

DESARROLLO DE UN ALGORITMO INTELIGENTE PARA LA DIFERENCIACION ENTRE MOVIMIENTOS RUTINARIOS Y ARRITMIAS CARDÍACAS EN MONITOREO ECG WEARABLE

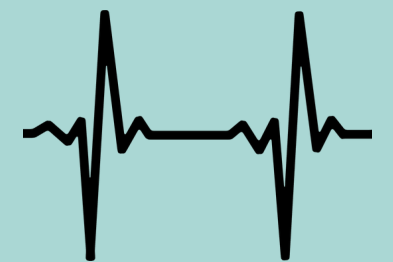
GRUPO 6:

- DIEGO ALBERTO CUBA MONTEROLA
- ROGGER A HUAMAN GONZALES
- HIROSHI JULIO KAMEYA INAFUKU
- ALEJANDRO JAVIER ROSAS
GONZÁLEZ-ZÚÑIGA
- DAVID PUMA SILVA

ESTADO DEL ARTE

BeamO

Dispositivo médico 4 en 1 que combina un termómetro sin contacto, un oxímetro, un estetoscopio digital y un **electrocardiograma (ECG) de una derivación**, permitiendo realizar chequeos médicos completos desde casa.



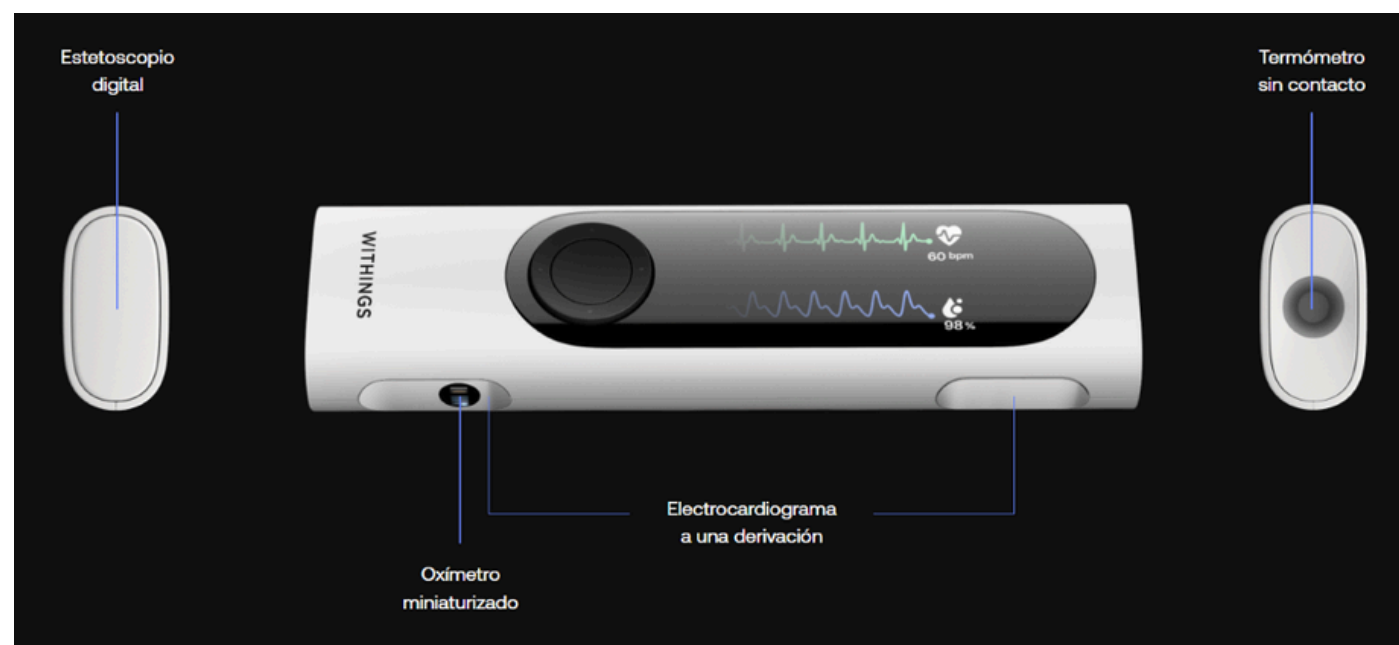
ECG de una derivación: Captura señales eléctricas del corazón con alta precisión, adecuado para chequeos rutinarios o monitoreo continuo.



Detección de fibrilación auricular: Identifica arritmias cardíacas comunes, proporcionando alertas oportunas sobre posibles problemas de salud.



Conectividad con la aplicación: Los registros de ECG se sincronizan con la app de Withings, permitiendo compartir resultados fácilmente con médicos para evaluaciones detalladas.



Apple Watch

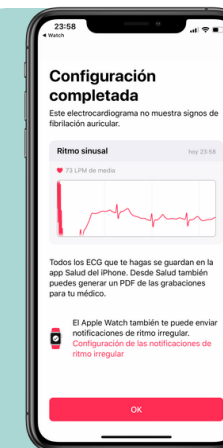
Smartwatch avanzado que combina funciones de conectividad, salud y fitness. Su tecnología de monitoreo de ECG permite registrar un electrocardiograma de una derivación directamente desde la muñeca, ayudando a identificar ritmos cardíacos irregulares, como la fibrilación auricular, y alertar sobre posibles problemas de salud cardíaca.



Electrodos integrados: Utiliza electrodos en la parte trasera del reloj y la corona digital para generar un ECG de una derivación en 30 segundos.



Detección de anomalías: Analiza el ritmo cardíaco para detectar signos de fibrilación auricular u otras irregularidades.



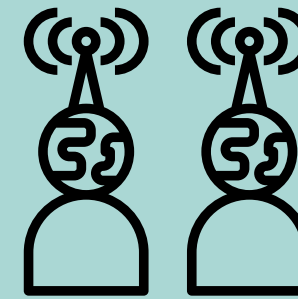
Historial y reportes: Los datos se almacenan en la aplicación Salud y pueden compartirse fácilmente con profesionales médicos para análisis detallados.

Wearable Device – US9955887B2

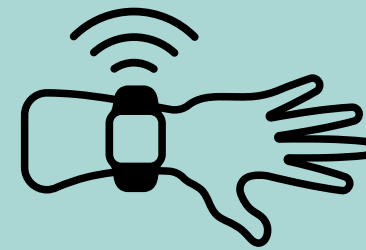
La patente US9955887 pertenece a iRhythm Technologies, Inc. y describe un dispositivo monitor portátil diseñado para registrar datos cardíacos. El sistema incluye métodos y sistemas para extraer características específicas de interés de los datos recogidos, que posteriormente son transmitidas para proporcionar información de salud relacionada.



Registro continuo de datos cardíacos: El dispositivo capta datos como los intervalos R-R, lecturas de acelerómetro y la hora del registro, permitiendo un monitoreo detallado del ritmo cardíaco.



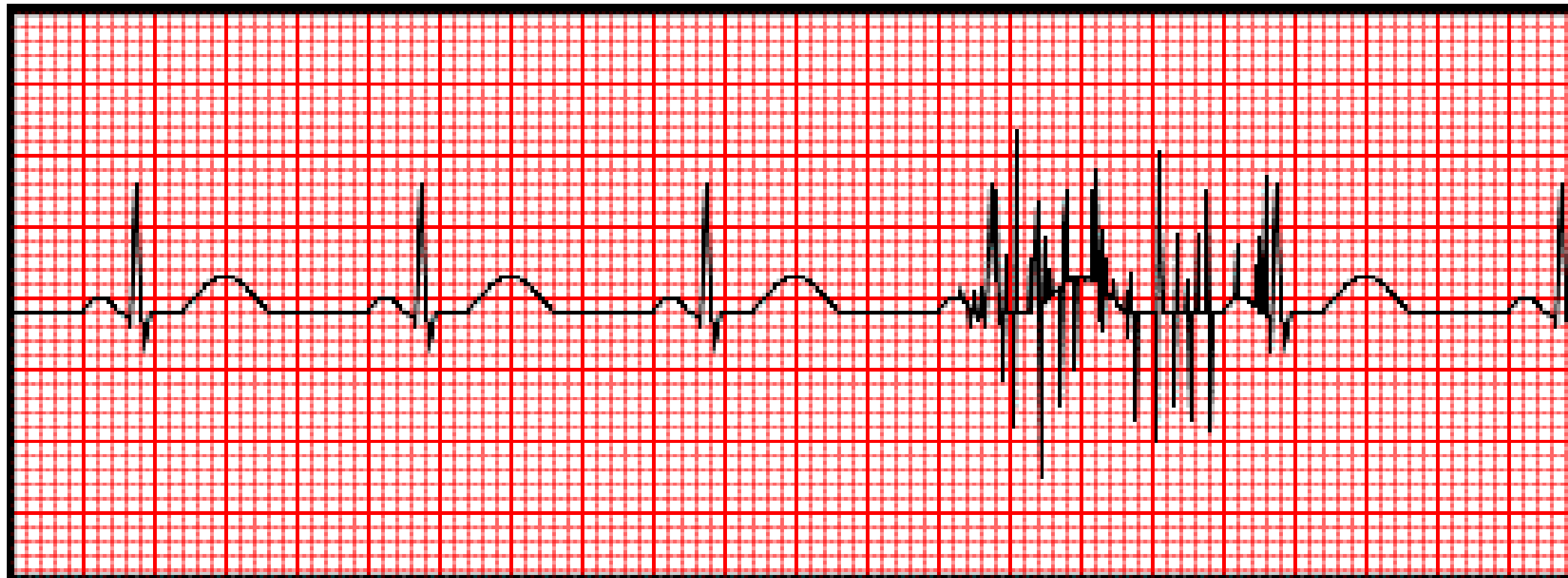
Análisis y transmisión remota: Extrae características de los datos cardíacos y las envía a un servidor para su análisis, lo que facilita diagnósticos en tiempo real o diferido.



Compatibilidad con tecnologías wearables: Diseñado para ser usado en dispositivos como parches, relojes inteligentes u otros sensores portátiles, maximizando la comodidad y la integración en la vida diaria del usuario.

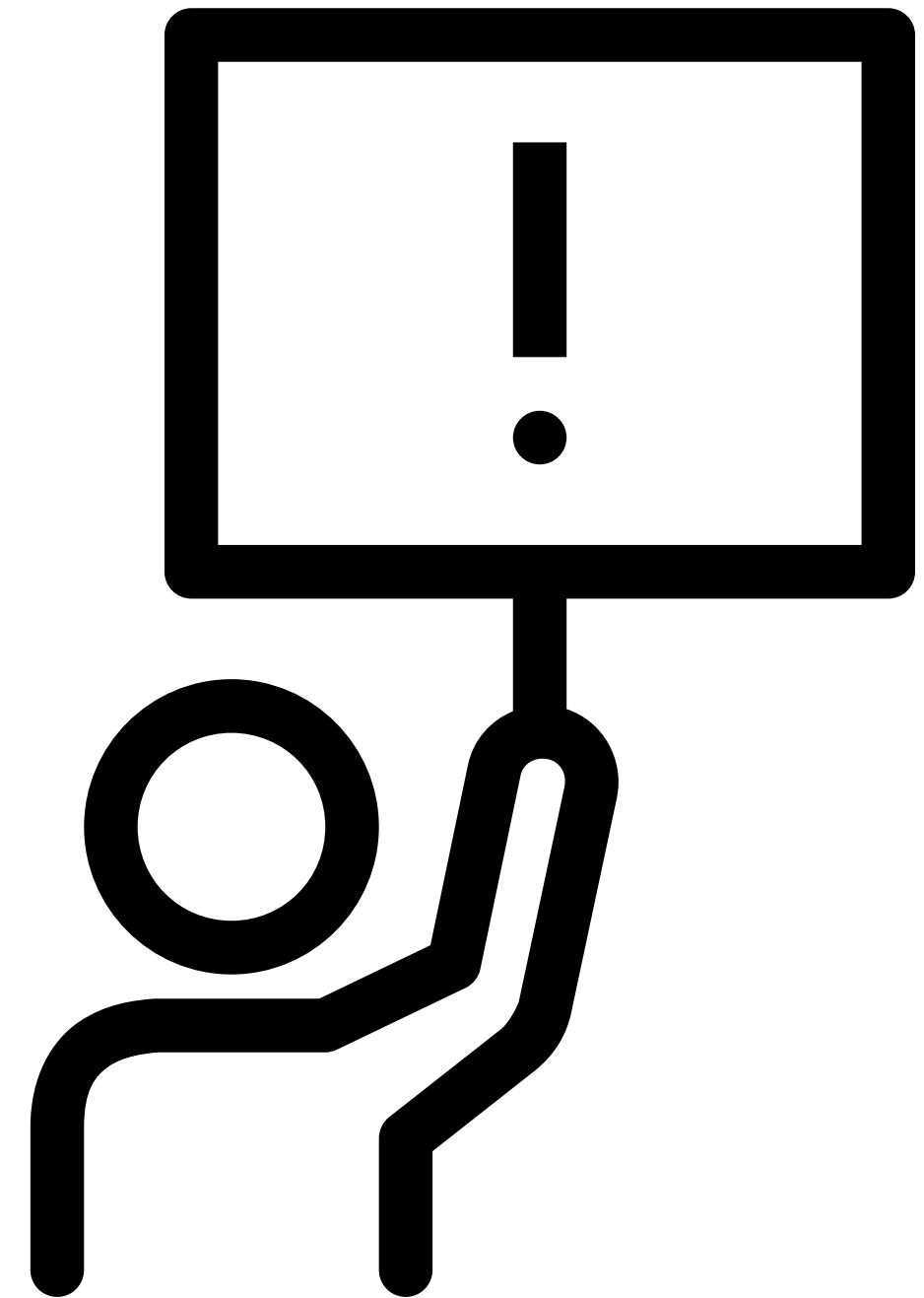
PROBLEMA

LOS DISPOSITIVOS WEARABLES DE ECG ENFRENTAN DESAFÍOS DE PRECISIÓN, CON RIESGOS DE FALSOS POSITIVOS O NEGATIVOS. EL PRINCIPAL PROBLEMA SON LOS ARTEFACTOS, ESPECIALMENTE LOS DE MOVIMIENTO (MA), QUE ALTERAN EL TRAZADO Y DIFICULTAN SU INTERPRETACIÓN. REDUCIR ESTOS ARTEFACTOS ES UNO DE LOS MAYORES RETOS EN EL MONITOREO AMBULATORIO.



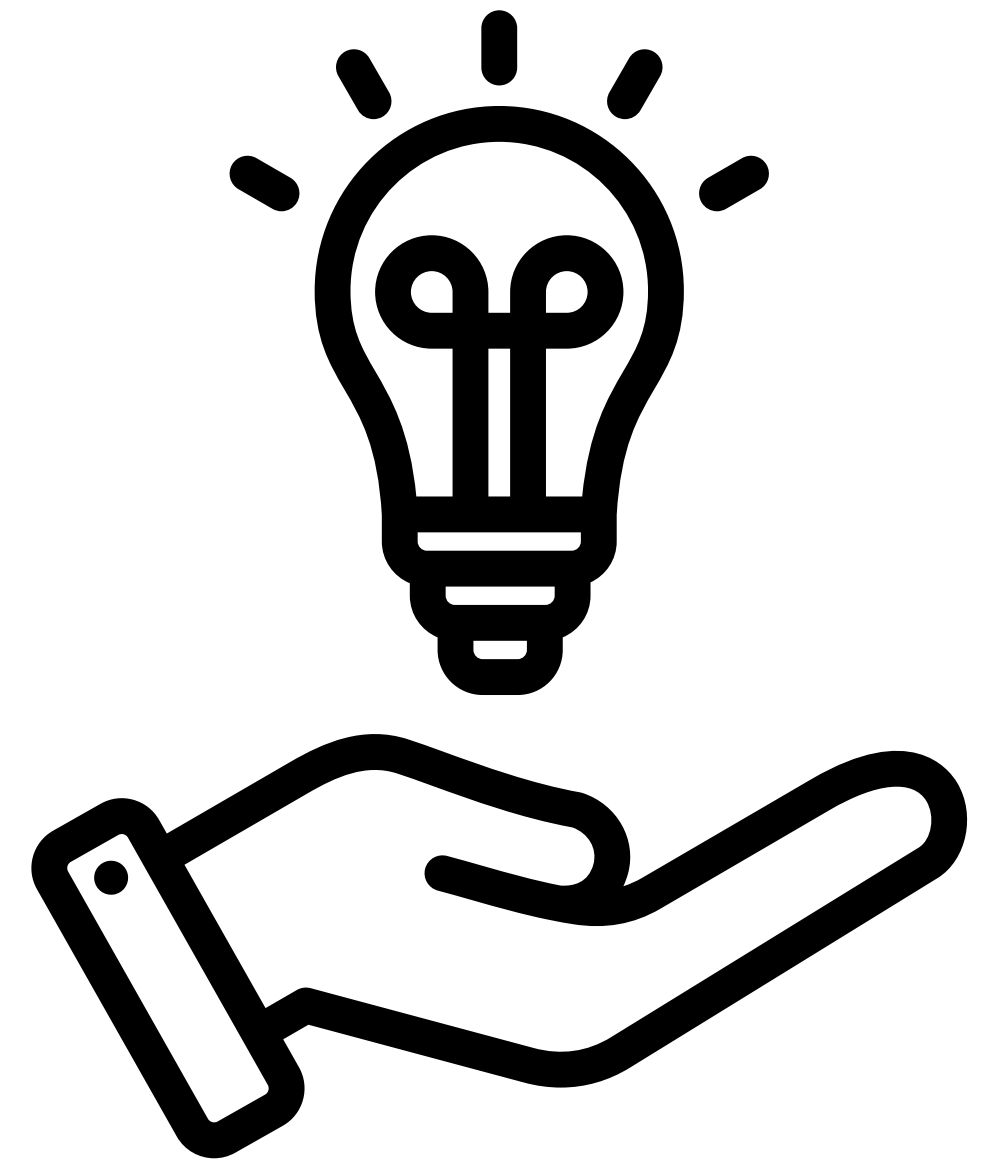
NECESIDAD

Carencia de softwares especializados para dispositivos cardiacos wearables que permitan distinguir las diferencias entre actividades rutinarias, como rascarse el pecho y cepillarse los dientes, con patologías a nivel de bioseñal



PROPUESTA DE SOLUCIÓN

Desarrollar un modelo de Machine Learning que permita diferenciar entre señales ECG alteradas por movimientos rutinarios (rascarse) y señales que reflejan patologías reales (arritmias)

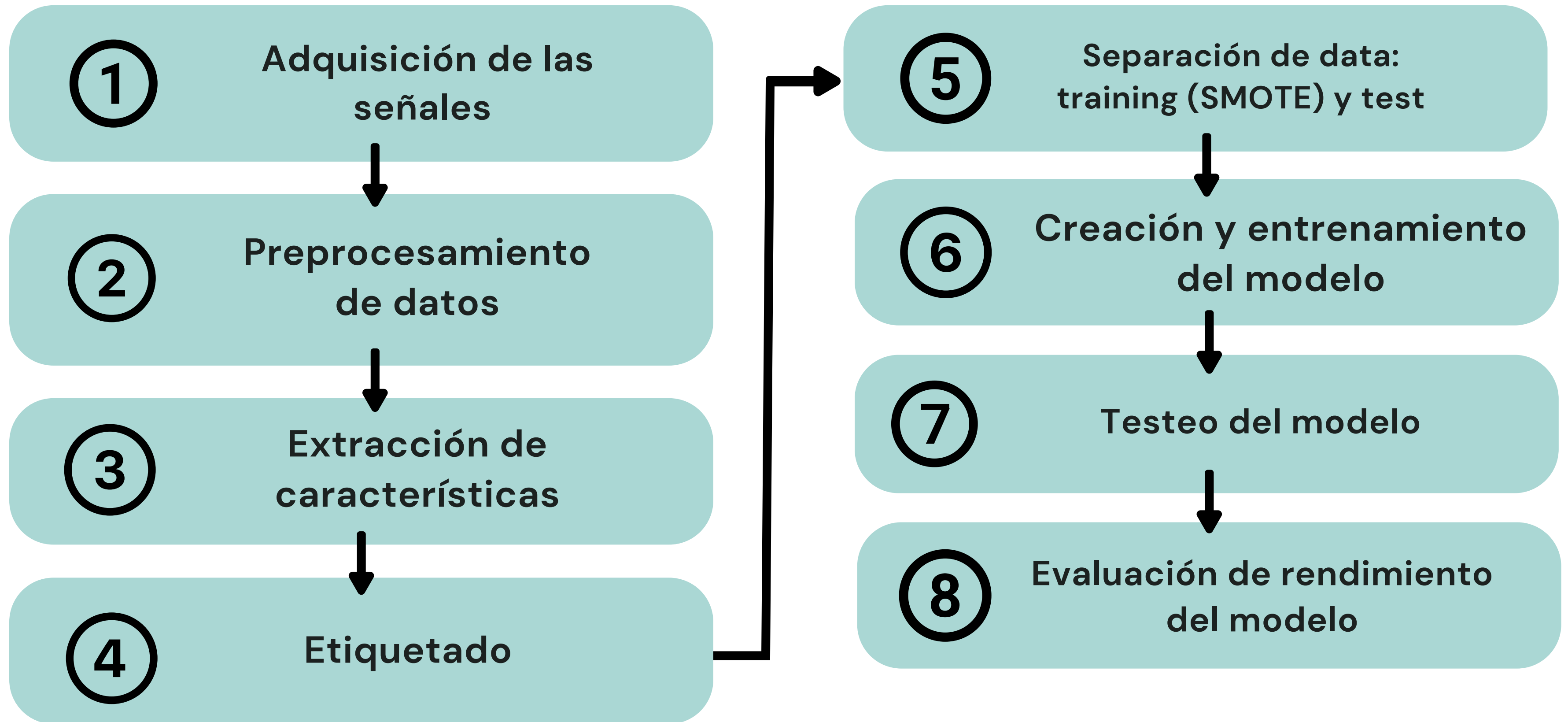


METODOLOGÍA

Random Forest

TABLE 7. SUMMARY OF PERFORMANCE RESULTS OF THE PROPOSED ECG CLASSIFICATION MODELS

Classifier Type	Avg. Accuracy %	Avg. Sensitivity %	Avg. Specificity %
RBF Network	92.00	91.90	90.80
Multilayer Perceptron	93.74	93.70	92.10
Logistic Model Trees	95.54	95.50	94.80
NR Tree	95.60	95.70	95.20
Random Forest Tree	97.45	97.50	95.90



Extracción de características

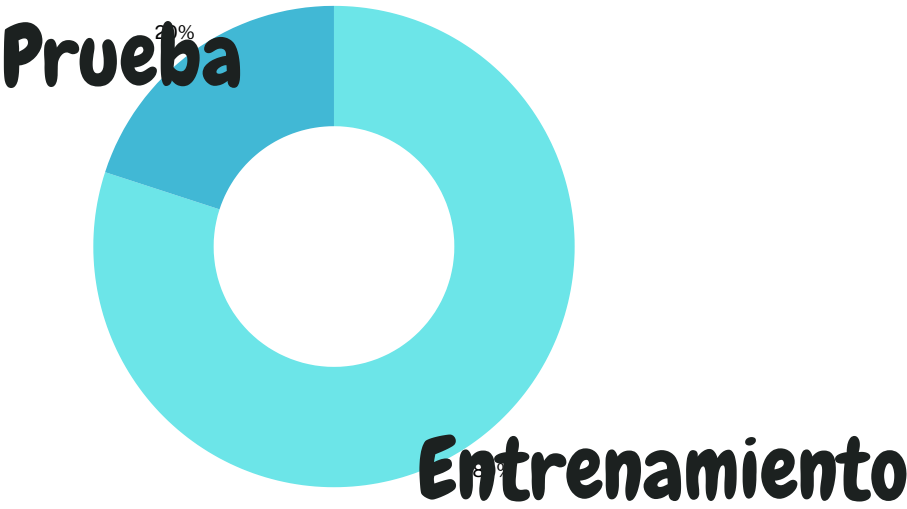
Table 1. ECG features set I and set II for classification

number 23 features(set I)		14 feature(set II)
1	P wave	✓
2	R wave	✓
3	R wave start point	✓
4	R wave end point	
5	T1 wave	✓
6	T2 wave	✓
7	P wave start point	
8	P wave end point	
9	T wave start point	
10	T wave end point	
11	PP	✓
12	QRS	✓
13	PR	✓
14	ST	✓
15	QT	✓
16	RR	✓
17	TT	
18	P	✓
19	T	
20	Total beat	✓
21	T1 T2	
22	ST	
23	PR	✓

Modelado

| División del conjunto de datos

Entrenamiento (80%)
Prueba (20%)



| Métricas de evaluación

Recall
Precision

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

$Precision = \frac{TP}{TP + FP}$

$Recall = \frac{TP}{TP + FN}$



PROTOTIPO

Base de datos

Physionet

Database Open Access

A large scale 12-lead electrocardiogram database for arrhythmia study

Jianwei Zheng , Hangyuan Guo , Huimin Chu 

Published: Aug. 24, 2022. Version: 1.0.0

Fs: 500 Hz
Duración: 10s
12 derivaciones



Base de datos

Kaggle



ARJUNASCAGNETTO · UPDATED 4 YEARS AGO

▲ 17

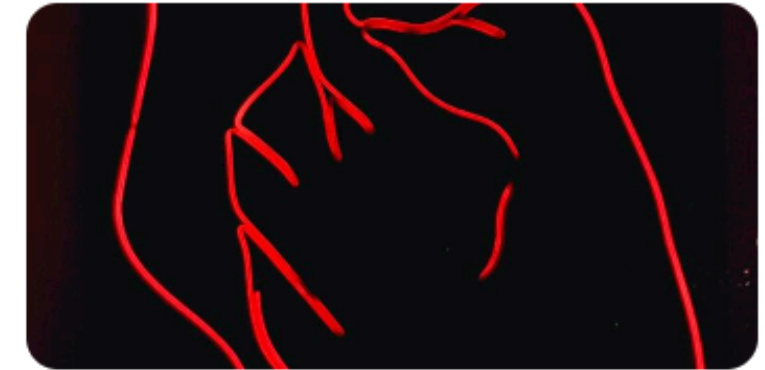
New Notebook

Download

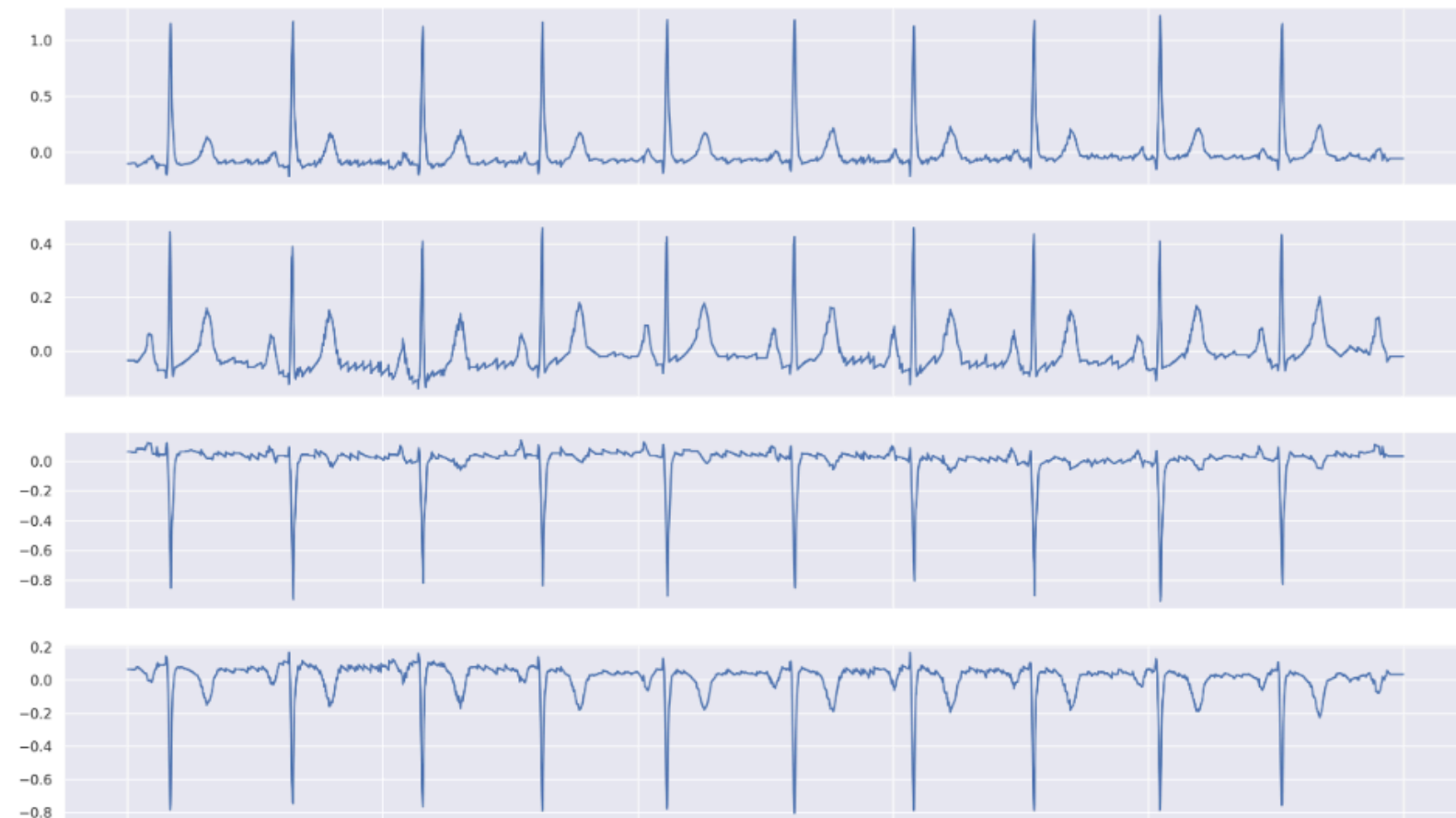


PTB-XL - Atrial Fibrillation Detection

Subset PTB-XL- 3 ecg rhythms: Normal, Atrial Fibrillation, all other arrhythmia.



- **6528** pacientes
- **10** segundos
- **500** Hz
- **12** derivaciones



Base de datos

Personas sanas con ECG , realizando movimiento de rascase



Fs: 500 Hz

Duración: 10s

3 Derivadas (DI,DII,DIII)

LECTURA DE SEÑALES

ESP32 - AD8232

```
✓ #include <Arduino.h>
  #include <XSpaceBioV10.h>

XSpaceBioV10Board MyBioBoard;

const int SAMPLE_RATE = 500;      // Frecuencia de muestreo (Hz)
const int SAMPLE_INTERVAL = 1000 / SAMPLE_RATE; // Intervalo entre muestras (ms)

// Variables de tiempo
unsigned long lastSampleTime = 0;

✓ void setup() {
  MyBioBoard.init();

  // Despertar los sensores AD8232
  MyBioBoard.AD8232_Wake(AD8232_XS1);
  MyBioBoard.AD8232_Wake(AD8232_XS2);

  // Inicializar comunicación serial
  Serial.begin(115200);
}

✓ void loop() {
  // Capturar la hora actual
  unsigned long currentTime = millis();

  // Tomar una muestra cada SAMPLE_INTERVAL ms
  ✓ if (currentTime - lastSampleTime >= SAMPLE_INTERVAL) {
    lastSampleTime = currentTime;

    // Leer derivaciones
    double DerivationI = MyBioBoard.AD8232_GetVoltage(AD8232_XS1);
    double DerivationII = MyBioBoard.AD8232_GetVoltage(AD8232_XS2);
    double DerivationIII = DerivationII - DerivationI;

    // Enviar datos al puerto serial
    Serial.println(String(DerivationI) + " " + String(DerivationII) + " " + String(DerivationIII));
  }
}
```

GUARDADO DE ARCHIVOS

```
import serial
import time

# Configuración del puerto serial
port = 'COM6' # Cambia según tu configuración
baud_rate = 115200 # Debe coincidir con el baud rate del microcontrolador
capture_time = 3 # Tiempo de captura en segundos

# Conexión al puerto serial
ser = serial.Serial(port, baud_rate)

def calculate_sampling_frequency(capture_time):
    """Calcula la frecuencia de muestreo basada en un tiempo de captura."""
    timestamps = []
    print(f"Leyendo datos durante {capture_time} segundos para calcular la frecuencia de muestreo...")

    start_time = time.time()
    while time.time() - start_time < capture_time:
        try:
            ser.readline().decode('utf-8').strip() # Leer datos (no se guardan)
            timestamps.append(time.time()) # Registrar tiempo de llegada de cada muestra
        except:
            continue # Ignorar errores de lectura

    # Calcular frecuencia de muestreo
    num_samples = len(timestamps) # Total de muestras recibidas
    total_time = timestamps[-1] - timestamps[0] # Tiempo total entre la primera y última muestra
    sampling_frequency = num_samples / total_time # Frecuencia de muestreo (Hz)

    return sampling_frequency, num_samples, total_time

try:
    # Calcular la frecuencia de muestreo
    fs, num_samples, total_time = calculate_sampling_frequency(capture_time)
    # Define el tiempo de duración de la captura en segundos
    capture_duration = 50
    start_time = time.time()

    #-----para establecer el nombre-----
    carpeta_ubicacion = "DATA/CUBA/"
    nombre_archivo = "rascado_4"
    #-----

    with open(carpeta_ubicacion+nombre_archivo+".txt", "w") as file:
        try:
            while time.time() - start_time < capture_duration: # Corre durante 10 segundos
                if ser.in_waiting > 0: # Revisa si hay datos disponibles
                    line = ser.readline().decode('utf-8').strip() # Lee y decodifica la línea
                    print(line) # Opcional: muestra los datos en la terminal
                    file.write(line + '\n') # Escribe la línea en el archivo
            except KeyboardInterrupt:
                print("Detenido por el usuario")
            finally:
                ser.close()
            print(f"\nResultados:")
            print(f"Frecuencia de muestreo estimada: {fs:.2f} Hz")
            print(f"Número total de muestras: {num_samples}")
            print(f"Tiempo total de captura: {total_time:.2f} segundos")
        except Exception as e:
            print(f"Error: {e}")
    finally:
        if ser.is_open:
            ser.close()
```

ACONDICIONAMIENTO DE LA SEÑAL

```
def butter_bandpass(lowcut, highcut, fs, order=5):  
    nyquist = 0.5 * fs # Frecuencia de Nyquist  
    low = lowcut / nyquist  
    high = highcut / nyquist  
    b, a = butter(order, [low, high], btype='band')  
    return b, a  
  
def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):  
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)  
    y = filtfilt(b, a, data)  
    return y
```

#Filtro IIR: NOTCH

```
def notch_filter(data, fs, freq=60.0, Q=20.0):  
    w0 = freq / (fs / 2) # Normalizar la frecuencia  
    b, a = iirnotch(w0, Q)  
    filtered_signal = filtfilt(b, a, data)  
    return filtered_signal, b, a
```

Specs b,a P/Z Info Fixpoint

LOAD > Ellip. LP (default) > SAVE

Bandpass IIR Butterworth

Order: ☐ Minimum $N = 5$

DESIGN FILTER

Quit

Freq. Unit Hz 0...1/2

$f_s = 500$

Target Specifications

Frequency in Hz

$f_{SB} = 50$ Ripple in dB

$f_{PB} = 100$ $A_{SB} = 60$

$f_{PB2} = 150$ $A_{PB} = 2$

$f_{SB2} = 200$

Frequency Specs in Hz

$f_c = 0.5$

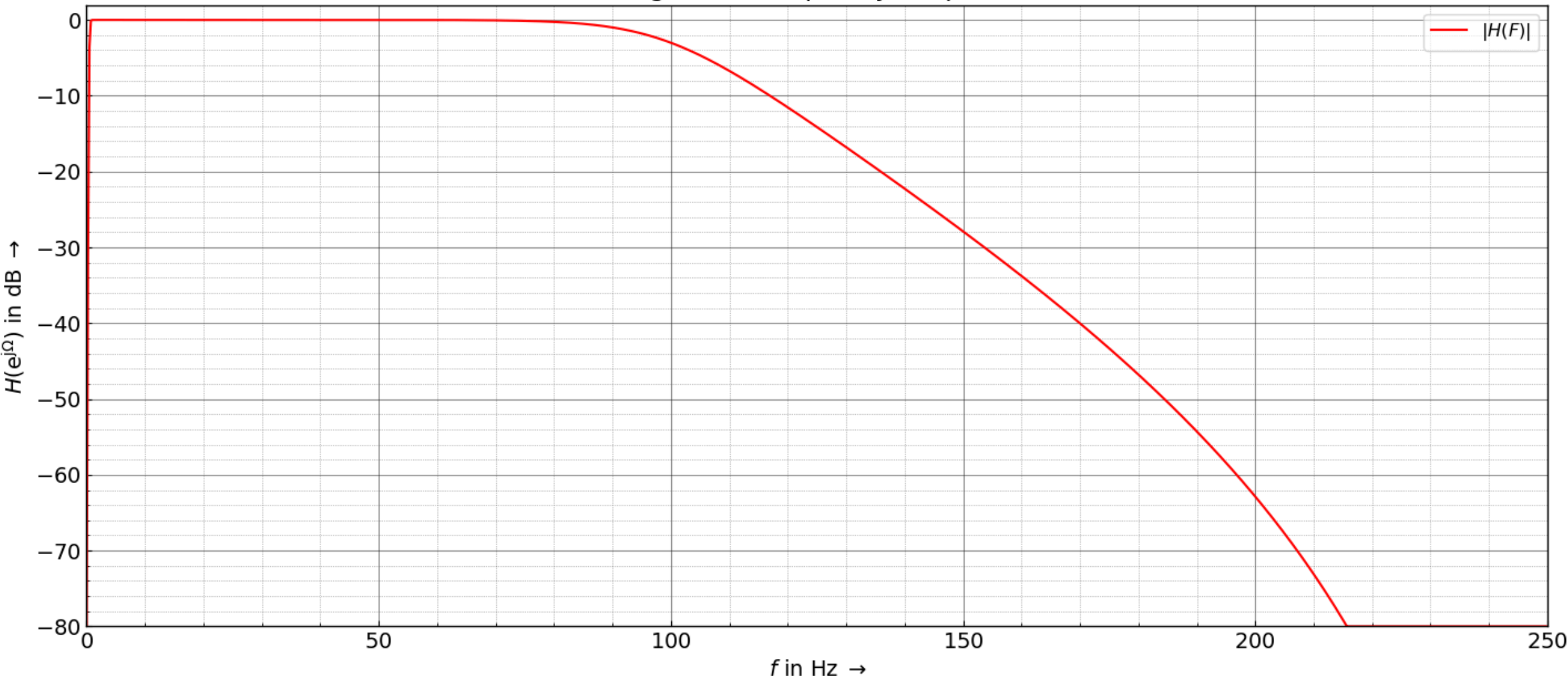
$f_{c2} = 100$

Enter the filter order N and the -3 dB corner frequency(ies) F_c .

|H(f)| $\phi(f)$ $\tau(f)$ P/Z y[n] 3D



Magnitude Frequency Response



|H| ☒
re{H} ☐
im{H} ☐

Unit: Auto

min = -80.0 dB

Zero phase

Inset off

Specs

Phase

AVANCE DEL CÓDIGO DE ML (RANDOM FOREST)

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
4 from imblearn.over_sampling import SMOTE
5 import seaborn as sns
6 import matplotlib.pyplot as plt
7
8 # Dividir dataset en datos de entrenamiento y testeo
9 X_train, X_test, y_train, y_test = train_test_split(features, etiquetas, test_size=0.3, random_state=42)
10
11 # Aplicar SMOTE al conjunto de entrenamiento para balancear
12 smote = SMOTE(random_state=42)
13 X_train, y_train = smote.fit_resample(X_train, y_train)
14
15 model = RandomForestClassifier(n_estimators=200, random_state=42) #Creacion del modelo
16 model.fit(X_train, y_train) # Entrenar el modelo con los datos balanceados
17
18 y_pred = model.predict(X_test) # Testeo del modelo con la data reservada para el test
19
20 # Evaluar rendimiento
21 accuracy = accuracy_score(y_test, y_pred)
22 print(f"Precisión del modelo: {accuracy:.2f}")
23 print("\nReporte de Clasificación:")
24 print(classification_report(y_test, y_pred))
25
26 # Matriz de confusión
27 conf_matrix = confusion_matrix(y_test, y_pred)
28 print("\nMatriz de Confusión:")
29 print(conf_matrix)
30
31 plt.figure(figsize=(6, 6))
32 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Arritmia', 'No arritmia'], yticklabels=['Arritmia', 'No arritmia'])
33 plt.xlabel('Predicción')
34 plt.ylabel('Valor Real')
35 plt.title('Matriz de Confusión')
36 plt.show()
```

**Thank you
very much!**