

Análisis de datos para aplicaciones médicas

Jose A. Govea-García¹, Augusto Ley Rodríguez² and Angel Esparza Enriquez³

¹ Tecnológico de Monterrey, Escuela de Ingeniería y Ciencia, Jalisco, Mexico

Reception date of the manuscript: 10/05/2025

Acceptance date of the manuscript: 10/05/2025

Publication date: 10/05/2025

Abstract— Este proyecto explora el uso de sensores integrados en teléfonos móviles para reconocer actividades físicas en tiempo real mediante modelos de clasificación supervisada. Se recolectaron señales de movimiento de distintos participantes al realizar una serie de ejercicios, y se entrenaron varios modelos de aprendizaje automático para predecir la actividad correspondiente. A lo largo del proyecto se aplicaron técnicas de validación cruzada, optimización de hiperparámetros y selección de características con el objetivo de mejorar el rendimiento de los clasificadores. La versión final se integró en una aplicación funcional que recibe datos en tiempo real desde un celular. Se concluye que una correcta combinación de modelos, ajustes y reducción de variables es clave para implementar soluciones eficientes y estables en contextos prácticos.

Keywords— Modelos de clasificación, evaluación de modelos, recolección de datos, optimización, aplicaciones médicas

I. INTRODUCCIÓN

En los últimos años, el uso de sensores integrados en dispositivos como relojes, teléfonos, audífonos o anillos ha cobrado gran relevancia. Esta tecnología ha abierto un abanico de oportunidades para el monitoreo de la actividad humana, permitiendo a personas, entrenadores o especialistas en salud contar con información valiosa para la planificación y ejecución de actividades físicas, el seguimiento del estado de salud o cualquier otro aspecto relevante relacionado con el bienestar.

Aplicaciones comerciales como Google Fit, Apple Health o Fitbit ya aprovechan estos sensores para identificar patrones de movimiento y clasificar actividades como caminar, correr o subir escaleras. Sin embargo, dichas soluciones suelen estar limitadas a un conjunto reducido de acciones predefinidas. Estudios recientes destacan el valor de personalizar estas herramientas para fines específicos, lo que abre la puerta a soluciones más especializadas desarrolladas por usuarios o investigadores [1].

El objetivo de este trabajo es poder hacer uso de los sensores integrados en el celular para, en tiempo real, detectar qué tipo de actividad física estamos realizando. Para lograrlo, utilizaremos modelos de clasificación que nos permitan etiquetar los datos recolectados dependiendo de la actividad. Estos modelos serán entrenados y optimizados para que, una vez utilizados, puedan asignar de manera acertada la actividad que se está realizando.

El desarrollo del proyecto se dividirá en las siguientes eta-

pas:

1. Recolección de datos
2. Evaluación de modelos de clasificación
3. Optimización de modelos
4. Clasificación en línea
5. Elaboración del reporte

II. METODOLOGÍA

En esta sección se describe el proceso completo seguido para desarrollar el sistema de clasificación de actividades físicas. Desde la recolección de datos utilizando sensores del teléfono móvil, hasta el entrenamiento, evaluación y optimización de distintos modelos. La metodología se divide en cuatro etapas principales: recolección de datos, procesamiento y extracción de características, evaluación de modelos, y optimización con selección de variables.

a. Recolección de Datos

El primer paso que realizamos fue la recolección de datos. Para ello, utilizamos la aplicación Phyphox, la cual permite acceder a distintos sensores integrados en el celular. Para facilitar y automatizar la recolección, desarrollamos un script en Python que nos permitió ajustar todos los parámetros necesarios del experimento. Además, se aprovechó la funcionalidad de Phyphox para transmisión en tiempo real mediante IP, lo cual permitió al script obtener los datos en formato obj y almacenarlos directamente.

Primeramente, definimos las condiciones del experimento, en donde escogimos seis distintas actividades:

1. Jumping jacks
2. Saludar
3. Saltar cuerda

4. Boxear
5. Rotación de tronco
6. Parado

A continuación, configuramos los parámetros necesarios del experimento, desde la cantidad de pruebas por actividad hasta el número de muestras registradas en cada prueba. Una vez definidos los parámetros, procedimos a realizar la recolección de datos. Cada integrante del equipo llevó a cabo el experimento en dos sesiones, por razones que se explicarán más adelante en este reporte.

En la primera sesión, se utilizó exclusivamente el acelerómetro, capturando datos en los ejes X, Y y Z. En la segunda sesión, además del acelerómetro, se emplearon los sensores del giroscopio, también en los ejes X, Y y Z.

b. Procesamiento y Características Extraídas

Una vez recolectados los datos, se procedió con su procesamiento previo a la etapa del modelado. Para cada ventana de señal, se extrajeron múltiples características que nos permiten resumir el comportamiento del movimiento en esa ventana de tiempo. Estas características permiten mejorar la capacidad del modelo para diferenciar entre actividades.

Las siguientes características fueron extraídas para cada eje:

1. Media
2. Desviación estándar
3. Curtosis
4. Asimetría (Skewness)
5. Mediana
6. Valor máximo
7. Valor mínimo
8. Rango total (Peak-to-peak)
9. Rango intercuartílico (IQR)
10. Cruces por cero (Zero Crossing Rate)
11. Energía
12. Contribución al RMS general

Mientras que las siguientes características se extrajeron de la magnitud de aceleración y magnitud del giroscopio:

1. Media
2. Desviación estándar
3. Energía

Para extraer estas características se utilizó un script de Python, el cual abre el archivo obj, realiza el cálculo de las características para cada ventana de señal y nos entrega los resultados en un archivo txt.

c. Evaluación de Modelos

Con las características calculadas, el siguiente paso del proyecto fue la evaluación de modelos de clasificación. Se realizaron tareas de limpieza inicial, como la eliminación de registros incompletos o en blanco. Posteriormente, el conjunto de datos se dividió en variables independientes (X) y la variable objetivo (Y).

Antes de entrenar los modelos, se aplicó un proceso de estandarización de los datos, utilizando escalado estándar para garantizar que todas las características tuvieran la misma escala, lo cual es especialmente importante para modelos sensibles a la magnitud de las variables.

La evaluación de modelos se realizó mediante validación cruzada (cross-validation) para poder utilizar la totalidad de los datos y que los resultados fueran más robustos. Específicamente, se utilizaron medidas como accuracy, precision y recall para evaluar el rendimiento de los distintos clasificadores.

En total, se evaluaron 10 modelos de clasificación, cinco modelos vistos en clase:

1. SVM
2. SVM base radial
3. LDA
4. K-NN
5. MLP (2 capas)

Y cinco modelos adicionales fueron investigados y probados por nuestra cuenta:

1. Extra trees
2. Random forest
3. Regresión logística
4. Impulso por gradiente
5. Naive bayes

Para llevar a cabo la evaluación de los modelos de clasificación, se trabajó en un notebook en Python donde se implementaron los algoritmos previamente mencionados. Entre los modelos probados, se seleccionaron cuatro algoritmos representativos por su relevancia teórica y diversidad metodológica: SVM, K-NN, MLP y Naive Bayes. Cada uno de ellos ofrece un enfoque distinto para abordar el problema de clasificación.

SVM opera proyectando los datos en un espacio de mayor dimensión, lo que permite separar clases que no serían distinguibles linealmente en su forma original. En ese nuevo espacio, identifica un hiperplano que actúa como límite entre las categorías. Una vez entrenado, el modelo puede usar las características de nuevos datos para predecir a qué clase pertenecen [2].

TABLE 1: REPORTE DE CLASIFICACIÓN PARA SVM (RBF)

Clase	Precisión	Recall	F1-Score
1	0.97	0.99	0.98
2	1.00	0.97	0.98
3	1.00	1.00	1.00
4	1.00	0.99	0.99
5	0.97	0.94	0.95
6	0.95	1.00	0.98
Accuracy			0.98
Macro avg	0.98	0.98	0.98
Weighted avg	0.98	0.98	0.98

El modelo SVM con núcleo radial mostró un rendimiento altamente consistente, alcanzando valores de F1-score cercanos o iguales a 1.00 en la mayoría de las clases, con una exactitud general del 98%.

K-NN clasifica un nuevo registro comparándolo con los ejemplos más cercanos en el conjunto de entrenamiento. La decisión se basa en la mayoría de clases entre sus k vecinos más próximos, determinados mediante una medida de distancia. Este modelo no requiere un proceso de entrenamiento complejo y es especialmente útil cuando se busca una solución intuitiva y basada en similitudes [3].

**TABLE 2:** REPORTE DE CLASIFICACIÓN PARA K-NN

Clase	Precisión	Recall	F1-Score
1	0.99	0.99	0.99
2	1.00	0.98	0.99
3	0.99	1.00	1.00
4	1.00	0.99	0.99
5	0.98	0.94	0.96
6	0.94	1.00	0.97
Accuracy			0.98
Macro avg	0.98	0.98	0.98
Weighted avg	0.98	0.98	0.98

K-NN logró una precisión destacada en todas las clases, con especial solidez en la clase 3 y 6, alcanzando un F1-score perfecto (1.00) en algunas de ellas. La exactitud global fue también del 98%.

El Perceptrón Multicapa (MLP) es una red neuronal artificial que se compone de al menos una capa oculta entre la entrada y la salida. Cada capa está formada por unidades llamadas neuronas, que están completamente conectadas con la siguiente capa. El MLP entrena el modelo utilizando el algoritmo de retropropagación del error, ajustando los pesos internos para minimizar la diferencia entre las predicciones y las etiquetas verdaderas. Esta arquitectura permite aprender relaciones complejas y no lineales entre las variables de entrada y salida [4].

TABLE 3: REPORTE DE CLASIFICACIÓN PARA MLP (2 CAPAS)

Clase	Precisión	Recall	F1-Score
1	0.98	0.99	0.99
2	1.00	0.97	0.98
3	0.99	1.00	1.00
4	0.99	0.98	0.98
5	0.96	0.95	0.95
6	0.95	0.98	0.97
Accuracy			0.98
Macro avg	0.98	0.98	0.98
Weighted avg	0.98	0.98	0.98

El perceptrón multicapa obtuvo un desempeño sólido, con resultados equilibrados entre clases y un F1-score máximo de 1.00 en la clase 3. La exactitud general del modelo fue del 98%.

El clasificador de Bayes Ingenuo (Naive Bayes) es un algoritmo basado en el Teorema de Bayes que asume independencia estadística entre las características. Estima la probabilidad de que un dato pertenezca a una clase específica considerando cada atributo por separado, lo que lo hace eficiente y escalable incluso con conjuntos de datos grandes o de alta dimensión [5].

TABLE 4: REPORTE DE CLASIFICACIÓN PARA NAIVE BAYES

Clase	Precisión	Recall	F1-Score
1	0.98	0.99	0.99
2	1.00	0.98	0.99
3	1.00	1.00	1.00
4	1.00	1.00	1.00
5	0.98	0.95	0.96
6	0.95	0.99	0.97
Accuracy			0.98
Macro avg	0.99	0.98	0.99
Weighted avg	0.99	0.98	0.99

El modelo Naive Bayes alcanzó una exactitud del 98% y F1-scores de 1.00 en varias clases, destacando por su eficiencia y capacidad de generalización.

d. Optimización y Selección de Variables

Una vez evaluado el rendimiento general de los modelos de clasificación, se procedió a optimizar aquellos que mostraron mejores resultados. En esta etapa, se ajustaron hiperparámetros clave y se aplicaron técnicas de selección de características con el objetivo de mejorar el rendimiento y reducir la complejidad del modelo. Los modelos se escogieron son SVM radial y MLP.

Se seleccionaron dos modelos para la optimización: SVM radial y MLP, debido a su alto desempeño. Para ambos modelos, se definieron los hiperparámetros más relevantes y se analizaron diferentes combinaciones, utilizando validación cruzada para evaluar el rendimiento de cada configuración.

Como primer paso en el proceso de optimización, nos enfocamos exclusivamente en ajustar los hiperparámetros de los modelos seleccionados, dejando la selección de variables para una etapa posterior.

Para el modelo SVM con núcleo radial (RBF), se exploraron diferentes valores del parámetro C, observándose que el mejor desempeño se alcanzó con $C = 0.1$. Sin embargo, las variaciones en el valor de accuracy entre configuraciones fueron mínimas, situándose todas en torno al 98.3–98.5%.

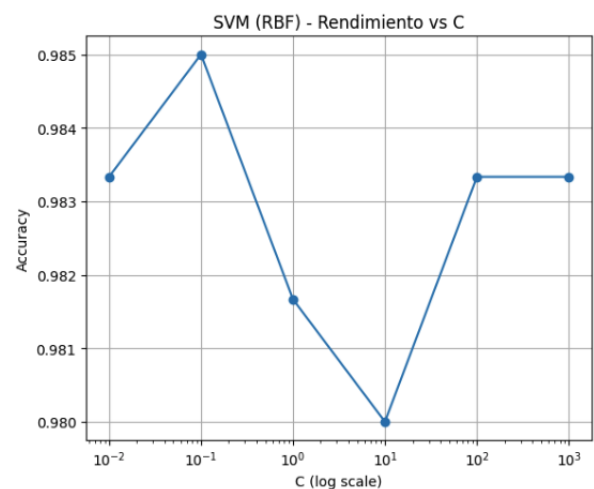


Fig. 1: Rendimiento del modelo SVM con núcleo RBF para distintos valores del hiperparámetro C.

En el caso del Perceptrón Multicapa (MLP), se probaron cinco configuraciones distintas de capas ocultas. Sorprenden-

temente, la arquitectura más sencilla (10,) ofreció la mayor precisión, mientras que la configuración (50,), que en principio arrojó un resultado ligeramente inferior, fue finalmente la seleccionada para la demostración en vivo.

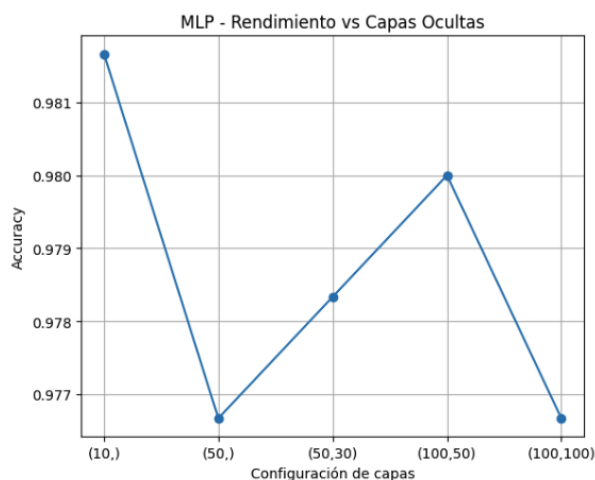


Fig. 2: Rendimiento del modelo MLP con diferentes configuraciones de capas ocultas.

Esta decisión respondió a observaciones realizadas durante la fase de prueba en tiempo real. Aunque la arquitectura (10,) obtuvo la mayor precisión en validación cruzada, en la demostración en vivo mostró un comportamiento inadecuado: el modelo no era capaz de reconocer correctamente las seis actividades, limitándose a predecir únicamente una o dos clases. En contraste, la configuración (50,), que había registrado una precisión ligeramente inferior (una diferencia menor al 0.2%), logró identificar adecuadamente todas las clases durante la ejecución práctica.

Como segundo paso del proceso de optimización, se aplicó una técnica de selección univariada de características mediante SelectKBest con la función de puntuación f_{classif} . El objetivo fue identificar el número mínimo de variables necesarias para mantener un rendimiento alto, reduciendo así la complejidad del modelo y mejorando su eficiencia computacional.

Se evaluaron distintos valores de k (número de características seleccionadas), específicamente: 5, 10, 15, 20, 25, 30, 35 y 39. Para cada valor de k , se llevó a cabo una validación cruzada utilizando los modelos previamente optimizados: SVM con $C=10$ y MLP con configuración de capas ocultas (50,).

En el caso de SVM, el rendimiento fue creciendo de forma consistente hasta alcanzar su punto máximo con 39 características, logrando una exactitud de 99.0% (véase Figura 3). No obstante, se observaron valores muy similares a partir de las 25 características, lo cual sugiere que es posible reducir la dimensionalidad sin afectar significativamente el desempeño.

Por otro lado, MLP alcanzó su mejor rendimiento a partir de las 30 características, manteniendo una exactitud estable de 98.6% con cualquier valor superior (véase Figura 4). Llama la atención que incluso con solo 10 características, el modelo ya lograba una precisión superior al 98.3%, lo cual resalta la capacidad del algoritmo para adaptarse a espacios de menor dimensión sin pérdida de generalización.

Estos resultados evidencian que tanto SVM como MLP pueden beneficiarse de una reducción en el número de vari-

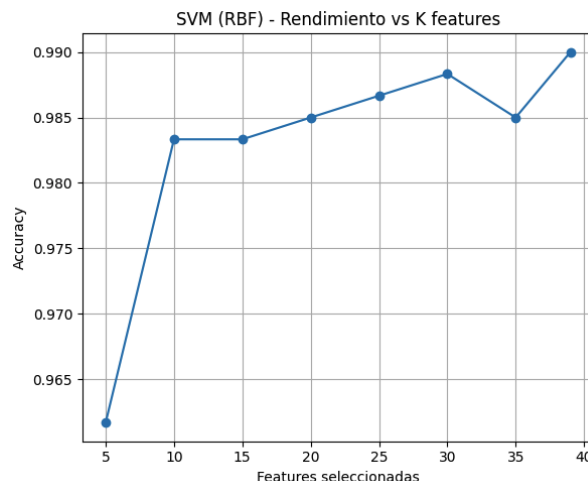


Fig. 3: Rendimiento del modelo SVM (RBF) con la configuración $C = 10$, utilizando diferentes cantidades de características seleccionadas.

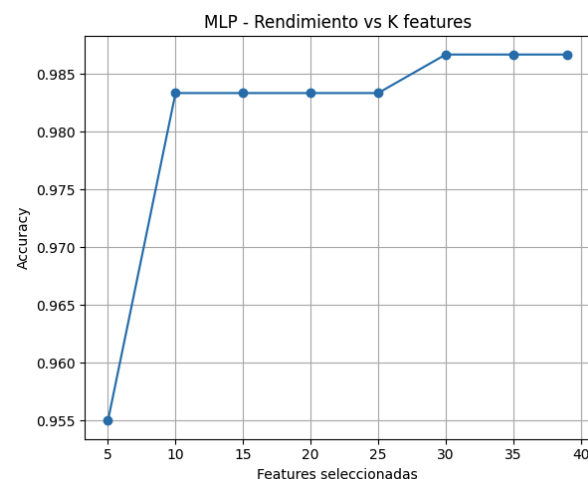


Fig. 4: Rendimiento del modelo MLP con configuración de capas ocultas (50,) utilizando diferentes cantidades de características seleccionadas.

ables, siempre que se mantengan aquellas con mayor capacidad predictiva. Esto no solo mejora la eficiencia, sino que también puede facilitar la implementación práctica del modelo.

e. Modelo para la App

Para la implementación en la aplicación en tiempo real, se construyó un modelo optimizado utilizando un pipeline de scikit-learn. Este pipeline incluye tres componentes principales: StandardScaler para normalizar los datos, SelectKBest para seleccionar las 10 características más relevantes y un MLPClassifier con arquitectura (50,).

Se aplicó una búsqueda en malla (GridSearchCV) con validación cruzada estratificada para optimizar los hiperparámetros: tamaño de capas ocultas, α (regularización) y $\text{learning_rate_init}$. El mejor modelo, con alto rendimiento promedio, fue serializado con joblib para su uso directo en la aplicación.



III. RESULTADOS


El modelo seleccionado fue integrado exitosamente en la aplicación de clasificación en tiempo real. Durante las pruebas en vivo, el sistema fue capaz de detectar correctamente los seis ejercicios definidos, con tiempos de respuesta adecuados para su uso inmediato tras la captura de datos.

Para esta versión en vivo, se utilizó un script diseñado específicamente para recibir los datos del celular en tiempo real, de forma similar al proceso de recolección original. La señal se capturó desde la aplicación Phyphox, que permitió obtener datos del acelerómetro y giroscopio directamente desde el dispositivo móvil mediante conexión local.

Aunque el modelo basado en MLP con una sola capa oculta de 50 neuronas no fue el que obtuvo la precisión más alta durante la validación cruzada, demostró ser el más estable y confiable en condiciones reales. Las configuraciones más complejas tendían a sobreajustarse y, en pruebas en vivo, no lograban distinguir adecuadamente los seis ejercicios, clasificando incorrectamente múltiples repeticiones en una o dos clases.

En contraste, el modelo seleccionado logró mantener una precisión en tiempo real cercana al 98%, con una respuesta rápida y sin pérdidas de desempeño. Estos resultados respaldan su robustez y viabilidad para su uso en aplicaciones interactivas, validando el pipeline diseñado para el despliegue final.

Es importante mencionar que, si bien se recolectaron datos de los tres integrantes del equipo durante la etapa experimental, el modelo fue entrenado específicamente con los datos de uno de ellos para su implementación en tiempo real. Esto se debe a que, para que el sistema funcione de manera ideal, el modelo debe ser entrenado con datos representativos de la persona que realizará las actividades en vivo. Aunque el sistema conserva cierta capacidad de generalización, pueden existir diferencias sutiles en la forma de ejecutar los ejercicios entre individuos, lo que puede afectar la precisión del reconocimiento. Esta observación subraya la importancia de desarrollar futuras versiones del modelo que contemplen mecanismos de personalización o estrategias más robustas de generalización para ampliar su aplicabilidad.

Para más detalles, consulte el repositorio del proyecto. 

REFERENCES

- [1] Harvard Health Publishing. (2021) Exercising with your smartphone: How activity trackers and health apps help. <https://www.health.harvard.edu/staying-healthy/exercising-with-your-smartphone-how-activity-trackers-and-health-apps-help>. Consultado el 8 de junio de 2025.
- [2] IBM Corporation. (2024) Cómo funciona svm (support vector machine). <https://www.ibm.com/docs/es/spss-modeler/saas?topic=models-how-svm-works>. Consultado el 8 de junio de 2025.
- [3] I. Corporation. (2024) ¿qué es knn? <https://www.ibm.com/mx-es/think/topics/knn>. Consultado el 8 de junio de 2025.
- [4] Scikit-learn Developers, *Neural network models (supervised)*, https://scikit-learn.org/stable/modules/neural_networks_supervised.html#mlp, scikit-learn, 2024, consultado el 8 de junio de 2025.
- [5] IBM Corporation. (2024) Naive bayes. <https://www.ibm.com/think/topics/naive-bayes>. Consultado el 8 de junio de 2025.