uc3m | Universidad Carlos III de Madrid

Máster Universitario en Ciencias Actuariales y Financieras
2023-2025

Trabajo Fin de Máster

"Optimización de las Predicciones de Severidad mediante el Uso de Redes Neuronales en Seguros Paramétricos de Riesgos Catastróficos Climáticos"

Víctor Mérida Martínez

Tutor
Raquel Pérez Calderón

Madrid, junio 2025

DETECCIÓN DEL PLAGIO
La Universidad utiliza el programa **Turnitin Feedback Studio** para comparar la originalidad del trabajo entregado por cada estudiante con millones de recursos electrónicos y detecta aquellas partes del texto copiadas y pegadas. Copiar o plagiar en un TFM es considerado una **Falta Grave**, y puede conllevar la expulsión definitiva de la Universidad.

RESUMEN

La combinación de seguros paramétricos y redes neuronales artificiales será el futuro en la industria aseguradora porque además de que no existe ninguna otra herramienta matemática cuya capacidad de aprendizaje sea ilimitada es la única forma de minimizar los costes, maximizar el valor real de las indemnizaciones y optimizar la transferencia de recursos ante cualquier riesgo asegurable, gracias en cierta medida a la evolución computacional y a la paulatina disponibilidad de información de carácter público en internet. Por otro lado, destacar la colaboración público – privada para implementar los seguros paramétricos ante riesgos catastróficos porque es el estado quien tiene la capacidad y autoridad económica suficiente para hacer el seguro indexado más accesible, el sector asegurador tiene el conocimiento y calidad del dato suficiente para implementar las modelizaciones y , por último, es el sector financiero quien tiene la habilidad de absorber el impacto a través de derivados financieros.

El presente trabajo sienta las bases para la modelización de seguros paramétricos catastróficos de huracanes a través de redes neuronales artificiales en el Condado de Lee en Florida, utilizando exclusivamente datos meteorológicos y de asegurados disponibles en internet. Además, se pretende implementar una técnica novedosa en la arquitectura neuronal a través de funciones de activación ajustadas a las indemnizaciones históricas, reduciendo significativamente los errores de predicción que otros métodos más convencionales son incapaces, como los modelos generales lineales.

Palabras clave: Seguros paramétricos, Redes neuronales artificiales feedforward, Optimización, Datos climáticos, Inteligencia artificial, Predicción indemnizaciones, Funciones de activación específicas, Huracanes, Riesgos catastróficos.

ABSTRACT

The combination of parametric insurance and artificial neural networks will shape the future of the insurance industry. Not only is there no other mathematical tool with an unlimited learning capacity, but it is also the only way to minimize costs, maximize the real value of payouts, and optimize the transfer of resources in the face of any insurable risk. This is largely thanks to advances in computing and the gradual availability of public data on the internet. Additionally, public–private collaboration is essential for implementing parametric insurance in response to catastrophic risks. The government has the authority and economic capacity to make indexed insurance more accessible, the insurance sector possesses the expertise and data quality necessary for modeling, and the financial sector has the ability to absorb the impact through financial derivatives.

This work lays the foundation for modeling catastrophic parametric hurricane insurance using artificial neural networks in Lee County, Florida, relying solely on meteorological and policyholder data available online. Furthermore, it aims to implement an innovative technique in neural architecture by incorporating activation functions calibrated to historical payouts, significantly reducing prediction errors that more conventional methods, such as generalized linear models, are unable to achieve.

Keywords: Parametric insurance, Feedforward artificial neural networks, Optimization, Climate data, Artificial intelligence, Indemnity prediction, Specific activation functions, Hurricanes, Catastrophic risks.

DEDICATORIA

Deseo expresar mi agradecimiento a todos los que me han apoyado durante mi trayectoria académica, especialmente a mi familia, sin cuyo apoyo no habría sido posible, y a los que dedico esta obra. Mi agradecimiento asimismo a los profesores y al equipo de la Universidad Carlos III de Madrid por las oportunidades que me han ofrecido.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Motivación.

El objetivo de investigación surgió gracias a la necesidad de demostrar cómo a raiz de unas capacidades computacionales moderadas y una calidad del dato significativo se puede modelizar riesgos catastróficos asociados a huracanes a través de seguros paramétricos y redes neuronales artificiales, lo cual si se implementara a gran escala maximizaría la eficiencia y minimizaría el error de predicción de todos los seguros indexados comercializados. Además, concienciar de que la colaboración público – privada es estríctamente necesaria en los seguros paramétricos catastróficos ya que el estado tienen la capacidad y autoridad económica suficiente para crear un seguro catastrófico más económico, el sector asegurador tiene el conocimiento y la calidad del dato suficiente para implementar las modelizaciones necesarias y, por último, el sector financiero tiene la capacidad de absorber el posible impacto económico a través de derivados financieros.

1.2 Objetivos.

Asentar las bases de la arquitectura de las redes neuronales artificiales *feedforward* para su posterior implementación en la modelización de seguros paramétricos catastróficos de huracanes. Además, demostrar cómo a través de la implementación novedosa de funciones de activación ajustadas a los datos indemnizatorios históricos exclusivos a ciertas sumas aseguradas y zonas a través de redes neuronales artificiales es el método que más reduce el error de predicción de las indemnizaciones. Por último, mostrar que la calidad de datos públicos disponibles en internet es posible utilizarlos como base en las modelizaciones.

1.3 Descripción del trabajo

Se centra en la modelización de seguros paramétricos de huracanes en el Condado de Lee en Florida a través de redes neuronales artificiales. Por un lado, se demuestra la facilidad de interpretación en el pago de las indemnizaciones en comparación con los seguros indexados tradicionales ya que solo se requiere de dos partes para activarlos: que la agencia estadounidense Centro Nacional de Huracanes determine una tormenta tropical como huracán y que este se encuentre dentro de un perímetro preestablecido. Por otro lado, la tabla de pagos sería muy diferente a los seguros paramétricos tradicionales ya que solo haría falta la arquitectura final de la red neuronal y los pesos que se han atribuido a cada conexión neuronal en la fase de entrenamiento.

Por tanto, se enfocará en la modelización a traves de redes neuronales artificiales *feedforward* teniendo en cuenta dos aspectos innovadores a la hora de su creación: funciones de activación exclusivas aplicadas para el Condado de Lee y la transformación de las predicciones para recoger de una manera más eficiente las indemnizaciones nulas cuando se activa el seguro indexado. Por último, establecer qué metricas son las necesarias a monitorizar para que el modelo aprenda.

1.4 Resumen.

Las principales aportaciones de esta investigación son muy significativas ya que, a pesar de la calidad de los datos y la limitación computacional presente, se ha podido reducir aproximadamente en un 50 % los errores de predicción de las indemnizaciones reales, identificandolas significativamente en comparación con otras técnicas metemáticas tradicionales como los modelos generales lineales. En definitiva, se ha reducido tanto el error cuadrático medio como el error medio absoluto en un 46.09 % y 46.06 % respectivamente además de obtener un R^2 de 0.4295 y acertar 41 incidencias de un total de 96 en el huracán Milton ante la nulidad de los modelos de predicción tradicionales.

1.5 Contribuciones.

Las contribuciones del presente trabajo son numerosas, destacando: asentar las arquitecturas y algoritmos de las redes neuronales artificiales para llevar a cabo la modelización de seguros paramétricos catastróficos de huracanes, introducir nuevas herramientas matemáticas que son capaces de predecir significativamente las indemnizaciones a través del histórico de indemnizaciones ajustada a un area y suma asegurada específicas, afirmar la posibilidad de modelizar con datos que se encuentren de manera pública en internet y aportar nuevas publicaciones concernientes a la modelización de seguros paramétricos con redes neuronales artificiales ya que son escasas.

2. REVISIÓN DE LA LITERATURA

2.1 Seguros Paramétricos.

Un seguro paramétrico es un tipo de seguro que solo indemniza cuando se cumple una condición especifica, sin necesidad de evaluar daños reales a través de un peritaje. Es decir, es un tipo de seguro basado en índices predefinidos los cuales activaran unas indemnizaciones predeterminadas siempre y cuando se cumplan dentro de unos escenarios preestablecidos (SwissRe, 2024). Además, su implementación cada vez será mas latente gracias a la mayor información disponible en internet y a las nuevas técnicas matemáticas de predicción.

Sin embargo, esto no siempre fue así. Los seguros paramétricos comenzaron a comercializarse a finales del siglo XX de forma limitada (UNDP y Generali, 2024) pero fueron tres aspectos los cuales provocaron que un cambio radical en la implementación de este tipo de seguros: facilidad de datos públicos relevantes, inteligencia artificial y machine learning (AON, 2024). Es decir, la facilidad de obtención de datos relevantes, su preprocesamiento y la utilización de nuevas técnicas matemáticas a raíz de la evolución de la velocidad computacional ha hecho que los seguros paramétricos sean hoy una realidad.

Además, cabe destacar que los seguros paramétricos no surgieron por la necesitdad de un ahorro en los gastos operativos sino como una necesidad del mercado a la hora de cubrirse frente a riesgos que un seguro tradicional es incapaz de hacerlo. De acuerdo con (SwissRe, 2024), de los casi 300 mil millones de dólares que fueron a causa de desastres naturales en el año 2023 más del 50 % de las pérdidas no estaban aseguradas, pudiendo haberse evitada a través de seguros indexados y cuyas pérdidas las absorviera el mercado financiero. Por tanto, los seguros paramétricos son muy útiles a la hora de cubrir la brecha de protección, es decir, aquellas pérdidas que no están aseguradas ya sea porque no se ha contemplado ese riesgo o por la imposibilidad de medirlo.

Un seguro basado en índices se fabrica teniendo en cuenta las siguientes componentes (AON, 2024):

- Índice o parámetro: métrica utilizada para definir y determinar el montante de las indemnizaciones.
- Umbrales: determina en qué punto se activa el seguro paramétrico y siempre vinculado al parámetro.
- Cuadro de compensaciones: previamente se establece la magnitud de las indemnizaciones

dependiendo de las intensidades y umbrales del índice.

- Proveedor de información externo: agencia o empresa independiente la cual provee de los datos necesarios para la toma de decisiones cuando ocurra un evento especifico que active el seguro.
- Objeto asegurado: establecer inequivocablemente que activos se han de asegurar.
- Evaluación de las garantías propuestas: definir los casos en los que el asegurado es elegible para la indemnización además de establecer límites contractuales.

Por otro lado, y a pesar de la laboriosidad de la creación de este tipo de seguros, también tiene sus ventajas como (García Ocampo y López Moreira, 2024):

- Indemnizaciones: su principal ventaja es la rapidez con la que se producen los cobros ya que solo dependen de las métricas que se añaden al modelo paramétrico.
- Ahorro: solo se necesita a un equipo reducido de expertos para crearlos sin la necesidad posterior de peritos que deban cuantificar físicamente los daños reales.
- Reducción brecha de protección: pueden cubrir situaciones que no son posibles para seguros tradicionales por su incapacidad de medición.

Además, según (Ramy Vélez, 2023) se pueden dividir los seguros paramétricos en tres tipos:

- Paramétrico de Índice de Pérdidas Agregadas: hace referencia a los contratos cuya indemnización solo se basa en las pérdidas agregadas registradas en una región específica.
- Paramétrico Puro: solo se indemniza un montante preestablecido si se activa el seguro.
- Paramétrico Indexado: se basa en modelos matemáticos complejos cuyas indemnizaciones vienen determinadas por un índice predefinido que está relacionado con la severidad del evento.

Si se quiere establecer un seguro paramétrico indexado existen varias arquitecturas para construirlas, siendo las mas comunes Coberturas por Intensidad y "Cat in a Box". Por un lado, los seguros paramétricos de Cobertura por Intensidad se construyen a través de la potencia que manifiesta el índice que ocurre en un lugar concreto. Por otro lado, los "Cat in a Box" son aquellos que dependen de la intensidad del siniestro siempre vinculados a indemnizaciones y perímetros predefinidas contractualmente (SwissRe, 2024).

Por último, se debe de mencionar los posibles problemas que conllevan la adopción de los seguros paramétricos (UNDP y Generali, 2024). Se debe de entender que todos y cada uno de los seguros comercializados a nivel global contraen el problema llamado riesgo básico el cual se define como la divergencia entre la indemnización pagada con la pérdida real. Además, gracias a sus características específicas surgen dos tipos de riesgos básicos: riesgo básico positivo y negativo. El riesgo básico positivo se traduce como una sobreindemnización por la activación del seguro en relación con la pérdida real y el riesgo básico negativo cuando no se activa el seguro, pero existen pérdidas reales lo que conlleva a una infraindemnización. Sin embargo, con el avance tecnológico y matemático el riesgo básico se está reduciendo significativamente y cuya tendencia es cero.

2.2.1 Inicios de los seguros paramétricos climáticos.

Los seguros paramétricos se están convirtiendo en una de las fuentes más importantes de mitigación de riesgos actualmente. El problema residía en que existe una comonotonia entre el crecimiento de la volatilidad de los riesgos catastróficos y la solvencia económica de las empresas para poder mitigarlos, haciendo que los seguros tradicionales no sean idóneos para cubrir las pérdidas ó porque las aseguradoras rechazan asumir esos riesgos ó porque la prima propuesta es inasumible. Pero todo cambió gracias al fuerte avance tecnológico y matemático que ha sufrido la humanidad recientemente haciendo posible que floreciesen nuevas formas tanto de análisis del dato como de predicción siendo las redes neuronales artificiales la mejor propuesta.

Al presente, los seguros indexados se utilizan en todos los continentes y ámbitos económicos. Los ámbitos económicos de actuación son aparentemente ilimitados pero su eso más común son en desastres naturales, cambio climático, problemas en la cadena de suministro, crisis financieras, presupuestos de las empresas y ventas comerciales (SwissRe, 2024). Sin embargo, con un 56 % de cuota de mercado la actividad predominante de los seguros paramétricos son los relacionados con catástrofes naturales siendo Estados Unidos su mayor comercialización con una cuota del 35 % del mercado global, debido a su gran exposición a las catástrofes naturales (AON, 2024). En cuanto al área predominante de actuación, Europa es el mayor mercado en areas tan diversas como recursos energéticos, transporte marítimo, infraestructuras o climatología (Ramy Vélez, 2023). Además, el volumen de mercado de los seguros paramétricos es de alrededor de 18 mil millones de dólares en el año 2023 a una tasa de crecimiento anual compuesto entre un 6.6 % y un 11.5 % para mediados de 2030 (García Ocampo y López Moreira, 2024). Esto se traduce en un 0.8 por ciento de las primas totales anuales de los seguros de daños y patrimoniales (P&C) y un 13.7 por ciento de las pérdidas

aseguradas de los bonos catastróficos en 2023 (García Ocampo y López Moreira, 2024).

Además, en el presente trabajo se ha tenido en cuenta distintas fuentes oficiales de publicación y en la medida de lo posible las más actuales, siempre atendiendo de forma rigurosa y asertiva el objeto de investigación sobre los seguros paramétricos catastróficos a través de redes neuronales artificiales.

Las primeras publicaciones sobre cómo se podría implementar los seguros paramétricos climáticos fue gracias al economista Indio J.S. Chakravarti el cual planteaba en su libro Seguro Agrícola un seguro agrario que se basaba en el parámetro climático acumulación de lluvia por área predefinida para fijar el pago de las indemnizaciones, afirmando que aunque esta no fuera una forma completamente suficiente al no tener en cuentas todos los factores de riesgo que pudieran influir en una mala cosecha sí era una forma de poder tener controlado este problema siendo el estado quien proveyese de este instrumento (AON, 2024). La primera vez que se utilizó esta herramienta fueron a finales de 1990 en ciertos países asiáticos en donde a través de un índice reducían la exposición de las cosechas al tiempo atmosférico (García Ocampo y López Moreira, 2024).

Ante la constante aceptación de estos seguros otros países siguieron los mismos pasos. Un hecho que marcó un antes y después en los seguros paramétricos fue la temporada de huracanes en el Caribe durante el periodo 2004 – 2005 el cual provocó, además de resurgir la importancia de asegurarse ante riesgos climáticos por la pérdida económica del país caribeño Granada que sufrió del 200 % de su PIB haciendo insostenible cualquier actividad económica a largo plazo (World Bank, 2012), que el coste de los reaseguros catastróficos climáticos en ciertos tramos se doblasen en todos los países caribeños creandoles un grave problema ya que era la única forma que tenían de asegurarse.

Por tanto, nuevas fórmulas surgieron que además de cubrirlos minimizaban el coste estos contratos. Ante la casuística anteriormente mencionada, en 2007 el organismo internacional el Banco Mundial propuso la creación del Seguro contra Riesgos de Catástrofes del Caribe (CCRIF) en donde a través de una diversificación del riesgo entre todos los tomadores y con una nueva técnica matemática actuarial de cálculo usando seguros paramétricos se abrió una nueva posibilidad de asegurarse climáticamente con el menor coste posible, además de una rápida respuesta frente al cobro de las indemnizaciones.

Para lograr el objetivo planteado, el presente seguro paramétrico estaba basado en las siguientes hipótesis: mitigación y diversificación del riesgo a través de un seguro compartido entre todos los países tropicales, reducción del coste de las primas a través de reservas iniciales muy altas, costes operacionales mínimos por la naturaleza del seguro paramétrico y por la distribución de los gastos entre los integrantes, probabilidad de impago del siniestro de un 0.5 % ó que no pudiera pagarse uno

de cada 200 años, curvas hazard de frecuencia e intensidad de huracanes y terremotos además de la determinación de las indemnizaciones a través de la curva de un índice hazard propio, cuyos datos de entrada son factores de riesgo como la velocidad del viento o la intensidad del seísmo en los centros económicos más importantes de cada país (World Bank, 2007).

La mitigación del riesgo a través del seguro combinado es gracias a que no existe una correlación muy fuerte de los fenómenos naturales extremos entre las distintas regiones del caribe, haciendo que el seguro se reduzca casi a la mitad si lo comparamos con adquisiciones individuales. Además, las reservas totales se reducirían de media un 70 % (World Bank, 2007). Por otro lado, se podría reducir el coste de las primas hasta en un 30 % si las reservas iniciales fueran notables.

Gracias a la naturaleza de los seguros paramétricos su idoneidad es más que notable en cuanto a los costes operacionales ya que una vez construido toda la estructura del seguro, el mantenimiento sería aproximadamente de un 5 % de la prima total ya que se beneficiaria de un coste cero sobre la monitorización de la actividad ni ajustes en la función de pagos (World Bank, 2007).

En cuanto a las matemáticas utilizadas podemos destacar varios aspectos. Por un lado, se establece como norma que la probabilidad máxima de pérdida se limite a un VaR del 99.5 % en donde 199 de cada 200 años se podrá pagar los siniestros que pudieran acaecer. Por otro lado y más importante, los pagos de las indemnizaciones se basan en una curva de índice hazard formado por: intensidad de los siniestros basado en un factor de riesgo climático como la velocidad del viento, determinación de las curvas hazard de huracán y terremoto donde en el eje x se establece la intensidad del fenómeno (velocidad de viendo para el huracán y aceleración sísmica del terremoto) y en el eje y la probabilidad de que ocurra; se creará una curva hazard para cada una de las distintas áreas más importantes del país y las cuales se ponderarán su peso según su impacto económico en el país para crear el índice hazard de pago.

Una vez construidas las curvas hazard para cada zona económica relevante del país, se crea para cada región específica la función de daños la cual representa el coste medio del siniestro dependiendo de la intensidad de la propia curva hazard teniendo en cuenta las edificaciones y negocios económicos que se pudieran verse afectados ante los fenómenos naturales extremos. Una vez construido todas las curvas hazard y funciones de daños se establece el índice responsable de activar el seguro paramétrico y definir la indemnización del siniestro denominado índice hazard. Para crear el índice hazard, se combinan las intensidades y frecuencias de la curva hazard de cada zona estudiada las cuales se combinan con una ponderación preestablecida por la relevancia de impacto económico. Los factores de riesgo climáticos que se utilizan son tales como la velocidad del viendo o la evolución sísmica.

Por último y después de crear el índice hazard se tiene que establecer dos puntos de corte: el punto de activación en el cual se activa las indemnizaciones del seguro paramétrico y el punto de agotamiento donde se establece la máxima indemnización posible por este seguro el cual coincide con el VaR al 99.5 %. El punto de activación se calcula de forma única por cada país y coincide con el mínimo punto que puede haber un detrimento significativo en las arcas públicas. El punto de agotamiento se establece como el máximo dispuesto a pagar por la prima cada país. (World Bank, 2007).

Sin embargo, la carrera de los seguros paramétricos solo había comenzado. Tanto es así que la recopilación de datos climáticos en tiempo real junto con la inteligencia artificial y machine learning (AON, 2024) ha hecho posible que este tipo de seguros pudieran aplicarse a en su máximo esplendor optimizando tanto los costes operacionales como el valor de las primas. Aparentemente, la mejor optimización posible en cuanto a los seguros paramétricos se encuentra en las redes neuronales artificiales ya que su capacidad de aprendizaje no tiene límites además de su simplificación en la determinación de los pagos lo cual hará que el mercado tienda a comercializar este tipo de seguros.

2.3 Redes Neuronales Artificiales.

2.3.1 Contexto - Evolución de las Redes Neuronales Artificiales.

La primera idea que hizo emerger el estudio de la estructura de una red neuronal fue a finales del siglo XIX gracias a Alexander Bain el cual formuló cómo las redes neuronales biológicas funcionaban gracias a múltiples interconexiones neuronales las cuales cada una estaba destinada a una acción específica (Pires et al., 2023). Más tarde, a mediados del siglo XX, fueron los investigadores americanos Walter Pitts y Warren McCulloch los cuales definieron la primera red neuronal artificial y, lo más importante, que demostraron que combinaciones de operaciones lógicas binarias eran similares a las redes neuronales artificiales (Pires et al., 2023). La aportación más relevante sin duda fue la estructura neuronal artificial llamada perceptrón creada por Frank Rosenblatt el cual describió matemáticamente cómo si un problema tiene solución siempre se podrá resolver estimando los parámetros únicos de la estructura de la red neuronal artificial previamente entrenada (Pires et al., 2023). Es decir, demostró cómo la capacidad de aprendizaje de este tipo de estructura aparentemente no tiene límites.

Las redes neuronales artificiales son también llamadas Unidades de Procesamiento o Estructura de Procesamiento de Elementos (Gurney, 1999) las cuales su tamaño es reducido y construido paralelamente (Hertz et al., 1991). Todo cambió cuando se empezaron a estudiar la rapidez

computacional que estas estructuras podían dar.

Sin embargo, las redes neuronales artificiales no provocaban mejoras significativas en relación con otro métodos menos sofisticados y sencillos de implementar hasta que en 2010 los estudiantes computacionales de la Universidad de Toronto llamados Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton crearon la red neuronal convolucional profunda comúnmente conocida como "AlexNet". El único objetivo de esta red neuronal era la clasificación de imágenes cuya arquitectura neuronal estaba definida por cuatro pilares: 60 millones de parámetros, 650.000 neuronas, algoritmos basados en el descenso del gradiente para actualizar los pesos de los parámetros de forma efectiva y una función softmax de mil salidas para la propia clasificación (Krizhevsky et al., 2012).

Por tanto, su funcionamiento se puede dividir en varias partes en donde las más importantes son: por un lado, los valores de entrada han sido modificados a través del Análisis de Componentes Principales además de ser normalizados en cada capade la red neuronal. Por otro lado, la arquitectura de la red neuronal tiene cinco capas convolucionales las cuales están ligadas únicamente a los nuevos datos introducidos siendo estos las imágenes y tres capas completamente conectadas cuyo aprendizaje viene determinado por las otras capas anteriores. Además, las funciones de activación de cada neurona se hacen a través del método Unidades Lineales Rectificadas (ReLUs), es decir, el input de las neuronas viene determinadas por la función Max (0, x) donde x es el valor del input. Por último, se introduce una nueva técnica muy novedosa la cual especifica a ciertas neuronas que no actualicen sus pesos cuando está aprendiendo la red y así evitar de forma significativa sobreajustes en el modelo. Esta forma de proceder demostró como las redes neuronales eran superiores a otros modelos matemáticos sentando las bases para su posterior crecimiento exponencial.

2.3.2 Arquitectura.

El objetivo de todas las redes neuronales artificiales se basa en la predicción. Un conjunto de datos se subdivide en cada registro único en donde individualmente se van introduciendo en la red neuronal y en el cual se selecciona qué parte de la información proporcionada es la más relevante gracias a la adjudicación de los pesos. Los pesos han sido propuestos de la manera en que en la fase de training set se haya minimizado la función de pérdida, es decir, el error de todos los datos del training set. Este proceso ha sido posible porque previamente se han seleccionado el número de capas y neuronas, las funciones de activación de cada neurona, la función de pérdida y el algoritmo retropropagación que actualiza los pesos en base a la corrección del error de la función de pérdida. A continuación, se muestran las conexiones de una red neuronal artificial básica:

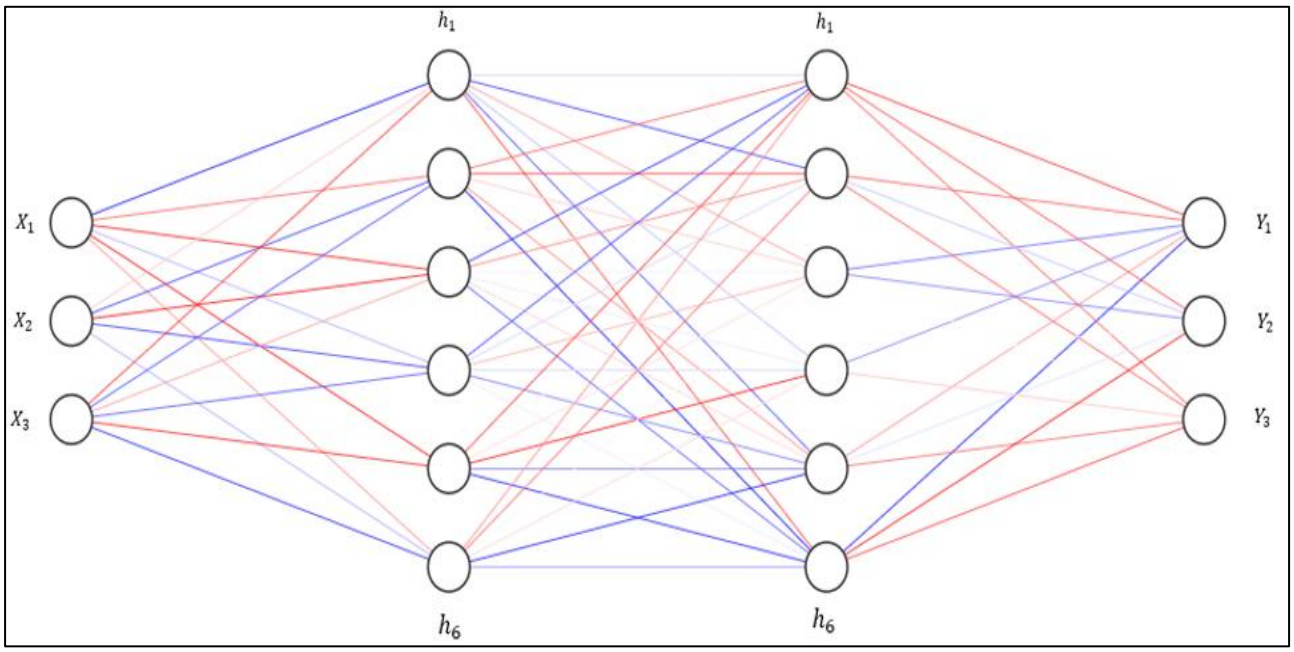

Figura 1 Arquitectura Red Neuronal Artificial. Fuente: Elaboración Propia

En base a la ilustración anterior, podemos observar cómo aparentemente su estructura es muy sencilla. Solo necesitamos saber el número de inputs X_n y número de outputs Y_n , el número de capas las cuales tendrán h_n neuronas y los pesos de cada conexión neuronal. Sin embargo, esto es solo la capa superficial para hacerlas funcionar. Por norma general, las redes neuronales artificiales se basan en alrededor de 14 familias de hiperparámetros los cuales solo existe un set único que funciona para cada problema existente. La tabla 1 muestra la familia de hiperparámetros más comunes que configuran una red neuronal:

Tabla 1 Familia de Hiperparámetros en Redes Neuronales Artificiales.

Hiperparámetros	Explicacion
Preprocesamiento	Transforma los datos mediante técnicas matemáticas
Función de perdida	Mide el error de predicción e influye en la actualización de los pesos
Optimizadores	Algoritmo que minimiza el error y responsible de actualizar los pesos
Funciones de activación	Modifican la salida de las neuronas
Pesos Iniciales	Atribuyen pesos según algoritmos predefinidos
Ratio de aprendizaje	Velocidad con la que aprende el modelo según su error
Tasa de abandono	Elimina aleatoriamente ciertas actualizaciones de los pesos
Coeficiente de penalización	Penaliza aquellos pesos que tienen mucho peso
Nueronas	Conexiones
Capas Ocultas	Donde se concetran las nueronas
Épocas	Veces que se actualiza el modelo
Tamaño del batch	tamaño en el que se dividen los datos
Early Stopping	Métrica que sirve para frenar el entrenamiento del modelo
Callbacks	Controla el comportamiento de la red para que mejore

Fuente:Elaboración Propia

Sin embargo, los hiperparámetros a su vez se subdividen en más categorías haciendo que las elecciones sean prácticamente infinitas. A continuación se definen todos los hiperparámetros.

2.3.3 Neuronas y capas.

Las neuronas solo tienen dos únicas responsabilidades. Por un lado, son las encargadas de transmitir la información a otras neuronas. Por otro lado, modifican el valor de los datos de entrada a través de una función predefinida. En un primer momento, reciben la información numérica de otras neuronas anteriores sumando todos los valores recibidos y posteriormente transformarlo a través de una función de activación. A continuación, como todas las capas están formadas por neuronas y por funciones de activación únicas a cada capa (práctica común), se vuelve a dividir el nuevo valor obtenido entre todas las neuronas de la siguiente capa las cuales recibirán más o menos información según la relevancia de la neurona a la solución final a través de unos pesos.

2.3.4 Funciones de activación.

El objetivo de la función de activación es transformar un valor numérico por otro a través de una fórmula definida. Por tanto, teóricamente existen tantas funciones de activación como funciones matemáticas. Sin embargo, normalmente se recurren a las mismas ya que se han demostrado su utilidad en diferentes campos.

Las diferentes funciones de activación tienen diferentes propósitos. Por ejemplo, las funciones Sigmoide o SofMax sirven para redes neuronales de clasificación mientras que la funciones ReLU o ELU trabajan en problemas de regresión. Sin embargo, existe un total de 29 funciones de activación (disponibles en Keras) las cuales, dependiendo de la arquitectura neuronal utilizada, la combinación entre ellas pueden ser ilimitadas. Las funciones de activación no mencionadas son: celu (Barron, 2017), gelu (Hendrycks y Gimpel, 2023), glu (Dauphin at al., 2017), hard shrink, hard sigmoide, hard silu, hard tanh, leaky relu, lineal, sigmoide logarítmica, softmax logarítmica, mish (Misra, 2020), relu6, selu, silu (Ramachandran et al., 2017), soft shrink, sparse plus, sparse max (Martins y Astudillo, 2016), squareplus (Barron, 2021), tangente hiperbólica, tangente hiperbólica shrink ó threshold. Por tanto, la elección de la función de activación es una parte crucial de la arquitectura de la red neuronal ya que solo existe una única combinación para obtener la solución en cada problema.

2.3.5 Funciones de pérdida.

Solo se utiliza en la fase de entrenamiento del modelo en donde se comenten errores de predicción. Su propósito es medir el error que, en sentido matemático, se traduce como la distancia media entre el valor real y la predicción. Su objetivo es minimizar o maximizar ese error a través de optimizadores. También las diferentes funciones de pérdida tienen distintas aplicaciones. Por ejemplo, la Entropía Cruzada Binaria es específica para cuando nuestro problema se basa en clasificar dos estados. Alternativamente se encuentra el error cuadrático medio que comúnmente es utilizado para problemas de regresión. Sin embargo, las funciones de pérdida son múltiples que comúnmente de dividen en tres familias de un total de 38 funciones disponibles en Keras: funciones de pérdida de probabilidad, funciones de pérdida de regresión y funciones de pérdida “Hinge” para máximo margen de clasificación. Por consiguiente, es importante identificar qué problema estas intentando resolver para aplicar el correcto.

2.3.6 Optimizadores y algoritmo de retropropagación.

La piedra angular de las redes neuronales se encuentra en el algoritmo de retropropagación y su optimizador correspondiente. Son los responsables de que la estructura aprenda, es decir, como se deben de actualizar los pesos para minimizar el error de la función de pérdida. Los pasos a seguir para actualizar los pesos son: elección de la función de pérdida, elección del optimizador, el número de veces o épocas que se van a actualizar los pesos y usar el algoritmo definido como retropropagación.

Desde un primer momento, se utilizó el algoritmo del descenso del gradiente para problemas de optimización. Sin embargo, en las redes neuronales presenta dos grandes inconvenientes. Por un lado, si la función de pérdida total no es convexa puede converger con un mínimo local en vez de un mínimo global. Por otro lado, la inversión de la matriz Hessiana puede que no exista un valor numérico y en el caso que existiese sería muy ineficiente (Denuit et al., 2019).

Por tanto, la alternativa que se propuso cambió en cierto modo la visión que se tenia sobre la eficiencia de las redes neuronales artificiales. El algoritmo retropropagación tiene como objetivo actualizar los pesos con pasos muy sutiles en la dirección contraria al gradiente (Denuit et al., 2019). Es decir, este algoritmo se divide en tres pasos:

1. Seleccionar arbitrariamente los pesos en las conexiones de cada neurona
2. Seleccionar un tamaño del paso inicial
3. Para cada vez que se actualicen los pesos:
    a. Cálculo del gradiente
    b. Actualizar el tamaño del paso
    c. Modificar los vectores de pesos

Figura 2. Algoritmo Retropropagación. Fuente: Elaboración propia a partir de Denuit et al. (2019)

Recordamos que el algoritmo Retropropagación es una forma específica de actuar frente a un problema de optimización del error y que, dentro de este algoritmo, lo más importante que se puede definir es qué fórmula se aplica con los gradientes para actualizar los pesos de las neuronas llamados optimizadores.

Por tanto, el optimizador se encarga de como usar los gradientes para que la red neuronal aprenda. Sin embargo, existe una gama amplia de optimizadores que en la mayoría de los casos el investigador no sabe cual es mejor para sus datos. En la actualidad, los optimizadores más importantes son Adam o Estimación de Momento Adaptativo (Kingma y Ba, 2015), SGD o Descenso de Gradiente Estocástico, RMSprop o Propagación de la raíz cuadrada media y Adagrad o Algoritmo de gradiente adaptativo (Duchi et al., 2011). Una explicación sencilla se puede llevar a cabo con el optimizador SGD cuya fórmula está presente en la figura 3:

$$\theta_{t+1} = \theta_t - \alpha \times g_t$$

Figura 3. Optimizador SGD. Fuente: Elaboración propia a partir de Denuit et al. (2019)

El cual la actualización de los pesos θ_{t+1} se consigue gracias a que los pesos actualizados se restan al producto de una tasa de aprendizaje α y del gradiente g_t, moviéndose en la dirección opuesta al gradiente (Denuit et al., 2019). El proceso es iterativo y se va repitiendo hasta que los pesos no cambien y se estabilicen.

Sin embargo, existen otros optimizadores tales como AdamW (Loshchilov y Hutter, 2019), Adadelta (Zeiler, 2012), Adagrad (Duchi et al., 2011), Adamax, Adafactor (Shazeer y Stern, 2018), Nadam (Dozat, 2016), Ftrl (McMahan et al., 2013), Lion (Chen et al., 2023) o Lamb (You et al., 2020) que multiplican el nivel de dificultad al no existir ningún procedimiento que exponga cual es el mejor excepto si utilizas todos y comparas los errores de predicción.

2.3.7 Tuner – optimizador de hiperparámetros.

La elección de los mejores hiperparámetros es una cuestión muy compleja ya que las posibles combinaciones son literalmente infinitas, pero existen ciertos algoritmos llamados optimizadores de hiperparámetros o *Tuners* cuyo propósito es aproximarse a los verdaderos parámetros generando sets aleatorios de estos a través de un algoritmo. No obstante, el problema inherente en este tipo de soluciones es la capacidad computacional que se necesita para encontrar una solución aproximada, pero ante el avance significativo de este campo este problema cada vez es menor.

Entonces se disponen de tres algoritmos optimizadores de hiperparámetros que facilitan su búsqueda, siendo estos el algoritmo de búsqueda aleatoria o *Random Search*, Optimización Bayesiana e

Hyperband. El objetivo principal de estos algoritmos es siempre la misma: quedarse con los hiperparámetros definidos en el modelo cuyo error de predicción sea el mínimo.

2.3.7.1 Búsqueda aleatoria o *Random Search*.

El presente optimizador es el mas básico e intuitivo de todos. A través de la determinación de los posibles valores que se puedan utilizar el algoritmo, de forma aleatoria, selecciona los hiperparámetros quedándose siempre con el modelo que haya dado mejores resultados. Se puede intuir como la búsqueda aleatoria no es la forma más eficiente de buscar las soluciones deseadas y, por tanto, se optan por otros métodos.

2.3.7.2 Optimización Bayesiana.

El optimizador Bayesiano toma un acercamiento más probabilístico al probar los diferentes hiperparámetros. Para ello, se mide el error de los modelos seleccionados y aprende de forma probabilística qué otras posibles modificaciones dentro de los hiperparámetros ya seleccionados se pueden implementar para reducir ese error de predicción. El presente algoritmo funciona de manera más eficiente que el de búsqueda aleatoria y, por tanto, sus resultados son más significativos.

2.3.7.3 *Hyperband*.

Este algoritmo también esta basado en la idea sobre la búsqueda aleatoria de los hiperparámetros, pero de una forma que la asignación computacional sea lo más eficiente posible adaptando las configuraciones de los hiperparámetros a aquellos que mejor resultado dan y limitando el número de configuraciones a probar (Li et al., 2018). Se configura el número máximo de hiperparámetros a utilizar y a través de búsquedas aleatorias se van obteniendo diferentes modelos, donde se van eliminando los modelos con mayor error y los mejores van avanzando hasta que se eliminen todos y quede solo el mejor modelo.

Para ello, solo hace falta la configuración de dos valores: R y μ. Por un lado, R representa el número máximo de hiperparámetros a utilizar en el algoritmo. Por otro lado, $\frac{1}{\mu}$ representa tanto por ciento de modelos que se van a ir eliminando en cada ronda y el número de modelos a probar en cada ronda se reduce a $(\log_n n) + 1$ comprobaciones (Li et al., 2018). Se ha comprobado que este algoritmo de

optimización de hiperparámetros funciona mejor que el de búsqueda aleatoria u optimización Bayesiana (Li et al., 2018).

2.3.8 Otros aspectos a tener en cuenta.

Además de tener en cuenta el número de neuronas y capas a utilizar, la función o funciones de activación, la función de pérdida y el optimizador para actualizar los pesos, se requieren más técnicas para llegar al objetivo deseado. Las más destacadas son las siguientes: establecer en qué capa o capas normalizar los datos, especificar qué función vas a usar y sus limitación al elegir los pesos aleatorios iniciales, concluir qué ratios de aprendizaje se utilizarán para actualizar los pesos, qué restricciones vas a poner a los pesos para que no haya un sobreajuste del modelo, establecer qué reducción utilizar para reducir el valor en la función de perdida y precisar el tamaño del lote o *batch size* para saber que subconjunto de datos utilizar en el entreno y predicción entre otros.

Por consiguiente, el término infinito está presente en las redes neuronales artificiales ya que la combinación de los diferentes hiperparámetros es ilimitado y solo existe una única arquitectura que mejor predice los valores.

3. DESCRIPCIÓN DE LA MUESTRA/ HIPÓTESIS DE PARTIDA

3.1 Base de datos asegurados.

En el año 1968 el congreso de los Estados Unidos, con el objetivo de dar estabilidad a la economía frente a eventos climáticos extremos, aprobó la Ley Nacional de Seguro contra Inundaciones. A raíz de esta ley, la Agencia Federal de Gestión de Emergencias (FEMA) y el sector asegurador propusieron crear una protección especial para las edificaciones que pudieran verse severamente afectadas por eventos meteorológicos severos, dando la opción de asegurar tanto el continente como el contenido (FEMA, 2025).

Por consiguiente, la presente base de datos sobre los asegurados corresponde a un total de 2.709.121 registros sobre incidencias por desastres naturales en Estados Unidos, de los cuales 241.738 corresponden al estado de Florida entre los periodos 2004 – 2024 y en donde surgieron un total de 23 tormentas tropicales que fueron denominadas huracanes. Sin embargo, el presente trabajo se enfoca exclusivamente en el Condado de Lee sobre aquellos asegurados cuya suma asegurada total del continente no supere los 50.000 dólares; Existe un total de 1745 registros cuyas incidencias corresponden a un total de 16 huracanes. La tabla 2 muestra más detalladamente la subdivisión de los registros:

Tabla 2. Incidencias por huracanes en el Condado de Lee.

Huracán	Fecha	Categoria	Número registros
Milton	2024 - Octubre	4	96
Helene	2024 - Septiembre	4	46
Debby	2024 - Agosto	1	1
Idalia	2023 - Agosto	4	2
Ian	2022 - Septiembre	5	870
Elsa	2021 - Julio	1	1
Eta	2020 - Noviembre	1	2
Sally	2020 - Septiembre	2	2
Irma	2017 - Septiembre	4	94
Matthew	2016 - Octubre	4	1
Isaac	2012 - Agosto	1	1
Wilma	2005 - Octubre	3	10
Jeanne	2004 - Septiembre	3	4
Ivan	2004 - Septiembre	4	3
Frances	2004 - Agosto	2	16
Charley	2004 - Agosto	4	596

Fuente: Elaboración propia

La actual base de datos presenta un total de 73 variables únicas en base a las características específicas de la edificación asegurada, y en donde finalmente se han seleccionado un total de 13 variables. Los criterios de selección de estos parámetros se han basado en dos puntos: encapsular la mayor cantidad de información posible del asegurado teniendo en cuenta las limitaciones computacionales existentes y que en la medida de lo posible presenten un coeficiente de variación significativo para que la red neuronal pueda aprender. A continuación, la tabla 15 muestra todas aquellas variables seleccionadas:

Tabla 3. Parámetros seleccionados y su coeficiente de variación.

Parámetros	Coeficiente Variación
ONG	20.86
Pequeño negocio	7.43
En alquiler	4.5
Edificaciones asegurados	2.52
Sector Censal	2.35
Indemnzación pagada por siniestro	1.61
Edificación Elevada	1.23
Primera residencia	1.22
Residencial	0.92
Ubicación Inmobiliario	0.65
Número de pisos	0.61
Suma asegurada continente	0.49
Fecha Construcción	0.006

Fuente:Elaboración Propia

Además, se debe de tener en cuenta que se han aplicado otros criterios de selección ya que se entiende como, por ejemplo, la fecha de construcción incide significativamente en la percepción de indemnización cuando ocurre un fenómeno meteorológico extraordinario ya que las recientes construcciones deben de seguir unos requisitos muy exigentes contra huracanes e inundaciones. Por tanto, cabe destacar que el presente trabajo se enfoca en la predicción de las indemnizaciones de los asegurados y, por tanto, la variable objetivo es la indemnización neta pagada por siniestro.

3.2 Base de datos climáticos.

La presente base de datos climáticos utilizada ha sido posible gracias al contenido facilitado por la empresa suiza “Open-Meteo”, la cual ofrece información meteorológica histórica global. A pesar de ofrecer decenas de variables climáticas, se han utilizado un total de cinco en base a dos criterios: que su coeficiente de variación sea significativo y que su efecto sea directo en cuanto a las indemnizaciones ya que los motivos de indemnizaciones frente a huracanes se reducen aproximadamente a dos consecuencias: velocidad del viento y precipitaciones totales. Las variables dependientes seleccionadas son: velocidad del viento máxima diaria a 10 metros, velocidad máxima de ráfaga de viendo a 10 metros, suma precipitación diaria, velocidad media de ráfaga de viento a 10 metros y velocidad media del viendo a 10 metros. A continuación, en la tabla 16 se muestra los diferentes coeficientes de variación en relación con cada parámetro.

Tabla 4. Parámetros seleccionados y su coeficiente de variación.

Parámetros meteorológicos	Coeficiente Variación
suma precipitación diaria	0.34
velocidad media del viendo a 10 metros	0.28
velocidad media de rafaga de viento a 10 metros	0.24
Velocidad viento maxima diaria 10 metros	0.14
velocidad máxima de ráfaga de viendo a 10 metros	0.13

Fuente:Elaboración Propia

Los datos correspondientes a cada asegurado se han obtenido de la siguiente manera: por un lado, se han contabilizado los registros únicos del sector censal y fecha ocurrencia del huracán. Por otro lado, para cada registro se ha tenido en cuenta todos los días en los que ha sido declarado el huracán y a través de APIs proporcionados por Open-Meteo y diferentes VPNs ha sido posible recopilar información climática de todos los días que ha durado el huracán por cada asegurado ya que la extracción de datos era limitada. Después, al tener recopilada la información climática según los días que había durado el huracán se ha decidido obtener el valor máximo recogido en ese intervalo de tiempo al entender que existen mayores probabilidades de indemnización en cuanto mayor es la velocidad del viendo o la suma de las precipitaciones diarias.

Sin embargo, los datos meteorológicos ofrecidos presentan una gran limitación: la precisión de los parámetros. Es decir, todos los datos históricos vienen recopilados por técnicas estadísticas cuya

precisión viene determinada por información recogida de las estaciones meteorológica en un perímetro determinado. Actualmente existen tres modelos: ERA 5 - Ensamble cuyos parámetros vienen determinados por la media de las estaciones climatológicas en un perímetro de 50 kilómetros, ERA5 contempla un perímetro de 25 kilómetros y ECMWF IF5 retiene un perímetro de 9 kilómetros. Sin embargo, los dos únicos modelos que ofrecen datos históricos de los últimos 25 años son ERA5 - Ensamble y ERA 5 por lo que se ha utilizado el modelo ERA5 por su mayor precisión y así poder utilizar información histórica para tener más datos para entrenar la red neuronal.

A pesar de que el presente trabajo necesite una mayor precisión meteorológica sobre la latitud y longitud de los diferentes asegurados, esta pequeña limitación hará que el entrenamiento de la red neuronal no sea tan preciso; pero gracias a las técnicas usadas por las redes neuronales hará que esas limitaciones no sean tan fuertes. Es por ello por lo que el presente trabajo ha recopilado información meteorológica de solo los días que ha ocurrido los últimos 16 huracanes que afectaron a algún asegurador en el Condado de Lee, extrayendo por cada sector censal de cada asegurado los datos climáticos específicos y extrayendo el valor máximo por las razones mencionadas anteriormente. A continuación, se muestra en la figura cuatro las 117 coordenadas utilizadas en la extracción de datos meteorológicos:

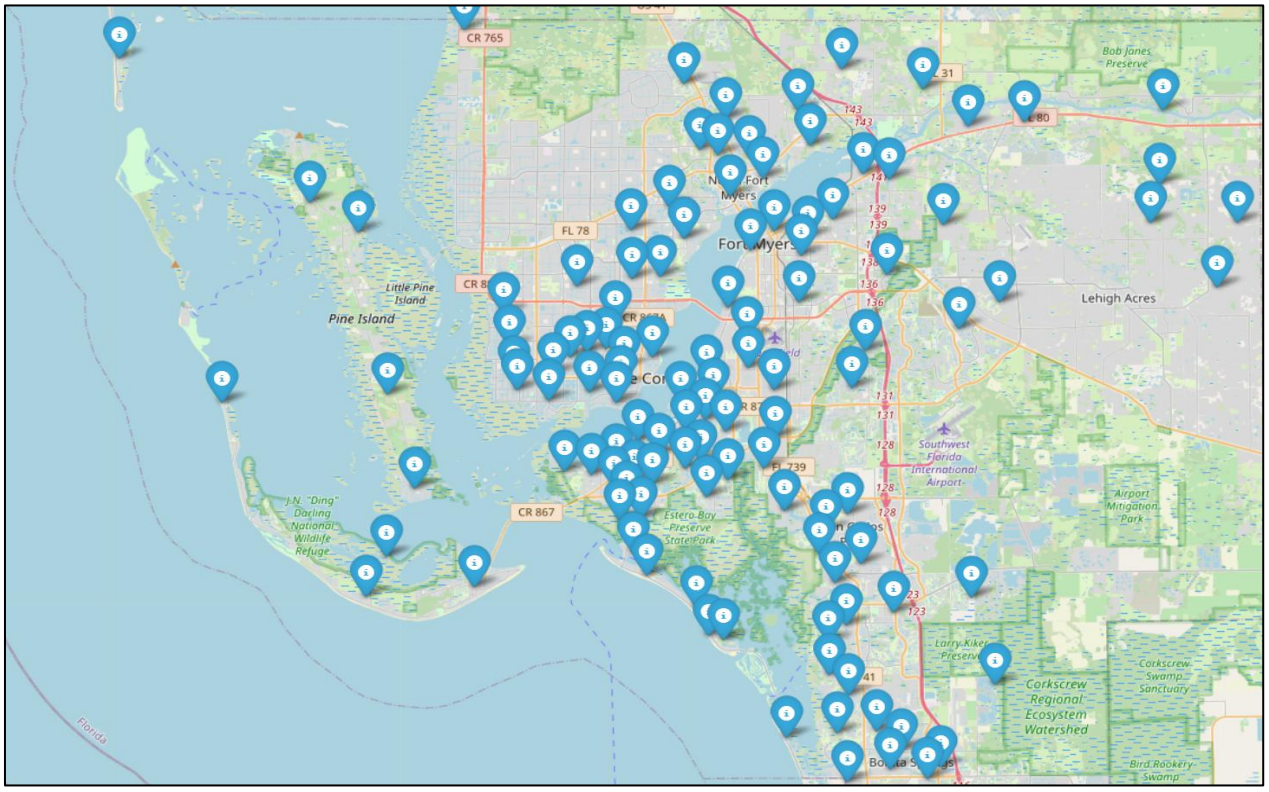

Figura 4. Localizaciones únicas de todos los asegurados en el huracán Milton. Fuente: Elaboración Propia

Además, hay que tener en cuenta un aspecto clave en cuanto al porqué de las indemnizaciones relacionados con los fenómenos atmosféricos: a pesar de que en una misma zona hubiese la misma intensidad de viento y lluvia, no afectará a los asegurados por igual ya que parte de las indemnizaciones dependen exclusivamente de las particularidades del terreno en el que se ubique además de los materiales, arquitectura y fecha de la construcción. Es por ello por lo que las redes neuronales artificiales son perfectas ante estas situaciones: generan patrones de comportamiento muy complejos que son indetectables para otros procedimientos estadísticos.

3.3 Base de datos elaboración propia.

La presente base de datos utilizada ha sido obtenida por elaboración propia, recopilando información relevante y relativa a los huracanes y asegurados. En particular, se ha contabilizado los días que el huracán estuvo cerca del estado de Florida, si hubo otros huracanes ese mismo año, si ha golpeado el huracán a Florida con la máxima intensidad, la distancia mínima en kilómetros entre las dos mayores intensidades del huracán frente al asegurado y la categoría máxima del huracán. A continuación, en la tabla 5 se muestra las variables utilizadas y sus coeficientes de variación:

Tabla 5. Parámetros seleccionados y su coeficiente de variación.

Parámetros	Coeficiente Variación
Huracanes mismo año	4.48
Golpea con máxima intensidad	1.29
Distancia mínima segunda mayor intensidad	1.02
Distancia mínima mayor intensidad	0.75
Duración huracán	0.29
Categoría Máxima huracán	0.14

Fuente:Elaboración Propia

Además, un aspecto muy significativo a la hora establecer las indemnizaciones es la intensidad de los huracanes y la distancia mínima que existe entre los asegurados y estos. Para obtener la información sobre la variable distancia mínima entre asegurado y huracán se ha procedido de la siguiente manera: por un lado, se ha recopilado los datos relativos a la longitud y latitud de las intensidades de cada uno de los huracanes y, por otro lado, las coordenadas relativas a las incidencias lo cual a través del código número uno del anexo se han calculado las distancias. En la figura 5 se muestra las distancias mínimas entre asegurados y huracanes individualmente:

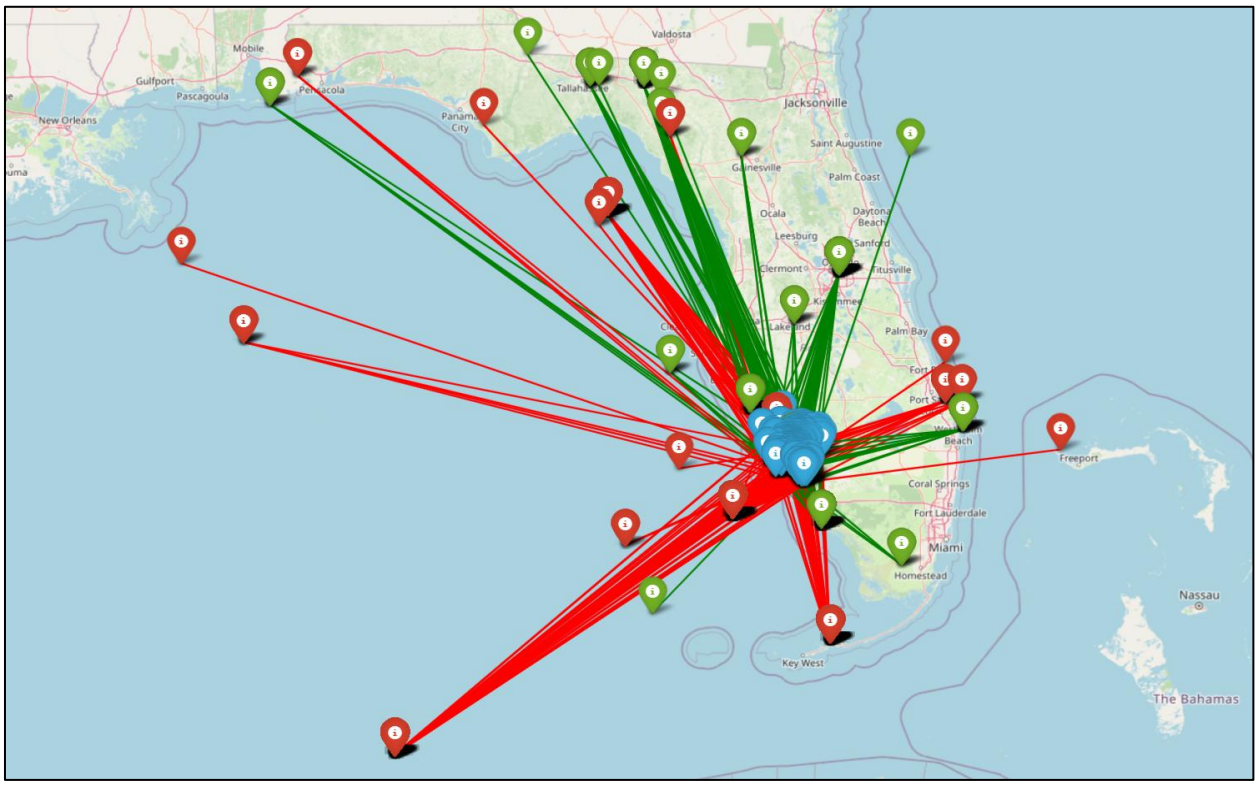

Figura 5. Distancia histórico huracanes y asegurados. Fuente: Elaboración Propia.

Como se puede comprobar en la figura 5, tenemos hasta un total de tres tipos de geolocalizaciones siendo máxima intensidad del huracán más cercano a Florida designado con un marcador rojo, segunda máxima intesidad del huracán más cercano a Florida establecido con una ubicación verde y con una localización azul los siniestros registrados en el Condado de Lee en Florida. Por otro lado, los vectores rojos y verdes corresponden con las distancias mínimas entre las dos máximas intensidades del huracán con los asegurados.

Por tanto, una vez recopiladas las tres bases de datos se ha procedido a juntarlas en una única. Así se tienen las suficientes variables para poder ser empleadas en el entrenamiento de las redes neuronales artificiales.

4. METODOLOGÍA

El objetivo del presente trabajo es capturar patrones de comportamiento complejos que son indetectables por otros métodos estadísticos excepto por la inteligencia artificial, es decir, por las redes neuronales artificiales *feedforward* la cual predecirá las indemnizaciones individuales de cada asegurado en el último huracán registro en Florida llamado Milton. Sin embargo, cabe destacar que se ha aplicado una metodología innovadora y diferente a lo comúnmente establecido en el entrenamiento de estos modelos. El factor principal diferenciador viene determinado por la creación de dos funciones de activación personalizadas que serán introducidas como parte de la arquitectura de la red neuronal de predicción de indemnizaciones ante huracanes en el Condado de Lee. Además de las funciones de activación únicas para la construcción y entrenamiento del modelo, se ha utilizado los procedimientos estándares de redes neuronales *feedforward*.

Por otra parte, en la arquitectura y entrenamiento de la red neuronal se ha utilizada el ecosistema desarrollado por Google llamado TensorFlow y, concretamente, la API Keras la cual utiliza exclusivamente Python. A pesar de la completitud de la librería sostiene un inconveniente importante, es decir, la única limitación que presenta es la baja velocidad computacional que ofrece a la hora de entrenar redes neuronales lo cual limita encarecidamente el número de modelos posibles a probar.

4.1 Hiperparámetros del modelo.

Siempre que estemos seleccionando qué hiperparámetros aplicar debemos de reducir todo en dos casuísticas: problemas de regresión y problemas probabilísticos. El objetivo del presente trabajo es predecir la indemnización final de cada asegurado y, por tanto, los únicos hiperparámetros que utilizaremos serán los relacionados con problemas de regresión. Una vez identificado el tipo de modelo a utilizar debemos de seleccionar hasta un total de 14 familias de hiperparámetros o configuraciones que hacen infinitas las combinaciones. Las familias de hiperparámetros corresponden a: preprocesadores de datos, optimizadores, funciones de activación, funciones de pérdida, ratio de aprendizaje, tasa de abandono, coeficiente de penalización, pesos iniciales entre las conexiones neuronales, pesos iniciales del sesgo, número de capas ocultas, número de neuronas en cada capa, métricas a monitorizar, número de épocas – actualizaciones de los pesos del modelo, cantidad de muestras en los que se divide el entrenamiento para actualizar los pesos y configuración del *Early stopping*.

4.1.1 Preprocesamiento de los datos.

Transformar los datos es una de las partes más cruciales en la construcción de una red neuronal ya que es parcialmente responsable del aprendizaje correcto del modelo. En un primer momento, se pueden emplear individualmente cuatro transformaciones comunes: escala mínimo - máximo en donde se ajustan los datos a un rango 0 – 1, normalizar los datos, escalar robustamente los datos para tratar mejor los atípicos y reducir la dimensión de los datos con técnicas como el Análisis de Componentes Principales.

De entre las cuatro técnicas de preprocesamiento utilizadas en la presente red neuronal, la única capaz de aprender los datos de una forma significativa fue a través de la estandarización. Por tanto, se ha concluido que las variables independientes empleadas deben de ser estandarizadas. Por último, se han dividido los datos en dos partes: 80 % datos de entrenamiento y 20 % datos de validación, además de utilizar los datos procedentes del huracán Milton como datos de evaluación. Por último, se ha estructurado la base de datos de la forma más eficiente posible, es decir, a través de tensores con el único objetivo de aumentar significativamente la velocidad de aprendizaje del modelo.

4.1.2 Funciones de pérdida.

La elección de esta familia de hiperparámetros viene determinada si estamos ante un problema de regresión o de clasificación. Utilizaremos aquellas funciones de pérdida de regresión que hayan demostrado, en pruebas previas con la actual base, que aumenta significativamente la capacidad de aprendizaje del modelo siendo estos el Error Cuadrático Medio, Error Absoluto Medio y la función pérdida Log-Cosh. A continuación, en la figura 6 se expresan las distintas fórmulas de cada función de pérdida a utilizar:

$MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y})^2$ donde y = valor real, $\hat{y}$ = valor predicho, n = número de datos

$MAE = \frac{1}{n} \times \sum_{i=1}^{n} |y_i - \hat{y}|^2$ donde y = valor real, $\hat{y}$ = valor predicho, n = número de datos

$\text{Log Cosh} = \sum_{i=1}^{n} \log\left(\frac{(e^{(y_i - \hat{y})} + e^{-(y_i - \hat{y})})}{2}\right)$ donde y = valor real, $\hat{y}$ = valor predicho, n = número de datos

Figura 6. Funciones de pérdida a utlizar como hiperparámetros. Fuente: Elaboración Propia

Esta familia de hiperparámetros es una de las más importantes en la arquitectura de las redes neuronales por dos razones: es la única métrica empleada por el modelo para minimizar el error de predicción además de ser la principal responsable de reducir ese error a través de la información que proporciona a los optimizadores. Como se ha mencionado anteriormente, a pesar de existir otras funciones de pérdida se ha comprobado en pruebas iniciales que las demás no producen aprendizaje alguno.

4.1.3 Optimizadores.

El pilar fundamental de las redes neuronales se encuentra en el algoritmo de retropropagación y su optimizador correspondiente. Los optimizadores son los únicos responsables de que el modelo aprenda a través de la actualización de los pesos vinculados a las neuronas que a su vez afectan a la magnitud numérica que se introduce en las funciones de activación, siendo esta actualización de los pesos por medio de los gradientes. Por tanto, el optimizador se encarga de estudiar como utilizar los gradientes para posteriormente modificar los pesos de las neuronas para que el modelo prediga correctamente.

Sin embargo, no existen optimizadores predilectos para los problemas de regresión sino que dependen significativamente de la función de activación que se esté utilizando. Es por ello por lo que empleamos siete optimizadores de un total de 13 los cuales funcionan mejor con las funciones de activación que mencionaremos posteriormente. Estos siete optimizadores son: Adam o Estimación Adaptativa de Momentos (Kingma y Ba, 2015), Nadam o Estimación Adaptativa de Momentos con Aceleración de Nesterov (Dozat, 2016), Adamax, RMSProp o Propagación de la Raíz del Cuadrado Medio, Lion o Optimizador de Momento de Signo Evolucionado (Chen et al., 2023), Adafactor o Factorización Adaptativa (Shazeer y Stern, 2018) y SGD o Descenso Estocástico del Gradiente.

4.1.4 Funciones de activación.

Una de las piezas angulares más importantes en cuanto a la construcción de una red neuronal es definir aquellas transformaciones numéricas que surgen dentro de cada neurona, es decir, definir una función de activación. Una de las características más innovadoras del presente trabajo nace de la necesidad de capturar de la manera que más cercana a la realidad las indemnizaciones de los asegurados y, por tanto, se crea una función a trozos específica la cual captura la evolución histórica de las indemnizaciones y sumas aseguradas inferiores a 50.000 dólares que surgen de los huracanes

que han afectado al Condado de Lee en Florida.

Es por ello por lo que, además de las funciones de activación comúnmente utilizadas en problemas de regresión, se haya ajustado una función a trozos que recoge perfectamente el histórico de las indemnizaciones en esta región. La propuesta novedosa para solventar la determinación de las indemnizaciones a través de seguros paramétricos climáticos demuestra cómo las indemnizaciones en situaciones meteorológicas extremas se pueden predecir con un alto grado de confianza. Por tanto, la elección de la función de activación es una parte crucial de la arquitectura de la red neuronal ya que teóricamente solo existe una única combinación de estos hiperparámetros para cada problema.

4.1.4.1 Funciones de activación comunes en problemas de regresión.

A pesar de la existencia de un total de 29 funciones de activación disponibles en Keras se utilizarán estrictamente aquellas relacionadas con problemas de regresión y, teniendo en cuenta las limitaciones computacionales existentes, se probarán únicamente tres. A continuación, se muestra en la tabla 7 las funciones de activación a utilizar: relu, swish (Howard et al., 2019) y lineal.

Lineal $\rightarrow f(x) = x \quad donde\ x = \ solución\ de\ la\ neurona$

Relu $\rightarrow f(x) = \max(x, 0) \quad donde\ x = \ solución\ de\ la\ neurona$

Swish $\rightarrow f(x) = \mathrm{x} * \mathrm{sigmoide(x)} \quad donde\ x = \ solución\ de\ la\ neurona, \mathrm{sigmoide(x)} = \frac{1}{1+e^{-x}}$

Figura 7. Funciones de activación a usar como hiperparámetros.

4.1.4.2 Funciones de activación propias a las indemnizaciones del Condado de Lee.

Tras múltiples modelos implementados sin éxito surgió la idea de capturar de una forma más realista, sin tener que depender de funciones genéricas, las indemnizaciones cuya suma asegurada del continente sea inferior a 50.000 dólares de este territorio. Por tanto, surgió este nuevo concepto en donde las predicciones de las indemnizaciones se pudieran capturar gracias de los datos históricos de indemnizaciones a través de una función de densidad definida por tramos ya que no es posible capturar todo el histórico con una única función de densidad.

Los datos históricos de las indemnizaciones reales que se han utilizado para ajustarles una función de densidad son los recogidos hasta con anterioridad al huracán Milton, es decir, al último huracán registrado en 2024. Del total de 1745 registros, las indemnizaciones se dividen en: 921 incidencias (53 %) cuyo pago neto final es de cero dólares, 34 sucesos (2 %) cuyo pago neto final es de 50.000 dólares y 790 asegurados (45 %) que percibieron entre 0 - 50.000 dólares. Por tanto, los valores indemnizatorios históricos pueden representarse mediante una función a trozos con dos tramos fijos de 0 y 50.000 dólares, y un tercero ajustado a una función polinómica de cuarto grado, basada en las indemnizaciones históricas restantes. A continuación, en la figura 8 se muestra el ajuste de la función polinómica:

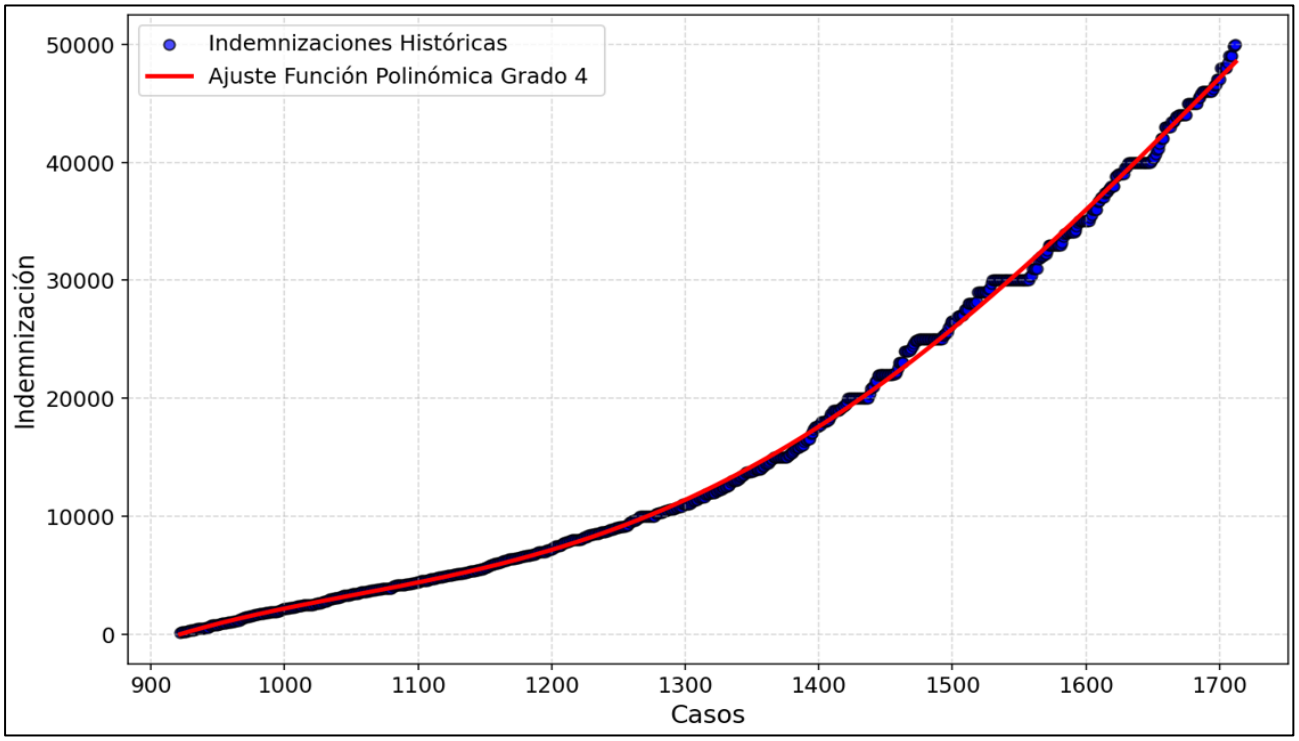

Figura 8. Ajuste de las indemnizaciones históricas a una función polinómica de grado cuatro por Maximum Likelihood de las indemnizaciones del Condado de Lee en Florida.

Una vez identificada la función del tercer tramo se procede, en la figura 9, a mostrar la función de densidad a trozos que se implementará como última función de activación de la red neuronal:

$$f(x) = \begin{cases} 0 & si\ x < 921 \\ -1{,}67 \times 10^{-7} \times x^4 + 9{,}22 \times 10^{-4} \times x^3 - 1{,}80 \times x^2 + 1.522{,}74 \times x - 477.265 & si\ 922 \leq x \leq 1712 \\ 50.000 & si\ x > 1712 \end{cases}$$

Figura 9. Función por tramos de los datos históricos de indemnizaciones del Condado de Lee en Florida.

Sin embargo, esta función al estar acotada se debe implementar posteriormente a otra función de activación propia la cual recoge exclusivamente los números comprendidos entre 1 y 1745. Por consiguiente, se ha modificado e implementado una nueva función de activación la cual recoge de forma equiprobable los números entre 1 y 1745, lo cual sumado a la función de activación polinómica hace que se recoja ese histórico de probabilidades y densidades de las indemnizaciones históricas. A continuación, la figura 10 se muestra la penúltima función de activación utilizada en la arquitectura de la red neuronal:

$$f(x) = \min(\max(1 + 1744 \times sigmoide(x^2), 1), 1744) \ \ donde\ x = solución\ neurona, \ \ sigmoide(x) = \frac{1}{1 + e^{-x}}$$

Figura 10. Función Sigmoide modificada para capturar las indemnizaciones por probabilidades.

Por tanto, el objetivo de crear dos funciones propias es el siguiente: en un primer momento capturar de forma equiprobable los recuentos históricos entre 1 y 1745 y por último transformarlo teniendo en cuenta las probabilidades y densidades históricas de las indemnizaciones del Condado de Lee cuyas sumas aseguradas del continente sean inferiores a 50.000 dólares.

4.1.5 Pesos iniciales de las conexiones y de los sesgos.

El segundo pilar fundamental sobre la construcción de redes neuronales artificiales *feedforward* se basa en qué pesos iniciales se atribuyen a cada conexión neuronal antes de entrenar el modelo. Además, se debe de tener en cuenta que la modificación de los pesos por los gradientes por parte de los optimizadores viene determinada por el tamaño y signo asignado inicialmente a estos por lo que es de suma importancia entender que pesos atribuir a la red completa.

En consecuencia, la selección del mejor hiperparámetro de la familia de los pesos iniciales se hará probando todos y cada uno de los inicializadores de *Kernel* o pesos que no necesiten ninguna modificación por parte del investigador como definir los parámetros de una distribución Normal sino aquellos cuya fórmula sea cerrada. Por tanto, probaremos los hiperparámetros Glorot Normal y Uniforme (Glorot y Bengio, 2010), He Normal y Uniforme (He et al., 2015) y Lecum Normal y Uniforme (Klambauer et al., 2017). Cabe destacar que para optimizar la velocidad computacional

solo se aplicarán estos pesos a las conexiones entre las neuronas y no a los pesos de los sesgos los cual se les dejará con peso cero.

Por tanto, los inicializadores de pesos entre las conexiones seguirán una distribución normal o uniforme cuyos parámetros vienen definidos por una fórmula preestablecida la cual siempre dependerá del número de conexiones que entren y salgan en una misma neurona. Es por ello por lo que los pesos se asignarán siempre de forma aleatoria según los parámetros y distribución seleccionados. A continuación, en la figura 11 se muestra las fórmulas que se aplican para la asignación de pesos:

$$Glort\ Uniforme \sim U\left(-\sqrt{\frac{6}{x+y}}, \sqrt{\frac{6}{x+y}}\right)\ donde\ x = conexiones\ de\ entrada, y = conexiones\ de\ salida$$

$$Glort\ Normal \sim N\left(0, \frac{2}{x+y}\right)\ donde\ x = conexiones\ de\ entrada, y = conexiones\ de\ salida$$

$$He\ Uniforme \sim U\left(-\sqrt{\frac{6}{x}}, \sqrt{\frac{6}{x}}\right)\ donde\ x = conexiones\ de\ entrada$$

$$He\ Normal \sim N\left(0, \frac{2}{x}\right)\ donde\ x = conexiones\ de\ entrada$$

$$Lecum\ Uniforme \sim U\left(-\sqrt{\frac{3}{x}}, \sqrt{\frac{3}{x}}\right)\ donde\ x = conexiones\ de\ entrada$$

$$Lecum\ Normal \sim N\left(0, \frac{1}{x}\right)\ donde\ x = conexiones\ de\ entrada$$

Figura 11. Fórmulas Inicializadores de pesos. Fuente: Adaptado de Goodfellow et al. (2016)

4.1.6 Ratio de aprendizaje, tasa de abandono y coeficiente de penalización.

Otros aspectos a tener en cuenta a la hora de construir un modelo es el ratio de aprendizaje que tendrá el modelo y de qué manera se puede penalizar al modelo para evitar sobreajustes. El ratio de aprendizaje que se utilizará, el cual determina la magnitud de cambio en los pesos, vendrá determinado por una distribución uniforme cuyos valores estarán comprendidos entre 0.0001 y 0.1

inicialmente; A pesar de existir funciones creadas en Keras para optimizar de cierto modo la búsqueda del mejor ratio de aprendizaje se ha decidido actuar de esta manera específica ya que la velocidad computacional es mucho mayor.

Por otro lado, al tratarse de un modelo no muy complejo, se tiene que decidir entre si aplicar una tasa de abandono o un coeficiente de penalización. En el presente trabajo se ha decidido utilizar únicamente la tasa de abandono la cual de forma aleatoria desconecta un tanto por ciento de los pesos totales para así evitar el sobreajuste. Sin embargo, para no penalizar demasiado el sobreajuste no se utilizará el coeficiente de penalización el cual castiga aquellos pesos que destacan reduciendo su magnitud.

4.1.7 Cantidad de capas ocultas y neuronas.

Al establecer el número de capas ocultas, neuronas y épocas a utilizar siempre se ha tenido en cuenta un aspecto crucial: velocidad computacional. Por tanto, a pesar de que cuanto mayor es el modelo mejor se interpretan los datos y mejores predicciones dará se ha decidido crear un modelo reducido por las limitaciones mencionadas anteriormente. La configuración del número de capas ocultas, neuronas y épocas vienen influenciadas unas por otras.

La profundidad que contempla la arquitectura de la presente red neuronal *feedforward* está definida por seis capas ocultas: las tres primeras corresponden a las funciones de activación predefinidas de Keras con un rango de neuronas comprendidas entre 2 y 64 cada una, una cuarta capa con una única neurona cuya función de activación es lineal para que así se aplique mejor los valores introducidos a la quinta capa de una única neurona cuya función de activación da los valores comprendidos entre 1 y 1745, siendo una de las dos funciones de activación propias al Condado de Lee. Por último, la sexta capa con una única neurona corresponde a la función polinómica de grado cuatro cuya solución determinará la predicción de la indemnización. A continuación, en la figura 12 se muestra una simulación de la arquitectura del modelo:

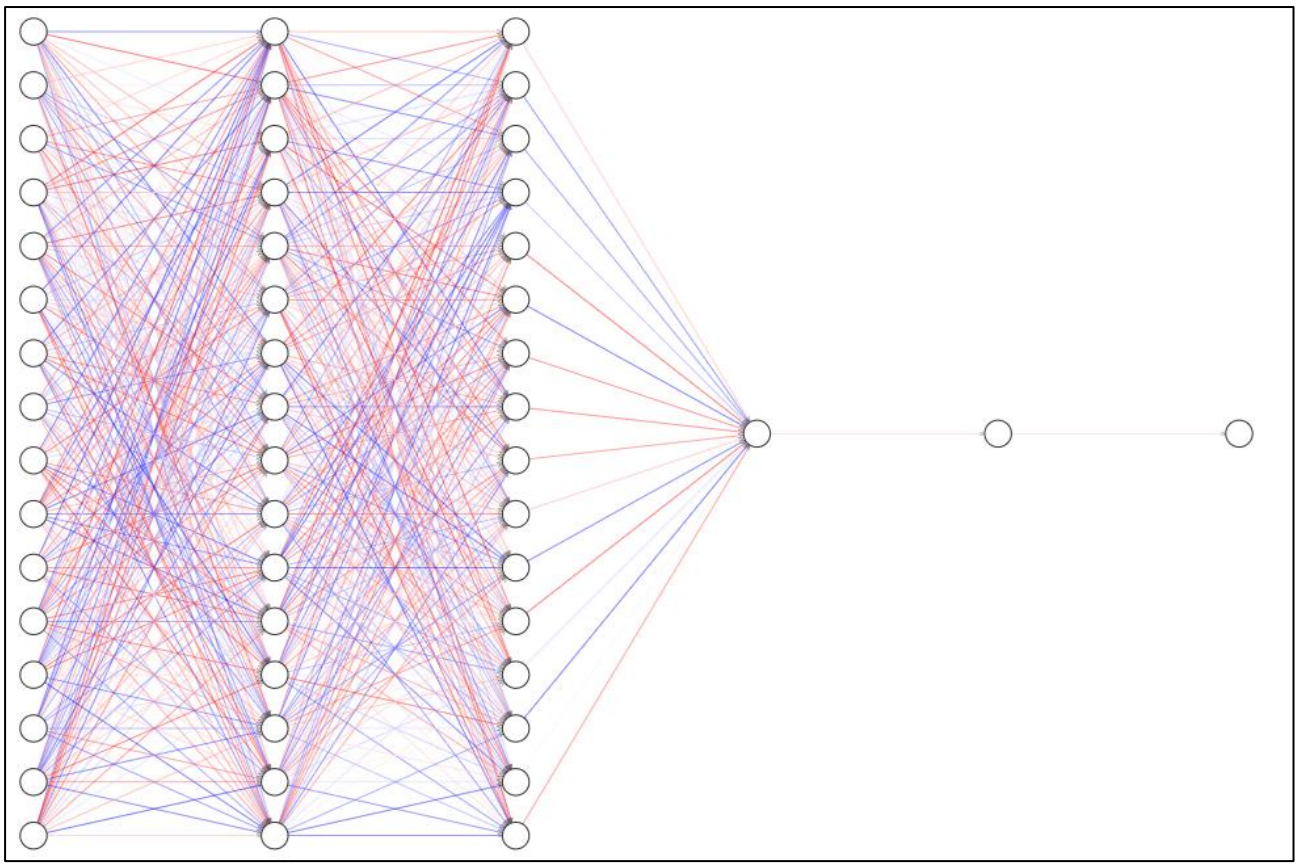

Figura 12. Simulación arquitectura de la red neuronal artificial *feedforward* a usar. Fuente. Elaboracion Propia.

4.1.8 Épocas, tamaño del *batch* y *Early Stopping*.

Por último, nos queda por definir tres familias de hiperparámetros importantes. Por un lado, debemos de determinar el número de veces que se actualizarán los pesos a través de las épocas y el tamaño de los datos de entrenamiento, que influye directamente en la actualización de los pesos por el valor que se le da a la función de pérdida derivado del tamaño de los datos. Por otro lado, determinar cuando se debe de parar el entrenamiento de un modelo si se cumplen ciertos criterios, como por ejemplo la estabilización del error cuadrático medio, a través del llamado *Early Stopping*.

El tamaño de los datos de entrenamiento se ha establecido para maximizar la velocidad computacional del entrenamiento del modelo y, por tanto, se han seleccionado tres posibles tamaños del *batch* siendo 16, 32 y 64. Además, el número de épocas vendrá influenciado por el *Early Stopping* ya que el modelo dejará de entrenar cuando pasados, inicialmente, 20 actualizaciones de los pesos y no se haya conseguido reducir el error cuadrático medio de los datos de validación.

4.2 Proceso de entrenamiento y evaluación del desempeño del modelo.

4.2.1 Procedimiento obtención mejores hiperparámetros.

La elección de los mejores hiperparámetros es una cuestión muy compleja ya que las posibles combinaciones son literalmente infinitas. Podemos utilizar ciertos algoritmos que nos ayudan a simplificar su búsqueda siendo el predilecto el algoritmo *Hyperband* como ya se ha mencionado en el presente trabajo y, sin embargo, debemos de evitar su uso ya que su velocidad computacional es muy limitada. Es por ello por lo que utilizaremos el algoritmo de *RandomSearch* sin usar el que está predefinido en Keras sino utilizando la librería de Numpy para seleccionar de forma aleatorio y equiprobable todos los casos posibles de las familias de hiperparámetros seleccionados.

Una vez establecido la forma en la que se seleccionará los hiperparámetros, a continuación se explicará un nuevo algoritmo que se utilizará en el presente trabajo. El algoritmo en cuestión contará de cuatro fases las cuales en cada una se probará un total de 5.000 combinaciones aleatorias de hiperparámetros. En la primera ronda se seleccionarán los mejores cuatro modelos de un total de 5.000, con sus hiperparámetros correspondientes, según unas métricas de evaluación definidas en el siguiente apartado. En la segunda ronda se probarán un total de 5.000 modelos y todas las familias de hiperparámetros se incluirán de la misma forma excepto las funciones de pérdida, optimizadores, funciones de activación predefinidas y pesos iniciales de las conexiones entre neuronas las cuales solo se seleccionarán aquellas que hayan quedado entre los cuatro mejores modelos; además solo los mejores tres modelos serán los que pasarán a la siguiente ronda. En la tercera ronda se aplicará el mismo procedimiento que en la segunda ronda excepto que solo pasarán a la cuarta fase los dos mejores modelos. En la cuarta y última ronda, se probarán un total de 10.000 modelos que combinen los mejores hiperparámetros de las mejores redes neuronales de la tercera fase y además se tendrán en cuenta y se acotarán el número de neuronas, tamaño del *batch* y ratio de aprendizaje de los mejores últimos dos modelos.

En consecuencia, se probarán hasta un total de 25.000 combinaciones de hiperparámetros en donde solo nos quedaremos con aquel modelo que haya superado a todos y cada una de las arquitecturas seleccionadas aleatoriamente. Por último, se estudiará a profundidad los mejores hiperparámetros de la mejor arquitectura para así poder intentar acercarnos al mejor modelo de predicción de indemnizaciones en época de huracanes en el Condado de Lee en Florida para sumas aseguradas de continente inferiores a 50.000 dólares.

4.2.2 Métricas a utilizar para definir el mejor modelo.

Una vez definido el algoritmo que vamos a utilizar en la selección de los mejores hiperparámetros, se debe seleccionar las métricas correspondientes que servirán para evaluar la idoneidad del modelo. Se trata de un total de seis métricas las cuales son: MSE, RMSE, MAE, diferencia observado – predicho, R^2 y varianza explicada (EVS). La obtención de estas métricas se obtendrá, una vez entrenado el modelo, a través de la comparación entre los valores reales y los predichos.

Sin embargo, tres serán los indicadores a tener en cuenta a la hora de decidir que modelo a predicho mejor: RMSE, R^2 y diferencia observado – predicho. Por tanto, la mejor predicción vendrá determinada por qué hiperparámetros y arquitectura ha sido la responsable minimizar el RMSE y diferencia observado – predicho además de maximizar el R^2. En consecuencia, nos centraremos en las siguientes métricas ya que existe una relación directa entre la disminución del RMSE y MAE de los datos de validación con la mejor idoneidad del modelo a la hora de predecir además de aplicar otras transformaciones a la hora de predecir las indemnizaciones.

5. RESULTADOS

La acción de búsqueda de los mejores hiperparámetros se ha definido por un total de cuatro fases. A continuación, se mostrarán los resultados pertinentes a cada una de las fases en las cuales se aportará cada vez más información relevante en relación con los modelos que mejores métricas han sacado y, por tanto, los mejores modelos de predicción de las indemnizaciones.

5.1 Fase 1: Exploración integral del espacio hiperparamétrico.

El propósito de la presente fase es acotar significatívamente el campo hiperparamétrico de búsqueda a través de la elección de los cuatro modelos con el mejor rendimiento, cuyos hiperparámetros continuarán a la siguiente fase. Para ello, se han evaluado un total de 5.000 modelos distintos en los que el rendimiento viene determinado por los criterios de máximo R^2 y mínimo MSE.

Por tanto, en una primera instancia se presentarán los cuatro mejores modelos con las características de: número de neuronas, diferencia indemnización total real y predicha, R^2, MSE, RMSE, MAE, y varianza explicada (EVS) además del optimizador, función de activación, función de pérdida, ratio de aprendizaje, pesos inicializadores, *batch* y número de aciertos. A continuación, en la tabla 6 se muestran los mejores modelos:

Tabla 6. Mejores modelos de la fase 1.

	Top 1	Top 2	Top 3	Top 4
Capa 1	12	16	19	16
Capa 2	21	12	23	14
Capa 3	22	10	21	32
Optimizador	RMSprop	Adam	RMSprop	RMSprop
Funcion Activación	Swish	Relu	Elu	Relu
Función Pérdida	Log Cosh	Log Cosh	MSE	MSE
Ratio Aprendizaje	0.044744544	0.0893988	0.0408626	0.02358
Pesos Inicializador	HeUniforme	LecunUniforme	GlorotNormal	HeNormal
Batch	64	64	32	64
Aciertos	6	16	11	2
Diferencia	45,427	2,672	89,603	266,618
R2	0.25196577	0.245373	0.21343	0.204255
EVS	0.2533344	0.245378	0.218756	0.251399
MSE	127,644,804	128,754,409	134,235,396	135,792,409
RMSE	11,298	11,347	11,586	11,653
MAE	9,061	8,457	8,221	10,031

Fuente:Elaboración Propia

En consecuencia, hemos reducido la cantidad de hiperparámetros a probar significativamente por lo que la segunda fase solo utilizaremos aquellos que hayan destacado según hemos las métricas mencionadas anteriormente. Por tanto, se están filtrando las posibles combinaciones en forma de embudo en donde a raíz que se avanza a la siguiente fase reducimos las posibles combinaciones y aumentamos las posibilidades de encontrar aquella arquitectura que predice robustamente las indemnizaciones.

5.2 Fase 2 : Análisis acotado del espacio hiperparamétrico de los modelos con mejor rendimiento.

La finalidad de la fase 2 es acotar significatívamente el campo hiperparamétrico de búsqueda de la fase 1 a través de la elección de los tres modelos con el mejor rendimiento, cuyos hiperparámetros continuarán a la siguiente fase. Para ello, se han evaluado un total de 5.000 modelos distintos en los que el rendimiento viene determinado por los criterios de máximo R^2 y mínimo MSE. A continuación la tabla 7 muestra el conjunto de hiperparámetros que se utilizarán:

Tabla 7. Conjunto hiperparámetros a probar en Fase 2.

	2º Fase Posibles Combinaciones
Capa 1	8 - 32
Capa 2	8 - 32
Capa 3	8 - 32
Optimizador	RMSprop - Adam
Funciones Activación	Relu - Swish - Elu
Funciones Pérdida	Log Cosh - MSE
Ratio Aprendizaje	0.00001 - 0.1
Pesos Inicializador	HeUniforme - HeNormal - GlorotNormal - LecunUniform
Batch	32 - 64

Fuente:Elaboración Propia

Por tanto, y como se ha mencionado anteriormente, se probarán un total de 5.000 configuraciones aleatoriamente las cuales solo nos quedaremos con los mejores tres modelos. Además, todavía no vale la pena profundizar más en los mejores modelos y solo con las métricas vistas anteriormente nos es suficiente. A continuación, en la tabla 8 se muestra los mejores tres modelos y sus métricas correspondientes:

Tabla 8. Mejores modelos de la Fase 2.

	Top 1	Top 2	Top 3
Capa 1	23	16	25
Capa 2	15	13	15
Capa 3	16	21	18
Optimizador	RMSprop	Adam	Adam
Función Activación	Swish	Relu	Elu
Función Pérdida	Log Cosh	MSE	MSE
Ratio Aprendizaje	0.0159397	0.044744544	0.040159571
Pesos Inicializador	HeUniforme	HeUniforme	HeNormal
Batch	64	64	64
Aciertos	7	3	3
Diferencia	157,613	152,590	238,832
R2	0.283512	0.2358141	0.234284302
EVS	0.299987	0.251255987	0.27197
MSE	122,266,094	130,406,122	130,667,761
RMSE	11,057	11,420	11,431
MAE	8,557	9,428	8,354

Fuente:Elaboración Propia

Como se puede comprobar, el conjunto reducido de hiperparámetros afloró un nuevo mejor modelo hasta el momento ya que a pesar de reducir el MEA y RMSE, aumentó en cierta media el R^2y la varianza explicada roza el 30 por ciento. Por tanto, tenemos otra ventana de posibilidades reduciendo aún más nuestra búsqueda de los mejores hiperparámetros quedándonos solo con los tres mejores modelos de las dos primeras fases.

5.3 Fase 3: Estudio focalizado del espacio hiperparamétrico remanente

El objetivo de la fase 3 es acotar significatívamente el campo hiperparamétrico de búsqueda de la fase 2 a través de la elección de los dos modelos con el mejor rendimiento, cuyos hiperparámetros continuarán a la siguiente fase. Para ello, se han evaluado un total de 5.000 modelos distintos en los que el rendimiento viene determinado por los criterios de máximo R^2 y mínimo MSE. A continuación, la tabla 9 muestra los hiperparámetros que se tendrán en cuenta:

Tabla 9. Conjunto hiperparámetros a probar en Fase 3.

	3º Fase Posibles Combinaciones
Capa 1	8-25
Capa 2	8-25
Capa 3	8-25
Optimizador	RMSprop - Adam
Funciones Activación	Relu - Swish
Funciones Pérdida	Log Cosh - MSE
Ratio Aprendizaje	0.00001 - 0.1
Pesos Inicializador	HeUniforme
Batch	64

Fuente:Elaboración Propia

En la presente fase, se seleccionarán un total de dos modelos los cuales se analizarán más detalladamente aportando nueva información sobre su comportamiento y forma de predicción. A continuación, se muestra la tabla 10 la cual exhibe los dos mejores modelos hasta el momento siguiendo los criterios mencionados anteriormente:

Tabla 10. Mejores modelos de la fase 3.

	Top 1	Top 2
Capa 1	20	10
Capa 2	11	22
Capa 3	16	12
Optimizador	Relu	RMSprop
Función Activación	Adam	Swish
Función Pérdida	MSE	Log Cosh
Ratio Aprendizaje	0.089685527	0.0765986
Pesos Inicializador	HeUniforme	HeUniforme
Batch	64	64
Aciertos	6	8
Diferencia	59935	225483
R2	0.3152	0.258114
EVS	0.3175	0.291833
MSE	116856100	126585001
RMSE	10810	11251
MAE	8702	8401

Fuente:Elaboración Propia

Por tanto, después de 15.000 modelos probados hemos llegado a la conclusión que la mejor familia de hiperparámetros en relación con la predicción de las indemnizaciones en el huracán Milton son los llevados a cabo por un optimizador Relu, una función de pérdida de error cuadrático medio, un inicializador de pesos con He Uniforme y el tamaño del *batch* de 64 datos. Para poder hacer un análisis más exhaustivo a los dos mejores modelos, hemos recopilado información sobre cómo ha evolucionado el entrenamiento del modelo además de las predicciones y errores hechos por la red neuronal artificial. A continuación, las figuras 13, 14, 15 y 16 muestran como se ha desarrollado el segundo mejor modelo:

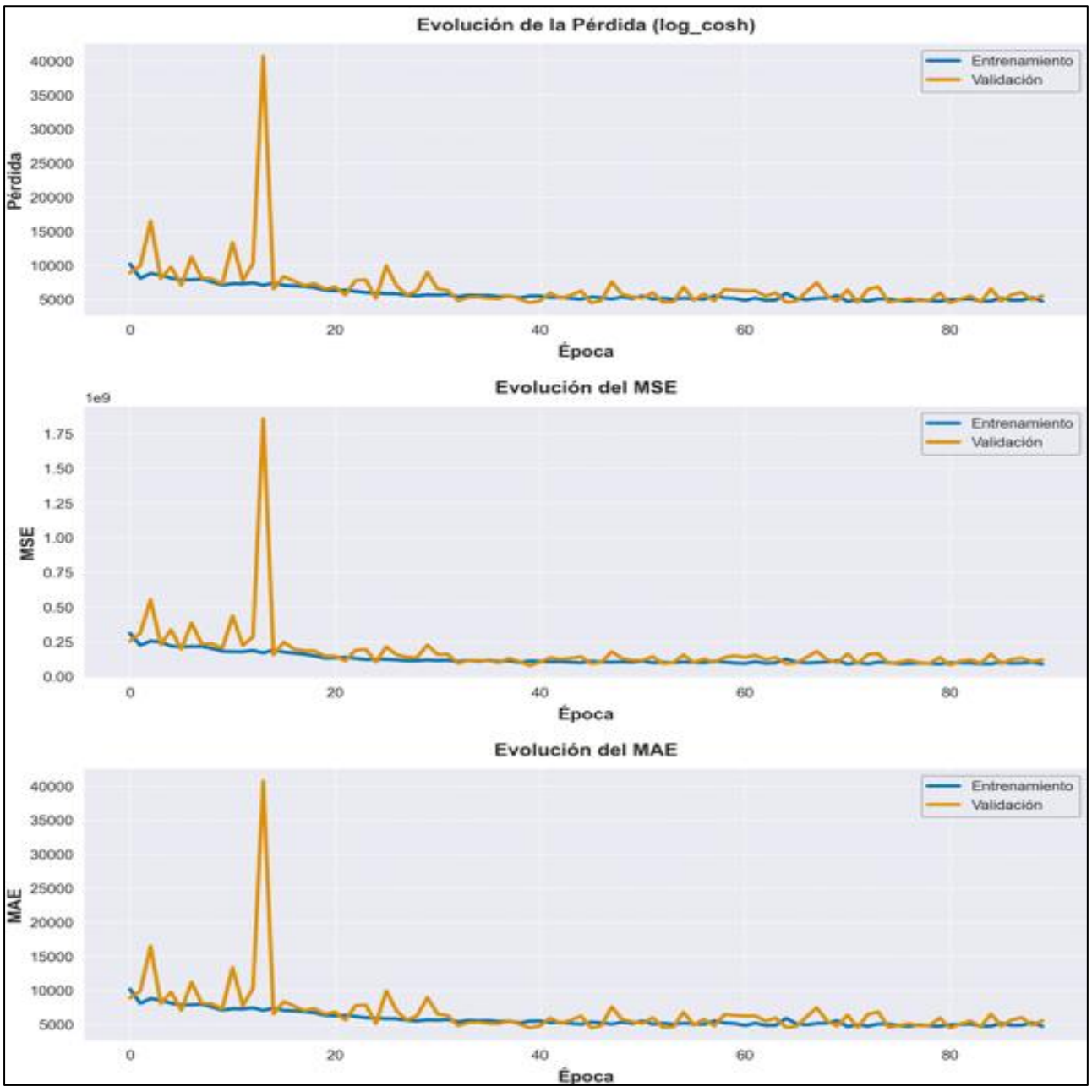

Figura 13. Seguimiento de la función de pérdida y métricas de evaluación del error por época durante el entrenamiento y validación del segundo mejor modelo de la fase 3.

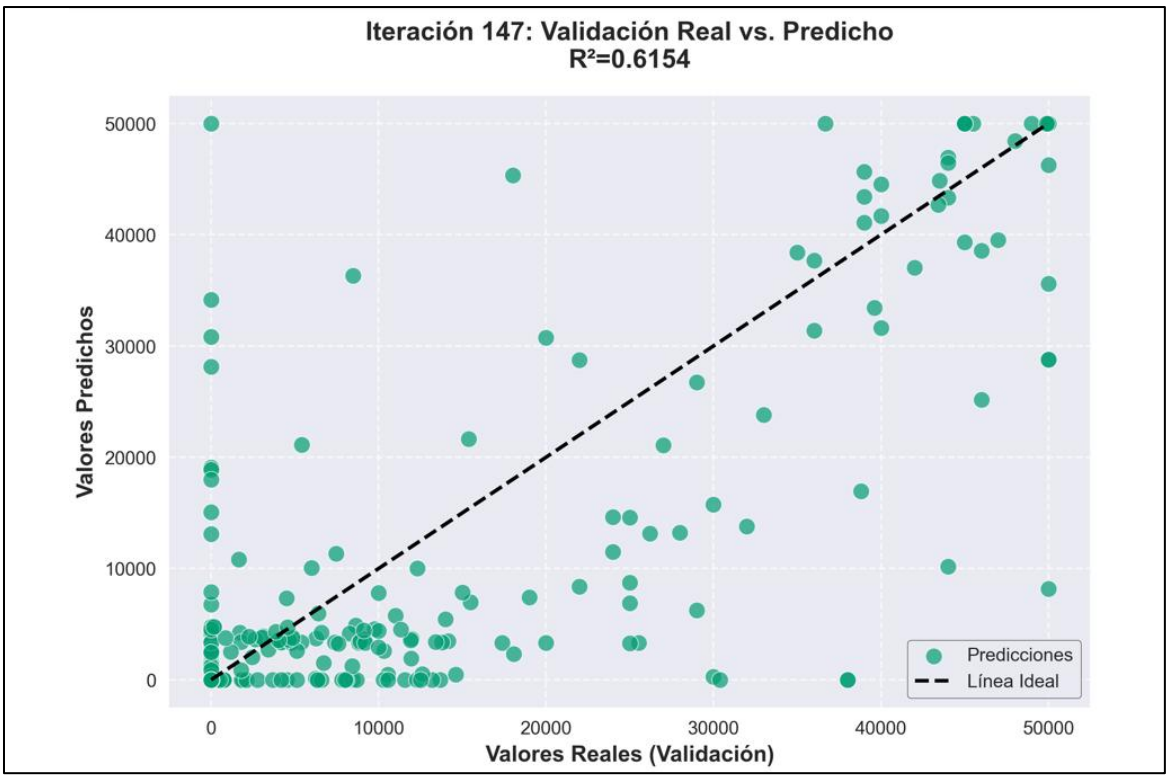

Figura 14. Valores reales vs predichos sobre los datos de validación del segundo mejor modelo de la fase 3.

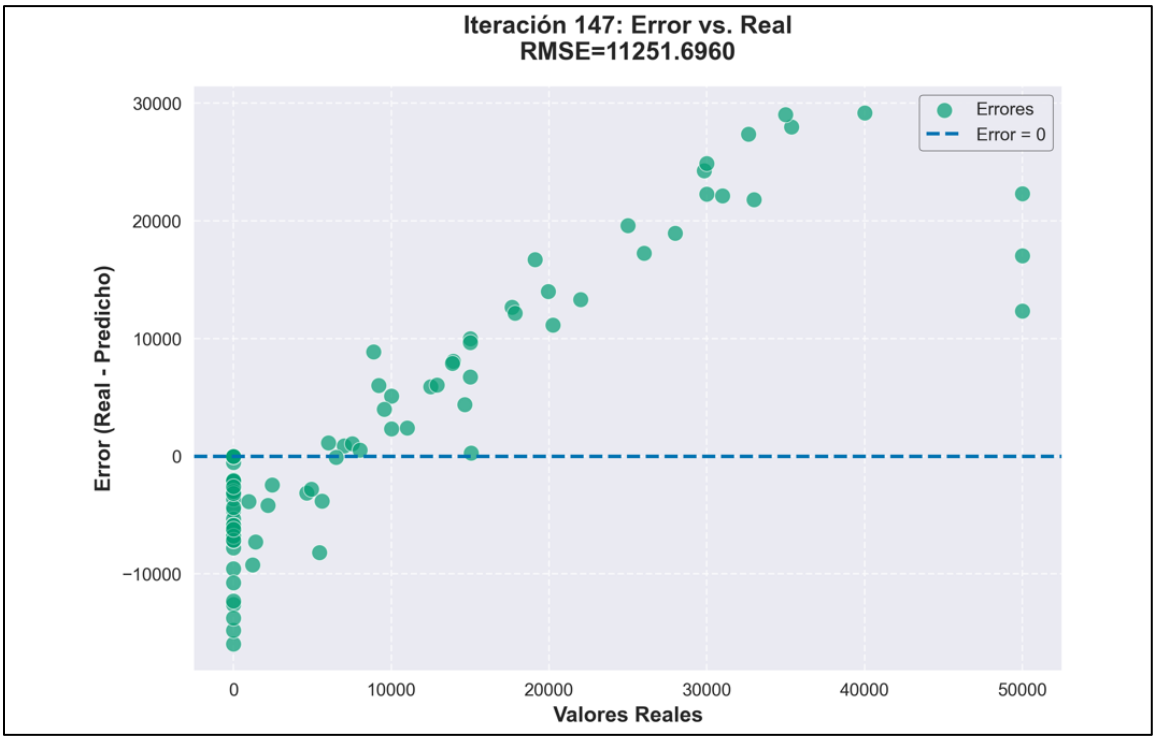

Figura 15. Error predicción vs real de las indemnizaciones en el huracán Milton del segundo mejor modelo de la fase 3.

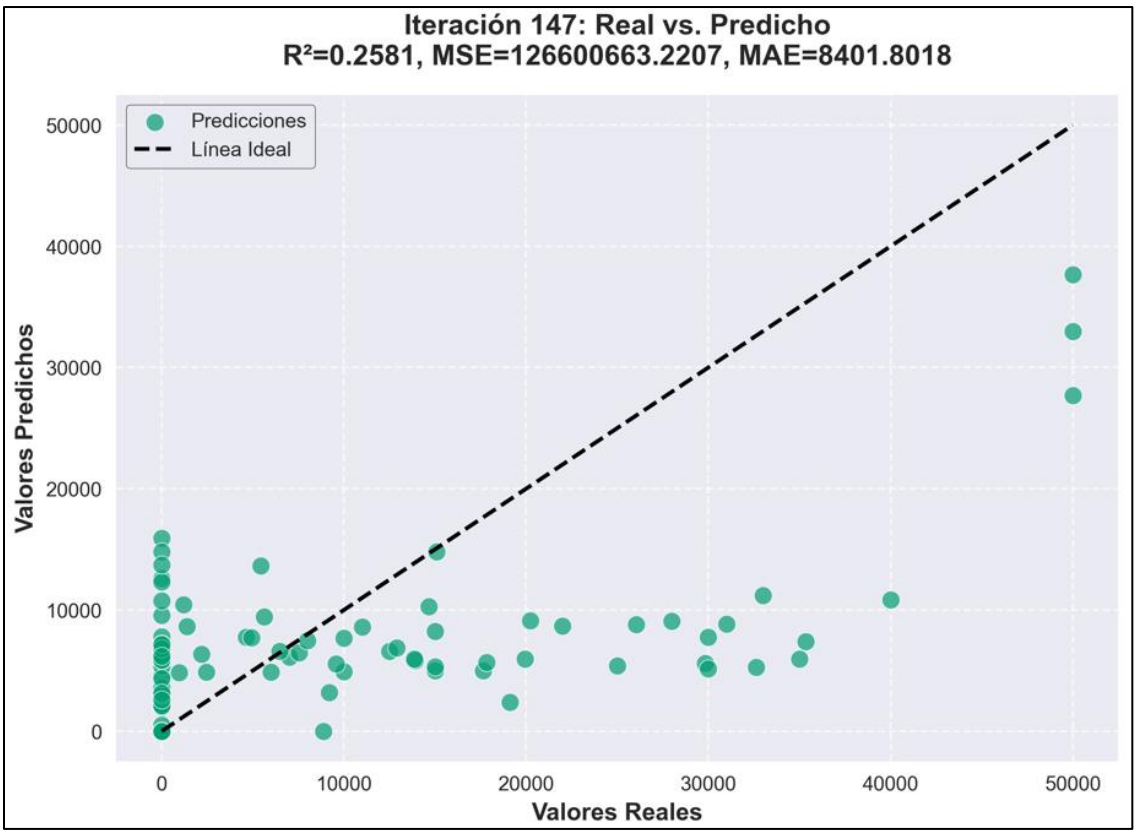

Figura 16. Predicción vs real de las indemnizaciones en el huracán Milton del segundo mejor modelo de la fase 3.

Una vez expuesto los gráficos relacionados con la arquitectura dos, podemos dirimir las siguientes conclusiones: por un lado, al cabo de 40 actualizaciones de los pesos el modelo ha dejado de reducir las métricas MSE y MAE lo cual aparentemente demuestra que rápidamente el modelo deja de aprender sobre sus propios datos. Por otro lado, al aplicarlo a los datos de validación se muestra un R^2 de 0.6154 demostrando una explicación moderada de los datos excepto que muestra un problema notorio en las predicciones: no sabe predecir bien cuando las indemnizaciones son cero. Por último, observamos que con las incidencias del huracán Milton con R^2 igual a 0.2581 se ha reducido la predicción cuando no hay indemnizaciones, pero el modelo obtenido no se puede considerar que haga buenas predicciones.

A continuación, las figuras 17, 18, 19 y 20 muestran como se ha desarrollado el mejor modelo después de las tres fases:

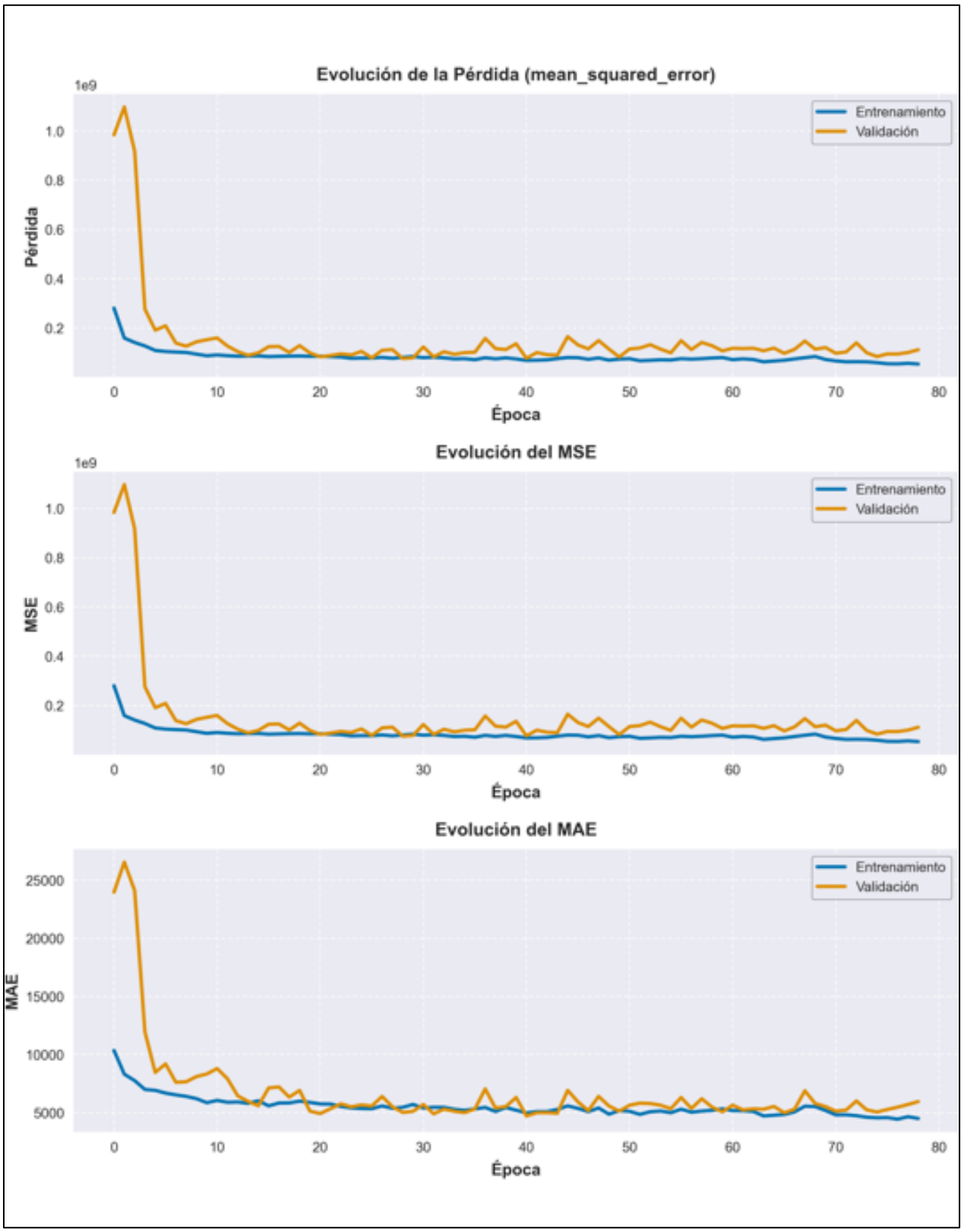

Figura 17. Seguimiento de la función de pérdida y métricas de evaluación del error por época durante el entrenamiento y validación del mejor modelo de la fase 3.

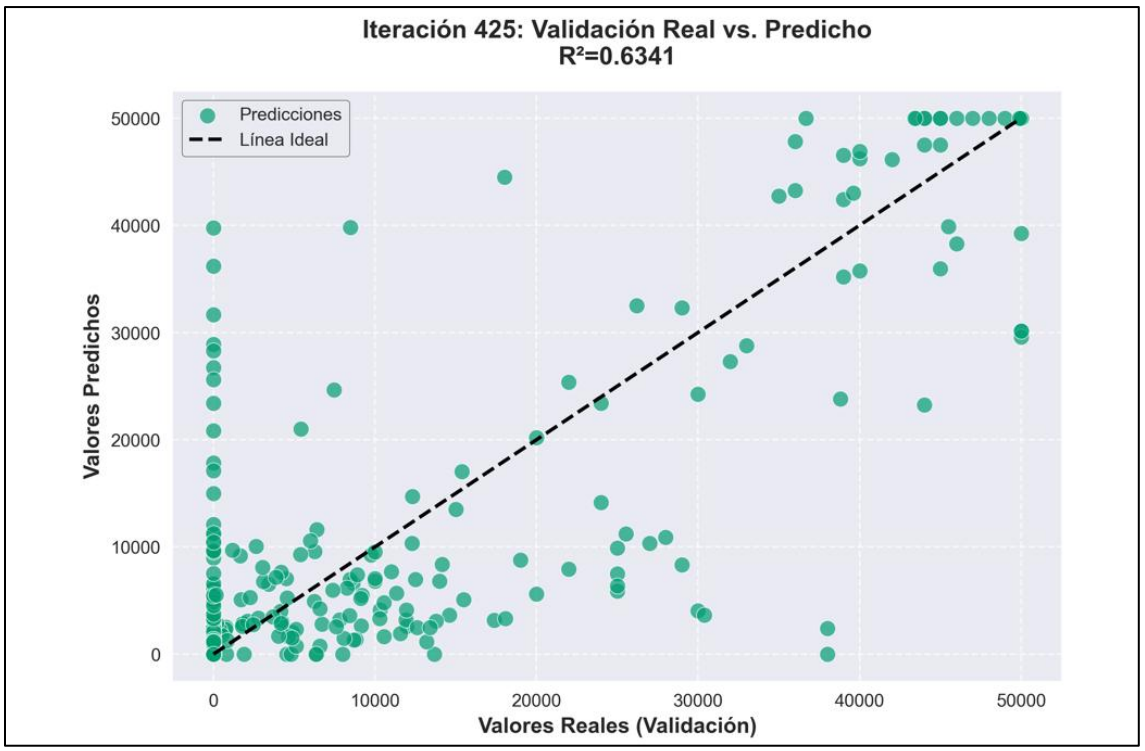

Figura 18. Valores reales vs predichos sobre los datos de validación del mejor modelo de la fase 3.

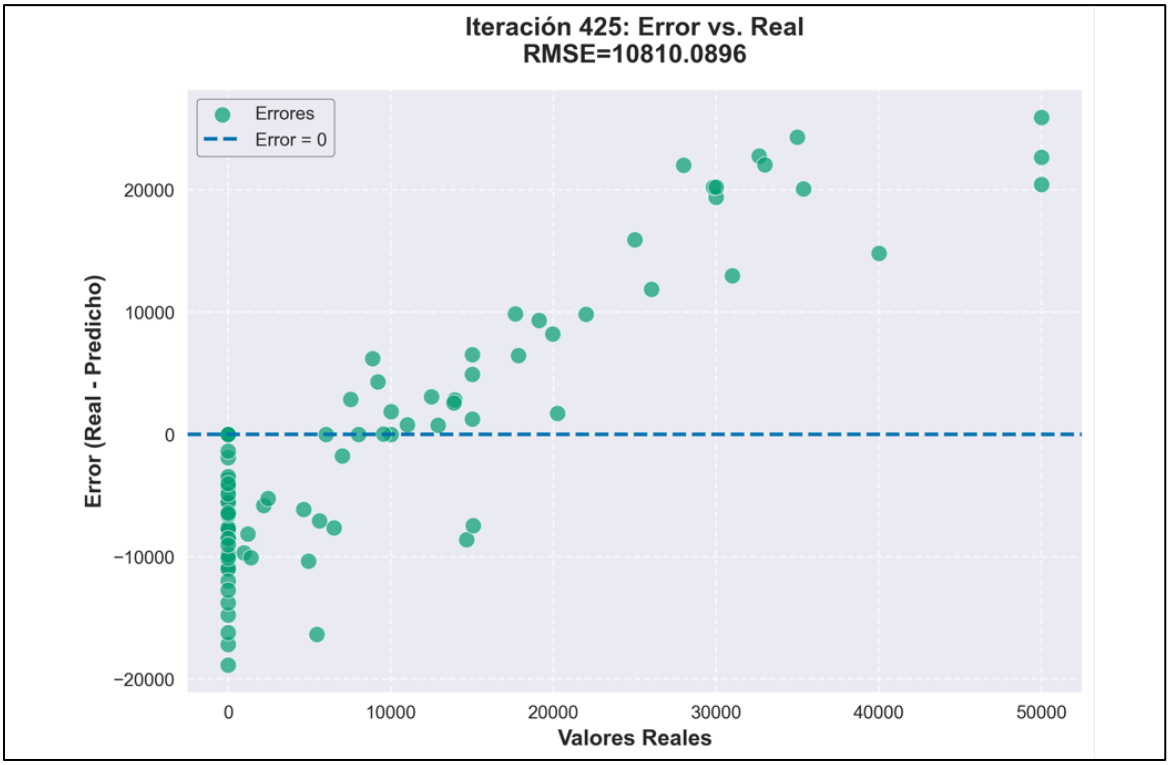

Figura 19. Error predicción vs real de las indemnizaciones en el huracán Milton del mejor modelo de la fase 3.

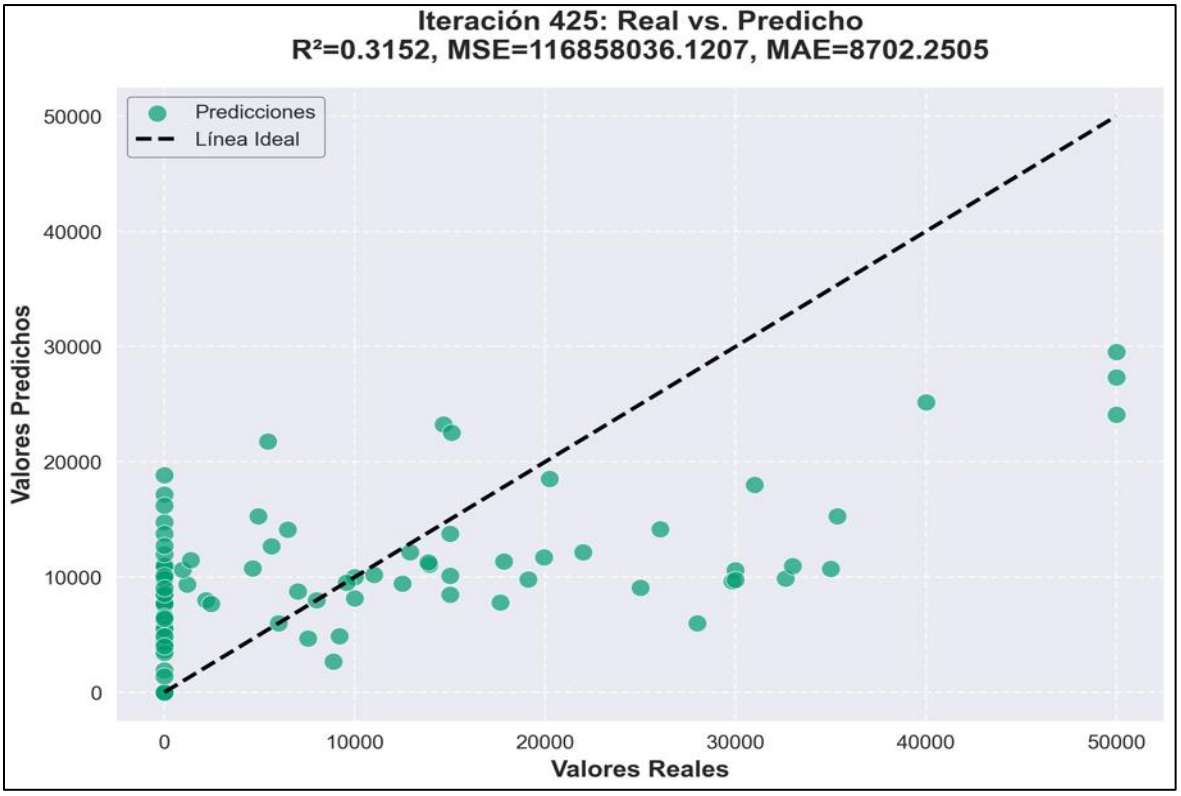

Figura 20. Predicción vs real de las indemnizaciones en el huracán Milton del mejor modelo de la fase 3.

Una vez presentado los gráficos relacionados con la arquitectura del mejor modelo, podemos expresar las siguientes conclusiones: la capacidad de aprendizaje del modelo es similar al visto anteriormente excepto que el R^2 de los datos de validación aumentó en 0.02 hasta los 0.6341 lo cual aumenta ínfimamente la capacidad de explicar la varianza. Sin embargo, aparentemente ese diminuto aumento del R^2 de validación hace que el R^2 del huracán Milton aumente un 20 % hasta los 0.3152 y además disminuye el RMSE y MAE haciendo que las tres métricas sean las mejores de todas las arquitecturas probadas.

No obstante, se puede observar cual es el verdadero problema de todos los modelos que hemos estado probando y es que no predicen bien tanto las mayores indemnizaciones como las indemnizaciones cero las cuales provocan que esa explicación moderada en este fenómeno tan complejo. Es por ello por lo que se debe de plantear algún método por el cual se ajuste los datos predichos a la función de densidad histórica definida por tramos vista anteriormente.

5.4 Fase 4: Examen exhaustivo del conjunto óptimo de hiperparámetros.

La función de la cuarta y última fase es, una vez limitado los hiperparámetros al mejor modelo obtenido hasta el momento, obtener la arquitectura óptima para la predicción de las indemnizaciones teniendo en cuanta los criterios de máximo R^2 y mínimo MSE. Sin embargo, se ha decido implementar un nuevo método con el objetivo de reducir los errores que provocan los casos no indemnizatorios. Es decir, a través del conteo de los siniestros históricos con indemnización nula se reemplazarán en las predicciones con menores valores por un cero siguiendo siempre la cantidad de indemnizaciones nulas históricas por cada suma asegurada.

Además, se aplicará la nueva metodología a un total de 10.000 modelos los cuales se dividirán en 5.000 cuyos hiperparámetros quedan fijos, teniendo en cuenta el mejor modelo hasta ahora, cambiando solo los valores numéricos de los pesos y, por otro lado, 5.000 modelos cuyos hiperparámetros quedan fijos excepto que el número de neuronas en cada capa además del ratio de aprendizaje trabajarán en un rango reducido como se puede comprobar en la tabla 16. A continuación, en las tablas 11, 12 y 13 se recogen todas las incidencias históricas sin contabilizar el huracán Milton:

Tabla 11. Datos históricos huracanes excepto Milton sobre las indemnizaciones nulas de sumas aseguradas entre 30.000 y 50.000 dólares.

Histórico Huracanes			
Suma Asegurada	Casos Totales	Indemnizacion Cero	Probabilidad No Indemnización
50,000	131	57	43.51%
49,000	23	10	43.48%
48,000	19	9	47.37%
47,000	11	3	27.27%
46,000	17	6	35.29%
45,000	22	8	36.36%
44,000	28	8	28.57%
43,000	16	4	25.00%
42,000	12	6	50.00%
41,000	10	8	80.00%
40,000	71	32	45.07%
39,000	15	6	40.00%
38,000	16	7	43.75%
37,000	13	6	46.15%
36,000	8	2	25.00%
35,000	29	15	51.72%
34,000	6	3	50.00%
33,000	36	20	55.56%
32,000	9	4	44.44%
31,000	21	12	57.14%
30,000	108	46	42.59%

Fuente:Elaboración Propia

Tabla 12. Datos históricos huracanes excepto Milton sobre las indemnizaciones nulas de sumas aseguradas entre 29.000 y 10.000 dólares.

Histórico Huracanes			
Suma Asegurada	Casos Totales	Indemnizacion Cero	Probabilidad No Indemnización
29,000	12	4	33.33%
28,000	20	10	50.00%
27,000	16	8	50.00%
26,000	7	4	57.14%
25,000	69	27	39.13%
24,000	16	8	50.00%
23,000	8	2	25.00%
22,000	36	16	44.44%
21,000	11	8	72.73%
20,000	93	42	45.16%
19,000	17	9	52.94%
18,000	13	6	46.15%
17,000	11	8	72.73%
16,000	8	6	75.00%
15,000	57	21	36.84%
14,000	15	11	73.33%
13,000	11	7	63.64%
12,000	7	4	57.14%
11,000	27	20	74.07%
10,000	55	35	63.64%

Fuente:Elaboración Propia

Tabla 13. Datos históricos huracanes excepto Milton sobre las indemnizaciones nulas de sumas aseguradas entre 9.000 y 1.000 dólares.

Histórico Huracanes			
Suma Asegurada	Casos Totales	Indemnizacion Cero	Probabilidad No Indemnización
9,000	12	10	83.33%
8,000	11	6	54.55%
7,000	12	10	83.33%
6,000	18	14	77.78%
5,000	34	31	91.18%
4,000	2	2	100.00%
3,000	3	3	100.00%
2,000	3	3	100.00%
1,000	1	1	100.00%

Fuente:Elaboración Propia

Posteriormente se decide multiplicar las probabilidades de no indemnización a cada total de casos de cada suma asegurada y, redondeándolo a la cifra entera más cercana, obtenemos el número de asegurados que se le asignará un cero por indemnización y los cuales se seleccionarán por su menor indemnización de predicción en el modelo de redes neuronales artificiales seleccionada. Además, a pesar de que el método tiene limitaciones es una buena forma de filtrar aquellos casos que son difíciles de detectar por la red neuronal y así mejorar la predicción. Seguidamente se muestra en las tablas 14, 15 y 16 a cuantos se le implementará la indemnización cero por cada suma asegurada en el huracán Milton:

Tabla 14. Determinación casos con indemnización nula en base a las sumas aseguradas entre 30.000 y 50.000 dólares.

Incidencias Huracán Milton		
Suma asegurada	Total	Casos con Prob No Milton
50,000	16	7
49,000	3	1
48,000	1	0
47,000	1	0
45,000	2	1
44,000	1	0
43,000	1	0
42,000	1	1
40,000	7	3
38,000	2	1
36,000	2	1
35,000	4	2
34,000	2	1
33,000	3	2
31,000	3	2
30,000	6	3

Fuente:Elaboración Propia

Tabla 15. Determinación casos con indemnización nula en base a las sumas aseguradas entre 28.000 y 4.000 dólares.

Incidencias Huracán Milton		
Suma asegurada	Total	Casos con Prob No Milton
28,000	2	1
27,000	1	1
25,000	6	2
24,000	2	1
22,000	4	2
21,000	2	1
20,000	4	2
19,000	1	1
18,000	1	0
17,000	1	1
16,000	1	1
15,000	5	2
11,000	3	2
10,000	2	1
8,000	2	1
6,000	2	2
5,000	1	1
4,000	1	1

Fuente:Elaboración Propia

Por tanto, de un total de 10.000 combinaciones las cuales la mitad se probarán con los hiperparámetros cercanos al mejor modelo y la otra mitad se probarán exclusivamente los hiperparámetros del mejor modelo a excepción de la aleatoriedad de la asignación de los pesos iniciales, además de reemplazar aquellos casos cuya predicción sean las menores por cero según las tablas 14 y 15. A continuación, en la tabla 16 se muestra las arquitecturas que se van a probar en la última fase:

Tabla 16. Conjunto hiperparámetros a probar en Fase 4.

	5.000 Combinaciones	5.000 Combinaciones
Capa 1	10 - 20	20
Capa 2	10 - 20	11
Capa 3	10 - 20	16
Optimizador	Adam	Adam
Funciones Activacion	Relu	Relu
Funciones Pérdida	MSE	MSE
Ratio Aprendizaje	0.06 - 0.1	0.089685527
Pesos Inicializador	HeUniforme	HeUniforme
Batch	64	64

Fuente:Elaboración Propia

Se probaron un total de 10.000 combinaciones las cuales solo dieron resultados significativos aquellas combinaciones de los hiperparámetros fijos. Además, con el nuevo modelo implementado hemos mejorado todas las métricas utilizadas gracias al menor error de predicción provocado por la selección a través de los datos históricos de aquellos casos cuya indemnización final fue nula. Por tanto, a continuación se muestra en las figuras 21, 22, 23 y 24 como se ha desarrollado el mejor modelo después de 25.000 arquitecturas diferentes probadas:

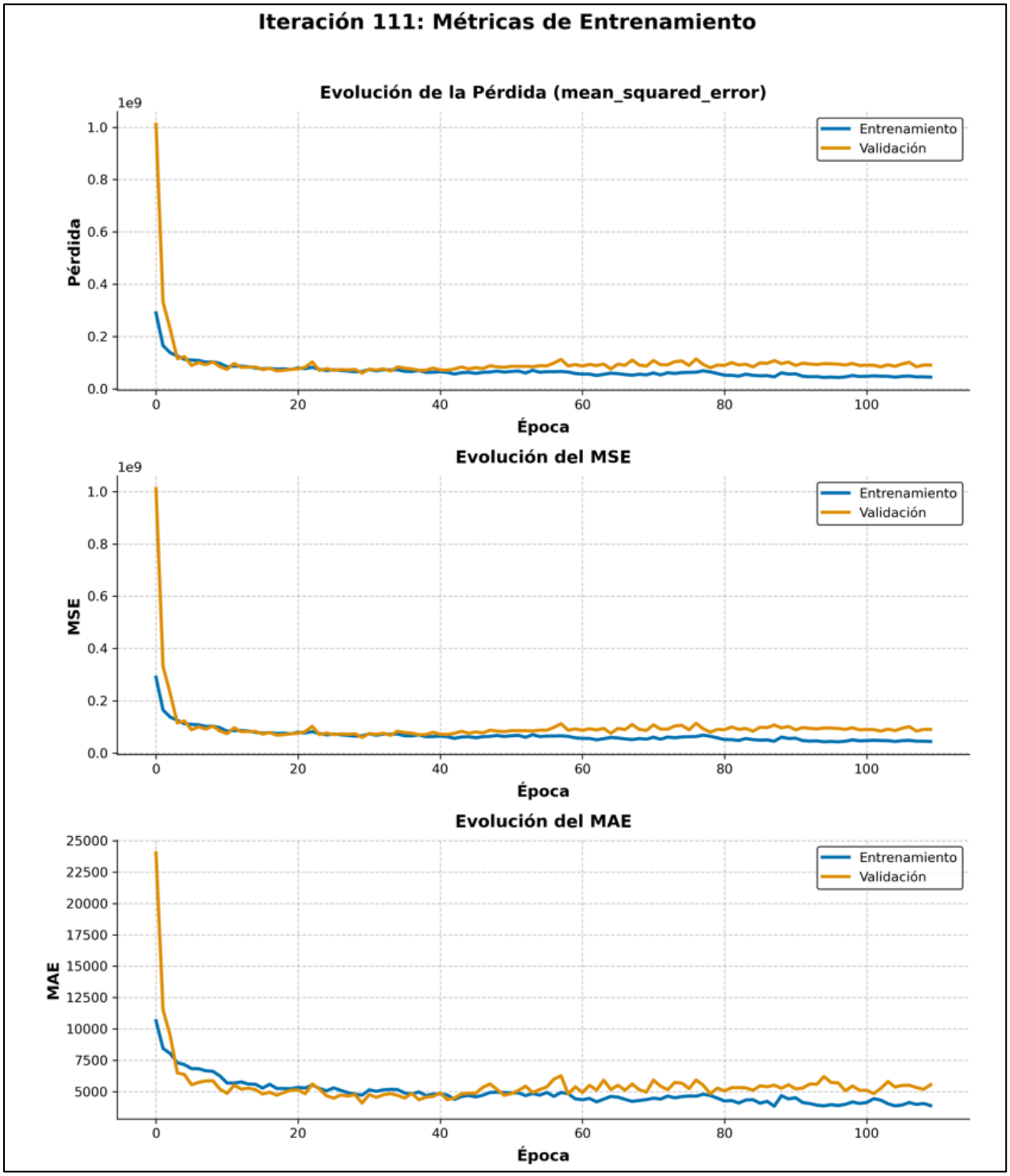

Figura 21. Seguimiento de la función de pérdida y métricas de evaluación del error por época durante el entrenamiento y validación del mejor modelo de la fase 4.

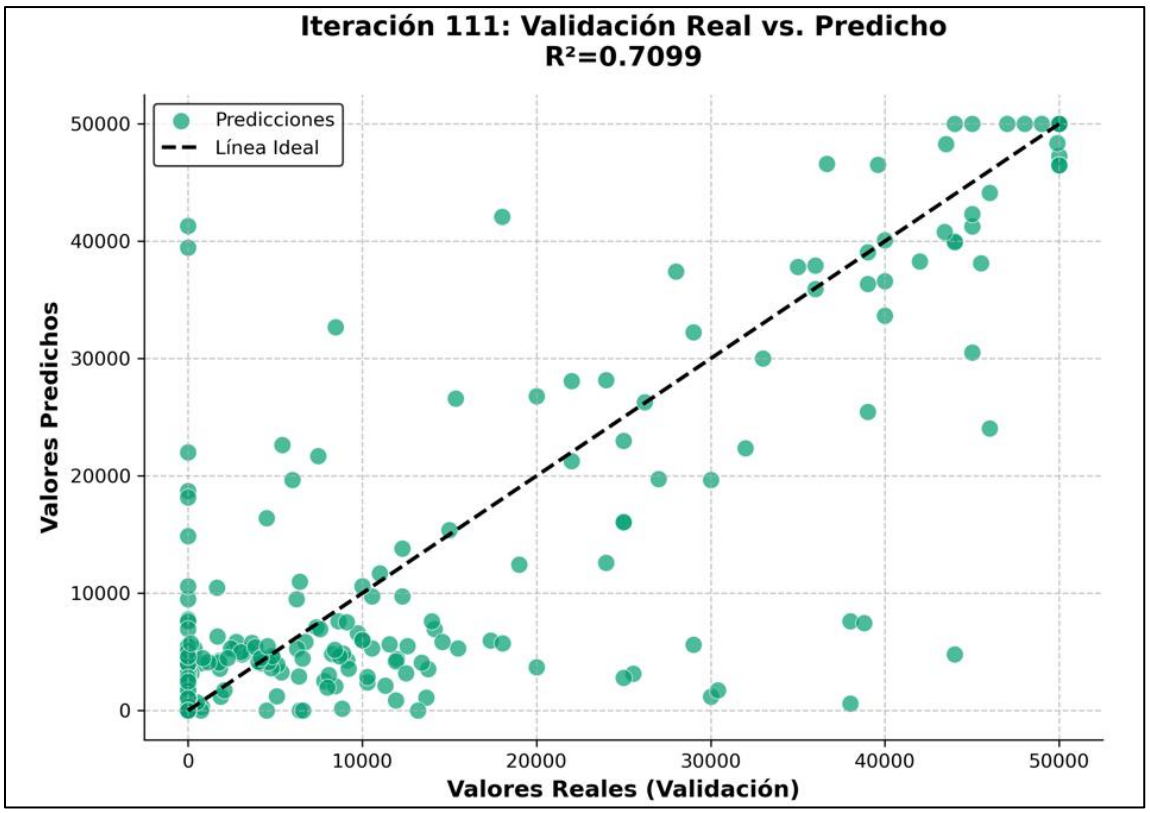

Figura 22. Valores reales vs predichos sobre los datos de validación del mejor modelo de la fase 4.

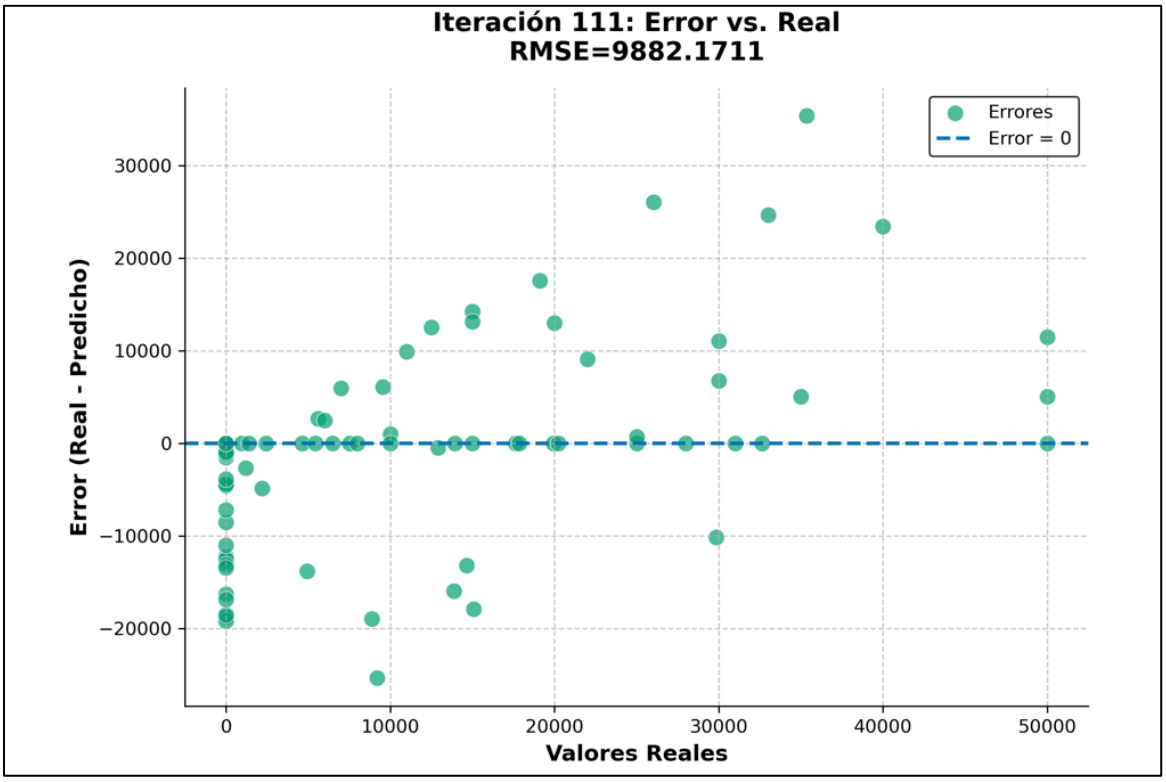

Figura 23. Error predicción vs real de las indemnizaciones en el huracán Milton del mejor modelo de la fase 3.

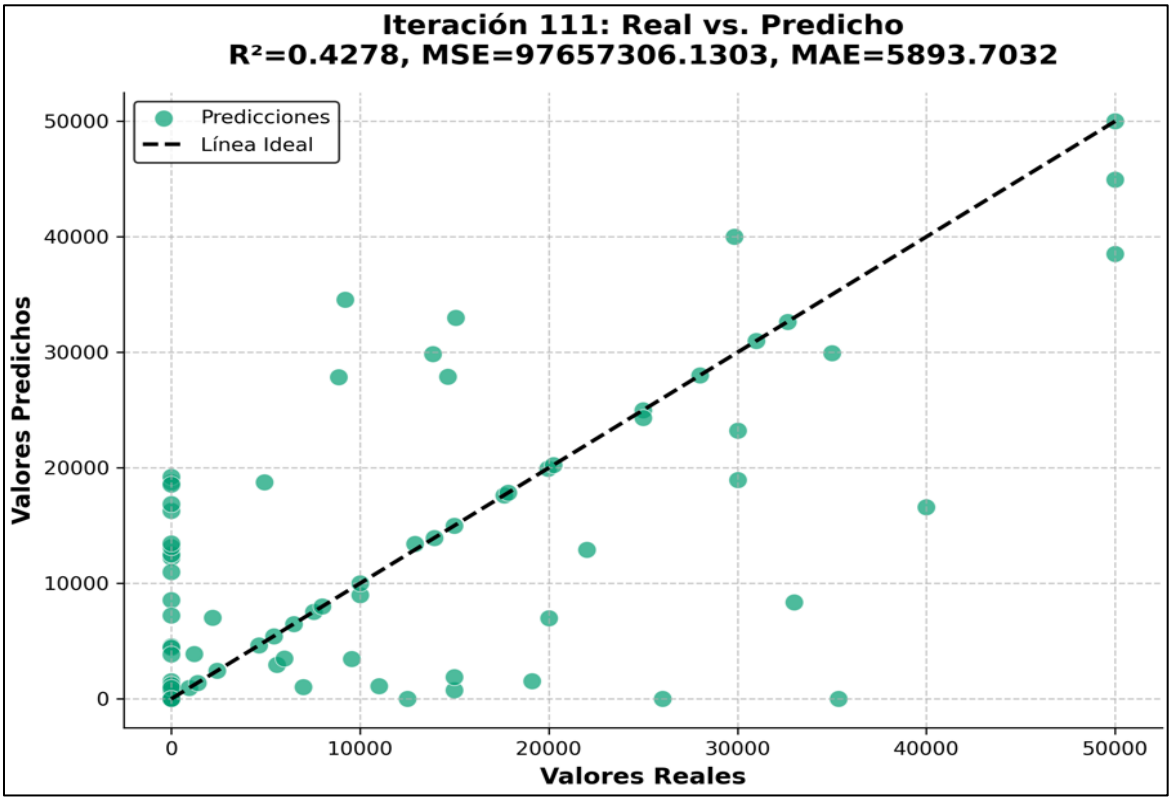

Figura 24. Predicción vs real de las indemnizaciones en el huracán Milton del mejor modelo de la fase 3.

Una vez analizado los gráficos que ilustran la arquitectura del mejor modelo encontrado, podemos formular las siguientes conclusiones: la capacidad de aprendizaje del modelo es significativamente superior obteniendo un R^2 con los datos de validación de 0.7099, es decir, aproximadamente un 12 % más indicando que se han encontrado unos pesos que predicen mejor las indemnizaciones. Además, el aumento significativo del R^2 de validación hace que el R^2 con los datos predichos de las indemnizaciones del huracán Milton aumente un 35.7 % , respecto al mejor modelo de la fase 3, hasta los 0.4278. Además, como se puede apreciar en la figura 23, los errores de predicción que se comenten son mas livianos y de menor intensidad además de que aumentan considerablemente el número de casos cuyo error de predicción es cero en la franja de los 10.000 – 30.000 dólares. Por tanto, a mayor precisión y aumento de los errores de predicción cero hace que disminnuya significativamente el RMSE y MAE.

Tras la evaluación exhaustiva de 25.000 modelos, se concluye que la forma más eficaz para predecir las indemnizaciones por huracanes en el Condado de Lee es a través de una red neuronal feedforward cuya arquitectura viene determinada por los siguientes hiperparámetros: funciones de activación

ReLu, funciones de activación ajustadas a las indemnizaciones históricas, función de pérdida clásica como el MSE, profundidad del modelo a través de seis capas ocultas, múltiples neuronas en las capas ocultas, optimizadores comunes como Adam y un inicializador de pesos que sigue una distribución uniforme con los parámetros descritos por He Uniforme.

Sin embargo, como podemos observar, a pesar de que el modelo haya mejorado sus métricas existen predicciones erróneas, especialmente al detectar aquellos cuya indemnización real es nula. Por otro lado, los valores reales que si tienen indemnización existe un porcentaje muy alto de aciertos y aproximaciones demostrando que a pesar de las limitaciones computacionales y de calidad de datos podemos aproximarnos a la realidad necesitando solamente un modelo de red neuronal *feedforward*.

Otro de los aspectos más importantes de un modelo son los pesos. Cabe destacar que, por la interconexión de las distintas neuronas, parcialmente se pierde la vista sobre qué variables utilizadas son los más importantes del modelo, cuya respuesta vendría determinada por la creación de modelos independientes sin tener en cuenta esas variables y ver la calidad de predicción resultante, aunque teóricamente cuantos más parámetros tenga nuestra red neuronal mayor será la capacidad de aprendizaje y menor el error de predicción. A continuación, la figura 25 representa los pesos que se han utilizado para las predicciones del mejor modelo:

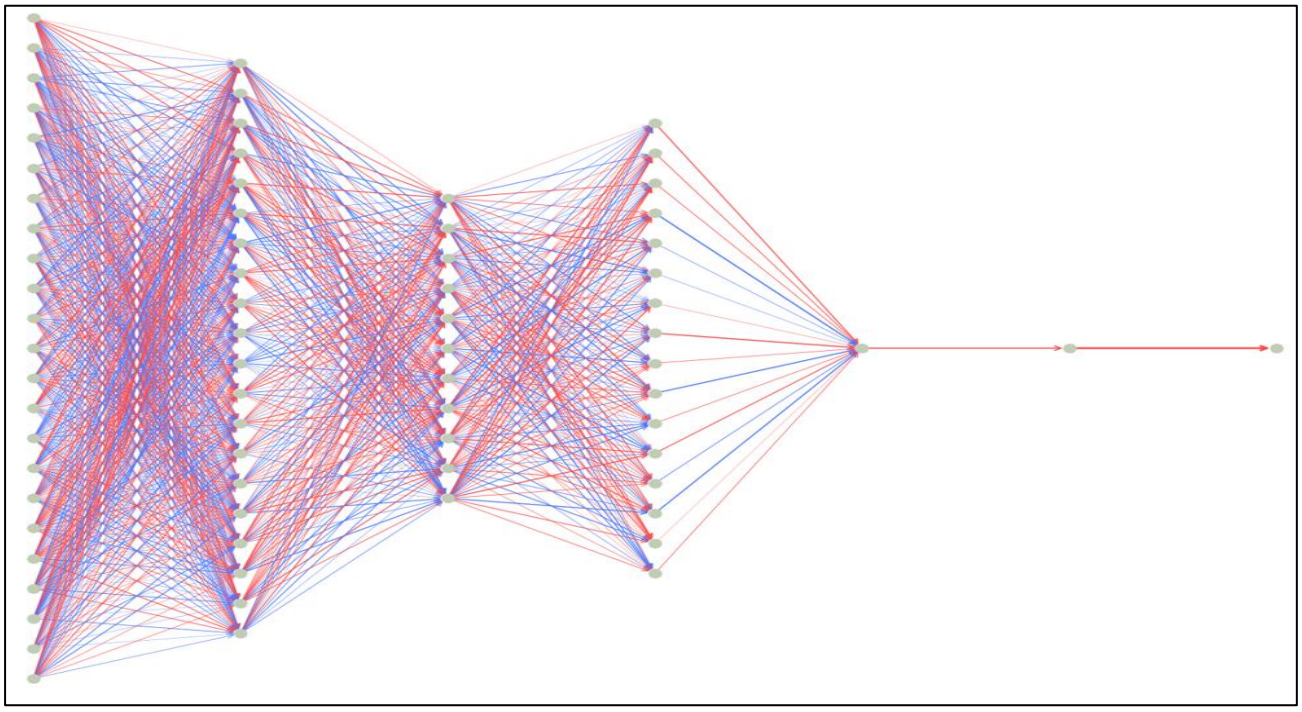

Figura 25. Arquitectura del mejor modelo de la fase 4 donde se recogen las magnitudes de los pesos de las conexiones entre neuronas. El color rojo indica pesos positivos y el color azul pesos negativos donde cuanto más intenso el color más peso tiene.

Finalmente se presentan las métricas obtenidas por la mejor red neuronal artificial *feedforward* encontrada las cuales se compararán con las de un modelo lineal generalizado o GLM de regresión lineal como forma tradicional de predicción. A continuación, la tabla 17 muestra las métricas obtenidas por los diferentes modelos:

Tabla 17. Métricas obtenidas por el mejor modelo de la fase 4 y por el modelo general lineal (GLM)

	Red Neuronal	GLM
Aciertos	41	0
Diferencia	51,731	356,869
R2	0.4278	0
EVS	0.4295	0.0193
MSE	97,653,924	181,171,600
RMSE	9,882	13,460
MAE	5,893	10,927

Fuente:Elaboración Propia

Por tanto, se demuestra de forma muy significativa como la mejor forma de predecir las indemnizaciones de los siniestros por seguros paramétricos de huracanes en el Condado de Lee en Florida es a través de las redes neuronales artificiales *feedforward* las cuales teóricamente tienen una capacidad de aprendizaje ilimitado creando relaciones muy complejas a través de los pesos que hace imposible poder utilizar otros modelos matemáticos. Además, y en especial en seguros paramétricos catastróficos meteorológicos debemos de tener en cuenta no solo las funciones comunes de las redes neuronales, sino que cabe pensar que existe otra forma de capturar esas relaciones a través de funciones de activación propias hechas con los datos de indemnización históricos.

6. CONCLUSIONES

La evolución computacional y la gran accesibilidad a los datos en los últimos años ha hecho posible que se desarrollen herramientas capaces de resolver problemas muy complejos que serían imposibles con las técnicas matemáticas tradicionales. Las redes neuronales artificiales han cambiado la forma de pensar en cuanto a la forma de resolver un problema por una única premisa teórica, y es que estos modelos tienen una capacidad ilimitada de aprendizaje haciendo que ninguna otra modelización pueda competir con ella siempre y cuando se tenga la capacidad computacional y calidad del dato necesaria para poder utilizar todo el potencial de esta herramienta.

El presente trabajo ha estudiado cómo es la forma más optima, eficiente y precisa de aplicar los seguros paramétricos ante cualquier desastre meteorológico haciendo especial atención en los huracanes que afecten al Condado de Lee en Florida. La forma más óptima porque si se cumplen los requisitos mencionados anteriormente no haría falta probar todos los diferentes modelos matemáticos existentes más complejos que pudieran aplicar a este caso concreto, y los cuales no funcionarían, sino que solo haría falta precisión en los datos climáticos y de los asegurados de un área concreta y de un rango de sumas aseguradas específicas para crear un código de programación capaz de encontrar la arquitectura de una red neuronal artificial *feedforward* capaz de predecir todos y cada una de las indemnizaciones de los asegurados ante cualquier intensidad y forma del huracán, el cual se debería de añadir otras técnicas como las redes neuronales generativas. Además, el mismo código de programación se podría reutilizar e implementar en todas las zonas y sumas aseguradas de todos los Estados Unidos simplemente a través de un único código.

Es la forma más eficiente de implementar este tipo de seguros porque no se necesitarían ninguna comprobación de desperfectos posterior al huracán de un perito, sino que es el propio modelo quien definiría esos desperfectos lo cual resultaría en unos ahorros muy significativos porque teóricamente no habría gastos operativos a excepción de los modeladores de la red neuronal y se evitarian fraudes. Por otro lado, la eficiencia surge del pago de las indemnizaciones las cuales serían automáticas. Es la forma más precisa porque las redes neuronales es la única técnica que puede minimizar el error de las predicciones y, teóricamente, a medida que aumentásemos su capacidad de aprendizaje el error tendería a cero.

En cuanto al estudio de la presente investigación, se ha podido demostrar cómo utilizando redes neuronales artificiales *feedforward* e implementado esta nueva técnica novedosa de las funciones de activación ajustadas a datos indemnizatorios históricos exclusivos a ciertos asegurados y zonas, unido a la limitación computacional y calidad de los datos han hecho posible que solo probando 25.000

configuraciones diferentes se haya obtenido una calidad de aprendizaje y predicción significativo.

Por tanto, ante las relevantes conclusiones, se podría plantear una nueva forma de asegurar los huracanes en Florida siendo: por un lado, implementar las técnicas de redes neuronales artificiales *feedforward* presentes en los 66 condados restantes de Florida añadiendo otras técnicas matemáticas de simulación de escenarios los cual servirán para entrenar el modelo. Por otro lado, que el estado y las aseguradoras trabajasen conjuntamente para poder llevar a cabo este seguro paramétrico ya que las aseguradoras tienen la capacidad, conocimiento y calidad del dato necesarios para poder implementar las redes neuronales artificiales y que fuera el estado quien pusiera la masa y autoridad financiera para hacer el seguro paramétrico lo más accesible posible. Por último, utilizar la capacidad que nos da el sector financiero para absorber gran parte del impacto económico que pudiera ocasionar a través de derivados financieros como bonos catastróficos.

Por ende, las redes neuronales artificiales son la mejor herramienta de predicción que las matemáticas hayan podido crear y, por tanto, el ámbito de actuación no solo debería quedarse en los seguros paramétricos de huracanes en Florida sino se debería implementar en todos los seguros que comercializan las aseguradas siempre y cuando la casuística lo permita y la calidad del dato sea lo suficientemente buena. Tanto las aseguradoras como los asegurados se verían beneficiados si se implementase estos modelos ya que, por un lado, los seguros serían significativamente más económicos al reducirse casi por completo los gastos operativos que estos conllevan a las aseguradoras, además de eliminarse el fraude y redistribuir eficientemente los recursos retenidos por las aseguradoras ante una pobre medición del riesgo debido a otras modelizaciones.

7. REFERENCIAS

Barron, J. T. (2017). *Continuously differentiable exponential linear units*. Arxiv. https://arxiv.org/pdf/1704.07483.

Barron, J. T. (2021). *Squareplus: A softplus-like algebraic rectifier*. Arxiv. https://arxiv.org/pdf/1606.08415.

AON. (2024). *I estudio sobre seguros paramétricos en España*. (inf. téc.). Comisión Técnica de Seguros Paramétricos de AON.

Courbariaux, M., Bengio, Y., y David, J.-P. (2015). BinaryConnect: Training deep neural networks with binary weights during propagations. Arxiv. https://arxiv.org/pdf/1511.00363.

Dauphin, Y. N., Fan, A., Auli, M., y Grangier, D. (2017). Language modeling with gated convolutional networks. Proceedings of the 34th International Conference on Machine Learning (ICML), PMLR 70. https://arxiv.org/pdf/1612.08083.

Denuit, M., Hainaut, D., y Trufin, J. (2019). *Effective statistical learning methods for actuaries III: Neural networks and extensions*. Springer Actuarial.

Dozat, T. (2016). *Incorporating Nesterov momentum into Adam*. ICLR Workshop Paper. https://cs229.stanford.edu/proj2015/054_report.pdf.

Duchi, J., Hazan, E., y Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12, 2121–2159.

FEMA. (2025). *FimaNfipClaims* [Dataset]. https://www.fema.gov/openfema-data-page/fima-nfip-redacted-claims-v2.

Garcia Ocampo, D., y Lopez Moreira, C. (2024). *Uncertain waters: Can parametric insurance help bridge NatCat protection gaps?. BIS. https://www.bis.org/fsi/publ/insights62.pdf.*

Glorot, X., y Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 249–256.

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*.-. https://www.deeplearningbook.org/.

Gurney, K. (1999). *Computer and symbols versus nets and neurons*. Departamento de Ciencias

Humanas, Brunel University.

He, K., Zhang, X., Ren, S., y Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1026–1034.

Hendrycks, D., y Gimpel, K. (2023). *Gaussian error linear units (GELUs)*. Arxiv. https://arxiv.org/pdf/1606.08415.

Hertz, J., Krogh, A., y Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Addison-Wesley.

Kingma, D. P., y Ba, J. (2015). Adam: A method for stochastic optimization. Proceedings of the International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1412.6980.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097–1105.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., y Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18(185), 1–52. https://jmlr.org/papers/v18/16-558.html.

Loshchilov, I., y Hutter, F. (2019). Decoupled weight decay regularization. Proceedings of the International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1711.05101.

Martins, A. F. T., y Astudillo, R. F. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. Proceedings of the 33rd International Conference on Machine Learning (ICML). https://arxiv.org/pdf/1602.02068.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., ... y Kubica, J. (2013). Ad click prediction: A view from the trenches. Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1222–1230. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41159.pdf.

Misra, D. (2020). *Mish: A self-regularized non-monotonic activation function*. Arxiv. https://arxiv.org/pdf/1908.08681.

Mohseni Salehi, S. S., Erdogmus, D., y Gholipour, A. (2017). Tversky loss function for image segmentation using 3D fully convolutional deep networks. Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, 1–9. https://arxiv.org/pdf/1706.05721.

Pires, P. B., Santos, J. D., y Pereira, I. V. (2023). Artificial neural networks: History and state of the art. En *Encyclopedia of Information Science and Technology*. IGI Global.

Ramachandran, P., Zoph, B., y Le, Q. V. (2017). *Searching for activation functions*. Arxiv. https://arxiv.org/pdf/1710.05941.

Ramy Vélez, S. (2023). *Seguros paramétricos: Transformando la resiliencia ante desastres naturales. Tarificación y gestión del riesgo* [Trabajo fin de máster, Universidad Carlos III de Madrid]. .-.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408.

Shazeer, N., y Stern, M. (2018). *Adafactor: Adaptive learning rates with sublinear memory cost*. Arxiv. https://arxiv.org/abs/1804.04235.

Swiss Re. (2024). *Comprehensive guide to parametric insurance*. (inf. téc.). Swiss Re Corporate Solutions.

UNDP y Generali. (2024). *Parametric insurance to build financial resilience*. (inf. téc.). UNDP y Generali Global Corporate and Commercial.

World Bank. (2007). *Results of preparation work on the design of a Caribbean Catastrophe Risk Insurance Facility*. (inf. téc.).

World Bank. (2012). *Caribbean Catastrophe Risk Insurance Facility (CCRIF): Pooling risk to protect against natural disasters*. (inf. téc.).

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., ... y Hsieh, C.-J. (2020). Large batch optimization for deep learning: Training BERT in 76 minutes. Proceedings of the International Conference on Learning Representations (ICLR). https://arxiv.org/pdf/1904.00962.

Zeiler, M. D. (2012). *ADADELTA: An adaptive learning rate method*. Arxiv. https://arxiv.org/abs/1212.5701.

8. ANEXO

CODIGO DE PYTHON

1. CREACION DE LA BASE DE DATOS Y OBTENCION DATOS METEOROLOGICOS

```
import numpy as np
import openmeteo_requests
import requests_cache
from retry_requests import retry
from datetime import datetime
from geopy.distance import geodesic
import pandas as pd
import numpy as np


df_fema = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/Dataset +
Dataset  Links/Dataset/Insurance Claims/Archivo
Original Polizas 2M/FimaNfipClaims.csv")

# Eliminamos longitude y latitude porque luego lo
añadiremos con censusTract
df_fema = df_fema.drop(["latitude", "longitude"],
axis=1)

# latitude y longitude de censusTract
census_Tract = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/USA
COUNTY/supreme_FLcensusTract.csv")


df_fema = pd.merge(census_Tract, df_fema, on=[
    "censusTract"], how="inner")  # FINAL


def formacion_dataset(df_fema, modelo_tiempo):

    ### 1 - AJUSTE FEMA ###

    # Florida
    df_fema = df_fema[df_fema["state"] == "FL"]  #
Just in Florida

    # Solo huracanes
    df_fema =
df_fema[df_fema["floodEvent"].str.contains(
        "Hurricane", na=False, case=False)]

    df_fema = df_fema.drop(["asOfDate",
```

```
"basementEnclosureCrawlspaceType",
"crsClassificationCode", "dateOfLoss",
"elevationCertificateIndicator",
"elevationDifference", "baseFloodElevation",
"lowestAdjacentGrade", "lowestFloorElevation",
"obstructionType", "originalNBDate",
"amountPaidOnIncreasedCostOfComplianceClaim",
"rateMethod", "condominiumCoverageTypeCode",
"disasterAssistanceCoverageRequired",
"eventDesignationNumber",
                            "ficoNumber",
"floodCharacteristicsIndicator", "floodWaterDuration",
"iccCoverage", "netIccPaymentAmount",
"nfipRatedCommunityNumber",
"nfipCommunityNumberCurrent", "nfipCommunityName",
"nonPaymentReasonContents",
"nonPaymentReasonBuilding", "buildingReplacementCost",
"contentsReplacementCost", "replacementCostBasis",
"waterDepth", "floodZoneCurrent",
"buildingDescriptionCode", "reportedCity", "id",
"amountPaidOnBuildingClaim",
"amountPaidOnContentsClaim"], axis=1)

    # fill na with 0s
    df_fema = df_fema[~(
        (df_fema["originalConstructionDate"].fillna(0)
== 0))]

    # Only huricanes after 2000
    df_fema = df_fema[df_fema["yearOfLoss"] > 2000]

    # Formato Construccion de la casa
    df_fema["originalConstructionDate"] =
df_fema["originalConstructionDate"].str[:4]

    # Formato integer
    df_fema["originalConstructionDate"] =
df_fema["originalConstructionDate"].astype(
        int)

    # Casas mayores a 1900
    df_fema =
df_fema[df_fema["originalConstructionDate"] > 1900]

    # Eliminar duplicados
    df_fema = df_fema.drop_duplicates()

    # Todos los nan con 0
    df_fema = df_fema.fillna(0)  # Se puede rellenar
con k - means?

    ### Reemplazamos letras por valores numericos -
Deducibles ###

    reemplazos = {
        "0": 500,
        "1": 1000,
```

```
        "2": 2000,
        "3": 3000,
        "4": 4000,
        "5": 5000,
        "9": 750,
        "A": 10000,
        "B": 15000,
        "C": 20000,
        "D": 25000,
        "E": 50000,
        "F": 1250,
        "G": 1500,
        "H": 200
    }

    # Los deducibles en valor numerico
    df_fema["contentsDeductibleCode"] =
df_fema["contentsDeductibleCode"].replace(
        reemplazos)
    df_fema["buildingDeductibleCode"] =
df_fema["contentsDeductibleCode"].replace(
        reemplazos)

    ####### 2 - add segundo dataframe con mas datos
sobre huracanes #########

    # Informacion adicional sobre los huracanes
    df_hurricane_important_information =
pd.DataFrame({

        "floodEvent": ["Hurricane Charley", "Hurricane
Frances", "Hurricane Ivan", "Hurricane Jeanne",
"Hurricane Dennis", "Hurricane Katrina",
                       "Hurricane Wilma", "Hurricane
Ike", "Hurricane Isaac", "Hurricane Hermine",
"Hurricane Matthew", "Hurricane Irma",
                       "Hurricane Michael", "Hurricane
Sally", "Hurricane Eta", "Hurricane Elsa", "Hurricane
Ian", "Hurricane Nicole",
                       "Hurricane Idalia", "Hurricane
Debby", "Hurricane Francine", "Hurricane Helene",
"Hurricane Milton"],


        "Start_date": [20040813, 20040903, 20040914,
20040925, 20050709, 20050825, 20051023, 20080910,
                       20120826, 20160828, 20161006,
20170908, 20181009, 20200911, 20201108, 20210706,
                       20220927, 20221009, 20230829,
20240805, 20240912, 20240925, 20241009],


        "End_date": [20040814, 20040907, 20040916,
20040927, 20050711, 20050829, 20051025, 20080913,
                      20120830, 20160902, 20161008,
20170911, 20181011, 20200917, 20201112, 20210708,
                      20220930, 20221111, 20230830,
```

```
20240807, 20240913, 20240927, 20241010],


        "Hurricane_days": [2, 5, 3, 3, 3, 5, 3, 4,
                           5, 6, 3, 4, 3, 7, 5, 3,
                           4, 3, 2, 3, 2, 3, 2],


        "Max_Hurricane_Category": [4, 2, 4, 3, 4, 5,
3, 2,
                                   1, 1, 4, 4, 5, 2,
1, 1,
                                   5, 1, 4, 1, 1, 4,
4],


        "Hurricanes_same_year": [0, 1, 1, 1, 0, 1, 1,
0,
                                 0, 0, 1, 0, 1, 0, 1,
1,
                                 0, 1, 0, 0, 1, 1, 1],

        "Max_Intensity_Hurricane_Latitude": [26.9,
27.2, 27.8, 27.2, 28.5, 27.1, 25.9, 26.3, 28.6,
                                                 33.3,
26.7, 24.7, 30, 30.5, 25.7, 26.5, 26, 27.6, 29, 29.9,
                                                 29.1,
29.1, 23.5],

        "Max_Intensity_Hurricane_Longitude": [-82.1, -
80.2, -88.1, -80.01, -86.2, -89.1, -81.6, -91.1,
                                                  -88.8, -
83.9, -78.9, -81.5, -85.4, -87.5, -83.8, -83.2,
                                                  -82.6, -
80.2, -84.1, -83.3, -92.7, -84, -86.4],

        "Second_Max_Intensity_Hurricane_Latitude":
[28.5, 30.4, 30.2, 28, 30.4, 30.2, 26.9, 23.7, 25,
30.6,

29.7, 25.9, 30.7, 25.5, 29.7, 30.3, 26.7, 30.4, 30,

27.5, 30.3, 30.4, 27.1],

        "Second_Max_Intensity_Hurricane_Longitude":
[-81.4, -84.2, -87.8, -81.9, -87.1, -89.5,

-80, -85.1, -83.5, -83.8, -80.6, -81.6, -84.9,

-80.7, -82.5, -83.4, -81.9, -84.1, -83.4, -83.3, -
91.1, -83.6, -82.4],

        "Hits_Florida_Maximum_Intensity": [1, 0, 0, 1,
0, 0, 1, 0, 0, 1, 0, 0,  1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
0]

    })
```

```
    # Juntamos los dos dataframes anteriores
    df_fema =
df_hurricane_important_information.merge(
        df_fema, on="floodEvent", how="left")

    ###### 3 - RATIOS ######

    # Añadir columnas - ratios que nos interesan #

    df_fema["valormercado_CC"] =
df_fema["buildingPropertyValue"] + \
        df_fema["contentsPropertyValue"]  # Valor
Mercado Continente y Contenido

    df_fema["TOTAL_Covarage"] =
df_fema["totalBuildingInsuranceCoverage"] + \
        df_fema["totalContentsInsuranceCoverage"]  #
Total Asegurado

    df_fema["TOTAL_NET_Payments"] =
df_fema["buildingDamageAmount"] + \
        df_fema["contentsDamageAmount"]  # Pagos Netos
Totales

    df_fema["Ratio_Contenido_Pago"] =
df_fema["contentsDamageAmount"] / \
        df_fema["totalContentsInsuranceCoverage"]  #
Pago Final / Suma asegurada

    df_fema["Ratio_Continente_Pago"] =
df_fema["buildingDamageAmount"] / \
        df_fema["totalBuildingInsuranceCoverage"]  #
Pago Final / Suma asegurada

    df_fema["Ratio_Pago_SumaAsegurada"] =
df_fema["TOTAL_NET_Payments"] / \
        (df_fema["TOTAL_Covarage"])  # Pago total /
suma asegurada

    df_fema["TOTAL_Deducible"] =
df_fema["contentsDeductibleCode"] + \
        df_fema["buildingDeductibleCode"]  # Total
Deducible

    # No indemnizacion por deducible; No indemnizacion
= 0
    df_fema["No_Ind_Deducible"] = (
        df_fema["TOTAL_Deducible"] -
df_fema["TOTAL_NET_Payments"] > 1).astype(int)

    ##### 4 - Añadimos nuevos datos que pueden ser
importantes #######

    # ---- Calculo distancia huracanes y asegurados en
KM
    # Nearest distance to the hurricane
```

```
    Max_Intensity_Hurricane_distance_most_Proxy = []
    # Second Nearest distance to the hurricane
    Second_Intensity_Hurricane_distance_most_Proxy =
[]

    for i in range(df_fema.shape[0]):

        # Latitude/ longitude of the hurricanes
        Max_Intensity_Hurricane_Latitude_Longitude =
tuple(
            df_fema.loc[i,
["Max_Intensity_Hurricane_Latitude",
"Max_Intensity_Hurricane_Longitude"]])

Second_Max_Intensity_Hurricane_Latitude_Longitude =
tuple(
            df_fema.loc[i,
["Second_Max_Intensity_Hurricane_Latitude",
"Second_Max_Intensity_Hurricane_Longitude"]])

        # Latitude /longitude insurance policy
        Insurance_Policy_Latitude_Longitude = tuple(
            df_fema.loc[i, ["latitude", "longitude"]])

        # Appending the distance (km) between
latitudes/longitudes

Max_Intensity_Hurricane_distance_most_Proxy.append(geo
desic(

Max_Intensity_Hurricane_Latitude_Longitude,
Insurance_Policy_Latitude_Longitude).kilometers)

Second_Intensity_Hurricane_distance_most_Proxy.append(
geodesic(

Second_Max_Intensity_Hurricane_Latitude_Longitude,
Insurance_Policy_Latitude_Longitude).kilometers)

        print(i)  # Timer

    # Reemplazar todos los nan con 0
    df_fema = df_fema.fillna(0)

    # Cogemos los continentes menores a 50.000
    df_fema =
df_fema[df_fema["totalBuildingInsuranceCoverage"] <
50001]

    # Eliminamos huracanes que no queremos
    df_fema = df_fema[df_fema["yearOfLoss"] > 2000]

    ##### 5 - Pasamos a valor numerico y eliminamos lo
que no nos interesa ####

    df_fema = df_fema.applymap(
        lambda x: pd.to_numeric(x,
```

```
errors="coerce")).reset_index()

    ###### 6 - Obtenemos los datos climaticos #####

    # Obtenemos los valores unicos del dataset para
que nos funcione la API


df_UniqueCombinations_Latitude_Longitude_StartEnd_Date
= df_fema[[
        "latitude", "longitude", "Start_date",
"End_date"]].drop_duplicates()

    print("Numero de combinaciones unicas: ",

df_UniqueCombinations_Latitude_Longitude_StartEnd_Date
)

    # Setup the Open-Meteo API client with cache and
retry on error ; API proporcionada por OpenMeteo con
modificaciones
    cache_session =
requests_cache.CachedSession(".cache", expire_after=-
1)
    retry_session = retry(cache_session, retries=5,
backoff_factor=0.2)
    openmeteo =
openmeteo_requests.Client(session=retry_session)
    url = "https://archive-api.open-
meteo.com/v1/archive"

    df_all_records = pd.DataFrame()

    # Agrupar datos en 250 para que el código vaya mas
rápdido
    batch_size = 250
    total_records =
len(df_UniqueCombinations_Latitude_Longitude_StartEnd_
Date)

    for batch_start in range(0, total_records,
batch_size):
        batch_end = min(batch_start + batch_size,
total_records)
        batch =
df_UniqueCombinations_Latitude_Longitude_StartEnd_Date
.iloc[
            batch_start:batch_end]

        for idx, unique_record in batch.iterrows():
            latitude = unique_record["latitude"]
            longitude = unique_record["longitude"]
            Start_date = unique_record["Start_date"]
            End_date = unique_record["End_date"]

            params = {
                "latitude": latitude,
```

```
                "longitude": longitude,
                "start_date":
(datetime.strptime(str(int(Start_date)),
"%Y%m%d")).strftime("%Y-%m-%d"),
                "end_date":
(datetime.strptime(str(int(End_date)),
"%Y%m%d")).strftime("%Y-%m-%d"),
                "daily": ["wind_speed_10m_max",
"wind_gusts_10m_max", "precipitation_sum",
"shortwave_radiation_sum", "surface_pressure_min",
"wind_gusts_10m_mean", "wind_speed_10m_mean"],
                "models": modelo_tiempo,
                "timezone": "America/New_York"
            }

            responses = openmeteo.weather_api(url,
params=params)
            response = responses[0]

            daily = response.Daily()
            daily_data = {
                "date": pd.date_range(
                    start=pd.to_datetime(daily.Time(),
unit="s", utc=True),
                    end=pd.to_datetime(
                        daily.TimeEnd(), unit="s",
utc=True),

freq=pd.Timedelta(seconds=daily.Interval()),
                    inclusive="left"
                ),
                "daily_wind_speed_10m_max":
daily.Variables(0).ValuesAsNumpy(),
                "daily_wind_gusts_10m_max":
daily.Variables(1).ValuesAsNumpy(),
                "daily_precipitation_sum":
daily.Variables(2).ValuesAsNumpy(),
                "daily_shortwave_radiation_sum":
daily.Variables(3).ValuesAsNumpy(),
                "daily_surface_pressure_min":
daily.Variables(4).ValuesAsNumpy(),
                "daily_wind_gusts_10m_mean":
daily.Variables(5).ValuesAsNumpy(),
                "daily_wind_speed_10m_mean":
daily.Variables(6).ValuesAsNumpy(),
                "latitude": latitude,
                "longitude": longitude,
                "Start_date": Start_date,
                "End_date": End_date
            }

            daily_dataframe =
pd.DataFrame(data=daily_data)
            df_all_records = pd.concat(
                [df_all_records, daily_dataframe],
ignore_index=True)
```

```
        print(f"Batch {batch_start} a {batch_end}
procesado.")

    df_fema = pd.merge(df_all_records, df_fema, on=[
        "longitude", "latitude", "Start_date",
"End_date"], how="inner")  # FINAL

    df_fema = df_fema.applymap(
        lambda x: pd.to_numeric(x,
errors="coerce")).reset_index()

    return df_fema


df_desire = formacion_dataset(df_fema, "era5")


# Modificar ; Cogemos los valores máximos entre todos
los dias del huracan

df_max = df_desire.groupby(
    ["index"]).max().reset_index()

df_max.to_csv(
    "BBDD_HURACANES_menor_50000_MAX.csv", index=False)

#

filas_filtradas = df_max[df_max["countyCode"] ==
12071.0]  # Lee County

filas_filtradas.to_csv(
    "12071_BBDD_HURACANES_menor_50000_MAX.csv",
index=False)
```

2. PREPROCESAMIENTO DE LA BASE DE DATOS

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler,
MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import streamlit as st
from sklearn.decomposition import PCA

# ↕ DATA

st.header("IMPORTANT DATASET DECISION")

Mean_Max_Dataset_decision = st.multiselect("MEAN or
MAX dataset", ["MAX", "MEAN"],
```

```
                                                "MAX")

st.header("BINARY CLASSIFICATION DECISIONS -> [0,1],
None & [-1,1]")


Binary_Classification_decision =
st.multiselect("Binary Classification for the target -
output - y like -> Yes = [0,1] ; No = Nothing ;
ONE&minusONE = [-1,1]", ["Yes", "No", "ONE&minusONE"],
                                                    "Yes")

# Binary_Classification_01_OR_minus1AND1 =
st.multiselect("Binary Classification like [0,1] or [-
1,1] ", ["First", "Second"],
#
"First")

st.header("PCA")

PCA_decision = st.multiselect("PCA in the dataset? ",
["Yes", "No"],
                                "No")

PCA_columns_decision = st.slider(
    "Number of columns using PCA ", 1, 100, 20)

st.header("HOT - ENCODER FOR CATEGORICAL DATA")

HotEncoder_Decesion = st.multiselect("HotEncoder for
the Categorical Data", ["Yes", "No"],
                                        "No")

st.header("SCALAR DECISION")

Scalar_Decision = st.multiselect("How do you Want to
Scalar your Dataset? ", ["MinMax", "Standarizing",
"None"],
                                    "None")

st.header("SPLITTER DECISION -> TEST, TRAIN,
VALIDATION ")

Data_SpliterTest_Decision = st.slider(
    "Split of the Test Set:  ", 0.0, 1.0, 0.2)

Data_SpliterVAL_Decision = st.slider("Split of the VAL
Set:  ", 0.0, 1.0, 0.3)

st.header("SEED FOR RANDOMNES: RECOVERY TOOL")

Seed_Random_Recovery_Decision = st.slider(
    "Seed use for randomize recovery: ", 1, 1000, 55)


def categorical_non_categorical_data(path_to_data):
```

```
    df_imputed = pd.read_csv(path_to_data)

    interested_columns = ["occupancyType"]

    df_categorical = df_imputed[interested_columns]
    df_non_categorical =
df_imputed.drop(columns=interested_columns)

    return df_categorical, df_non_categorical


def data_scaler(tipo_escala, df_categorical,
df_non_categorical):

    if PCA_decision == "Yes":

        df_preprocessed = pd.concat(
            [df_categorical, df_non_categorical],
axis=1)

        Target_Y_Output = df_preprocessed.iloc[:, -1]
        df_preprocessed = df_preprocessed.iloc[:, :-1]

        # PCA
        pca = PCA(n_components=PCA_columns_decision)
        df_preprocessed =
pca.fit_transform(df_preprocessed)

        if tipo_escala == "MinMax":

            escalar = MinMaxScaler()
            df_preprocessed =
escalar.fit_transform(df_preprocessed)

        if tipo_escala == "Standarizing":

            escalar = StandardScaler()  #
Normalizacion de los datos
            df_preprocessed =
escalar.fit_transform(df_preprocessed)

        df_preprocessed =
pd.DataFrame(df_preprocessed)

        df_preprocessed = pd.concat([df_preprocessed,
Target_Y_Output], axis=1)

        df_preprocessed = df_preprocessed.to_numpy()

        return df_categorical, df_non_categorical,
df_preprocessed

    elif HotEncoder_Decesion == "Yes":

        codificador_categoricos = OneHotEncoder(
            drop="first", sparse_output=False)
        df_categorical_scaled =
```

```
codificador_categoricos.fit_transform(
            df_categorical)

    else:

        df_categorical_scaled = df_categorical

    if tipo_escala == "MinMax":

        escalar = MinMaxScaler()
        df_non_categorical_scaled =
escalar.fit_transform(df_non_categorical)

    elif tipo_escala == "Standarizing":

        escalar = StandardScaler()  # Normalizacion de
los datos
        df_non_categorical_scaled =
escalar.fit_transform(df_non_categorical)

    else:

        df_non_categorical_scaled = df_non_categorical

    df_preprocessed = np.hstack(
        (df_categorical_scaled, df_non_categorical))

    return df_categorical_scaled, df_non_categorical,
df_preprocessed


def data_splitter_train_test(dataset_preprocessed,
test_size, split_data_percentage_Validation,
set_sed_for_randommes):

    X_independent = dataset_preprocessed[:, -2:]
    Y_dependent = dataset_preprocessed[:, :-2]

    X_train, X_test, y_train, y_test =
train_test_split(
        X_independent, Y_dependent,
test_size=test_size,
random_state=set_sed_for_randommes)
    X_train, X_val, y_train, y_val = train_test_split(

        X_train, y_train,
test_size=split_data_percentage_Validation,
random_state=set_sed_for_randommes)

    return X_train, X_test, X_val, y_train, y_test,
y_val


def
Binary_nonBinary_Classification_Scalar_Splitter_Data(B
inary_Classification, scaler_trype,
split_data_percentage_xy,
```

```
split_data_percentage_Validation, seed_data_splitter):

    Binary_Classification =
Binary_Classification_decision

    scaler_trype = Scalar_Decision

    split_data_percentage_xy =
Data_SpliterTest_Decision

    split_data_percentage_Validation =
Data_SpliterVAL_Decision

    Seed_Random_Recovery_Decision = seed_data_splitter

    if Mean_Max_Dataset_decision == "MAX":

        filename =
"C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER UC3M/TFM/TFM
FINAL - Parametrics Insurence/FINALE ULTIME
SCRIPTS/bbdd_NeuralNetwork/ddbb_era5_max_NN.csv"

    else:

        filename =
"C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER UC3M/TFM/TFM
FINAL - Parametrics Insurence/Ultimate GOD
database/ddbb FINAL/GOD_MAX_Dataset.csv"

    df_categorical, df_non_categorical =
categorical_non_categorical_data(
        filename)

    if Binary_Classification == "Yes":


df_non_categorical["TotalAmountPaid_by_insurance"] =
df_non_categorical["TotalAmountPaid_by_insurance"].mas
k(
            # Binary Classification

df_non_categorical["TotalAmountPaid_by_insurance"] >
0, 1)

        # Move the column
"TotalAmountPaid_by_insurance" to the end because when
data_scaler function there is no column names
        df_non_categorical =
df_non_categorical.drop("TotalAmountPaid_by_insurance"
, axis=1).join(

df_non_categorical["TotalAmountPaid_by_insurance"])

        if scaler_trype == "MinMax":

            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
```

```
                scaler_trype, df_categorical,
df_non_categorical)

        elif scaler_trype == "Standarizing":

            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

            # Drop the last column because it has been
standarized the target
            df_preprocessed = df_preprocessed[:, :-1]

            df_preprocessed = np.hstack(
                (df_preprocessed,
df_non_categorical["TotalAmountPaid_by_insurance"].to_
numpy().reshape(-1, 1)))

            # Data splitter
            X_train, X_test, X_val, y_train, y_test,
y_val = data_splitter_train_test(
                df_preprocessed,
split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)

            return X_train, X_test, X_val, y_train,
y_test, y_val

        else:  # Not Scalating any data

            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

        X_train, X_test, X_val, y_train, y_test, y_val
= data_splitter_train_test(
            df_preprocessed, split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)
# Split the data

        return X_train, X_test, X_val, y_train,
y_test, y_val

    elif Binary_Classification == "ONE&minusONE":

        # Setting the tarjet values to binary

df_non_categorical["TotalAmountPaid_by_insurance"] =
df_non_categorical["TotalAmountPaid_by_insurance"].mas
k(
            # Binary Classification

df_non_categorical["TotalAmountPaid_by_insurance"] >
0,
1).mask(df_non_categorical["TotalAmountPaid_by_insuran
```

```
ce"] == 0, -1)

        # Move the column
"TotalAmountPaid_by_insurance" to the end because when
data_scaler function there is no column names
        df_non_categorical =
df_non_categorical.drop("TotalAmountPaid_by_insurance"
, axis=1).join(

df_non_categorical["TotalAmountPaid_by_insurance"])

        if scaler_trype == "MinMax":

            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

        elif scaler_trype == "Standarizing":

            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

            # Drop the las column because it has been
standarized the target
            df_preprocessed = df_preprocessed[:, :-1]

            df_preprocessed = np.hstack(
                (df_preprocessed,
df_non_categorical["TotalAmountPaid_by_insurance"].to_
numpy().reshape(-1, 1)))

            # Data splitter
            X_train, X_test, X_val, y_train, y_test,
y_val = data_splitter_train_test(
                df_preprocessed,
split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)

            return X_train, X_test, X_val, y_train,
y_test, y_val

        else:  # Not Scalating any data

            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

        X_train, X_test, X_val, y_train, y_test, y_val
= data_splitter_train_test(
            df_preprocessed, split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)
# Split the data
```

```
        return X_train, X_test, X_val, y_train,
y_test, y_val

    else:

        # Move the column
"TotalAmountPaid_by_insurance" to the end because when
data_scaler function there is no column names
        df_non_categorical =
df_non_categorical.drop(["Ratio_Continente_Pago",
"Ratio_Contenido_Pago"], axis=1).join(

df_non_categorical[["Ratio_Continente_Pago",
"Ratio_Contenido_Pago"]])

        if scaler_trype == "MinMax":
            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

            # Drop the las column because it has been
standarized the target
            df_preprocessed = df_preprocessed[:, :-1]

            # Join the two datasets
            df_preprocessed = np.hstack(
                (df_preprocessed,
df_non_categorical["TotalAmountPaid_by_insurance"].to_
numpy().reshape(-1, 1)))

            # Data splitter
            X_train, X_test, X_val, y_train, y_test,
y_val = data_splitter_train_test(
                df_preprocessed,
split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)

            return X_train, X_test, X_val, y_train,
y_test, y_val

        elif scaler_trype == "Standarizing":
            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)

            # Drop the las column because it has been
standarized the target
            df_preprocessed = df_preprocessed[:, :-1]

            # Join the two datasets
            df_preprocessed = np.hstack(
                (df_preprocessed,
df_non_categorical["TotalAmountPaid_by_insurance"].to_
numpy().reshape(-1, 1)))
```

```
            # Data splitter
            X_train, X_test, X_val, y_train, y_test,
y_val = data_splitter_train_test(
                df_preprocessed,
split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)

            return X_train, X_test, X_val, y_train,
y_test, y_val

        else:  # Not Scalating any data
            df_non_categorical_scaled,
df_categorical_scaled, df_preprocessed = data_scaler(
                scaler_trype, df_categorical,
df_non_categorical)  # No data Scalated ; "None"

        X_train, X_test, X_val, y_train, y_test, y_val
= data_splitter_train_test(
            df_preprocessed, split_data_percentage_xy,
split_data_percentage_Validation, seed_data_splitter)
# Split the data

        return X_train, X_test, X_val, y_train,
y_test, y_val


if st.sidebar.button("Execute Splitter Data Program"):

    # Convert to non list the one lenghts outputs
$$$$$$
    for variable_name, value in
globals().copy().items():

        print(variable_name, value)

        if isinstance(value, list) and len(value) ==
1:  # 1º is a list? 2º has len value 1?

            # if len(value) == 1:

            globals()[variable_name] = value[0]

            # print(variable_name, value)

    # $$$$$$$$$$

    y_train, y_test, y_val, X_train, X_test, X_val =
Binary_nonBinary_Classification_Scalar_Splitter_Data(
        Binary_Classification_decision,
Scalar_Decision, Data_SpliterTest_Decision,
Data_SpliterVAL_Decision,
Seed_Random_Recovery_Decision)

    path_directory =
"C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER UC3M/TFM/TFM
FINAL - Parametrics Insurence/Python
Scripts/ORDENADO/Ordenado
```

```
Master/TEST_TRAIN_VAL_SPLITTER_DATA/SPLITTERDATA.npz"

    np.savez(path_directory, y_train=y_train,
y_test=y_test, y_val=y_val,
             X_train=X_train, X_test=X_test,
X_val=X_val)

    print("PYTHON PROGRAM HAS JUST FINISHED")
```

3. TUNER RED NEURONAL

```
from keras_tuner.tuners import BayesianOptimization
from keras_tuner.tuners import Hyperband
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint, ReduceLROnPlateau, TensorBoard,
CSVLogger
import keras
from keras import layers
import keras_tuner
from tensorflow.keras.layers import BatchNormalization
import tensorflow as tf
import streamlit as st
import numpy as np
import random
import sys
random_seed_decision = st.multiselect("Random Seed for
randomnes reproductibility?", ["Yes", "No"],
                                      "Yes")
random_seed_decision_number = st.slider(
    "Number for the Random Seed: ", 1, 1000, 123)

if random_seed_decision == "Yes":
    random.seed(random_seed_decision_number)

else:
    pass


st.title("NEURAL NETWORK STRUCTURE")

st.header("NEURONS")

min_number_neurons = st.slider("Min Number of
Neurons", 1, 10, 2)
max_number_neurons = st.slider("Max Number of
Neurons", 1, 10, 4)

optimizers_neural_network_all = ["adam", "sgd",
                                 "rmsprop", "adagrad",
"adadelta", "nadam", "ftrl"]

optimizers_neural_network = st.multiselect("Optimizer
Function", optimizers_neural_network_all,

default=["adam", "adagrad"])
```

```
batch_normalization_decision_before_neurons =
st.multiselect("Batch Normalization BEFORE Neurons?",
["Yes", "No"],

"Yes")

batch_normalization_decision_between_neurons =
st.multiselect("Batch Normalization BETWEEN Neurons?",
["Yes", "No"],

"Yes")

st.header("LAYERS")

activation_function_layer_all = ["sigmoid", "swish",
"tanh", "relu",
                                 "softplus",
"hard_sigmoid", "gelu", "elu",
                                 "exponential",
"selu", "softsign", "softmax"]

activation_function_layer = st.multiselect("Acivation
Function", activation_function_layer_all,

default=["relu", "softplus"])

activation_function_last_neuron =
st.multiselect("Acivation Function",
activation_function_layer_all,

default="softmax")  # Its the last layer - neuron

min_layer_value = st.slider("Min Layer value", 1,
1000, 1)
max_layer_value = st.slider("Max Layer Value", 1,
1000, 42)
step_layer_value = st.slider("Step Layers Value", 1,
1000, 1)

st.header("OTHERS IN THE NEURAL NETWORK STRUCTURE")

input_shape_data = st.slider(
    "Number of Inputs in the Dataset (columns): ", 1,
100, 42)

set_Normal_Random_Weights_beforehand =
st.multiselect("Random Weights Beforehand? ", ["Yes",
"No"],

"Yes")

# Only works if set_Normal_Random_Weights_Beforehand
is a "Yes"
Normal_Distribution_Mean = st.slider("Normal MEAN", 0,
20, 0)
# Only works if set_Normal_Random_Weights_Beforehand
```

```
is a "Yes"
Normal_Distribution_Sd = st.slider("Normal SD", 0, 20,
2)

Learning_Rate_Min_Value = st.number_input(
    "Learning Rate MIN Value", 0.0, 1.0, 0.0005,
0.0005)
# min_value=0, max_value=2, 0.5, step=0.01
Learning_Rate_Max_Value = st.number_input(
    "Learning Rate MAX Value", 0.0, 2.0, 0.5, 0.01)

ascending_layers = st.multiselect("Ascending Layers in
the Neural Network?", ["Yes", "No"],
                                  "No")

descending_layers = st.multiselect("Descending Layers
in the Neural Network?", ["Yes", "No"],
                                   "No")

st.title("MODEL.COMPILE")

all_possible_loss = [
    "mean_squared_error",
    "mean_absolute_error",
    "mean_absolute_percentage_error",
    "mean_squared_logarithmic_error",
    "binary_crossentropy",
    "categorical_crossentropy",
    "sparse_categorical_crossentropy",
    "kullback_leibler_divergence",
    "poisson",
    "cosine_similarity",
    "hinge",
    "squared_hinge",
    "categorical_hinge",
    "logcosh",
    "huber"
]

loss_to_use = st.multiselect(
    "Loss for the Model Compile:", all_possible_loss,
default=["binary_crossentropy"])

all_possible_Metrics = [
    "accuracy",
    "binary_accuracy",
    "categorical_accuracy",
    "sparse_categorical_accuracy",
    "top_k_categorical_accuracy",
    "sparse_top_k_categorical_accuracy",
    "mean_squared_error",
    "mean_absolute_error",
    "mean_absolute_percentage_error",
    "mean_squared_logarithmic_error",
    "cosine_proximity",
    "AUC", "precision", "recall"  # Binary
Classification
```

```
]

metric_to_use = st.multiselect(
    "Metric for the Model Compile:",
all_possible_Metrics, default=["AUC"])

st.title("EARLY_STOP")

all_EarlyStop_Monitor = ["loss",
                         "accuracy",
                         "binary_accuracy",
                         "categorical_accuracy",

"sparse_categorical_accuracy",
                         "mean_squared_error",
                         "mean_absolute_error",
                         "cosine_proximity",
                         "val_loss",
                         "val_accuracy",
                         "val_binary_accuracy",
                         "val_categorical_accuracy",

"val_sparse_categorical_accuracy",
                         "val_mean_squared_error",
                         "val_mean_absolute_error",
                         "val_cosine_proximity"]

early_stopping_monitor = st.multiselect(
    "Monirtor:", all_EarlyStop_Monitor,
default=["val_loss"])

patience_desire = st.slider("Patient", 0, 100, 10)
min_improving_Delta = st.slider(
    "Minimum of improvement to triger the patient",
0.0, 1.0, 0.005)

mode_desire = st.multiselect(
    "Want to Max or Min the Metric Chosen?", ["max",
"min"], "min")

st.title("TUNER -> RandomSearch - Bayes - Hyperband")

all_objectives_Hyperband = [
    "val_loss",
    "loss",
    "val_accuracy",
    "accuracy",
    "val_binary_accuracy",
    "binary_accuracy",
    "val_categorical_accuracy",
    "categorical_accuracy",
    "val_sparse_categorical_accuracy",
    "sparse_categorical_accuracy",
    "val_mean_squared_error",
    "mean_squared_error",
    "val_AUC",   # if you"ve defined an AUC metric
    "auc"        # some people define it for the
```

```
training set as well
    # and so on, depending on any custom or built-in
metrics you log
]

objective_Hyperband_desire = st.multiselect(
    "Select the Objective of the TUNER",
all_objectives_Hyperband, "val_AUC")


tuner_decision = st.multiselect("RandomSearch - Bayes
- Hyperband for the TUNER", ["RandomSearch", "Bayes",
"Hyperband"],
                                "RandomSearch")


max_trials_decision = st.slider(
    "MAX. TRIAL FOR RANDOMSEARCH - BAYES ", 1, 1000,
1)


executions_per_trial_desire = st.slider(
    "Executions per trial PER unit: ", 1, 100, 1)


st.title("TUNER.SEARCH")

max_epochs_Desire = st.slider("Max Epochs to train",
1, 1000, 100)

factor_desire = st.slider(
    "Factor of the Hyperband (if 3, It will get 1/3 of
the best hyperparameters):", 1, 20, 3)

hyperband_iterations_desire = st.slider(
    "Number of times repeting the factor
desire(hyperband-iterations)[More computation the
bigger]", 1, 100, 3)


batch_size_train_desire = st.slider(
    "Batch Size of the Train Data", 1, 5000, 500)
validation_batch_size_desire = st.slider(
    "Batch Size of the Validation Data", 1, 5000, 500)
steps_per_epoch_Train_desire = st.slider(
    "Epoch in Train Data", 1, 1000, 20)
steps_per_epoch_validation_desire = st.slider(
    "Epoch in Validation Data", 1, 1000, 20)

# Convert to non list the one lenghts outputs
for variable_name, value in globals().copy().items():

    # print(variable_name, value)

    if isinstance(value, list) and len(value) == 1:

        globals()[variable_name] = value[0]
```

```
print(min_layer_value, type(min_layer_value))

# $$$$$$$$$$$$$$$$$

def build_model(hp):

    model = keras.Sequential()


model.add(layers.InputLayer(input_shape=(input_shape_d
ata,)))

    if batch_normalization_decision_before_neurons ==
"Yes":  # You decide if BachNormalization before
Nuerons
        model.add(BatchNormalization())

    if descending_layers == "Yes":  # Sometimes do not
work properly due to differences between randomnes of
the units + randomnes of the library random

        descending = []

        for i in range(hp.Int("num_layers",
min_number_neurons, max_number_neurons)):

            if i == 0:

                model.add(
                    layers.Dense(
                        # Gives randomly a number of
units for each neuron
                        units=hp.Int(f"units_{i}",
min_value=min_layer_value,

max_value=max_layer_value, step=step_layer_value),
                        activation=hp.Choice(
                            "activation",
activation_function_layer),

                    )
                )

                if
batch_normalization_decision_between_neurons == "Yes":
                    model.add(BatchNormalization())

                else:
                    pass

                descending.append(random.randrange(
                    min_layer_value, max_layer_value +
1, step_layer_value))

            else:
```

```
                if len(descending) == 1:

                    max_value_input = descending[0]

                    print(descending)

                elif len(descending) > 1:

                    max_value_input = min(descending)

                else:
                    pass

                model.add(
                    layers.Dense(
                        # Gives randomly a number of
units for each neuron
                        units=hp.Int(f"units_{i}",
min_value=min_layer_value,

max_value=max_value_input, step=step_layer_value),
                        activation=hp.Choice(
                            "activation",
activation_function_layer),
                        )
                )

                if
batch_normalization_decision_between_neurons == "Yes":
                    model.add(BatchNormalization())

                else:

                    pass

                descending.append(random.randrange(
                    min_layer_value, max_layer_value +
1, step_layer_value))

    if ascending_layers == "Yes":

        ascending = []
        for i in range(hp.Int("num_layers",
min_number_neurons, max_number_neurons)):

            if i == 0:

                model.add(
                    layers.Dense(
                        # Gives randomly a number of
units for each neuron
                        units=hp.Int(f"units_{i}",
min_value=min_layer_value,

max_value=max_layer_value, step=step_layer_value),
```

```
                        activation=hp.Choice(
                            "activation",
activation_function_layer)
                        )
                    )

                    if
batch_normalization_decision_between_neurons == "Yes":
                        model.add(BatchNormalization())

                    ascending.append(random.randrange(
                        min_layer_value, max_layer_value +
1, step_layer_value))

                    print(ascending,
"HEEEEEEEEEEEEELLOOOOOOOOO WORLD")

                else:

                    model.add(
                        layers.Dense(
                            # Gives randomly a number of
units for each neuron
                            units=hp.Int(f"units_{i}",
min_value=sum(ascending),
                                         # El segundo Max
value es el max value prefijado anteriormente

max_value=(sum(ascending) + max_layer_value),
step=step_layer_value),
                            activation=hp.Choice(
                                "activation",
activation_function_layer)
                        )
                    )

                    ascending.append(random.randrange(
                        min_layer_value, max_layer_value +
1, step_layer_value))

                    print(ascending,
"HEEEEEEEEEEEEELLOOOOOOOOO WORLD")

                    if
batch_normalization_decision_between_neurons == "Yes":
                        model.add(BatchNormalization())

        if ascending_layers == "No" or descending_layers
== "No":

            # Tune the number of layers.
            for i in range(hp.Int("num_layers",
min_number_neurons, max_number_neurons)):  # Number of
Neurons

                model.add(
                    layers.Dense(
```

```
                    # Gives randomly a number of units
for each neuron
                    units=hp.Int(f"units_{i}",
min_value=min_layer_value,

max_value=max_layer_value, step=step_layer_value),
                    activation=hp.Choice(
                        "activation",
activation_function_layer),
                )
            )

            # You decide if BachNormalization between
Nuerons
            if
batch_normalization_decision_between_neurons == "Yes":
                model.add(BatchNormalization())

    # Output link to the target
    model.add(layers.Dense(2,
activation=activation_function_last_neuron))

    if set_Normal_Random_Weights_beforehand == "Yes":
        # Random weights following a normal
distribution
        for layer in model.layers:
            if hasattr(layer, "kernel_initializer"):
                layer.kernel_initializer =
tf.keras.initializers.RandomNormal(
                    mean=Normal_Distribution_Mean,
stddev=Normal_Distribution_Sd)

    # When choosing a learning_rate, it will choose it
based on a logarithmic function
    learning_rate = hp.Float("lr",
min_value=Learning_Rate_Min_Value,

max_value=Learning_Rate_Max_Value, sampling="log")

    optimizer_name = hp.Choice(
        "optimizer", optimizers_neural_network)

    if optimizer_name == "adam":
        opt =
keras.optimizers.Adam(learning_rate=learning_rate)
    elif optimizer_name == "sgd":
        opt =
keras.optimizers.SGD(learning_rate=learning_rate)
    elif optimizer_name == "rmsprop":
        opt =
keras.optimizers.RMSprop(learning_rate=learning_rate)
    elif optimizer_name == "adagrad":
        opt =
keras.optimizers.Adagrad(learning_rate=learning_rate)
    elif optimizer_name == "adadelta":
        opt =
keras.optimizers.Adadelta(learning_rate=learning_rate)
```

```
    elif optimizer_name == "nadam":
        opt =
keras.optimizers.Nadam(learning_rate=learning_rate)
    elif optimizer_name == "ftrl":
        opt =
keras.optimizers.Ftrl(learning_rate=learning_rate)
    else:
        raise ValueError("Optimizer not recognized")

    model.compile(
        optimizer=opt,
        loss=loss_to_use,
        metrics=[metric_to_use]
    )

    return model


# Load the data Spliter dataset for Working the Neural
Network
data =
np.load("C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/Python
Scripts/ORDENADO/Ordenado
Master/TEST_TRAIN_VAL_SPLITTER_DATA/SPLITTERDATA.npz")

y_train = data["y_train"]
y_test = data["y_test"]
y_val = data["y_val"]
X_train = data["X_train"]
X_test = data["X_test"]
X_val = data["X_val"]
# $$$$

if st.sidebar.button("Execute Hyperband Tunning"):

    # Build the model with hyperparameters
    build_model(keras_tuner.HyperParameters())

    early_stopping = keras.callbacks.EarlyStopping(

        monitor=early_stopping_monitor,  # val_loss ->
mode = "min"
        patience=patience_desire,
        # Waht considers improving for not to trigger
the patience
        min_delta=min_improving_Delta,
        restore_best_weights=True,
        mode=mode_desire
        # verbose=1
        # baseline=0.7

    )

    ############
    max_epochs_Desire
    factor_desire, hyperband_iterations_desire,
```

```
executions_per_trial_desire

    if tuner_decision == "Hyperband":

        print("$$$$$$$ HYPERBAND FOR THE TUNER
$$$$$$$")

        tuner = Hyperband(
            hypermodel=build_model,
            objective=[objective_Hyperband_desire],
            # max number of epochs to train any given
set of hyperparameters
            max_epochs=max_epochs_Desire,
            factor=factor_desire,
            # reduction factor for Hyperband

hyperband_iterations=hyperband_iterations_desire,

executions_per_trial=executions_per_trial_desire,
            overwrite=True,  # True           #
Overwrite the previous results in directory
            directory="Tuner_Directory",  # Name of
the folder inside C:\Users\vmeri
            project_name="Hyperband",  # Name of the
folder inside C:\Users\vmeri\my_dir
        )

    elif tuner_decision == "RandomSearch":

        print("$$$$$$$ RANDOMSEARCH FOR THE TUNER
$$$$$$$")

        tuner = keras_tuner.RandomSearch(
            hypermodel=build_model,
            objective=[objective_Hyperband_desire],
            max_trials=max_trials_decision,

executions_per_trial=executions_per_trial_desire,
            overwrite=True,  # Overwrite the previous
results in directory
            directory="Tuner_Directory",
            project_name="Random_Search",
        )

    else:

        print("$$$$$$$ BAYES OPTIMIZATION FOR THE
TUNER $$$$$$$")

        tuner = BayesianOptimization(
            hypermodel=build_model,
            objective="AUC",
            max_trials=max_trials_decision,

executions_per_trial=executions_per_trial_desire,
            directory="Tuner_Directory",
            project_name="Bayes",
```

```
            overwrite=True
        )

    tuner.search(
        X_train,
        y_train,
        epochs=max_epochs_Desire,
        batch_size=batch_size_train_desire,
        validation_data=(X_val, y_val),
        callbacks=[early_stopping],
        steps_per_epoch=steps_per_epoch_Train_desire,

validation_steps=steps_per_epoch_validation_desire,
validation_batch_size=validation_batch_size_desire
    )
```

4. LONGITUD – LATITUD DEL TRACTO CENSAL

```
# Se han descargado las coordenadas de latitud y
longitud de las circunscripciones de Condado de Lee

with open("C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/USA
COUNTY/CenPop2020_Mean_TR12.txt", "r") as txt_file:
    lines = txt_file.readlines()


# Leer el archivo txt (ajusta el delimitador si es
necesario, por ejemplo "\t" para tabulaciones)
df = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/USA
COUNTY/supreme_FLcensusTract.csv")


df_CensusTract = df.drop(["LATITUDE", "LONGITUDE"],
axis=1)

# Convertir cada valor en string y concatenarlos
df_CensusTract["censusTract"] =
df_CensusTract.astype(str).agg("".join, axis=1)

# Convertir el resultado concatenado a un número
df_CensusTract["censusTract"] =
df_CensusTract["censusTract"].astype(int)


# Mostrar el DataFrame resultante
print(df)


df.to_csv("CensusTractFL_geolocation.csv",
index=False)


df_fema = pd.merge(df_all_records, df_fema, on=[
```

```
    "longitude", "latitude", "Start_date",
"End_date"], how="outer")  # FINAL


filas_filtradas = filas_filtradas.drop(["latitude",
"longitude"], axis=1)


zdf_one = pd.merge(df, filas_filtradas, on=[
    "censusTract"], how="inner")  # FINAL
```

5. MAPA COORDENADAS ASEGURADOS

```
from geopy.distance import geodesic
import folium
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, metrics,
optimizers
from tensorflow.keras.layers import Dense,
BatchNormalization, Dropout, Input, Lambda
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import GlorotNormal,
Zeros
from tensorflow.keras import regularizers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
import random
from tensorflow.keras.initializers import (Zeros, Ones,
RandomNormal, RandomUniform,

TruncatedNormal, Constant, GlorotNormal, GlorotUniform,
                                          HeNormal,
HeUniform, LecunNormal, LecunUniform)

df_12071 = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/bbdd 5
mayo/12071/virgen/12071_BBDD_HURACANES_menor_50000_MAX.
csv")

coordenadas_12071 = df_12071[["longitude", "latitude"]]

# Eliminamos coordenadas duplicadas
coordenadas_12071 = coordenadas_12071.drop_duplicates()


# Crear un mapa centrado en el promedio de las
coordenadas
centro_lat = coordenadas_12071["latitude"].mean()
centro_lon = coordenadas_12071["longitude"].mean()
```

```
mapa = folium.Map(location=[centro_lat, centro_lon],
zoom_start=12)

# Añadir marcadores para cada coordenada
for idx, row in coordenadas_12071.iterrows():
    folium.Marker(
        location=[row["latitude"], row["longitude"]],
        popup=f"Punto {idx + 1}",  # Etiqueta para cada
marcador
        icon=folium.Icon(color="blue")
    ).add_to(mapa)


# Guardar el mapa como archivo HTML
mapa.save("mapa_coordenadas66.html")


# Distancia asegurados - huracan

coordenadas_12071_mas_huracan = df_12071[["longitude",
"latitude", "Max_Intensity_Hurricane_Latitude",

"Max_Intensity_Hurricane_Longitude",
"Second_Max_Intensity_Hurricane_Latitude",
"Second_Max_Intensity_Hurricane_Longitude"]]

# Eliminamos coordenadas duplicadas
coordenadas_12071_mas_huracan =
coordenadas_12071_mas_huracan.drop_duplicates()


# Función para calcular la distancia entre dos puntos
(lat, lon)
def calcular_distancia(punto1, punto2):
    return geodesic(punto1, punto2).kilometers


# Crear un mapa centrado en el promedio de las
coordenadas porque sino no funciona el mapeo
centro_lat =
coordenadas_12071_mas_huracan["latitude"].mean()
centro_lon =
coordenadas_12071_mas_huracan["longitude"].mean()
mapa = folium.Map(location=[centro_lat, centro_lon],
zoom_start=12)


for idx, row in
coordenadas_12071_mas_huracan.iterrows():
    # Latitud - Longitud Asegurado
    folium.Marker(
        location=[row["latitude"], row["longitude"]],
        popup=f"Punto {idx + 1}",
        icon=folium.Icon(color="blue")
    ).add_to(mapa)

    # Latitud - Longitud Max Intensity huracan
```

```
    folium.Marker(

location=[row["Max_Intensity_Hurricane_Latitude"],

row["Max_Intensity_Hurricane_Longitude"]],
        popup=f"Max Intensity {idx + 1}",
        icon=folium.Icon(color="red")
    ).add_to(mapa)

    # Latitud - Longitud Second Max Intensity huracan
    folium.Marker(

location=[row["Second_Max_Intensity_Hurricane_Latitude"
],

row["Second_Max_Intensity_Hurricane_Longitude"]],
        popup=f"Second Max Intensity {idx + 1}",
        icon=folium.Icon(color="green")
    ).add_to(mapa)

    # Calcular distancias
    punto_actual = (row["latitude"], row["longitude"])
    punto_max_intensity = (
        row["Max_Intensity_Hurricane_Latitude"],
row["Max_Intensity_Hurricane_Longitude"])
    punto_second_max_intensity = (
        row["Second_Max_Intensity_Hurricane_Latitude"],
row["Second_Max_Intensity_Hurricane_Longitude"])

    distancia_max = calcular_distancia(punto_actual,
punto_max_intensity)
    distancia_second_max = calcular_distancia(
        punto_actual, punto_second_max_intensity)

    # Dibujar línea entre asegurado y Max Intensity
    folium.PolyLine(
        locations=[punto_actual, punto_max_intensity],
        color="red",
        weight=2,
        popup=f"Distancia a Max Intensity:
{distancia_max:.2f} km"
    ).add_to(mapa)

    # Dibujar línea entre asegurado y Second Max
Intensity}\
    folium.PolyLine(
        locations=[punto_actual,
punto_second_max_intensity],
        color="green",
        weight=2,
        popup=f"Distancia a Second Max Intensity: {
            distancia_second_max:.2f} km"
    ).add_to(mapa)

# Guardar el mapa como archivo HTML
mapa.save("mapa_con_lineas.html")
```

6. AJUSTE FUNCION AL HISTORICO

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import pandas as pd

# Cargar los datos
datos_12071 = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/bbdd 5
mayo/12071/Param Deseados/Script OWN
Function/Ratio_Continente_Pago.csv")

# Convertir los datos a un array de numpy
datos_12071_fit = datos_12071.to_numpy()

# Crear un rango de datos para x y los datos
correspondientes para y

# Datos cuyas indemnizaciones no son ni 0 ni 50.000 del
historico
x_data = np.arange(922, 1713).reshape(-1)
y_data = datos_12071_fit.reshape(-1)


# Funcion Polinomica de grado 4
def modelo_grado4(x, a, b, c, d, e):
    return a * x**4 + b * x**3 + c * x**2 + d * x + e


# Ajustar el modelo
# Lo ajusta por Maximum Likelihood
parametros, _ = curve_fit(modelo_grado4, x_data,
y_data)
a, b, c, d, e = parametros

# Mostrar todos los decimales sin notación científica
np.set_printoptions(suppress=True, precision=15)
print(parametros)

# Generar datos ajustados
x_fit = np.linspace(min(x_data), max(x_data), 200)
y_fit = modelo_grado4(x_fit, a, b, c, d, e)

# Crear fórmula en string ; POLINOMIO DE GRADO 4
# formula = f"y = {a:.2e}x⁴ + {b:.2e}x³ + {c:.2e}x² +
{d:.2e}x + {e:.2e}"
# Graficar
plt.scatter(x_data, y_data, color="blue",
label="Indemnizaciones Históricas")
plt.plot(x_fit, y_fit, color="red")
plt.legend()
# plt.title("Ajuste Polinomial de Grado 4")
```

```
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.show()
```

7. MODELO GENERAL LINEAL (GLM) Y METRICAS

```
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
import numpy as np
import pandas as pd
import numpy as np


URL_NOMilton = "https://raw.githubusercontent.com/UC3M-
student/NOMILTON/refs/heads/main/NOMILTON.csv"
df_NOMilton = pd.read_csv(URL_NOMilton)

df_Milton = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/bbdd 5
mayo/12071/Param Deseados/Script OWN
Function/MILTON/MILTON.csv")

X1 = df_Milton.drop("Ratio_Continente_Pago", axis=1)
Y1 = df_Milton["Ratio_Continente_Pago"]

# Elegir de que quieres que dependa
X = df_NOMilton.drop("Ratio_Continente_Pago", axis=1)
Y = df_NOMilton["Ratio_Continente_Pago"]

# Add a constant to the independent variables (for the
intercept in the regression)
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(Y, X)
results = model.fit()

# Display the summary
print(results.summary())


# Realizamos las predicciones con X_test
X1 = sm.add_constant(X1, has_constant="add")  #
Reasigna el resultado
y_pred = results.predict(X1)

# Calculamos las métricas; y_test1 corresponde a las
indemnizaciones reales
r2 = r2_score(y_test1, y_pred)
```

```
evs = explained_variance_score(y_test1, y_pred)
mse = mean_squared_error(y_test1, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test1, y_pred)

valores_coincidentes = np.sum(y_pred ==
y_test1.astype(int))
```

8. FASE 1 – 2 – 3 DE LA RED NEURONAL

```
# Imports
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, metrics,
optimizers
from tensorflow.keras.layers import Dense,
BatchNormalization, Dropout, Input, Lambda
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.initializers import GlorotNormal,
Zeros
from tensorflow.keras import regularizers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
import random
from tensorflow.keras.initializers import (Zeros, Ones,
RandomNormal, RandomUniform,

TruncatedNormal, Constant, GlorotNormal, GlorotUniform,
                                          HeNormal,
HeUniform, LecunNormal, LecunUniform)


from sklearn.metrics import explained_variance_score
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score


kernel_inicializadores = [GlorotNormal(),
GlorotUniform(),
                          HeNormal(), HeUniform(),
LecunNormal(), LecunUniform()]

batch_size_posibilidades = [16, 32, 64]

loss_posibilidades = ["mean_squared_error",
"mean_absolute_error", "log_cosh"]


def create_dataset(X, y, batch_size, shuffle=False):
    dataset = tf.data.Dataset.from_tensor_slices((X,
y))
```

```
    if shuffle:
        # Cambian 1700 registros de forma aleatoria
        dataset = dataset.shuffle(buffer_size=1700)
    dataset =
dataset.batch(batch_size).cache().prefetch(tf.data.AUTO
TUNE)
    return dataset


def piecewise_function(x):  # funcion a trozos ajustada
al historico de indemnizaciones
    part1 = tf.where(x < 921, tf.constant(0.0,
dtype=tf.float32),
                     tf.constant(0.0,
dtype=tf.float32))
    part2 = tf.where((x >= 922) & (x <= 1712),
                     -0.000000167446211 *
tf.math.pow(x, 4) +
                     0.000922142109855 * tf.math.pow(x,
3) +
                     -1.797998779184797 *
tf.math.pow(x, 2) +
                     1522.7422060756344 * x +
                     -477264.731371976,
                     tf.constant(0.0))
    part3 = tf.where(x > 1712, tf.constant(
        50000.0, dtype=tf.float32), tf.constant(0.0))

    return part1 + part2 + part3


def custom_activation(x):
    x = x * x
    sigmoid = tf.keras.activations.sigmoid(x)
    scaled = 1.0 + 1744.0 * sigmoid
    return tf.clip_by_value(scaled, 1.0, 1744.0)


def splitter_data(df_desire, peso1, peso2):
    X = df_desire.drop(columns="Ratio_Continente_Pago")
    y = df_desire["Ratio_Continente_Pago"]

    X_train, X_temp, y_train, y_temp =
train_test_split(
        X, y, test_size=peso1, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=peso2,
random_state=42)

    y_train =
y_train.to_numpy(dtype=np.float64).reshape(-1, 1)
    y_test =
y_test.to_numpy(dtype=np.float64).reshape(-1, 1)
    y_val = y_val.to_numpy(dtype=np.float64).reshape(-
1, 1)
    X_train =
X_train.to_numpy(dtype=np.float64).reshape(-1, 23)
```

```
    X_test =
X_test.to_numpy(dtype=np.float64).reshape(-1, 23)
    X_val = X_val.to_numpy(dtype=np.float64).reshape(-
1, 23)

    print("LOS OUTPUTS SALEN COMO: y_train, y_test,
y_val, X_train, X_test, X_val")
    return y_train, y_test, y_val, X_train, X_test,
X_val


# Load data
df_NOMilton = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/bbdd 5
mayo/12071/Param Deseados/Script OWN
Function/NOMILTON/NOMILTON.csv")

ddbb_NOMilton = splitter_data(df_NOMilton, 0.2, 0.01)
# 80 train y 20 test

y_train = ddbb_NOMilton[0]
y_val = ddbb_NOMilton[2]
X_train = ddbb_NOMilton[3]
X_val = ddbb_NOMilton[5]

# usar Milton2?
df_Milton = pd.read_csv(
    "C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER
UC3M/TFM/TFM FINAL - Parametrics Insurence/bbdd 5
mayo/12071/Param Deseados/Script OWN
Function/MILTON/MILTON.csv")

ddbb_Milton = splitter_data(df_Milton, 0.02, 0.01)
y_test1 = ddbb_Milton[0]
X_test1 = ddbb_Milton[3]

indemnizacion_maxima = X_test1[:, 17]
indemnizacion_maxima = indemnizacion_maxima.reshape(-1,
1)


# Estandarizar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test1_scaled = scaler.transform(X_test1)


# Set random seed
seed = 100
np.random.seed(seed)
tf.random.set_seed(seed)

# Model building
```

```
def build_model(
    input_shape=12,
    optimizer_name="adam",
    activation_function="relu",
    loss_function="mean_squared_error",
    metrics=["mean_squared_error", "mae"],
    learning_rate=0.015,
    neuron_structure=[16, 8, 4],
    dropout_rate=0.1,
    l2_lambda=0.01,
    kernel_initializer=None
):
    model = Sequential()
    model.add(Input(shape=(input_shape,)))

    for neurons in neuron_structure:
        model.add(Dense(
            units=neurons,
            activation=activation_function,
            kernel_initializer=kernel_initializer,
            bias_initializer=Zeros(),
            #
kernel_regularizer=regularizers.l2(l2_lambda)
        ))
        model.add(BatchNormalization())
        model.add(Dropout(dropout_rate))

    model.add(Dense(1))  # Output lineal
    model.add(Lambda(custom_activation))
    model.add(Lambda(piecewise_function))

    optimizer_dict = {
        "adam":
optimizers.Adam(learning_rate=learning_rate),
        "rmsprop":
optimizers.RMSprop(learning_rate=learning_rate),
        "nadam":
optimizers.Nadam(learning_rate=learning_rate),
        "adamax":
optimizers.Adamax(learning_rate=learning_rate),
        "lion":
optimizers.Lion(learning_rate=learning_rate),
        "adafactor":
optimizers.Adafactor(learning_rate=learning_rate),
        "sgd":
optimizers.SGD(learning_rate=learning_rate),
    }

    optimizer = optimizer_dict[optimizer_name.lower()]

    model.compile(
        optimizer=optimizer,
        loss=loss_function,
        metrics=metrics
    )

    return model
```

```
def train_and_evaluate_model(
    optimizer="adam",
    activation_function="relu",
    loss_function="mean_squared_error",
    metrics=["mse", "mae"],
    learning_rate=0.001,
    neuron_structure=[64],
    train_dataset=None,
    val_dataset=None,
    test_dataset=None,
    early_stopping=None,
    epochs=100,
    dropout_rate=0.05,
    l2_lambda=0.01,
    kernel_initializer=None,
    input_shape=23  # Añadido para especificar la forma
de entrada
):
    model = build_model(
        input_shape=input_shape,
        optimizer_name=optimizer,
        activation_function=activation_function,
        loss_function=loss_function,
        metrics=metrics,
        learning_rate=learning_rate,
        neuron_structure=neuron_structure,
        dropout_rate=dropout_rate,
        l2_lambda=l2_lambda,
        kernel_initializer=kernel_initializer
    )

    # Entrenar el modelo con datasets
    history = model.fit(
        train_dataset,
        epochs=epochs,
        validation_data=val_dataset,
        callbacks=[early_stopping] if early_stopping
else [],
        verbose=0
    )

    # Predecir usando el test_dataset
    y_pred = model.predict(test_dataset)

    # Convertir y_test1 a NumPy para calculo métricas
    y_test = y_test1

    y_pred = np.minimum(y_pred, indemnizacion_maxima)

    r2 = r2_score(y_test, y_pred)

    # Puntuación de varianza explicada
    evs = explained_variance_score(y_test, y_pred)

    mse = mean_squared_error(y_test, y_pred)
```

```
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)

    return r2, y_pred, evs, mse, rmse, mae


# Early stopping
early_stopping = EarlyStopping(
    monitor="val_mean_squared_error", patience=20,
restore_best_weights=True)

# Results DataFrame
df_soluciones = pd.DataFrame(columns=[
    "Neurona 1", "Neurona 2", "Neurona 3",
"optimizador_usar", "func_activacion_usar",
"learning_rate_usar",
    "R2", "EVS", "DIFERENCIA", "ACIERTOS",
"Kernel_Inicializador", "Bach_Size", "loss", "MSE",
"RMSE", "MAE"])

# Training loop
for i in range(500):

    batch_size_seleccionado =
random.choice(batch_size_posibilidades)
    kernel_inicial_pesos_seleccionado =
random.choice(kernel_inicializadores)
    loss_posibilidades_seleccionado =
random.choice(loss_posibilidades)

    # Create datasets
    train_dataset = create_dataset(
        X_train_scaled, y_train,
batch_size=batch_size_seleccionado, shuffle=True)
    val_dataset = create_dataset(
        X_val_scaled, y_val,
batch_size=batch_size_seleccionado)
    test_dataset = create_dataset(
        X_test1_scaled, y_test1,
batch_size=batch_size_seleccionado)

    neurona1 = random.randint(2, 32)
    neurona2 = random.randint(2, 32)
    neurona3 = random.randint(2, 32)

    optimizador_usar = np.random.choice([
        "adam",
        "rmsprop",
        "nadam",
        "adamax",
        "lion",
        "adafactor",
        "sgd"
    ])

    # Funcion activacion selecionada aleatoriamente
    func_activacion_usar = np.random.choice([
```

```
        "linear",
        "relu",
        "swish",
    ])

    learning_rate_usar = random.uniform(0.0001, 0.2)

    r2, y_pred, evs, mse, rmse, mae =
train_and_evaluate_model(
        optimizer=optimizador_usar,
        activation_function=func_activacion_usar,
        loss_function=loss_posibilidades_seleccionado,
        metrics=["mean_squared_error", "mae"],
        learning_rate=learning_rate_usar,
        neuron_structure=[neurona1, neurona2],
        train_dataset=train_dataset,
        val_dataset=val_dataset,
        test_dataset=test_dataset,
        early_stopping=early_stopping,
        epochs=500,
        dropout_rate=0.01,
        l2_lambda=0,

kernel_initializer=kernel_inicial_pesos_seleccionado,
        input_shape=X_train.shape[1]
    )

    diferenciaRealPredicho = abs(937999 - sum(y_pred))

    diferencia_porcentual_absoluta = abs((937999 -
sum(y_pred)) / (937999))

    aciertos = np.round(y_pred, decimals=1) ==
np.round(y_test1, decimals=1)
    aciertos = np.sum(aciertos)

    df_soluciones.loc[i] = [
        neurona1, neurona2, neurona3, optimizador_usar,
func_activacion_usar, learning_rate_usar,
        r2, evs, diferenciaRealPredicho, aciertos,
kernel_inicial_pesos_seleccionado,
batch_size_seleccionado,
        loss_posibilidades_seleccionado, mse, rmse, mae
    ]

    print("SE HA COMPLEATADO LA ESTRUCTURA NUMERO : ",
i + 1)
```

9. FASE 4 - RED NEURONAL

```
import gc
import keras.backend as K
from datetime import datetime
from sklearn.metrics import r2_score,
explained_variance_score, mean_squared_error,
```

```
mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras.initializers import Zeros, HeUniform,
GlorotNormal, GlorotUniform, HeNormal, LecunNormal,
LecunUniform
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization,
Dropout, Input, Lambda
from keras import layers, models, metrics, optimizers
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random
import os
import sys
sys.stdout.reconfigure(encoding="utf-8")


def create_dataset(X, y, batch_size, shuffle=False):
    """Crea un dataset de TensorFlow a partir de datos
X e y."""
    dataset = tf.data.Dataset.from_tensor_slices((X,
y))
    if shuffle:
        dataset = dataset.shuffle(buffer_size=1700)
    dataset =
dataset.batch(batch_size).cache().prefetch(tf.data.AUTO
TUNE)
    return dataset


def piecewise_function(x):
    """Función segmentada para procesar la salida del
modelo."""
    part1 = tf.where(x < 921, tf.constant(0.0,
dtype=tf.float32),
                     tf.constant(0.0,
dtype=tf.float32))
    part2 = tf.where((x >= 922) & (x <= 1712),
                     -0.000000167446211 *
tf.math.pow(x, 4) +
                     0.000922142109855 * tf.math.pow(x,
3) +
                     -1.797998779184797 *
tf.math.pow(x, 2) +
                     1522.7422060756344 * x +
                     -477264.731371976,
                     tf.constant(0.0))
    part3 = tf.where(x > 1712, tf.constant(
        50000.0, dtype=tf.float32), tf.constant(0.0))
    return part1 + part2 + part3
```

```
def custom_activation(x):
    """Función de activación personalizada con sigmoide
escalada."""

    x = x * x
    sigmoid = tf.keras.activations.sigmoid(x)
    scaled = 1.0 + 1744.0 * sigmoid
    return tf.clip_by_value(scaled, 1.0, 1744.0)


def splitter_data(df_desire, peso1, peso2):
    """Divide los datos en conjuntos de entrenamiento,
validación y prueba."""
    X = df_desire.drop(columns="Ratio_Continente_Pago")
    y = df_desire["Ratio_Continente_Pago"]
    X_train, X_temp, y_train, y_temp =
train_test_split(
        X, y, test_size=peso1, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=peso2,
random_state=42)
    y_train =
y_train.to_numpy(dtype=np.float64).reshape(-1, 1)
    y_test =
y_test.to_numpy(dtype=np.float64).reshape(-1, 1)
    y_val = y_val.to_numpy(dtype=np.float64).reshape(-
1, 1)
    X_train =
X_train.to_numpy(dtype=np.float64).reshape(-1, 23)
    X_test =
X_test.to_numpy(dtype=np.float64).reshape(-1, 23)
    X_val = X_val.to_numpy(dtype=np.float64).reshape(-
1, 23)
    print("Outputs: y_train, y_test, y_val, X_train,
X_test, X_val")
    return y_train, y_test, y_val, X_train, X_test,
X_val


# Donde guardar los gráficos
OUTPUT_FOLDER =
"C:/Users/vmeri/OneDrive/ESTUDIOS/MASTER UC3M/TFM/TFM
FINAL - Parametrics Insurence/bbdd 5 mayo/12071/Param
Deseados/Script OWN Function/Graficos Fase 4"

URL_NOMilton = "https://raw.githubusercontent.com/UC3M-
student/NOMILTON/refs/heads/main/NOMILTON.csv"
df_no_milton = pd.read_csv(URL_NOMilton)

ddbb_no_milton = splitter_data(df_no_milton, 0.2, 0.01)
y_train = ddbb_no_milton[0]
y_val = ddbb_no_milton[2]
X_train = ddbb_no_milton[3]
X_val = ddbb_no_milton[5]

URL_Milton = "https://raw.githubusercontent.com/UC3M-
student/MILTON2.2/refs/heads/main/MILTON2.2.csv"  #
```

```
MILTON 2.2
df_milton = pd.read_csv(URL_Milton)

ddbb_milton = splitter_data(df_milton, 0.02, 0.01)
y_test1 = ddbb_milton[0]
X_test1 = ddbb_milton[3]
indemnizacion_maxima = X_test1[:, 17].reshape(-1, 1)
y_test1 = np.ravel(y_test1)


# Historico contador indemnizacion 0 aplicado a Milton
Sumas Aseguradas
df_milton_historico_contador_cero =
"https://raw.githubusercontent.com/UC3M-
student/Contador_Cero/refs/heads/main/Milton_casos_cero
.csv"
df_milton_historico_contador_cero = pd.read_csv(
    df_milton_historico_contador_cero)

# Estandarizar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test1_scaled = scaler.transform(X_test1)

# Establecer semilla para reproducibilidad
KERNEL_INITIALIZERS = [HeUniform()]
BATCH_SIZE_OPTIONS = [64]
LOSS_FUNCTIONS = ["mean_squared_error"]
SEED = 338

np.random.seed(SEED)
tf.random.set_seed(SEED)


def build_model(
    input_shape=23,
    optimizer_name="adam",
    activation_function="relu",
    loss_function="mean_squared_error",
    metrics=["mean_squared_error", "mae"],
    learning_rate=0.015,
    neuron_structure=[16, 8, 4],
    dropout_rate=0.1,
    l2_lambda=0.01,
    kernel_initializer=None
):
    model = Sequential()
    model.add(Input(shape=(input_shape,)))
    for neurons in neuron_structure:
        model.add(Dense(
            units=neurons,
            activation=activation_function,
            kernel_initializer=kernel_initializer,
            bias_initializer=Zeros()
        ))
        model.add(BatchNormalization())
```

```
            model.add(Dropout(dropout_rate))
        model.add(Dense(1))
        model.add(Lambda(custom_activation))
        model.add(Lambda(piecewise_function))
        optimizer_dict = {
            "adam":
optimizers.Adam(learning_rate=learning_rate),
            "rmsprop":
optimizers.RMSprop(learning_rate=learning_rate),
            "nadam":
optimizers.Nadam(learning_rate=learning_rate),
            "adamax":
optimizers.Adamax(learning_rate=learning_rate),
            "lion":
optimizers.Lion(learning_rate=learning_rate),
            "adafactor":
optimizers.Adafactor(learning_rate=learning_rate),
            "sgd":
optimizers.SGD(learning_rate=learning_rate),
        }
        optimizer = optimizer_dict[optimizer_name.lower()]
        model.compile(optimizer=optimizer,
loss=loss_function, metrics=metrics)
        return model


def train_and_evaluate_model(
        optimizer="adam",
        activation_function="relu",
        loss_function="mean_squared_error",
        metrics=["mse", "mae"],
        learning_rate=0.001,
        neuron_structure=[64],
        train_dataset=None,
        val_dataset=None,
        test_dataset=None,
        early_stopping=None,
        epochs=100,
        dropout_rate=0.05,
        l2_lambda=0.01,
        kernel_initializer=None,
        input_shape=23,
        indemnizacion_maxima=None
):
        model = build_model(
            input_shape=input_shape,
            optimizer_name=optimizer,
            activation_function=activation_function,
            loss_function=loss_function,
            metrics=metrics,
            learning_rate=learning_rate,
            neuron_structure=neuron_structure,
            dropout_rate=dropout_rate,
            l2_lambda=l2_lambda,
            kernel_initializer=kernel_initializer
        )
        history = model.fit(
```

```
        train_dataset,
        epochs=epochs,
        validation_data=val_dataset,
        callbacks=[early_stopping] if early_stopping
else [],
        verbose=0
    )

    y_pred = model.predict(test_dataset)
    y_test = y_test1

    # -- Parte Nueva - Historico 0 Milton
    y_pred = y_pred.ravel()
    indemnizacion_maxima = indemnizacion_maxima.ravel()

    # Crear el DataFrame
    df_pred = pd.DataFrame({
        "indemnizacion_maxima": indemnizacion_maxima,
        "y_pred": y_pred
    })

    # Creamos el nuevo DataFrame resultado
    df_resultado = df_pred.copy()

    for _, row in
df_milton_historico_contador_cero.iterrows():
        suma = row["suma_asegurada"]
        n_zeros = int(row["contador_cero"])

        # Seleciona los indices del dataframe cuyos
resultados sean los menores
        mask = df_resultado["indemnizacion_maxima"] ==
suma
        indices = df_resultado[mask].nsmallest(n_zeros,
"y_pred").index

        # Reemplazar esos valores por cero
        df_resultado.loc[indices, "y_pred"] = 0

    # Extraemos la prediccion ya que primera fila son
las sumas aseguradas
    df_resultado_pred = df_resultado.iloc[:, 1]
    df_resultado_pred = df_resultado_pred.to_numpy()  #
Lo convierte a array de NumPy

    # ----- FINAL PARTE NUEVA

    y_pred = np.minimum(df_resultado_pred,
indemnizacion_maxima)

    r2 = r2_score(y_test, y_pred)
    evs = explained_variance_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)

    return r2, y_pred, evs, mse, rmse, mae, history,
```

```
model, df_resultado


# === Configuración de entrenamiento ===
early_stopping = EarlyStopping(
    monitor="val_mean_squared_error", patience=15,
restore_best_weights=True)

# DataFrame para almacenar resultados
df_soluciones = pd.DataFrame(columns=[
    "Neurona 1", "Neurona 2", "Neurona 3",
"Optimizador", "Función Activación", "Learning Rate",
    "R2", "EVS", "Diferencia", "Aciertos", "Kernel
Inicializador", "Batch Size", "Loss", "MSE", "RMSE",
"MAE"
])

count_r2_mayor_0 = 0
lista_r2 = []

# === Bucle de entrenamiento ===
for i in range(1000):
    # Selección de hiperparámetros aleatorios
    batch_size = BATCH_SIZE_OPTIONS
    kernel_initializer = KERNEL_INITIALIZERS
    loss_function = LOSS_FUNCTIONS

    # Crea dataset nuevos siendo esto aleatorios para
cada i del bucle
    train_dataset = create_dataset(
        X_train_scaled, y_train, batch_size=batch_size,
shuffle=True)
    val_dataset = create_dataset(X_val_scaled, y_val,
batch_size=batch_size)
    test_dataset = create_dataset(
        X_test1_scaled, y_test1, batch_size=batch_size)

    # Definir estructura de la red neuronal
    neurona1 = 20  # random.randint(10, 20)
    neurona2 = 11  # random.randint(10, 20)
    neurona3 = 16  # random.randint(10, 20)
    optimizador = np.random.choice(["adam"])
    funcion_activacion = np.random.choice(["relu"])
    learning_rate = 0.089685527  #
random.uniform(0.0001, 0.2)

    # Entrenar y evaluar modelo
    r2, y_pred, evs, mse, rmse, mae, history, model,
df_resultado = train_and_evaluate_model(
        optimizer=optimizador,
        activation_function=funcion_activacion,
        loss_function=loss_function,
        metrics=["mean_squared_error", "mae"],
        learning_rate=learning_rate,
        neuron_structure=[neurona1, neurona2,
neurona3],
        train_dataset=train_dataset,
```

```
        val_dataset=val_dataset,
        test_dataset=test_dataset,
        early_stopping=early_stopping,
        epochs=500,
        dropout_rate=0.01,
        l2_lambda=0,
        kernel_initializer=kernel_initializer,
        input_shape=X_train.shape[1],
        indemnizacion_maxima=indemnizacion_maxima
    )

    if r2 > 0:
        count_r2_mayor_0 = count_r2_mayor_0 + 1
        lista_r2.append(round(r2, 3))

    print("MODELO SUPERIORRES A 0 ES DE: ",
count_r2_mayor_0,
          "LISTA R2 SUPERIOR A 0: ", lista_r2)

    # Guardar resultados si R2 es significativo
    if r2 > 0.4:
        diferencia = abs(937999 - sum(y_pred))
        aciertos = np.sum(np.round(y_pred, decimals=1)
==
                          np.round(y_test1,
decimals=1))
        df_soluciones.loc[i] = [
            neurona1, neurona2, neurona3, optimizador,
funcion_activacion, learning_rate,
            r2, evs, diferencia, aciertos,
kernel_initializer, batch_size, loss_function, mse,
rmse, mae
        ]
        print(f"R2 encontrado: {r2:.4f}")

        # === Generación de gráficos ===
        sns.set(style="whitegrid",
palette="colorblind", font_scale=1.2)
        plt.style.use("seaborn-v0_8")

        y_test1 = np.ravel(y_test1)

        # 1. Gráfico Predicho vs Real
        plt.figure(figsize=(8, 6), dpi=300)
        plt.scatter(y_test1, y_pred, alpha=0.7,
color=sns.color_palette("colorblind")[2],
                    edgecolors="white", linewidth=0.5,
s=80, label="Predicciones")
        plt.plot([y_test1.min(), y_test1.max()],
[y_test1.min(), y_test1.max()],
                 color="black", linestyle="--",
linewidth=2, label="Línea Ideal")
        plt.xlabel("Valores Reales", fontsize=12,
weight="bold")
        plt.ylabel("Valores Predichos", fontsize=12,
weight="bold")
        plt.title(f"Iteración {i}: Real vs.
```

```
Predicho\nR²={r2:.4f}, MSE={mse:.4f}, MAE={mae:.4f}",
                    fontsize=14, weight="bold", pad=15)
        plt.legend(frameon=True, fontsize=10,
loc="best", edgecolor="black")
        plt.grid(True, linestyle="--", alpha=0.7)
        sns.despine()
        plt.tight_layout()
        plt.savefig(os.path.join(OUTPUT_FOLDER,
f"grafico_real_vs_predicho_{
                    i}.png"), dpi=300,
bbox_inches="tight")
        plt.close()

        # 2. Gráfico Error vs Real
        errores = y_test1 - y_pred
        plt.figure(figsize=(8, 6), dpi=300)
        plt.scatter(y_test1, errores, alpha=0.7,
color=sns.color_palette("colorblind")[2],
                    edgecolors="white", linewidth=0.5,
s=80, label="Errores")
        plt.axhline(0,
color=sns.color_palette("colorblind")[
                    0], linestyle="--", linewidth=2,
label="Error = 0")
        plt.xlabel("Valores Reales", fontsize=12,
weight="bold")
        plt.ylabel("Error (Real - Predicho)",
fontsize=12, weight="bold")
        plt.title(f"Iteración {i}: Error vs.
Real\nRMSE={rmse:.4f}",
                  fontsize=14, weight="bold", pad=15)
        plt.legend(frameon=True, fontsize=10,
loc="best", edgecolor="black")
        plt.grid(True, linestyle="--", alpha=0.7)
        sns.despine()
        plt.tight_layout()
        plt.savefig(os.path.join(OUTPUT_FOLDER,
f"grafico_error_vs_real_{
                    i}.png"), dpi=300,
bbox_inches="tight")
        plt.close()

        # 3. Gráfico Predicho vs Real
        y_val_pred = model.predict(X_val_scaled)
        val_r2 = r2_score(y_val, y_val_pred)
        plt.figure(figsize=(8, 6), dpi=300)
        plt.scatter(y_val, y_val_pred, alpha=0.7,
color=sns.color_palette("colorblind")[2],
                    edgecolors="white", linewidth=0.5,
s=80, label="Predicciones")
        plt.plot([y_val.min(), y_val.max()],
[y_val.min(), y_val.max()],
                 color="black", linestyle="--",
linewidth=2, label="Línea Ideal")
        plt.xlabel("Valores Reales (Validación)",
fontsize=12, weight="bold")
```

```
        plt.ylabel("Valores Predichos", fontsize=12,
weight="bold")
        plt.title(f"Iteración {i}: Validación Real vs.
Predicho\nR²={val_r2:.4f}",
                  fontsize=14, weight="bold", pad=15)
        plt.legend(frameon=True, fontsize=10,
loc="best", edgecolor="black")
        plt.grid(True, linestyle="--", alpha=0.7)
        sns.despine()
        plt.tight_layout()
        plt.savefig(os.path.join(OUTPUT_FOLDER,
f"grafico_predicciones_validacion_{
                    i}.png"), dpi=300,
bbox_inches="tight")
        plt.close()

        # 4. Gráfico Evolución de Pérdida, MSE y MAE
        plt.figure(figsize=(10, 12), dpi=300)
        plt.subplot(3, 1, 1)
        plt.plot(history.history["loss"],
label="Entrenamiento",

color=sns.color_palette("colorblind")[0],
linewidth=2.5)
        plt.plot(history.history["val_loss"],
label="Validación",

color=sns.color_palette("colorblind")[1],
linewidth=2.5)
        plt.xlabel("Época", fontsize=12, weight="bold")
        plt.ylabel("Pérdida", fontsize=12,
weight="bold")
        plt.title(f"Evolución de la Pérdida
({loss_function})",
                  fontsize=13, weight="bold", pad=10)
        plt.legend(frameon=True, fontsize=10,
loc="best", edgecolor="black")
        plt.grid(True, linestyle="--", alpha=0.7)
        sns.despine()

        plt.subplot(3, 1, 2)
        plt.plot(history.history["mean_squared_error"],
label="Entrenamiento",

color=sns.color_palette("colorblind")[0],
linewidth=2.5)

plt.plot(history.history["val_mean_squared_error"],
label="Validación",

color=sns.color_palette("colorblind")[1],
linewidth=2.5)
        plt.xlabel("Época", fontsize=12, weight="bold")
        plt.ylabel("MSE", fontsize=12, weight="bold")
        plt.title("Evolución del MSE", fontsize=13,
weight="bold", pad=10)
```

```
        plt.legend(frameon=True, fontsize=10,
loc="best", edgecolor="black")
        plt.grid(True, linestyle="--", alpha=0.7)
        sns.despine()

        plt.subplot(3, 1, 3)
        plt.plot(history.history["mae"],
label="Entrenamiento",

color=sns.color_palette("colorblind")[0],
linewidth=2.5)
        plt.plot(history.history["val_mae"],
label="Validación",

color=sns.color_palette("colorblind")[1],
linewidth=2.5)
        plt.xlabel("Época", fontsize=12, weight="bold")
        plt.ylabel("MAE", fontsize=12, weight="bold")
        plt.title("Evolución del MAE", fontsize=13,
weight="bold", pad=10)
        plt.legend(frameon=True, fontsize=10,
loc="best", edgecolor="black")
        plt.grid(True, linestyle="--", alpha=0.7)
        sns.despine()

        plt.suptitle(f"Iteración {
                        i}: Métricas de Entrenamiento",
fontsize=16, weight="bold", y=1.02)
        plt.tight_layout()
        plt.savefig(os.path.join(OUTPUT_FOLDER,
f"grafico_evolucion_metricas_{
                       i}.png"), dpi=300,
bbox_inches="tight")
        plt.close()

    print(f"Se ha completado la estructura número: {i +
1}")

    # Limpiamos el cache de la memoria
    K.clear_session()
    gc.collect()


# Guardamos el CSV con las metricas segun la hora que
termine para poder identificarlo
hora_actual = datetime.now().strftime(
    "%Y%m%d_%H%M%S")

nombre_archivo = os.path.join(
    OUTPUT_FOLDER,
f"FASE4_SOLUCIONES_{hora_actual}.csv")

df_soluciones.to_csv(nombre_archivo, index=False)
```