Creación Regresión Lineal mediante Algoritmos Genéticos

Julen Rodríguez Meneses

Índice

Descripción del problema

A lo largo de este trabajo vamos a averiguar cómo poder realizar una *Regresión Lineal* utilizando para ello el uso de *Algoritmos Genéticos*.

Explicación Regresión Lineal

Una *Regresión Lineal* es un algoritmo mediante el cual creamos una Inteligencia Artificial capaz de predecir un valor "y" a base de otros varios valores "x". Pensemos en la ecuación de la recta, siendo esta:

$$f(x) = ax + b$$

Ecuación de la recta

En este caso, "f(x)" / ("y") es el valor a predecir. Esto es posible gracias a que utilizamos un valor "x" conocido. El resto de los valores de la recta, "a" y "b", son parámetros fijos, utilizados para poder realizar la predicción. Si esto lo trasladamos a uno de los dataset utilizados tenemos las siguientes variables:

X		Y
age	*deficit*	*C_peptide*

Tabla de variables

Como vemos, tenemos prácticamente la misma estructura, solo que en este caso existen dos variables utilizadas en la predicción, añadiendo un nuevo parámetro nuevo a la ecuación. Una vez mostrado esto, realizamos la ecuación de la *Regresión Lineal*:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Ecuación de la Regresión Lineal

En este caso la ecuación es un poco distinta pero funcionalmente es la misma. Cada uno de los valores "C_peptide" se calcula mediante el valor de ambos valores "X" mediante la ayuda de los *hiper-paramtros* θ (theta). La principal complicación en este algoritmo es averiguar los valores de los *thetas* que obtengan los mejores resultados.

Explicación Algoritmo Genético

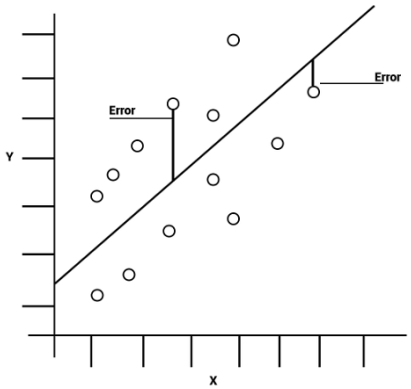
Una vez explicado cómo funciona la *Regresión Lineal*, tenemos que obtener los mejores *hiper-parametros* para que su funcionalidad sea la mejor posible. Por lo tanto, los elementos a optimizar son estos mismos. Tenemos tantos *hiper-parametros* como variables "X" más 1, siendo este el valor independiente. Utilizando el mismo dataset descrito anteriormente:

$$\underbrace{[\overbrace{\theta_0}^{Gen}, \overbrace{\theta_1}^{}, \overbrace{\theta_2}^{}]}_{Cromosoma}$$

El cromosoma que se va a utilizar a lo largo del algoritmo es bastante sencillo de comprender ya que solamente son los *hiper-paramtetros* de la *Regresión Lineal*. Cada uno de los genes son cada uno de ellos.

Función de coste

Uno de los pasos más importante del *Algoritmo Genético* que es necesario explicar es la función **fitness**. Como medida única hemos decidido tomar el **error cuadrático medio**.



Gráfica ejemplo MSE

Como se puede observar en la gráfica, mediante la *Regresión Lineal* se realiza una *recta* mediante la cual se predicen los valores de salida. Obviamente estos datos predichos no son 100% efectivos por lo que se debe comprobar el nivel de error del modelo creado. Para ello se utilizan los mismos ejemplos que se han utilizado para crear el propio modelo. La medición de este error es bastante sencilla e intuitiva. Sigue la misma dinámica para todos ejemplos como se puede comprobar en la gráfica. **Se mide la distancia entre el valor predicho y el real**, se eleva al cuadrado cada una de ellas para evitar anulaciones de errores, y se realiza la media de todos ellos.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \tilde{y})^2$$

Ecuación del Error Cuadrático Medio

El modelo que obtenga un valor mas bajo en esta métrica es el mejor de todos ellos.

Como inciso he de mencionar que, para utilizar esta métrica de rendimiento, vamos a utilizar la función **mean_squared_error** de la librería **scikit-learn** la cual es una librería especializada en métodos y funciones de aprendizaje automático.

No vamos a utilizar ninguna otra métrica de rendimiento ya que no nos interesa medir en ningún momento la complejidad del modelo o la rapidez en predecir. Lo más importante para nosotros es averiguar cual de todos los modelos creados es más útil en el momento de predecir correctamente.

Marco Experimental

Una vez explicado todo lo relacionado con la *Regresión Lineal*, la cual es el dónde vamos a intentar predecir sus hiper-parametros, es momento de empezar a experimentar con el *Algoritmo Genético*.

Pasos

Vamos a comentar cada paso existente dentro del propio *Algoritmo Genético*, los parámetros utilizados como las modificaciones a los mismos.

Población inicial

Como población inicial hemos decidido crear **10 cromosomas** de **n+1 genes**, siendo *n* la cantidad de variables que posee el dataset para predecir. Todos los genes son números aleatorios entre -1 y 1.

Selección de progenitores

Este paso es el más crucial en para el algoritmo ya que nos encargamos de elegir una cantidad de cromosomas de la población para que sigan en la ejecución. Se ha realizado 3 métodos distintos de selección:

- **Ruleta**. Este método, como bien dice su nombre, sirve como una ruleta. Simplemente obtenemos el fitness de cada cromosoma, los normalizamos para que la suma de todos ellos sea 1, y creamos una ruleta con ellos. Siendo así que la suma de cada fitness con el anterior genere intervalos. Generamos un número aleatorio entre 0 y 1, y elegimos el cromosoma de ese mismo intervalo como progenitor. Este método tiene sus propios problemas ya que puede que los valores fitness sean muy parecidos o exista uno muy predominante que no de opciones a los demás. Por ello se han realizado 3 modificaciones en la creación de la ruleta:
  - **Sin modificaciones**. Creación de la ruleta normal.
  - **Goldberg**. Se elimina el problema del fitness predominante.
  - **Windowing**. Se elimina el problema de los fitness muy similares.

- **Ranking**. Este método realiza un ranking de los fitness y da una probabilidad de elección a cada posición. Sigue la misma mecánica que la ruleta en cuanto a la elección de cromosomas.

- **Torneo**. En este método realizamos enfrentamientos entre los cromosomas 1vs1. El que mejor fitness tenga es el elegido.
  - **Con reemplazamiento**. Los cromosomas pueden enfrentarse entre sí.
  - **Sin reemplazamiento**. Los cromosomas no pueden enfrentarse entre sí

En todos los métodos elegimos tantos cromosomas para la población como la población introducida.

Cruzamiento

Para el cruzamiento tenemos una probabilidad de un **95%** que suceda. Es bastante alta por lo que el cruzamiento está prácticamente asegurado. Se trata simplemente de *cruzar* dos cromosomas dando a lugar otros dos cromosomas completamente distintos. Se han realizado 3 métodos distintos.

- **Único**. En este método se realiza un corte completo en alguna posición del cromosoma, se separa e intercambia con otro distinto.

- **Doble**. La diferencia esque no se separa una mitad entera sino una cantidad determinada de genes.

- **Múltiple**. En este caso se comprueba si se realiza el intercambio de gen en gen.

Mutación

En cuanto a la mutación, se la ha dado una probabilidad mucho mas baja, siendo de un **5%**. Hemos comprobado lo que ocurriría tanto si se realiza o no este método. Se han realizado 2 tipos de mutaciones distintas.

- **Reseteo**. Si se da la probabilidad, se modifica por completo el valor del gen por uno entre -1 y 1.

- **Desplazamiento**. La diferencia esque no se elimina el valor original, solamente se *desplaza* el valor ligeramente entre -0.15 y 0.15.

Selección supervivientes

En el último paso tenemos que elegir entre los cromosomas seleccionados como progenitores, y los creados nuevos tanto en cruzamiento y mutación. Se han realizado 3 métodos diferentes.

- **Edad**. La población superviviente son simplemente todos los cromosomas nuevos que se han creado a lo largo del algoritmo en cruzamiento y mutación.

- **Elitismo**. En este caso solamente sobreviven los cromosomas que mejor fitness den, independientemente que sean nuevos o antiguos.

- **Genitor**. Para este método se descartan los *k* peores cromosomas según su fitness, y se sustituyen por los mejores cromosomas creados nuevos.

Resultados obtenidos

Una vez explicados los distintos pasos, como sus modificaciones, nos topamos con que tenemos un total de **189 iteraciones** distintas. De entre todas ellas hemos obtenido los siguientes resultados.

Combinaciones Algoritmo	Fitness
Prog. 'ranking' **Cruz**. 'único' **Muta.** 'desplazamiento' **Supe.** 'elitismo'	0.024727
Prog. 'torneo', (NO, 3) **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'elitismo'	0.024735
Prog. 'ruleta' **Cruz.** 'múltiple' **Muta.** 'desplazamiento' **Supe.** 'elitismo'	0.024746
Prog. 'torneo', (NO, 3) **Cruz.** 'múltiple' **Muta.** 'desplazamiento' **Supe.** 'elitismo'	0.024751
Prog. 'torneo', (SI, 3) **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'elitismo'	0.024753

Top 5 mejores combinaciones

Combinaciones Algoritmo	Fitness
Prog. 'ruleta', 'windowing' **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	52.388028
Prog. 'ruleta', 'windowing' **Cruz.** 'único' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	43.070832
Prog. 'ruleta', 'windowing' **Cruz.** 'multiple' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	42.694315
Prog. 'ruleta' **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	31.983714
Prog. 'ruleta' **Cruz.** 'único' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	31.441874

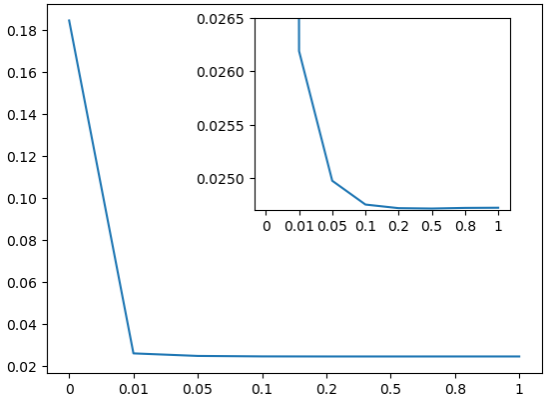
Top 5 peores combinaciones

Viendo estos resultados podemos tener varias conclusiones.

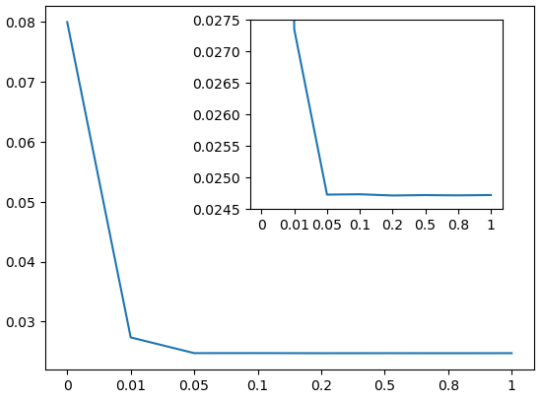
- El más claro de todos es el **gran peso** que tiene la **elección de supervivientes** en el algoritmo. Una elección de uno u otro método prácticamente empeora radicalmente los resultados obtenidos. En este caso está claro la gran superioridad de utilizar **elitismo** frente a **genitor**, incluso en su propio funcionamiento, ya que en el primero elegimos los mejores cromosomas de manera *bruta*, mientras que en el segundo intentamos ser continuistas.

- Por otro lado, es sorpresivo el efecto de los métodos de cruzamiento y mutación. En cuanto al **cruzamiento**, parece que no afecta en gran medida que método se utiliza, ya que tanto en las mejores como peores combinaciones existen los 3 indistintamente. Por otro lado, en la **mutación** solamente se utiliza el método de **desplazamiento** en ambos *tops*. Esto puede deberse a que

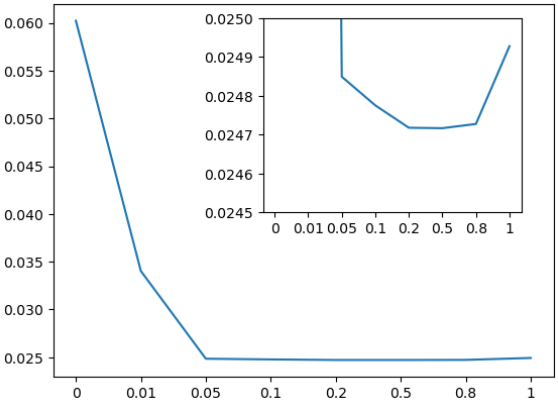
el porcentaje de uso no es excesivamente grande y no se tiene en cuenta, o que el método no es bastante significativo en cuanto a su uso. Vamos a realizar unas pruebas en cada una de las 5 mejores combinaciones para ver si el uso o no de este método de mutación afecta.



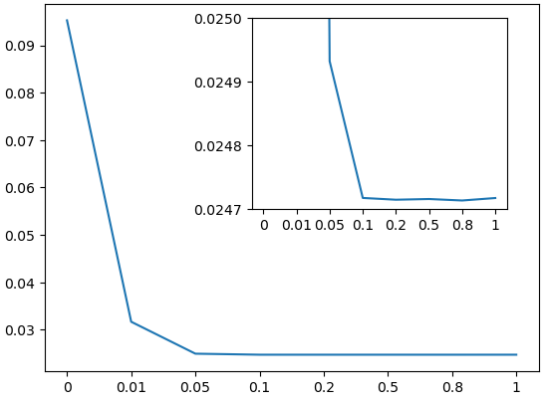
Efecto Mutación TOP 1 (Mejores)



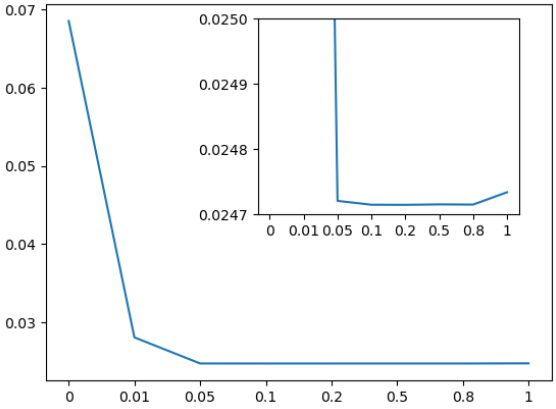
Efecto Mutación TOP 2 (Mejores)



Efecto Mutación TOP 3 (Mejores)



Efecto Mutación TOP 4 (Mejores)



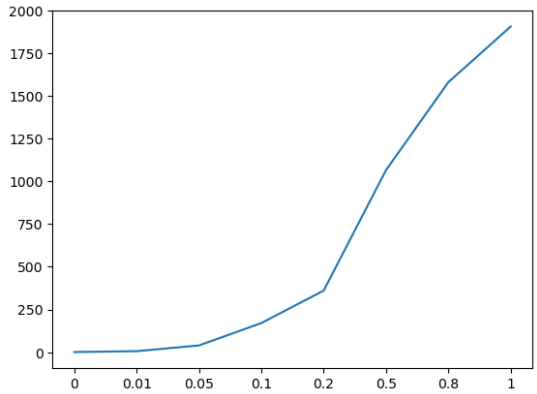
Efecto Mutación TOP 5 (Mejores)

(Eje X → Probabilidad mutación // Eje Y → Fitness obtenido)

Como se puede observar, definitivamente utilizar **mutación** afecta en gran medida al rendimiento del modelo. El método puede ser o no ser el causante del gran cambio, tanto como si lo es el propio método. Se puede observar a

simple vista que el paso de no usar este método a usarlo con una probabilidad muy baja aumenta en gran medida los resultados obtenidos. Al contrario, destaca tambien el efecto contrario, pero en una escala mucho menor. Asegurar al 100% su uso aumenta el fitness, pero no llega a los niveles iniciales ni por asomo. Un efecto interesante de porque este aumento de probabilidad de mutar mejore el valor fitness, puede deberse al uso de selección de progenitores del **elitismo**. Este método elige a los mejores cromosomas y elimina radicalmente a los peores, si realiza una mutación, es probable que pueda mejorar el fitness. Al final, todo suma.

Está claro que los **mejores resultados** obtenidos son sobre el porcentaje **0.2 y 0.5**. Veamos si ocurre lo mismo con el TOP 1 de los peores modelos.



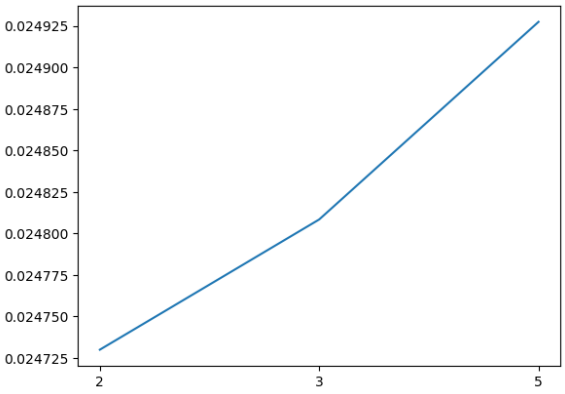
Efecto Mutación TOP 1 (Peores)

(Eje X → Probabilidad mutación // Eje Y → Fitness obtenido)

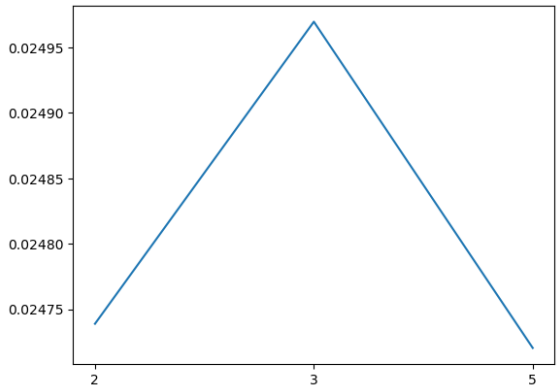
En este caso, los cambios producidos han ido a absolutamente peores resultados. Obteniendo cada vez peores y peores fitness cuanto mas aumenta la probabilidad de mutar. Esto lo podemos atribuir a que **el uso de la mutación no mejora ni empeora un modelo en general, solamente acentúa y aumenta sus cualidades.**

- Otro factor interesante por tratar es como afecta las variaciones al método del **torneo**, tanto si cambiamos el valor de los *k* contrincantes, como el uso de r*eemplazamiento*. Vamos a realizar unas pruebas y mostrar los resultados obtenidos, usando 3 tipos de contrincantes y el uso de *reemplazamiento*.

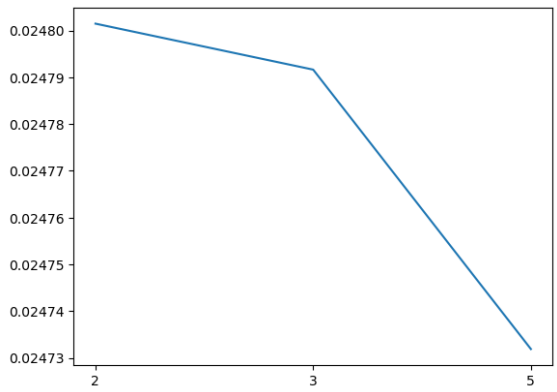
SIN Reemplazamiento



Efecto Torneo TOP 2 (Mejores)

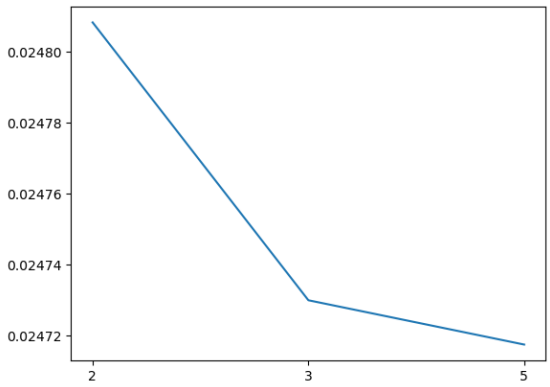


Efecto Torneo TOP 4 (Mejores)

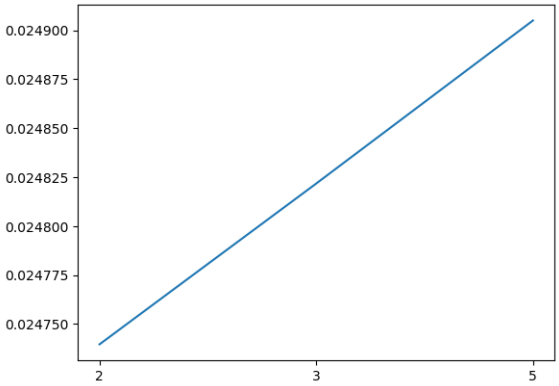


Efecto Torneo TOP 5 (Mejores)

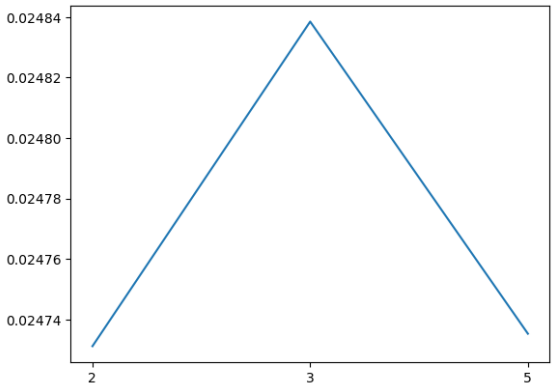
CON Reemplazamiento



Efecto Torneo TOP 2 (Mejores)



Efecto Torneo TOP 4 (Mejores)



Efecto Torneo TOP 5 (Mejores)

(Eje X → *k* contrincantes // Eje Y → Fitness obtenido)

No se puede observar unas diferencias claras como si se hizo anteriormente con la mutación. Los **cambios** de fitness producidos, en los 6 ejemplos, han sido en la **4 cifra decimal**. Esto no es tan alto, pero en un algoritmo donde se ha elegido un modelo sobre otro por la 5 cifra decimal, es importante tener

esta precisión milimétrica. Es recomendable probar todas las combinaciones distintas hasta obtener el mejor resultado.

Análisis mejor modelo

Una vez visto todo tipo de combinaciones de parámetros, así como modificaciones distintas a los mejores resultados, sin duda nos quedamos con esta combinación de *hiper-parametros* en cuanto a mutación.

Combinaciones Algoritmo	Fitness
Prog. 'torneo', (NO, 3) **Cruz.** 'doble' **Muta.** 'desplazamiento', (0.2 prob) **Supe.** 'elitismo'	0.024713

Y en cuanto a los cambios sugeridos en torneo.

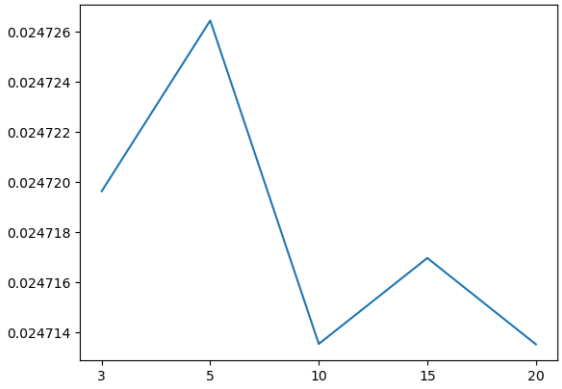
Combinaciones Algoritmo	Fitness
Prog. 'torneo', (SI, 5) **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'elitismo'	0.024717

Vemos que ambos pertenecen al *top 2* realizado con las combinaciones originales. Al realizar unos mínimos cambios, disminuye incluso más el valor de fitness. Veamos cual es el resultado si se juntan las dos modificaciones realizadas.

Combinaciones Algoritmo	Fitness
Prog. 'torneo', (SI, 5) **Cruz.** 'doble' **Muta.** 'desplazamiento' (0.2 prob) **Supe.** 'elitismo'	0.024714

Por cambios en la 6 cifra, nos quedamos como mejor modelo el obtenido en la primera modificación de la mutación.

La última modificación que nos interesa realizar para averiguar si se pudiera lograr una mejora o no, sería observar que ocurre si **cambiamos el número de cromosomas existente en la población inicial**.



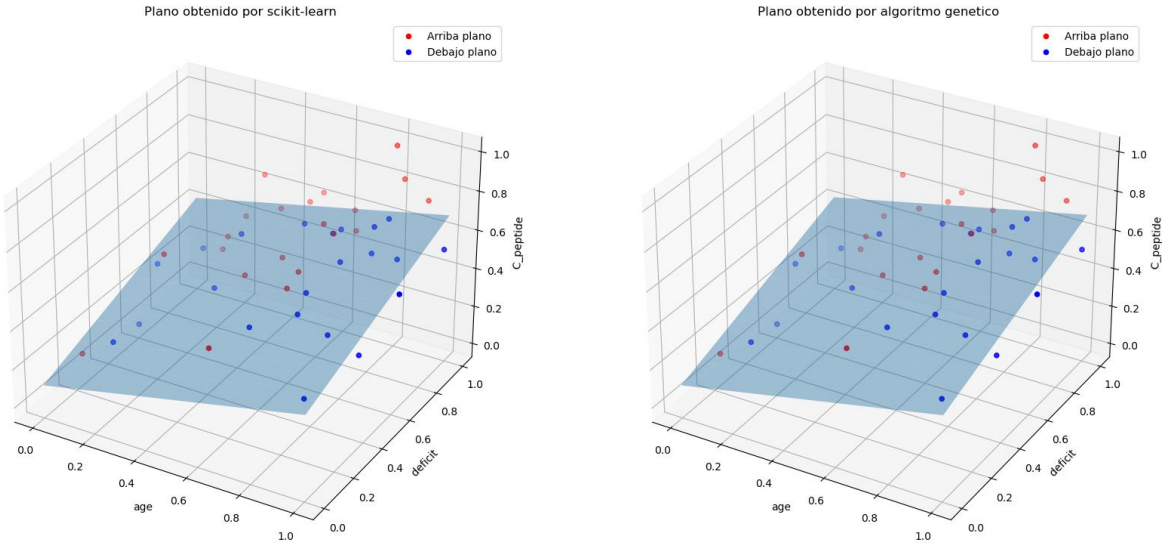
Efecto número cromosomas en población

(Eje X → número cromosomas // Eje Y → Fitness obtenido)

Los resultados difieren suficiente si se modifica o no la población, dando en este caso prácticamente el mismo valor fitness tanto con 10 cromosomas, como con 20 cromosomas. Vamos ha quedarnos con 10 cromosomas (iteración original).

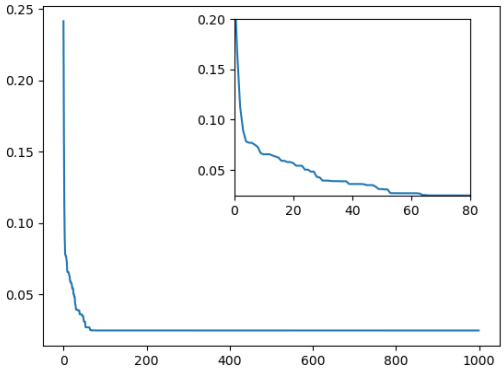
Comparación con scikit-learn

Como última medida de rendimiento, vamos ha realizar una comparación de este modelo con el realizado con **scikit-learn**. Esta librería tiene un modelo el cual puede crear *Regresiones Logísticas* de manera mucho mas precisa a la nuestra. Vamos ha tomar este modelo como el ideal y al que el nuestro debería semejarse lo máximo.



Graficas comparación scikit-learn con algoritmo genético

Salta a simple vista lo tremendamente parecidas con son ambas gráficas, incluso parecen copiadas. Esto se debe a los cambios, modificaciones e iteraciones que hemos ido realizando al *Algoritmo Génetico*, que hemos llegado a una solución prácticamente igual a la ideal.



Descenso valor fitness

(Eje X → iteraciones algoritmo // Eje Y → Fitness obtenido)

Por último, podemos ver como desciende el valor del fitness a lo largo del algoritmo, siendo así claramente hasta la iteración 70 mas o menos. A partir de ese momento, el cambio de valores es muchisimo más insignificante en comparación.

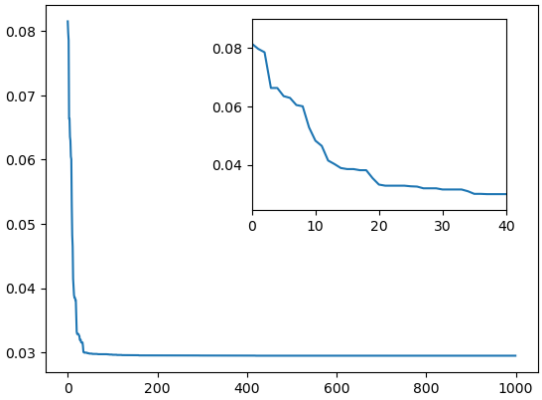
Rendimiento con otros datasets

Disponemos de otros 2 *datasets* distintos con los que comprobar nuestro modelo. Vamos ha utilizar para ello el dictado anteriormente como mejor y peor modelo

Dataset con 3 variables

Algoritmo	Fitness
Prog. 'torneo', (NO, 3) **Cruz.** 'doble' **Muta.** 'desplazamiento', (0.2 pb) **Supe.** 'elitismo'	0.029478

Algoritmo	Fitness
Prog. 'ruleta', 'windowing' **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	54.352618



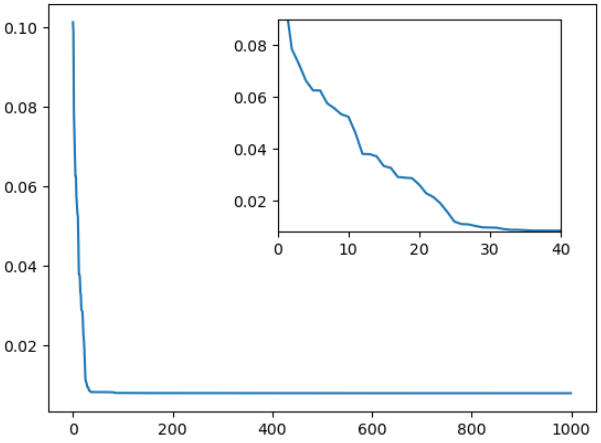
Descenso valor fitness (Mejor caso)

(Eje X → iteraciones algoritmo // Eje Y → Fitness obtenido)

Dataset con 4 variables

Algoritmo	Fitness
Prog. 'torneo', (NO, 3) **Cruz.** 'doble' **Muta.** 'desplazamiento', (0.2 pb) **Supe.** 'elitismo'	0.008066

Algoritmo	Fitness
Prog. 'ruleta', 'windowing' **Cruz.** 'doble' **Muta.** 'desplazamiento' **Supe.** 'genitor', 4	27.049497



Descenso valor fitness (Mejor caso)

(Eje X → iteraciones algoritmo // Eje Y → Fitness obtenido)

Conclusiones y líneas futuras

Se han realizado diversos experimentos, con muchas combinaciones de *hiper-parametros*, asi como modificaciones y permutaciones sobre estos mismos. De todo esto podemos obtener diversas conclusiones.

- Primero de todo, he de mencionar que cada detalle o mínimo cambio que se produzca dentro del algoritmo genético cuenta. Un simple cambio en uno de los tantos *hiper-parametros* da unos resultados completamente distintos. Un ejemplo claro de esto es lo visto mediante el **uso o no de la mutación**.

- Otro punto muy interesante se trata de la diversa y compleja combinación de métodos, y diversificaciones que existen dentro de los mismo, que se pueden dar entre sí. Está comprobado que unas combinaciones simplemente obtienen más *química* entre sí que otras. Una mala combinación de métodos con los que trabajar dan un algoritmo que no sabe cómo trabajar eficientemente, y como resultado los resultados pueden llegar a ser terribles. Esto se ha comprobado perfectamente **con el top mejores y peores resultados descrito anteriormente**.

- Por último, mencionar los resultados obtenidos con los otros dos *datasets* distintos. Los valores fitness han seguido bien lo obtenido anteriormente, dando como mejor y peor combinación, valores muy bajos y altos. Esto puede deberse ya que se **ha podido encontrar unas muy eficientes y poco eficientes combinaciones de *hiper-parametros* a lo largo de la investigación**.

Para finalizar, como mejora a futuro, podemos probar incluso más tipos de combinaciones, así como volver a realizar todas estas con otro tipo de *datasets* con distintos ejemplos y variables. Existe la posibilidad de que hayamos sobre-entrenado sobre los datos originales y se puedan conseguir mejores resultados con distintas combinaciones en diferentes *datasets*.