Facultad de Ciencias Económicas y Empresariales.

Análisis de la dependencia a la semilla en modelos de redes neuronales con análisis post-hoc basado en NeuralSens:

implicaciones para los modelos de rendimiento académico y abandono en educación superior.

Pablo Cadierno Redondo

5º E3 + Analytics

Director: José Luis Arroyo Barriguete.

Madrid, 2024-25

Quiero expresar mi agradecimiento, en primer lugar, al profesor José Luis Arroyo Barrigüete, no solo por su orientación, paciencia y valiosas sugerencias durante el proceso, sino también por darme la oportunidad de realizar este trabajo tan especial.

Quiero agradecer también a mis padres, pues sin ellos nada de esto habría sido posible. Gracias por las oportunidades que me habéis ido dando a lo largo de mi vida y por vuestro apoyo incondicional.

Resumen

Este trabajo estudia la dependencia a la semilla en modelos de redes neuronales (NN) aplicados al análisis post-hoc mediante la herramienta NeuralSens, que permite dotar de explicabilidad a estos modelos a través del cálculo de derivadas parciales. Las redes neuronales, especialmente en campos como las ciencias sociales y la educación, suelen operar en contextos con altos niveles de ruido, lo que puede comprometer la estabilidad y fiabilidad de los análisis interpretativos. Si bien este problema ha sido señalado en modelos agnósticos como SHAP o LIME, no se había abordado específicamente en el contexto de NeuralSens, una herramienta centrada exclusivamente en redes neuronales.

Para ello, se plantea una experimentación basada en tres escenarios con complejidad creciente: una relación lineal definida, la introducción de una variable irrelevante y la inclusión de una variable cuadrática adicional. En cada caso, se comparan los coeficientes estimados por una red neuronal con los obtenidos mediante un modelo clásico de regresión lineal, incrementando progresivamente el nivel de ruido en los datos. Esta variabilidad se evalúa a través del Signal-to-Noise Ratio (SNR), que permite cuantificar el impacto de la señal frente al ruido en los modelos entrenados.

El objetivo principal del estudio es analizar cómo afecta la aleatoriedad de la semilla a la estabilidad del análisis de sensibilidades post-hoc, así como proponer estrategias para mitigar los efectos de dicha aleatoriedad en entornos con ruido. En este contexto, se identifican y caracterizan dos tipos de riesgo asociados al uso de una única semilla: uno de carácter catastrófico y otro vinculado a la fluctuación sistemática. Finalmente, se plantea una propuesta metodológica basada en el uso de múltiples redes entrenadas con distintas semillas, con el fin de mejorar la estabilidad y fiabilidad de los resultados obtenidos mediante NeuralSens, especialmente en aplicaciones educativas o sociales donde los datos presentan alto ruido.

Palabras Clave

Redes neuronales, análisis post-hoc, NeuralSens, sensibilidad, semilla, ruido, explicabilidad, regresión lineal, análisis educativo, SNR, estabilidad, machine learning.

Abstract

This paper studies the seed dependence in neural network (NN) models applied to post-hoc analysis using the NeuralSens tool, which allows them to provide explainability to these models through the computation of partial derivatives. Neural networks, especially in fields such as social sciences and education, often operate in contexts with high levels of noise, which can compromise the stability and reliability of interpretive analyses. While this problem has been pointed out in agnostic models such as SHAP or LIME, it had not been specifically addressed in the context of NeuralSens, a tool focused exclusively on neural networks.

To this end, an experimentation based on three scenarios with increasing complexity is proposed: a defined linear relationship, the introduction of an irrelevant variable and the inclusion of an additional quadratic variable. In each case, the coefficients estimated by a neural network are compared with those obtained by a classical linear regression model, progressively increasing the level of noise in the data. This variability is evaluated through the Signal-to-Noise Ratio (SNR), which allows us to quantify the impact of the signal versus the noise in the trained models.

The main objective of the study is to analyze how seed randomness affects the stability of the post-hoc sensitivity analysis, as well as to propose strategies to mitigate the effects of such randomness in noisy environments. In this context, two types of risk associated with the use of a single seed are identified and characterized: one of a catastrophic nature and the other linked to systematic fluctuation. Finally, a methodological proposal based on the use of multiple networks trained with different seeds is proposed, in order to improve the stability and reliability of the results obtained by NeuralSens, especially in educational or social applications where the data present high noise.

Key Words

Neural networks, post-hoc analysis, NeuralSens, sensitivity, seed, noise, explainability, linear regression, educational analysis, SNR, stability, machine learning.

Graphical abstract

Análisis de la dependencia a la semilla en modelos de redes neuronales con análisis post-hoc basado en NeuralSens

Contexto

El análisis post-hoc de redes neuronales (NN) mediante el cálculo de sensibilidades (NeuralSens, Pizarroso et al., 2022) dota de explicabilidad a este tipo de modelos, considerados tradicionalmente como "cajas negras". Esto permite el uso de NN con un enfoque explicativo en ciencias sociales en general y específicamente en educación.
Sin embargo, investigaciones realizadas en el campo de la educación han señalado posibles problemas de estabilidad de las NN debido al alto nivel de ruido en los datos. Este problema ya ha sido abordado en modelos agnósticos como SHAP o LIME, pero aún no ha sido investigado específicamente en NeuralSens.

Objetivo

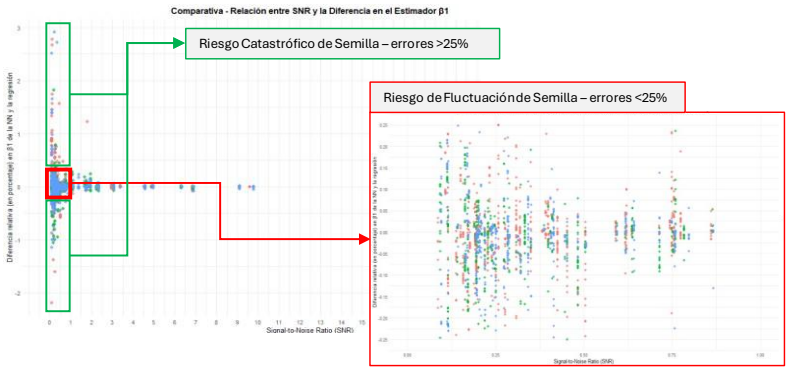
Evaluar el efecto del nivel de ruido en el desempeño de NN y en el subsiguiente análisis post-hoc basado en NeuralSens.

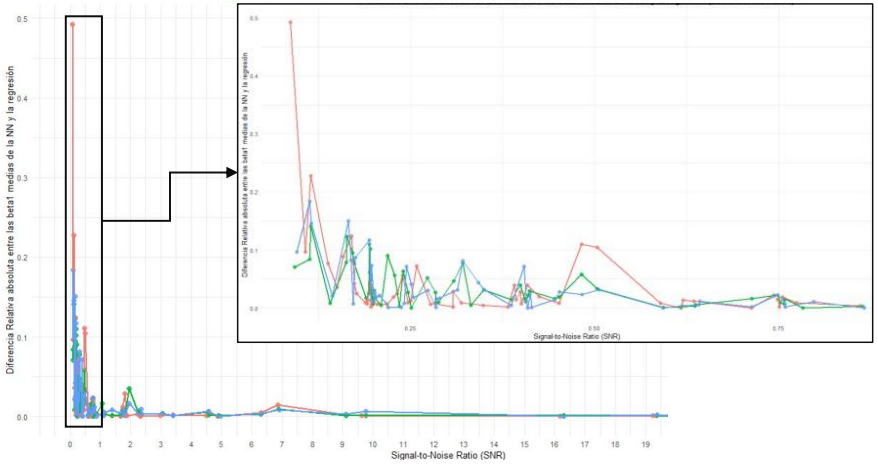
Metodología

Tres escenarios. El primero de ellos, parte de una relación matemática definida entre la variable y y otras dos variables predictoras, x1 y x2, con relación directa e inversa con y respectivamente. El segundo escenario añade una variable neutral (irrelevante) con respecto a y, x3. Finalmente, el tercer escenario añade al anterior una variable adicional cuadrática, x4. El resultado obtenido por la NN para cada escenario se comparará con su homónimo obtenido por el modelo clásico de regresión. Para cada escenario se evalúa el desempeño para distintos niveles de ruido en los datos (evaluado mediante SNR (*signal-to-noise, SNR* = R^2 /[1- R^2]),

Resultados

Muy similares en los tres escenarios: problemas de estabilidad para SNR < 1





Conclusión

Estos problemas pueden mitigarse sustancialmente aplicando una estrategia simple pero eficaz: **entrenar múltiples redes con distintas semillas y promediar sus resultados**. La estrategia planteada reduce el margen de error en la totalidad de valores del SNR, reducción especialmente notable en la zona de alto ruido dónde los riesgos antes presentes, llevando a errores incluso superiores al 25 %, se han visto reducidos enormemente hasta lograr un error no superior al 4%

Contenido

1. Introducción ..... - 1 -
1.1. Motivación del trabajo ..... - 1 -
1.2. Objetivos de la investigación ..... - 1 -
1.3. Metodología ..... - 1 -
1.4. Estructura del trabajo ..... - 2 -
2. Marco Teórico ..... - 3 -
2.1. Modelos predictivos basados en redes neuronales ..... - 3 -
2.2. Funcionamiento de una Red Neuronal Artificial ..... - 4 -
2.3. La semilla en una Red Neuronal ..... - 6 -
2.4. Explicabilidad de redes neuronales ..... - 7 -
2.5. NeuralSens ..... - 8 -
2.6. NeuralSens en educación universitaria ..... - 11 -
3. Material y Métodos ..... - 13 -
3.1. Análisis del primer script, “Generación de Datos” ..... - 13 -
3.2. Análisis del segundo script, “Análisis de Resultados” ..... - 15 -
4. Resultados y Discusión ..... - 17 -
5. Conclusiones ..... - 25 -
6. Anexo ..... - 26 -
7. Declaración respecto al uso de Chat GPT u otras herramientas de IAG ..... - 56 -
8. Referencias ..... - 57 -

1. Introducción

1.1. Motivación del trabajo

Este TFG forma parte de un proyecto de investigación dirigido por el Profesor Jose Luis Arroyo y se engloba en el marco de la Cátedra Santalucía de Analytics for Educación de la Universidad Pontificia Comillas.

1.2. Objetivos de la investigación

La totalidad del proyecto de investigación mencionado anteriormente tiene por objetivo el estudio y cuantificación del efecto del ruido en la estabilidad de los beta estimados por las redes neuronales o *neural networks* (NN) e interpretados por la tecnología NeuralSens. Este problema ya se ha estudiado con modelos similares como SAHP o LIME. Sin embargo, a diferencia de éstos, NeuralSens es un modelo centrado exclusivamente en Redes Neuronales, para el que todavía no se ha llevado a cabo la mencionada investigación.

En este contexto, en el presente trabajo se propone evaluar la robustez y validez del análisis de sensibilidad realizado con NeuralSens en entornos con elevado ruido en los datos, proponiendo estrategias para mitigar los efectos del riesgo de semilla en redes neuronales artificiales. Asimismo, se comparará la sensibilidad obtenida a partir de una única red neuronal frente al cálculo de valores promedio de sensibilidad derivados del entrenamiento de múltiples redes con distintas semillas para lograr los mejores resultados posibles.

1.3. Metodología

Siguiendo la línea de la investigación del Profesor. J. L. Arroyo, la metodología del presente trabajo será la usada para la obtención de los primeros resultados del proyecto de investigación.

De esta forma, para obtener los resultados se han comparado los valores de beta obtenidos mediante un modelo de regresión lineal con los derivados a través de NeuralSens aplicado a una red neuronal, evaluando diferentes niveles de ruido en los datos. Los niveles de ruido se han medido mediante el SNR (*Signal-to-Noise*, SNR = $R^2/(1-R^2)$). La relación tratada sigue la siguiente fórmula:

$$y = \beta 0 + \beta 1 * x + \beta 2 * x2$$

dónde β0 es cero, β1 es positivo y β2 es negativo. Sin embargo, esta es la realidad del primer escenario. A esta base, como se verá posteriormente, se le añadirán una tercera y una cuarta variable para aportar una mejor visión del estudio que se llevará a cabo. Para estas variables, β3 será cero, al igual que β0; y β4 será positiva, pero en esta ocasión se tratará de una variable cuadrática.

En cuanto a los datos base, éstos han sido generados sintéticamente, y se fue incorporando progresivamente un mayor ruido, con 50 iteraciones para cada nivel de ruido.

Para entrenar las redes neuronales, se ajustaron los hiperparámetros mediante una búsqueda en malla, considerando configuraciones de entre 1 y 4 neuronas en la capa intermedia y valores de *decay* entre 10-7 y 10−1. Además, se utilizó validación cruzada de 10 particiones (10-fold cross-validation) para evitar sobreajuste.

Por último, y debido al alto coste computacional, los experimentos se llevaron a cabo en un servidor equipado con un procesador Intel® Xeon® Gold 6330 (con una frecuencia base de 2.0 GHz) y 128 GB de memoria RAM.

1.4. Estructura del trabajo

Este trabajo se organiza en cinco bloques principales.

Comenzando con el presente apartado, la *Introducción*, presenta la motivación principal del estudio: la necesidad de evaluar la estabilidad y fiabilidad de los coeficientes de sensibilidad generados por NeuralSens en redes neuronales artificiales, especialmente en contextos con presencia de ruido, comunes en ciencias sociales y educativas. También se expone los objetivos específicos de la investigación, centrados dar solución a los dos tipos de riesgo identificados y asociados al uso de semillas aleatorias en estos campos de estudio.

En el *Marco Teórico*, se abordarán los principales conceptos necesarios para comprender el estudio. Se revisarán los fundamentos de los modelos predictivos basados en redes neuronales, explicando su estructura, funcionamiento y comportamiento durante el proceso de aprendizaje. Se analizará el papel de la semilla en estos modelos, destacando cómo puede generar variabilidad en los resultados. Además, se presentará la herramienta NeuralSens, explicando sus bases matemáticas, funcionamiento y aplicaciones, con un apartado específico dedicado a su uso en el ámbito universitario como instrumento para el análisis educativo.

La sección de *Material y Métodos* describirá con detalle el diseño experimental del estudio. Se explicarán los scripts utilizados: el primero para la generación sintética de datos basada en una función lineal con diferentes niveles de ruido, y el segundo para el análisis de resultados utilizando redes neuronales entrenadas con distintas configuraciones de semilla. Se justificará el uso de validación cruzada y la elección de hiperparámetros, y se introducirá el enfoque propuesto de promediar sensibilidades obtenidas de múltiples redes para reducir la variabilidad.

En *Resultados Y Discusión*, se presentarán los resultados derivados de los procesos previamente analizados. Se analizarán las desviaciones observadas en los coeficientes de sensibilidad en función del nivel de ruido y del tipo de semilla utilizada. A partir de los resultados, se presentarán soluciones a los problemas identificados y a los que se pretende dar solución

El trabajo finaliza con las *Conclusiones*, se sintetizarán y explicarán las aportaciones del estudio realizado en el apartado que precede, desatacando las soluciones ingeniadas para solventar los problemas objeto de investigación. Se añade además un anexo con los scripts, en base RStudio, utilizados en el proyecto; una declaración sobre el uso responsable de herramientas de inteligencia artificial generativa como ChatGPT, y una sección final de referencias que recoge las fuentes bibliográficas empleadas a lo largo del trabajo.

2. Marco Teórico

2.1. Modelos predictivos basados en redes neuronales

Las redes neuronales han demostrado ser efectivas en una amplia gama de aplicaciones en muy diversos campos y para muy diversos problemas, como la ecología (Olden et al., 2004), la medicina (Zhang et al., 2018), el reconocimiento de imágenes (Bishop , 2006; Tang et al., 2022), el procesamiento del lenguaje natural (Goldberg , 2017), las finanzas (El Bannany et al., 2021), etc., Resulta relevante y de utilidad hacer una breve introducción previa, explicando qué es una red neuronal y su origen, para una mejor compresión posterior, no solo de su funcionamiento, sino de la totalidad del trabajo aquí presentado.

Una Red Neuronal es un modelo de *machine learning* cuya estructura pretende asemejarse a la del cerebro humano, utilizando modelos de computación que pretenden imitar la forma en la que trabajan las neuronas biológicas de nuestro cerebro en los procesos de identificación de fenómenos, toma de decisiones y elaboración de conclusiones o respuestas a problemas complejos. (Bishop, 2006; Goodfellow et al., 2016).

Normalmente, una red neuronal está compuesta por diversas capas de procesamiento, que pueden agruparse en tres clases o niveles según su posición dentro del modelo: una primera capa de entrada, una o más capas ocultas, y una capa de salida (Nielsen, 2015):

- Capa de entrada (*input layer*): Esta capa recibe los datos iniciales que se van a procesar. Cada nodo en la capa de entrada corresponde a una característica o variable en el conjunto de datos.
- Capas ocultas (*hidden layers*): En estas capas tiene lugar la mayor parte del procesamiento. Las neuronas de estas capas toman los datos de la capa de entrada, aplican una función de activación y luego pasan los resultados a la siguiente capa. Las capas ocultas permiten a la red aprender patrones y representaciones complejas a partir de los datos. Cuantas más capas ocultas tiene una red, más "profunda" es.
- Capa de salida (*output layer*): Esta capa proporciona el resultado final del procesamiento de la red. En modelos predictivos, la salida es una predicción numérica.

En cada una de ellas se trabaja de manera paralela y todas se encuentran conectadas a la siguiente capa, por uniones con un peso asociado; que toman como input el output de la capa de la que surgen, y que, a su vez, se convierte en el input que utilizará la siguiente capa para realizar la correspondiente función.

Normalmente, todo este proceso tiene lugar en secciones ocultas de la complicada arquitectura de las redes, lo que supone que adivinar o prever el resultado es prácticamente imposible. A esta oscuridad se le puede denominar profundidad como antes se ha mencionado; y en función de la cantidad de capas que se encuentren ocultas, pueden clasificarse en redes neuronales poco profundas y redes neuronales profundas.

Las redes poco profundas, también llamadas redes neuronales superficiales, requieren, como era de esperar, menos potencia de procesamiento y, además, menos tiempo de trabajo. Cuanto más profunda y compleja es una red neuronal, el procesamiento requerirá mayor potencia de computación y tiempo. Siguiendo este esquema, podemos diferenciar aquellas estructuras más sencillas, esas que carecen de capas ocultas, y cuentan con únicamente dos capas, una de entrada y otra de salida.

Conforme se añaden más capas e interacciones entre los nodos, se añade también complejidad al modelo, obteniendo distintas modalidades de capas neuronales.

Para entender lo siguiente, debe hacerse una breve referencia a la unidad básica de estos sistemas, el *Perceptrón*.

Un perceptrón es el bloque básico y fundamental dentro de una red neuronal, desarrollada por Frank Rosenblatt entre las décadas de los 50 y 60. Es una neurona artificial que recibe varias entradas, las procesa y genera una salida. En una red, los perceptrones se agrupan en las capas ya mencionadas y trabajan juntos para procesar datos y aprender patrones complejos. Son estos los centros dónde se llevan a cabo las operaciones pertinentes. Por tanto, los perceptrones son los elementos que componen las redes neuronales más complejas. Aunque individualmente son bastante simples, al combinarse en grandes cantidades, pueden resolver problemas complejos (Pizarroso et al, 2020; Shalev-Shwartz & Ben-David, 2014).

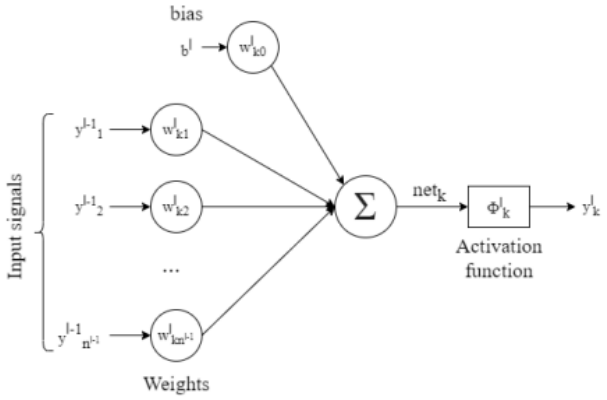
Este perceptrón de Rosenblatt tiene una única capa y puede resolver problemas simples de clasificación lineal, es decir, aquellos en los que los datos pueden separarse mediante una línea recta (o un hiperplano en dimensiones mayores); o problemas de clasificación no lineal, en los que se sustituye la recta por una curva (y el hiperplano por hipersuperficies en dimensiones mayores). Sin embargo, para resolver otros problemas de mayor complejidad, se utiliza el perceptrón multicapa (*multilayer perceptron*, MLP), que organiza varios perceptrones en capas ocultas, permitiendo a la red neuronal aprender patrones más complejos. Así pues, el perceptrón multicapa es una de las formas más comunes de red neuronal.

2.2. Funcionamiento de una Red Neuronal Artificial

El funcionamiento de una red neuronal puede verse como un proceso de aprendizaje automático iterativo de varias fases: la propagación de datos a través de las capas, el cálculo de los errores y la actualización de los pesos para mejorar el rendimiento del modelo.

La primera fase del funcionamiento de una red neuronal es la propagación hacia adelante. En esta fase, los datos de entrada se propagan a través de la red, capa por capa, hasta llegar a la capa de salida. En cada capa oculta, los nodos procesan los datos aplicando una función de activación a los valores ponderados de sus entradas (Figura 1).

Figura 1: Esquema de una neurona (capa k-ésima de un MLP)



Fuente: Pizarroso et al. (2020)

El cálculo en cada neurona es una suma ponderada (productos de las variables por sus pesos) de las entradas más un sesgo (b), que es otro parámetro ajustable del modelo. El resultado de esta suma se pasa a través de la función de activación.

Para un mejor entendimiento de cómo operan estas funciones de activación, analizaremos aquella desarrollada por el científico Frank Rosenblatt, que será el punto de partida de la explicación siguiente.

Rosenblatt propuso un modelo matemático basado en la ponderación de los respectivos inputs, que habían de ser binarios. En este sentido, los valores de salida 0 o 1 (también binarios) son asignados según se llegue o no a un umbral prefijado previamente. El modelo matemático presentado por Rosenblatt es el que sigue:

$$output = \begin{cases} 0 \; if \; \sum x_i \cdot \; w_i \leq threshold \\ 1 \; if \; \sum x_{i\cdot} \cdot w_i \geq threshold \end{cases}$$

A esta asunción básica, se le pueden añadir ciertas modificaciones para obtener una inecuación. Pasando ese umbral preestablecido transformándolo en un *bias* o sesgo asociado al perceptrón, resultando en el siguiente sistema (Nielsen, M. 2015):

$$output = \begin{cases} 0 \; if \; \sum x_i \cdot w_i \; + \; b \leq \; 0 \\ 1 \; if \; \sum x_i \cdot \; w_i + b \geq \; 0 \end{cases}$$

Así era la realidad de los primeros perceptrones. Con el tiempo, este modelo se dividió en distintas partes o funciones: una función o unión sumadora y una función de activación.

La unión sumadora, o función de entrada, es la encargada de obtener la suma de las entradas que recibe el modelo (*z*). Esta adición es una suma ponderada de las variables por sus respectivos pesos a lo que se suma el sesgo (b),. Es este resultado el que pasará a través de la función de activación.

La función del *bias* en una red neuronal es ajustar la salida de la función de activación, permitiendo mayor flexibilidad en el aprendizaje. Tanto los pesos como el *bias* son ajustados automáticamente durante el entrenamiento, y el método más utilizado para esto es el de *Backpropagation*, posteriormente explicado (Menacho C., 2014).

Así las cosas, la función de entrada (o unión sumadora) queda tal que:

$$z \; = \; b \; + \; \sum w_i x_i$$

Dónde:

- w_i son los pesos.
- x_i son los valores de las variables.
- *b* es el sesgo prefijado de antemano por el investigador.

Una función de activación es una operación matemática que determina si una neurona debe activarse o no, en función de la información que recibe. En este sentido, existen numerosos tipos de funciones que han sido desarrollados con el paso del tiempo: de

escalón, ReLu, tangencial, softmax, etcétera. La más simple de todas, aquella presentada por Rosenblatt y referida anteriormente, es la que se conoce como función de tipo escalón y cuya finalidad más común era la clasificación lineal. Sin embargo, el trabajo aquí presentado se centrará en otro tipo, más moderno y utilizado, de función de activación: la sigmoide.

En las funciones sigmoides, se aprecia un cambio con respecto a la de escalón es la diminución de la importancia relativa de los pesos antes explicados. Así las cosas, esta función de activación sigmoidal quedaría de la siguiente manera:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Gráficamente, presenta la siguiente morfología (Figura 2) suavizada con respecto a la presentada por la función de escalón (Figura 3).

Figura 2: Función Sigmoide:

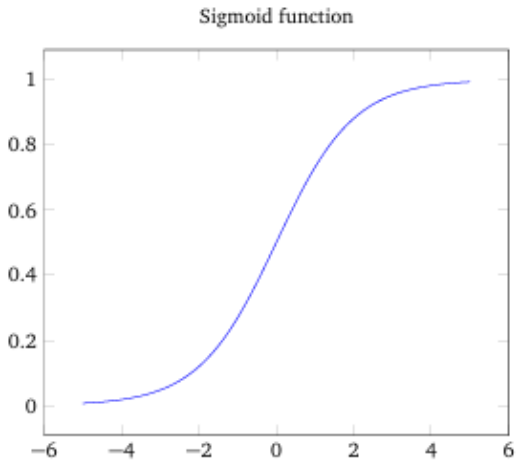
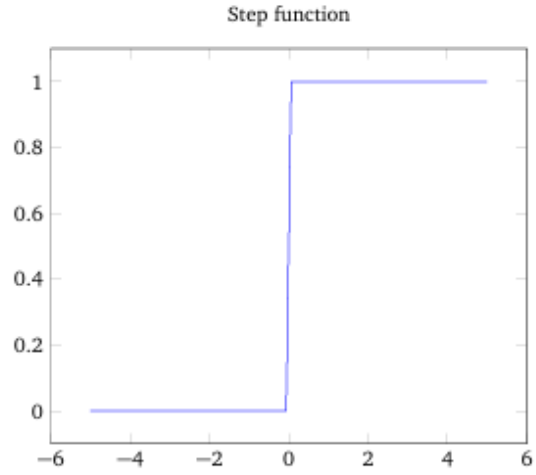


Figura 3: Función de Escalón:



Fuente: Nielsen (2015).

Este proceso se repite para todas las neuronas en cada capa hasta que los datos llegan a la capa de salida. La salida de la red es entonces una predicción basada en las entradas originales.

Una vez generada una salida, el siguiente paso es comparar esta predicción con el resultado real. Para medir la discrepancia entre la salida de la red y el valor real, se utiliza una función de pérdida (*loss function*). La función de pérdida cuantifica el error del modelo, proporcionando una medida del ajuste de este. En consecuencia, cabe afirmar que el objetivo de la red neuronal es minimizar esta función de pérdida, reduciendo el error entre las predicciones y los valores reales y, en consecuencia, mejorando la predicción (Nielsen, 2015; Zhang et al., 2020).

Un ejemplo de función de pérdida es el error cuadrático medio (MSE), que es común en problemas de regresión. Este tipo de funciones pretenden cuantificar la distancia o

discrepancia entre los valores reales y los predichos, por lo que se pretende conseguir los menores valores posibles, siendo 0 una predicción perfecta por parte del modelo.

He aquí la función de pérdida mencionada (Zhang et al., 2020):

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

Dónde:

- *L* es la función de pérdida.
- y_i es el valor real.
- $\hat{y}_i$ es el valor predicho por la red.
- *N* es el número de ejemplo.

Una vez calculado el error, el siguiente paso es ajustar los pesos de la red para mejorar las predicciones. Esto se logra mediante el proceso de *Backpropagation* (o retropropagación). En esta fase, el error se propaga hacia atrás a través de la red, comenzando desde la capa de salida hasta llegar a la capa de entrada.

Durante este proceso de *backpropagation*, se calculan las derivadas parciales de la función de pérdida con respecto a los pesos y sesgos de cada conexión (algoritmo de descenso por gradiente). Este cálculo permite saber cómo cambiar los pesos para reducir el error en la siguiente iteración. Este algoritmo ajusta los pesos en la dirección opuesta al gradiente de la función de pérdida con respecto a los pesos (dirección de máximo decrecimiento de la función), lo que garantiza que los cambios reduzcan el error. La magnitud del ajuste está controlada por una tasa de aprendizaje, de acuerdo con el siguiente esquema:

I. Al inicio del entrenamiento, los pesos de la red neuronal se inicializan, a menudo de manera aleatoria.
II. Se calculan los gradientes de la función de pérdida
III. Se actualizan los pesos utilizando una fórmula similar a la siguiente:

$$w_n = w_o - \eta \cdot \nabla L$$

Dónde:

- w_n son los nuevos pesos.
- w_o son los pesos antiguos.
- η es la tasa de aprendizaje.
- ∇L es el gradiente de la función de pérdida con respecto a los pesos.

Cuando la tasa de aprendizaje es baja, el proceso de entrenamiento es más lento, ya que los cambios en los pesos son pequeños. Sin embargo, puede ser útil para encontrar un mínimo en la función de pérdida más preciso. Por el contrario, cuando la tasa de aprendizaje es alta, aunque el proceso de entrenamiento es más rápido, puede ocurrir que el algoritmo no termine de encontrarlo. Por esto, suele utilizarse un enfoque dinámico de la tasa de aprendizaje, donde se comienza con un valor más alto y se reduce paulatinamente según avanza el proceso entrenamiento (Rumelhart et al., 1986).

El proceso de *backpropagation* y ajuste de pesos se repite muchas veces sobre el conjunto de datos de entrenamiento, lo que permite a la red mejorar continuamente su rendimiento.

Como se ha expuesto, el entrenamiento de una red neuronal es un proceso iterativo. Durante cada iteración, el conjunto de datos se pasa a través de la red, se calculan los errores y se ajustan los pesos. Este ciclo continúa hasta que la red alcanza un nivel aceptable de precisión, es decir, cuando el error es lo suficientemente bajo. Cuantas más iteraciones se ejecuten, mejor se ajustará el modelo a los datos de entrenamiento.

Sin embargo, es importante evitar el sobreajuste (*overfitting*), que ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento y pierde capacidad de generalización para nuevos datos. Para mitigar este problema, se utilizan diversas técnicas de regularización (Nielsen, 2015; Zhang et al., 2020):

- *Dropout*: supone la desactivación aleatoria de neuronas en cada iteración para evitar que el modelo dependa demasiado de ciertas neuronas y así mejore su capacidad de generalización.
- Regularización L2: consiste en agregar a la función un término que penaliza los pesos grandes, obligando al modelo a aprender las representaciones más sencillas.

Estas técnicas de regularización ayudan a mejorar la generalización del modelo y están basadas en elementos de aleatoriedad durante el entrenamiento. Este carácter aleatorio, presente a lo largo de la red, influye no sólo en el ajuste si no también en la capacidad de aprendizaje del modelo neuronal.

2.3. La semilla en una Red Neuronal

Las redes neuronales son sistemas complejos con numerosos parámetros e hiperparámetros, y en su funcionamiento recurren a operaciones aleatorias en distintos momentos del proceso de entrenamiento. Esta aleatoriedad proviene de diversas fuentes, desde la inicialización de pesos de las conexiones entre neuronas, hasta la selección de los datos de entrenamiento. Cabe mencionar que no existe la aleatoriedad total en computación, sino una generación de valores acorde a algoritmos que pretenden asemejarse a esa aleatoriedad.

En este contexto, una semilla (*seed*) es un valor inicial que se utiliza para asegurar la reproducibilidad de los resultados, al controlar la secuencia de números pseudoaleatorios en un modelo de redes neuronales. Esto es particularmente relevante para tareas como la inicialización de los pesos, la organización de los datos de entrenamiento y la implementación de técnicas de regularización, que suelen depender de elementos aleatorios.

Sin una semilla específica, los resultados podrían variar entre ellos. De hecho, cada vez que se entrena una red neuronal sin una semilla específica, el modelo se comportará de manera ligeramente diferente debido a la variabilidad en los valores iniciales de los pesos y en las operaciones aleatorias del entrenamiento. Esta variabilidad afecta a la reproducibilidad del modelo, que es crítica en investigación académica. En consecuencia, resulta imprescindible documentar las semillas usadas para garantizar la transparencia y la reproducibilidad de los resultados.

En este contexto, el nivel de ruido puede afectar considerablemente la sensibilidad a la semilla de los modelos. El término *nivel de ruido* se refiere a la cantidad de variabilidad no explicada, ambigüedad o factores impredecibles que afectan los resultados y dificultan la identificación de patrones claros. En el caso de las ciencias sociales y en educación en particular, el ruido es más complejo y difícil de aislar debido a la existencia de emociones humanas, contextos culturales, motivaciones individuales, etc. En estos ámbitos del saber, la conducta humana es menos predecible, y factores externos influyen de maneras no siempre medibles y, por tanto, el nivel de ruido es mayor que en otras ciencias.

Así, al trabajar con datos con elevado ruido, puede existir una elevada dependencia a la semilla. Es decir, que los resultados pueden variar considerablemente según la semilla empleada, lo que llamaremos riesgo de semilla. La situación de riesgo de semilla resulta especialmente grave cuando se empelan técnicas de análisis post-hoc como NeuralSens, debido a que se pretende dar un uso explicativo y no predictivo al modelo. En consecuencia, este será, precisamente, el objetivo de este trabajo: evaluar el impacto de la semilla en los resultados obtenidos por una red neuronal con análisis post-hoc basado en NeuralSens.

2.4. Explicabilidad de redes neuronales

Cuanto más complejo y potente es un modelo de *Machine Learning*, mayor es la complejidad de los problemas y tareas que podrá resolver. Sin embargo, frecuentemente esta ganancia se consigue mediante la pérdida de interpretabilidad.

Este problema surge, como quisieron transmitir Samek y Müller (2019), cuando los modelos de *machine learning* se convierten en esenciales en las ciencias e industrias. El problema ha ido a más con el incremento del número de datos, del tamaño de las bases que los contienen y de la precisión esperada de los resultados. Si bien estos modelos están en la vanguardia de esta búsqueda de precisión, su estructura no lineal ha conseguido que se los catalogue como modelos de caja negra (*black box*) por su falta de transparencia en el procesamiento de esos datos.

Las redes neuronales son un ejemplo de estas cajas negras. Y aunque estos modelos son muy habituales, su utilidad ha sido criticada y se ha visto limitada ya que adolecen de una falta de interpretabilidad importante, pues su complejidad dificulta entender e interpretar cómo la red neuronal utiliza las variables de entrada para generar sus predicciones.

Antes de continuar, parece conveniente hacer una distinción de términos frecuentemente confundidos. En primer lugar, la interpretabilidad de un modelo es un concepto de alto nivel, dado que se refiere a la habilidad de entender y explicar cómo ese modelo genera sus predicciones, es decir, cómo informa sobre la importancia de las variables del modelo, sus interacciones, las reglas que producen esas predicciones, etc. En segundo lugar, la explicabilidad es, sin embargo, un concepto de más bajo nivel dado que pretende explicar determinadas predicciones en concreto.

Al principio del trabajo se hace referencia a las distintas capas que conforman la estructura de una red neuronal, que según la tipología de estas pueden considerarse redes neuronales de un tipo u otro (redes neuronales superficiales, poco profundas, profundas, etcétera...). En este caso, la carencia de ambas características (interpretabilidad y explicabilidad) se da en las capas ocultas del algoritmo, pues estas pueden considerarse como unidades operacionales o de computación no observables; lo que se traduce en una complicación en su interpretabilidad. Estos modelos no permiten saber si una predicción se debe a

determinadas características de los datos de entrenamiento o a una característica que está correlacionada de manera espontánea con el objetivo, pero no de manera causal; como tampoco permiten conocer posibles sesgos del modelo. Además, la carencia de interpretabilidad y explicabilidad de estos modelos atañe a aspectos legales. De acuerdo con la regulación europea de Protección de Datos (EU General Data Protection Regulation) los modelos o las personas que los usan deben aportar información relevante sobre la lógica de estos, para cubrir el derecho a la explicación amparado por dicha regulación, pero esto es muy difícil en el caso de las redes neuronales. Por último, esta falta de interpretabilidad y explicabilidad de los modelos de redes neuronales dificulta la posibilidad de extraer conocimiento de estos y destruye la posibilidad de aprender de ellos, de manera que, a través de ellos, no es posible en la mayoría de las ocasiones, mejorar el conocimiento que se tiene de ciertos problemas.

Hay diversas maneras de acercarse a una interpretación de su funcionamiento, englobadas todas en lo conocido como XAI (*Explainable artificial intelligence*) (Mandler y Weigand, 2024). Existen diversos modelos y tecnologías que pretenden afrontar el problema que supone la falta de interpretabilidad y explicabilidad en estos modelos de caja negra, siendo los más conocidos SHAP o LIME. Este trabajo se centra en la aproximación a través de NeuralSens, tecnología que se explica a continuación, y que se diferencia de SHAP y LIME en ser específico para redes neuronales (a diferencia de SHAP o LIME, que son modelos agnósticos, es decir, aplicables a distintos modelos de machine learning) y tener una aproximación más cuantitativa.

2.5. NeuralSens

En el contexto de este TFG, hablaremos de análisis de sensibilidad como un análisis post-hoc (se efectúa sobre el resultado o predicción de un modelo) que consiste en estudiar el impacto que las diferentes variables independientes o explicativas de un modelo tienen sobre el resultado de la variable dependiente. Dicho de otra manera, un análisis de sensibilidad sirve para determinar cómo el cambio de valor de una variable independiente afecta a la variable dependiente del modelo para así poder analizar la importancia de la variable de entrada en la salida. En consecuencia, uno de los métodos más extendidos para realizar esos análisis de sensibilidad es el método de las derivadas parciales, precisamente porque el concepto de derivada parcial responde a cuál es la variación de una variable dependiente ante variaciones (pequeñas) de las correspondientes variables independientes.

El uso del método de derivadas parciales en redes neuronales tiene algunos inconvenientes: como señalan Pizarroso et al. (2022), las operaciones necesarias para calcular las derivadas parciales son más lentas en comparación con otros métodos, y el tiempo de cálculo aumenta a medida que crece el tamaño de la red neuronal o el tamaño de la base de datos utilizada Sin embargo, las ventajas de este método en comparación con otros alternativos hacen del análisis de sensibilidad basado en derivadas parciales una técnica muy útil para interpretar los modelos de redes neuronales.

El funcionamiento de esta técnica es el siguiente (Muñoz-San Roque et al., 2025): Una vez ajustado el modelo de redes neuronales con un conjunto de datos, se busca el efecto de cada variable de entrada en la salida, midiendo la sensibilidad a través de la derivada parcial de la salida con respecto a cada una de las variables de entrada. La diferencia con respecto a los modelos de regresión, donde la derivada parcial de cada variable es constante (un único beta), en las redes neuronales se obtiene un valor de la derivada parcial (también denominada pendiente por su interpretación geométrica) en cada dato. Esto implica que una variable tendrá una distribución de pendientes. En ausencia de

efectos no lineales, la distribución será muy estrecha y su valor medio coincidirá exactamente con el beta obtenido en un modelo de regresión. Si una sensibilidad es muy alta, es de esperar que genere derivadas con un valor importante, al menos en algunos puntos. Y al revés, si una variable no es relevante o no es utilizada en el modelo, es de esperar que la derivada parcial correspondiente sea prácticamente nula.

Con esta idea, Pizarroso et al. (2022) desarrollaron una librería R (NeuralSens) en la que se calculan y representan las distribuciones de las derivadas parciales en redes neuronales. En este momento vamos a centrarnos en la estructura del paquete NeuralSens, para poder entender cuáles son las funcionalidades que ofrece dicho paquete y cómo interpretar los resultados que ofrece. Al respecto de las funcionalidades, estas son las más destacadas:

1. SensAnalysisMLP():

Esta función realiza el análisis de sensibilidad utilizando derivadas parciales de las salidas con respecto a las entradas del modelo MLP (multilayer perceptrón). Esta función devuelve un objeto 'SensMLP' con los resultados del análisis de sensibilidad. Este objeto, tiene los siguientes componentes:

- *sens*: 'list' de 'data.frames', uno por cada neurona en la capa de salida, con las medidas de sensibilidad. Cada fila del data.frame contiene las medidas de sensibilidad con respecto a una entrada específica.
- *raw_sens*: 'list' of 'matrices', una por cada neurona en la capa de salida, con las sensibilidades calculadas. Cada columna de cada 'matrix' contiene las sensibilidades de la salida con respecto a una entrada específica y cada fila contiene las sensibilidades respecto a todas las entradas correspondientes a la misma fila del componente trData.
- *mlp_struct*: 'numeric' 'vector' que indica el número de neuronas de cada capa en el modelo MLP.
- *trData*: 'data.frame' que contiene el conjunto de datos usado para calcular las sensibilidades
- *coefnames*: 'character' 'vector' con los nombres de las variables de entrar en el modelo MLP.

2. SensitivityPlots():

Esta función representa gráficamente las medidas de sensibilidad de un objeto 'SensMLP'.

3. SensFeaturePlot():

Esta función representa gráficamente la relación entre las sensibilidades de un objeto 'SensMLP' y el valor de las variables de entrada

4. SensTimePlot():

Esta función representa gráficamente la evolución con respecto al tiempo de las sensibilidades del objeto 'SensMLP'.

La aplicación de SensAnalysisMLP() produce una salida de las sensibilidades en forma de data.frame (matriz) cuyas filas dan cuenta de las entradas del modelo (variables X) y cuyas columnas proporcionan información sobre la relación entre las variables de entrada y las variables de salida (variables Y): La columna "mean" muestra el efecto promedio de la variable de entrada sobre la salida. La columna "std" muestra la variabilidad del efecto de la variable de entrada sobre la salida (desviación típica).

Dichas columnas, se interpretan de la siguiente manera:

1. Si tanto "mean" como "std" están cerca de cero, la salida no está relacionada con la entrada, ya que, para todos los datos de entrenamiento, la sensibilidad de la

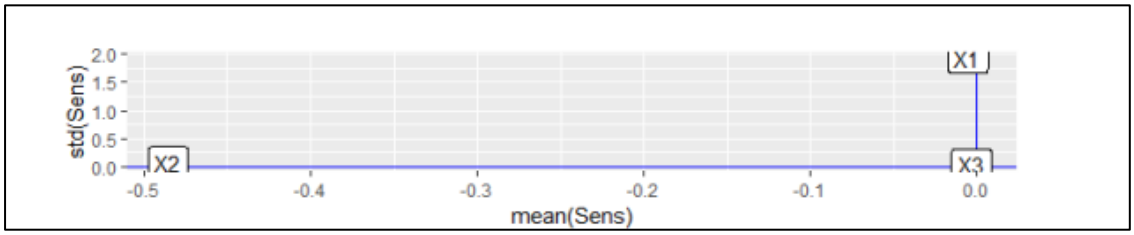
salida con respecto a esa entrada es aproximadamente cero. La variable es por tanto irrelevante.

2. Si "mean" es diferente de cero y "std" está cerca de cero, la salida tiene una relación lineal con la entrada, ya que para todos los datos de entrenamiento la sensibilidad de la salida con respecto a esa entrada es aproximadamente constante.

3. Si "std" es diferente de cero, independientemente del valor del promedio (mean), la salida tiene una relación no lineal con la entrada, porque la relación entre la salida y la entrada varía dependiendo del valor de la entrada.

Adicionalmente, la aplicación de SensitivityPlots() del paquete permite visualizar las sensibilidades, cuya interpretación es similar a la anterior (ejemplo tomado de Pizarroso et al., 2020): En la figura 4 , aparece la desviación típica "std" de la sensibilidad. La variable *x1* tiene una "std" diferente de cero (relación no lineal con *y*) y una media nula. La variable *x2* tiene una "std" muy próxima a cero y una "mean" próxima a -0,5 (relación lineal) y la variable *x3* tiene una "std" muy próxima a cero y una "mean" muy próxima a 0 (no relación).

Figura 4. Salida de NeuralSens: Std (Sens) y mean (Sens)



Fuente: Pizarroso et al. (2022)

En la figura 5, se muestra la media de las sensibilidades al cuadrado. Se trata de una medida de la importancia de cada variable, que se calcula elevando al cuadrado las sensibilidades (para evitar compensaciones entre valores negativos y positivos) y, posteriormente, calculando la media. En el caso de *x3* su valor es 0 (variable irrelevante) la de *x2* es pequeña (variable relevante, pero de importancia moderada), y la de *x1* es la más grande (variable relevante e importancia elevada).

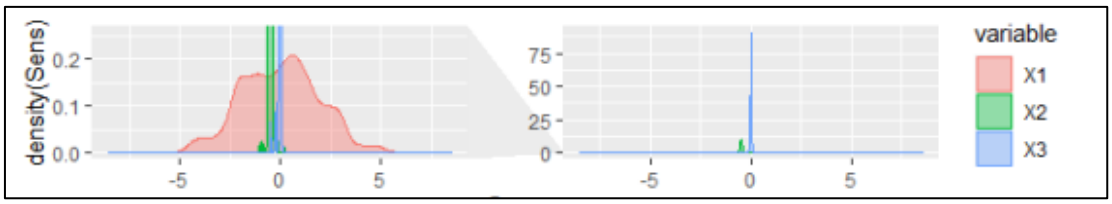
Figura 5. Salida de NeuralSens: Mean (Sens^2)

Fuente: Pizarroso et al. (2022)

Finalmente, la figura 6, muestra la función de densidad de la sensibilidad: la sensibilidad de la variable *x1* tiene una media próxima a cero, pero gran variabilidad (distribución "ancha"), lo que indica una relación no lineal con la variable dependiente Y. La variable *x2*

tiene una media distinta de cero, pero una distribución muy estrecha, con valores altamente concentrados en torno a la media, indicando casi nula variabilidad (relación lineal). Finalmente, la variable *x3* tiene una muy próxima a cero y una distribución con valores altamente concentrados en torno a la media, indicando casi nula variabilidad (no relación).

Figura 6. Salida de NeuralSens: Density (Sens)



Fuente: Pizarroso et al. (2022)

El paquete NeuralSens también dispone de gráficos especialmente diseñados para series temporales. Dado que nuestra aplicación no va a realizarse en ese tipo de datos, omitimos estas funcionalidades del paquete por simplicidad.

2.6. NeuralSens en educación universitaria

A pesar de lo reciente del desarrollo de NeuralSens, su utilización como método de interpretabilidad de redes neuronales está ya relativamente extendido. A modo de ejemplo, el número de citas en *Google Scholar* del trabajo de Pizarroso et al (2022) a fecha de 26 de febrero de 2025 era de 101. De esas citas, la mayor parte de ellas corresponden a modelos de redes neuronales aplicados a la ingeniería, dado que ese fue el campo de aplicación para el que se concibió inicialmente la técnica. Las aplicaciones en educación, sin embargo, son menos numerosas, pero también en este campo la técnica ha sido aplicada con éxito. En particular, Arroyo-Barrigüete et al. (2023a) emplearon NeuralSens para analizar las diferencias de género en el desempeño en la asignatura de Matemáticas; Arroyo Barrigüete et al. (2023b) analizaron el efecto del itinerario de bachillerato en el desempeño universitario en Matemáticas; y Ortiz-Lozano et al. (2024) utilizaron NeuralSens para interpretar los resultados de sus modelos para analizar el abandono universitario en estudiantes universitarios.

En estos artículos, la finalidad principal de NeuralSens fue testar la especificación funcional de los modelos econométricos usados (regresión o logit). Efectivamente, tanto los modelos de regresión lineal como logística requieren que el investigador proponga a priori una especificación funcional del modelo. Para confirmar su validez, en los trabajos mencionados también se ajustaron modelos de redes neuronales empleando las mismas variables. La principal ventaja de los modelos de redes neuronales respecto a los modelos de regresión es que no requieren una especificación funcional a priori, y cualquier efecto no lineal presente en los datos será identificado automáticamente por la red, sin necesidad de que el investigador lo formule explícitamente. Aunque, históricamente, los modelos de redes neuronales ofrecían buenas predicciones, no eran útiles para hacer trabajos explicativos, dado que se desconocía el efecto de cada variable porque no resultaba posible interpretarlos. Sin embargo, el algoritmo NeuralSens resuelve el problema, dado que permite la interpretación de un modelo de redes neuronales de una manera sencilla, como hemos visto en el apartado correspondiente.

De esta manera, en los trabajos antes mencionados, si los resultados de la red neuronal coincidían con los del modelo de regresión o logit, se confirmaba que la especificación funcional de los modelos era correcta. Por el contrario, resultados diferentes indicarían que el modelo estaría mal especificado y sería necesario re-especificarlo, en general, para incluir algún efecto no lineal que no hubiera estado incluido inicialmente (Arroyo-Barrigüete et al., 2023a, 2023b).

3. Material y Métodos

A continuación, se presentará el desarrollo de un código en R, base de la tecnología NeuralSens, que tiene como objetivo analizar la influencia del ruido en la estimación de coeficientes dentro de un modelo de regresión lineal. Para ello, se presentarán tres escenarios. El primero de ellos, parte de una relación matemática definida entre la variable y y otras dos variables predictoras, *x1* y *x2*, con relación directa e inversa con *y* respectivamente. El segundo escenario añade una variable neutral (irrelevante) con respecto a *y*, *x3*. Finalmente, el tercer escenario añade al anterior una variable adicional cuadrática, *x4*. Además, el resultado obtenido por la NN para cada escenario se comparará con su homónimo obtenido por el modelo clásico de regresión, lo que nos permitirá hacer un exhaustivo análisis posterior de los resultados.

De esta forma, el primer escenario planteado seguirá la fórmula:

$$y = a * x1 - b * x2$$

El siguiente escenario, añade una tercera variable neutra, x3, al anterior:

$$y = a * x1 - b * x2 + c * x3, siendo\ c = 0$$

Para el último de los casos, se añade una variable cuadrática, x4, al tercer escenario:

$$y = a * x1 - b * x2 + c * x3 + x4\text{^}2, siendo\ c = 0$$

Cabe mencionar que el código se ajusta a según qué escenario añadiendo un "apellido" a cada uno de los campos de relevancia en el estudio que indica el escenario de que se trata, véase; para el primer escenario, los datos obtenidos se guardarán en el data.frame *Resultados_lineal*, mientras que los del segundo y tercero se guardarán en los data.frame *Resultados_lineal2, Resultados_lineal3* respectivamente.

De esta forma, primero se presentará el análisis del script "Generación de Datos". Para finalizar con un análisis del segundo script, "Análisis de Resultados".

3.1. Análisis del primer script, "Generación de Datos"

1. Carga de paquetes, establecimiento del directorio de trabajo
2. Se establece la configuración para la validación cruzada *"k-folds con k = 10"*:

La validación cruzada K-*fold* es una técnica robusta para evaluar la capacidad de generalización de modelos predictivos. Consiste en dividir el conjunto de datos en k particiones del mismo tamaño, utilizando cada una de ellas como conjunto de validación mientras el resto se emplea para entrenar el modelo. Este proceso se repite k veces, asegurando que cada observación sea utilizada una vez para validar y k-1 veces para entrenar.

El valor escogido de k influye en el equilibrio entre el sesgo y la varianza: los valores bajos ofrecen estimaciones más sesgadas a cambio de un menor costo computacional, mientras que valores más altos proporcionan estimaciones más estables.

3. Configuración de los parámetros generales de los datos sintéticos. Configuración de la semilla:

Primero se fija una semilla aleatoria para asegurar la reproducibilidad de los resultados en caso de necesidad, como se alegaba con anterioridad en este trabajo. Se indican, además,

el número de repeticiones a realizarse en cada nivel de ruido (50), así como la cuantía de niveles de ruido que se aplicará progresivamente en el modelo. Finalmente, se indica cuanto aumenta el ruido en cada paso y cuál es la proporción de los datos reservada al entrenamiento del modelo. En el presente escenario, la totalidad de los datos va destinada al entrenamiento del modelo, como es habitual en modelos explicativos.

4. Determinación de las variables del análisis y creación de un data.frame para almacenar resultados.

Se crea un data.frame con una longitud igual al número de variables definidas. En el caso del primer escenario, este data.frame contendrá los campos relativos a las dos variables independientes (*x1,* x2). Para los siguientes escenarios se añade una variable *correlación_x3y* (o *correlacion_x4y* para el tercer caso de estudio). Una variable *mean_beta*, *std_beta* y *reg_beta* con la correspondiente nomenclatura según la variable independiente a al que haga referencia.

Adicionalmente, habrá otras partes del código que se actualizarán según en qué escenario nos encontremos. En este caso, tanto el nombre del data.frame como el de las variables *neuronas_capa_oculta* y *valor_decay* irán actualizándose con un 2 o un 3 al final, diferenciándose para cada escenario.

5. Generación de datos, definición de la variable dependiente y ajuste del modelo de regresión lineal:

Se define la variable dependiente *y* y cómo ésta se relaciona con las variables independientes *x1* y *x2*. Esta parte cierra con el ajuste del modelo de regresión lineal para predecir *y* en función de las variables independientes. Para los escenarios restantes, esa regla de dependencia se modifica de la siguiente manera:

    a. Para el segundo escenario: se incluye la variable neutra *x3*
    b. Para el tercer escenario: se incluye la variable cuadrática *x4* al segundo escenario.

6. Bucle aditivo de ruido, cálculo de las correlaciones, iteraciones para cada nivel de ruido en bucle interno y creación del conjunto de datos:

En esta parte del modelo se estudia cómo se comporta la función conforme los niveles de ruido aumentan para la variable dependiente. Esto supone ver si la relación entre las variables independientes, *x1* y *x2* en este caso (*x3* en caso del escenario dos y *x4* para el tercero) sufre por ese incremento de ruido.

7. Preparación del conjunto de entrenamiento, ajuste de la red neuronal y selección del mejor modelo:

Se procede a entrenar la red mediante la práctica ya explicada de *cross-validation*, acompañada de la optimización de los hiperparámetros. Además, permite observar cómo varía la estructura óptima de la red neuronal a medida que ese ruido en *y* se incrementa.

8. Ajuste de la red neuronal a los hiperparámetros óptimos identificados en la etapa anterior y análisis de la media y deviación estándar de las betas de las correspondientes:

Se procede a ajustar la red a los hiperparámetros antes identificados, de manera que el análisis llevado a cabo se haga con el mejor modelo posible. El uso de la función

SensAnalysisMLP para estimar la sensibilidad de y respecto a cada variable independiente. El análisis desarrollado cuantifica la influencia de cada una sobre la salida del modelo y visualiza los resultados posteriormente, extrayendo la media y variabilidad de la sensibilidad de cada variable.

Este último punto permite analizar cuál de las variables tiene una mayor influencia y cómo ésta cambia con respecto se incrementa en nivel de ruido o el número de variables independientes.

9. Comparación con un modelo de regresión. Cálculo de ruido y guardado de resultados:

Esta parte de la programación ajustará un modelo OLS (mínimos cuadrados ordinarios, *ordinary least squares* en inglés) con todas las variables disponibles, extrayendo los coeficientes de cada variable predictora. Éstos se almacenarán en una variable individual para su posterior análisis permitiendo comparar los distintos coeficientes con los de la red neuronal. Además, se mide el SNR (*Signal to Noise Ratio*).

10. Guardado de resultados y finalización del proceso

3.2. Análisis del segundo script, "Análisis de Resultados"

El código aquí descrito toma como sustrato para la representación gráfica, los resultados obtenidos con el código anterior. Cada escenario, como se explicará a continuación, cuenta con tres gráficas que detallan relaciones entre distintos campos de interés para realizar un análisis posterior de los mismos y poder obtener los efectos que tiene el añadir una u otra variable a la precisión del modelo.

1. Carga de paquetes, establecimiento del directorio de trabajo. En esta ocasión, se trabajará con paquetes adicionales centrados en la representación gráfica.
2. Carga de datos y preparación de las representaciones gráficas.

Como se ha mencionado con anterioridad, se cargan los datos, diferenciando por escenarios, para poder realizar una primera representación gráfica individual. Se crean métricas, concretamente se calculan la diferencia entre los coeficientes estimados por la red neuronal y los obtenidos por el método de regresión lineal. Se crean tanto diferencias relativas como absolutas, lo que da la oportunidad de evaluar los efectos del ruido y de la presencia de nuevas variables a la estimación de los coeficientes.

3. Gráficas de relación entre SNR y la diferencia relativa en β1.

Para cada escenario, se representan los valores de SNR frente a la diferencia relativa en la estimación de β1 entre la red neuronal y el modelo OLS, permitiendo visualizar la estabilidad de β1 en función del nivel de ruido con el que se trabaja.

Además, se diferencian dos tipologías de tablas, ya que la segunda hace foco en la zona de alto ruido, lo que supone una ayuda en cuanto a identificar pautas de comportamiento del modelo en condiciones extremas.

En este segundo caso, se crea un subconjunto con los casos donde el nivel de señal es inferior al del ruido. Entonces, se calcula la media de las diferencias en β1 dentro de este subconjunto con el objetivo es cuantificar en qué medida se degrada la estimación del coeficiente en esas condiciones dónde la fiabilidad del modelo se pone más en duda.

4. Preparación de las gráficas comparativas.

Pese a explicar aquí el código generativo, las gráficas y su análisis de explicaran en el siguiente apartado.

En este apartado se unifican los resultados de los tres escenarios en un único data.frame. De igual forma, se genera otro data.frame con los subgrupos enfocados en la zona de alto ruido.

Para cada uno de los data.frame, se visualizan los tres escenarios simultáneamente, siguiendo un código de colores, de manera que la comparación, y su posterior análisis sea más sencillo y acertado.

Se representan la relación entre el SNR y la diferencia relativa en β1. Al representar gráficamente los tres escenarios en un mismo panel, diferenciados por color, se visualiza con claridad cómo la estimación de beta sufre los efectos del incremento de ruido o del número de variables, y el tipo de estas últimas, como la neutral o al cuadrática.

4. Resultados y Discusión

Con el objetivo de evaluar la capacidad de generalización del modelo neuronal en distintos contextos de complejidad y niveles de ruido, se ha llevado a cabo un análisis comparativo basado en tres escenarios experimentales. Esta configuración permite estudiar de forma controlada cómo la inclusión de variables irrelevantes y cuadráticas influye en la estabilidad y precisión de la estimación del coeficiente β1.

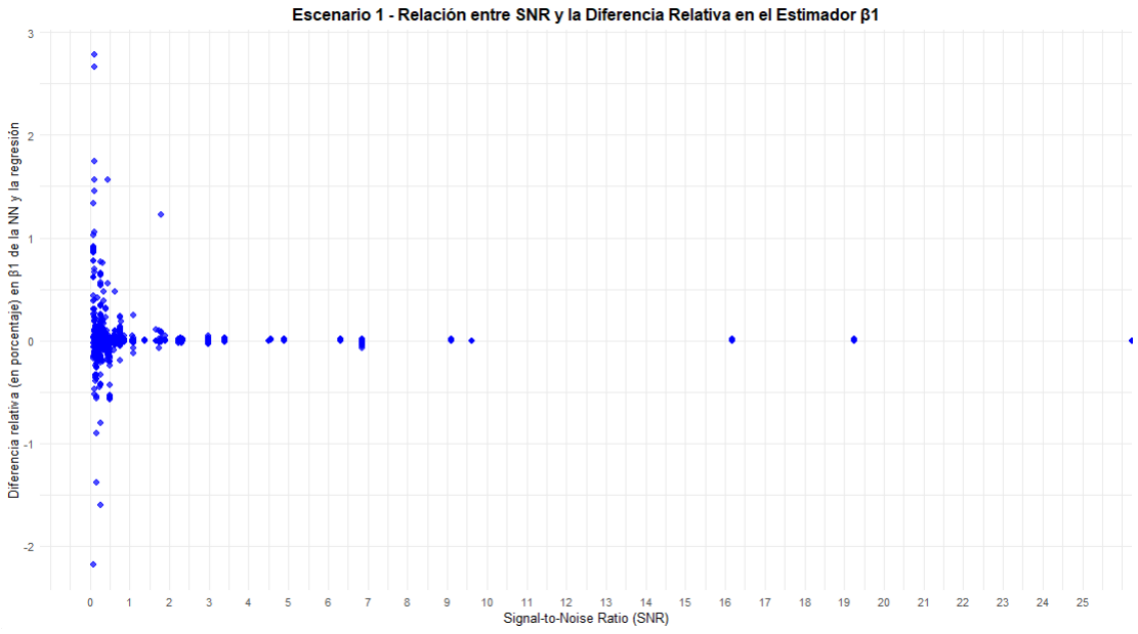
El análisis se articula en torno a la desviación entre la estimación obtenida por la red neuronal y la referencia proporcionada por un modelo de regresión lineal. Se pretende enfocar la atención en la evolución de dicha estimación en función del SNR.

Para ello, se han generado distintas visualizaciones que permiten observar la evolución del modelo en cada uno de los escenarios:

Las primeras gráficas (figuras 7-12), específicas para cada escenario, representan la relación entre el SNR y la diferencia relativa en la estimación de β1. Estas permiten estudiar el impacto del ruido de manera aislada, sin la interferencia de otras configuraciones, ofreciendo así una lectura individual del grado de precisión del modelo en función de la incertidumbre. Se incluyen tanto versiones generales como focalizadas en entornos de alto ruido (SNR < 1), donde la calidad del dato es más limitada.

Las gráficas comparativas (figuras 13-16), por su parte, enfrentan simultáneamente los tres escenarios con el fin de contrastar su comportamiento relativo. La primera de ellas permite visualizar de forma directa cómo varía la precisión de la estimación de β1 a lo largo del espectro del SNR, y la segunda ofrece un análisis centrado en la zona crítica de bajo SNR, donde los modelos suelen mostrar una mayor fragilidad.

Figura 7. Escenario 1 - Relación entre SNR y la Diferencia Relativa en el Estimador β1

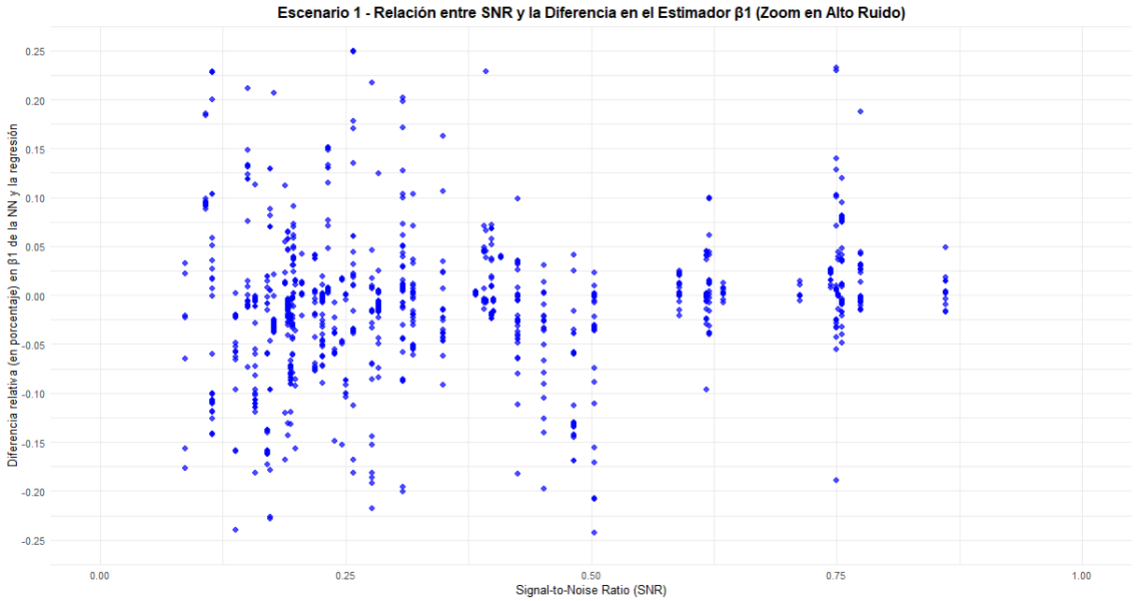


Fuente: Elaboración propia

La figura 7 analiza la diferencia relativa entre la estimación del parámetro β1 obtenida por una red neuronal y la generada mediante regresión lineal, en función del nivel de ruido expresado mediante el SNR. El primer escenario es el más simple, ya que únicamente considera dos variables. Cuando el SNR es elevado, es decir, cuando la señal domina

claramente sobre el ruido, la diferencia entre ambos enfoques es prácticamente nula. A partir de SNR superiores a 3-4, la estimación de β1 por parte de la red neuronal converge de forma precisa con la regresión. En cambio, en entornos con SNR bajo, especialmente por debajo de uno, se observa una mayor dispersión en la diferencia relativa, reflejando una pérdida de precisión por parte del modelo de red neuronal. Esta evolución gradual sugiere que el modelo neuronal es sensible al ruido, pero ofrece un rendimiento fiable en la medida en que la calidad de los datos mejora.

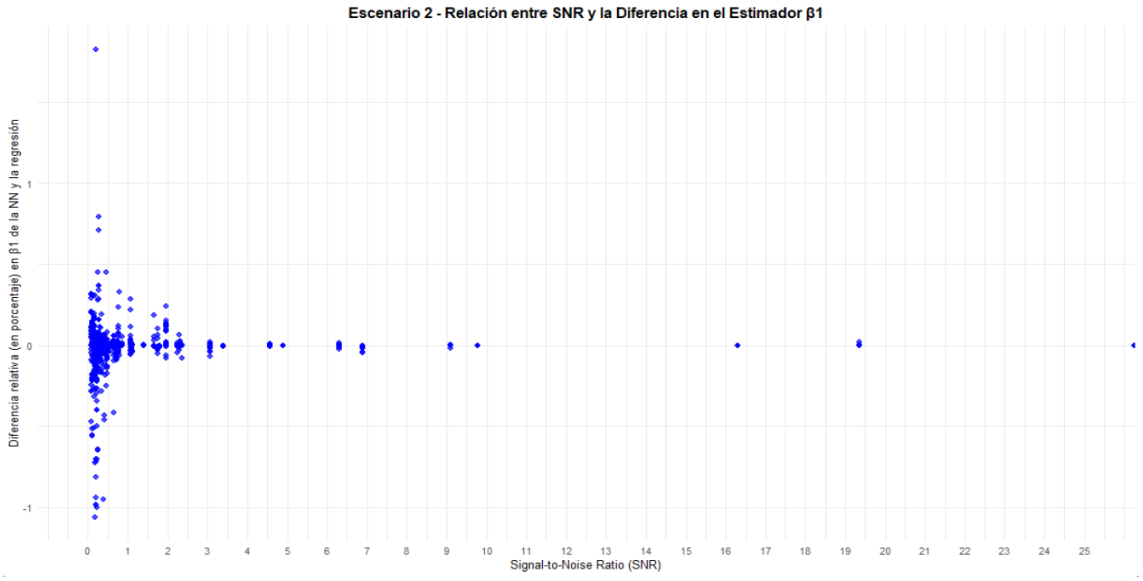
Figura 8. Escenario 1 - Relación entre SNR y la Diferencia Relativa en el Estimador β1 (Zoom en Alto Ruido)



Fuente: Elaboración propia

La figura 8 se centra en el intervalo más crítico del modelo, donde el SNR es menor a 1 y, por tanto, el ruido domina sobre la señal. En este contexto, se observa una mayor dispersión en la diferencia relativa entre la estimación de β1 de la red neuronal y la regresión lineal, lo que indica una pérdida de precisión bajo condiciones adversas. Aun así, la mayoría de los puntos se mantienen dentro de un rango contenido, principalmente entre -25% y 25%, aunque con algunas desviaciones extremas.

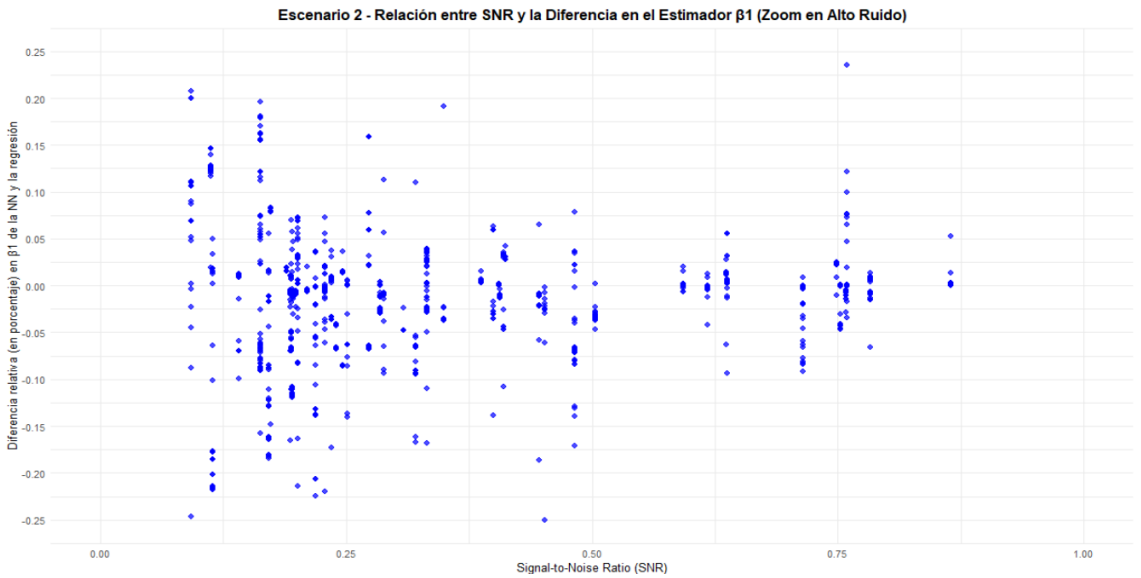
Figura 9. Escenario 2 - Relación entre SNR y la Diferencia Relativa en el Estimador β1



Fuente: Elaboración propia

La figura 9 representa el comportamiento del modelo en el Escenario 2, donde se ha añadido una variable neutra sin relación con la variable dependiente. El patrón es muy similar al del escenario 1 y a medida que el SNR aumenta, el modelo recupera estabilidad y la diferencia relativa tiende a cero.

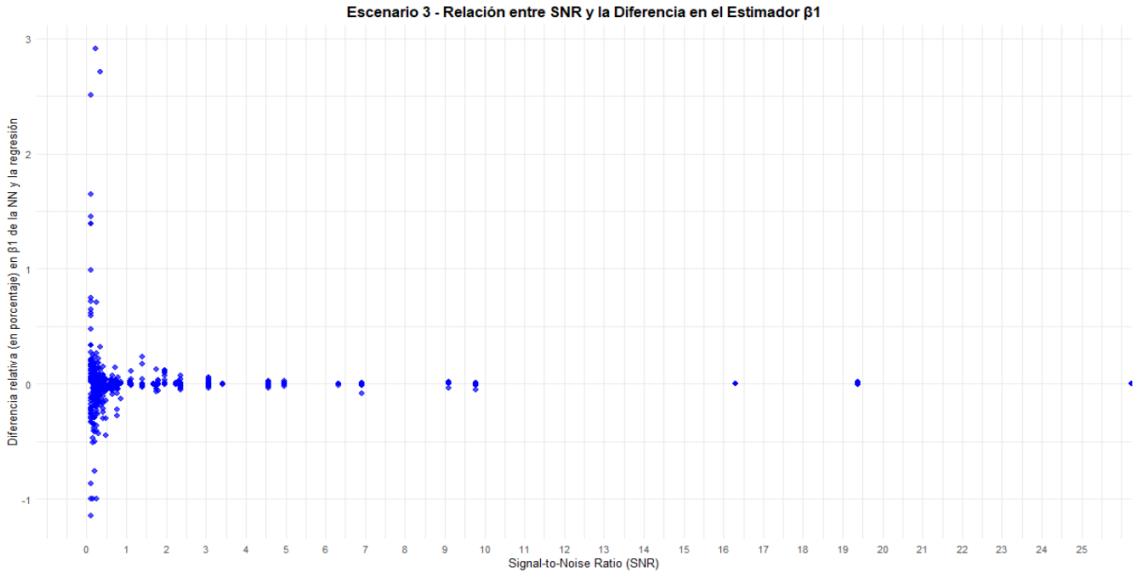
Figura 10. Escenario 2 - Relación entre SNR y la Diferencia en el Estimador β1 (Zoom en Alto Ruido)



Fuente: Elaboración propia

La dispersión de las diferencias relativas en zonas de bajo SNR en el escenario 2 (Figura 10) es similar a la del escenario 1.

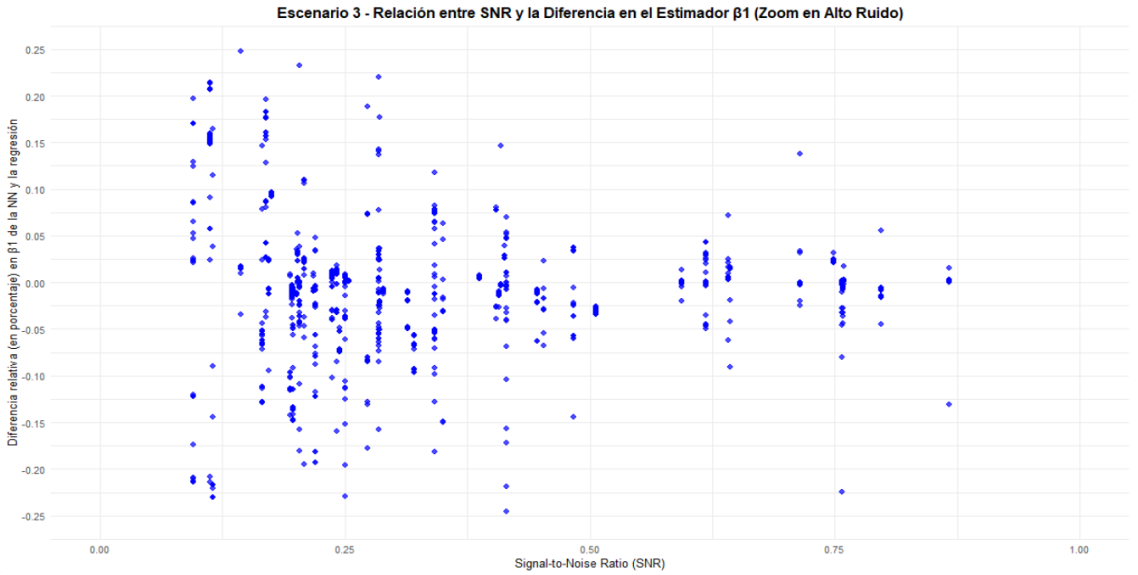
Figura 11. Escenario 3 - Relación entre SNR y la Diferencia en el Estimador β1



Fuente: Elaboración propia

En el escenario 3 (figura 11) de nuevo puede observarse una estructura similar a la de los anteriores escenarios, pero la distinción más llamativa viene en los niveles más bajos de ruido, donde pueden verse diferencias relativas más notables. Parece que el modelo de NN tiende a interpretar ruido o estructuras no lineales como parte de la señal asociada a β1. A medida que el SNR mejora, las estimaciones se estabilizan progresivamente en torno a cero. La zona de alto ruido, donde la señal es muy débil, presenta un comportamiento especialmente errático.

Figura 12. Escenario 3 - Relación entre SNR y la Diferencia Relativa en el Estimador β1 (Zoom en Alto Ruido)

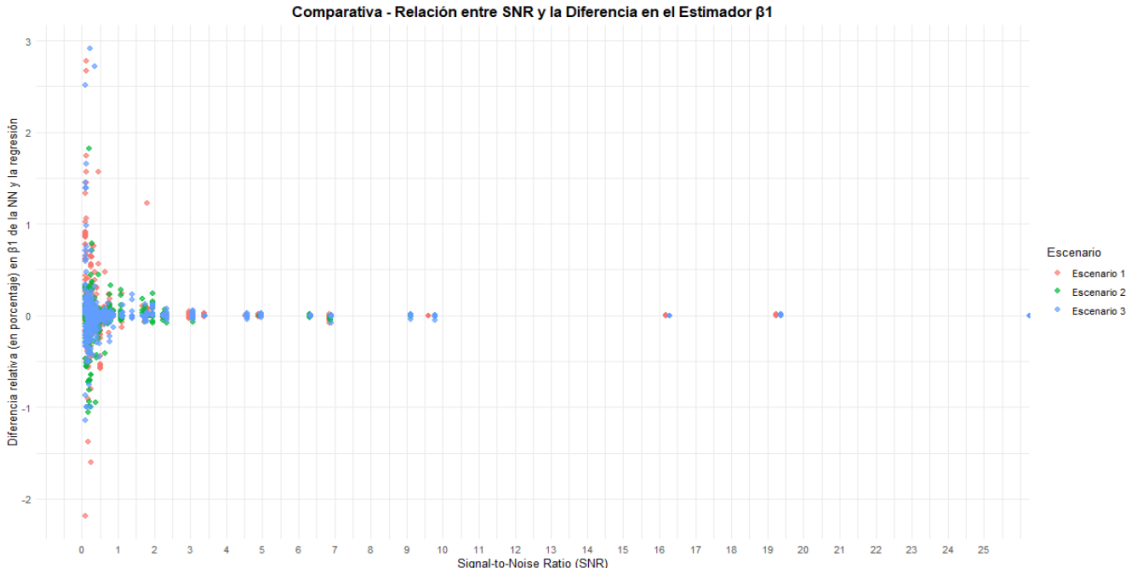


Fuente: Elaboración propia

En la zona de alto ruido del escenario 3 (figura 12), observamos el mismo patrón que en los escenarios anteriores.

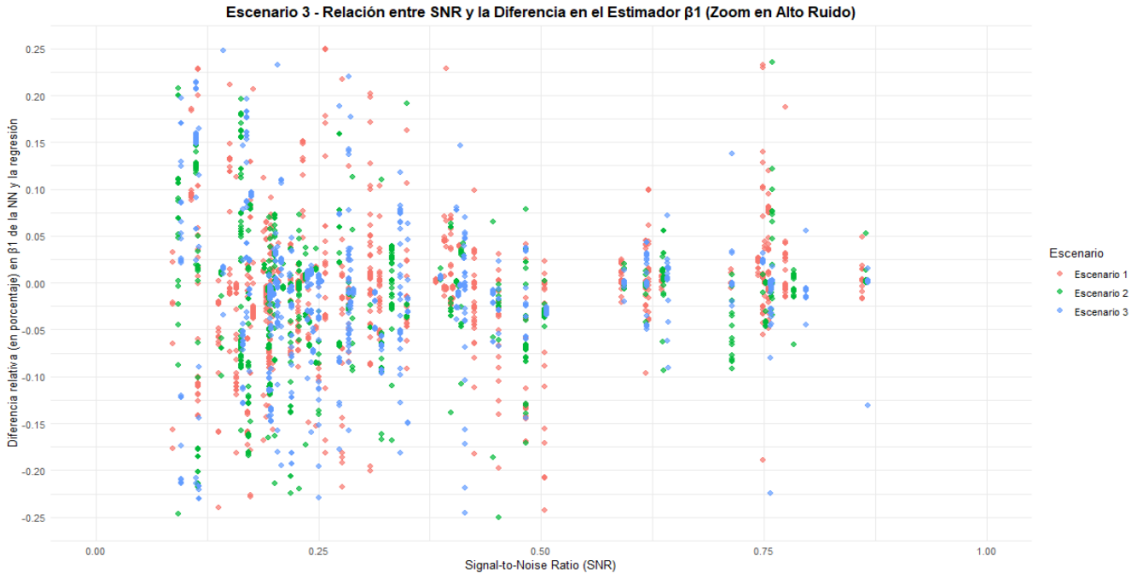
La figura 13 muestra la comparativa de los tres escenarios, y la Figura 14 esa misma comparativa en la zona de alto ruido. En condiciones de muy bajo SNR (alta presencia de ruido), todos los escenarios presentan una notable dispersión en sus estimaciones, con errores que pueden llegar a superar el umbral del 50 %, aunque la mayoría se encuentran por debajo del 25%. No se aprecian diferencias evidentes entre los tres escenarios, lo que apunta a que la complejidad estructural del modelo no parece ser un factor crítico en la calidad de la estimación.

Figura 13. Comparativa - Relación entre SNR y la Diferencia Relativa en el Estimador β1



Fuente: Elaboración propia

Figura 14. Comparativa - Relación entre SNR y la Diferencia Relativa en el Estimador β1 (Zoom en Alto Ruido)



Fuente: Elaboración propia

Las últimas gráficas presentadas (Figuras 15 y 16) permiten ver cómo evoluciona la diferencia relativa entre la estimación del coeficiente β1 obtenido por la Red Neuronal y el obtenido por la regresión lineal, en función del Signal-to-Noise Ratio (SNR) y a través de distintos escenarios. En cada escenario se ha calculado la desviación relativa promedio en valor absoluto (para evitar compensaciones por signo). Cuando el SNR es elevado, es decir, cuando la señal de información supera claramente al ruido, ambas estimaciones tienden a converger, mostrando que el modelo neuronal reproduce fielmente la estructura lineal de referencia. Esta estabilidad en entornos con buen SNR confirma que, en condiciones informativamente favorables, métodos como NeuralSens ofrecen resultados coherentes con los modelos clásicos como el OLS (Ordinary Least Squares, o Mínimos Cuadrados Ordinarios).

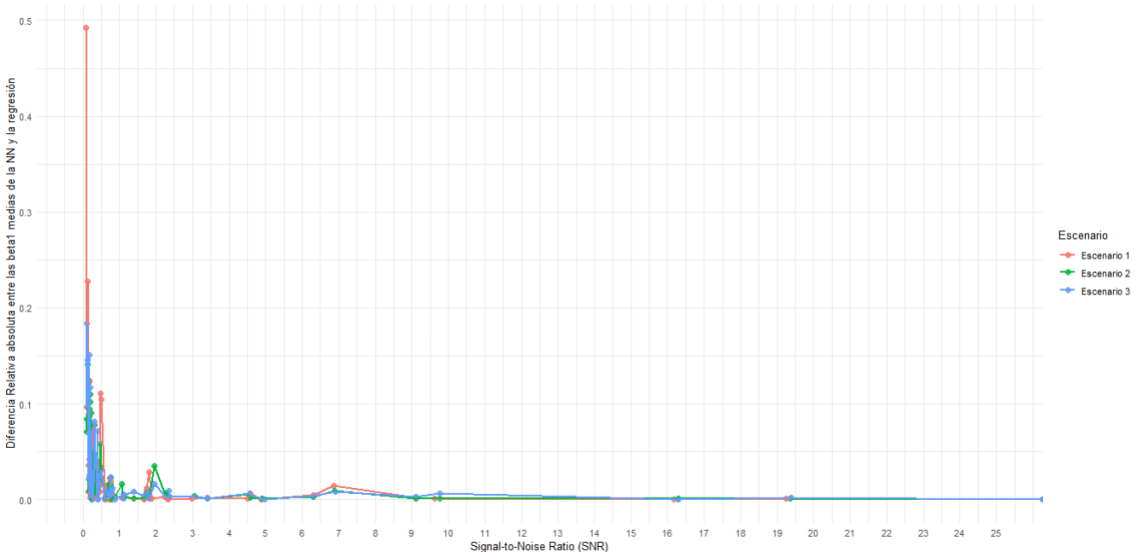
Sin embargo, el comportamiento del modelo cambia radicalmente cuando el SNR es bajo, es decir, cuando el dominio va de la mano del ruido y no de la señal. La realidad definida por los valores de $R^2 < 0.5$ es la norma para un amplio abanico de campos de las ciencias sociales, incluido la educación, dónde el ruido de los datos suele ser mucho mayor que en los entornos de las ciencias exactas por la cantidad de influencias difíciles de controlar o cuantificar. En este sentido, el entrenamiento de las redes neuronales descubre dos tipos de riesgos relacionados con la aleatoriedad de la semilla.

Por un lado, nos encontramos con el *Riesgo Catastrófico de Semilla*, poco frecuente, pero de altísimo impacto por posibles desviaciones superiores al 25 % en los valores esperados. Este primer riesgo reflejaría bien una infraestimación o sobreestimación del modelo, lo que supondría que las sensibilidad estimada de la variable podría diferir, en ocasiones, por más de un 25 % con respecto a su valor en el modelo de referencia (OLS) .

Por otro lado, tenemos el *Riesgo de Fluctuación de Semilla*, mucho más probable que el anterior, pero con valores más moderados al tratarse de errores inferiores a ese umbral del 25%. Este error, a diferencia del anterior, reflejaría una grave falta de estabilidad en el modelo neuronal cuando el ruido es elevado, pues la red tendría serias dificultades para

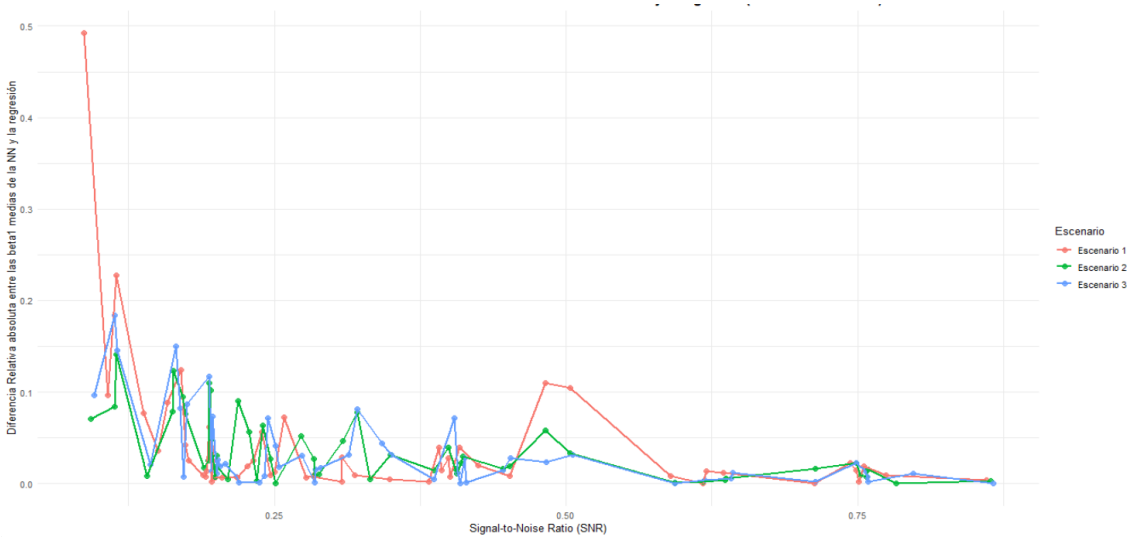
separar y aislar la señal de información del ruido, generando un error más moderado pero sistemático.

Figura 15. Indicador – Diferencia Relativa (en Valor Absoluto) entre las β1 medias de las NN y la Regresión



Fuente: Elaboración propia

Figura 16. Indicador – Diferencia Relativa (en Valor Absoluto) entre las β1 medias de las NN y la Regresión (Zoom en Alto Ruido)



Fuente: Elaboración propia

Así las cosas, puede decirse que ambos tipos de riesgo comprometen la estabilidad de los resultados y reflejan una limitación estructural en modelos que se apoyan en aleatorización interna para la inicialización de parámetros. No obstante, estos problemas

se pueden mitigar aplicando una estrategia: entrenar múltiples redes neuronales con diferentes semillas y calcular la media de las sensibilidades obtenidas en todas las redes entrenadas. Cuando se aplica este procedimiento, se comprueba que se neutraliza en gran medida la variabilidad introducida por la aleatoriedad de los valores obtenidos, además de reducir la diferencia notablemente, dejándola, en la mayoría de los casos, en valores inferiores al 1% con respecto de los coeficientes para β1 obtenidos por el modelo de regresión lineal. En la zona de alto ruido la desviación media pasa a estar en torno al 4%, estando la mayoría de los niveles de ruido de dicha zona por debajo del 1.5% de error.

5. Conclusiones

Las últimas gráficas muestran cómo varía la diferencia relativa entre la estimación del coeficiente β_1 obtenida por una red neuronal y la obtenida por regresión lineal, en función del nivel de ruido medido a través del Signal-to-Noise Ratio (SNR). Cuando el SNR es alto, es decir, cuando la señal predomina claramente sobre el ruido, ambas estimaciones tienden a coincidir, lo que demuestra que el modelo neuronal es capaz de reproducir con precisión la estructura lineal subyacente. Este resultado valida el uso de NeuralSens en contextos donde los datos son informativamente ricos, ya que sus resultados son coherentes con los de modelos clásicos como OLS.

Sin embargo, la situación cambia en entornos con bajo SNR, donde el ruido domina sobre la señal, una realidad frecuente en las ciencias sociales en general y en educación en particular, debido a la alta variabilidad y dificultad para controlar todas las influencias. En este escenario, el entrenamiento de redes neuronales revela dos tipos de riesgos asociados a la aleatoriedad de la semilla: el riesgo catastrófico de semilla, relativamente poco frecuente pero potencialmente grave, con errores superiores al 25 %, y el riesgo de fluctuación de semilla, más común, pero con desviaciones moderadas. Ambos riesgos evidencian una limitación estructural de las redes neuronales, derivada de su proceso de inicialización aleatoria.

No obstante, el trabajo confirma que estos problemas pueden mitigarse sustancialmente aplicando una estrategia simple pero eficaz: entrenar múltiples redes con distintas semillas y promediar sus resultados. La estrategia planteada reduce el margen de error en la totalidad de valores del SNR. Esta reducción y adecuación a los valores obtenidos por el modelo de regresión lineal es especialmente notable en la zona de alto ruido dónde los riesgos antes presentes, llevando a errores incluso superiores al 25 %, se han visto reducidos enormemente hasta lograr un error no superior al 4% como antes se adelantaba. Así, este enfoque no solo mejora la precisión, sino que también devuelve al análisis neuronal la capacidad explicativa e interpretativa, convirtiendo a estos modelos de NN en una alternativa válida frente a modelos clásicos como la regresión, que hasta ahora han sido la opción de referencia en los campos de las ciencias sociales. Así, el análisis desarrollado en este trabajo apunta a que, pese al carácter inherentemente ruidoso de los datos en estos campos, es posible emplear NN.

6. Anexo

Script 1: Generación de Resultados

```
################################################################################
# TFG - Analytics
# Pablo Cadierno Redondo
# Parte 1: generación de datos
################################################################################
# Carga de paquetes necesarios
library(caret)
library(NeuralSens)
library(gridExtra)
library(boot)
library(MBESS) # Para calcular el Signal to noise
################################################################################
# Se establece directorio de trabajo
remove(list=ls())
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
################################################################################
# Se establece la configuracion de la validacion cruzada 10-folds
control <- trainControl(
  method = "cv", number = 10,
  summaryFunction = defaultSummary,
  returnResamp = "final", savePredictions = TRUE)
################################################################################
#########
# Se configuran los parámetros generales de los datos sinteticos
semilla_inicial <- 2025
iteraciones <- 50 # iteraciones para cada paso de ruido
step_ruido <- 80 # pasos de incremento de ruido
```

```
incremento_ruido <- 0.1 # parametro de incremento de ruido: 0.1
ratio_train <- 1


#############################################################################
# ESCENARIO 1: Relacion lineal y = a x1 + b x2
#############################################################################
# En este escenario se va a testar el efecto del ruido en una relacion lineal
# y = a x1 + b x2
# se irá incrementando progresivamente el nivel de ruido en sucesivas iteraciones
#############################################################################
# Creamos un df donde guardar los resultados de las iteraciones
variables_lineal <- c ("sd_ruido",
                       "correlacion_x1y",
                       "correlacion_x2y",
                       "neuronas_capa_oculta",
                       "valor_decay",
                       "mean_beta1",
                       "std_beta1",
                       "mean_beta2",
                       "std_beta2",
                       "reg_beta1",
                       "reg_beta2",
                       "SNR")

Resultados_lineal <- data.frame(matrix(ncol = length(variables_lineal), nrow = 0))
colnames(Resultados_lineal) <- variables_lineal
summary(Resultados_lineal)


set.seed(semilla_inicial)
```

```
x1 <- rnorm(n = 300, mean = 10, sd = 2)
x2 <- rnorm(n = 300, mean = 10, sd = 2)

y <- x1 - x2
modelo_OLS <- lm(y~x1+x2)
summary(modelo_OLS)

for (i in 1:step_ruido)
{
 set.seed(semilla_inicial*i)
 sd_ruido <- incremento_ruido*(i+1) #nivel de ruido
 y <- (x1 - x2) + rnorm(n = 300, mean = 0, sd = sd_ruido)
 cor(x1, y)
 cor(x2, y)
for (j in 1:iteraciones)
 {
  semilla <- semilla_inicial*j
     #Guardado de la correlacion entre y-x
  correlacion_x1y <- cor(x1, y)
  correlacion_x2y <- cor(x2, y)

  df <- data.frame(x1,x2, y)
  set.seed(semilla)
  train_Index <- createDataPartition(df$y,
                                     p = ratio_train,
                                     list = FALSE,
                                     times = 1)

  fTR <- df[train_Index, ] # train set
    # Scale data
```

```
fTR$x1 <- scale(fTR$x1)
fTR$x2 <- scale(fTR$x2)
fTR$y <- scale(fTR$y)

# Ajuste de la red neuronal
set.seed(semilla)

Modelo_red_neuronal <- train(
 form = y ~ ., data = fTR,
 method = "nnet",
 linout = TRUE,
 maxit = 250,
 tuneGrid = expand.grid(
  size = seq(1, 4, length.out = 4), # Malla de entre 1 y 4 neuronas en capa oculta
  decay = 10 ^ seq(-7, -1, length.out = 7)),# decay entre 10^-7 y 0.1
 trControl = control,
 metric = "RMSE",
 trace=FALSE)

summary(Modelo_red_neuronal)

mejor_tune <- Modelo_red_neuronal$bestTune
neuronas_capa_oculta <- mejor_tune$size
valor_decay <- mejor_tune$decay

# Ajuste de la red neuronal con los parametros optimos identificado
set.seed(semilla)
Modelo_red_neuronal <- train(
 form = y ~ ., data = fTR,
 method = "nnet",
```

```
 linout = TRUE,
 maxit = 250,
 tuneGrid = data.frame(
  size = neuronas_capa_oculta,
  decay = valor_decay),
 trControl = control,
 metric = "RMSE",
 trace=FALSE)

summary(Modelo_red_neuronal)

set.seed(semilla)
Sens_Analysis <- SensAnalysisMLP(Modelo_red_neuronal,
                        plot = FALSE)

summary(Sens_Analysis)
print(sd_ruido)

mean_beta1 <- Sens_Analysis$sens$.outcome$mean[1]
std_beta1 <- Sens_Analysis$sens$.outcome$std[1]
mean_beta2 <- Sens_Analysis$sens$.outcome$mean[2]
std_beta2 <- Sens_Analysis$sens$.outcome$std[2]

# Comparación con un modelo de regresión
modelo_reg <- lm(y ~ x1 + x2, data = fTR)
summary(modelo_reg)
betas_reg <- coef(modelo_reg)
reg_beta1 <- betas_reg["x1"]
reg_beta2 <- betas_reg["x2"]
```

```
  # Cálculo del Signal to Noise Ratio (SNR)
  resumen_modelo <- summary(modelo_reg)
  R2 <- resumen_modelo$r.squared
  N <- nrow(fTR)
  p <- 2 # Numero de predictores
  SNR <- signal.to.noise.R2(R.Square=R2, N=N, p=p)
  SNR <- SNR$phi2.hat

  # Guardado de los resultados
  resultados_iteracion <- data.frame(sd_ruido,correlacion_x1y,correlacion_x2y,
                    neuronas_capa_oculta, valor_decay,
                    mean_beta1, std_beta1,mean_beta2, std_beta2,
                    reg_beta1,reg_beta2, SNR)
  colnames(resultados_iteracion) <- variables_lineal
  Resultados_lineal <- rbind(Resultados_lineal,resultados_iteracion)
 }
}

Resultados_lineal <- Resultados_lineal
View(Resultados_lineal)
plot(Resultados_lineal$correlacion_x1y,Resultados_lineal$mean_beta1)
save(Resultados_lineal, file = "Resultados_lineal.RData")
```

```
#########################################################################
# ESCENARIO 2: Relacion lineal y = a*x1 + b*x2  + c*x3
#########################################################################
# En este escenario se va a testar el efecto del ruido en una relacion lineal
# y = a*x1 + b*x2  + c*x3
# se irá incrementando progresivamente el nivel de ruido en sucesivas iteraciones
#########################################################################
# Creamos un df donde guardar los resultados de las iteraciones
variables_lineal2 <- c ("sd_ruido",
                        "correlacion_x1y",
                        "correlacion_x2y",
                        "correlacion_x3y",
                        "neuronas_capa_oculta2",
                        "valor_decay2",
                        "mean_beta1",
                        "std_beta1",
                        "mean_beta2",
                        "std_beta2",
                        "mean_beta3",
                        "std_beta3",
                        "reg_beta1",
                        "reg_beta2",
                        "reg_beta3",
                        "SNR")

Resultados_lineal2 <- data.frame(matrix(ncol = length(variables_lineal2), nrow = 0))
colnames(Resultados_lineal2) <- variables_lineal2
summary(Resultados_lineal2)

set.seed(semilla_inicial)
```

```
x1 <- rnorm(n = 300, mean = 10, sd = 2)
x2 <- rnorm(n = 300, mean = 10, sd = 2)
x3 <- rnorm(n = 300, mean = 10, sd = 2)

y <- x1 - x2 #al ser x3 una variable neutra, su relación con "y" es equivalente a 0
modelo_OLS2 <- lm(y~x1+x2+x3)
summary(modelo_OLS2)

for (i in 1:step_ruido)
{
 set.seed(semilla_inicial*i)
 sd_ruido <- incremento_ruido*(i+1) #nivel de ruido
 y <- (x1 - x2) + rnorm(n = 300, mean = 0, sd = sd_ruido)
 cor(x1, y)
 cor(x2, y)
 cor(x3, y)
 for (j in 1:iteraciones)
 {
  semilla <- semilla_inicial*j
  #Guardado de la correlacion entre y-x
  correlacion_x1y <- cor(x1, y)
  correlacion_x2y <- cor(x2, y)
  correlacion_x3y <- cor(x3, y)

  df <- data.frame(x1,x2, x3, y)
  set.seed(semilla)
  train_Index <- createDataPartition(df$y,
                                     p = ratio_train,
                                     list = FALSE,
                                     times = 1)
```

```
fTR <- df[train_Index, ] # train set
  # Scale data
fTR$x1 <- scale(fTR$x1)
fTR$x2 <- scale(fTR$x2)
fTR$x3 <- scale(fTR$x3)
fTR$y <- scale(fTR$y)

# Ajuste de la red neuronal
set.seed(semilla)

Modelo_red_neuronal2 <- train(
 form = y ~ ., data = fTR,
 method = "nnet",
 linout = TRUE,
 maxit = 250,
 tuneGrid = expand.grid(
  size = seq(1, 4, length.out = 4), # Malla de entre 1 y 4 neuronas en capa oculta
  decay = 10 ^ seq(-7, -1, length.out = 7)), # decay entre 10^-7 y 0.1
 trControl = control,
 metric = "RMSE",
 trace=FALSE)

summary(Modelo_red_neuronal2)

mejor_tune2 <- Modelo_red_neuronal2$bestTune
neuronas_capa_oculta2 <- mejor_tune2$size
valor_decay2 <- mejor_tune2$decay

# Ajuste de la red neuronal con los parametros optimos identificados
```

```
set.seed(semilla)
Modelo_red_neuronal2 <- train(
 form = y ~ ., data = fTR,
 method = "nnet",
 linout = TRUE,
 maxit = 250,
 tuneGrid = data.frame(
  size = neuronas_capa_oculta2,
  decay = valor_decay2),
 trControl = control,
 metric = "RMSE",
 trace=FALSE)

summary(Modelo_red_neuronal2)

set.seed(semilla)
Sens_Analysis2 <- SensAnalysisMLP(Modelo_red_neuronal2,
                    plot = FALSE)

summary(Sens_Analysis2)
print(sd_ruido)

mean_beta1 <- Sens_Analysis2$sens$.outcome$mean[1]
std_beta1 <- Sens_Analysis2$sens$.outcome$std[1]

mean_beta2 <- Sens_Analysis2$sens$.outcome$mean[2]
std_beta2 <- Sens_Analysis2$sens$.outcome$std[2]

mean_beta3 <- Sens_Analysis2$sens$.outcome$mean[3]
```

```
  std_beta3 <- Sens_Analysis2$sens$.outcome$std[3]

  # Comparación con un modelo de regresion
  modelo_reg2 <- lm(y ~ x1 + x2 + x3, data = fTR)
  summary(modelo_reg2)
  betas_reg2 <- coef(modelo_reg2)
  reg_beta1 <- betas_reg2["x1"]
  reg_beta2 <- betas_reg2["x2"]
  reg_beta3 <- betas_reg2["x3"]

  # Cálculo del Signal to Noise Ratio (SNR)
  resumen_modelo2 <- summary(modelo_reg2)
  R2 <- resumen_modelo2$r.squared
  N <- nrow(fTR)
  p <- 3 # Numero de predictores
  SNR <- signal.to.noise.R2(R.Square=R2, N=N, p=p)
  SNR <- SNR$phi2.hat

  # Guardado de los resultados
  resultados_iteracion2 <- data.frame(sd_ruido,correlacion_x1y,correlacion_x2y,
correlacion_x3y,
                      neuronas_capa_oculta2, valor_decay2,
                      mean_beta1, std_beta1,mean_beta2, std_beta2, mean_beta3,
std_beta3,
                      reg_beta1,reg_beta2, reg_beta3, SNR)
  colnames(resultados_iteracion2) <- variables_lineal2

  Resultados_lineal2 <- rbind(Resultados_lineal2,resultados_iteracion2)
 }
}
```

```
Resultados_lineal2 <- Resultados_lineal2

View(Resultados_lineal2)
plot(Resultados_lineal2$correlacion_x1y,Resultados_lineal2$mean_beta1)
save(Resultados_lineal2, file = "Resultados_lineal2.RData")

#########################################################################
# ESCENARIO 3: Relacion lineal y = a*x1 + b*x2  + c*x3 + d*x4^2
#########################################################################
# En este escenario se va a testar el efecto del ruido en una relacion lineal
# y = a*x1 + b*x2  + c*x3 + d*x4^2
# se irá incrementando progresivamente el nivel de ruido en sucesivas iteraciones
#########################################################################
# Creamos un df donde guardar los resultados de las iteraciones
variables_lineal3 <- c ("sd_ruido",
                        "correlacion_x1y",
                        "correlacion_x2y",
                        "correlacion_x3y",
                        "correlacion_x4y",
                        "neuronas_capa_oculta3",
                        "valor_decay3",
                        "mean_beta1",
                        "std_beta1",
                        "mean_beta2",
                        "std_beta2",
                        "mean_beta3",
                        "std_beta3",
                        "mean_beta4",
                        "std_beta4",
```

```
                "reg_beta1",
                "reg_beta2",
                "reg_beta3",
                "reg_beta4",
                "SNR")

Resultados_lineal3 <- data.frame(matrix(ncol = length(variables_lineal3), nrow = 0))
colnames(Resultados_lineal3) <- variables_lineal3
summary(Resultados_lineal3)

set.seed(semilla_inicial)
x1 <- rnorm(n = 300, mean = 10, sd = 2)
x2 <- rnorm(n = 300, mean = 10, sd = 2)
x3 <- rnorm(n = 300, mean = 10, sd = 2)
x4 <- rnorm(n = 300, mean = 10, sd = 2)

y <- x1 - x2 + x4^2
modelo_OLS3 <- lm(y~x1+x2+x3+x4)
summary(modelo_OLS3)

for (i in 1:step_ruido)
{
 set.seed(semilla_inicial*i)
 sd_ruido <- incremento_ruido*(i+1) #nivel de ruido
 y <- (x1 - x2) + rnorm(n = 300, mean = 0, sd = sd_ruido)
 cor(x1, y)
 cor(x2, y)
 cor(x3, y)
 for (j in 1:iteraciones)
 {
```

```
semilla <- semilla_inicial*j



#Guardado de la correlacion entre y-x
correlacion_x1y <- cor(x1, y)
correlacion_x2y <- cor(x2, y)
correlacion_x3y <- cor(x3, y)
correlacion_x4y <- cor(x4, y)

df <- data.frame(x1,x2, x3, x4, y)
set.seed(semilla)
train_Index <- createDataPartition(df$y,
                                   p = ratio_train,
                                   list = FALSE,
                                   times = 1)

fTR <- df[train_Index, ] # train set
# Scale data
fTR$x1 <- scale(fTR$x1)
fTR$x2 <- scale(fTR$x2)
fTR$x3 <- scale(fTR$x3)
fTR$x4 <- scale(fTR$x4)
fTR$y <- scale(fTR$y)

# Ajuste de la red neuronal
set.seed(semilla)

Modelo_red_neuronal3 <- train(
 form = y ~ ., data = fTR,
 method = "nnet",
```

```
 linout = TRUE,
 maxit = 250,
 tuneGrid = expand.grid(
  size = seq(1, 4, length.out = 4), # Malla de entre 1 y 4 neuronas en capa oculta
  decay = 10 ^ seq(-7, -1, length.out = 7)),# decay entre 10^-7 y 0.1
 trControl = control,
 metric = "RMSE",
 trace=FALSE)

summary(Modelo_red_neuronal3)

mejor_tune3 <- Modelo_red_neuronal3$bestTune
neuronas_capa_oculta3 <- mejor_tune3$size
valor_decay3 <- mejor_tune3$decay

# Ajuste de la red neuronal con los parametros optimos identificados
set.seed(semilla)
Modelo_red_neuronal3 <- train(
 form = y ~ ., data = fTR,
 method = "nnet",
 linout = TRUE,
 maxit = 250,
 tuneGrid = data.frame(
  size = neuronas_capa_oculta3,
  decay = valor_decay3),
 trControl = control,
 metric = "RMSE",
 trace=FALSE)
  summary(Modelo_red_neuronal3)
set.seed(semilla)
```

```
Sens_Analysis3 <- SensAnalysisMLP(Modelo_red_neuronal3,
                                  plot = FALSE)

summary(Sens_Analysis3)
print(sd_ruido)

mean_beta1 <- Sens_Analysis3$sens$.outcome$mean[1]
std_beta1 <- Sens_Analysis3$sens$.outcome$std[1]

mean_beta2 <- Sens_Analysis3$sens$.outcome$mean[2]
std_beta2 <- Sens_Analysis3$sens$.outcome$std[2]

mean_beta3 <- Sens_Analysis3$sens$.outcome$mean[3]
std_beta3 <- Sens_Analysis3$sens$.outcome$std[3]

mean_beta4 <- Sens_Analysis3$sens$.outcome$mean[4]
std_beta4 <- Sens_Analysis3$sens$.outcome$std[4]

# Comparación con un modelo de regresion
modelo_reg3 <- lm(y ~ x1 + x2 + x3 + x4, data = fTR)
summary(modelo_reg3)
betas_reg3 <- coef(modelo_reg3)
reg_beta1 <- betas_reg3["x1"]
reg_beta2 <- betas_reg3["x2"]
reg_beta3 <- betas_reg3["x3"]
reg_beta4 <- betas_reg3["x4"]

# Cálculo del Signal to Noise Ratio (SNR)
resumen_modelo3 <- summary(modelo_reg3)
R2 <- resumen_modelo3$r.squared
```

```
  N <- nrow(fTR)
  p <- 4 # Numero de predictores
  SNR <- signal.to.noise.R2(R.Square=R2, N=N, p=p)
  SNR <- SNR$phi2.hat

  # Guardado de los resultados
  resultados_iteracion3 <- data.frame(sd_ruido,correlacion_x1y,correlacion_x2y,
correlacion_x3y, correlacion_x4y,
                      neuronas_capa_oculta3, valor_decay3,
                      mean_beta1, std_beta1,mean_beta2, std_beta2, mean_beta3,
std_beta3, mean_beta4, std_beta4,
                      reg_beta1,reg_beta2, reg_beta3, reg_beta4, SNR)
  colnames(resultados_iteracion3) <- variables_lineal3

  Resultados_lineal3 <- rbind(Resultados_lineal3,resultados_iteracion3)
 }
}

Resultados_lineal3 <- Resultados_lineal3
View(Resultados_lineal3)
plot(Resultados_lineal3$correlacion_x1y,Resultados_lineal3$mean_beta1)
save(Resultados_lineal3, file = "Resultados_lineal3.RData")
```

Script 2: Análisis de Resultados

```
#####################################################################
# Parte 2: Analisis de resultados
#####################################################################
# Carga de paquetes necesarios
library(caret)
```

```
library(NeuralSens)
library(gridExtra)
library(boot)
library(MBESS)
library(dplyr)
library(ggplot2)

######################################################################
# Se establece directorio de trabajo
remove(list=ls())
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
######################################################################
# ESCENARIO 1: Relacion lineal y = a x1 + b x2
######################################################################

temp_env <- new.env()
load("resultados_lineal.RData", envir = temp_env)
Resultados_lineal <- temp_env[[ls(temp_env)[1]]]

load("Resultados_lineal.RData")
summary(Resultados_lineal)
Resultados_lineal<- Resultados_lineal
View(Resultados_lineal)

Resultados_lineal$dif_b1 <- Resultados_lineal$mean_beta1-Resultados_lineal$reg_beta1
Resultados_lineal$dif_b1_relativa <-
(Resultados_lineal$dif_b1)/(Resultados_lineal$reg_beta1)

Resultados_lineal$dif_b2 <- Resultados_lineal$mean_beta2-Resultados_lineal$reg_beta2
```

```
Resultados_lineal$dif_b2_relativa <-
(Resultados_lineal$dif_b2)/(Resultados_lineal$reg_beta2)


summary(Resultados_lineal)


#####################################################################
# Relacion SNR diferencia relativa en beta 1
ggplot(Resultados_lineal, aes(x = SNR, y = dif_b1)) +
 geom_point(size = 2, color = "blue", alpha = 0.7) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia en β1 de la NN y la regresión",
    title = "Escenario 1 - Relación entre SNR y la Diferencia en el Estimador β1") +
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))



ggplot(Resultados_lineal, aes(x = SNR, y = dif_b1_relativa)) +
 geom_point(size = 2, color = "blue", alpha = 0.7) +
 labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
    title = "Escenario 1 - Relación entre SNR y la Diferencia Relativa en el Estimador β1") +
 scale_x_continuous(limits = c(0, 25), breaks = seq(0, 25, by = 1)) +  # Zoom en la zona de
alto ruido
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))


#####################################################################
# Relacion SNR diferencia relativa en beta 1: ampliación de la zona de alto ruido


#SNR = R2/(1-R2)
```

```
#SNR<1 → El ruido es mayor que la señal

#SNR≈1 → La señal y el ruido están equilibrados

#SNR>3 → La señal es más fuerte que el ruido

#SNR>10 → El modelo explica casi toda la variabilidad


# Desviación media en zona de alto ruido

alto_ruido <- subset(Resultados_lineal,Resultados_lineal$SNR<1)

summary(alto_ruido)

mean(alto_ruido$dif_b1)

mean(alto_ruido$dif_b1_relativa)


ggplot(alto_ruido, aes(x = SNR, y = dif_b1_relativa)) +

 geom_point(size = 2, color = "blue", alpha = 0.7) +

 labs(x = "Signal-to-Noise Ratio (SNR)",

    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",

    title = "Escenario 1 - Relación entre SNR y la Diferencia en el Estimador β1 (Zoom en Alto Ruido)") +

 scale_x_continuous(limits = c(0, 1), breaks = seq(0, 1, by = 0.25)) +  # Zoom en la zona de alto ruido

 scale_y_continuous(limits = c(-0.25, 0.25),

            breaks = seq(-0.25, 0.25, by = 0.05),

            labels = function(x) format(round(x, 2), nsmall = 2, scientific = FALSE))  +

 theme_minimal() +

 theme(plot.title = element_text(hjust = 0.5, face = "bold"))


#####################################################################

# ESCENARIO 2: Relacion lineal y = a*x1 + b*x2 + c*x3

#####################################################################

load("Resultados_lineal2.RData")

summary(Resultados_lineal2)
```

```
Resultados_lineal2<- Resultados_lineal2

View(Resultados_lineal2)


Resultados_lineal2$dif_b1 <- Resultados_lineal2$mean_beta1-
Resultados_lineal2$reg_beta1

Resultados_lineal2$dif_b1_relativa <-
(Resultados_lineal2$dif_b1)/(Resultados_lineal2$reg_beta1)


Resultados_lineal2$dif_b2 <- Resultados_lineal2$mean_beta2-
Resultados_lineal2$reg_beta2

Resultados_lineal2$dif_b2_relativa <-
(Resultados_lineal2$dif_b2)/(Resultados_lineal2$reg_beta2)


Resultados_lineal2$dif_b3 <- Resultados_lineal2$mean_beta3-
Resultados_lineal2$reg_beta3

Resultados_lineal2$dif_b3_relativa <-
(Resultados_lineal2$dif_b3)/(Resultados_lineal2$reg_beta3)


summary(Resultados_lineal2)

######################################################################
# Relacion SNR diferencia relativa en beta 1


ggplot(Resultados_lineal2, aes(x = SNR, y = dif_b1)) +
 geom_point(size = 2, color = "blue", alpha = 0.7) +
 labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia en β1 de la NN y la regresión",
    title = "Escenario 2 - Relación entre SNR y la Diferencia en el Estimador β1") +
 theme_minimal() +  # Tema limpio y moderno
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))



ggplot(Resultados_lineal2, aes(x = SNR, y = dif_b1_relativa)) +
```

```
 geom_point(size = 2, color = "blue", alpha = 0.7) +
 labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
    title = "Escenario 2 - Relación entre SNR y la Diferencia en el Estimador β1") +
 scale_x_continuous(limits = c(0, 25), breaks = seq(0, 25, by = 1)) +  # Zoom en la zona de alto ruido
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))
######################################################################
# Relacion SNR diferencia relativa en beta 1: ampliación de la zona de alto ruido
# Desviación media en zona de alto ruido
alto_ruido2 <- subset(Resultados_lineal2,Resultados_lineal2$SNR<1)
summary(alto_ruido2)
mean(alto_ruido2$dif_b1)
mean(alto_ruido2$dif_b1_relativa)



ggplot(alto_ruido2, aes(x = SNR, y = dif_b1_relativa)) +
 geom_point(size = 2, color = "blue", alpha = 0.7) +
 labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
    title = "Escenario 2 - Relación entre SNR y la Diferencia en el Estimador β1 (Zoom en Alto Ruido)") +
 scale_x_continuous(limits = c(0, 1), breaks = seq(0, 1, by = 0.25)) +  # Zoom en la zona de alto ruido
 scale_y_continuous(limits = c(-0.25, 0.25),
                    breaks = seq(-0.25, 0.25, by = 0.05),
                    labels = function(x) format(round(x, 2), nsmall = 2, scientific = FALSE))  +
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
#####################################################################
# ESCENARIO 3: Relacion lineal y = a*x1 + b*x2 + c*x3 + x4^2
#####################################################################

load("Resultados_lineal3.RData")

summary(Resultados_lineal3)

Resultados_lineal3<- Resultados_lineal3

View(Resultados_lineal3)


Resultados_lineal3$dif_b1 <- Resultados_lineal3$mean_beta1-
Resultados_lineal3$reg_beta1

Resultados_lineal3$dif_b1_relativa <-
(Resultados_lineal3$dif_b1)/(Resultados_lineal3$reg_beta1)


Resultados_lineal3$dif_b2 <- Resultados_lineal3$mean_beta2-
Resultados_lineal3$reg_beta2

Resultados_lineal3$dif_b2_relativa <-
(Resultados_lineal3$dif_b2)/(Resultados_lineal3$reg_beta2)


Resultados_lineal3$dif_b3 <- Resultados_lineal3$mean_beta3-
Resultados_lineal2$reg_beta3

Resultados_lineal3$dif_b3_relativa <-
(Resultados_lineal2$dif_b3)/(Resultados_lineal3$reg_beta3)


Resultados_lineal3$dif_b4 <- Resultados_lineal3$mean_beta4-
Resultados_lineal3$reg_beta4

Resultados_lineal3$dif_b4_relativa <-
(Resultados_lineal3$dif_b4)/(Resultados_lineal3$reg_beta4)

summary(Resultados_lineal3)


#############################################
# Relacion SNR diferencia relativa en beta 1
```

```
ggplot(Resultados_lineal3, aes(x = SNR, y = dif_b1)) +
  geom_point(size = 2, color = "blue", alpha = 0.7)
   labs(x = "Signal-to-Noise Ratio (SNR)",
      y = "Diferencia en β1 de la NN y la regresión",
      title = "Escenario 3 - Relación entre SNR y la Diferencia en el Estimador β1") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))


ggplot(Resultados_lineal3, aes(x = SNR, y = dif_b1_relativa)) +
  geom_point(size = 2, color = "blue", alpha = 0.7) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
      y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
      title = "Escenario 3 - Relación entre SNR y la Diferencia en el Estimador β1") +
  scale_x_continuous(limits = c(0, 25), breaks = seq(0, 25, by = 1)) +  # Zoom en la zona de alto ruido
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))
################################################
# Relacion SNR diferencia relativa en beta 1: ampliación de la zona de alto ruido
# Desviación media en zona de alto ruido
alto_ruido3 <- subset(Resultados_lineal3,Resultados_lineal3$SNR<1)
summary(alto_ruido3)
mean(alto_ruido3$dif_b1)
mean(alto_ruido3$dif_b1_relativa)


ggplot(alto_ruido3, aes(x = SNR, y = dif_b1_relativa)) +
  geom_point(size = 2, color = "blue", alpha = 0.7) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
```

```
    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
    title = "Escenario 3 - Relación entre SNR y la Diferencia en el Estimador β1 (Zoom en Alto Ruido)") +
 scale_x_continuous(limits = c(0, 1), breaks = seq(0, 1, by = 0.25)) +  # Zoom en la zona de alto ruido
 scale_y_continuous(limits = c(-0.25, 0.25),
                    breaks = seq(-0.25, 0.25, by = 0.05),
                    labels = function(x) format(round(x, 2), nsmall = 2, scientific = FALSE))  +
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))

######################################################################
#COMPARATIVA DE RESULTADOS
######################################################################
#Unificación de los dataframes de Resultados

Resultados_lineal$Escenario <- "Escenario 1"
Resultados_lineal2$Escenario <- "Escenario 2"
Resultados_lineal3$Escenario <- "Escenario 3"

resultadostc <- unique(c(colnames(Resultados_lineal),
                         colnames(Resultados_lineal2),
                         colnames(Resultados_lineal3)))

# Función para agregar columnas faltantes con NA
completar_columnas <- function(df, resultadostc) {
 for (col in resultadostc) {
  if (!(col %in% colnames(df))) {
   df[[col]] <- NA  # Si falta la columna, se añade con valores NA
  }
```

```
 }
 return(df[, resultadostc])  # Reordenamos para que todas las tablas tengan el mismo orden
}

# Aplicar la función a cada data.frame
Resultados_lineal <- completar_columnas(Resultados_lineal, resultadostc)
Resultados_lineal2 <- completar_columnas(Resultados_lineal2, resultadostc)
Resultados_lineal3 <- completar_columnas(Resultados_lineal3, resultadostc)

# Combinar los data.frames
Resultados <- rbind(Resultados_lineal, Resultados_lineal2, Resultados_lineal3)

#####################################################################
#Unificación de los dataframes de Alto Ruido

alto_ruido$Escenario <- "Escenario 1"
alto_ruido2$Escenario <- "Escenario 2"
alto_ruido3$Escenario <- "Escenario 3"



alto_ruidotc <- unique(c(colnames(alto_ruido),
                         colnames(alto_ruido2),
                         colnames(alto_ruido3)))

# Función para agregar columnas faltantes con NA
completar_columnas <- function(df, alto_ruidotc) {
 for (col in alto_ruidotc) {
   if (!(col %in% colnames(df))) {
     df[[col]] <- NA
```

```
  }
 }
 return(df[, alto_ruidotc])
}
# Aplicar la función a cada data.frame
alto_ruido <- completar_columnas(alto_ruido, alto_ruidotc)
alto_ruido2 <- completar_columnas(alto_ruido2, alto_ruidotc)
alto_ruido3 <- completar_columnas(alto_ruido3, alto_ruidotc)

Alto_RuidoR<- rbind(alto_ruido, alto_ruido2, alto_ruido3)

######################################################################
#Visualización de dataframes conjuntos

ggplot(Resultados, aes(x = SNR, y = dif_b1, color = Escenario)) +
 geom_point(size = 2, alpha = 0.7) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia en β1 de la NN y la regresión",
    title = "Comparativa - Relación entre SNR y la Diferencia en el Estimador β1") +
  theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))

ggplot(Resultados, aes(x = SNR, y = dif_b1_relativa, color = Escenario)) +
 geom_point(size = 2, alpha = 0.7) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
    title = "Comparativa - Relación entre SNR y la Diferencia en el Estimador β1") +
 scale_x_continuous(limits = c(0, 25), breaks = seq(0, 25, by = 1)) +
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
ggplot(Alto_RuidoR, aes(x = SNR, y = dif_b1_relativa, color = Escenario)) +
 geom_point(size = 2, alpha = 0.7) +
 labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia relativa (en porcentaje) en β1 de la NN y la regresión",
    title = "Escenario 3 - Relación entre SNR y la Diferencia en el Estimador β1 (Zoom en Alto Ruido)") +
 scale_x_continuous(limits = c(0, 1), breaks = seq(0, 1, by = 0.25)) +  # Zoom en la zona de alto ruido
 scale_y_continuous(limits = c(-0.25, 0.25),
            breaks = seq(-0.25, 0.25, by = 0.05),
            labels = function(x) format(round(x, 2), nsmall = 2, scientific = FALSE))  +
 theme_minimal() +
 theme(plot.title = element_text(hjust = 0.5, face = "bold"))  # Centra el título y lo pone en negrita



######################################################################
#GRÁFICA: Estimación de β1 (NN) en función del ruido
######################################################################
#Diferencia respecto a la beta obtenida por regresión (Resultados)

drelativa <- Resultados %>%
 group_by(SNR, Escenario) %>%
 summarise(
  media_sensibilidad_res = mean(mean_beta1, na.rm = TRUE),
  media_regresion_res = mean(reg_beta1, na.rm = TRUE),
  diferencia_relativa_res = abs(media_sensibilidad_res - media_regresion_res)/media_regresion_res,
  .groups = "drop"
 )
```

```
ggplot(drelativa, aes(x = SNR, y = diferencia_relativa_res, color = Escenario)) +
  geom_point(size = 2.5, alpha = 0.8) +
  geom_line(size = 1) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia Relativa absoluta entre las beta1 medias de la NN y la regresión",
    title = "Indicador: Diferencia Relativa Absoluta entre las beta1 medias de la NN y la regresión " ) +
  scale_x_continuous(limits = c(0, 25), breaks = seq(0, 25, by = 1)) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold")
  )



drelativa_alto_ruido <- Alto_RuidoR %>%
  group_by(SNR, Escenario) %>%
  summarise(
    media_sensibilidad = mean(mean_beta1, na.rm = TRUE),
    media_regresion = mean(reg_beta1, na.rm = TRUE),
    diferencia_relativa = abs(media_sensibilidad - media_regresion)/media_regresion,
    .groups = "drop"
  )


ggplot(drelativa_alto_ruido, aes(x = SNR, y = diferencia_relativa, color = Escenario)) +
  geom_point(size = 2.5, alpha = 0.8) +
  geom_line(size = 1) +
  labs(x = "Signal-to-Noise Ratio (SNR)",
    y = "Diferencia Relativa absoluta entre las beta1 medias de la NN y la regresión",
```

```
  title = "Indicador: Diferencia Relativa Absoluta entre las beta1 medias de la NN y la regresión (Zoom en Alto Ruido)"  ) +
 theme_minimal() +
 theme(
  plot.title = element_text(hjust = 0.5, face = "bold")
 )
```

7. Declaración respecto al uso de Chat GPT u otras herramientas de IAG

Por la presente, yo, Pablo Cadierno Redondo, estudiante de E3 Analytics (doble grado en Derecho y Business Analytics), de la Universidad Pontificia Comillas al presentar mi Trabajo Fin de Grado titulado "Análisis de la dependencia a la semilla en modelos de redes neuronales con análisis post-hoc basado en NeuralSens: implicaciones para los modelos de rendimiento académico y abandono en educación superior.", declaro que he utilizado la herramienta de Inteligencia Artificial Generativa ChatGPT u otras similares de IAG de código sólo en el contexto de las actividades descritas a continuación:

1. Usado junto con otras herramientas, como Science, para identificar referencias preliminares que luego he contrastado y validado.
2. **Interpretador de código:** Para realizar análisis de datos preliminares.
3. **Constructor de plantillas:** Para diseñar formatos específicos para secciones del trabajo.
4. **Corrector de estilo literario y de lenguaje:** Para mejorar la calidad lingüística y estilística del texto.
5. **Sintetizador y divulgador de libros complicados:** Para resumir y comprender literatura compleja.
6. **Generador de datos sintéticos de prueba:** Para la creación de conjuntos de datos ficticios.
7. **Revisor:** Para recibir sugerencias sobre cómo mejorar y perfeccionar el trabajo con diferentes niveles de exigencia.
8. **Traductor:** Para traducir textos de un lenguaje a otro.

Afirmo que toda la información y contenido presentados en este trabajo son producto de mi investigación y esfuerzo individual, excepto donde se ha indicado lo contrario y se han dado los créditos correspondientes (he incluido las referencias adecuadas en el TFG y he explicitado para que se ha usado ChatGPT u otras herramientas similares). Soy consciente de las implicaciones académicas y éticas de presentar un trabajo no original y acepto las consecuencias de cualquier violación a esta declaración.

Firmado:

8. Referencias

Arroyo-Barrigüete, J. L., Carabias-López, S., Borrás-Pala, F., & Martín-Antón, G. (2023a). Gender differences in Mathematics Achievement: The case of a business school in Spain. *SAGE Open*, *13*(2), 21582440231166922.

Arroyo-Barrigüete, J.L.; Carabias López, S.; Hernández, A. y Seguram M. (2023b). Effect of advanced high school major on mathematical performance at university: A comparative study in Business Administration degrees. *Revista de Educacion*, 1, 115–140. https://doi.org/10.4438/1988- 592X-RE-2023-402-597

Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

El Bannany, M., Khedr, A. M., Sreedharan, M., & Kanakkayil, S. (2021). Financial distress prediction based on multi-layer perceptron with parameter optimization. *IAENG International Journal of Computer Science*, *48*(3), 1-12.

Goldberg, Y. (2017). *Neural network methods in natural language processing*. Morgan & Claypool Publishers.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press.

Mandler, H., & Weigand, B. (2024). A review and benchmark of feature importance methods for neural networks. *ACM Computing Surveys*, *56*(12), 1-30. https://doi.org/10.1145/3679012

Menacho Chiok, C. H. (2014). *Modelos de regresión lineal con redes neuronales. Anales Científicos, 75 (2), 253-260*. https://doi.org/10.21704/ac.v75i2.961

Muñoz San Roque, A., Alfaya, D., Pizarroso, J., & Portela, J. (4 de febrero de 2025). *Tendencias en explicabilidad de modelos MLP* [Conferencia]. YouTube. https://www.youtube.com/watch?v=nJnyPQei3zs

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25, pp. 15-24). San Francisco, CA, USA: Determination press.

Olden, J. D., Joy, M. K., & Death, R. G. (2004). An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological modelling*, *178*(3-4), 389-397. https://doi.org/10.1016/j.ecolmodel.2004.03.013

Ortiz-Lozano, J. M., Aparicio-Chueca, P., Triadó-Ivern, X. M., & Arroyo-Barrigüete, J. L. (2024). Early dropout predictors in social sciences and management degree students. *Studies in Higher Education*, *49*(8), 1303-1316. https://doi.org/10.1080/03075079.2023.2264343

Pizarroso, J., Portela J., & Muñoz, A. (2022) NeuralSense: Sensitivity analysis of neural networks. Journal of Statistical Software, 102(7). 10.18637/jss.v102.i07

RStudio Team. (2024). RStudio: Integrated Development for R (Versión R 4.2.2) [Software]. Posit Software, PBC. https://posit.co

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536. https://doi.org/10.1038/323533a0

Samek, W., & Müller, K. R. (2019). Towards explainable artificial intelligence. Explainable AI: interpreting, explaining and visualizing deep learning, 5-22. https://doi.org/10.1007/978-3-030-28954-6_1

Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge University Press.

Tang, Z., Wang, H., Yi, X., Zhang, Y., Kwong, S., & Kuo, C. C. J. (2022). Joint graph attention and asymmetric convolutional neural network for deep image compression. *IEEE Transactions on Circuits and Systems for Video Technology*, *33*(1), 421-433. 10.1109/TCSVT.2022.3199472

Zhang, Z., Beck, M. W., Winkler, D. A., Huang, B., Sibanda, W., & Goyal, H. (2018). Opening the black box of neural networks: methods for interpreting neural network models in clinical applications. *Annals of translational medicine*, *6*(11), 216. 10.21037/atm.2018.05.32

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2020). Dive into deep learning. Cambridge University Press.