len(segments)):
        f_path_parts = fileName.split("/")
        f_name, f_ext = f_path_parts[-1].split(".")
        segment_file_name = dir_segmentado + f_name + "_" + str(i) + "." + f_ext
        if os.path.exists(segment_file_name):
            os.remove(segment_file_name)
        segments[i].to_csv(segment_file_name)

def out_gravity(df, per=.5):
    gravity = [0.0, 0.0, 0.0]
    alpha = 0.8
    for index, row in df.iterrows():
        gravity[0] = alpha * gravity[0] + (1 - alpha) * row["x"]
        gravity[1] = alpha * gravity[1] + (1 - alpha) * row["y"]
        gravity[2] = alpha * gravity[2] + (1 - alpha) * row["z"]

        df.loc[index, "x"] = row["x"] - gravity[0]
        df.loc[index, "y"] = row["y"] - gravity[1]
        df.loc[index, "z"] = row["z"] - gravity[2]
    return df[0:int(per*len(df))]


df_0 = pd.read_csv('data_original/milton_caminando_varon_adulto_20210306083705.csv')
df_1 = pd.read_csv('data_original/milton_sentado_varon_adulto_20210306174604.csv')
df_2 = pd.read_csv('data_original/milton_saltando_varon_adulto_20210306114600.csv')
df_3 = pd.read_csv('data_original/milton_corriendo_varon_adulto_20210324082429.csv')
df_4 = pd.read_csv('data_original/milton_subiendo_varon_adulto_20210324095953.csv')
df_0.shape, df_1.shape, df_2.shape, df_3.shape, df_4.shape


dx = np.linspace(0, 2000, 250)

plt.figure(figsize=(25, 8))
plt.plot(dx, df_0.x, 'k', label='Caminando', linestyle="dashed")
plt.plot(dx, df_1.x, 'r', label='Sentado', linestyle="dotted")
plt.plot(dx, df_2.x, 'c', label='Saltando', linestyle="dashed")
plt.plot(dx, df_3.x, 'b', label='Corriendo', linestyle="dashdot")
plt.plot(dx, df_4.x, 'g', label='Subiendo', linestyle="dashed")
plt.xlabel('tiempo $t(s)$', fontsize=18)
plt.ylabel('acelerometro $(m/s^2)$', fontsize=18)
plt.title('Se al acelerometro en el x', fontsize=18)
plt.legend()
plt.grid()
#plt.show()
plt.savefig('foo1.png')

plt.figure(figsize=(25, 8))
plt.plot(dx, df_0.y, 'k', label='Caminando', linestyle="dashed")
plt.plot(dx, df_1.y, 'r', label='Sentado', linestyle="dotted")
plt.plot(dx, df_2.y, 'c', label='Saltando', linestyle="dashed")
plt.plot(dx, df_3.y, 'b', label='Corriendo', linestyle="dashdot")
plt.plot(dx, df_4.y, 'g', label='Subiendo', linestyle="dashed")
plt.xlabel('tiempo $t(s)$', fontsize=18)
plt.ylabel('acelerometro $(m/s^2)$', fontsize=18)
plt.title('Se al acelerometro en el eje y', fontsize=18)
plt.legend()
plt.grid()
#plt.show()
plt.savefig('foo2.png')
plt.figure(figsize=(25, 8))
plt.plot(dx, df_0.z, 'k', label='Caminando', linestyle="dashed")
plt.plot(dx, df_1.z, 'r', label='Sentado', linestyle="dotted")
plt.plot(dx, df_2.z, 'c', label='Saltando', linestyle="dashed")
plt.plot(dx, df_3.z, 'b', label='Corriendo', linestyle="dashdot")
plt.plot(dx, df_4.z, 'g', label='Subiendo', linestyle="dashed")
plt.xlabel('tiempo $t(s)$', fontsize=18)
plt.ylabel('acelerometro $(m/s^2)$', fontsize=18)
plt.title('Se al acelerometro en el z', fontsize=18)
plt.legend()
plt.grid()
plt.savefig('foo3.png')
plt.show()


# ## Aplicando un filtro de suavisado

```

```

df_0 = pd.read_csv('data_original/
    emperatriz_caminando_mujer_adulto_20210306122006
    .csv')

ejex = int(1*250)
plt.figure(figsize=(25, 8))
ss = gaussian_filter1d(df_0.z,1.2)

plt.plot(np.linspace(0, 2000, ejex), df_0.z, 'k',
    label='Data Original')
plt.legend()
plt.show()


plt.figure(figsize=(25, 8))
plt.plot(np.linspace(0, 2000, ejex),ss, 'r', label
    ='Data Filtrada')
plt.legend()
plt.show()

plt.figure(figsize=(25, 8))
plt.plot(np.linspace(0, 2000, ejex), df_0.z, 'k',
    label='Data Original')
plt.plot(np.linspace(0, 2000, ejex),ss, 'r', label
    ='Data Filtrada')
plt.legend()
plt.show()


plt.figure(figsize=(25, 8))
fs = len(ss)
plt.plot(np.linspace(0, 5, fs), ss, color="g")


# Mostrando el gr fico
plt.show()

spectrum = fft.fft(ss)
freq = fft.fftfreq(len(spectrum))
plt.plot(freq, abs(spectrum))



plt.figure(figsize=(25, 8))
sos = signal.cheby1(1, 2, 0.3, 'lowpass', output='
    sos')
lb_x = signal.sosfilt(sos,ss)
lb_x1 = signal.sosfilt(sos,df_0.z)
fs = len(ss)

# plt.plot(np.array([i for i in range(0, fs)]),
    data)
plt.plot(np.linspace(0, 5, fs), lb_x, color="g")
plt.plot(np.linspace(0, 5, fs), df_0.z, color="b")
plt.plot(np.linspace(0, 5, fs), ss, color="r")
plt.plot(np.linspace(0, 5, fs), lb_x, color="
    orange")
plt.plot(np.linspace(0, 5, fs), lb_x1, '--',color=
    "c")


# Mostrando el gr fico
plt.show()

spectrum = fft.fft(ss)
freq = fft.fftfreq(len(spectrum))
plt.plot(freq, abs(spectrum))


# ## Creaci n de dataset


df_0 = out_gravity(pd.read_csv('data_original/
    emperatriz_caminando_mujer_adulto_20210306122006
    .csv'))
df_0.head(5)


def spectral_centroid(x, spect_x):
    magnitudes = np.abs(spect_x)
    length = len(x)
    freqs = np.abs(np.fft.fftfreq(length)[:length
        //2 + 1])
    magnitudes = magnitudes[:length//2 + 1]
    return np.sum(magnitudes*freqs) / np.sum(
        magnitudes)



def magnitude(x, y, z):
    ln = len(x)
    sum = 0
    for i in range(ln):
        sum = sum + math.sqrt(x[i]**2 + y[i]**2 +
            z[i]**2)
    return sum/ln



dataset_array = []

dir_original_data = 'data_original/'  # carpeta
    con todos los csv
dir_segmentos = 'segmentado/'  # carpeta donde se
    almacenaran los nuevos csv creados


# Generando toda las muestras (3 por cada una)
data_original = os.listdir(dir_original_data)
for mfile in data_original:
    complete_name = dir_original_data + mfile

    if not os.path.isfile(complete_name):
        continue

    generateSamples(complete_name, dir_segmentos)


data_contents = os.listdir(dir_segmentos)
row_id = 0
for mfile in data_contents:
    complete_name = dir_segmentos + mfile

    if not os.path.isfile(complete_name):
        continue
    df_content = pd.read_csv(complete_name)

    lb_x = gaussian_filter1d(df_content.x, 1)
    lb_y = gaussian_filter1d(df_content.y, 1)
    lb_z = gaussian_filter1d(df_content.z, 1)

    spect_x = abs(fft.fft(lb_x))
    spect_y = abs(fft.fft(lb_y))
    spect_z = abs(fft.fft(lb_z))
    data_class = mfile.split("_")
    class_label = data_class[1]

    spectral_centroid_x = spectral_centroid(lb_x,
        spect_x)
    spectral_centroid_y = spectral_centroid(lb_y,
        spect_y)
    spectral_centroid_z = spectral_centroid(lb_z,
        spect_z)

    magnit = magnitude(lb_x, lb_y, lb_z)
    adm_x = np.average(abs(lb_x - magnit))
```

```
    adm_y = np.average(abs(lb_y - magnit))
    adm_z = np.average(abs(lb_z - magnit))

    magnitSpect = magnitude(spect_x, spect_y,
        spect_z)
    admSpect_x = np.average(abs(spect_x -
        magnitSpect))
    admSpect_y = np.average(abs(spect_y -
        magnitSpect))
    admSpect_z = np.average(abs(spect_z -
        magnitSpect))

    row = []
    row.append(row_id) # id
    row.append(np.mean(lb_x)) # media
    row.append(np.mean(lb_y)) # media
    row.append(np.mean(lb_z)) # media
    row.append(np.mean(spect_x))  # media
    row.append(np.mean(spect_y))  # media
    row.append(np.mean(spect_z))  # media
    row.append(np.median(lb_x)) # mediana
    row.append(np.median(lb_y)) # mediana
    row.append(np.median(lb_z)) # mediana
    row.append(np.median(spect_x)) # mediana
    row.append(np.median(spect_y)) # mediana
    row.append(np.median(spect_z)) # mediana
    row.append(spectral_centroid_x) # spectral
        centroid
    row.append(spectral_centroid_y) # spectral
        centroid
    row.append(spectral_centroid_z) # spectral
        centroid
    row.append(magnit) # magnitude
    row.append(adm_x) # Average Difference from
        Mean
    row.append(adm_y) # Average Difference from
        Mean
    row.append(adm_z) # Average Difference from
        Mean
    row.append(magnitSpect)  # magnitude spect
    row.append(admSpect_x)  # Average Difference
        from Mean Spect
    row.append(admSpect_y)  # Average Difference
        from Mean Spect
    row.append(admSpect_z)  # Average Difference
        from Mean Spect
    row.append(class_label)
    dataset_array.append(row)
    row_id = row_id + 1

cols = ["id", "mean_x", "mean_y", "mean_z", "
    mean_fft_x", "mean_fft_y", "mean_fft_z",
        "median_x", "median_y", "median_z", "
            median_fft_x", "median_fft_y", "
            median_fft_z",
        "spec_cent_x", "spec_cent_y", "spec_cent_z
            ", 'magnitude',
        "adm_x", "adm_y", "adm_z",
        'magnitudeSpect', "admSpect_x", "
            admSpect_y", "admSpect_z",
        "class"]

dataset = pd.DataFrame(data=dataset_array, columns
    =cols)
dataset.head()



dataset.describe()


# # Modelado / Entrenamiento


from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import
    train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from scipy import stats
from sklearn.preprocessing import Normalizer
import joblib
import itertools


# ## Read Dataset


ds = dataset.drop(['id'], axis=1)
ds.shape


# ## Datset Cleaning

# ### Verificando Outliers


ds.skew()

ds.boxplot(figsize=(25, 8))



zsc = np.abs(stats.zscore(ds.drop(["class"], axis
    =1), axis=1))
print("Filas que tiene m nimo un outlier en
    alguna columna:", zsc[(zsc > 2.2)].shape[0])
print("Filas que tiene m nimo un outlier en todas
     las columnas:", zsc[(zsc > 2.2).all(axis=1)].
    shape[0])


Q1 = ds.iloc[:, :-1].quantile(0.25)
Q3 = ds.iloc[:, :-1].quantile(0.75)
IQR = Q3 - Q1
print(IQR)


ds.iloc[:, :-1] = np.where(ds.iloc[:, :-1] < (Q1 -
     1.5 * IQR), Q1, ds.iloc[:, :-1])


ds.iloc[:, :-1] = np.where(ds.iloc[:, :-1] > (Q3 +
     1.5 * IQR), Q3, ds.iloc[:, :-1])


ds.boxplot(figsize=(25, 8))


# ### Verificando Balanceo de data

ds.groupby("class", as_index=False).agg({"class":
    "count"}).rename(columns = {"class":"
    total_x_class"})


# ##  Data Preprocessing

ds["class"].value_counts()


# ### Codificar variables categ ricas

ds["class"] = np.where(ds["class"] == "caminando",
     0,
        np.where(ds["class"] == "sentado", 1,
        np.where(ds["class"] == "saltando", 2,
        np.where(ds["class"] == "corriendo", 3, 4
```

```
))))


ds["class"].value_counts()


# ### Normalizar

#features = ds.columns
X = ds.iloc[:, : -1]
y = ds["class"]

data_cols = list(X)
scaler = Normalizer().fit(X)
tmp_scaled = scaler.transform(X)
X = pd.DataFrame(tmp_scaled)
X.columns = data_cols
X.describe()


# ### Generando dato de prueba y entranamiento




X_train, X_test, y_train, y_test =
        train_test_split(X, y, stratify = y,
            test_size = 0.3)




list(set(y_train))


# ## Modelo, afinamiento

# Entrenamiento con varios parametros

nodes_per_layer= [30, 40, 50, 60, 70, 80]
learn_rate_init = [10e-6, 10e-5,10e-4, 10e-3]
train_par_res = []
for n_nodes in nodes_per_layer:
    for learn_rate in learn_rate_init:
        clf = MLPClassifier(hidden_layer_sizes=(
            n_nodes, n_nodes), max_iter=500,
                            learning_rate_init=
                                learn_rate, solver
                                ='adam')
        clf.fit(X_train, y_train)
        train_score = clf.score(X_train, y_train)
        test_score = clf.score(X_test, y_test)

        y_pred = clf.predict(X_test)

        acc_i = accuracy_score(y_test, y_pred)
        vals = []
        vals.append(2) #num hidden layers
        vals.append(n_nodes) #num nodes per layer
        vals.append(clf.n_iter_) #num iterations
        vals.append(learn_rate)
        vals.append(clf.loss_) #loss
        vals.append(acc_i) #accuracy
        vals.append(train_score) #train_score
        vals.append(test_score) #test_score

        train_par_res.append(vals)

for n_nodes in nodes_per_layer:
    for learn_rate in learn_rate_init:
        clf = MLPClassifier(hidden_layer_sizes=(
            n_nodes, n_nodes, n_nodes), max_iter
            =500,
                            learning_rate_init=
                                learn_rate, solver
                                ='adam')
        clf.fit(X_train, y_train)
        train_score = clf.score(X_train, y_train)
        test_score = clf.score(X_test, y_test)

        y_pred = clf.predict(X_test)
        acc_i = accuracy_score(y_test, y_pred)
        vals = []
        vals.append(3) #num hidden layers
        vals.append(n_nodes) #num nodes per layer
        vals.append(clf.n_iter_) #num iterations
        vals.append(learn_rate)
        vals.append(clf.loss_) #loss
        vals.append(acc_i) #accuracy
        vals.append(train_score) #train_score
        vals.append(test_score) #test_score

        train_par_res.append(vals)

cols = ["Hidden Layers", "Nodes per layer", "#
    iter", "learning rate",
            "loss", "accuracy", "train_score", "
                test_score"]
trains_params_dt = pd.DataFrame(data=train_par_res
    , columns=cols)
trains_params_dt.head()


trains_params_dt.sort_values(by=['accuracy'],
    ascending=False).head(10)


get_ipython().run_cell_magic('time', '', '# clf.
    best_loss_, clf.loss_, clf.n_iter_\n#clf =
    MLPClassifier(hidden_layer_sizes=(10,10),
    max_iter=300)\nclf = MLPClassifier(
    hidden_layer_sizes=(70, 70), max_iter=160,
    learning_rate_init=0.01, solver=\'adam\')\n#
    clf = MLPClassifier(hidden_layer_sizes=(100,
    100), max_iter=200, learning_rate_init=0.001)\
    nclf.fit(X_train, y_train)\nlr = clf.
    loss_curve_\nplt.plot(range(len(lr)), lr,
    color="g")\nclf.score(X_train, y_train)')


clf.best_loss_, clf.loss_, clf.score(X_train,
    y_train), clf.score(X_test, y_test)


filename = 'modelo_deteccion_v4.uni'
joblib.dump(clf, filename)


y_pred = clf.predict(X_test)


temp = pd.DataFrame(clf.predict_proba(X_test),
    columns=["a", "b", "c", "d", "e"])
temp.columns
rg1 = 0.15
rg2 = 0.35

temp[((temp.a < rg2) & (temp.a > rg1)) & ((temp.b
    < rg2) & (temp.b > rg1))]
# Parece que nunca va encontrar ninguno


accuracy_score(y_test, y_pred)


cm = confusion_matrix(y_test.values, y_pred)
cm
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    fig, ax = plt.subplots(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90, fontsize=20)
    plt.yticks(tick_marks, classes, fontsize=20)

    fmt = '0.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black", fontsize=20)

    plt.tight_layout()
    plt.ylabel('Etiquetas verdaderas', )
    plt.xlabel('Etiquetas predecidas')



labels = ['CAMINANDO', 'SENTADO', 'SALTANDO', 'CORRIENDO', 'SUBIENDO']
plot_confusion_matrix(cm, classes=labels, normalize=True, title='Matriz de confusi n normalizada', cmap = plt.cm.Greens)


var_tmp = pd.DataFrame(y_test)
var_tmp["pred"] = y_pred
var_tmp["caminando"] = np.where((var_tmp["class"] == var_tmp["pred"]) & (var_tmp["class"] == 0), 1, 0)
var_tmp["sentado"] = np.where((var_tmp["class"] == var_tmp["pred"]) & (var_tmp["class"] == 1), 1, 0)
var_tmp["saltando"] = np.where((var_tmp["class"] == var_tmp["pred"]) & (var_tmp["class"] == 2), 1, 0)
var_tmp["corriendo"] = np.where((var_tmp["class"] == var_tmp["pred"]) & (var_tmp["class"] == 3), 1, 0)
var_tmp["subiendo"] = np.where((var_tmp["class"] == var_tmp["pred"]) & (var_tmp["class"] == 4), 1, 0)
var_tmp

var_tmp.groupby(["caminando", "sentado", "saltando", "corriendo", "subiendo"], as_index=False).agg({"caminando": "sum", "sentado": "sum", "saltando": "sum", "corriendo": "sum", "subiendo": "sum"})
cm


model = joblib.load("modelo_deteccion_v4.uni")


X_test.iloc[1:2,:]

y_pred = model.predict(X_test)

cm = confusion_matrix(y_test.values, y_pred)
cm


labels = ['CAMINANDO', 'SENTADO', 'SALTANDO', 'CORRIENDO', 'SUBIENDO']
plot_confusion_matrix(cm, classes=labels, normalize=False, title='Matriz de confusi n normalizada', cmap = plt.cm.Greens)


model.score(X_test, y_test)
```

Código Fuente