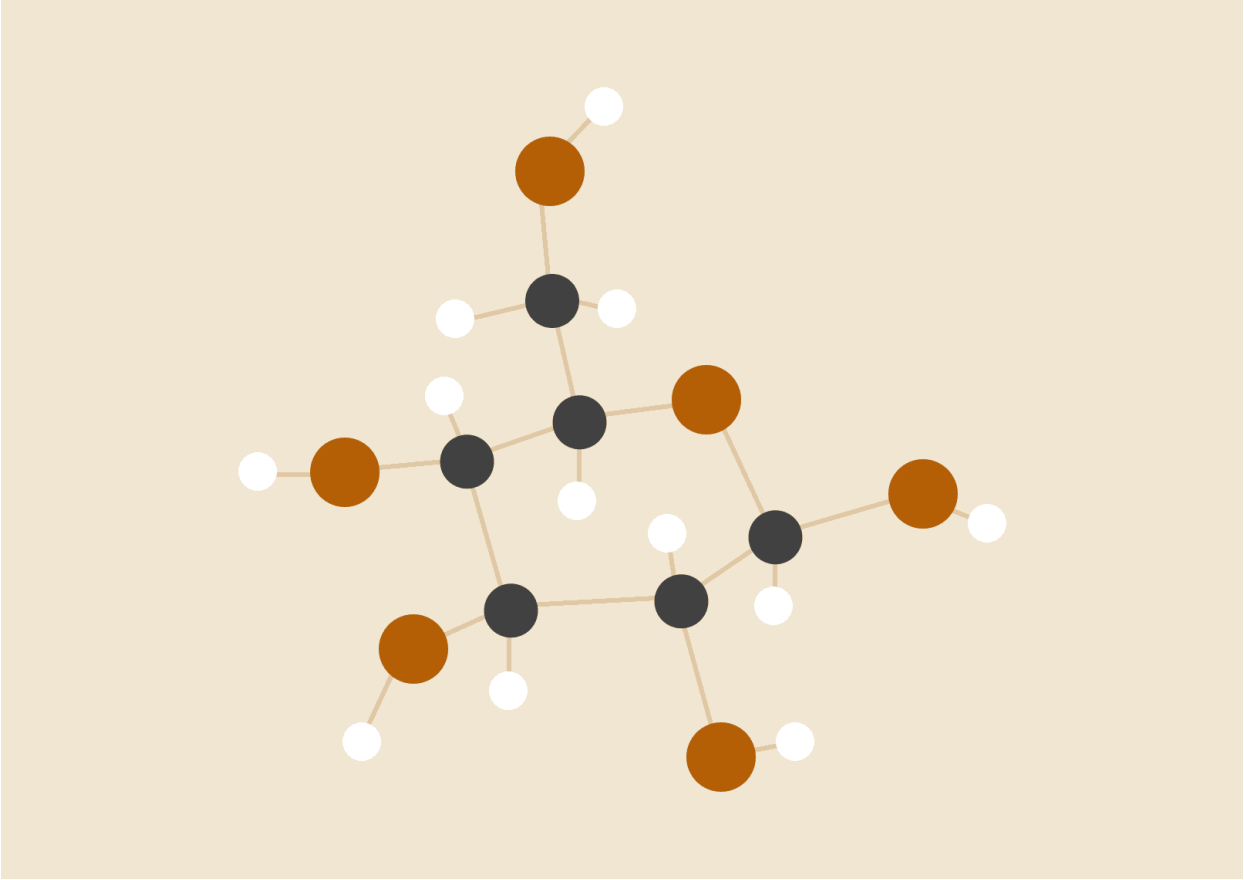
MANUAL TÉCNICO

Proyecto 2 Red Neuronal



Carlos Benjamin Pac Flores
201931012

28/04/2025
Inteligencia Artificial
Ingeniería en Ciencias y Sistemas CUNOC

ÍNDICE

INTRODUCCIÓN

El presente manual técnico tiene como objetivo documentar la estructura, el funcionamiento interno y las consideraciones de desarrollo de la aplicación para la clasificación de tumores mamarios mediante un perceptrón simple, implementada en el lenguaje de programación Python. Esta herramienta ha sido diseñada con fines educativos para ilustrar el proceso de entrenamiento, evaluación y uso de un modelo de red neuronal básica aplicado a un problema de clasificación binaria.

La aplicación utiliza el conjunto de datos Breast Cancer Wisconsin, el cual contiene características morfológicas de núcleos celulares obtenidas mediante imágenes de biopsias. Cada muestra en el dataset ha sido previamente clasificada como maligna (0) o benigna (1), lo que permite emplear técnicas de aprendizaje supervisado para el entrenamiento del modelo.

El sistema ha sido desarrollado bajo una arquitectura modular, con la finalidad de asegurar una correcta organización del código, facilitar su mantenimiento y permitir futuras extensiones o mejoras. Entre los principales componentes de la aplicación se encuentran:

- La implementación del perceptrón simple, encargado del proceso de aprendizaje y predicción.
- El módulo de entrenamiento, que gestiona la preparación de los datos, la configuración de los parámetros y la ejecución del aprendizaje.
- La interfaz gráfica de usuario (GUI), desarrollada con la librería PyQt5, que permite la interacción con el sistema de forma intuitiva.
- Los mecanismos de visualización de resultados, incluyen la representación gráfica de la frontera de decisión y la evolución del error durante las épocas.

Este manual describe en detalle la estructura del proyecto, las funciones de cada uno de sus módulos, el proceso de entrenamiento, las consideraciones de diseño adoptadas, así como las instrucciones necesarias para la instalación, configuración y ejecución del sistema.

TECNOLOGÍAS UTILIZADAS

Python 3.11

Lenguaje de programación base del sistema, elegido por su versatilidad, claridad sintáctica y amplia disponibilidad de bibliotecas para procesamiento de datos, visualización y desarrollo de interfaces gráficas.

BIBLIOTECAS Y HERRAMIENTAS

- ### PyQt5

  Biblioteca utilizada para la construcción de la interfaz gráfica de usuario (GUI). Facilita la interacción del usuario con el sistema, permitiendo configurar los parámetros de entrenamiento, seleccionar las características a utilizar, visualizar las gráficas generadas y realizar pruebas con datos introducidos manualmente.

- ### pandas

  Biblioteca esencial para la manipulación y el análisis de datos tabulares. Se utiliza para la carga, lectura y procesamiento del conjunto de datos Breast Cancer Wisconsin, almacenado en formato CSV.

- ### numpy

  Herramienta fundamental para el manejo eficiente de arreglos y operaciones matriciales. Es empleada tanto en la implementación del perceptrón simple como en el procesamiento de los datos de entrada, facilitando las operaciones matemáticas requeridas para el entrenamiento y la predicción.

- ### scikit-learn (sklearn)

  Biblioteca especializada en aprendizaje automático y minería de datos. Se utiliza específicamente para la obtención del conjunto de datos Breast Cancer Wisconsin, así como para la normalización de las características mediante el uso de MinMaxScaler y la división del conjunto de datos en subconjuntos de entrenamiento y prueba.

- ### matplotlib

  Biblioteca de visualización gráfica utilizada para generar las representaciones visuales del proceso de entrenamiento. Permite graficar la evolución del error a lo largo de las épocas y la frontera de decisión que delimita las clases predichas por el modelo, facilitando la interpretación de los resultados obtenidos.

PERCEPTRÓN SIMPLE

El perceptrón simple es uno de los modelos más básicos de redes neuronales artificiales, diseñado para resolver problemas de **clasificación binaria**. Su funcionamiento se basa en la combinación lineal de las características de entrada y la aplicación de una función de activación para determinar a qué clase pertenece una muestra.

Este modelo recibe como entradas las características seleccionadas de cada muestra y calcula una suma ponderada de estas mediante pesos y un sesgo (bias). Posteriormente, aplica una función de activación (función escalón o signo) que decide si la muestra pertenece a una clase u otra, basándose en si el resultado de la suma ponderada supera un umbral.

El entrenamiento del perceptrón consiste en ajustar los pesos y el sesgo mediante un proceso iterativo, conocido como **regla de aprendizaje del perceptrón**. Durante este proceso, el modelo compara las predicciones con las etiquetas reales del conjunto de datos, calcula el error y actualiza los pesos para minimizar dicho error. Este procedimiento se repite a lo largo de un número determinado de épocas.

El perceptrón simple es ideal para problemas donde las clases pueden ser separadas mediante una **línea recta (frontera de decisión lineal)**. Aunque no es capaz de resolver problemas no linealmente separables, su simplicidad lo convierte en una herramienta útil para comprender los conceptos fundamentales del aprendizaje supervisado y el funcionamiento de las redes neuronales.

En el contexto de este proyecto, el perceptrón simple ha sido implementado desde cero utilizando únicamente operaciones matriciales, sin el uso de librerías especializadas en redes neuronales. Su aplicación permite clasificar los tumores como **malignos (0)** o **benignos (1)**, empleando dos características seleccionadas por el usuario a partir del conjunto de datos **Breast Cancer Wisconsin**.

APLICACION PYTHON

El presente proyecto aborda el problema de la **clasificación de tumores mamarios** a partir del análisis de características morfológicas de núcleos celulares, mediante la implementación de un modelo de red neuronal básica, específicamente un **perceptrón simple**. El objetivo principal es determinar, a partir de dos características seleccionadas, si una muestra corresponde a un tumor **maligno (0)** o **benigno (1)**.

El modelo recibe como entrada las características elegidas por el usuario, realiza el proceso de entrenamiento ajustando los pesos y el sesgo mediante la regla de aprendizaje del perceptrón, y finalmente permite evaluar el rendimiento del modelo tanto a través de pruebas automáticas como mediante la introducción manual de nuevos datos.

El desarrollo del sistema se realizó utilizando el lenguaje de programación **Python**, seleccionado por sus capacidades orientadas a objetos, su sintaxis clara y la amplia disponibilidad de bibliotecas que facilitan la manipulación de datos, la visualización gráfica y la creación de interfaces gráficas de usuario.

La aplicación está estructurada en distintos módulos que separan las responsabilidades de cada componente:

- **Carga y preparación de datos:** mediante el procesamiento del conjunto de datos **Breast Cancer Wisconsin**, disponible en scikit-learn, facilitando su normalización y división entre entrenamiento y prueba.
- **Entrenamiento del modelo:** a través de la implementación de un perceptrón simple desde cero, sin el uso de frameworks externos para redes neuronales.
- **Interfaz gráfica de usuario (GUI):** desarrollada con **PyQt5**, permite la configuración de los parámetros del modelo, la selección de las características, el inicio del proceso de entrenamiento, la visualización de resultados y la prueba con datos manuales.
- **Visualización de resultados:** mediante gráficos generados con **matplotlib**, que representan tanto la evolución del error durante las épocas de entrenamiento como la frontera de decisión aprendida por el modelo.

MÉTODOS DE ENTRENAMIENTO IMPLEMENTADOS

Para lograr una clasificación efectiva de las muestras del conjunto de datos Breast Cancer Wisconsin, se ha implementado un modelo de red neuronal básica basado en el perceptrón simple. Este modelo realiza la clasificación binaria de las muestras como malignas (0) o benignas (1), ajustando sus parámetros internos (pesos y sesgo) a través de un proceso iterativo de aprendizaje supervisado.

Inicialización de Pesos y Sesgo

El proceso de entrenamiento inicia con la asignación aleatoria de los pesos y el sesgo del perceptrón. Los pesos son valores numéricos asociados a cada una de las dos características seleccionadas por el usuario, mientras que el sesgo es un valor adicional que permite desplazar la frontera de decisión. Estos parámetros son generados de forma aleatoria dentro de un rango controlado (valores entre -1 y 1), garantizando diversidad inicial en las configuraciones posibles del modelo.

Esta inicialización aleatoria permite que el modelo comience desde un estado no sesgado y mejore su rendimiento progresivamente a medida que avanza el proceso de entrenamiento.

Función de Activación

Para la clasificación, el perceptrón utiliza una función escalón o función de activación binaria, definida de la siguiente manera:

$$f(z) = \begin{cases} 1, & \text{si } (w_1 x_1 + w_2 x_2 + b) > 0 \\ 0, & \text{en otro caso} \end{cases}$$

Donde:

- *w1, w2 son los pesos,*
- *x1, x2 son las entradas (características seleccionadas),*
- *b es el sesgo.*

Esta función permite al modelo decidir si una muestra pertenece a la clase maligna (0) o benigna (1) según la posición de los datos respecto a la frontera de decisión.

Proceso de Entrenamiento (Regla de Aprendizaje del Perceptrón)

El modelo se entrena mediante la conocida regla de aprendizaje del perceptrón, la cual ajusta los pesos y el sesgo en cada iteración en función del error cometido en la predicción de cada muestra.

El proceso sigue los siguientes pasos:

1. Para cada muestra del conjunto de entrenamiento:
    - Se calcula la salida del modelo mediante la función de activación.
    - Se compara la salida con la etiqueta real.
    - Si la predicción es incorrecta, se calcula el error y se actualizan los pesos y el sesgo de acuerdo con la fórmula:

    $$w_i := w_i + \eta \cdot (y - \hat{y}) \cdot x_i$$

    $$b := b + \eta \cdot (y - \hat{y})$$

    Donde:

    - η es la tasa de aprendizaje definida por el usuario.
    - y es la etiqueta real.
    - $\hat{y}$ es la predicción realizada por el modelo.
2. Este proceso se repite durante el número de épocas (iteraciones) especificado por el usuario, permitiendo que el modelo ajuste progresivamente sus parámetros hasta minimizar los errores.

Evaluación del Modelo y Cálculo del Error

Durante el proceso de entrenamiento, se calcula el error total por época, definido como la suma de los errores cuadrados de las predicciones incorrectas. Este valor permite monitorear la convergencia del modelo a lo largo del tiempo y determinar si el aprendizaje es efectivo.

Adicionalmente, se calcula la exactitud (accuracy) del modelo sobre los datos de prueba, entendida como el porcentaje de muestras correctamente clasificadas:

$$\text{Accuracy} = \frac{\text{Numero de predicciones correctas}}{\text{Total de muestras}} \times 100$$

Esta métrica es mostrada al finalizar el proceso de entrenamiento como un indicador del rendimiento del modelo.

Visualización de Resultados

El sistema permite visualizar dos gráficos principales que facilitan el análisis del proceso de aprendizaje:

- Error vs Épocas: Muestra la evolución del error a lo largo del tiempo, permitiendo observar la estabilidad o la mejora del modelo durante el entrenamiento.
- Frontera de Decisión: Representa gráficamente la línea de separación aprendida por el perceptrón para distinguir las clases malignas y benignas en el espacio bidimensional de las características seleccionadas.

Consideraciones y Limitaciones

Debido a que el perceptrón simple es un modelo lineal, su capacidad de clasificación se limita a problemas que son linealmente separables. Esto implica que, si las muestras de las dos clases no pueden ser separadas por una línea recta en el espacio de las características seleccionadas, el modelo podría no alcanzar una precisión adecuada.

Sin embargo, para fines educativos y de comprensión del funcionamiento de las redes neuronales básicas, el perceptrón simple representa una herramienta eficaz y adecuada para ilustrar los conceptos fundamentales del aprendizaje supervisado.

ARQUITECTURA DEL SISTEMA

Interfaz Gráfica de Usuario (GUI)

La aplicación presenta una interfaz gráfica desarrollada utilizando **PyQt5**, una de las bibliotecas más robustas para la creación de interfaces gráficas en Python. Esta interfaz permite al usuario:

- Cargar el conjunto de datos **Breast Cancer Wisconsin** de manera sencilla.
- Seleccionar dos características (atributos) del dataset para utilizarlas como entradas del modelo.
- Configurar los parámetros del entrenamiento, incluyendo la **tasa de aprendizaje**, el **número de épocas** y el **porcentaje de datos destinados al entrenamiento**.
- Ejecutar el proceso de entrenamiento con un solo clic.
- Visualizar los resultados mediante dos tipos de gráficas:
    - **Error vs Épocas** (evolución del error durante el aprendizaje).
    - **Frontera de Decisión** (representación gráfica de cómo el modelo separa las dos clases).
- Probar el modelo con datos introducidos manualmente por el usuario.
- Consultar los últimos gráficos generados sin necesidad de repetir el entrenamiento.

La capa gráfica está completamente desacoplada de la lógica de negocio y del modelo, limitándose a la interacción con el usuario y la presentación de los resultados.

Módulo de Datos

La carga y preparación de los datos se realiza a través del módulo carga.py, que utiliza la biblioteca **pandas** para almacenar el conjunto de datos en formato CSV. Los datos corresponden al conjunto **Breast Cancer Wisconsin**, que contiene diversas características morfológicas de los núcleos celulares, así como la clasificación de cada muestra como **maligna (0)** o **benigna (1)**.

Este módulo garantiza que el dataset se encuentre disponible y correctamente estructurado para el posterior proceso de entrenamiento y evaluación.

Además, en el módulo entreno.py, se implementan procesos de:

- **Normalización de características** utilizando MinMaxScaler de **scikit-learn**, asegurando que los valores de entrada estén dentro de un rango uniforme para mejorar la estabilidad del entrenamiento.
- **División del dataset** en conjuntos de entrenamiento y prueba mediante train_test_split.

Núcleo del Modelo (Perceptrón Simple)

El corazón del sistema es la clase PerceptronSimple, ubicada en el archivo perceptron.py. Esta clase representa la implementación desde cero de un **perceptrón simple**, el cual realiza las siguientes funciones:

- Inicialización de los **pesos y sesgo** con valores aleatorios.
- Ejecución del proceso de **entrenamiento** mediante la regla clásica del perceptrón, ajustando los pesos y el sesgo en función del error cometido en cada predicción.
- Cálculo y registro del **error total por época**, permitiendo monitorear la convergencia del modelo.
- Realización de **predicciones** sobre nuevos datos, tanto en el conjunto de prueba como en datos introducidos manualmente por el usuario.

Este módulo es completamente independiente de la capa gráfica, lo que facilita su reutilización, modificación o prueba por separado.

Visualización de Resultados

La visualización de los resultados se realiza a través de la librería **matplotlib**, que permite generar las siguientes gráficas:

- **Error vs Épocas:** Representa cómo varía el error total durante el entrenamiento, facilitando la evaluación del proceso de aprendizaje.
- **Frontera de Decisión:** Muestra gráficamente la separación entre las dos clases (maligno y benigno) en el plano definido por las características seleccionadas. Los

puntos son coloreados según la clase real, y el fondo del gráfico muestra las regiones donde el modelo calificaría nuevas muestras.

Estas gráficas se integran en ventanas independientes dentro de la interfaz gráfica, ofreciendo una retroalimentación inmediata y clara al usuario.

NÚCLEO DE FUNCIONAMIENTO

El núcleo de procesamiento del sistema está conformado por la implementación del modelo de perceptrón simple, el cual es responsable del proceso completo de entrenamiento, ajuste de pesos y predicción. Este comportamiento se encuentra encapsulado en la clase **PerceptronSimple**, ubicada en el archivo **perceptron.py,** así como en las funciones de preparación de datos y entrenamiento definidas en **entreno.py**.

Este núcleo se encarga de recibir los parámetros configurados por el usuario a través de la interfaz gráfica, preparar los datos de entrada, ejecutar el proceso de aprendizaje supervisado y generar las predicciones y métricas necesarias para la evaluación del rendimiento del modelo.

Clase PerceptronSimple (Modelo del Perceptrón)

La clase PerceptronSimple implementa desde cero la lógica del perceptrón simple, el cual realiza la clasificación binaria mediante una frontera de decisión lineal.

Principales Atributos:

- pesos: Arreglo que almacena los pesos asignados a cada característica de entrada.
- sesgo: Valor escalar que permite ajustar la posición de la frontera de decisión.
- lr: Tasa de aprendizaje (\eta), que controla la magnitud de las actualizaciones de los pesos.
- epochs: Número de épocas o iteraciones sobre el conjunto de entrenamiento.

Principales Métodos:

- **activacion(entradas)**:
  Implementa la función escalón que define la salida del perceptrón (0 o 1) según el

valor de la suma ponderada de las entradas y el sesgo.

- **entrenar(X, y)**:
  Ejecuta el proceso de ajuste de los pesos y el sesgo a lo largo de las épocas especificadas. Calcula el error en cada predicción y actualiza los parámetros según la regla de aprendizaje del perceptrón.

- **predecir(X)**:
  Realiza la clasificación de las muestras ingresadas, aplicando la función de activación sobre la combinación lineal de las características y los pesos aprendidos.

Módulo entreno.py (Proceso de Entrenamiento)

El módulo entreno.py contiene la función **entrenar_red()**, la cual representa el flujo principal de entrenamiento del sistema. Esta función se encarga de:

- **Preprocesar los datos:**
  Utiliza MinMaxScaler de scikit-learn para normalizar las características seleccionadas, asegurando la estabilidad del aprendizaje.

- **Dividir el conjunto de datos:**
  Separa las muestras en subconjuntos de entrenamiento y prueba mediante train_test_split, respetando el porcentaje configurado por el usuario.

- **Inicializar y entrenar el modelo:**
  Crea una instancia de la clase PerceptronSimple y ejecuta el método entrenar() para ajustar los pesos y el sesgo del modelo.

- **Evaluar el rendimiento:**
  Calcula la **exactitud (accuracy)** del modelo sobre el conjunto de prueba y registra el historial del error por época.

- **Devolver resultados:**
  Retorna la lista de errores por época, la precisión alcanzada, el objeto scaler utilizado y el perceptrón entrenado, permitiendo su uso posterior para predicción y visualización.

Interfaz con el Núcleo desde la GUI (gui.py)

La interfaz gráfica, implementada en el archivo gui.py, se comunica directamente con el módulo de entrenamiento mediante el llamado a **entrenar_red()**. A través de esta interacción, la GUI permite:

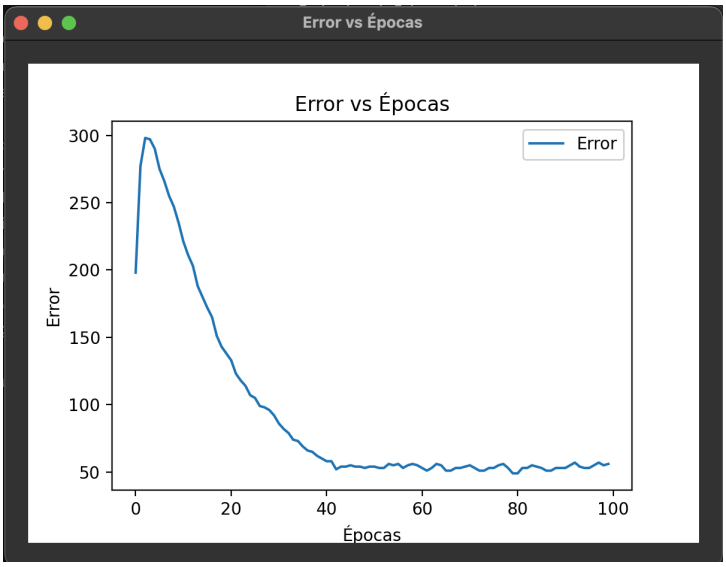
- Configurar los parámetros del modelo.
- Iniciar el proceso de entrenamiento y recibir los resultados.
- Visualizar el error por época y la frontera de decisión generada.
- Realizar pruebas manuales de predicción con datos nuevos introducidos por el usuario.

El modelo, los datos preprocesados y los resultados del entrenamiento se mantienen disponibles para su reutilización y análisis posterior dentro de la misma sesión de la aplicación.

ANÁLISIS DEL COMPORTAMIENTO DEL ENTRENAMIENTO

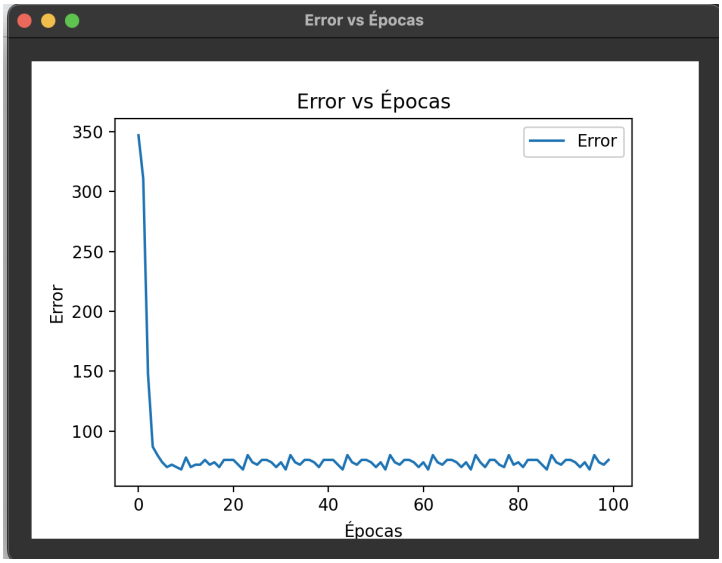
Durante la experimentación, se evaluó el desempeño del modelo de perceptrón simple utilizando distintos valores de la **tasa de aprendizaje η**. Este parámetro controla el ritmo con el que el modelo ajusta sus pesos en cada iteración. Los valores evaluados fueron: **0.001**, **0.01**, **0.1**, **0.5** y **1.0**. A continuación se presenta el análisis correspondiente:

η = 0.001 (muy bajo)



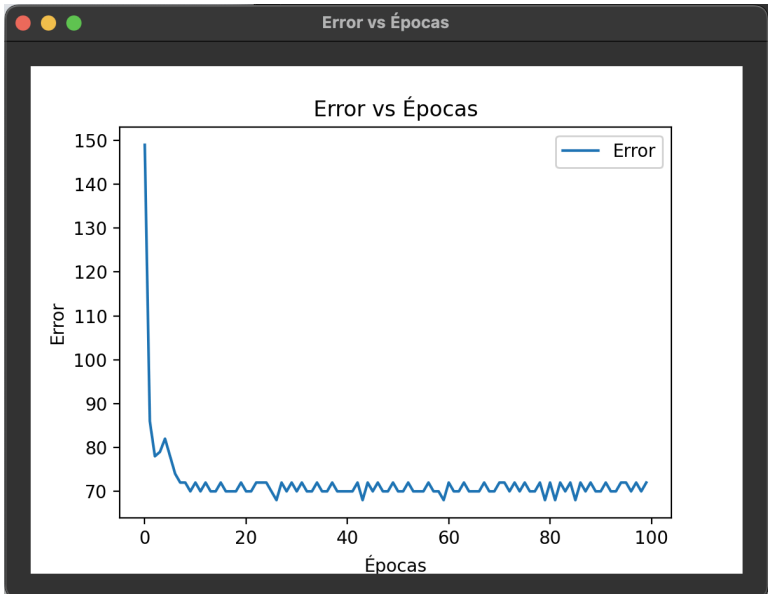
El modelo mostró un descenso gradual y estable del error, logrando una convergencia suave. Sin embargo, el número de épocas necesarias para alcanzar un error mínimo fue elevado, lo cual evidencia un proceso de aprendizaje **lento pero estable**. Esta configuración puede ser adecuada en casos donde se requiere evitar oscilaciones, aunque a costa de una mayor duración del entrenamiento.

η = 0.01 (bajo)



Este valor presentó un **balance adecuado** entre velocidad de aprendizaje y estabilidad. El error disminuyó rápidamente durante las primeras épocas, estabilizándose de forma consistente a partir de la época 10. Se considera una tasa de aprendizaje recomendable para el perceptrón en este tipo de problemas.

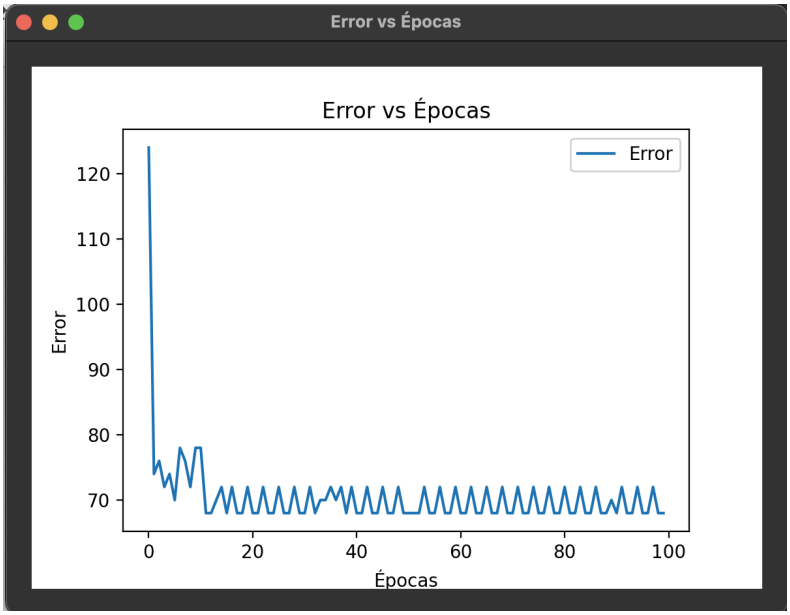
η = 0.1 (moderado)



El error descendió de forma rápida en las primeras iteraciones, pero se observó una

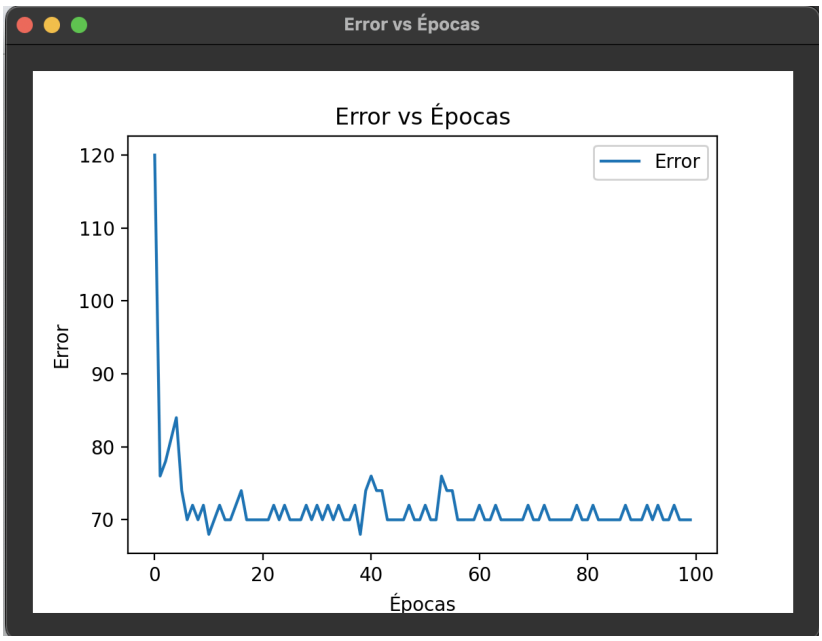
ligera oscilación en torno a un valor promedio después de la convergencia inicial. Aunque sigue siendo aceptable, puede presentar pequeñas variaciones en la precisión si se utiliza con conjuntos de datos más ruidosos o no linealmente separables.

η = 0.5 (alto)



El error mostró una caída abrupta y luego oscilaciones notorias durante el resto de las épocas. Esta configuración indica una **inestabilidad creciente**, provocando un comportamiento de "rebotar" alrededor de la solución en lugar de estabilizarse. Aunque el modelo aún logra aprender, la precisión puede verse afectada por la falta de convergencia fina.

η = 1.0 (muy alto)



El comportamiento del error es inestable. Se observan múltiples **picos y fluctuaciones irregulares**, lo que sugiere que el modelo está sobreajustando constantemente los pesos y **no logra estabilizar una solución óptima**. Este valor se considera inapropiado para el problema abordado, ya que compromete tanto la estabilidad como la precisión del modelo.

Conclusión

La tasa de aprendizaje **influye de forma directa en la velocidad, estabilidad y precisión del entrenamiento**. Valores muy bajos hacen que el modelo aprenda lentamente, mientras que valores excesivamente altos impiden la convergencia adecuada. En este caso, $\eta = 0.01$ se identifica como la mejor opción, al ofrecer una convergencia rápida y estable sin producir oscilaciones importantes en el error.

Estos resultados recalcan la importancia de seleccionar adecuadamente el valor de η, según la naturaleza del conjunto de datos y los requisitos de estabilidad del modelo.

REQUISITOS TÉCNICOS PARA EJECUTAR EL SISTEMA

Para garantizar una correcta instalación y ejecución del sistema de generación de horarios mediante algoritmos genéticos, es necesario cumplir con los siguientes requerimientos técnicos:

Versión mínima de Python

- **Python 3.10 o superior**
  Se recomienda utilizar versiones recientes de Python para asegurar la compatibilidad con bibliotecas modernas y aprovechar mejoras de rendimiento.

Dependencias del sistema

El sistema requiere la instalación de las siguientes bibliotecas de Python, especificadas en el archivo requirements.txt.

```
pip install -r requirements.txt
```

> **Nota:** Es necesario contar con wkhtmltopdf instalado en el sistema operativo para que pdfkit pueda generar correctamente los archivos PDF.

Comando para iniciar la aplicación

Una vez instaladas las dependencias, el sistema puede ejecutarse desde la terminal con:

```
python main.py
```

Estructura del Proyecto

La organización del proyecto sigue una estructura modular y clara:

```
Proyecto2IA/
│
├── data/             # Almacena el conjunto de datos CSV
│   └── breast_cancer_data.csv
│
├── src/                  # Código fuente del proyecto
│   ├── carga.py          # Lógica para cargar y guardar el dataset
│   ├── entreno.py        # Función de entrenamiento del perceptrón simple
│   ├── gui.py            # Implementación de la interfaz gráfica (PyQt5)
│   ├── main.py           # Punto de entrada de la aplicación
│   └── perceptron.py     # Implementación del perceptrón simple (modelo)
│
├── requirements.txt    # Lista de dependencias del proyecto
├── README.md           # Guía rápida y descripción general del proyecto
├── .gitignore          # Archivos y carpetas ignorados por Git
└── venv/               # Entorno virtual (no incluido en el repositorio)
```