

**Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)**

**Momento de Retroalimentación: Módulo 2**

**Implementación de una técnica de aprendizaje máquina sin el uso de un framework. (Portafolio Implementación)**

**Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución. (Portafolio Implementación)**

**Maxime Vilcocq Parra**

**A01710550**

Contenido

[Implementación de una técnica de aprendizaje máquina sin el uso de un framework. 3](#_Toc208785822)

[Gradient descent: 3](#_Toc208785823)

[Elaboración de código sin framework: 5](#_Toc208785824)

[Train, validate y test: 12](#_Toc208785825)

[Cómo evaluamos el modelo: 12](#_Toc208785826)

[En código: 13](#_Toc208785827)

[Estandarización 16](#_Toc208785828)

[Resultados: 18](#_Toc208785829)

[Resultados y comparación del Modelo: 19](#_Toc208785830)

[Mejora: 20](#_Toc208785831)

[¿Qué son los hiperparámetros? 20](#_Toc208785832)

[Resultados de mejora: 21](#_Toc208785833)

[Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución. 22](#_Toc208785834)

[Pytorch: 22](#_Toc208785835)

# Implementación de una técnica de aprendizaje máquina sin el uso de un framework.

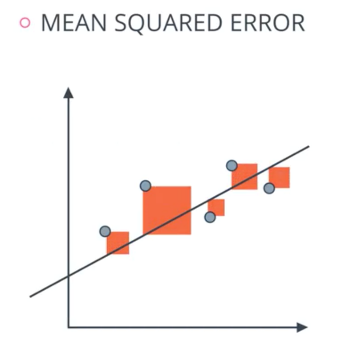
Implementación de una técnica de aprendizaje máquina sin el uso de un framework:

Para esta entrega estaré programando el algoritmo de gradient descent para generar un modelo predictivo (calculando coeficientes de distintas variables) reduciendo el MSE (mean squared error).

Para mi programa estaré utilizando el dataset de [abalone](https://archive.ics.uci.edu/dataset/1/abalone), el cual es un set de datos para predecir la edad de este tipo de árbol a partir de medidas físicas (en lugar del tedioso método tradicional). Sin embargo, el código debe de funcionar con otros sets de datos siempre y cuando se introduzcan de forma correcta.

## Gradient descent:

El algoritmo de gradient descent nos permite crear un modelo predictivo al reducir el error (MSE) de un modelo predictivo inicial (propuesto por nosotros) hasta que ese error sea el más bajo posible.

Nota: en ocasiones no es posible llegar a un MSE de 0, esto se puede deber a tener datos insuficientes (ya sea que falten considerar variables, o hacer más mediciones). Cuando esto sucede es probable que no conozcamos uno o más factores que impactan a nuestra variable dependiente.

La siguiente imagen del blog de Krystian Safjan (["Comprehensive Guide to Interpreting R\xB2, MSE, and RMSE for Regression Models."](https://safjan.com/interpreting-r2-mse-rmse-for-regression-models/)). Muestra visualmente como se ve este MSE. La línea recta es un modelo de predicción y mientras más cerca estén los puntos (valores reales de la variable independiente), menor será el área naranja (MSE).

De forma matemática el mean squared error se calcula de la siguiente manera (imagenes sacada del video [Machine Learning Tutorial Python - 4: Gradient Descent and Cost Function](https://www.youtube.com/watch?v=vsWrXfO3wWw&t=311s)):  
Diagrama, Texto

Descripción generada automáticamenteDiagrama, Texto

Descripción generada automáticamente con confianza media

Esto es de forma literal, restarle cada Y que predice nuestro modelo a cada Y real del set de datos, elevar cada resta al cuadrado (evita que los valores negativos cancelen a los positivos), sumar los resultados y promediar el total de dicha suma.

Lo que queremos es ver cómo cada una de nuestros coeficientes “m” y nuestro bias “b” afectan este mean squared error. Para ello derivamos MSE con respecto a cada “m” y al “b”.

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza media

Imagen sacada del video [Machine Learning Tutorial Python - 4: Gradient Descent and Cost Function](https://www.youtube.com/watch?v=vsWrXfO3wWw&t=311s).

Esta derivada nos dice la dirección a la que cada bias y coeficiente está moviendo nuestro MSE, la idea es irnos moviendo hacia el MSE más pequeño posible.

Esto se consigue utilizando un step (que es lo mismo que el learning rate), que nos dice qué tanto movernos hacia la dirección que hace que el MSE disminuya.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Imagenes tomadas del video [Machine Learning Tutorial Python - 4: Gradient Descent and Cost Function](https://www.youtube.com/watch?v=vsWrXfO3wWw&t=311s).

En la imagen anterior podemos ver como el valor de b se va actualizando en cada iteración obteniendo un MSE cada vez más pequeño.

El algoritmo de gradient descent suele requerir de varias épocas o iteraciones para llegar a un buen resultado.

### Elaboración de código sin framework:

Estoy empleando numpy simplemente para manejar los datos en matrices y facilitar el manejo de estos para hacer producto punto entre estas. En otras palabras, para poder multiplicar cada coeficiente de nuestras variables independientes por su respectivo valor y con eso obtener una predicción. Si el código de Python no funciona al ejecutarse, es probable que falte hacer pip install de numpy.

1. El primer paso es identificar y separar nuestra variable dependiente (Y) de las variables independientes (nuestras X) y cargarlas.

En el caso del dataset de abalone, el valor que queremos conseguir es el número de anillos que tiene el árbol (con eso se determina la edad). Además el dataset incluye la columna de Sex que es una variable no cuantitativa por lo que se requiere hacer un hot encoding para poder procesar el dato.

Carga el archivo:

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Convierte de cualitativo a cuantitativo:

Texto

Descripción generada automáticamente

Separa la variable dependiente y junta todas las independientes en una sola matriz:

Texto

Descripción generada automáticamente

1. Darles un valor inicial a nuestros pesos (los coeficientes de cada una de nuestras variables independientes) y a nuestro bias.

Para esto estoy creando un vector de coeficientes, un coeficiente por cada variable independiente, todos inicializados en 0.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

n\_samples son las filas y n\_features, las columnas.

1. Saco el cálculo de la predicción con el producto punto.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente 

En esta línea se está multiplicando cada fila de X (cada fila tiene n columnas, una por cada variable independiente) por su respectivo coeficiente dentro del vector de pesos w y luego le suma b.

Lo que obtiene la variable y\_pred es un arreglo de predicciones del mismo tamaño que las filas de nuestro set de datos. Es decir, si tenemos 15 filas en el dataset, habrá 15 predicciones dentro de y\_pred.

1. Sacamos el MSE con la predicción recién calculada:  
   Texto

   Descripción generada automáticamente

Diagrama, Texto

Descripción generada automáticamente

A cada valor real de y se le resta el valor de la predicción y se eleva al cuadrado (esto nos deja un arreglo de resultados). Con np.sum sumamos cada valor del arreglo para tener un solo valor numérico el cual se divide entre el número de filas para saber el promedio (MSE).

1. Sacamos un vector de derivadas para saber cuánto y cómo (+/-) influye cada variable independiente y el bias a nuestra variable dependiente.

Texto

Descripción generada automáticamente

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza media

dw nos muestra un vector (usa np.dot) con la magnitud y signo de la derivada respecto a cada variable independiente. Db (usa.sum) nos da únicamente un valor el cual representa el impacto del bias en el MSE.

1. Utilizando un learning rate actualizamos el valor de nuestro bias y nuestros coeficientes.



Aquí uso un learning rate default de .001.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Finalmente repetimos este proceso las veces que queramos actualizando cada vez los valores del bias y los pesos.

El default está colocado en 25000. Texto

Descripción generada automáticamente

1. Una vez concluidas las épocas regresamos el vector de pesos y el bias final.



Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

En la foto anterior uso “X” y “y” pero en la aplicación real uso el “X” y “y” de mi dataset de entrenamiento.

Ya que tenemos los valores podemos comenzar a hacer predicciones.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. Crear un modelo funcional.

Para ver que tanto se ajusta a nuevos valores, separamos nuestro dataset en dos, uno para entrenar y otro para probar.

Texto

Descripción generada automáticamente

El código funciona igual solo que en lugar de enviar toda el dataset, enviamos solamente la X y Y de entrenamiento para obtener coeficientes. Después con los coeficientes obtenidos y los valores de X y Y de test podemos comparar los valores que predecimos con valores reales.

1. Predecir y comparar

Con los coeficientes obtenidos y el segmento de prueba comparamos los valores reales con los que pronosticamos.

Estoy utilizando R squared,

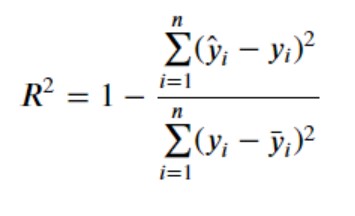


Imagen de [R-squared in Linear Regression Models: Concepts, Examples](https://vitalflux.com/r-squared-explained-machine-learning/).

Texto

Descripción generada automáticamente

Este valor nos sirve para explicar qué tanto nuestro modelo explica la variación de nuestra variable dependiente. Va de 0 a 1, donde un 1 significa que el modelo es perfecto.

1. Graficar los resultados.

Utilicé matplotlib para mostrar un visual que nos permite entender más fácilmente nuestra predicción contra los valores reales.

Texto

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

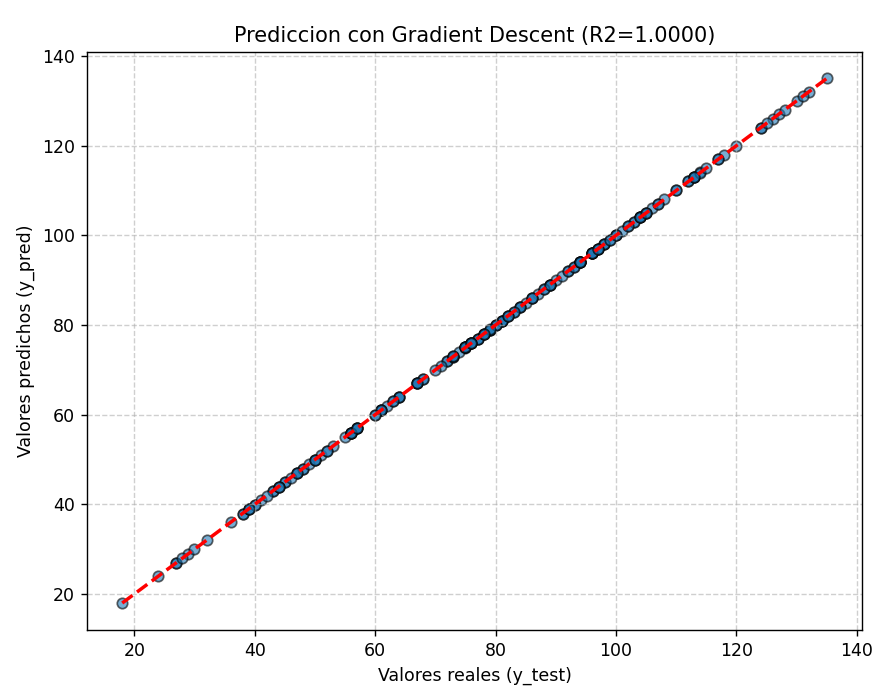
1. Comprobar los datos y el funcionamiento del código:

Tras ejecutar mi código con el set de datos de abalone, el resultado obtenido es un R2 de .4 lo que significa que mi modelo predice un alrededor de un 40% de la variabilidad. Este valor no es muy alto, esto probablemente se debe a que estamos tratando de hacer una regresión lineal y que otro modelo de regresión (por ejemplo, un exponencial) podría ajustarse mejor al dataset.

Sin embargo, esto no significa que el código no funciona, para demostrar esto creé un dataset artificial con valores estrictamente lineares. Podemos verificar si el código funciona si vemos que los coeficientes son iguales a los que utilicé. Por ser un set de datos artificial, se puede alcanzar un MSE de casi 0, y una R2 de casi 1.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## Train, validate y test:

Actualmente, el código separa el dataset en dos partes. Una para entrenar y otra para probar el modelo entrenado. El punto de esto es que los valores con los que se prueba no afecten el entrenamiento (overfitting). Sin embargo, esto no es suficiente para confiar en que nuestro modelo funcionará correctamente cuando se enfrente a nuevos datos. Por ello, agregamos una capa extra llamada validation.

Validation no es más que un test adicional. A nuestro set de entrenamiento le quitamos un fragmento para realizar una prueba adicional. Esto nos deja con un set de entrenamiento ligeramente más pequeño, pero nos da la posibilidad de probar con un set de datos adicional. Tras comparar los valores de R2 que se obtienen de probar dentro de training, de validate y de test podemos ver si nuestro modelo realmente funciona o si se está sobre ajustando. Además, podemos comparar como evoluciona el MSE de los sets de validate y de train en cada iteración para identificar si hay un underfit o un overfit. Unerfit si ambas líneas se encuentran muy altas y overfit si ambas gráficas son muy distintas.

### Cómo evaluamos el modelo:

Bias: Qué tanto aprende el modelo (que tanta variabilidad es capaz de modelar). Si el R2 es alto >.7, entonces el bias es bajo. Es decir, el modelo aprende bien los patrones presentes en el data set. Si el R2<.3 entonces el bias es alto y por ello no aprende bien los patrones. Un punto entre .3 y .7 es un aprendizaje limitado.

Varianza: Qué tan sensible es el modelo a los datos de entrenamiento. Si la varianza es muy alta, significa que el modelo se está “memorizando” los datos de entrenamiento. En este problema estoy utilizando una diferencia de R2 entre un set de pruebas y otro para calcularla. Si la diferencia de R2 es grande, es porque se está aprendiendo los valores de entrenamiento y no es capaz de hacer una buena predicción con nuevos valores.

Ajuste: Overfitting, underfitting y fit. Si el bias es alto es underfit (no captura el patrón de comportamiento), si el bias es bajo, pero la varianza alta es overfit (se memoriza datos de entrenamiento y falla con datos externos). Finalmente un modelo fit, tendría un bias de mediano a bajo (es mejor si es bajo) y una varianza baja (es bueno para generalizar).

### En código:

Texto

Descripción generada automáticamente

Hacemos una separación adicional para validación.

Texto

Descripción generada automáticamente

Calculamos la R2 de nuestra predicción en validación y en test y con eso calculamos el Bias, la varianza y el ajuste (fit).

Texto

Descripción generada automáticamente

Separamos los datos, hacemos predicciones para train, validate y test y corremos el diagnóstico para saber como se comporta el modelo dentro de train a comparación de validate.

### Estandarización

Para mejorar la R2 de mi modelo estoy haciendo una estandarización z (z-Standardization).

A cada valor de x le resto la media de ese valor y lo divido entre la desviación estándar del mismo.

Este tipo de estandarización es útil para mi data set ya que permite hacer que variables medidas de distintas formas sean comparables (por ejemplo milímetros contra gramos [situación real del data set de Abalone]).

En pocas palabras lo que hace esta normalización es cambiar la magnitud de los valores por una medida de qué tan diferente se comportó una variable a su promedio. Esto hace que variables con magnitudes grandes y pequeñas tengan la misma importancia para el modelo.

Esto no cambia la relación que existe entre las variables dependientes e independientes, simplemente cambia como el modelo ve la magnitud de cada variable (una magnitud grande puede afectar el step del gradiente, haciendo que una variable sea “priorizada” sobre otra).

En código:  
Texto

Descripción generada automáticamente

Hacemos una función para estandarizar los valores de X.

Texto

Descripción generada automáticamente

Le aplicamos la estandarización a las X (variables independientes).

## Resultados:

*Test y train solamente:*  
Gráfico

Descripción generada automáticamente

Si corremos el código únicamente con un set de entrenamiento y otro de prueba, obtenemos una R2 de .405. Sin embargo, como solo hay un set para probar, no sabemos realmente si estamos cayendo en un overfitting, underfitting o si está bien nuestro fit. Por ello, agregamos un nuevo segmento llamado validate para hacer pruebas.

|  |  |
| --- | --- |
| MSE de validate vs train | MSE de validate vs train (data set estandarizado). |
| Gráfico  Descripción generada automáticamente | Gráfico  Descripción generada automáticamente |
| Como podemos ver en la gráfica el error disminuye significativamente y ambas líneas son muy parecidas por lo que no hay un underfitting ni un overfitting significativo. | Como podemos ver en la gráfica el error disminuye significativamente y ambas líneas son muy parecidas por lo que no hay un underfitting ni un overfitting significativo.  Además, podemos ver que la línea de train es ligeramente más cerca a la de validation. |

### Resultados y comparación del Modelo:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Train** | **Validate** | **Test** |
| **Gráfica** | Gráfico  Descripción generada automáticamente | Gráfico, Gráfico de dispersión  Descripción generada automáticamente | Gráfico  Descripción generada automáticamente |
| **R2** | .3755 | .3729 | .4085 |
| **MSE** | .6838 | 5.6165 | 6.4050 |
| **Bias** | Medio (modelo limitado) | Medio (modelo limitado) | Medio (modelo limitado) |
| **Varianza** | La varianza entre los 3 modelos es muy baja (la R2 y el MSE son prácticamente iguales), esto significa que no hay una memorización del set de entrenamiento. | | |
| **Fit** | El fit está entre underfit y fit. La R2 es algo baja por lo que el patrón que está aprendiendo el modelo no es capaz de representar la mayoría de la variabilidad de los datos reales. Sin embargo la varianza es muy buena ya que casi no cambia entre un set y otro. | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Train (Estandarizado)** | **Val (Estandarizado)** | **Test (Estandarizado)** |
| **Gráfica** | Gráfico, Gráfico de dispersión  Descripción generada automáticamente | Gráfico, Gráfico de dispersión  Descripción generada automáticamente | Gráfico, Gráfico de dispersión  Descripción generada automáticamente |
| **R2** | .5320 | .4978 | .5331 |
| **MSE** | 5.0085 | 4.4980 | 5.0556 |
| **Bias** | Medio (modelo limitado) | Medio (modelo limitado) | Medio (modelo limitado) |
| **Varianza** | La varianza entre los 3 modelos es muy baja (la R2 y el MSE son prácticamente iguales), esto significa que no hay una memorización del set de entrenamiento. | | |
| **Fit** | El fit está entre underfit y fit. La R2 es algo baja por lo que el patrón que está aprendiendo el modelo no es capaz de representar gran parte de la variabilidad de los datos reales. Sin embargo, estos valores son mucho mejores que el set sin estandarización. La varianza sigue siendo muy buena ya que casi no cambia entre un set y otro. | | |

## Mejora:

Para obtener buenos resultados con el algoritmo de gradient descent, lo que se puede hacer (además de estandarizar los datos) es mover los hiperparámetros.

¿Qué son los hiperparámetros?  
Son literalmente valores arbitrarios que nosotros elegimos.

* Número de epochs (iteraciones).
* Learning rate (tamaño del step).
* Pesos iniciales. (Puede hacer que converja más rápido, pero no tiene tanto impacto en una regresión lineal).
* Batch size. (Mientras más grande es más lento, pero generaliza mejor).

Para nuestro modelo no moveremos los pesos iniciales ni el batch size ya que el impacto de estos en nuestro algoritmo y con nuestro set de datos no representaría una mejora tan significativa.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Epochs |  | Learning rate | | | |
| .00001 | .0001 | .001 | .01 |
| 10000 |  |  |  |  |
| 15000 |  |  |  |  |
| 20000 |  |  |  |  |
| 25000 |  |  |  |  |

Aquí podemos ver cómo va evolucionando los resultados con distintos parámetros. Con un learning rate bajo, se requieren de muchísimas iteraciones por lo que no llegamos a un valor de R2 útil.

Con un learning rate más alto, el modelo puede convergir más rápido, por lo que podemos llegar en menos iteraciones a un buen resultado.

Con 15000 epochs y un learning rate (lr) de .01, podemos llegar a una R2 de .5465, en lugar de .5331 con el learning rate de .001 y 20000 iteraciones.

Por pura experimentación intenté correr 150000 épocas, el resultado dejo de cambiar en la iteración 120000 y se quedó en una R2 de .5459 (prácticamente la misma, pero tomando 8 veces más tiempo para entrenar).

En mi caso, elegí quedarme con 15000 iteraciones y learning rate de .01, ya que el valor de R2 es muy cercano a obtenido con 150000 épocas y el mismo coeficiente de aprendizaje, y el valor de R2 es superior al obtenido con el resto de las combinaciones de epochs y lr.

**Comportamiento de MSE:**

|  |  |  |
| --- | --- | --- |
|  | Learning rate bajo (.00001) | Learning rate alto (.01) |
| Pocas epochs (10000) |  |  |
| Muchas epochs (250000) | Gráfico  Descripción generada automáticamente |  |

Con un lr alto, la diferencia es mínima cuando se comparan épocas más grandes (incluso con 2000 iteraciones el resultado es bastante similar al obtenido con 15000). Como mi data set no es demasiado grande estas 15000 iteraciones no toman tanto tiempo, pero en data sets más grandes sí podría haber una diferencia significativa entre usar más o menos epochs.

### Resultados de mejora:

|  |  |  |  |
| --- | --- | --- | --- |
| MSE in training: | Train: | Validate: | Test: |
| Convergencia rápida. | R2=.5444 | R2=.4950 | R2=.5465 |
| A pesar de que el bias sigue siendo medio, es un poco más bajo que antes ya que la R2 es más grande. Además, la varianza sigue igual. El modelo sigue estando entre fit y underfit. | | | |

# Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución.

## Pytorch:

Para esta entrega estaré usando pytorch, un framework de meta para machine learning.

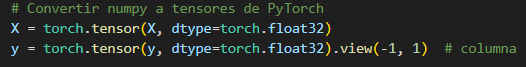
Texto

Descripción generada automáticamente

En el código, lo que estamos haciendo es exactamente lo mismo que antes, pero con funciones ya incluidas en la biblioteca de PyTorch.

Qué hace PyTorch: Hace el cálculo de derivadas y la actualización de parámetros.

PyTorch maneja tensores en lugar de arrays, por lo que convertimos nuestros datos a este formato.



Definimos el modelo que vamos a usar: linear y le decimos que vamos a tener un bias (termino independiente b). El n\_features es la cantidad de variables independientes.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Definimos el criterio que vamos a usar, para gradient descent se usa la perdida de MSE.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Definimos como vamos a optimizar el modelo: SGD=Stochastic Gradient Descent, con nuestro learning rate.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Entrenamos el modelo por el número de iteraciones:  
Texto

Descripción generada automáticamente

Lo siguiente corresponde al forward pass:

Texto

Descripción generada automáticamente

Se calcula la perdida (MSE).

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

loss.backward se encarga de derivar para saber el cambio de la pérdida respecto a nuestras w y nuestras b.

Texto

Descripción generada automáticamente

Finalmente, extraemos los datos para regresar cada coeficiente.

Texto

Descripción generada automáticamente

Al graficar los resultados, obtenemos los mismos resultados (resultados muy parecidos) que para nuestra implementación manual. Se muestran una versión usando torch con valores no estandarizados y estandarizados.

Gráfico

Descripción generada automáticamente Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Tras usar los hiperparámetros de la mejora (lr=.01, iteraciones=15000):

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Los resultados son los mismos que los obtenidos antes con los mismos hiperparámetros.