Algoritmos genéticos para problemas de regresión no lineal

Javier Ruíz Garrido
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
2210jrg@gmail.com — javruigar2@alum.us.es

Javier Santos Martín
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
javier.jsm21@gmail.com — javsanmar5@alum.us.es

***Resumen*—El objetivo principal de este trabajo fue desarrollar y evaluar un modelo de regresión no lineal utilizando algoritmos genéticos. Estos algoritmos, inspirados en los procesos de selección natural y evolución biológica, fueron empleados para optimizar los parámetros del modelo, representados en forma de cromosomas que contienen coeficientes y exponentes. Se realizaron experimentos con tres conjuntos de datos distintos: `toy1`, `housing` y `synt1`, evaluando el desempeño del modelo mediante el cálculo del error cuadrático medio (RMSE) y el coeficiente de determinación (R2).**

El algoritmo creado cuenta con distintas ideas clave que fueron dando mejoras de rendimiento y precisión. Además, el control de la semilla de aleatoriedad y un sistema de registro de récords han sido muy importantes para garantizar los mejores resultados posibles del algoritmo. Entre las mejoras futuras, se sugiere paralelizar partes del código y aplicar este algoritmo en conjuntos de datos más grandes y complejos para validar su robustez y generalizabilidad.

***Palabras clave*—Algoritmo genético, regresión no lineal, cromosoma, mutación, cruce, elitismo.**

I. Introducción

En el campo de la inteligencia artificial y el aprendizaje automático, los algoritmos genéticos se han destacado como una poderosa herramienta para resolver problemas de optimización y búsqueda en espacios complejos. Estos algoritmos, inspirados en los procesos de selección natural y evolución biológica, se aplican a una variedad de dominios, desde la ingeniería hasta la economía, proporcionando soluciones efectivas donde otros métodos tradicionales pueden fallar. Los problemas de regresión no lineal, en particular, presentan desafíos significativos debido a la naturaleza intrínseca de sus relaciones no lineales entre variables. La búsqueda de modelos precisos y eficientes para estas relaciones es crucial en muchas aplicaciones prácticas, como la predicción financiera, la modelización de sistemas físicos y la medicina.

La problemática central de este proyecto radica en la precisión insuficiente de los métodos tradicionales de regresión lineal para modelar relaciones no lineales complejas en conjuntos de datos. El objetivo es desarrollar un modelo de regresión no lineal más preciso mediante el uso de algoritmos genéticos. La solución implementada se basa en la creación de cromosomas que representan coeficientes y exponentes para cada característica del conjunto de datos. Estos cromosomas son evaluados utilizando el error cuadrático medio (RMSE) como métrica de aptitud. El algoritmo genético itera sobre una población de cromosomas, aplicando operadores de selección por torneo, cruce y mutación, para optimizar los parámetros del modelo. El enfoque permite explorar de manera eficiente el espacio de búsqueda y ajustar los coeficientes y exponentes para minimizar el RMSE, mejorando así la capacidad del modelo de predecir futuros ejemplos.

El documento está estructurado de la siguiente manera: Tras la introducción, en la Sección II se describen los fundamentos teóricos con los que se ha trabajado el algoritmo genético y la regresión no lineal. La Sección III detalla la metodología propuesta, incluyendo la representación de los cromosomas, los operadores genéticos utilizados y el proceso de evaluación de los modelos. En la Sección IV se presentan los experimentos realizados y los resultados obtenidos, destacando las ventajas y limitaciones del enfoque propuesto. Finalmente, en la Sección V se presentan las conclusiones y se sugieren posibles líneas de investigación futura.

II. Preliminares

En este trabajo, se busca aplicar la idea de los algoritmos genéticos para resolver problemas de regresión. Para ello, los cromosomas modelarán los parámetros de la función que intentamos predecir, y su valoración se realizará utilizando datos de prueba. Esto se llevará a cabo prediciendo los datos y evaluando la precisión de las predicciones comparándolas con los resultados esperados.

Todo esto se desarrollará dentro de un algoritmo genético estándar, que incluirá componentes como el elitismo, el cruce entre cromosomas y la mutación aleatoria. Finalmente, se explicarán algunas medidas para mejorar la eficiencia en este contexto específico.

A. Cromosoma

En el contexto de nuestro trabajo, el cromosoma se ha conceptualizado como una clase de Python. Para ello, se ha diseñado una clase abstracta denominada *AbstractChromosome*, la cual establece los métodos fundamentales que debe poseer cualquier cromosoma. A partir de esta clase abstracta, se permite definir nuevas clases de cromosomas que extienden sus funcionalidades. En nuestro caso, hemos implementado la

clase *Chromosome(AbstractChromosome)*, la cual se ajusta a nuestros requerimientos, dado que podemos enmarcar todos los problemas de regresión no lineal en ella, sin tener que crear una clase distinta para cada fichero o problema a resolver.

El modelado del cromosoma en nuestro caso incluye los siguientes aspectos:

1) Propiedades:

- **Lista de coeficientes:** Esta lista contiene $n+1$ elementos de tipo `float`, donde n es el número de variables independientes. El elemento j de la lista representa el coeficiente de la variable x_j, mientras que el último elemento representa el término independiente.
- **Lista de exponentes:** Esta lista contiene n elementos de tipo `float`, donde n es el número de variables independientes. El elemento j de la lista representa el exponente de la variable x_j.

Aunque, a priori, todos los elementos de ambas listas no tienen restricciones en cuanto a los valores que pueden adoptar, existen dos situaciones que podrían limitar estos valores:

1. **Variable x_i negativa:** En este caso, el exponente de la función no puede asumir valores fraccionarios cuyo denominador sea par, es decir, de la forma k/m donde m es par, en el ámbito de los números reales. A priori, este sería el único problema, pero obtenemos otro por la forma de trabajar de Python con las potencias fraccionarias: cualquier número negativo elevado a una potencia fraccionaria, ya sea con un denominador par o impar, resulta en un número complejo. Esto no es deseable, dado que estamos trabajando en el conjunto de los números reales. Podemos ver este fallo de Python ejemplificado a continuación, donde se eleva un número negativo -1 a un número fraccionario con denominador impar $0.2 = \frac{1}{5}$, lo que resulta en la raíz quinta de -1, que es igual a -1:

Esperado:

$$\sqrt[5]{-1} = -1$$

Obtenido:

```
>>> base, exponent = -1, 0.2
>>> base ** exponent
(0.8090169943749475+0.5877852522924731j)
```

Hemos planteado diversas opciones para la resolución de este problema, como intentar modificar la fracción para garantizar que el denominador sea impar (lo cual hemos observado que no es efectivo debido al error en Python) o alterar la base (representada por la variable x_j), lo cual consideramos completamente incorrecto. Modificar la base implicaría permitir exponentes que generan este problema y exigiría a quienes utilicen esta solución que ajusten sus variables para trabajar con números positivos. Además, sabemos que si alteramos la base, es decir, los datos de entrada, el valor esperado y_i ya no sería válido, dado que el dato de entrada no sería el original.

La alternativa seleccionada como solución al problema consiste en utilizar el módulo del resultado de la operación $x_j^{e_j}$, ya que el error en Python parece ser un problema de aproximación. A continuación, se muestra que el módulo del número complejo devuelto es el mismo que el módulo del valor esperado (1):

```
>>> base, exponent = -1, 0.2
>>> base ** exponent
(0.8090169943749475+0.5877852522924731j)
>>> res = base ** exponent
>>> res
(0.8090169943749475+0.5877852522924731j)
>>> abs(res)
1.0
```

Lo único que se requiere es multiplicar este valor por la función signo de la base (sign(x_j)). Esto se justifica de la siguiente manera: en el contexto de los números complejos, el número de soluciones a una raíz de índice i es igual a i, y todas estas soluciones comparten una característica común: el módulo de estas soluciones es idéntico, y todas están espaciadas por el mismo ángulo entre ellas: $\frac{360}{i}$. Véase la imagen Fig. 1.

Esta solución ha proporcionado, además, los mejores resultados en nuestras pruebas.

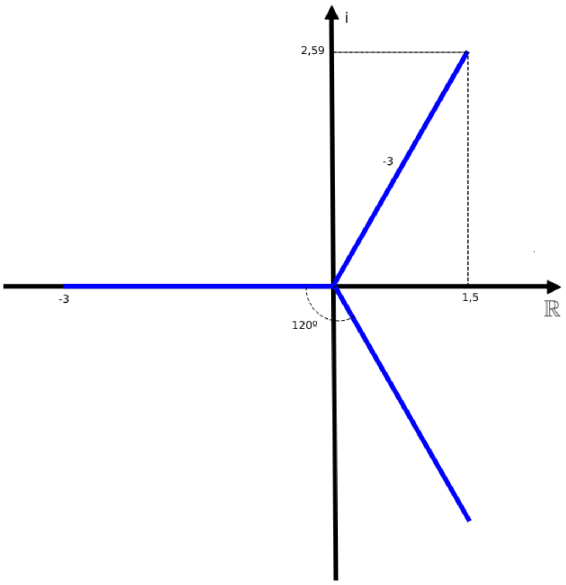

Fig. 1. Ejemplificación de las soluciones complejas a una raíz.

Para una mejor comprensión sobre el manejo de números complejos en Python, se sugiere consultar el estudio mencionado en [3].

2. **Variable x_i igual a cero:** Aquí, el problema surge con los exponentes negativos, pues esta operación está indeterminada. Por tanto, para resolver este problema, utilizamos el valor absoluto del exponente, ya que al saber que la función a interpolar está definida en $x_i = 0$, podemos afirmar que $e_j > 0$.

2) Fitness: En esta sección, se realizará la evaluación de los cromosomas. Una idea fundamental en los algoritmos genéticos es la capacidad de valorar con la mayor precisión

posible cada cromosoma, con el objetivo de determinar si un cromosoma es mejor o peor que otro. En el contexto de un problema de regresión, esta evaluación se basa en la comparación entre los valores predichos por el cromosoma y los valores reales.

Para evaluar la precisión de las predicciones, utilizamos el error cuadrático medio (RMSE, por sus siglas en inglés), una métrica ampliamente aceptada en problemas de regresión. El RMSE se calcula mediante la siguiente fórmula:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$$

Una vez calculado el RMSE para un cromosoma específico, existen dos enfoques posibles para su utilización en la función de fitness: devolver el inverso del RMSE para maximizar el valor de la función de fitness, o devolver el RMSE directamente y minimizar su valor. En nuestro caso, hemos optado por la segunda opción, es decir, minimizar el RMSE.

A continuación, describiremos el proceso de predicción utilizando los datos del cromosoma en cuestión, que es crucial para la evaluación de su fitness.

3) Predicción: En esta sección, explicaremos cómo obtener una predicción $\hat{y}_i$ para un cromosoma específico utilizando datos de prueba concretos. Este proceso se repetirá para cada uno de los datos de prueba que deseamos evaluar. Dado que estamos abordando un problema de regresión no lineal, debemos elevar cada una de las variables de los datos de prueba x_j a un exponente e_j y multiplicarlas por un coeficiente c_j, donde j representa los índices de todas las variables en la ecuación de regresión.

Posteriormente, sumamos estos términos junto con un último coeficiente, que actúa como término independiente, para obtener la predicción final. Utilizando los valores de un cromosoma determinado, la predicción $\hat{y}_i$ se calcula de la siguiente manera:

$$\hat{y}_i = \sum_{j=0}^{n} c_j x_j^{e_j} + c_{n+1}$$

4) Cruce de Cromosomas: El proceso de cruce de cromosomas tiene como objetivo combinar dos cromosomas parentales, eligiendo uno o varios puntos de cruce de manera aleatoria. Una vez determinado el o los puntos de cruce, se crean dos cromosomas hijos seleccionando distintas parte de cada cromosoma padre.

Consideramos la posibilidad de emplear uno o dos puntos de cruce en este proceso. Para tomar esta decisión, consultamos investigaciones previas sobre algoritmos genéticos (véase [1] y [2]). Sin embargo, finalmente llegamos a la conclusión de utilizar únicamente un punto de cruce. Esta elección se basa en nuestra consideración de que este enfoque se ajusta mejor a la documentación proporcionada y es una visión más sencilla y suficientemente eficaz para el problema a resolver.

Para decidir cuándo realizar el cruce, se utiliza una tasa de cruce, la cual es un valor entre 0 y 1. Se elige un número aleatorio, también entre 0 y 1, y la probabilidad de que este número sea menor que la tasa de cruce es precisamente la tasa de cruce. Por lo tanto, si el número aleatorio es mayor que la tasa de cruce, los cromosomas se devuelven sin cambios.

En caso de que se deba realizar el cruce, se genera aleatoriamente un punto de corte en los cromosomas parentales. En este punto, ambos cromosomas se dividen y se combinan emparejando una mitad de uno con la mitad complementaria del otro, y viceversa. Podemos ver este proceso ejemplificado en Fig. 2

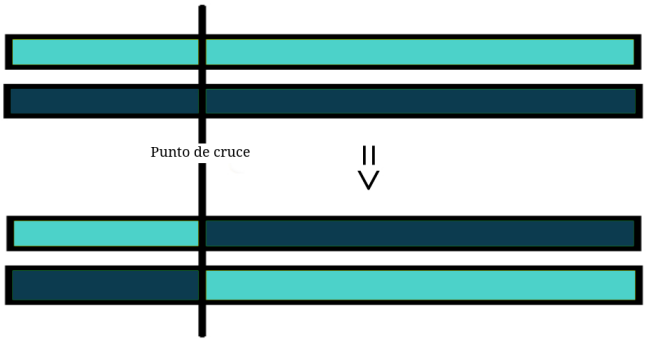

Fig. 2. Proceso de cruce ilustrado.

En nuestro caso, dado que manejamos dos listas, una de coeficientes y otra de exponentes, se generan dos puntos de corte y se realiza el cruce de la manera descrita anteriormente en ambas listas. Creando dos nuevos objetos Cromosoma, y devolviendo estos dos.

5) Mutación de Cromosomas: De primeras, contemplamos que la mutación de cromosomas se llevara a cabo de manera aleatoria, siguiendo un proceso similar al de cruce de cromosomas, pero, en este caso, generando un número aleatorio por cada parámetro del cromosoma a mutar.

Generando un número con el que determinar si un parámetro ha de ser mutado o no; cuando es seleccionado para mutación, se modifica sumándole o restando un valor aleatorio dentro de un rango predeterminado. Este rango incluye números negativos, lo que permite tanto el incremento como la disminución del parámetro con igual probabilidad. Y repitiendo este proceso para cada parámetro del cromosoma hasta completar el proceso de mutación.

Sin embargo, tras realizar diversas pruebas, observamos que se obtenían mejores resultados al generar un único número aleatorio y, en función de este, modificar de manera ligera todos los parámetros del cromosoma, en lugar de generar aleatoriedad para cada valor del cromosoma de forma individual. A pesar de que pueda parecer contraintuitivo, esta modificación no solo mejora el rendimiento, debido a la reducción en la generación de números y en las comprobaciones necesarias (lo cual es prácticamente insignificante), sino que también conduce a mejores resultados y a RMSE más reducidos.

B. Algoritmo genético

En esta sección, describimos el proceso completo del algoritmo genético, utilizando los componentes definidos anteriormente. El algoritmo comienza con la generación de una población inicial de un tamaño determinado. Cada cromosoma de la población es evaluado utilizando la función de fitness descrita previamente.

Una vez que todos los cromosomas han sido evaluados, se ordenan de menor a mayor según su valor de fitness, y se seleccionan los mejores para formar la población de la siguiente generación. Con los cromosomas restantes, se procede de la siguiente manera:

1) Selección mediante torneo: Para seleccionar los padres de la siguiente generación, se utiliza un método de selección mediante torneo. Este proceso consiste en seleccionar aleatoriamente k cromosomas y escoger los dos con mejor fitness, es decir, los dos con el menor RMSE respecto a los datos reales.

Una vez seleccionados los dos padres, se generan dos hijos para la siguiente generación utilizando el proceso de cruce explicado anteriormente. Posteriormente, estos hijos pasan por un proceso de mutación aleatoria.

Este ciclo se repite tantas veces como el número de iteraciones especificado. Al finalizar la última iteración, se devuelve el cromosoma con el menor RMSE como la solución final al problema.

C. Medidas para mejorar la eficiencia

Además de buscar el menor RMSE posible, también teníamos la intención de optimizar los tiempos de ejecución de nuestro algoritmo, de manera que pudiera llegar a soluciones esperadas en el menor tiempo posible. Para alcanzar este objetivo, hemos implementado varias medidas destinadas a mejorar la eficiencia del algoritmo. Estas medidas son:

1) Conjunto de datos para evaluar un cromosoma: En principio, cuanto mayor sea el número de datos en nuestro conjunto de prueba, mejor será la valoración del cromosoma, ya que una mayor variedad de datos generalmente conduce a una regresión más precisa. Sin embargo, aumentar el número de casos de prueba también incrementa el tiempo necesario para evaluar cada cromosoma.

Para aprovechar las ventajas de disponer de un gran número de datos de prueba y, al mismo tiempo, reducir el costo computacional, propusimos la siguiente solución: introducir un nuevo parámetro, con un valor entre 0 y 1, que indique el porcentaje de datos a utilizar en cada valoración. Estos datos se seleccionarán de forma aleatoria para cada evaluación de un cromosoma. Sin embargo, esta idea presentaba un problema: en conjuntos de datos con un gran volumen de información, esta solución era beneficiosa, pero en aquellos con menor cantidad de datos, se perdía demasiada diversidad. Por lo tanto, consideramos que la mejor solución, sin necesidad de ajustar este valor en función del tamaño del conjunto de prueba, fue establecer un límite máximo de datos de prueba para cada proceso de evaluación. En conjuntos de entrenamiento con un número de datos menor que el límite, se utilizarían todos; mientras que en ficheros extensos se seleccionaría un subconjunto del tamaño de este límite, variando en cada proceso de evaluación de un cromosoma. En nuestro caso, hemos determinado que este límite de datos sea de 500 muestras.

De esta manera, un cromosoma que progresa a través de distintas generaciones debido a su buen desempeño en los datos asignados, será evaluado con diferentes subconjuntos de datos en cada iteración. Esto no solo asegura una evaluación más robusta, sino que también reduce significativamente el costo computacional de valorar un cromosoma.

2) Almacenamiento de la valoración de un cromosoma: Durante la ejecución del algoritmo genético, debemos ordenar todos los cromosomas de la población actual en función de su valoración. Estas valoraciones también se utilizan posteriormente en la selección por torneo.

Inicialmente, la población se representaba como una lista de cromosomas, y el cálculo de la valoración se realizaba cada vez que era necesario obtener la valoración de un cromosoma. Este enfoque era extremadamente ineficiente.

Para mejorar la eficiencia, propusimos cambiar esta lista de cromosomas por una lista en la que se almacenen pares $[\text{cromosoma}, \text{cromosoma.fitness()}]$. Consideramos varias opciones para implementar esta estructura, y finalmente optamos por utilizar una lista de listas. Aunque esta elección podría ser ligeramente menos eficiente que usar un diccionario o una lista de tuplas, vimos que la manipulación de listas era lo más comodo para nosotros. Las listas son variables mutables, lo que nos permite crear la población de la siguiente generación y realizar operaciones de mutación y cruce con facilidad. Al comienzo de cada nueva generación, podemos establecer las valoraciones utilizando nuevos datos de prueba, tal como se explicó anteriormente.

III. Metodología

Esta sección presenta los pseudocódigos de cada una de las funciones que se usan en el diseño del algoritmo genético, los cuales han sido explicados en el apartado II.

fitness$(self, train\_data, data\_percentage)$
Entrada: El cromosoma actual, una lista de tuplas de floats que representan los datos de entrenamiento y un float que representa el porcentaje de datos con los que será entrenado el cromosoma
Salida: Un float que representa el error cuadrático medio (RMSE)

```
predicted ← lista vacía
expected ← lista vacía
subset ← seleccionar aleatoriamente
    (train_data * data_percentage) elementos de
        train_data
para cada datum en subset hacer
    agregar llamada a la funcion predict con datum
        como parametro a predicted
        agregar último elemento de datum a expected
    rmse ← root_mean_squared_error(expected, predicted)
    devolver rmse
```

Fig. 3. Función de evaluación del cromosoma (`fitness`) en pseudocódigo

predict(*self, datum*)
Entrada: El cromosoma actual, una tupla de floats que representa una línea de los datos de entrenamiento
Salida: Un float que representa el valor predicho

```
prediction ← 0
para cada j en rango len(datum) - 1 hacer
    si el dato recibido j es igual que 0 entonces
        el exponente j ← hacer el valor absoluto
    power_term ← sign de datum j * modulo de
        (datum j ** exponente j)
    prediction += el coeficiente j × power_term
prediction += término independiente
devolver prediction
```

Fig. 4. Función de predicción (`predict`) en pseudocódigo

crossover(*self, parent1, parent2*, *cross_rate*)
Entrada: Padre1, padre2 y la tasa de cruce
Salida: Una lista de los dos cromosomas resultantes

```
si el número random generado es mayor que cross_rate
    entonces
        devolver una lista con parent1 y parent2
            cromosoma con el que iba a ser cruzado
point ← número random entre el uno y la longitud de
    la lista de exponentes de parent1 menos uno
child1_coefficients ← los coeficientes
    de parent1 hasta "punto" + los coeficientes de
        parent2 desde "punto"
child1_exponents ← los exponentes de parent1
     hasta "punto" + los exponentes de
        parent2 desde "punto"
child2_coefficients ← los coeficientes de parent2
     hasta "punto" + los coeficientes de
        parent1 desde "punto"
child2_exponents ← los exponentes de parent2
     hasta "punto" + los exponentes de
        parent1 desde "punto"
devolver lista con el cromosoma child1_coefficients
    y child1_exponents, y el cromosoma
        child2_coefficients y child2_exponents
```

Fig. 5. Función de cruza (`crossover`) en pseudocódigo

mutate(*self*, *mutation_rate*, *mutation_range*)
Entrada: El cromosoma actual, tasa de mutación y rango de mutación
Salida: Ninguna

```
random_number ← Número aleatorio entre 0 y 1
si random_number es menor que mutation_rate entonces
    para cada i en rango longitud de la lista de exponentes
        del cromosoma actual hacer
            coeficiente i del cromosoma actual  += un valor
                aleatorio entre -mutation_range y
                    mutation_range
            exponente i del cromosoma actual  += un valor
                aleatorio entre -mutation_range y
                    mutation_range
    último coeficiente del cromosoma actual += un valor
        aleatorio entre -mutation_range y mutation_range
```

Fig. 6. Función de mutación (`mutate`) en pseudocódigo

tournament_selection(*self*, *k*)
Entrada: Instancia de la clase AG y número de cromosomas seleccionados para el torneo (*k*)
Salida: Cromosoma con la mejor aptitud

```
tournament ← k elementos aleatorios de población actual
devolver elemento con menor fitness
```

Fig. 7. Selección de torneo en pseudocódigo

test(*self*, *chromosome*)
Entrada: Instancia de la clase AG y cromosoma a probar
Salida: Lista de valores predichos
1 **devolver** *[predicción **de** dato **por** dato **en** conjunto de test]*

Fig. 8. Función de prueba (`test`) en pseudocódigo

run(*self*)
Entrada: Instancia de la clase AG
Salida: El cromosoma con la mejor aptitud encontrada y ese cromosoma sometido a la funcion test

```
1  elitism_count ← ratio de elitismo por el tamaño de
2      población
3  para generation en rango máximas iteraciones hacer
4      para pair en la población actual hacer
5          self.population ← lista de listas del primer
6              elemento de pair y de su función fitness
7      self.population ← se ordena por el segundo
8          elemento de cada tupla de la lista de
9              población
10     para pair en la población actual hasta el
11         elitism_count elemento hacer
12             next_generation ← añadir el primer
13                 elemento de pair
14     mientras longitud de next_generation menor que
15         tamaño de población hacer
16             parent1, parent2 ← cromosoma ganador
17                 del torneo cromosoma ganador del
18                     torneo
19             offspring ← crossover entre parent 1 y 2
20             para cada child en offspring hacer
21                 si longitud de next_generation menor
22                     que tamaño población
23                         child ← mutate cromosoma
24                         next_generation ← añadir
25                             child
26     best_fitness ← segundo elemento de la primera tupla
27         de la lista de población
28     self.population ← next_generation
29 winner_chromosome ← cromosoma con mejor fitness
30 devolver winner_chromosome, test de winner_chromosome
```

Fig. 9. Función de ejecución (`run`) en pseudocódigo

IV. Resultados

En esta sección se presentan los resultados del entrenamiento del modelo de regresión no lineal utilizando algoritmos genéticos. Para ello, se emplearon tres conjuntos de datos distintos: `toy1`, `housing` y `synt1`. El modelo fue entrenado y evaluado de manera independiente para cada conjunto de datos, utilizando un archivo de prueba específico para cada caso.

El tiempo de entrenamiento varía significativamente según el conjunto de datos aplicado. Los principales factores que influyen en el tiempo de entrenamiento son los siguientes:

- **Tamaño del conjunto de entrenamiento**: Este factor se refiere a la cantidad de datos disponibles para entrenar el modelo. En nuestro caso, su impacto es menor debido a la decisión de diseño de utilizar un conjunto de entrenamiento dinámico a lo largo de las diferentes generaciones del algoritmo, tal como se describe en la sección de mejoras de optimización.
- **Número de variables del conjunto de datos**: Este es un factor crucial, ya que un mayor número de variables ralentiza el proceso de predicción, el cual debe repetirse numerosas veces durante el entrenamiento. En particular, este factor es determinante para el tiempo de entrenamiento del conjunto de datos `synt1`.

Una vez entrenado el modelo, y antes de devolver el cromosoma óptimo, se evalúa su desempeño utilizando un conjunto de datos de prueba. Este conjunto permite determinar la eficacia del modelo. Tras esta evaluación, se selecciona el cromosoma ganador y se generan las predicciones $\hat{y}_i$ para todos los datos del conjunto de prueba. A continuación, se calcula el Error Cuadrático Medio (RMSE) entre los resultados esperados y los obtenidos, así como el coeficiente de determinación (R^2).

Durante todo este proceso, tanto en la fase de entrenamiento como en la de prueba, se inicia un temporizador para medir el tiempo total empleado en el modelado de la solución, siendo este uno de los datos clave reportados.

Es importante destacar que, aunque varios procesos dentro del algoritmo, como la mutación, el cruce y la asignación de datos a cada cromosoma, son inherentemente aleatorios, estos procesos se controlan mediante una semilla específica. Esta semilla, proporcionada por la librería de Python utilizada, asegura que el experimento es reproducible. Por lo tanto, si el programa se ejecuta múltiples veces sin cambiar ningún parámetro, los resultados serán consistentes. Para obtener diferentes resultados con los mismos parámetros, es necesario modificar la semilla.

Los parámetros que influyen en los posibles resultados del modelo son los siguientes: tamaño de la población de cromosomas, número de iteraciones (aunque otra forma de limitar el proceso podría ser alcanzar un RMSE determinado o un tiempo máximo especificado), tasa de mutación, rango de mutación (el intervalo en el que se encuentra el valor que se va a sumar o restar a las posiciones del cromosoma), rango de valores iniciales, tasa de cruce y tasa de elitismo. En nuestro estudio, los valores seleccionados para estos parámetros son:

TABLA I
Parámetros del modelo y sus valores.

Parámetro	**Valor**
Tamaño de la población	50
Número de iteraciones	300
Tasa de mutación	0.2
Rango de mutación	0.6
Rango de valores iniciales	1.0
Tasa de cruce	0.7
Tasa de elitismo	0.2

Es importante destacar que todos estos parámetros pueden afectar de manera tanto positiva como negativa a los resultados obtenidos. Lo que puede ser beneficioso para un conjunto de datos podría resultar perjudicial para otro. Por este motivo, hemos seleccionado valores que buscamos que sean adecuados para todos los conjuntos de datos empleados. No obstante, se recomienda ajustar estos parámetros según sea necesario para optimizar los resultados en un conjunto de datos específico. Estos parámetros pueden ser establecidos al crear la instancia del `AG` mediante los `**kwargs` del constructor.

El tiempo del algoritmo depende de, entre otras cosas, el equipo en el que se prueba (tanto hardware como software); en nuestro caso lo hemos probado en dos sistemas, estos son:

Sistema 1

- **CPU**: *AMD Ryzen 7 7000 Series*
- **Sistema operativo**: *Linux Fedora*
- **Terminal**: *Warp*

Sistema 2

- **CPU**: *Intel i7 10510U*
- **Sistema operativo**: *Windows 11*
- **Terminal**: *Powershell*

Recomendamos lanzarlo desde una terminal lo más optimizada posible, en lugar de la incorporada en un IDE.

Con los parámetros establecidos, los resultados obtenidos para cada fichero se presentan en la Tabla III para el primer sistema y en la Tabla **??** para el segundo.

Además, hemos implementado un sistema de registros que, de manera dinámica, actualiza los mejores valores obtenidos a medida que se prueban distintos parámetros o se cambian las semillas. Este registro puede consultarse en *./data/log.txt*, partiendo del directorio raíz del proyecto una vez descomprimido.

Para ejecutar el proyecto, ofrecemos dos opciones:

1) Lanzar el proyecto sin parámetros, utilizando el comando: *python ./src/prueba_AG.py*, que empleará el fichero predeterminado, en este caso *toy1*. Este fichero predeterminado puede ser modificado manualmente en el archivo *prueba_AG.py*, como se indicó en la plantilla proporcionada.
2) Modificar el fichero de manera más conveniente añadiendo el nombre del fichero como argumento en la línea de comandos: *python ./src/prueba_AG.py nombre_fichero*. Los nombres de fichero válidos son: *toy1*, *housing* y *synt1*.

Para cualquier duda sobre la instalación de dependencias o la configuración correcta del proyecto, consulte el archivo *README.md* incluido en el directorio raíz del proyecto.

TABLA II
RESULTADOS OBTENIDOS SISTEMA 1

Fichero	Toy1	Housing	Synt1
RMSE	0.3417	0.7564	190.8364
R2	0.9436	0.5739	-0.0474
Tiempo	6.62	20.61	160.97
Seed	123	124	123

TABLA III
RESULTADOS OBTENIDOS SISTEMA 2

Fichero	Toy1	Housing	Synt1
RMSE	0.3517	0.7564	190.8364
R2	0.9403	0.5739	-0.0474
Tiempo	16.29	49.65	487.17
Seed	123	124	123

V. CONCLUSIONES

En este proyecto, se ha desarrollado y evaluado un modelo de regresión no lineal utilizando algoritmos genéticos. La metodología incluyó la representación de coeficientes y exponentes en cromosomas, la aplicación de operadores genéticos como mutación y cruce, y la evaluación de la aptitud utilizando el error cuadrático medio (RMSE). Se realizaron experimentos con tres conjuntos de datos distintos: `toy1`, `housing` y `synt1`. Los resultados mostraron una mejora significativa en la precisión de las predicciones al optimizar el manejo de los números complejos con el uso de los modulos de estos, y en la velocidad de rendimiento al usar un límite de ejemplos en el entrenamiento y el almacenamiento de la valoracion de los cromosomas durante el algoritmo.

Los experimentos demostraron que el tiempo de entrenamiento varía según el tamaño del conjunto de datos y el número de variables. En particular, el conjunto de datos `synt1` presentó mayores desafíos debido a su complejidad, por lo que cualquier cambio podía suponer una diferencia notable en los resultados de este fichero. Para poder obtener los mejores resultados posibles, se llevo a cabo la implementación de un sistema de registro de records además del control de la semilla de aleatoriedad que aseguraron la reproducibilidad de los resultados más óptimos.

Para futuros trabajos, se sugiere una serie de mejoras y exploraciones adicionales. Entre estas mejoras, la paralelización de partes del código podría mejorar significativamente el rendimiento del algoritmo genético. La optimización de parámetros del algoritmo genético, como las tasas de mutación y cruce, y la incorporación de métodos avanzados de selección también podría conducir a mejoras adicionales en la precisión y eficiencia del modelo. Finalmente, sería beneficioso explorar la aplicación de este enfoque en otros dominios y con conjuntos de datos más grandes y complejos para validar su generalizabilidad y robustez.

REFERENCIAS

[1] Mathew, Tom V. "Genetic algorithm." Report submitted at IIT Bombay 53 (2012), pp 7-8.
[2] Pose, Marcos Gestal. "Introducción a los algoritmos genéticos." Departamento de Tecnologías de la Información y las Comunicaciones Universidad de Coruña (2000).
[3] van Rossum, G., and Drake, F. L. (Eds.). (2006). An introduction to Python. Network Theory Limited, pp 15-18.
[4] https://scsynth.org/t/fractional-powers-of-negative-numbers/4224