TP2 Métodos Computacionales

Josefina Casas Pardo, Paula Ho y Denis Wu

07 de julio 2025

1 Introducción

El presente trabajo práctico aborda el diseño e implementación de un clasificador binario capaz de distinguir entre personas sanas y pacientes con Parkinson, un trastorno neurodegenerativo cuyo diagnóstico temprano puede verse favorecido por el análisis automatizado de patrones motores finos. En nuestro caso, basaríamos nuestro análisis a través del trazado de espirales.

Para llevar a cabo este objetivo, se plantea una estrategia basada en técnicas de aprendizaje supervisado, buscando optimizar una función no lineal de tal manera que cometa el mínimo error posible. Así, implementaremos dos modelos muy conocidos, el Descenso por Gradiente y el Ascenso por Gradiente, entrenando con un conjunto de imágenes etiquetadas como "Healthy" o "Parkinson", para luego predecir las etiquetas de otras imágenes no consideradas en el modelo (testing).

El trabajo se organiza en distintas secciones que desarrollan progresivamente la solución al problema. En primer lugar, se desarrollará el pre-procesamiento de las imágenes dadas, para luego explicar la implementación de los dos modelos mencionados, su entrenamiento, evaluación y análisis de resultados. Finalmente, compararíamos ambos modelos midiendo la eficiencia en cada caso.

2 Pre-procesamiento de imágenes

Previo a la implementación de los modelos, necesitamos transformar las imágenes originales en una representación numérica compatible con los métodos de optimización. Cargar todas las imágenes en escala RGB puede generar una carga considerable en la memoria, por lo que en este contexto, convertimos las imágenes a una escala de grises, utilizando la fórmula estándar de NTSC:

$$\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{1}$$

donde cada variable RGB simboliza la magnitud de ese color en un píxel determinado.

Finalmente, la imagen en escala de grises se aplana a un vector unidimensional que conserva la información estructural de los píxeles, lo que permite tratarla como un vector en $\mathbb{R}^K$, siendo $K = \text{size}^2$. De esta manera, construimos una matriz $\mathbb{X} \in \mathbb{R}^{N \times K}$ donde cada fila representa una imagen vectorizada. Al mismo tiempo, se construye el vector de etiquetas $\mathbf{y} \in \{0, 1\}^N$, asignando el valor 0 a los casos etiquetados como "Healthy" y 1 a los que corresponden a "Parkinson".

3 Método de Descenso por Gradiente

Algoritmo. Partiendo desde algún punto inicial $x_1^{(0)}, \cdots, x_n^{(0)}$. El descenso por gradiente consiste en actualizar iterativamente el punto moviéndose en la dirección opuesta al gradiente hasta alcanzar algún mínimo local de la función. La **regla de actualización** para cada coordenada es:

$$\left(x_1^{(i+1)} \ \cdots \ x_n^{(i+1)}\right) := \left(x_1^{(i)} \ \cdots \ x_n^{(i)}\right) - \alpha \nabla g(x_1^{(i)}, \ldots, x_n^{(i)}). \tag{2}$$

Iterativamente, repetimos los siguientes pasos hasta llegar alguna convergencia, o hasta alcanzar a un número fijo de iteraciones:

i. Calcular el gradiente de la función en el punto actual.
ii. Mover el punto siguiendo la regla de actualización.

Aplicando a nuestro caso, el objetivo es encontrar una función predictora de las imágenes minimizando el error cuadrático medio (ECM) entre lo que predecimos y lo que sucede verdaderamente en la realidad. Siguiendo con esto, proponemos entonces buscar la mejor solución dentro de las funciones $f : \mathbb{R}^K \to (0, 1)$ que tengan la forma:

$$f_{\mathbf{w},b}(\mathbf{i}) = \frac{\tanh(\mathbf{w} \cdot \mathbf{i} + b) + 1}{2}, \tag{3}$$

donde $\mathbf{w}$ es un vector de $\mathbb{R}^K$ que indica la importancia del píxel para la detección de Parkinson, b un escalar que ajusta el sesgo, y tanh la tangente hiperbólica.

Nuestro problema entonces se reduce a encontrar $\mathbf{w}$ y b tal que minimicen la Ecuación 3. Podemos reescribir el problema:

$$\arg\min_{\mathbf{w},b} \mathcal{L}(\mathbf{w}, b) = \arg\min_{\mathbf{w},b} \frac{1}{N} \sum_{i=1}^{N} \left(f_{\mathbf{w},b}(\mathbf{i}_i) - d_i\right)^2. \tag{4}$$

Esta función tendrá mínimos en aquellos lugares donde las derivadas se anulen:

$$\frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{N} \sum_{i=1}^{N} \left(f_{\mathbf{w},b}(\mathbf{i}_i) - d_i\right)^2 \right) = 0 \tag{5}$$

$$\frac{\partial}{\partial b} \left(\frac{1}{N} \sum_{i=1}^{N} \left(f_{\mathbf{w},b}(\mathbf{i}_i) - d_i\right)^2 \right) = 0 \tag{6}$$

Utilizando la herramienta `MatrixCalculus.org`, obtuvimos las siguientes expresiones:

$$\frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{N} \sum_{i=1}^{N} \left(f_{\mathbf{w},b}(\mathbf{i}_i) - d_i\right)^2 \right) = \frac{1}{N} \sum_{i=1}^{N} (1 - t_{0,i}^2) \cdot \left(\frac{1 + t_{0,i}}{2} - d_i \right) \cdot \mathbf{i}_i \tag{7}$$

$$\frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{N} \sum_{i=1}^{N} \left(f_{\mathbf{w},b}(\mathbf{i}_i) - d_i\right)^2 \right) = \frac{1}{N} \sum_{i=1}^{N} (1 - t_{0,i}^2) \cdot \left(\frac{1 + t_{0,i}}{2} - d_i \right) \tag{8}$$

donde

$$t_0 = \tanh\left(b + \mathbf{w}^\top \mathbf{i}\right)$$

Con el cálculo de las derivadas parciales, partimos inicialmente de un vector $\mathbf{w}$ aleatorio y un valor b iniciado en cero. A partir de allí, aplicamos la regla de actualización iterativamente hasta alcanzar un número máximo de iteraciones prefijado (nosotros lo seteamos en 1000), o hasta que se cumpla el criterio de convergencia:

$$\left| f_{\mathbf{w}^{(k+1)},b^{(k+1)}}(\mathbf{i}_i) - f_{\mathbf{w}^{(k)},b^{(k)}}(\mathbf{i}_i) \right| < \epsilon$$

con un ϵ predeterminado de 1×10^{-6}.

Hechas estas aclaraciones, ya podemos empezar a implementar el método de Descenso por Gradiente. Para el entrenamiento del modelo, trabajamos con un conjunto de 1500 imágenes, conformado por 750 imágenes etiquetadas como "Healthy" y 750 como "Parkinson", garantizando así un equilibrio en las clases que favorece el entrenamiento supervisado del modelo. Luego, para el testing del modelo, utilizamos las restantes 888 imágenes, donde cada mitad está construida por una de las dos etiquetas.

A medida que entrenamos el modelo, comparamos la evolución del ECM y la accuracy de los resultados, entre el conjunto de entrenamiento y el conjunto de testing. El siguiente gráfico representa lo obtenido:

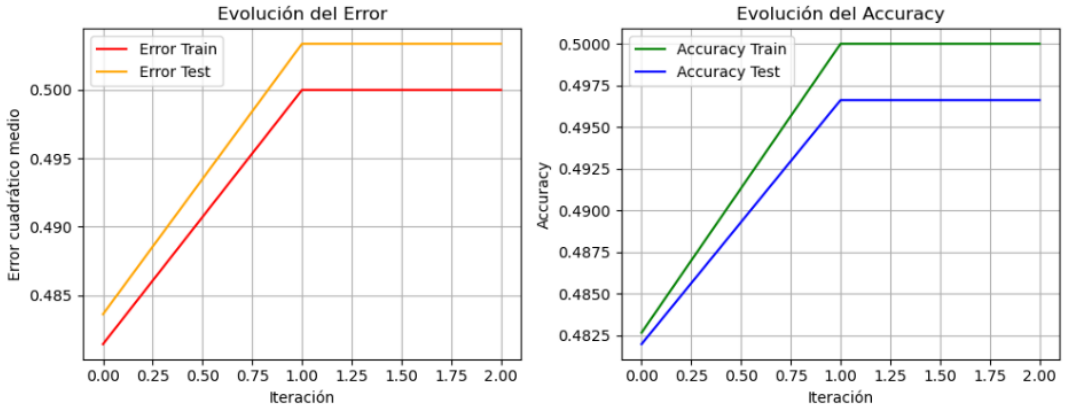

Figure 1: Evolución del ECM y accuracy en cada conjunto de imágenes

Observemos que el modelo resulta súper ineficiente, dado que incluso llega a converge ya desde la segunda iteración, no logrando mejorar la accuracy. ¿Por qué pasa esto? La razón es simple, al no normalizar la imagen, los valores de la matriz X pueden tomar valores entre 0 y 255. Al trabajar con una función de tanh, la aproximación de los valores se acercan a los extremos con números tan grandes, por lo que el modelo no se ajusta de manera correcta.

Si ahora normalizamos la matriz, los valores estarían acotados entre 0 y 1, con lo que obtendríamos los siguientes resultados:

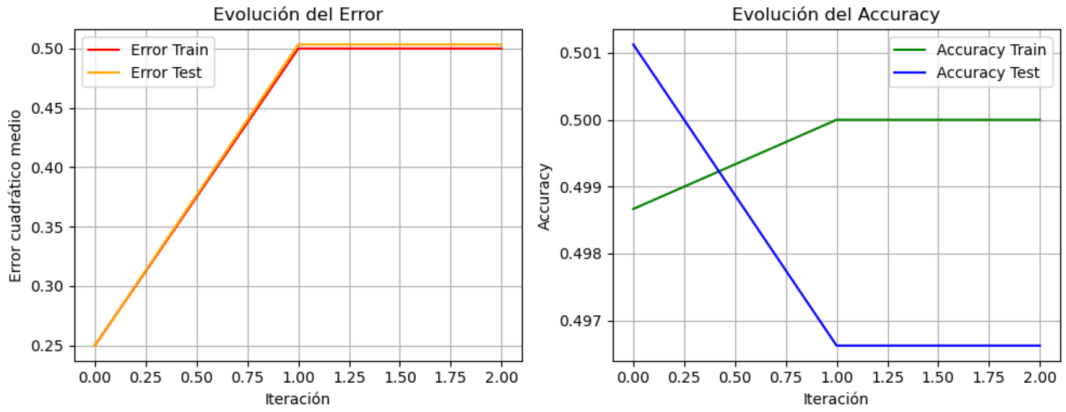

Figure 2: Evolución del ECM y accuracy en cada conjunto de imágenes, normalizando la matriz X

Pareciese ser el mismo gráfico, ¿qué es lo que está afectando ahora? Lo que observamos es el impacto del parámetro α (Ecuación 2), que debe ser ajustado manualmente.

Dicho parámetro afecta directamente nuestro modelo en cuestiones de performance, pero también de convergencia. Una selección incorrecta de este valor puede hacer que el algoritmo fracase completamente en su búsqueda del mínimo. Si el valor fuese muy grande, podría llegar a converger demasiado rápido, como el caso anterior. Y si fuese muy chico, podría tardar muchas iteraciones hasta llegar a encontrar algún mínimo óptimo.

Veamos qué obtenemos al considerar distintos valores de α en el modelo:

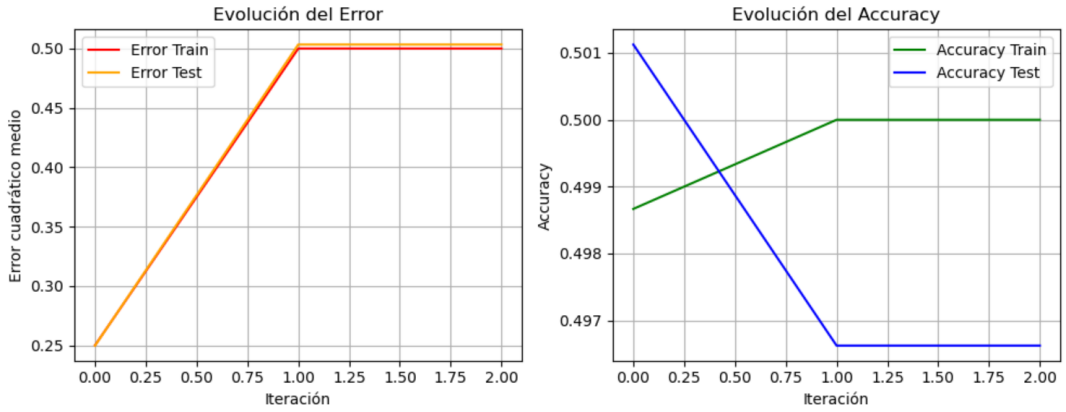

Figure 3: Evolución del ECM y accuracy utilizando a = 0.1

Este valor de α es elevado, y aunque produce una convergencia casi inmediata (en solo 2 iteraciones), los resultados obtenidos no son satisfactorios. La accuracy se estabiliza cerca de 0.50, lo cual indica un comportamiento cercano al azar, además de que el error cuadrático se estanca rápidamente en un valor alto. Esto sugiere que el modelo se sobreajusta rápidamente y que la tasa de aprendizaje es demasiado grande.

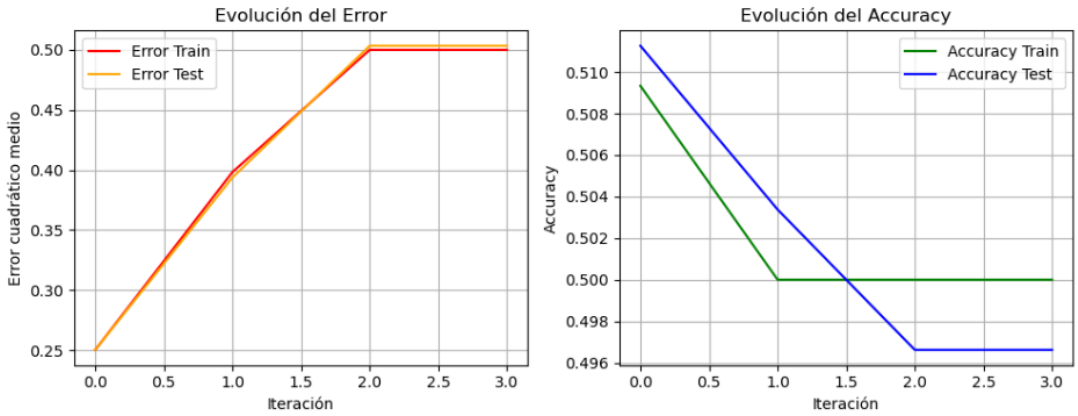

Figure 4: Evolución del ECM y accuracy utilizando a = 0.01

Al reducir el valor de α, la convergencia sigue siendo rápida (en 3 iteraciones), y no logra una mejora relevante en las métricas. Sigamos reduciendo el α...

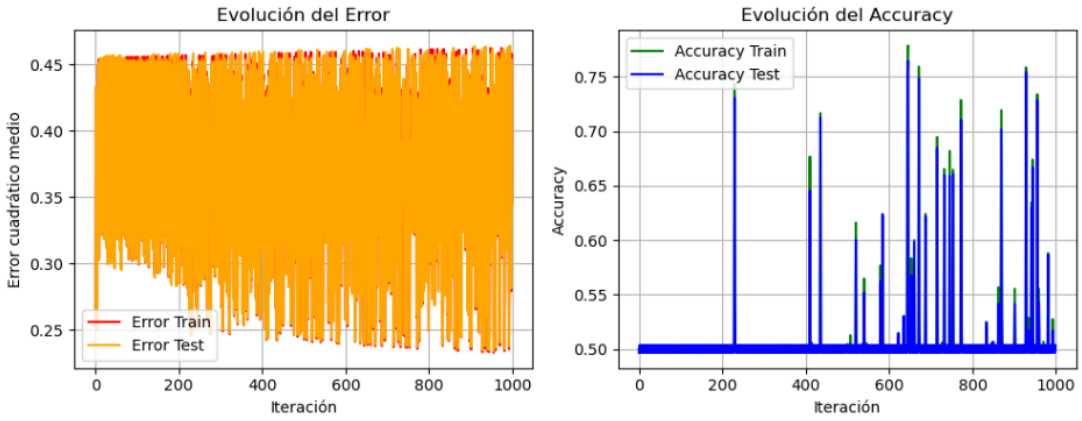

Figure 5: Evolución del ECM y accuracy utilizando a = 0.001

El modelo no logra todavía converger adecuadamente. Tanto el error como la accuracy oscilan constantemente, tal como se ve reflejado en ambas curvas. Se evidencia una falta de estabilidad y una gran sensibilidad a las actualizaciones de los parámetros, lo cual indica que el descenso es demasiado vulnerable a mínimos locales.

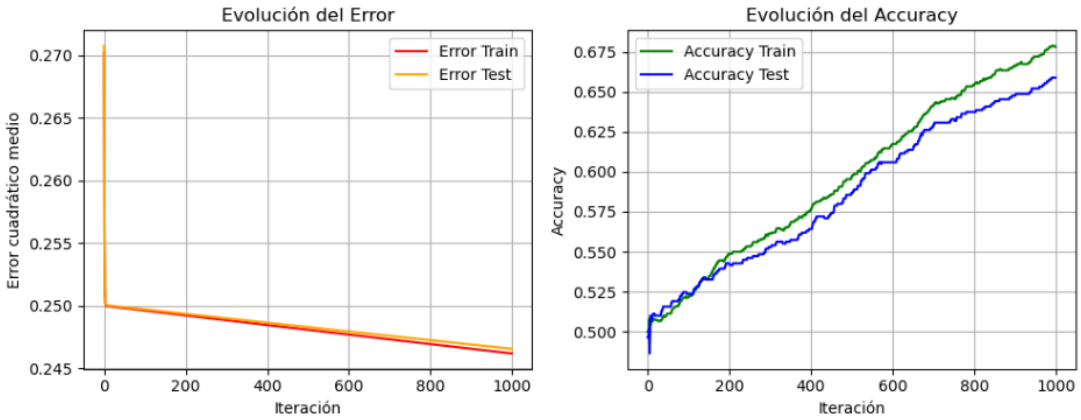

Figure 6: Evolución del ECM y accuracy utilizando a = 0.0001

Este caso representa una estabilidad adecuada entre velocidad y estabilidad. Aunque se requieren más iteraciones (el modelo no alcanza convergencia estricta), el descenso del error es constante y sostenido. Además, se observa una mejora significativa en la accuracy, que asciende por encima de 0.65 en el conjunto de test. Esto indica que el modelo logra captar mejor los patrones del conjunto de datos sin sobreajustarse.

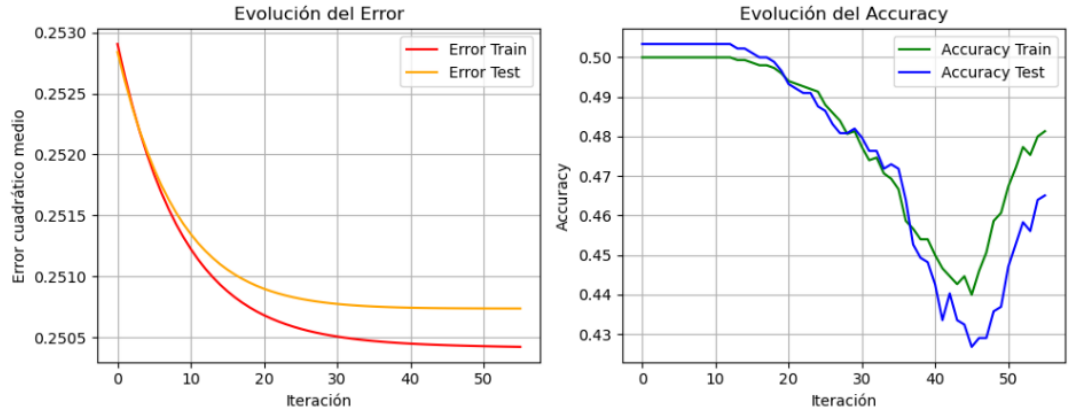

Figure 7: Evolución del ECM y accuracy utilizando a = 0.00001

Con una tasa de aprendizaje extremadamente baja, si bien logra alcanzar un error cuadrático medio bajo, la accuracy no presenta mejoras respecto al valor de α anterior. De hecho, observamos que curiosamente la accuracy baja al principio, y sube luego por la iteración 40, hasta que eventualmente converge y se da por finalizado el algoritmo. En síntesis, concluimos en que el mejor α que ajusta el modelo es 0.0001.

Analicemos ahora el impacto de otra variable fundamental, el escalado de imágenes. Hasta el momento, estuvimos utilizando un size de 128x128.

Nuestro objetivo ahora es analizar cómo influye el tamaño de entrada de las imágenes en dos dimensiones clave: la efectividad del modelo (en términos de exactitud y error cuadrático medio)

y el tiempo requerido para entrenarlo. Para ello, se evaluaron cuatro tamaños de imagen: 32, 64, 128 y 256 píxeles, utilizando el mismo conjunto de datos y manteniendo constantes los parámetros de aprendizaje ($\alpha = 0.0001$, repeticiones $= 1000$).

En términos de tiempo de cómputo, observamos los siguientes resultados:

1. Para size = 32x32: 141.17s
2. Para size = 64x64: 135.53s
3. Para size = 128x128: 131.34s
4. Para size = 256x256: 127.00s

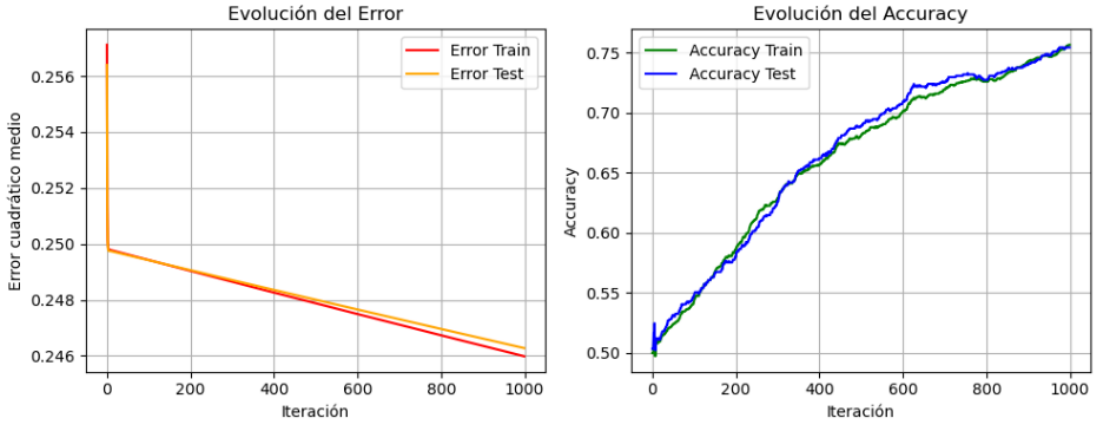

Figure 8: Evolución del ECM y accuracy utilizando size = 32x32

El modelo mostró una mejora progresiva durante el entrenamiento, alcanzando una exactitud final sobre el conjunto de test de 0.7545 y un error cuadrático de 0.2463. El tiempo de cómputo fue de 141.17 segundos, siendo sorprendentemente el más alto entre todos los tamaños evaluados. Esto sugiere que, aunque el tamaño disminuido de imagen parece requerir menos procesamiento por instancia, al reducir mucho la resolución se pierde estructura visual y se genera más ruido visual. Esto hace que el modelo tarde más en aprender a distinguir lo importante de lo irrelevante, y necesite más iteraciones efectivas para converger.

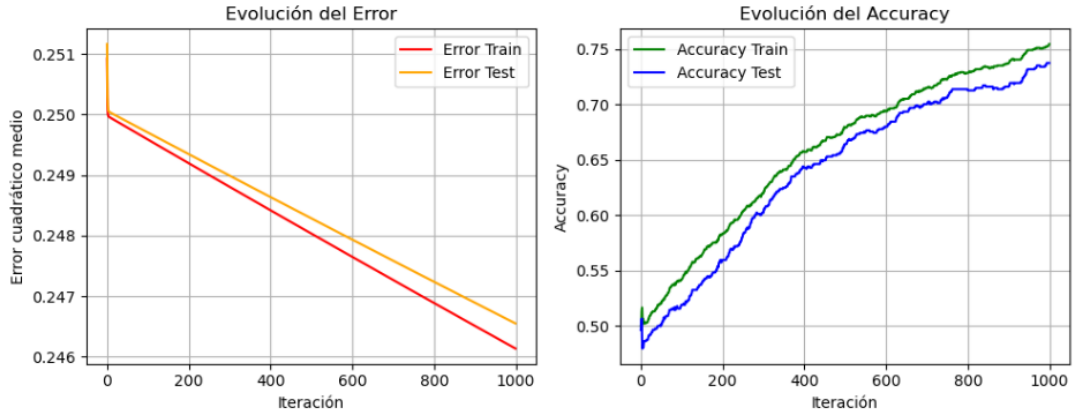

Figure 9: Evolución del ECM y accuracy utilizando size = 64x64

Con una resolución mayor, el modelo logró una accuracy de test similar (0.7376) y un error cuadrático final comparable (0.2461), pero con un leve descenso en el tiempo total de entrenamiento (135.53 segundos). Esto podría atribuirse a una representación más balanceada entre detalle visual y cantidad de información procesada, optimizando el tiempo sin afectar significativamente el rendimiento.

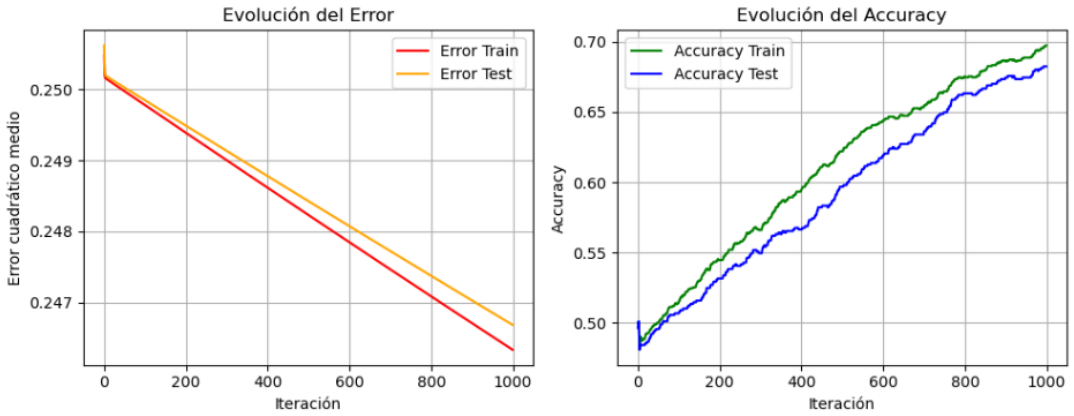

Figure 10: Evolución del ECM y accuracy utilizando size = 128x128

Este tamaño de imagen arrojó resultados competitivos con una accuracy de test de 0.6824 y ECM final de 0.2467, pero redujo aún más el tiempo de cómputo (131.34 segundos). Observamos también una mayor fluctuación de los datos, aunque ciertamente siempre termina reajustando de manera correcta las métricas.

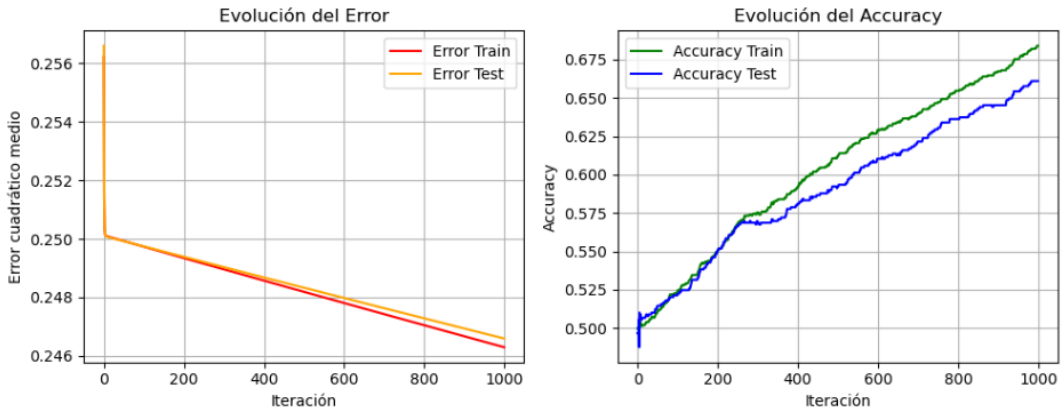

Figure 11: Evolución del ECM y accuracy utilizando size = 256x256

Finalmente, con el tamaño más grande evaluado, el modelo obtuvo una accuracy de test de 0.6610 y un error cuadrático medio de 0.2466, manteniéndose en valores similares de pérdida pero con un descenso visible en la precisión. Es destacable que fue el más eficiente en cuanto al tiempo (127.00 segundos). Esto podría explicarse por el hecho de que, al contener mayor cantidad de información, cada imagen permite al modelo realizar actualizaciones más significativas. En consecuencia, se requieren menos iteraciones efectivas para aprender, lo que reduce el tiempo total de entrenamiento.

El análisis evidencia que el tamaño del escalado de las imágenes impacta tanto en la efectividad como en la eficiencia del método. Los tamaños más pequeños (como 32×32, 64x64) presentan

tiempos de entrenamiento y exactitudes mayores, mientras que sucede lo contrario en los tamaños más grandes. Con lo visto, el tamaño 64×64 obtiene mejores resultados respecto a lograr exactitud en el conjunto de test minimizando el tiempo de cómputo simultáneamente. Por tanto, se concluye que 64×64 representa el tamaño óptimo dentro del rango evaluado.

Finalmente, terminamos eligiendo como óptimo un $\alpha = 0.0001$, utilizando imágenes de tamaño 64×64, pues presentaron el mejor compromiso entre exactitud y estabilidad en la convergencia. Con dichos valores, generamos las matrices de confusión para los conjuntos de entrenamiento y testing.

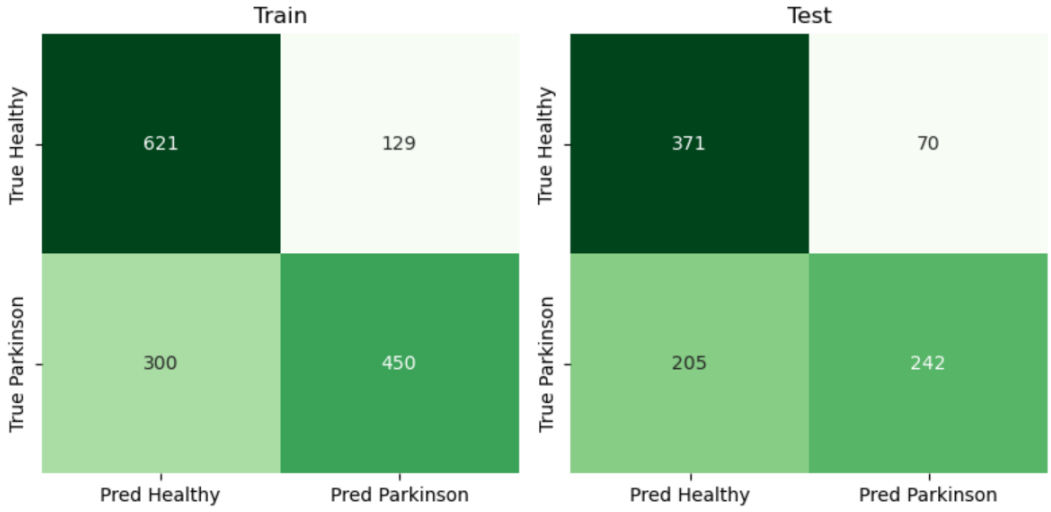

Figure 12: Matrices de confusión utilizando alpha = 0.0001 y size = 64x64

Como podemos ver en las visualizaciones, el modelo logra identificar correctamente una buena proporción de las clases tanto en entrenamiento como en test. En el conjunto de entrenamiento, se alcanzaron 621 verdaderos negativos y 450 verdaderos positivos, mientras que en test se mantuvo un desempeño razonable, con 371 negativos correctamente clasificados y 242 positivos identificados. Sin embargo, aún se evidencian errores en ambas clases, con un número considerable de falsos negativos (205) y falsos positivos (70) en test.

Estos resultados sugieren que el modelo es moderadamente efectivo, con una tendencia a clasificar correctamente más casos ““Healthy”” que ““Parkinson”’. La razón de esto podría deberse a características más marcadas en la clase negativa o a un sesgo en la representación de las imágenes, aunque podríamos seguir mejorando el modelo aumentando el número límite de iteraciones.

4 Método de Ascenso por Gradiente

Una alternativa al enfoque tradicional de Descenso por Gradiente consiste en aplicar Ascenso por Gradiente, que tiene como objetivo maximizar directamente la función de log-verosimilitud del modelo en lugar de minimizar una función de pérdida, definido como

$$\log \mathcal{L}(f) = \sum_{i=1}^{N} [d_i \log(f(\mathbf{i}_i)) + (1 - d_i) \log(1 - f(\mathbf{i}_i))]$$

siendo

$$f_{\mathbf{w},b}(\mathbf{i}) = \frac{1}{1 + e^{-(\mathbf{w}\cdot\mathbf{i}+b)}}$$

Este enfoque resulta especialmente adecuado en contextos de clasificación binaria, ya que permite interpretar las salidas del modelo como probabilidades. Nuestro problema se reduce entonces a encontrar los parámetros w y b que maximizan la log-verosimilitud, o equivalentemente, que minimizan la log-verosimilitud multiplicada por -1:

$$\arg\min_{\mathbf{w},\,b} \mathcal{L}(\mathbf{w}, b) = -\sum_{i=1}^{N} \left[d_i \log\left(f_{\mathbf{w},b}(\mathbf{i}_i)\right) + (1 - d_i) \log\left(1 - f_{\mathbf{w},b}(\mathbf{i}_i)\right)\right]$$

Al igual que en el Método de Descenso por Gradiente, empezamos por calcular las derivadas parciales con respecto a w y b, utilizando `MatrixCalculus.org`:

$$\frac{\partial}{\partial \mathbf{w}}(\arg\min_{\mathbf{w},\,b} \mathcal{L}(\mathbf{w}, b)) = \frac{t_0}{t_1} \cdot \mathbf{y} \cdot \mathbf{x}^\top - t_0 \left(t_1^2 \cdot \left(1 - \frac{1}{t_1}\right) \cdot (\mathbf{1} - \mathbf{y}) \cdot \mathbf{x}^\top \right)$$

$$\frac{\partial}{\partial b}(\arg\min_{\mathbf{w},\,b} \mathcal{L}(\mathbf{w}, b)) = \frac{t_0}{t_1} \cdot \mathbf{y} - t_0 \left(t_1^2 \cdot \left(1 - \frac{1}{t_1}\right) \cdot (\mathbf{1} - \mathbf{y}) \right)$$

donde

$$t_0 = \exp\left(-(b + \mathbf{w}^\top \cdot \mathbf{x})\right), \qquad t_1 = 1 + t_0$$

Al igual que en el método de Descenso por Gradiente, dividimos las derivadas parciales por la cantidad de columnas de X, es decir size^2, para trabajar directamente con la media de los datos.

Implementamos entonces el Ascenso por Gradiente, aplicando como regla de actualización:

$$\left(x_1^{(i+1)} \ \cdots \ x_n^{(i+1)}\right) := \left(x_1^{(i)} \ \cdots \ x_n^{(i)}\right) + \alpha \, \nabla g(x_1^{(i)}, \ldots, x_n^{(i)}). \tag{9}$$

La metodología de la experimentación fue similar a la del anterior método. A partir de un conjunto de 1500 imágenes igualmente distribuidos, entrenamos el modelo para luego testearlo con otro set de 888 imágenes distintas, midiendo la evolución de la accuracy y los valores de -log-verosimilitud para cada caso. Utilizando el valor de α óptimo encontrado en el Descenso por Gradiente (0.0001), y un size de 64x64 con un límite de iteraciones de 1000, obtuvimos los siguientes resultados:

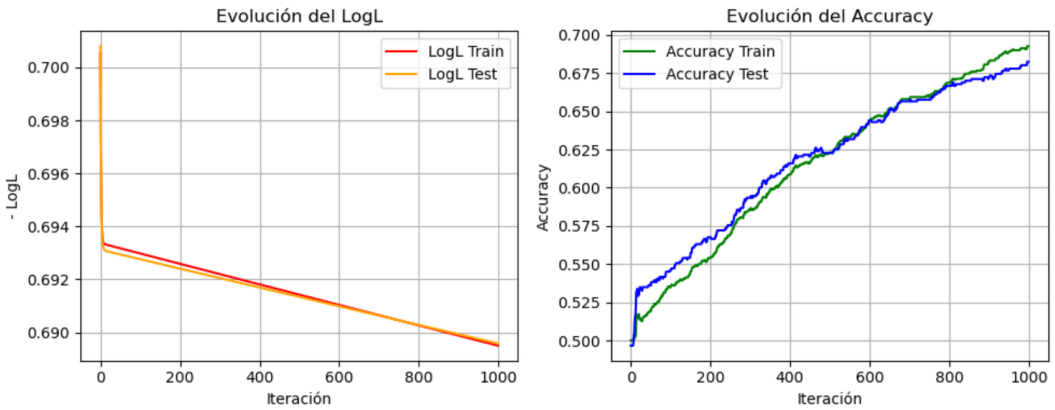

Figure 13: Evolución del -logL y accuracy en cada conjunto de imágenes

Como puede observarse en los gráficos generados, la log-verosimilitud negativa muestra una tendencia decreciente clara en ambos conjuntos, lo cual indica que el modelo logra mejorar su capacidad de ajuste de manera progresiva. En el caso del conjunto de entrenamiento, el valor inicial se aproxima a 0.7 y desciende hasta alcanzar aproximadamente 0.689 hacia el final de las iteraciones. En el conjunto de prueba, el comportamiento es similar, con valores levemente más altos, lo que resulta esperable dado que los datos no fueron utilizados durante la optimización.

En cuanto a la accuracy, se observa una mejora sostenida desde valores cercanos al 0.5 hasta alcanzar aproximadamente el 0.692 en entrenamiento y el 0.682 en testing. Este comportamiento refleja una buena capacidad del modelo para aprender la tarea de clasificación y, al mismo tiempo, generalizar adecuadamente a datos no vistos.

Analicemos ahora el impacto de α en la convergencia de este método, utilizando los dos mejores α encontrados anteriormente: 0.0001 y 0.00001.

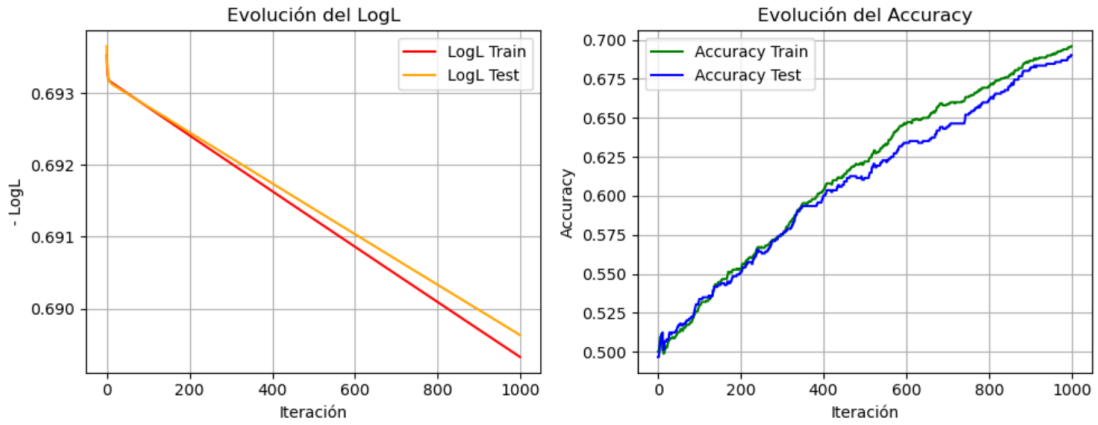

Figure 14: Evolución del -logL y accuracy utilizando a = 0.0001

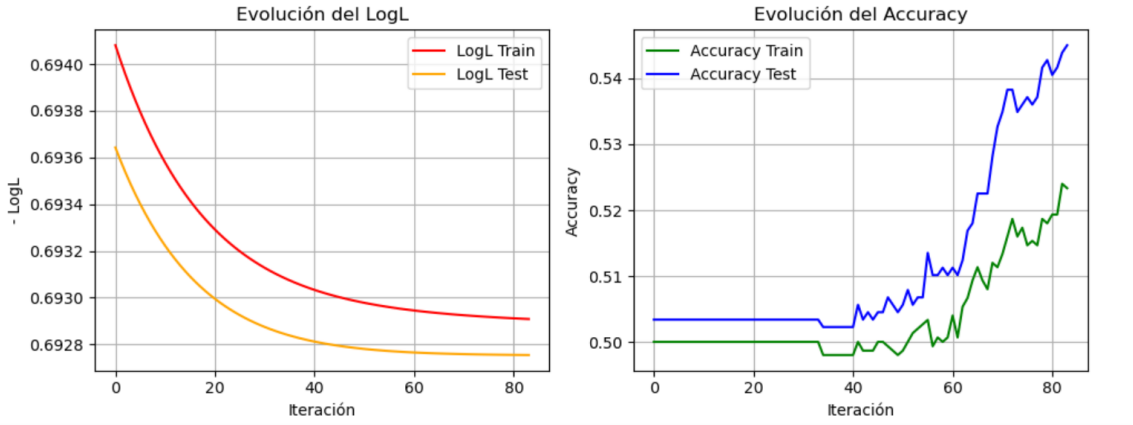

Figure 15: Evolución del -logL y accuracy utilizando a = 0.00001

Observando los gráficos, nuevamente 0.0001 es el valor que ofrece mejores resultados. La log-verosimilitud negativa disminuye de manera progresiva y sostenida durante todas las iteraciones, y la accuracy alcanza un valor cercano al 0.696 en entrenamiento y 0.690 en test. La evolución suave de la función objetivo y la mejora continua en el rendimiento sugieren que el modelo logra aprender

efectivamente sin sobreajustarse, lo cual lo convierte en el valor óptimo para α en este experimento.

Mientras tanto, para $\alpha = 0.00001$, como la tasa de aprendizaje es demasiado baja, el modelo requiere muchas iteraciones para realizar mejoras mínimas. Aunque se alcanza la convergencia alrededor de la iteración 80, tanto la log-verosimilitud como la accuracy se estancan, con un rendimiento final significativamente inferior al observado con $\alpha = 0.0001$. Esto evidencia que un valor de α tan pequeño limita la capacidad del modelo para ajustarse de manera eficiente.

5 Comparación entre métodos

La comparación entre ambos enfoques permite identificar diferencias sustanciales tanto en el rendimiento predictivo como en el comportamiento durante el proceso de optimización. Una de las principales distinciones radica en la formulación del objetivo. El descenso por gradiente busca minimizar el error cuadrático medio (ECM), mientras que el ascenso por gradiente está orientado a maximizar la log-verosimilitud, lo cual resulta más apropiado para tareas de clasificación binaria, ya que se alinea directamente con la interpretación probabilística de las salidas del modelo.

Comparando los resultados obtenidos, utilizando el mismo valor óptimo de $\alpha = 0.0001$, se observa que ambos modelos presentan una evolución favorable en su métrica objetivo y en la precisión, pero con diferencias significativas.

Para el método de Descenso por Gradiente, el error cuadrático medio disminuye de forma continua y estable, mientras que la accuracy alcanza un valor de aproximadamente 0.675 en la iteración 1000. Aunque no se observa convergencia estricta, el comportamiento es predecible y sin oscilaciones bruscas, lo cual indica una optimización relativamente estable.

Por otro lado, la log-verosimilitud decrece suavemente durante todas las iteraciones, lo cual sugiere un aprendizaje progresivo. La accuracy final supera los valores obtenidos en el otro método, alcanzando a 0.6903, lo cual marca una diferencia importante. Este resultado sugiere que el modelo de Ascenso por Gradiente no solo mejora en precisión, sino también en la capacidad de discriminar de forma más efectiva entre clases.

Comparando las matrices de confusión, en el conjunto de test, el primer modelo clasifica correctamente 371 casos sanos y 242 casos con Parkinson, mientras que con Ascenso por Gradiente estas cifras pasan a 378 y 210 respectivamente. Las diferencias son sutiles, aunque existe una disminución importante de verdaderos positivos (VP) en el segundo modelo, compensando lo mismo con un incremento notable en verdaderos negativos (VN).

Lo que sí resulta interesante observar, es la comparación entre ambos modelos para un valor de $\alpha = 0.00001$. En el primer caso, la accuracy presentaba ciertas fluctuaciones, mientras que en Ascesno por Gradiente, aunque también tiene sus altibajos, a simple vista no son tan graves como en el otro modelo. Esto puede demostrar una mejor estimación de valores en el segundo modelo.

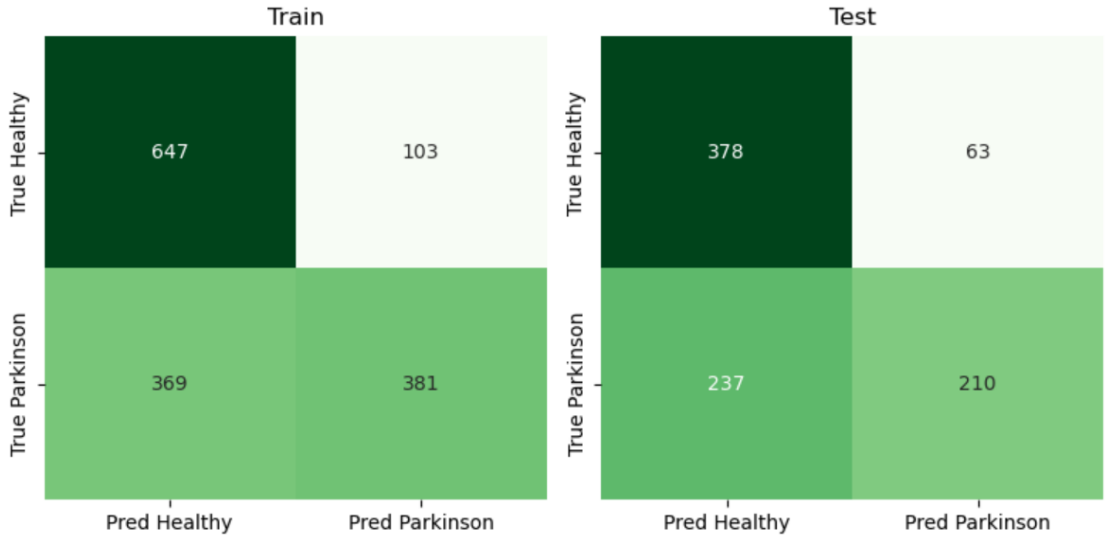

Figure 16: Matrices de confusión utilizando alpha = 0.0001 y size = 64x64

6 Conclusión

Resumiendo, a lo largo de este trabajo práctico se abordó el desarrollo de un clasificador binario capaz de distinguir entre personas sanas y pacientes con Parkinson, a partir del análisis de espirales dibujadas a mano. Para ello, se implementaron y compararon dos enfoques de aprendizaje supervisado: el Descenso por Gradiente, que optimiza el error cuadrático medio, y el Ascenso por Gradiente, que maximiza la log-verosimilitud de las predicciones.

En cada caso, evaluamos el entrenamiento y testing de los modelos mediante iteraciones controladas, con análisis de desempeño a través de métricas como el error, la accuracy y la matriz de confusión. Además, se evaluó el impacto de parámetros clave como la tasa de aprendizaje (α) y el tamaño de las imágenes (size), lo cual permitió explorar su influencia tanto en la precisión como en la eficiencia computacional.

Los resultados obtenidos indican que, si bien ambos enfoques pueden lograr convergencia y desempeños aceptables bajo determinadas configuraciones, el modelo basado en Ascenso por Gradiente presentó una capacidad de generalización superior, logrando mayor precisión en los datos de prueba. Esto se debe a que este método tiene en cuenta la interpretación probabilística del problema, lo cual lo vuelve más adecuado para tareas de clasificación binaria.

En definitiva, este trabajo permitió no solo aplicar técnicas vistas en clase, sino también reflexionar sobre los beneficios y limitaciones de distintos criterios de optimización en problemas reales, contribuyendo al entendimiento práctico de conceptos fundamentales del aprendizaje automático.