Máster en Ciencias Actuariales y Financieras

TRABAJO FINAL DE MÁSTER

Título: Técnicas de Machine Learning y Deep Learning: Una aplicación no vida en el ámbito asegurador.

Autoría: Jhonny Sinisterra Tobar

Tutoría: Salvador Torra Porras

Curso académico: 2021 - 2022

UNIVERSITAT DE BARCELONA
Facultat d'Economia i Empresa

Màster de Ciències Actuarials i Financeres

Facultad de Economía y Empresa

Universidad de Barcelona

Trabajo Final de Máster

Máster en Ciencias Actuariales y Financieras

Machine Learning y Deep Learning: Una aplicación no vida en el ámbito asegurador.

Autoría: Jhonny Sinisterra Tobar

Tutoría: Salvador Torra Porras

El contenido de este documento es de exclusiva responsabilidad del autor, quien declara que no ha incurrido en plagio y que la totalidad de referencias a otros autores se han expresado en el texto.

Agradecimientos

Agradezco a Dios por guiar mi camino y darme fortaleza para llegar a este sueño cumplido.

A mi madre María Delsa Tobar y mi padre, porque son un apoyo incondicional y darme las herramientas para lograr las metas.

A todos mis profesores del Máster por sus enseñanzas y acompañamiento tienen toda mi admiración en especial a Salvador Torra, Manuela Bosch Princep y a Miguel Santolino.

A todos mis compañeros también quiero agradecer por ser mi ayuda y parte de este proceso, Gracias José I. Von Lücken G.

También quiero agradecer a la Universidad del Valle.

A mi hijo Jhoan Esteban Sinisterra Carrillo quiero dedicarle este logro y motivarlo a que descubra su profesión.

A todos mis amigos que me apoyaron en y motivaron.

Machine Learning y Deep Learning: Una aplicación no vida en el ámbito asegurador

Máster en Ciencias Actuariales y Financieras

(2021-2022)

Jhonny Sinisterra Tobar

Tutor: Salvador Torra

Universidad de Barcelona

Resumen

Los modelos GLM cuentan con una tradición, confianza y respaldo por parte de la industria aseguradora en términos de medición de riesgos que ahora pueden complementarse con métodos y algoritmos que provienen de la inteligencia artificial. Los modelos XGBoost pueden ser una alternativa comparable en términos de precisión con los GLM, tienen una fácil implementación y pueden identificarse las variables con mayor importancia e influencia en los resultados. En este trabajo se han utilizados las herramientas y lenguajes de programación más reconocidos (R, Python. Scikit-Learn, TensorFlow y Keras) para estimar modelos GLM y de Inteligencia Artificial para clasificación y regresión tales como Random Forest, XGBoost y Redes Neuronales.

Rede Neuronal, Random, Forest, XGBoost, Seguros, Python

Abstract

GLM models have a tradition, trust, and support from the insurance industry in terms of risk measurement that can now be complemented with methods and algorithms that come from artificial intelligence. The XGBoost models can be a comparable alternative in terms of precision with the GLM, they are easy to implement and the variables with the greatest importance and influence on the results can be identified. In this work, the most recognized tools and programming languages (R, Python, Scikit-Learn, TensorFlow and Keras) have been used to estimate GLM and Artificial Intelligence models for classification and regression such as Random Forest, XGBoost and Neural Networks.

Neural Networks, Random, Forest, XGBoost, Insurance, Python

Índice

1. Introducción 10

2. Metodología 12

2.1. Modelos Lineales Generalizados 12
2.1.1. Articulación del modelo 13
2.1.2. Funciones de enlace 15
2.1.3. Transformación y compensación 16

2.2. Modelos de ensamble: Random Forest 17
2.2.1 Convergencia Random Forest 18
2.2.2. Fuerza y correlación 19
2.2.3 Métodos Random Forest: Regresión y Clasificación. 20

2.3. Red Neuronal Profunda: Tipologías 23
2.3.1 Feed Forward Neural Network 24

3. Aplicación Empírica 25

3.1. Bases de datos 25

3.2. Exploración de los datos 26

3.3. Estructura de las Variables 32

4. Estimación y Resultados 32

4.1 Modelos Lineales Generalizados 32

4.2. Machine Learning: Random Forest y XGBoost 39
4.2.1. Random Forest: Ensayo-error 39
4.2.1.1. Datos Desbalanceados: Penalización 42

4.2.2. Clasificación con Modelo XGBoost 44

4.2.3. Regresión XGBoost 45

4.3 Red Neuronal: Feed Forward Neuronal Network 49

4.3.1. Parámetros: Definición de la arquitectura 49
4.3.1.1. Scikit-Learn 49
4.3.1.2 TensorFlow y Keras 51

4.4. Compilación de Resultados. 54

5. Conclusiones 55

Bibliografía 57

Anexo A 58

Anexo B 61

Código en R y Python 61

Índice de Ilustraciones

Ilustración 1: Embolsado en paralelo. Fuente: Elaboración propia 21
Ilustración 2: Embolsado secuencial, Fuente: Elaboración propia 21
Ilustración 3: Modelo Random Forest de forma esquemática. Fuente: Elaboración propia 22
Ilustración 4: Estructura grafica de Red Neuronal Feed Forward. Fuente: Elaboración propia 24
Ilustración 5: Exposición de la cartera, medida anual. Fuente: Elaboración propia 27
Ilustración 6: Diagrama de cajas de la Exposición. Fuente: Elaboración propia 27
Ilustración 7: Frecuencia de Número de Reclamos Observado en la Cartera. Fuente: Elaboración propia 28
Ilustración 8: Volúmenes de pólizas por área. Fuente: Elaboración propia 28
Ilustración 9: Diagrama de cajas Area vs Exposure. Fuente: Elaboración propia 29
Ilustración 10: Mapa de Francia por regiones. Fuente: Wüthirch &Merz 2022 29
Ilustración 11: Frecuencia de la Potencia del Coche. Fuente: Elaboración propia 30
Ilustración 12: Distribución de edad de los conductores. Fuente: Elaboración propia 30
Ilustración 13: Valores de correlaciones entre las variables. Fuente: Elaboración propia 31
Ilustración 14: Gráficos de Correlaciones. Fuente: Elaboración propia 31
Ilustración 15: Efectos parciales en modelo glm_1; eje (x) categorías de cada variable contra eje (y) ClaimNb. Fuente: Elaboración propia 34
Ilustración 16: Efectos parciales en modelo glm_2; eje (x) categorías de cada variable contra eje (y) ClaimNb. Fuente: Elaboración propia 35
Ilustración 17: Resultado del modelo, frecuencia estimada, cada columna corresponde a valores estimados desde 0 hasta 5. Fuente: Elaboración propia 38
Ilustración 18: Resultado del modelo, frecuencia estimada, cada columna corresponde a valores estimados desde o hasta 5. Fuente: Elaboración propia 39
Ilustración 19:Random Forest ajustado con parámetros por defecto. Fuente: Elaboración propia 41
Ilustración 20:Random Forest, Balanceador por submuestras, arboles. Fuente: Elaboración propia 42
Ilustración 21:Random Forest, balanceado en las observaciones. Fuente: Elaboración propia 43
Ilustración 22: XGB Clasificador, entrenado con variables por defecto. Fuente: Elaboración propia 45
Ilustración 23: XGB Regresión por validación cruzada, arboles de decisión. Fuente: Elaboración propia 47
Ilustración 24: XGB Regresor importancia de los predictores, en datos de muestra. Fuente: Elaboración propia 47
Ilustración 25: XGB Regresión árbol de entrenamiento con todos los datos de la cartera. Fuente: Elaboración propia 48
Ilustración 26: XGB Regresión importancia de los predictores en datos totales. Fuente: Elaboración propia 48
Ilustración 27:'adam' algoritmo de optimización de los pesos. Fuente: Elaboración propia 50

Ilustración 28:Arquitectura de Modelo con funciones de activación Tanh y exponencial. función de perdida Poisson. Fuente: Elaboración propia 51

Ilustración 29: Resultados del entrenamiento Modelo Poisson, activación Tanh y Exponencial. Fuente: Elaboración propia 52

Ilustración 30: Evolución de Aprendizaje, Modelo Tanh, Exponencial. Fuente: Elaboración propia 53

Ilustración 31: XGB Predicciones en conjunto de prueba. Fuente: Elaboración propia 60

Ilustración 32: Conjunto de Prueba. Fuente: Elaboración propia 60

Índice de Tablas

Tabla 1: Funciones canónicas. Fuente: (De Jong & Heller, 2008) 15

Tabla 2: Descripción estadística. Fuente: Elaboración propia 26

Tabla 3: Frecuencia de los siniestros. Fuente: Elaboración propia 28

Tabla 4: Resultados en modelo (glm_1) Utilizando R y el comando glm. Fuente: Elaboración propia 34

Tabla 5: Resultados en modelo (glm_2) Utilizando R y el comando glm. Fuente: Elaboración propia 35

Tabla 6: Resultado Modelo (glm_1) Análisis de Sobredispersión de Poisson. Fuente: Elaboración propia 36

Tabla 7: Resultado modelo (glm_2) Análisis de Sobredispersión de Poisson. Fuente: Elaboración propia 36

Tabla 8: Comparación de resultados; Modelo, Parámetros estimados, Media de la variable ClaimNb estimada en el conjunto de prueba, Deviance del modelo y AIC's. Fuente: Elaboración propia 37

Tabla 9: Prueba de sobre dispersión modelo glm_1. Fuente: Elaboración propia 38

Tabla 10: Prueba de sobre dispersión modelo glm_2. Fuente: Elaboración propia 38

Tabla 11: Modelo Random Forest configuración por defecto. Fuente: Elaboración propia 41

Tabla 12: Random Forest, Valores de entrenamiento. Fuente: Elaboración propia 44

Tabla 13: XGB Resultados. Fuente: Elaboración propia 46

Tabla 14: Resultados MLPRegressor, funciones ReLu y Tanh. Fuente: Elaboración propia 51

Tabla 15: Resultados el modelo con TensorFlow y Keras, Funciones de activación Tanh, ReLU y Exponencial en los datos de prueba. Fuente: Elaboración propia 53

Tabla 16: Resumen de modelos; diferencias entre el valor promedio estimado y promedio de observaciones. Fuente: Elaboración propia 54

Tabla A 1: Modelo glm_1, resultados coeficientes. Fuente: Elaboración propia 58

Tabla A 2: Modelo glm_2, resultados coeficientes. Fuente: Elaboración propia 59

1. Introducción

La práctica Actuarial utiliza la combinación de diferentes herramientas para resolver fenómenos e identificar casi todo tipo de riesgos financieros y actuariales. El abanico de herramientas incluye metodologías que provienen de las matemáticas, estadísticas y ahora de una parte de la inteligencia artificial, es decir, hablamos de Machine Learning y Deep Learning. De la lectura Wüthrich & Merz (2022) surge la inspiración y motivación de este trabajo que busca entender el ajuste que pueden tener las diferentes metodologías utilizadas por la ciencia de datos en la estimación del número de siniestros. Hasta la fecha existe una metodología aplicable a la hora de evaluar fenómenos de este tipo: Los Modelos Lineales Generalizados (GLM, en adelante), los cuales son los predilectos para evaluar este tipo de riesgo por su facilidad de comprensión y cálculo. En la actualidad se puede decir que la mayoría de las empresas y reguladores utilizan y aceptan los modelos GLM por sus propiedades.

Este trabajo utiliza modelos de inteligencia artificial supervisados y no supervisados como una alternativa para analizar fenómenos actuariales. La búsqueda del mejor ajuste para medir el riesgo es una tarea incansable porque puede mejorar o empeorar el negocio asegurador, así una mala gestión de una variable tan importante como el número de siniestros puede llevar al riesgo de insolvencia, entre otros aspectos. Por ese motivo y de manera práctica se quiere comprobar la posible mejora sobre el ajuste de estas metodologías partiendo de los modelos GLM y aplicando en primer lugar, la metodología de ensamble[1] de modelos basados en arboles de decisión y, en segundo lugar, los modelos de redes neuronales artificiales.

La estructura de los datos del número siniestros constituye un reto para los modelos de inteligencia artificial por la estructura tabular de los mismos. El éxito de los modelos de Deep Learning en tareas de clasificación como en el reconocimiento de imágenes, procesamiento de texto o Lenguaje Natural (NLP) y sonido, se debe en gran medida a la estructura homogénea que presentan las imágenes, el sonido, el lenguaje, etc., por ejemplo, para analizar y/o traducir textos de un idioma a otro, existe un grupo de palabras conocido e igual para todas las muestras que proviene del lenguaje mismo, por tanto podemos decir que esta acotado el número de palabras que pueden ser utilizadas

[1] La metodología de ensamble se encuentra descrita en el apartado de metodología.

para la traducción, se expresa la homogeneidad de esta manera. Los datos tabulares tienen una estructura muy heterogénea en comparación con las imágenes o texto (Borisov, et. al. 2021). Todo esto supone un aliciente para comprobar como es el ajuste de las redes neuronales bajo el tipo de datos en la industria aseguradora.

Los algoritmos supervisados de Machine Learning basados en arboles de decision muestran un mejor ajuste a fenomenos de datos tabulares (Borisov, et. al. 2021). En nuestra propuesta se han estimado los modelos de Random Forest y XGBoost opteniendo resultados poco satisfactorios en el primer caso y resultados comparables con los GLM en el segundo caso.

El trabajo está dividido por secciones. Primero se presenta la metodología, donde se expone el proceso metodológico de los tres modelos a implementar. Segundo, se realiza una exploración descriptiva de la información detallando los datos involucrados para el análisis. En tercer lugar, se presentan los resultados de los cálculos y entrenamientos de los modelos bajo diferentes supuestos. Por último, se presenta el apartado de conclusiones donde se agrupan las principales aportaciones.

2. Metodología

Esta sección está basada en las lecturas de los trabajos citados y contiene la extracción de información que se considera necesaria para argumentar y clasificar la tesis que pretende resolverse con la elaboración de este trabajo.

2.1. Modelos Lineales Generalizados

Los Modelos Lineales generalizados (GLM) avanzan hacia las mejoras de las restricciones de sus predecesores los modelos de regresión lineal. Los GLM permiten identificar y cuantificar las relaciones entre la una variable dependiente o de respuesta y las variables independientes o explicativas (De Jong & Heller, 2008). Se presentan dos aspectos clave de los que se diferencia a los GLM de los modelos de regresión ordinarios:

i. La variable de respuesta tiene asociada una distribución de la familia de distribuciones exponencial. De forma que, a diferencia de los modelos de regresión lineales, por ejemplo, de mínimos cuadrados ordinarios, las distribuciones de la variable de respuesta no necesariamente deben ser normal o aproximarse a una distribución normal.

ii. La relación entre la variable respuesta y los predictores no es lineal. Por este motivo es necesario una transformación para la interpretación de los parámetros en términos de cambios marginales.

Una consecuencia de que la variable dependiente tenga asociada una distribución de la familia de las exponenciales es que puede ser, y casi siempre lo es, heterocedástica. Lo que rompe el supuesto de homocedasticidad de la regresión normal, permitiendo que la varianza varie con la media que, además, puede variar con las variables explicativas. De esta manera se refleja la importancia de los GLM en la modelación de los datos del sector asegurador. Las variables contienen información que se relaciona de manera compleja donde los supuestos de la regresión normal con frecuencia no son aplicables. Por ejemplo, el coste de los reclamos, la frecuencia de los reclamos, que es la variable objeto de análisis de este trabajo, son todas resultado de interacciones que no tienen un comportamiento que podamos denominar normal o gaussiano.

Sea una variable de respuesta aleatoria y, definimos el GLM como sigue,

$$f(y) = c(y, \phi) \exp\left\{\frac{y\theta - a(\theta)}{\phi}\right\} \quad (1)$$

$$\eta = g(\mu) = x'\beta \quad (2)$$

Donde $f(y)$ es explícitamente consecuencia de una función de la familia de distribuciones exponencial. La segunda ecuación representa la relación lineal entre la transformación de la media, $g(\mu)$, y las variables independientes o predictores x.

Según la elección de $a(\theta)$[2] se determina la distribución de respuesta.

Llamamos a η_i predictor lineal y la función $g(\mu)$ es también llamada *función de enlace (link function)* y permite obtener los predictores lineales. A la función de enlace se le exige que sea tanto monótona como diferenciable.

En la ecuación (1) se establece que, dado un conjunto de predictores x, la media μ se determina por medio de la función de enlace $g(\mu)$. Una vez obtenida o fijada la media μ, el parámetro canónico θ se determina de $a(\theta) = \mu$. Por último, dado θ, los valores de y se determinan extrayendo de la función de densidad de la exponencial especifica en $a(\theta)$ y el parámetro ϕ es un parámetro de dispersión.

Se asume que las observaciones y son independientes.

2.1.1. Articulación del modelo

Dada una variable dependiente y, definimos el modelo en los siguientes pasos:

- Selección de la distribución de la variable dependiente $f(y)$, lo que implica la elección de $a(\theta)$ en la ecuación (1) la distribución de respuesta depende del objeto de análisis, se elige la que más se adapte a la situación a estudiar, por ejemplo, en el caso estudiado en este trabajo la distribución Poisson para estimar el número de siniestro es ampliamente utilizada, junto con las distribuciones *Binomial negativa* o *Gamma*[3].

[2] Parámetro que está relacionado con la media de forma que $a(\theta) = \mu$.
[3] Véase (Wüthrich & Merz, 2022) y (De Jong & Heller, 2008)

- Selección de una función de enlace $g(\mu)$. Podemos elegir las opciones naturales que acompañan a algunas distribuciones de la familia exponencial las cuales son denominadas funciones de enlace "naturales" o canónicas. Bajo las funciones de enlace canónicas se da la circunstancia de que el parámetro canónico coincide con el predictor lineal de manera que: $\theta(\mu) = \eta$. Es la manera simplificada de elección de la función de enlace, pero, en general, se puede optar por otros enlaces que no sean canónicos como, por ejemplo, enlaces paramétricos (Boj et al., 2020).

- Selección de las variables independientes x de las cuales se modela $g(\mu)$.

- Definimos de la variable dependiente y el conjunto de las observaciones $y_i, \dots, y_n$ dadas las observaciones de las variables independientes $x_1, \dots, x_n$ de x. Suponemos que la muestra es aleatoria y las observaciones son independientes.

- Se procede a la estimación de los parámetros β' por máxima verosimilitud. Estos capturan las sensibilidades entre los predictores x y la variable de respuesta y.

- Dados los valores de los β, generamos las predicciones (o valores ajustados) de y para las diferentes combinaciones de x, luego examinamos las predicciones del modelo comparando con los valores reales, así como otros diagnósticos del modelo. La estimación de las betas también nos permite conocer cuál es el aporte que tienen cada variable explicativa sobre la variable explicada de manera que nos dicen cuáles son importantes para determinar la de μ.

El punto clave para obtener buenos resultados, teniendo en cuenta que no siempre los pasos se dan en el orden descrito, es una exploración previa de los datos para determinar una función de enlace que se ajuste lo mejor posible al fenómeno analizado. En la practica la industria de los seguros cuenta con información que antes de especificar el modelo sirve de orientación para un primer diagnóstico del comportamiento de la variable dependiente y la relación con las variables predictoras o independientes. Es probable que esta exploración inicial nos advierta de usar un modelo determinado y o una distribución en particular. De manera que, es importante conocer los datos como se comportan y de ser necesario realizar transformaciones a las variables o la exclusión de algunas que no tengan sentido para incluirse.

2.1.2. Funciones de enlace

Las funciones de enlace $g(\mu)$ más comunes se presentan en la (*Tabla* 1). En general la forma de las funciones de enlace es del tipo $g(\mu) = \mu^p$, con la diferencia de la función de tipo *Logística* (De Jong & Heller, 2008).

Si $g(\mu) = \theta$ implica que g es el enlace canónico que corresponde $a(\theta)$, en este caso $\theta = x'\beta$. Elegir un enlace canónico g correspondiente a una distribución de la variable de respuesta f se traduce en una simplificación de la estimación, que podía anteriormente ser de utilidad por las limitaciones de cálculo. Actualmente se cuenta con más opciones de computación que facilita la aplicación de diferentes funciones de enlace. Las constantes en θ generalmente se omiten en el enlace canónico. Veamos como un ejemplo, el enlace canónico $1/\mu^2$ esta ligado a la Gaussiana inversa de forma que $\theta = -1/(2\mu^2)$.

Función de enlace	$\boldsymbol{g(\mu)}$	**Función Canónica de enlace**
Identidad	μ	Normal
Logarítmica	$\ln(\mu)$	Poisson
Potencial	μ^p	Gamma $(p = -1)$ Inversa Gaussiana $(p = -2)$
Raíz cuadrada	$\sqrt{\mu}$	
Logística	$\ln\frac{\mu}{1-\mu}$	Binomial

Tabla 1: Funciones canónicas. Fuente: *(De Jong & Heller, 2008)*

A manera de ejemplo, consideremos las reclamaciones a terceros. Al considerar la relación entre el número de accidentes y el número de reclamos a terceros en un área, podríamos encontrar que el número de siniestros, así como la variación de siniestros tienen una correlación positiva con el número de accidentes en el área, es decir

aumentan si este último lo hace. De manera que podemos definir una relación logarítmica entre la media de los reclamos y la media de los accidentes. Lo que puede representarse matemáticamente como:

$$\ln \mu = \beta_0 + \beta_1 \ln z \tag{3}$$

Donde μ es el número medio de los siniestros, y z el número de los accidentes en un área. En la ecuación 3 vemos especificado un enlace logarítmico con la variable explicativa el logaritmo de los accidentes (De Jong & Heller, 2008).

2.1.3. Transformación y compensación

Los recuentos de modelos, como el número de reclamos, que analizaremos aquí, o muertes en un grupo de riesgo, requieren una corrección del número n expuesto al riesgo. Si μ es la media del conteo y, entonces la frecuencia de interés μ/n

$$g(\mu/n) = x'\beta \tag{4}$$

Cuando g es la función logarítmica tenemos entonces

$$\ln(\mu/n) = x'\beta \rightarrow \ln \mu = \ln n + x'\beta \tag{5}$$

La variable n se denomina exposición y la variable $\ln n$ es llamada “compensación”, una compensación es efectivamente una transformación de la variable x en la regresión, con un coeficiente $\beta = 1$. Con la compensación, y tiene un valor esperado directamente proporcional a la exposición.

$$\mu = ne^{x'\beta} \tag{6}$$

La compensación se aplica para acotar, corregir el tamaño de los grupos o diferenciar los periodos de tiempo de las observaciones (De Jong & Heller, 2008).

2.2. Modelos de ensamble: Random Forest

Uno de los algoritmos de aprendizaje automático supervisado utilizados ampliamente en problemas de clasificación o regresión es el Random Forest. El Modelo algorítmico Random Forest (RF, en adelante) utiliza la combinación de árboles de decisión predictivos de manera que cada árbol depende de los valores de un vector aleatorio muestreado de forma independiente y con la misma distribución para todos los árboles del conjunto incluidos (Breiman, 2001). Este algoritmo avanza a los desarrollos que le preceden realizados por el mismo creador Leo Brieman en 1996, donde la propuesta consistía en la selección "embolsado" de árboles de manera aleatoria, sin reemplazo, partiendo de los valores incluidos en el conjunto de entrenamiento. Dicho de otro modo, la técnica Random Forest sobre una característica seleccionada realiza una aleatorización para construir cada árbol que se incluye en el conjunto con la intención de mejorar la diversidad para así intentar reducir en una mayor medida la varianza de las predicciones (Pujol Vila, 2021).

La metodología de árboles de decisión es aplicable para analizar problemas de clasificación y de regresión. Estos trabajan de manera individual y a su vez generaran con cada respuesta más arboles hasta tener grupos homogéneos que son la información en la que se basa la predicción final obtenida como un promedio o una votación por mayoría entre dichos grupos. El proceso de estimación es secuencial, desde el primer al último árbol o nodo se divide la muestra en dos partes (los que, si cumplen la condición determinada y los que no la cumplen), luego son enviadas a los sub-nodos o ramas inferiores donde sigue la reclasificación por condiciones, hasta llegar a una clasificación de las muestras en grupos más pequeños y singulares diferenciados entre sí por los datos que contienen, y en última instancia, por votación o promedio dar una respuesta estimada final. Para evaluar el resultado del ajuste se tienen en cuenta dos factores claves y son: la convergencia, es decir, la cantidad de nodos, y la precisión, es decir, la variación de los resultados que se analiza en términos de fuerza y correlación.

2.2.1 Convergencia Random Forest

La profundidad del conjunto de árboles es un factor determinante en la precisión y así evitar el sobre ajuste del modelo. Veamos entonces como converge la selección del conjunto de clasificadores siguiendo a Brieman 2001.

Dado un conjunto de clasificadores $h_1(x), h_2(x), \dots, h_k(x)$, y definido el conjunto de entrenamiento de forma aleatoria de la distribución del vector aleatorio Y, X. Se define el *margen* de la función:

$$mg(X,Y) = av_k I(h_k(X) = Y) - \max_{j \neq Y} av_k\, I(h_k(X) = j) \tag{7}$$

siendo $I(\cdot)$ el indicador de la función. El *margen* computa la media en la que el número promedio de votos en Y, X para la clase derecha excede el voto promedio para cualquier otra clase. Cuanto mayor sea el margen, mayor confianza en el clasificador. El error de generalización viene dado por

$$PE^* = P_{X,Y}(mg(X,Y) < 0) \tag{8}$$

Donde los índices X, Y indican que la probabilidad esta dentro del espacio de la muestra X, Y.

En el modelo Random Forest se presenta la siguiente igualdad $h_k(X) = h(X, \Theta_k)$. Porque para un gran conjunto de árboles, aplicando la ley fuerte de los grandes números en la secuencia k, con un numero de k predictores el error converge, es decir, incluir un árbol o predictor más no mejora las predicciones, siendo este un resultado particular de la ecuación 9 como se muestra en el teorema 1.2 de convergencia presentado a continuación:

Teorema 1.2. A medida que aumenta el número de árboles, para casi seguramente todas las secuencias $\Theta_1, \dots, \Theta_k$ el PE^*converge a[4],

$$P_{X,Y}(P_\Theta(h(X,\Theta) = Y) - \max_Y P_\Theta\,(h(X,\Theta) = j) < 0) \tag{9}$$

[4] Véase en, (Brieman 2001, pág. 7)

Lo que implica este teorema va mucho más allá de la convergencia, indica una característica de las más importantes que tiene el algoritmo y es que evita el sobre ajuste. Este resultado explica porque la técnica Random Forest no genera "Ovéasefitting" a medida en que crece el conjunto de árboles, agregando predictores, sino que producen un valor limitante del error de generalización[5].

2.2.2. Fuerza y correlación

La información contenida en cada árbol debe no estar correlacionada con la de los demás, así de esta manera los resultados son más consistentes. En el algoritmo RF se puede definir un límite superior para el error de generalización en los dos parámetros, la precisión de los clasificadores individualmente y la dependencia o correlación con los demás. El funcionamiento general del modelo puede entenderse a partir de la interacción entre dichos parámetros. Se definen la fuerza y la correlación en el siguiente teorema.

La función margen para un Random Forest es,

$$mr(X,Y) = P_{\Theta}(h(X,\Theta) = Y) - \max_{j \neq Y} P_{\Theta}\,(h(X,\Theta) = j) \tag{10}$$

Sea la fuerza del conjunto de clasificadores $\{h(X,\Theta)\}$,

$$s = E_{X,Y} mr(X,Y) \tag{11}$$

Suponiendo que $s \geq 0$, la desigualdad de Chebychev es,

$$PE^{*} \leq var(mr)/s^{2} \tag{12}$$

Donde podemos expresar la varianza de forma que,

$$\hat{\jmath}(X,Y) = arg \max_{j \neq Y} P_{\Theta}\,(h(X,\Theta) = j) \tag{13}$$

$$PE^{*} \leq \frac{\bar{\rho}(1-s^{2})}{s^{2}}. \tag{14}$$

[5] EL teorema es demostrado en Brieman 2001 pg., 28.

Luego de algunas operaciones se obtienen los factores involucrados en el error (véase ecuación 14). el error es menor o igual que la proporción entre el valor medio de la correlación[6] $\bar{\rho}$ y el cociente $\frac{(1-s^2)}{s^2}$. Lo que da resultado a la proporción $(c/s^2) = \bar{\rho}/s^2$, esta relación entre la correlación dividida por el cuadrado de la fuerza ayuda a comprender el funcionamiento del RF, mientras más pequeña mejor será.

2.2.3 Métodos Random Forest: Regresión y Clasificación.

La diferencia entre regresión y clasificación en la utilización del algoritmo RF esta básicamente en que, en lugar de etiquetas de clase, los vectores aleatorios que contienen los predictores $h(X, \Theta)$, toman valores numéricos. En general el funcionamiento de los RF para ambas situaciones en la misma, caracterizar por medio del ensamble de modelos una predicción final proveniente de la votación por mayoría o el promedio para clasificación y regresión respectivamente. La combinación de varios modelos se realiza aplicando dos métodos fundamentales *embolsado e impulso*.

El **Bagging** (o embolsado) se puede denominar cómo una técnica de muestreo interna entre las diferentes etapas de entrenamiento de los árboles. Con los datos de entrenamiento de la muestra, crea subconjuntos de datos de entrenamiento en paralelo que contienen algunos datos compartidos, significa que en algunos árboles se repiten datos, luego del entrenamiento el pronóstico final se basa en la votación o promedio de la mayoría, la *Ilustración 1* representa el proceso.

[6] En el teorema 2.3 en Brieman 2001, se incluye el proceso matemático para obtener el valor

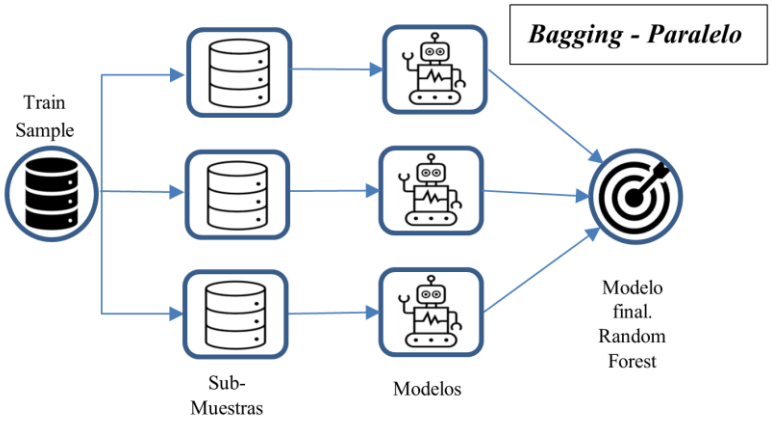


Ilustración 1: Embolsado en paralelo. Fuente: Elaboración propia

El **Boosting** (o Impulso) es la combinación de muestras y resultados de modelos mediante método secuencial. Se estima un primer modelo cuyos resultados se incluyen en los siguientes causando que se modifican los clasificadores de manera que se enfoquen en los casos más difíciles, para así tener una precisión más alta, el modelo resultante es mucho más preciso un ejemplo es el modelo Adaboost o XGBoost.

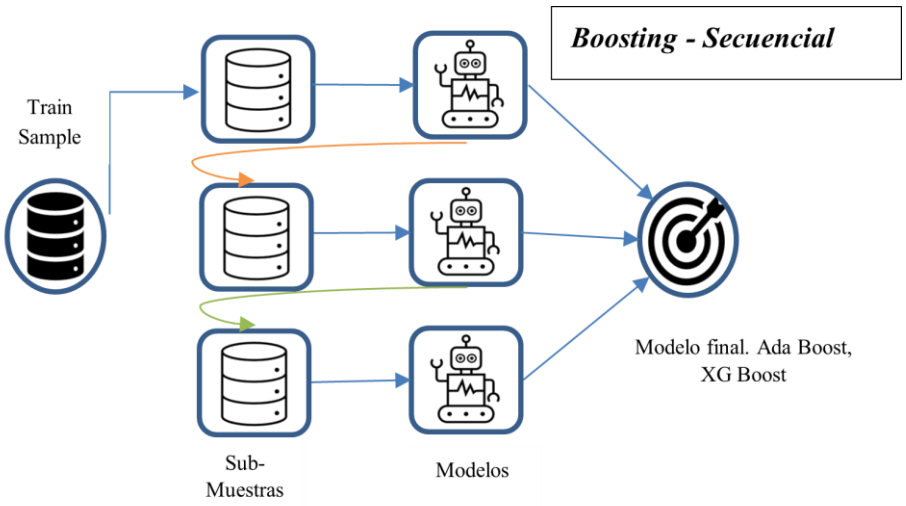


Ilustración 2: Embolsado secuencial, Fuente: Elaboración propia

Las muestras en un RF se generan aleatoriamente y cada modelo se entrena de forma independiente. Como hemos mencionado anteriormente el Embolsado o "Bagging" es el método de muestreo con reemplazo en el que se basa el RF para el entrenamiento

realizando divisiones de los datos de entrada, conocido como *Booststrap Aggregation.* En resumen, son dos fases, la primera Booststrap realiza el muestreo por filas y luego, en aggregation se realiza el paso de cálculo de la predicción que se obtiene de la votación por mayoría o el promedio según sea clasificación o regresión respectivamente.

De manera general podemos graficar la composición de los modelos RF como se observa en la ilustración 3. La parte superior principal son los n árboles que tienen muestras Bootstrap lo suficientemente diferenciadas que alimentan el modelo y que en cada paso hacia abajo se diferencian más las unas de las otras, y sus resultandos en los niveles de clasificación inferiores son nuevas clases. Luego las n clases resultantes se agregan según se el fenómeno analizado, etiquetas o valores, para luego crear una clase final basada en todo el proceso.

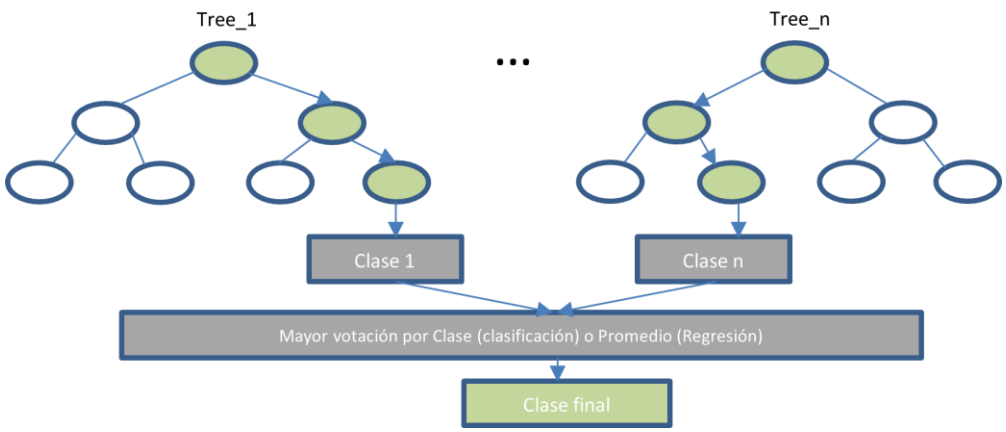


Ilustración 3: Modelo Random Forest de forma esquemática. Fuente: Elaboración propia

2.3. Red Neuronal Profunda: Tipologías.

Las redes neuronales tienen como inspiración la naturaleza humana, en concreto el cerebro. Así como otras muchas invenciones sugeridas de la observación del mundo animal, (como por ejemplo la aviación), del análisis de la arquitectura del cerebro y su funcionamiento se crea lo que algunos denominan redes neuronales artificiales, aunque en realidad, de la misma forma que los aviones a los pájaros, en su funcionamiento las neuronas y las redes neuronales artificiales se diferencian en su composición (Géron, 2019). Existen tres tipologías comúnmente utilizadas de redes neuronales y estas se diferencian según la manera en la que se conectan los perceptores o neuronas. Encontramos los nombres de Feed Forward Neural Network (FFNN), Recurrent Neural Network (RNN) y Convolutional Neural Network (CNN). Cada una tiene un factor diferencial según el fenómeno por analizar, de modo que según sea el problema, por ejemplo, el procesamiento de imagen, audio o texto se aplicara un tipo de red u otra. Así, para análisis de imagen es conveniente trabajar con CNN y para procesamiento de texto RNN (Borisov, et. al. 2022).

Feed Forward Neuronal Network (FFNN) en algunos textos se denomina Deep Feed forward Network es de las primeras formas de redes neuronales y de los modelos por excelencia (Goodfellow et. al, 2016). Se caracteriza porque sus neuronas se conectan de manera unidireccional iniciando desde las entradas hasta las salidas. En cambio, las redes CNN por su arquitectura, se especializan en imágenes, debido a la capacidad de extracción de características por medio múltiples filtros, es capaz de identificar patrones, estas primero se centran en pequeñas porciones de la imagen realizando un mapeo para la extracción de información relevante y luego repite el proceso comparando las características halladas con los datos de muestra. Finalmente, y con relación a los modelos RNN es una extensión de las FFNN que es una red neuronal especializada en el procesamiento de valores $x^1 \dots x^t$ donde el superíndice indica que se indexa en el tiempo, por esta característica se puede aplicar a modelar series de tiempo, pero es más comúnmente usada en el procesamiento del lenguaje natural, (Natural Language Process, NLP).

2.3.1 Feed Forward Neural Network

El modelo FFNN es el más utilizado dentro de las redes neuronales (Vitrià, 2022). Como hemos visto cada modelo nace o se aplica para analizar un fenómeno particular. En nuestro análisis utilizamos el FFNN por el tipo de información con la que se cuenta, la base de datos de pólizas de seguros de auto. Los FFNN están organizados en capas, de entrada, oculta y de salida. Las primeras son las capas de entrada contiene la información inicial que corresponde a las características, variables explicativas $x_1, \dots x_n$. En la capa interna o capa oculta, se presentan las interacciones que por medio de funciones de activación θ transforman la información estableciendo valores, denominados pesos, para caracterizar cada influencia de las variables de entrada en la variable de salida, en este nivel la información que recibe cada neurona es una copia de información procesada por todas las neuronas anteriores de manera unidireccional hasta generar el resultado final y en la capa de salida.

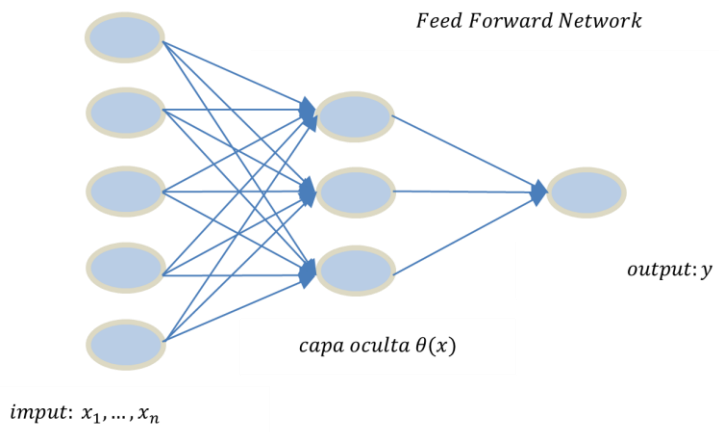


Ilustración 4: Estructura grafica de Red Neuronal Feed Forward. Fuente: Elaboración propia

Matemáticamente lo podemos escribir como,

$$y = f(x; \theta) \tag{1}$$

El objetivo es aproximar una función f^* dados unos valores de entrada x tal que se cumpla 1. Donde el valor de θ es el resultado de la mejor función de aproximación. La selección de la función de aproximación, también conocida como función de activación es clave, existiendo una familia de funciones en las que se incluyen las funciones *Sigmoide, tanh, ReLU.* Cada una tiene una característica que la hace diferente a las

demás y debemos fijarnos en las limitaciones que presentan cada una. La función *Sigmoide* tiene dos problemas importantes, el primero consiste en que las neuronas saturadas limitan el gradiente, es decir, el algoritmo que permite recorrer los puntos de datos evaluando los pesos hasta encontrar la combinación de pesos que genera el menor error posible, por tanto, es un factor clave para converger. Y el segundo, los resultados sigmoides no están centrados en cero. *tanh*, aunque está centrada en cero, tiene el mismo problema de neuronas saturadas. Finalmente, *ReLU* no es saturada, computacionalmente es eficiente y tiene una rápida convergencia, pero para los valores $x < 0$ la función no tiene sentido (Vitrià, 2022).

3. Aplicación Empírica

3.1. Bases de datos

La información que se utiliza en la aplicación práctica pertenece a una cartera de responsabilidad civil a terceros de automóviles francesa. Los datos que utilizamos en nuestra aplicación práctica son extraídos de la base de datos CASdatasetd[7], creada por Christophe Dutang. Es de acceso libre y se puede descargar desde el programa R. La exploración de los datos, procesamiento y limpieza se realiza siguiendo a Wüthirch y Merz (2022). El conjunto de datos este compuesto por FreMTPL2freq y FreMTPL2sev; el primero contienen la frecuencia de los siniestros y el segundo la gravedad de las reclamaciones de las pólizas. Se crea un conjunto de datos con las dos bases de datos que contiene un total de 678,007 datos de las pólizas de seguro de los cuales 26,383 corresponden a las reclamaciones, reporte de siniestros.

Las variables incluidas en la base de datos construida son:

1. **IDpol**: Número de la póliza (identificador único)
2. **Exposure**: Exposición total en unidades de año (años en riesgo, (0, 1])
3. **VehPower**: Potencia del coche (variable continua)
4. **VehAge**: Modelo del coche en años desde la matricula inicial

[7] Véase dirección web del repositorio de los datos: http://cas.uqam.ca/

5. **DrivAge**: Edad del (conductor usual) en años
6. **BonusMalus**: Nivel de BonusMalus entre 50 y 230 (con nivel de entrada 100%)
7. **VehBrand**: Marca del coche (variable categórica se incluyen 11 marcas)
8. **VehGas**: Combustión del coche, variable binaria (Diesel o combustible regular)
9. **Area**: Código de área (variable categórica, ordinal con 6 niveles)
10. **Density**: Densidad de la población por kilómetro cuadrado en el lugar de residencia del conductor
11. **Region**: Región de Francia (antes de 2016, categórica)
12. **ClaimNb**: Pólizas con reclamo por siniestro

3.2. Exploración de los datos

La tabla 2 contiene los indicadores estadísticos básicos, tamaño de la muestra, media, desviación estándar, mínimo, los cuartiles 25%, 50% y 75%, y el valor máximo de cada variable.

	Exposure	VehPower	VehAge	DrivAge	BonusMalus	Density
total	678007	678007	678007	678007	678007	678007
media	0.528547	6.454653	7.044218	45.49906	59.76159	1792.431
std	0.364081	2.050902	5.666235	14.13749	15.6367	3958.663
mínimo	0.002732	4	0	18	50	1
25%	0.18	5	2	34	50	92
50%	0.49	6	6	44	50	393
75%	0.99	7	11	55	64	1658
máximo	1	15	100	100	230	27000

Tabla 2: Descripción estadística. Fuente: Elaboración propia

Iniciamos el análisis con la variable **Exposición** (Exposure) que mide la duración del seguro en unidades anuales. Se grafica la distribución de frecuencia según el expuesto de las pólizas en (*Ilustración 5)*. En el diagrama de cajas (*Ilustración 6*) y se valida la información estadística de la tabla 2 de manera gráfica, la duración media es de un poco más de medio año, el mínimo periodo es de 0.002732 correspondiente a un día de

cobertura y el máximo es 1, un año. Aproximadamente 1/4 de la cartera tiene la duración completa de un año, y se observa que con la cantidad de pólizas con exposición igual a un año que superan los 175,000.

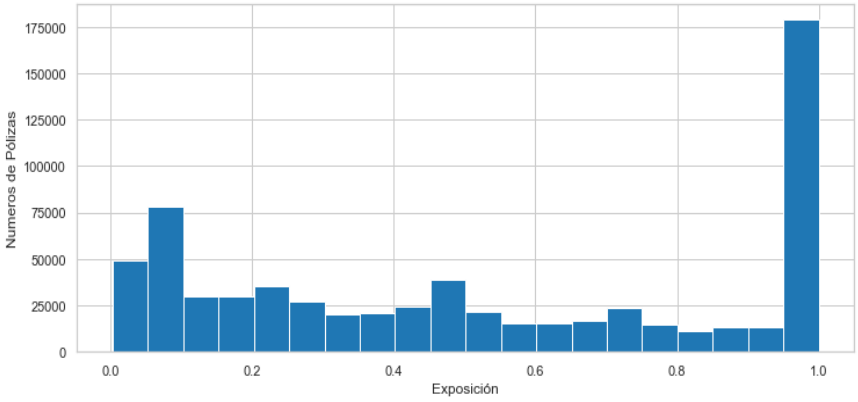


Ilustración 5: Exposición de la cartera, medida anual. Fuente: Elaboración propia

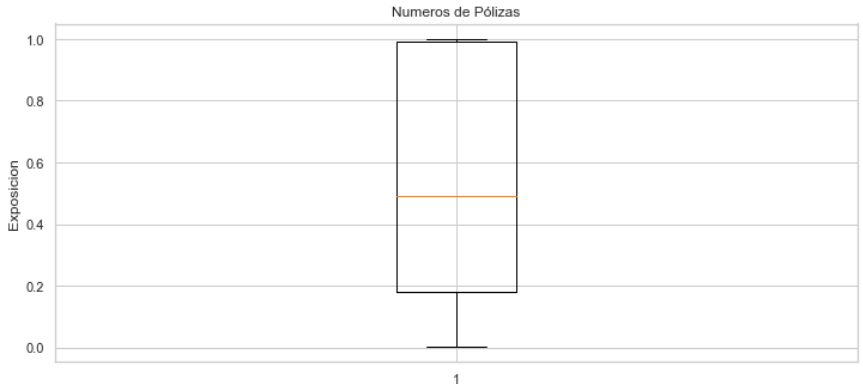


Ilustración 6: Diagrama de cajas de la Exposición. Fuente: Elaboración propia

La variable **número de reclamos** (ClaimNb), presentada en (*Ilustración 7)*, muestra que la mayoría de las pólizas no presenta reclamaciones. Se encuentra que el porcentaje de pólizas que declaran los siniestros entre el total de pólizas es el 3.89%. La tabla tres muestra la frecuencia de los siniestros. Las pólizas que presentaron reclamaciones son 24,938, de las cuales 23,571 reclamaron una vez, 1,298 pólizas reclamaron 2 veces, 62 pólizas tuvieron 3 siniestros, 5 pólizas tuvieron 4 y por último solo 2 pólizas reclamaron hasta 5 veces.

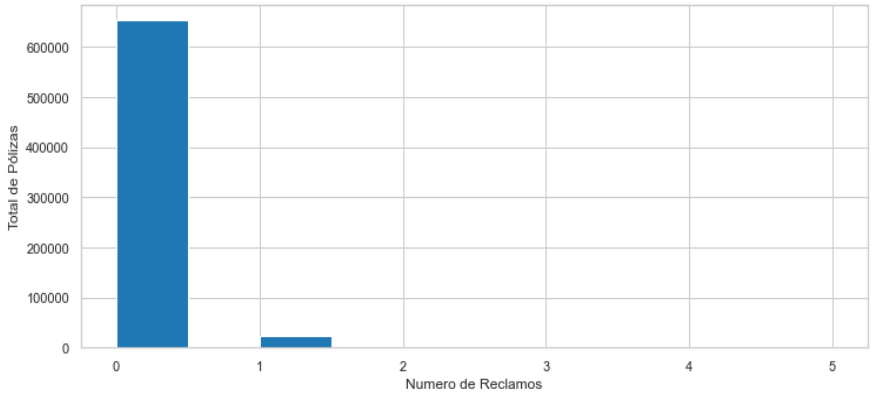


Ilustración 7: Frecuencia de Número de Reclamos Observado en la Cartera. Fuente: Elaboración propia

Frecuencia de siniestros	**0**	**1**	**2**	**3**	**4**	**5**
Número de Siniestros	653069	23571	1298	62	5	2

Tabla 3: Frecuencia de los siniestros. Fuente: Elaboración propia

La variable **Area** se ha agrupado en 6 niveles para mayor facilidad. La (*Ilustración 8*) muestra cual es la acumulación de las pólizas según la agrupación de áreas y podemos véase que existe un alto volumen en la región denominada C y D acumulando más de la mitad de las pólizas y la región con menor cantidad es el área F. El mapa (*Véase Ilustración 10)* muestra la clasificación de las 22 zonas. Por último, tenemos la (*Ilustración 9)* diagrama de cajas por cada zona y su Exposición. En las áreas F y la E son las de menor valor medio de expuesto, la de mayor exposición es la región A.

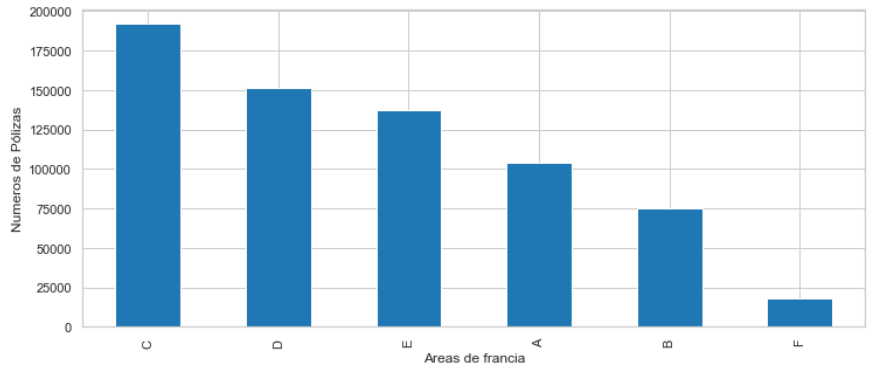


Ilustración 8: Volúmenes de pólizas por área. Fuente: Elaboración propia

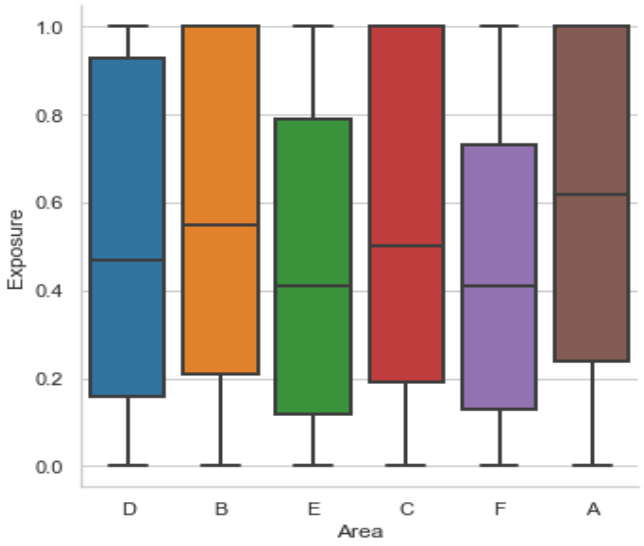


Ilustración 9: Diagrama de cajas Area vs Exposure. Fuente: Elaboración propia

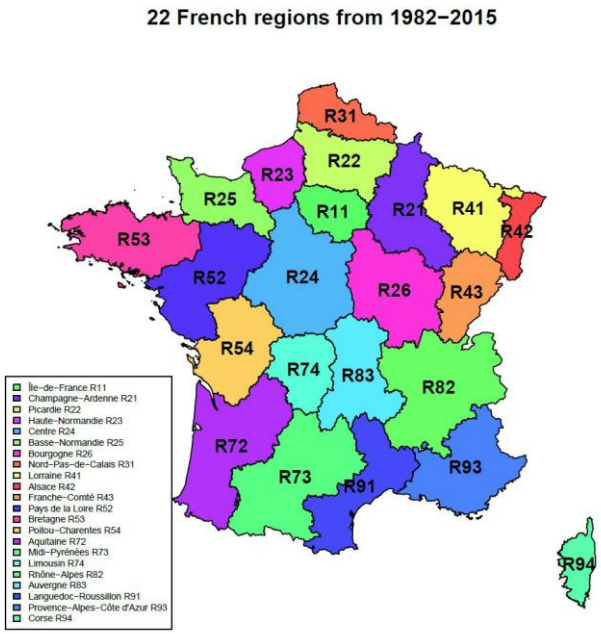


Ilustración 10: Mapa de Francia por regiones. Fuente: Wüthirch &Merz 2022

Potencia del coche en (*Ilustración 11)* muestra que, teniendo en cuenta que a mayor valor del eje X mayor potencia, la mayoría de los coches tienen una potencia media baja.

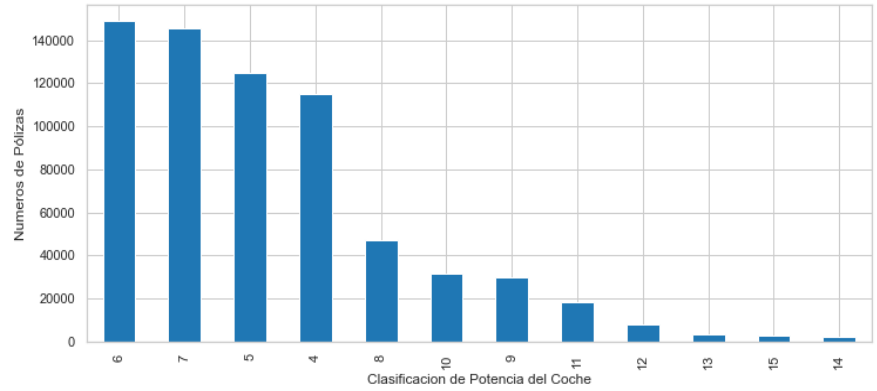


Ilustración 11: Frecuencia de la Potencia del Coche. Fuente: Elaboración propia

La distribución por edades en la cartera la podemos ver en *Ilustración 12.* Alrededor de los 40 años tenemos la mayor acumulación de pólizas y las personas mayores de 80 son una cantidad mínima.

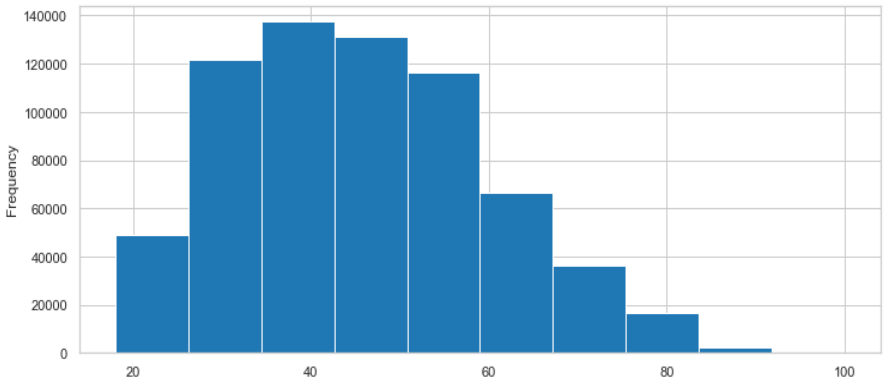


Ilustración 12: Distribución de edad de los conductores. Fuente: Elaboración propia

El grafico de correlaciones (véase *Ilustración 13)* no evidencia que las correlaciones son relevantes excepto el caso de las variables edad del conductor (DrivAge) y BonusMalus, en color rojo intenso, que presentan una correlación de 48% positiva entre ellas.

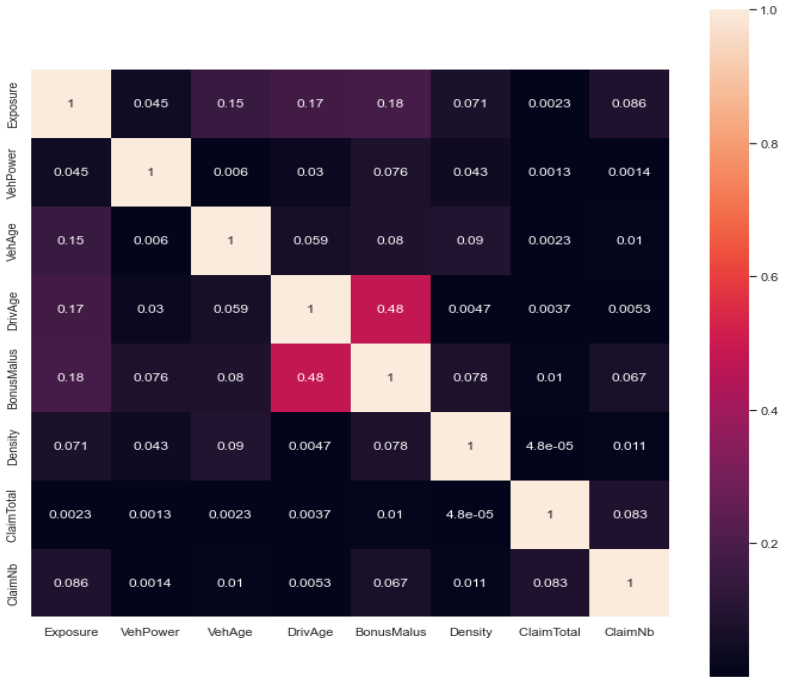


Ilustración 13: Valores de correlaciones entre las variables. Fuente: Elaboración propia

No se observa una relacion de tipo lineal entre ninguna de las varibales . La *Ilustracion 14* muestra de manera grafica como son las correlaciones entre las variables. En la Diagonal se encuentran la frecuencia de cada una y por fuera de la diagoal los graficos exbosan la correlacion entre las variables numericas que se presuponen con mayor peso.

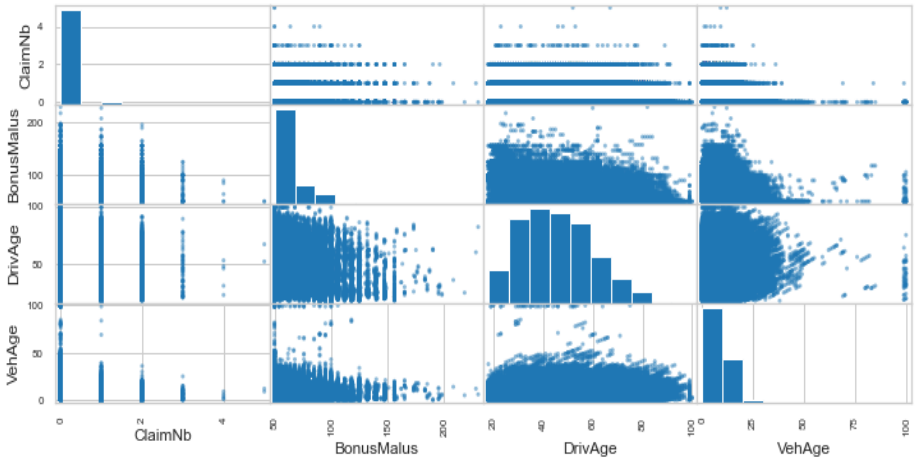


Ilustración 14: Gráficos de Correlaciones. Fuente: Elaboración propia

3.3. Estructura de las Variables

Para mejorar el rendimiento de los algoritmos se realizan las siguientes transformaciones a los datos:

La variable Densidad de la población (log (Density)) de utiliza en logaritmos, la potencia del coche (VehPower) se agrupa en las categorías que incluyen la clasificación de potencia de entre 6 hasta 9, incluyendo esta última categoría a las que están por encima hasta la categoría 15 para tener un total de 6 categorías. Area quedara como categórica de 21 niveles. La edad del vehículo (VehAge) solo tenga tres categorías que agrupan a los coches de entre 0-5, 6-13 y más de trece años. La edad de los conductores (DrivAge) será una variable categórica de 7 agrupando a las edades de 18-20, 21-25, 26-30, 31-40, 41-50, 51-70 y 70+. Y por último de limita la variable Bonusmalus hasta el nivel de 150. Los datos son divididos en particiones de entrenamiento(train) y prueba (test) con una fracción del 80% a 20%. Esta división se hace procurando que la distribución de los datos originales, en cuanto a frecuencia, se mantenga en los dos conjuntos de datos.

4. Estimación y Resultados

4.1 Modelos Lineales Generalizados

Se han estimado solo dos modelos porque más allá de realizar una aplicación profunda de la metodología GLM, que ya es ampliamente utilizada[8] y conocida para resolver este tipo de fenómenos, lo que pretende este trabajo es mostrar a grandes rasgos su utilidad, interpretación y características del modelo en general para luego poder comparar con las otras técnicas de Machine Learning y Deep Learning alternativas que se proponen en este trabajo.

La función Poisson nos permite modelar conteos y/o tasas de ocurrencia de una variable dependiente en función de unos predictores. Los modelos estimados tienen como variable dependiente el número de siniestros (ClaimNb) regresada por las independientes, VehPower, VehAge, DrivAge, Bonusmalus, VehBrand, VehGas,

[8] Si el lector desea profundizar remitirse a (Wüthrich & Merz, 2022, págs. 97-149).

Density, Region, Area. En ambos modelos el conjunto de datos dividido en *train* y *test* es el mismo, la función de distribución aplicada es Poisson con una función enlace logarítmica, como se muestra en la *Tabla 1* de la sección de Metodología. Lo que diferencia estos modelos es la aplicación del parámetro *offset = log (Exposure)*, que implica que estamos agregando un componente conocido a priori que captura el comportamiento en un espacio temporal. En síntesis, lo que se está modelando entonces es, $\log\big(E(ClaimNb)\big) = \beta X$ y cuando aplicamos el offset[9], $\log\left(\frac{E(ClaimNb)}{Exposure}\right) = \beta X$, siendo β los coeficientes estimados y X los predictores o regresores, el primero devuelve la media de siniestros y el segunda la tasa media de siniestros en un año.

Se muestran en las tablas de resultados, (véase en la Tabla 4 y 5), la fórmula de cálculo, la desviación de los residuales del modelo y las primeras 15 variables[10] y los valores de sus parámetros. En el modelo glm_1 (véase *Tabla 4*) son las primeras 15 variables significativas estadísticamente a diferentes niveles de significancia entre el 0.99 y el 0.90. El parámetro de intercepto toma el valor de $\exp\big(b_{0;1}\big) = 0.008159$, lo que implica que la media de los siniestros tiene un valor inicial igual al intercepto. Los predictores que aparecen en este resumen son la potencia del coche (VehPower) siendo la categoría 9, que incluye las clasificaciones de potencia de 9 y superiores, la de mayor aportación al número de siniestros, puede verse en *Ilustración 15* de efectos parciales. Los conductores (incluidos en DrivAge) con las edades de 70+ son el grupo de edad que más aportan a la media. La variable edad del vehículo (VehAge) en la categoría de 13 años de antigüedad tiene una menor aportación a la media de los siniestros. Y por último (Bonusmalus) presenta una aportación al valor de la media lineal creciente.

[9] Desplazamiento de la variable, para calcular la tasa de frecuencia.
[10] El resumen completo se encuentra en el Anexo A, glm_1, glm_2.

```
print(summary(glm_1))

Call:
glm(formula = ClaimNb ~ VehPowerGLM + VehAgeG + DrivAgeG + BonusMalusG +
    VehBrand + VehGas + DensityG + Region + Area, family = poisson,
    data = learn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1505  -0.2990  -0.2618  -0.2232   6.4440

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -4.8087429  0.1292022 -37.219  < 2e-16 ***
VehPowerGLM5             0.0707691  0.0230097   3.076  0.00210 **
VehPowerGLM6             0.1010213  0.0225358   4.483 7.37e-06 ***
VehPowerGLM7             0.0696916  0.0223497   3.118  0.00182 **
VehPowerGLM8             0.0768758  0.0317199   2.424  0.01537 *
VehPowerGLM9             0.1895051  0.0250311   7.571 3.71e-14 ***
VehAgeG6-12              0.0660175  0.0155247   4.252 2.11e-05 ***
VehAgeG13+              -0.2396979  0.0201663 -11.886  < 2e-16 ***
DrivAgeG21-25           -0.3418556  0.0492562  -6.940 3.91e-12 ***
DrivAgeG26-30           -0.4654403  0.0482580  -9.645  < 2e-16 ***
DrivAgeG31-40           -0.2219804  0.0464348  -4.780 1.75e-06 ***
DrivAgeG41-50            0.1226886  0.0471129   2.604  0.00921 **
DrivAgeG51-70            0.1231598  0.0472821   2.605  0.00919 **
DrivAgeG71+              0.2599672  0.0528776   4.916 8.82e-07 ***
BonusMalusG              0.0222057  0.0004026  55.160  < 2e-16 ***
```

Tabla 4: Resultados en modelo (glm_1) Utilizando R y el comando glm. Fuente: Elaboración propia

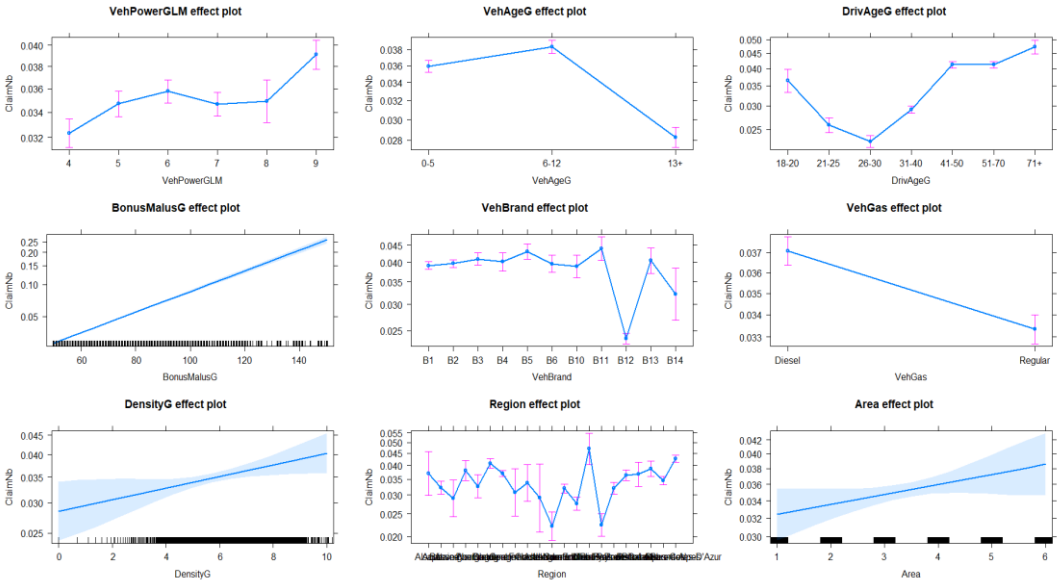


Ilustración 15: Efectos parciales en modelo glm_1; eje (x) categorías de cada variable contra eje (y) ClaimNb. Fuente: Elaboración propia

En el modelo glm_2 (véase *Tabla 5*) una de las primeras 15 variables explicativas no tiene significancia estadística, las 14 restantes en este resumen son estadísticamente significativas entre niveles del 0.99 y 0.90. las variables que aparecen en las primeras líneas son la potencia del coche (VehPower), edad del conductor (DrivAge), antigüedad del coche (VehAge) y Bonusmalus. Los efectos los seguimos en la *Ilustración 16*. La influencia de la variable de potencia del coche en el nivel 9 es la mayor dentro de las demás, bajo este enfoque las edades de 71+ tienen un efecto menor en comparación con el primer modelo, y para ver que más diferencias tenemos a nivel de modelo seguimos

nuestro análisis con algunos de los criterios que podemos analizar para entender cuál modelo ajusta mejor.

```
summary(glm_2)

Call:
glm(formula = ClaimNb ~ VehPowerGLM + VehAgeG + DrivAgeG + BonusMalusG +
    VehBrand + VehGas + DensityG + Region + Area, family = poisson,
    data = learn, offset = log(Exposure))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.4991  -0.3262  -0.2456  -0.1383   7.7830

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)         -4.4026729  0.1290559 -34.114  < 2e-16 ***
VehPowerGLM5         0.0539821  0.0230161   2.345 0.019006 *
VehPowerGLM6         0.0917432  0.0225288   4.072 4.66e-05 ***
VehPowerGLM7         0.0833632  0.0223660   3.727 0.000194 ***
VehPowerGLM8         0.1118720  0.0317714   3.521 0.000430 ***
VehPowerGLM9         0.2396135  0.0251195   9.539  < 2e-16 ***
VehAgeG6-12         -0.0153484  0.0154186  -0.995 0.319519
VehAgeG13+          -0.2766311  0.0200255 -13.814  < 2e-16 ***
DrivAgeG21-25       -0.4651509  0.0492650  -9.442  < 2e-16 ***
DrivAgeG26-30       -0.5829329  0.0482728 -12.076  < 2e-16 ***
DrivAgeG31-40       -0.3914048  0.0464294  -8.430  < 2e-16 ***
DrivAgeG41-50       -0.1069522  0.0470634  -2.273 0.023055 *
DrivAgeG51-70       -0.1829268  0.0471848  -3.877 0.000106 ***
DrivAgeG71+         -0.1804836  0.0527196  -3.423 0.000618 ***
BonusMalusG          0.0272392  0.0003862  70.535  < 2e-16 ***
```

Tabla 5: Resultados en modelo (glm_2) Utilizando R y el comando glm. Fuente: Elaboración propia

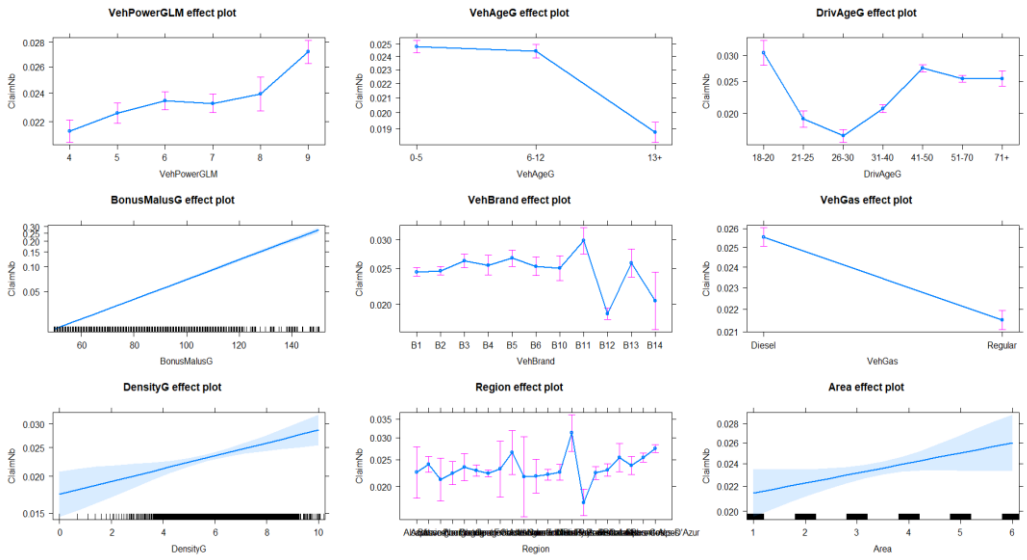


Ilustración 16: Efectos parciales en modelo glm_2; eje (x) categorías de cada variable contra eje (y) ClaimNb. Fuente: Elaboración propia

El modelo glm_2 comparativamente desde un análisis de desviaciones tiene mejores resultados. Cuando analizamos la tabla ANOVA y vemos los efectos de las variables y la varianza nula, encontramos que tiene menos varianza en el conjunto de las variables explicativas el modelo 2 si lo comparamos con el modelo 1, (véase *Tabla 6 y 7*).

```
anova(glm_1)
Analysis of Deviance Table

Model: poisson, link: log

Response: ClaimNb

Terms added sequentially (first to last)

             Df Deviance Resid. Df Resid. Dev
NULL                        610205     158221
VehPowerGLM   5   131.50    610200     158090
VehAgeG       2   436.16    610198     157654
DrivAgeG      6   586.26    610192     157067
BonusMalusG   1  2630.32    610191     154437
VehBrand     10   939.68    610181     153497
VehGas        1    29.95    610180     153467
DensityG      1   229.75    610179     153238
Region       20   419.77    610159     152818
Area          1     3.01    610158     152815
```

Tabla 6: Resultado Modelo (glm_1) Análisis Anova, desviaciones Poisson. Fuente: Elaboración propia

```
> anova(glm_2)
Analysis of Deviance Table

Model: poisson, link: log

Response: ClaimNb

Terms added sequentially (first to last)

             Df Deviance Resid. Df Resid. Dev
NULL                        610205     154252
VehPowerGLM   5     85.5    610200     154167
VehAgeG       2    197.3    610198     153969
DrivAgeG      6   1183.2    610192     152786
BonusMalusG   1   4321.4    610191     148465
VehBrand     10    260.9    610181     148204
VehGas        1     83.2    610180     148121
DensityG      1    539.2    610179     147581
Region       20    182.7    610159     147399
Area          1      3.7    610158     147395
```

Tabla 7: Resultado modelo (glm_2) Análisis Anova, desviaciones Poisson. Fuente: Elaboración propia

La *Tabla 8* muestra la información de resultados de los modelos y los compara. Teniendo en cuenta que la media de la variable ClaimNb en el conjunto total de datos de prueba es 0.03881, se utiliza como valor objetivo. Se encuentra que en los datos de prueba el primer modelo tiene un valor estimado de la media de siniestros mayor al observado, es el segundo modelo el que más se acerca al valor objetivo, el valor que explica la variabilidad del modelo es más alto en el modelo glm_2 lo que implica que con el mismo conjunto de predictores recoge mejor los efectos. Por último, el valor del criterio de información de Akaike (AIC) es menor en el modelo glm_2, sumando estos criterios nos podría llevar a elegir el modelo glm_2 sobre glm_1, pero antes es necesario probar, además de quitar las variables no significativas, que se cumpla el único supuesto que se plante al trabajar con una distribución Poisson y es la sobredispersión.

Modelo	Parámetros Estimados	Media estimada $\mu_{test} = 0.03881$	Deviance^2	AIC
glm_1	49	0.03901722	3.416991	198688.8
glm_2	49	0.03896313	4.445447	193268.9

Tabla 8: Comparación de resultados; Modelo, Parámetros estimados, Media de la variable ClaimNb estimada en el conjunto de prueba, Deviance del modelo y AIC's. Fuente: Elaboración propia

La prueba de diagnóstico más importante es la prueba de sobredispersión. Si nuestro modelo no pasa la prueba significa que al asumir que la distribución del número de siniestro se distribuye según una distribución de Poisson cometemos un error pues la característica principal de una función de Poisson es que $E(Y) = \mu = \lambda = \sigma^2$, la media y la varianza son iguales. El cociente debe ser $\left(\frac{Deviance}{G.L.residuales}\right) \approx 1$, si es mayor a uno no podemos asumir distribución Poisson y se debe ajustar un modelo diferente, si existe sobre-dispersión, la varianza mayor que λ la alternativa es estimar con una distribución *Binomial Negativa* o corregir utilizando *Poisson inflado de ceros.*

La función parametrizada de Comway-Maxwell-Poisson puede ayudar ajustando la varianza. Los efectos aleatorios también ayudan a controlar la varianza. Calculamos entonces la prueba de dispersión utilizando la librería de *R* llamada AER, donde la hipótesis nula es equi-dispersión frente a la alternativa de sobre-dispersión, y los resultados indican que hay evidencia de sobre dispersión al ser el parámetro en ambas pruebas cercanos a uno, con un p-value<0.05 rechazamos hipótesis nula, (véase *tabla 9 y 10*).

```
> dispersiontest(glm_1)

        Overdispersion test

data:  glm_1
z = 18.81, p-value < 2.2e-16
alternative hypothesis: true dispersion is greater than 1
sample estimates:
dispersion
   1.07619
```

Tabla 9: Prueba de sobre dispersión modelo glm_1. Fuente: Elaboración propia

```
> dispersiontest(glm_2)

        Overdispersion test

data:  glm_2
z = 6.9951, p-value = 1.326e-12
alternative hypothesis: true dispersion is greater than 1
sample estimates:
dispersion
  1.107777
```

Tabla 10: Prueba de sobre dispersión modelo glm_2. Fuente: Elaboración propia

En esta sección hemos analizado de manera general los principales aspectos que suponen la modelación del conteo y tasa de siniestros de la cartera por GLM de Poisson. El resultado grafico se las frecuencias modeladas según los resultados del modelo están en las figuras *(Ilustración 17 y 18)*. Existen como se ha mencionado maneras alternativas para mejorar el modelo desde la metodología de los modelos lineales generalizados que no abordamos en este trabajo que el lector podrá seguir dirigiéndose a la bibliografía. Hay que destacar que las variables no se le aplicaron ninguna transformación que pudiesen mejorar los resultados, que en caso de sobredispersión una alternativa es modelar un GLM con distribución Binomial Negativa, Poisson inflado de ceros y demás mejoras que se pueden aplicar. En los siguientes apartados se especifican los modelos alternativos a los GLM que son el objetivo principal de este trabajo.

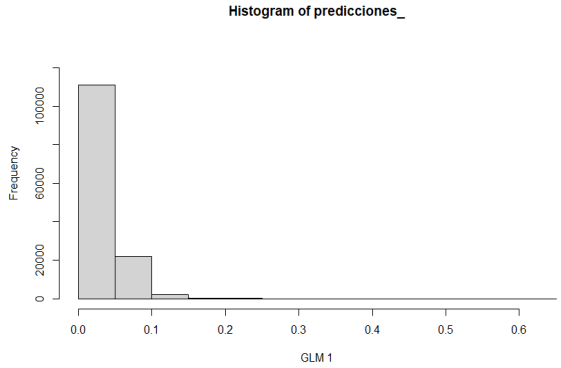


Ilustración 17: Resultado del modelo, frecuencia estimada, cada columna corresponde a valores estimados desde 0 hasta 5. Fuente: Elaboración propia

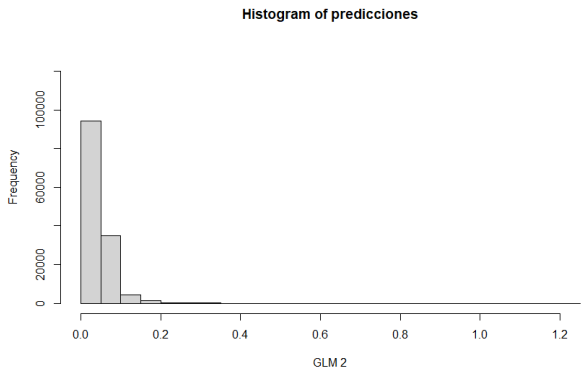


Ilustración 18: Resultado del modelo, frecuencia estimada, cada columna corresponde a valores estimados desde o hasta 5. Fuente: Elaboración propia

4.2. Machine Learning: Random Forest y XGBoost

4.2.1. Random Forest: Ensayo-error

La programación y entrenamiento de los datos se realiza en Python[11] como en la mayoría de la literatura presentada en este trabajo y por decisión del autor.

Para ejecutar los algoritmos es necesario realizar modificaciones a las variables de entrada para su correcta computación. Utilizando el método OneHotEncoder[12] y StandardScaler se han convertido las variables a categóricas y normalizado respectivamente. El método OneHotEncoder crea por cada categoría una columna con unos cuando tenga presencia la categoría dada y cero en los demás. Por ejemplo, para la variable tipo de combustible (VehGas) creara dos columnas, "Diesel" y "Regular", y asignara 1 en la primera columna y cero en la segunda para las pólizas con en uso de Diesel, y viceversa. El método StandardScaler toma las variables numéricas y las ajusta bajo una distribución normal estándar. El resultado de estas trasformaciones, incluyendo a la densidad de la población (Density) y Bonusmalus con el método StandardScaler y las demás variables como categóricas con el OneHotEncoder, 57

[11] Puede descargarse de la página oficial en: https://www.python.org/downloads/, es recomendable utilizar el entorno de gestor de paquetes *anaconda* disponible en: https://www.anaconda.com/products/distribution, debido a que ya vienen precargado la mayoría de las librerías necesarias para la ejecución de los notebooks y porque es una comunidad de respaldo.
[12] Todos los métodos de preprocesamiento de datos necesarios en los algoritmos de Machine Learning se encuentran en la librería de Scikit-Learn en la web: https://scikit-learn.org/stable/modules/preprocessing.html

variables totales. Luego de aplicar estas transformaciones pasamos a estimar los modelos.

Se han estimado varios modelos de manera exploratoria para buscar el de mejor ajuste. Se presentan los principales resultados partiendo del primer modelo siguiendo la configuración por defecto del algoritmo en la librería *sklearn*[13]. Iniciamos nuestro camino de ensayo error, 'éxito', encontrando que el primer modelo tiene un acierto del 95.166% entre el total de observaciones. El panorama cuando miramos las métricas de precisión por clase nos alerta, dado que el modelo a pesar de tener un valor de precisión alto, lo está haciendo bien solo en una de las clases, las demás clases no están siendo debidamente clasificadas y se puede véase en la matriz de confusión (véase *Ilustración 19*) del modelo que contiene todas predicciones como porcentaje del total. Vemos que no sen ha clasificado ningún perfil de las pólizas con número de siniestros mayores o iguales a dos y tan solo el 0.00077 se clasificaron correctamente dentro de las que tienen un siniestro al año.

Un criterio para identificar un buen ajuste de la clasificación es tener una clase con valor alto de precisión seguido de un valor alto de recall. La *precisión* es de 96% dentro de la clase 0 (pólizas con 0 siniestros), con un valor de *recall* de 99% en la misma clase, indica que está identificando muy bien la clase mayoritaria que es no tener siniestros. La misma tabla nos muestra que las pólizas con un reclamo solo tienen 6% y 2% en *precisión y recall* respectivamente, valores muy bajos. Las peores clasificaciones fueron en las clases minoritarias correspondientes a las pólizas que reportaron dos y más siniestros con ninguna clasificación exitosa.

Como consecuencia de una clasificación pobre en las pólizas con más de un siniestro la media de la siniestralidad de la cartera está siendo infra valorada. El promedio de siniestros estimados es 0.0137965, por lo cual tenemos una media de siniestralidad en realidad muy baja comparada con la observación de prueba real de 0.039069. Lo que es evidente es que la muestra esta desbalanceada, es decir, se acumulan una gran cantidad de datos en una de las clases aportando un mayor número de ejemplos a la hora de entrenar el modelo que genera que el modelo no logre identificar las otras clases y cometa errores. Es necesario realizar un ajuste para dar información al algoritmo de que

[13] Descripción del algoritmo en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

tiene una muestra desbalanceada. Existen varios métodos para generar muestras balanceadas con las originales y o trabajar con muestras desbalanceadas[14], aquí solo se aplican dos porque están disponibles en la configuración del entrenamiento bajo el mismo método y se muestra a continuación.

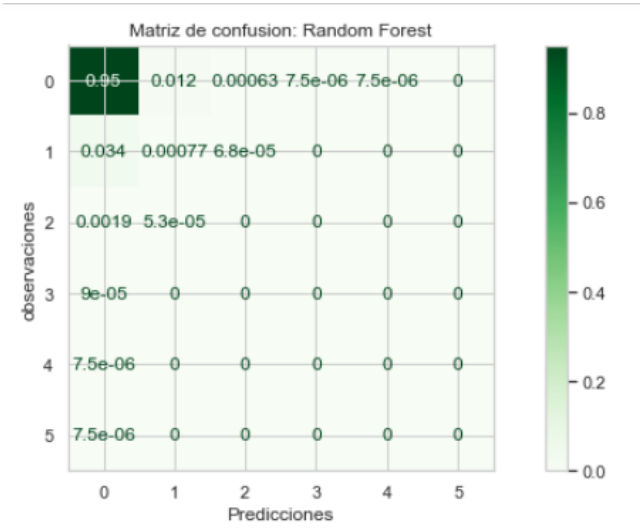


Ilustración 19:Random Forest ajustado con parámetros por defecto. Fuente: Elaboración propia

	precision	recall	f1-score	support
0	0.96	0.99	0.98	128232
1	0.06	0.02	0.03	4649
2	0.00	0.00	0.00	254
3	0.00	0.00	0.00	12
4	0.00	0.00	0.00	1
5	0.00	0.00	0.00	1
accuracy			0.95	133149
macro avg	0.17	0.17	0.17	133149
weighted avg	0.93	0.95	0.94	133149

Tabla 11: Modelo Random Forest configuración por defecto. Fuente: Elaboración propia

[14] Los métodos se pueden agrupar en tres tipologías: 1. Sobre muestreo, 2. Infra muestreo y 3. Penalización; véase https://datascience.foundation/sciencewhitepaper/understanding-imbalanced-datasets-and-techniques-for-handling-them

4.2.1.1. Datos Desbalanceados: Penalización

La penalización es una opción que se encuentra disponible en el método *RandomForestClassifier ()*. Utilizando la frecuencia de entrada de la variable número de siniestros, en este caso la variable dependiente, ajusta automáticamente los pesos inversamente proporcionales a las frecuencias de clase en los datos de entrada siguiendo la expresión $n_{muestras}/(n_{clases} * n_{ocurrencia})$. La opción también permite realizar una discriminación por submuestras, es decir, se aplica en función de la muestra inicial para cada submuestra incluida en cada árbol de decisión. Luego de aplicar el ajuste por penalización los resultados no son satisfactorios.

Lo primero que notamos en las matrices de confusión es que se ha reducido el porcentaje de acierto dentro de la clase con cero siniestros y mejorado muy poco la clase con un solo siniestro. Las matrices de confusión (véase *Ilustraciones 20 y 21)* son muy similares, se destaca que ahora se ha clasificado 1 pólizas, en ambas matrices, que tuvieron 5 siniestros en la clase con cero y uno siniestros, en las matrices son los valores de la parte derecha superior 7.05e-06. El riesgo de la cartera medido como el promedio de siniestralidad es ahora casi el doble del valor original de la muestra de prueba con un valor de 0.06992 y una precisión del 90.367% aplicando la penalización a toda la muestra. Cuando penalizamos por submuestras, dentro de los árboles, los datos no cambian mucho, una media de 0.06979 y una precisión 90.3776%. La *Tabla 12* refleja esta situación.

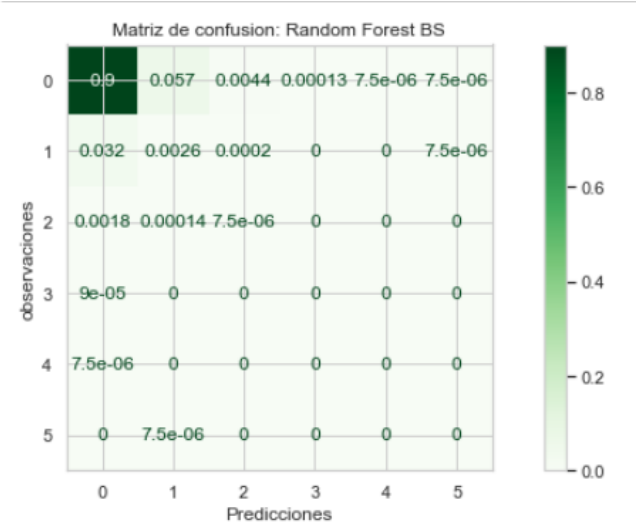


Ilustración 20:Random Forest, Balanceador por submuestras, arboles. Fuente: Elaboración propia

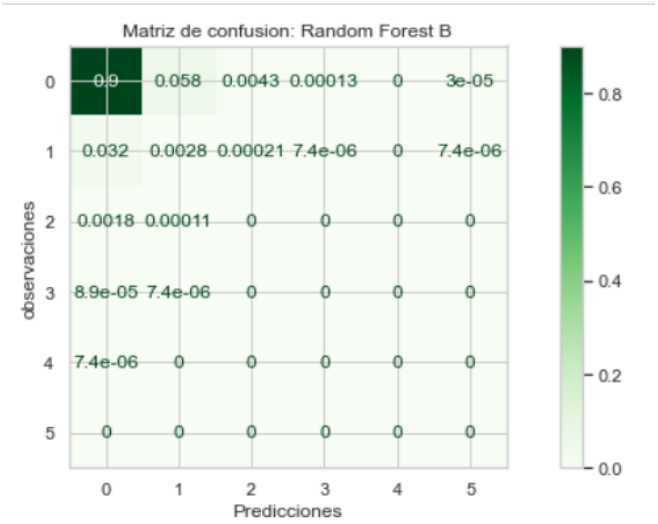


Ilustración 21:Random Forest, balanceado en las observaciones. Fuente: Elaboración propia

Una vez realizado el proceso de penalización, existe otra alternativa para crear modelo más robusto es la validación cruzada. Se aplica la opción que permite ajustar mejor el entrenamiento de los modelos con la modificación de hiperparámetros. Los hiperparámetros son básicamente los valores del algoritmo que se puede modificar, preestableciendo una lista de cambio que se asignan al modelo para estimarse de manera secuencial y luego extraer los mejores resultados. Por cuestiones de tiempo de cálculo solo de modifica el número de árboles, se incluyen la lista de (100, 200, 500, 1000) y se entrena por validación cruzada con el método *GridSearchCV*. La también llamada búsqueda de cuadricula por su traducción al castellano permite automatizar la validación cruzada, permitiendo el entrenamiento de múltiples modelos bajo múltiples combinaciones de hiperparámetros, en esencia es crear un bucle que almacena los resultados que optimizan el entrenamiento.

La aplicación de la validación cruzada genera los resultados indicando que el mejor modelo se ajusta con una cantidad de 1000 árboles, manteniendo los demás valores por defecto del método. Estos resultados presentaran una media de las predicciones de la muestra de 0.01288, claramente inferior al dato original, la precisión es muy similar a la estimada con solo 100 árboles tomando el valor de 95.2391% como se presenta la tabla *12.* Hasta este punto no se puede decir que este sea el modelo más adecuado por lo que seguimos con las demás posibilidades que ofrecen los algoritmos de Machine Learning en la siguiente sección.

Config. Modelo Random Forest	n.º Parámetros	Tiempo de estimación	Promedio de predicciones Obs.= 0.039069	Precisión %
Por defecto	100	26.59s	0.01379	95.166
Balanceado total	100	24.25s	0.06992	90.366
Balanceado submuestras	100	26.88s	0.06979	90.377
Validación Cruzada	1000	4411.85s	0.01288	95.239

Tabla 12: Random Forest, Valores de entrenamiento. Fuente: Elaboración propia

4.2.2. Clasificación con Modelo XGBoost

La técnica denominada Extreme Gradient Boosting (Aumento Extremo del Gradiente, XGB) es un algoritmo basado en arboles de decisión introducido por (Friedman, 2001) que se aplica tanto a regresión como a clasificación, mencionado en la metodología. (Importante, la traducción Boost se toma como sinónimos aumento e impulso). Aplicamos y comparamos con el método Random Forest encontrando una mejora (véase *Ilustración 22)*, aunque tenemos una mayor precisión esta sigue acumulándose en la clase mayoritaria, los perfiles de pólizas sin reclamaciones por siniestros. El 96.2996% de precisión total del modelo está basado en el acierto principalmente de la clase mayoritaria y continua sin ser bien clasificadas las demás clases, lo que causa un valor medio de 0.0001126 siniestros.

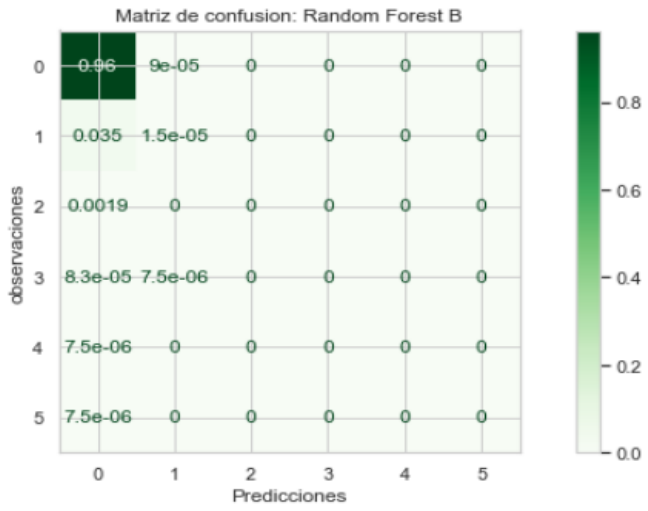


Ilustración 22: XGB Clasificador, entrenado con variables por defecto. Fuente: Elaboración propia

4.2.3. Regresión XGBoost

Generalmente los resultados de la metodología de Embolsado secuencial son mejores que los obtenido con Embolsado en Paralelo. Los resultados del modelo Random Forest son inferiores en términos de tiempo de ejecución y valores de precisión como se puede leer en (Borisov, et. al. 2021). La librería Scitkit-Learn[15] tiene integrado de manera nativa la alternativa para entrenar modelos secuenciales pero se ha utilizado directamente la librería XGBoost[16]. Los datos son los mismos utilizados anteriormente. Por sus diferentes propiedades la distribucion de Poisson es la más utilizada a la hora de modelar conteos y por ello se aplica como funcion de perdida en la regresion en lugar de un error cuadratico de la regresion, este último es un parametros que viene integrado en algunos metodos. El metodo *XGBRegressor (.)* permite establecer como funcion de perdida una distribusion Poisson.

Pasando directo a los resultados luego de aplicar validacion cruzada. Los parametros modificados fueron:

- Funcion objetivo (perdida): Poisson

[15] Para mayor detalle diríjase a: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor
[16] Toda la información en: https://xgboost.readthedocs.io/en/stable/

- Tasa de aprendisaje: (0.01, 0.03, 0.1, 0.3)
- Maximo de profundidad : 5
- Alpha : 10 (termino de regularización L1[17])
- Muestreo de columnas por arbol (colsample_bytree: 0.3)

A demas de los hiperpárametros la validacion cruzada[18] ha incluido:

- Numero de iteraciones de impulso: 50
- Numero de plieges: 3
- Parada temprana: 10
- Metrica de evaluacion: rmse

Se estimaron 50 modelos de los cuales el modelo con un error cuadratico medio menor tiene la configuracion de parametros con una tasa de aprendisaje de 0.3 y los hiperparámetros destacados arriba. Con un valor de RSME (rais cuadrada del error cuadratico medio) de 0.203468, el MAE (media del error adsoluto) es 0.04140 y una Devianza Media de Poisson de 0.246 en los datos de prueba el que denominamos modelo final presenta una media de 0.039086 muy cercana a la media del conjunto de prueba de 0.039069, (véase *Tabla 13)*, y podemos véase en la distribucion del primer nodo como se hizo la clasificacion (Véase *Ilustracion 23*).

Modelo: Función objetivo	**Tiempo**	**MSE**	**MAE**	**Promedio predicciones** **Obs=0.039069**	**Deviance de Poisson**
Poisson	5.8704s	0.04140	0.07401	0.039086	0.246

Tabla 13: XGB Resultados. Fuente: Elaboración propia

[17] Para una explicación de los modelos de regularización vease (Von Lücken, 2021))
[18] Véase el código para mayor detalle en https://xgboost.readthedocs.io/en/stable/python/python_api.html?highlight=.cv#xgboost.cv

La clasificacion de variables de importancia (véase *Ilustracion 24*) pone en primer lugar a BonusMalus como la más influyente seguida de la densidad de la poblacion (Density), luego la antigüedad de los vehiculos 6-12 años, dentro de las primeras 10 variables en la posicion 4 se encuentra las pólizas con conductores de entre 31-40 años.

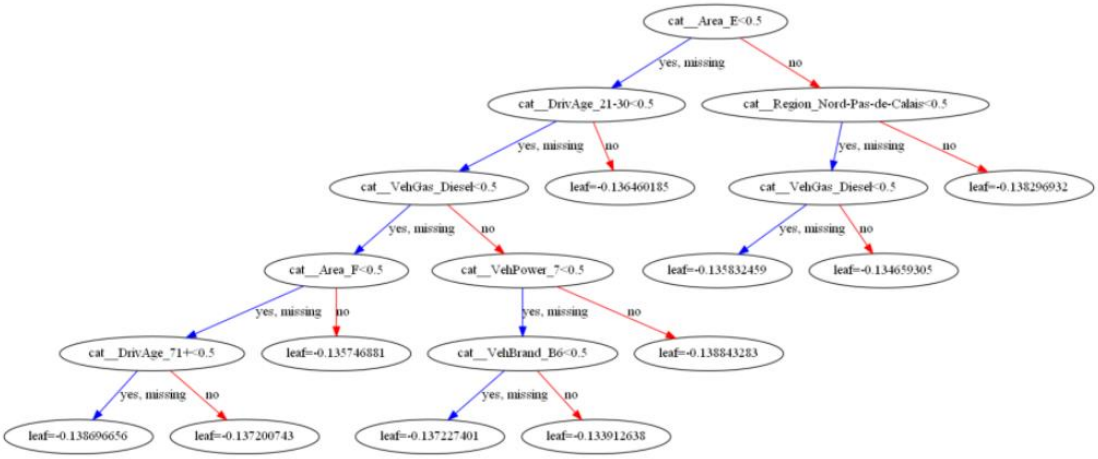


Ilustración 23: XGB Regresión por validación cruzada, arboles de decisión. Fuente: Elaboración propia

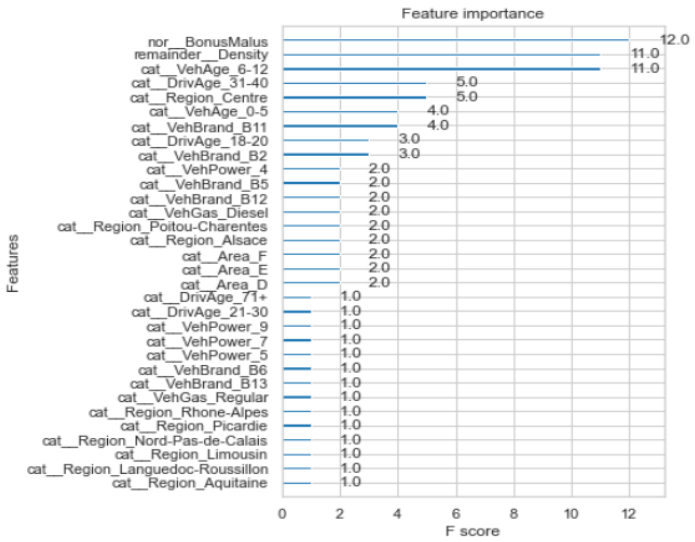


Ilustración 24: XGB Regresor importancia de los predictores, en datos de muestra. Fuente: Elaboración propia

Los resultados luego de entrenar el modelo con el conjunto total de observaciones son consistente con los datos de muestra. El RMSE es de 0.203054 y la influencia de las variables sigue una estructura similar, aunque por ejemplo cambia algunas posiciones de la (*Ilustración 24)* a la (*ilustración 26)*, ejemplo, la antigüedad de vehículos entre 6-12 ahora está en primer lugar, Bonusmalus se ubica ahora una posición más abajo.

Vemos en la (*Ilustración 25)* la composición de uno de los árboles cuando entrenamos toda la cartera.

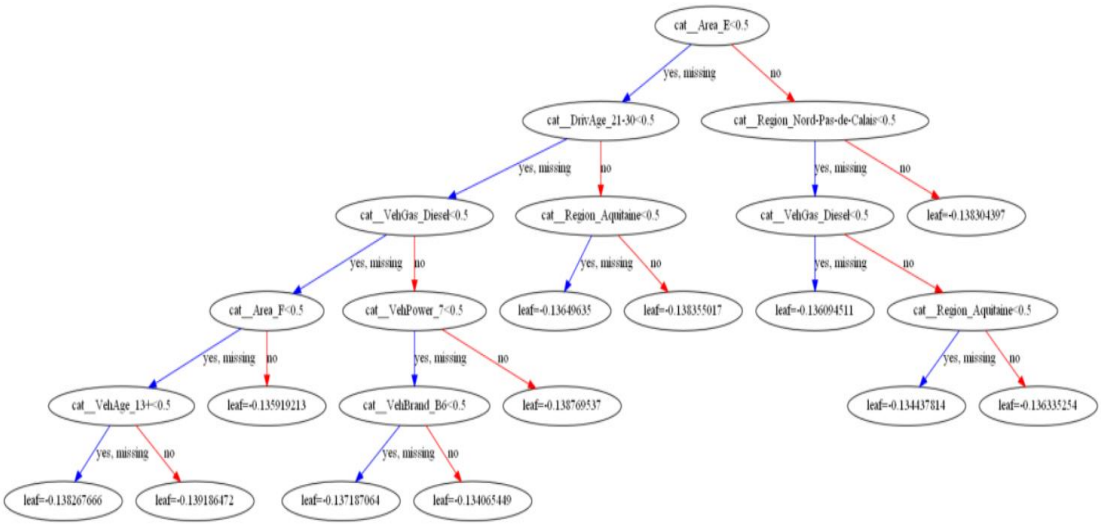


Ilustración 25: XGB Regresión árbol de entrenamiento con todos los datos de la cartera. Fuente: Elaboración propia

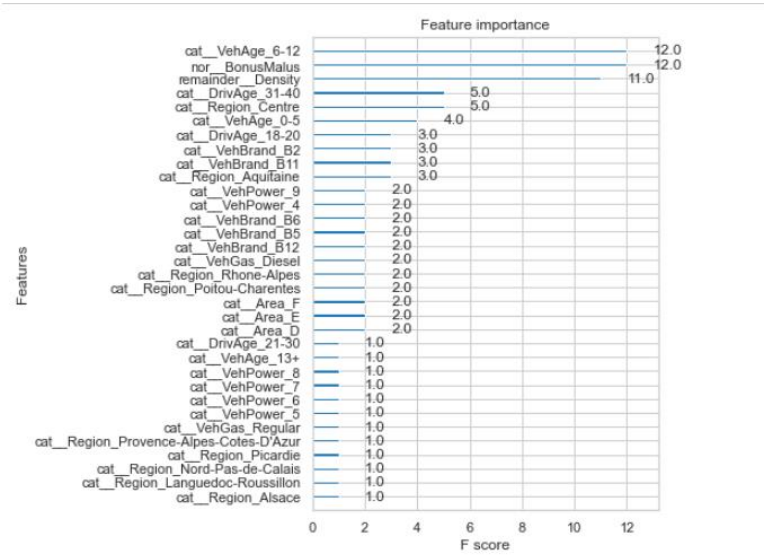


Ilustración 26: XGB Regresión importancia de los predictores en datos totales. Fuente: Elaboración propia

Los resultados en el (Anexo A, véase *Ilustración 31 y 32*) muestran cómo es la frecuencia de las predicciones sobre el número de siniestros. Vemos que es muy similar a los datos de prueba con la diferencia que ahora tiene una composición continua.

4.3 Red Neuronal: Feed Forward Neuronal Network

Para el entrenamiento y manejo de las diferentes tipologías de redes neuronales existen diferentes recursos que según la necesidad se ajustan a cada caso. Según el nivel de complejidad de la red, el volumen de información y los recursos informáticos necesarios para el diseño de las arquitecturas se puede acudir a un programa u otro. Cuando hablamos de redes neuronales con una arquitectura, complejidad media o baja, como lo es la red neuronal multicapa de tipo feed forward, una de las alternativas amigables y eficientes son *Scikit-Learn* para trabajar de manera local y, en remoto se puede utilizar H2O[19]. A medida que la arquitectura de la red es más compleja como pueden ser las redes recurrentes o convolucionales, surgen alternativas para su construcción y entrenamiento. Google y Facebook han desarrollado los entornos y herramientas más populares, *Keras*[20] *y TensorFlow*[21] por Google y *PyTorch*[22] Facebook, para trabajar en Deep Learning. Este trabajo se realiza bajo el entorno de Scikit-Learn por la facilidad y sencillez a la hora de implementar el algoritmo y alternativamente con TensorFlow y Keras se construyen arquitecturas en la búsqueda del mejor resultado.

4.3.1. Parámetros: Definición de la arquitectura

4.3.1.1. Scikit-Learn

El módulo *MLPRegressor* permite construir una red neuronal alimentada hacia adelante para regresión con todos los elementos más importantes. El método ya tiene precargados valores de hiperparámetros por defecto que pueden modificarse y así evaluar diferentes escenarios de ajuste de modelo. Inicialmente es necesario indicar el número de capas ocultas que tendrá el modelo, continuando con la función de activación que, como se mencionan en la metodología, es la que le permite establecer que tan buenas son las predicciones resultado de la estimación comparando con los valores reales, también es llamada función de perdida, (loss function, en inglés). También, se puede indicar cual debe ser la tasa de aprendizaje, esta permite identificar cuando la red deja de aprender,

[19] Véase más información en: https://docs.h2o.ai/h2o/latest-stable/h2o-py/docs/modeling.html#h2odeeplearningestimator
[20] Véase en: https://keras.io/
[21] Véase en: https://www.tensorflow.org/
[22] Véase en :https://pytorch.org/

dicho de otro modo, cuando se detienen la actualización de los pesos en cada época[23], dicha actualización de pesos depende para llegar al valor mínimo de los errores, del denominado 'solver' que funciona como optimizador, generando las actualizaciones de los pesos, el valor por defecto integra el algoritmo denominado '*adam'* que consiste en la combinación de dos algoritmos de optimización[24], el método de gradiente descendiente estocástico o momento y el algoritmo de optimización llamado por sus siglas en ingles RMSProp o rais cuadrada de la media de la propagación (Root Mean Square Propagation, en inglés), la principal función de *Adam* es capturar los pesos y, a la vez que se evalúan y se obtiene el resultado del error, actualizar de nuevo moviéndose hacia adelante hasta llegar al mínimo error que puede tener la relación entre pesos y características para definir el valor en la capa de salida de la red, es decir, actualizar los pesos en función de las predicciones no acertadas (*Véase Ilustración 29).*

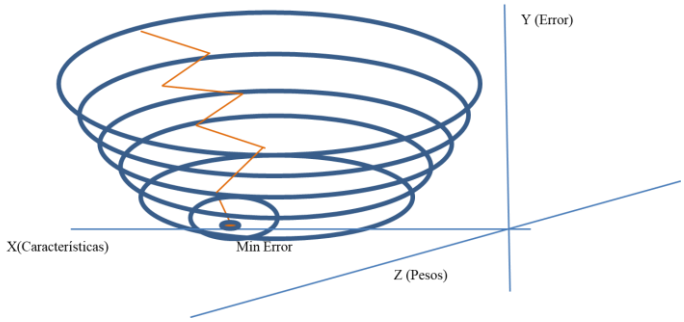


Ilustración 27:'adam' algoritmo de optimización de los pesos. Fuente: Elaboración propia

Teniendo en cuenta que la configuración del método está pensada para optimizar el error cuadrático, es decir, la función de perdida asignada es precisamente la diferencia al cuadrado entre las predicciones y observaciones, ya existe una limitación que se ve reflejada en los resultados. Incluyendo tres capas ocultas (*hidden_layer_sizes*) de (20, 15, 10) neuronas respectivamente, una tasa de aprendizaje o decaimiento exponencial para estimaciones del vector de primer momento en 'adam' es decir, la tasa de actualización de los pesos (*beta_1=0.01*) y la función de activación de 'ReLu' los resultaos de las estimaciones de media de los siniestros reportados y la media observaciones reales indicadas en la tabla (*Tabla 14*). Bajo estas consideraciones el modelo sobreestima la media en el conjunto de prueba. La diferencia en los estimadores

[23] Ciclos de ejecución del algoritmo, repeticiones del proceso de entrenamiento
[24] El optimizador permite especificar la forma exacta en la que se utilizara el gradiente de la pérdida para optimizar los parámetros (véase Chollet, 2017 y Géron, 2019, pág. 466).

del error, MSE y MAE es mínima igual que el tiempo de cálculo y aunque la Deviance de Poisson es igual el modelo con función de activación de *Tanh* es el que tiene mejor resultado de la media por encima del modelo con la función de activación ReLU.

Modelo: *MLPRegressor()*, Función Activación	N.º **Parámetros capas ocultas**	**Tiempo**	**MSE**	**MAE**	**Promedio predicciones** **Obs=0.038431**	**Deviance de Poisson**
ReLU	(20, 15, 10)	12.615s	0.04148	0.07679	0.042157	0.249
Tanh	(20, 15, 10)	14.017s	0.04145	0.07592	0.041151	0.249

Tabla 14: Resultados MLPRegressor, funciones ReLu y Tanh. Fuente: Elaboración propia

4.3.1.2 TensorFlow y Keras

Es posible construir una red neuronal que permita incluir parámetros adicionales a los vistos anteriormente, incluyendo por ejemplo diferentes funciones de activación entre capas ocultas. Para articular la infraestructura de la red como piezas de lego según las necesidades del investigador se encuentra la alternativa de TensorFlow y Keras. Los resultados mejoran al aplicar capa oculta para evitar sobre ajuste y en la capa de salida una función de activación exponencial (Véase *en Ilustración 28*).

```
In [166]: # Modelo 4 capaas densas totalmente conectadas, funcion de activacion Tanh y exponencial.
model = tf.keras.Sequential([
  layers.Dense(20, activation='tanh'),
  layers.Dense(15, activation='tanh'),
  layers.Dropout(.1),
  layers.Dense(1, activation='exponential')
])

model.compile(optimizer='adam',
              loss = tf.keras.losses.poisson,
              metrics=['accuracy','mean_squared_error', 'mean_absolute_error' ])
```

Ilustración 28:Arquitectura de Modelo con funciones de activación Tanh y exponencial. función de perdida Poisson. Fuente: Elaboración propia

En el bloque de código (Véase *Ilustración 28*) se crea una red neuronal densamente conectada (o totalmente conectada) con cuatro capas ocultas, la función optimizadora ‘adam’, función de perdida Poisson. La capa oculta, de nuevo, funcionara como una especie de filtro pasando a la siguiente la extracción de información más útil. Específicamente, se encargan de extraer representaciones de los datos de entrada, dichas

representaciones son lo más significativas para el problema a tratar (Chollet, 2017), en este caso, tomara la información más relevante para calcular el número de siniestros. La primera capa tiene 20 neuronas y una función de activación tangencial 'tanh' al igual que la capa 2 misma función de activación, pero con 15 neuronas. Se incluye también una capa para regularización con tasa de aprendizaje de 0.1, lo que ayuda a evitar el sobreajuste. Finalmente, la última capa que se integra de la neurona de respuesta con una función de activación de tipo 'exponencial'. Hecha la arquitectura se compila el modelo y se entrena incluyendo 30 iteraciones.

Los resultados del entrenamiento son mejorados bajo esta configuración en comparación con la *Tabla 14*. La *Ilustración 29* compila el registro del proceso de entrenamiento y el tiempo de ejecución donde se observa que en 4.7 minutos aproximadamente se realizaron los cálculos.

```
In [43]: # Entrenamiento del modelo  Poisson, Perdida Poisson, Activación tanh y salida exponencial
         tic = time.time()# huella de tiempo de entrada
         historial = model.fit(X_train, y_train,
                     validation_data=(X_val, y_val),
                     epochs=30, verbose=1)

         tac = time.time() # huella de tiempo de salida
         print('------------------------------------------------')
         print('Tiempo de Calculo:  {:.4f}m' .format((tac-tic)/60 ))
         print('Terminado')
         Epoch 27/30
         13315/13315 [==============================] - 9s 697us/step - loss: 0.1607 - accuracy: 0.9627 - mean_squared_error: 0.0417 -
         mean_absolute_error: 0.0747 - val_loss: 0.1583 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 - val_mean_absolute_er
         ror: 0.0725
         Epoch 28/30
         13315/13315 [==============================] - 9s 700us/step - loss: 0.1606 - accuracy: 0.9627 - mean_squared_error: 0.0417 -
         mean_absolute_error: 0.0747 - val_loss: 0.1587 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 - val_mean_absolute_er
         ror: 0.0743
         Epoch 29/30
         13315/13315 [==============================] - 9s 705us/step - loss: 0.1606 - accuracy: 0.9627 - mean_squared_error: 0.0417 -
         mean_absolute_error: 0.0747 - val_loss: 0.1584 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 - val_mean_absolute_er
         ror: 0.0742
         Epoch 30/30
         13315/13315 [==============================] - 9s 703us/step - loss: 0.1605 - accuracy: 0.9627 - mean_squared_error: 0.0417 -
         mean_absolute_error: 0.0747 - val_loss: 0.1582 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 - val_mean_absolute_er
         ror: 0.0732

         ------------------------------------------------
         Tiempo de Calculo:  4.6716m
         Terminado
```

Ilustración 29: Resultados del entrenamiento Modelo Poisson, activación Tanh y Exponencial. Fuente: Elaboración propia

En las primeras iteraciones de 0-15 el modelo aprende a un ritmo mayor que en las siguientes. La *Ilustración 30* muestra cómo se reduce el margen de error lo que indica que en cada época el modelo es más preciso, además, parece que puede mejorar aún más si se incluyeran más iteraciones, aunque con menor rapidez.

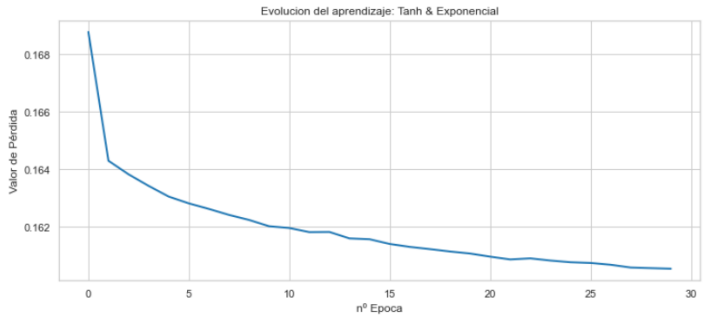


Ilustración 30: Evolución de Aprendizaje, Modelo Tanh, Exponencial. Fuente: Elaboración propia

Estimar un modelo con una capa oculta más y la función de activación de tipo ReLU no mejoran los resultados. Vemos en la *Tabla 15* los resultados de los dos modelos probando diferentes funciones de activación, el modelo que mejor se ajusta tiene la configuración de funciones Tanh y Exponencial, el primer modelo, la estimación de la media bajo esta configuración esta ligeramente sobre estimada comparada con el valor observado real. Es modelo es más eficiente en términos del número de parámetros estimados pues solo necesita estimar 1,491. También se evidencia que la Deviance, MAE y MSE es menor en comparación con el segundo modelo con función de activación ReLU[25].

Modelo: *TensorFlow.Keras*, Función Activación	**N.º Parámetros**	**Tiempo**	**MSE**	**MAE**	**Promedio predicciones Obs=0.038431**	**Deviance de Poisson**
Tanh & Exp.	1,491	4.67m	0.0414	0.0727	0.038653	0.249
ReLU & Exp.	2,815	4.69m	0.0415	0.0746	0.039912	0.250

Tabla 15: Resultados el modelo con TensorFlow y Keras, Funciones de activación Tanh, ReLU y Exponencial en los datos de prueba. Fuente: Elaboración propia

[25] Para mayor detalle véase código en Anexo B.

4.4. Compilación de Resultados.

El modelo que predice mejor el promedio del número de siniestros es el modelo XGBoost. El resultado de la diferencia entre la media aritmética de las predicciones de los modelos y, las observaciones promedio incluidas como conjunto de prueba se muestra en la *Tabla 16*. Con una clara diferencia el modelo XGBoost tiene el mejor resultado, seguido por el modelo glm_2, glm_1 y en último lugar en la tabla la diferencia mayor la tiene el modelo basado en redes neuronales. El tiempo aproximado de estimación es similar en todos los casos excepto para las redes neuronales que tienen una diferencia muy alta superando los 4 minutos, con la misma cantidad de datos, además, de estimar 1491 parámetros, número desproporcionada en comparación con los datos incluidos en la tabla. El algoritmo XGBoost necesito la misma cantidad de tiempo que los modelos GLM y teniendo en cuenta que por su construcción estos tienen una menor varianza porque se enfocan en la precisión reduciendo la diferencia entre las estimaciones y las observaciones secuencialmente, con las condiciones actuales tiene el mejor rendimiento.

Modelo: *** **GLM** ****Redes Neuronal** ***XGBoost**	**N.º Parámetros**	**Tiempo**	**Diferencia entre Promedios de predicciones y el observado:** $\hat{\mu} - \mu_{obs.}$
***XGBoost Poisson**	100	5.8s	0.000017
****Tanh & Exp.**	1,491	4.67m	0.000222
*****glm_1**	48	5s	0.000211
***** glm_2**	48	6s	0.000157

Tabla 16: Resumen de modelos; diferencias entre el valor promedio estimado y promedio de observaciones. Fuente: Elaboración propia

5. Conclusiones

El presente trabajo se realiza con la intención de dar una guía a la aplicación de las metodologías utilizadas en la ciencia de datos y que provienen de la inteligencia artificial. El mismo constituye una alternativa para el análisis de datos de carteras de seguros no vida en el ámbito asegurador, se han estimado modelos GLM, Random Forest, XGBoost y Redes Neuronales de tipo Feed Forward Neuronal Network, generándose las principales conclusiones a continuación.

En primer lugar, es importante destacar que las metodologías aplicadas son una alternativa con un fuerte respaldo por su aplicación en diversos campos de la automatización y que tienen un auge que se consolida por los resultados de precisión como puede verse en (Borisov et al., 2021). También, es necesario conocer el alcance y limitaciones de los modelos para generar resultados consistentes, las metodologías han sido pensadas para determinadas tareas inicialmente por lo que tienen limitaciones y ventajas. En el apartado de metodología se presenta la extracción de información que se considera necesaria para entender el objetivo y utilidad de cada modelo. Por otra parte, en el apartado de estimación y resultado se presenta la serie de modelos y resultados obtenidos en la estimación del número de siniestros.

En segundo lugar, se observa que el modelo XGBoost tiene los mejores resultados entre los modelos incluidos. Los modelos GLM cuentan con una tradición, confianza y respaldo por parte de la industria aseguradora en términos de medición de riesgos que ahora pueden complementarse con métodos y algoritmos que provienen de la inteligencia artificial. Los modelos XGBoost pueden ser una alternativa comparable en términos de precisión con los GLM, tienen una fácil implementación y pueden identificarse las variables con mayor importancia e influencia en los resultados, además su arquitectura permite tener una reducción de la varianza en las predicciones. Se destaca que los mejores resultados son obtenidos por el algoritmo XGBoost incluidos en la tabla resumen en el apartado 4.4. de compilación de modelo. No se considera los modelos Random Forest porque los resultados obtenidos bajo las mismas condiciones que los XGBoost están muy sesgados.

En tercer lugar, la eficiencia de los modelos XGBoost es mejor en comparación con los modelos de redes neuronales y equiparable a los GLM. Deep Learning tiene como una

de sus características principales que no es necesario realizar lo que en Machine Learning se denomina ingeniería de características que es un proceso crucial, pero es muy costoso computacionalmente. Las estimaciones realizadas con las redes neuronales a pesar ser muy aproximadas tienen un gran consumo de recursos computacional y de tiempo que limita la aplicación de los modelos neuronales, bajo condiciones similares, es decir, misma cantidad de datos, las redes neuronales de tipo Feed Forward presentaron un tiempo estimado muy prolongado, debido a que es necesario estimar muchos más parámetros las redes son menos eficientes en tiempo. A lo anterior se le suma que, para la fecha de realización de este trabajo, no es posible conocer como es la interacción que tienen las variables, es decir, cuáles son las que más influencia tienen por lo que reciben el nombre de cajas negras.

Por último, debido a que solo con un experimento es precipitado decidir cual modelo ajusta mejor, entre los modelos XGBoost y GLM se motiva a poner a prueba en otros escenarios comparativos la eficiencia y eficacia de los modelos como futuras líneas de investigación. La flexibilidad y alternativas que presenta los algoritmos para poder incluir, por ejemplo, una función de perdida de tipo Poisson y que pueden automatizarse puede considerarse como una alternativa viable a la hora de contrastar modelos. Los árboles de decisión, en especial para este trabajo el XGBoost, pueden servir de alternativa para entender la segmentación de la cartera, analizando como se despliegan y se agrupan en los nodos las submuestras.

Finalmente, existen diferentes métodos que aquí no se han incluido para compararse los resultados tales como el algoritmo de LightGBM[26] y CatBoost[27] que se encuentran en la revisión bibliográfica. Estos métodos también son basados en los denominados modelos de ensamble y se encuentran desarrollados en (Borisov et al., 2021). También probar con transformaciones de las variables y probar con la configuración de las redes neuronales que permita mejorar los resultados y que no sea una terea con alto consumo computacional.

[26] Véase en (Borisov, et. al. 2021) y en página web oficial: https://lightgbm.readthedocs.io/en/latest/
[27] Véase en (Borisov, et al. 2021) y en página web oficial: https://catboost.ai/

Bibliografía

Boj, E., Claramunt Bielsa, M., & Costa Cor, T. (2020). *Tarificacion y Provisiones.* Barcelona: Univéasesidad de Barcelona.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., & Kasneci, G. (2021). Deep Neural Networks and Tabular Data: A Survey. *arXiv:2110.01889 [cs.LG]*.

Breiman, L. (2001). Random Forest. *Machine Learning*, 45/1, 5-32.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees.* New York: Chapman & Hall/CRC.

Chollet, F. (2017). *Deep Learning with Python.* New York: Manning Publications Co.

De Jong , P., & Heller, G. (2008). *Generalized Linear Models for Insurance Data.* Cambridge: Cambridge Univéasesity Press.

Friedman, J. H. (October de 2001). *Greedy function approximation: A gradient boosting machine.* Obtenido de https://doi.org/10.1214/aos/1013203451

Géron, A. (2019). *Hand-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* Beijing: Oreilly Media Inc.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT press.

Pujol Vila, O. (2021). Machine Learning II. *Non-linear Models and Ensemble Learning.* Barcelona.

Vitrià, J. (2022). Neuronal Networks. Barcelona.

Von Lücken, J. I. (2021). *Métodos de Regularizaciòn Lasso, Ridge y Elastic Net: Una aplicacion a los seguros no vida.* Obtenido de Univéasesidad de Barcelona, Deposito digital: http://hdl.handle.net/2445/179200

Wüthrich, M., & Merz, M. (31 de Enero de 2022). *Statistical Foundations of Actuarial Learning and its Applications.* Obtenido de SSRN: https://ssrn.com/abstract=3822407

Anexo A

1.

```
print(summary(glm_1))

Call:
glm(formula = ClaimNb ~ VehPowerGLM + VehAgeG + DrivAgeG + BonusMalusG +
    VehBrand + VehGas + DensityG + Region + Area, family = poisson,
    data = learn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1505  -0.2990  -0.2618  -0.2232   6.4440

Coefficients:
                                    Estimate Std. Error z value Pr(>|z|)
(Intercept)                       -4.8087429  0.1292022 -37.219  < 2e-16 ***
VehPowerGLM5                       0.0707691  0.0230097   3.076  0.00210 **
VehPowerGLM6                       0.1010213  0.0225358   4.483 7.37e-06 ***
VehPowerGLM7                       0.0696916  0.0223497   3.118  0.00182 **
VehPowerGLM8                       0.0768758  0.0317199   2.424  0.01537 *
VehPowerGLM9                       0.1895051  0.0250311   7.571 3.71e-14 ***
VehAgeG6-12                        0.0660175  0.0155247   4.252 2.11e-05 ***
VehAgeG13+                        -0.2396979  0.0201663 -11.886  < 2e-16 ***
DrivAgeG21-25                     -0.3418556  0.0492562  -6.940 3.91e-12 ***
DrivAgeG26-30                     -0.4654403  0.0482580  -9.645  < 2e-16 ***
DrivAgeG31-40                     -0.2219804  0.0464348  -4.780 1.75e-06 ***
DrivAgeG41-50                      0.1226886  0.0471129   2.604  0.00921 **
DrivAgeG51-70                      0.1231598  0.0472821   2.605  0.00919 **
DrivAgeG71+                        0.2599672  0.0528776   4.916 8.82e-07 ***
BonusMalusG                        0.0222057  0.0004026  55.160  < 2e-16 ***
VehBrandB2                         0.0155504  0.0181749   0.856  0.39222
VehBrandB3                         0.0457395  0.0251377   1.820  0.06883 .
VehBrandB4                         0.0262739  0.0343381   0.765  0.44418
VehBrandB5                         0.0969656  0.0290946   3.333  0.00086 ***
VehBrandB6                         0.0135741  0.0327084   0.415  0.67814
VehBrandB10                       -0.0039316  0.0415621  -0.095  0.92464
VehBrandB11                        0.1169675  0.0439738   2.660  0.00782 **
VehBrandB12                       -0.5021131  0.0236064 -21.270  < 2e-16 ***
VehBrandB13                        0.0345016  0.0473217   0.729  0.46595
VehBrandB14                       -0.1953298  0.0928406  -2.104  0.03538 *
VehGasRegular                     -0.1061334  0.0140591  -7.549 4.38e-14 ***
DensityG                           0.0348496  0.0148918   2.340  0.01927 *
RegionAquitaine                   -0.1392444  0.1139142  -1.222  0.22157
RegionAuvergne                    -0.2402849  0.1417825  -1.695  0.09012 .
RegionBasse-Normandie              0.0300361  0.1198800   0.251  0.80216
RegionBourgogne                   -0.1246588  0.1230644  -1.013  0.31108
RegionBretagne                     0.0968398  0.1119428   0.865  0.38699
RegionCentre                       0.0015475  0.1103057   0.014  0.98881
RegionChampagne-Ardenne           -0.1855483  0.1608635  -1.153  0.24873
RegionCorse                       -0.0919620  0.1428410  -0.644  0.51970
RegionFranche-Comte               -0.2387255  0.2013587  -1.186  0.23579
RegionHaute-Normandie             -0.5113793  0.1304627  -3.920 8.86e-05 ***
RegionIle-de-France               -0.1464906  0.1113362  -1.316  0.18826
RegionLanguedoc-Roussillon        -0.2947312  0.1140733  -2.584  0.00977 **
RegionLimousin                     0.2387082  0.1335071   1.788  0.07378 .
RegionMidi-Pyrenees               -0.5018198  0.1225642  -4.094 4.23e-05 ***
RegionNord-Pas-de-Calais          -0.1440878  0.1126460  -1.279  0.20085
RegionPays-de-la-Loire            -0.0157105  0.1124155  -0.140  0.88885
RegionPicardie                    -0.0063187  0.1245473  -0.051  0.95954
RegionPoitou-Charentes             0.0460643  0.1155448   0.399  0.69014
RegionProvence-Alpes-Cotes-D'Azur -0.0686515  0.1108246  -0.619  0.53561
RegionRhone-Alpes                  0.1423403  0.1103793   1.290  0.19720
Area                               0.0347166  0.0200210   1.734  0.08292 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 158221  on 610205  degrees of freedom
Residual deviance: 152815  on 610158  degrees of freedom
AIC: 198689

Number of Fisher Scoring iterations: 6
```

Tabla A 1: Modelo glm_1, resultados coeficientes. Fuente: Elaboración propia

2.

```
summary(glm_2)

Call:
glm(formula = ClaimNb ~ VehPowerGLM + VehAgeG + DrivAgeG + BonusMalusG +
    VehBrand + VehGas + DensityG + Region + Area, family = poisson,
    data = learn, offset = log(Exposure))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.4991  -0.3262  -0.2456  -0.1383   7.7830

Coefficients:
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                     -4.4026729  0.1290559 -34.114  < 2e-16 ***
VehPowerGLM5                     0.0539821  0.0230161   2.345 0.019006 *
VehPowerGLM6                     0.0917432  0.0225288   4.072 4.66e-05 ***
VehPowerGLM7                     0.0833632  0.0223660   3.727 0.000194 ***
VehPowerGLM8                     0.1118720  0.0317714   3.521 0.000430 ***
VehPowerGLM9                     0.2396135  0.0251195   9.539  < 2e-16 ***
VehAgeG6-12                     -0.0153484  0.0154186  -0.995 0.319519
VehAgeG13+                      -0.2766311  0.0200255 -13.814  < 2e-16 ***
DrivAgeG21-25                   -0.4651509  0.0492650  -9.442  < 2e-16 ***
DrivAgeG26-30                   -0.5829329  0.0482728 -12.076  < 2e-16 ***
DrivAgeG31-40                   -0.3914048  0.0464294  -8.430  < 2e-16 ***
DrivAgeG41-50                   -0.1069522  0.0470634  -2.273 0.023055 *
DrivAgeG51-70                   -0.1829268  0.0471848  -3.877 0.000106 ***
DrivAgeG71+                     -0.1804836  0.0527196  -3.423 0.000618 ***
BonusMalusG                      0.0272392  0.0003862  70.535  < 2e-16 ***
VehBrandB2                       0.0069046  0.0181740   0.380 0.704008
VehBrandB3                       0.0693487  0.0251519   2.757 0.005830 **
VehBrandB4                       0.0429750  0.0343595   1.251 0.211027
VehBrandB5                       0.0865620  0.0291087   2.974 0.002942 **
VehBrandB6                       0.0357448  0.0327167   1.093 0.274590
VehBrandB10                      0.0247009  0.0416301   0.593 0.552952
VehBrandB11                      0.1957886  0.0439889   4.451 8.55e-06 ***
VehBrandB12                     -0.2611214  0.0235782 -11.075  < 2e-16 ***
VehBrandB13                      0.0565415  0.0473322   1.195 0.232256
VehBrandB14                     -0.1798543  0.0928557  -1.937 0.052755 .
VehGasRegular                   -0.1722812  0.0140513 -12.261  < 2e-16 ***
DensityG                         0.0497928  0.0148891   3.344 0.000825 ***
RegionAquitaine                  0.0672015  0.1139266   0.590 0.555280
RegionAuvergne                  -0.0586404  0.1417872  -0.414 0.679182
RegionBasse-Normandie           -0.0052915  0.1198855  -0.044 0.964794
RegionBourgogne                  0.0432858  0.1230965   0.352 0.725107
RegionBretagne                   0.0163224  0.1119330   0.146 0.884061
RegionCentre                    -0.0079554  0.1103169  -0.072 0.942511
RegionChampagne-Ardenne          0.0289639  0.1608821   0.180 0.857128
RegionCorse                      0.1654242  0.1429063   1.158 0.247039
RegionFranche-Comte             -0.0366955  0.2013597  -0.182 0.855395
RegionHaute-Normandie           -0.0285756  0.1304852  -0.219 0.826654
RegionIle-de-France             -0.0187598  0.1113580  -0.168 0.866218
RegionLanguedoc-Roussillon       0.0008798  0.1140816   0.008 0.993847
RegionLimousin                   0.3262905  0.1335349   2.443 0.014546 *
RegionMidi-Pyrenees             -0.2483135  0.1225947  -2.025 0.042818 *
RegionNord-Pas-de-Calais        -0.0044459  0.1126527  -0.039 0.968519
RegionPays-de-la-Loire           0.0208918  0.1124193   0.186 0.852572
RegionPicardie                   0.1238412  0.1245545   0.994 0.320090
RegionPoitou-Charentes           0.0576365  0.1155464   0.499 0.617908
RegionProvence-Alpes-Cotes-D'Azur 0.1229957 0.1108309   1.110 0.267103
RegionRhone-Alpes                0.1983406  0.1103754   1.797 0.072341 .
Area                             0.0386365  0.0200443   1.928 0.053910 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 154252  on 610205  degrees of freedom
Residual deviance: 147395  on 610158  degrees of freedom
AIC: 193269

Number of Fisher Scoring iterations: 6
```

Tabla A 2: Modelo glm_2, resultados coeficientes. Fuente: Elaboración propia

3.

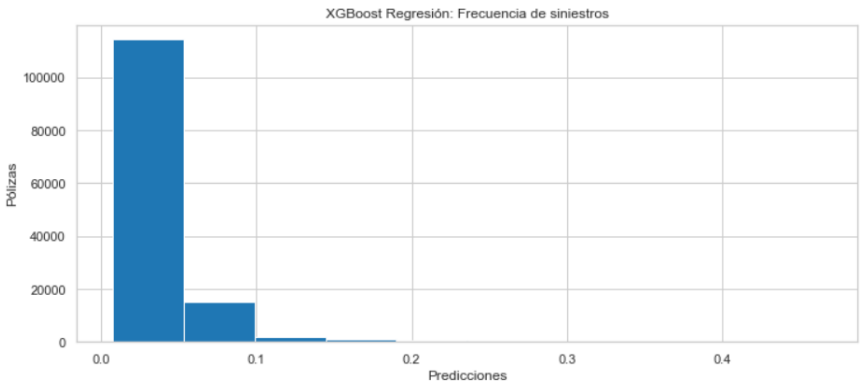


Ilustración 31: XGB Predicciones en conjunto de prueba. Fuente: Elaboración propia

4.

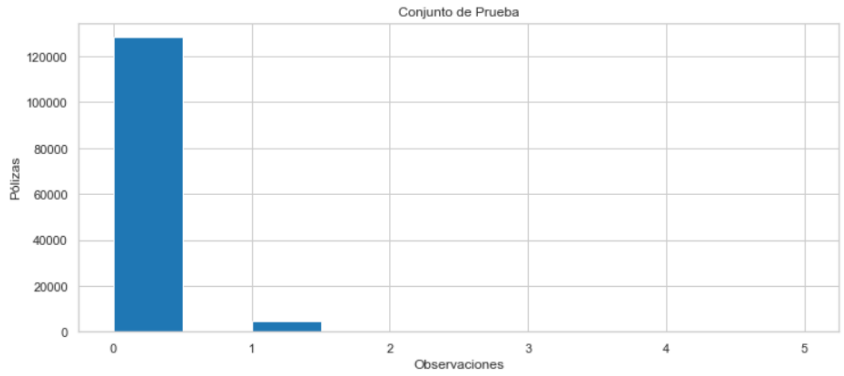


Ilustración 32: Conjunto de Prueba. Fuente: Elaboración propia

Anexo B
Código en R y Python

Código R

```
#################################################################
######## Descarga de librerías ###################################
#################################################################
install.packages("effects")
install.packages("OpenML")
install.packages("farff")
install.packages("CASdatasets", repos = "http://cas.uqam.ca/pub/",
            type="source")
#################################################################
######## Descarga de datos ######################################
#################################################################
library(knitr)
library(AER)
library(ggplot2)
library(OpenML)
library(farff)
library ( CASdatasets )
library(effects)
library("CASdatasets")
data("freMTPLfreq")
head(freMTPLfreq)
```

```
### Carga de datos ###

DatosMutuas<-read.csv(file="http://www.uv.es/pavia/seguros_no_vida/mutuas.csv",sep=";")

data(freMTPLsev)

data ( freMTPL2freq )

dat <- freMTPL2freq [, -2]

dat$VehGas <- factor ( dat$VehGas )

data ( freMTPL2sev )

sev <- freMTPL2sev

sev$ClaimNb <- 1

dat0 <- aggregate (sev , by= list ( IDpol = sev$IDpol ), FUN = sum )[c (1 ,3:4)]

names ( dat0 )[2] <- " ClaimTotal "

dat <- merge (x=dat , y=dat0 , by ="IDpol", all.x= TRUE )

dat [is.na( dat )] <- 0

dat <- dat [ which ( dat$ClaimNb <=5) ,]

dat$Exposure <- pmin ( dat$Exposure , 1)

sev <- sev [ which ( sev$IDpol %in% dat$IDpol ), c (1 ,2)]

dat$VehBrand <- factor ( dat$VehBrand , levels =c("B1","B2","B3","B4","B5","B6",
"B10","B11","B12","B13","B14"))

write.csv(dat, file = 'df_Mof')

freMTPL2freq1 <- getOMLDataSet(data.id = 41214)$data

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

######## Modificación de la data para cálculos #################

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

dat$DensityG    <- log(dat$Density)#Logaritmo

dat$VehPowerGLM <- as.factor(pmin(dat$VehPower, 9))#Reduccion de dimensiones de
potencia de coche

dat$Area       <- as.integer(dat$Area)#multicalse

dat$VehAgeG    <- as.factor(cut(dat$VehAge, c(0,5,12,101),

                        labels = c("0-5", "6-12", "13+"),

                        include.lowest = TRUE))
```

```
dat$DrivAgeG   <- as.factor(cut(dat$DrivAge, c(18,20,25,30,40,50,70,101),
                    labels = c("18-20", "21-25","26-30","31-40", "41-50", "51-70", "71+"),
                    include.lowest = TRUE))
dat$BonusMalusG  <- pmin(dat$BonusMalus, 150)
dat$ClaimNbG <- as.factor(dat$ClaimNb)
head(dat)
#####################################################################
######################### Division train y test #####################
#####################################################################
set.seed (33)
ll <- sample (c (1: nrow (dat )), round (0.8* nrow ( dat )), replace = FALSE )
learn <- dat [ll ,]
test <- dat [-ll ,]
str(learn$Region)
str(test)
head(learn$Region)
hist(learn$ClaimNb, nclass = 10)
str(learn)#informacion de la data
#############################################################################
#################### Models  ################################################
#############################################################################
glm_1 = glm(ClaimNb ~ VehPowerGLM + VehAgeG + DrivAgeG + BonusMalusG  + VehBrand+
VehGas +DensityG + Region + Area,
        data = learn,
        family = poisson
       )
glm_2 = glm(ClaimNb ~ VehPowerGLM + VehAgeG + DrivAgeG + BonusMalusG  + VehBrand+
VehGas +DensityG + Region + Area,
        data = learn,
        offset = log(Exposure),
        family = poisson
```

```
)
############################################################
######## Resultados  ########################################
############################################################
summary(glm_1)
summary(glm_2)
#Obtener los coeficientes estimados
coef(glm_1)
coef(glm_2)
############################################################
######## DEVIANCE ###########################################
############################################################
d_null_1 <- glm_1$null.deviance
d_null_2 <- glm_2$null.deviance
d_res_1 <-glm_1$deviance
d_res_2 <-glm_2$deviance
Deviance_1 = (d_null_1-d_res_1)*100/d_null_1
Deviance_2 = (d_null_2-d_res_2)*100/d_null_2
############################################################
######## AIC`s ##############################################
############################################################
AIC_1 <- AIC(glm_1)
AIC_2 <- AIC(glm_2)
############################################################
######## Prueba de Dispersión ###############################
############################################################
dispersiontest(glm_1)
dispersiontest(glm_2)
######################
mean(glm_1$fitted.values)
```

```
mean(test$ClaimNb)
############################################################
######## Gráfico de residuales  ############################
############################################################
plot(glm_1$residuals)
hist(resid(glm_1))
par(c(2,2))
plot(glm_1, which=c(1,2,3))
plot(resid(glm_1)~test)
############################################################
######## Grafico Efectos parciales ##########################
############################################################
win.graph()
plot(allEffects(glm_1))
win.graph()
plot(allEffects(glm_2))
############################################################
######## Predicciones  #####################################
############################################################
mean(predict.glm(glm_1, type = 'response', newdata = data.frame(test)))
mean(dat$ClaimNb)#prediccion sobre la muestra total
mean(predict.glm(glm_2, type = 'response', newdata = data.frame(test)))
mean(predict.glm(glm_1, type = 'response'))
mean(predict.glm(glm_2, type = 'response'))
############################################################
######## Análisis Anova ####################################
############################################################
anova(glm_1)
anova(glm_2)
```

Código Python
-
RF_XGB

June 7, 2022

[1]:
```
%reset -f
%clear
```

1 Random Forest and XG Boost

1.1 Importacion de librerias

[2]:
```
#importacion de librerias y modulos
import pandas as pd
import numpy as np
import matplotlib as mpl
import statsmodels.api as sm
import time

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_poisson_deviance


from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import StratifiedShuffleSplit
import xgboost as xgb
from xgboost import XGBRegressor
from xgboost import XGBClassifier
import multiprocessing
from sklearn.model_selection import RepeatedKFold
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from imblearn.combine import SMOTETomek
from sklearn.model_selection import ParameterGrid
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import FunctionTransformer
```

1.2 Configuración de salidas

[3]:
```
import warnings
warnings.filterwarnings('ignore')

# configuracion de la visualizacion
import matplotlib.pylab as plt
%matplotlib inline
plt.rc('font', size=12)
plt.rc('figure', figsize = (12, 5))

import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=1, rc={"lines.linewidth": 2,'font.family':
 ↪ [u'times']})

# configuracion de la data
pd.set_option('display.max_rows', 25)
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_colwidth', 50)
```

1.3 Imporación y Procesamiento de la data

[4]:
```
# importacion de datos

df = pd.read_table("C:/Users/JHONNY/OneDrive/TFM_CAF/df_Mof.txt", sep= ",")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 678007 entries, 0 to 678006
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Unnamed: 0    678007 non-null  int64
 1   IDpol         678007 non-null  float64
```

```
 2   Exposure      678007 non-null  float64
 3   VehPower      678007 non-null  int64
 4   VehAge        678007 non-null  int64
 5   DrivAge       678007 non-null  int64
 6   BonusMalus    678007 non-null  int64
 7   VehBrand      678007 non-null  object
 8   VehGas        678007 non-null  object
 9   Area          678007 non-null  object
 10  Density       678007 non-null  int64
 11  Region        678007 non-null  object
 12   ClaimTotal   678007 non-null  float64
 13  ClaimNb       678007 non-null  int64
dtypes: float64(3), int64(7), object(4)
memory usage: 72.4+ MB
```

[5]:
```
df = df.loc[df.VehAge<20]
df = df.loc[df.DrivAge<=90]
df = df.loc[df.BonusMalus<=150]

df.VehAge = np.where(df.VehAge>13, 14, df.VehAge)
df.DrivAge = np.where(df.DrivAge>70, 71, df.DrivAge)
df.VehPower = np.where(df.VehPower>9, 9, df.VehPower)
df.Density = np.log(df.Density)


df['VehAge'] = pd.cut(df.VehAge,
                              bins= [0., 5, 12,  np.inf],
                              labels=['0-5', '6-12', '13+'])

df['DrivAge'] = pd.cut(df.DrivAge,
                              bins= [18., 20, 30, 40, 50, 70, np.inf],
                              labels=['18-20', '21-30', '31-40', '41-50','51-70',␣
 ↪'71+'])
df.reset_index(drop=True, inplace=True)
```

1.3.1 División de datos en prueba y entrenamiento

[6]:
```
#Set de entrenamiento(train) y prueba (test)
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=33)
for train_index, test_index in split.split(df, df['ClaimNb']):
    strat_train_set = df.loc[train_index]
    strat_test_set = df.loc[test_index]
```

1.3.2 Separación de datos y aplicación de transformaciones

[7]:
```
#Set train
y_train = strat_train_set['ClaimNb'].copy()
X_train = strat_train_set.drop(['Unnamed: 0','ClaimNb', 'IDpol' ,' ClaimTotal ',␣
↪'Exposure'], axis=1)
X_train.head()

#Set test
X_test = strat_test_set.drop(['Unnamed: 0', 'ClaimNb', 'IDpol' ,' ClaimTotal ',␣
↪'Exposure' ], axis=1)
y_test = strat_test_set['ClaimNb'].copy()
```

[8]:
```
#Procesamiento de la data
# SELECCION DE LA DATA PARA TRANSFORMAR
# Train
cat_item = X_train[[ 'Area', 'Region','VehGas', 'VehBrand','VehPower', 'VehAge',␣
↪ 'DrivAge']] #CATEGORICAS
norm_item = X_train[['BonusMalus']] #NORMALIZACION


# Test
cat_item_test = X_test[['Area', 'Region', 'VehGas', 'VehBrand','VehPower',␣
↪'VehAge',  'DrivAge' ]] #CATEGORICAS
norm_item_test = X_test[['BonusMalus']] #NORMALIZACION

#Damos formato a los datos para introducir a los calculos.

full_pipeline = ColumnTransformer([("nor", MinMaxScaler(), list(norm_item)),
                                   ("cat", OneHotEncoder(), list(cat_item)),
                                   ], remainder = 'passthrough')
```

[9]:
```
#Transformamos la data, a categorica con onehotencoder y minmaxscaler
data_0 = full_pipeline.fit_transform(X_train)
data_test = full_pipeline.transform(X_test)
```

[10]:
```
#Convertimos a formato pandas.DataFrame
df_prepared = pd.DataFrame(data_0.toarray(), columns= full_pipeline.
↪get_feature_names_out(),
                           index = X_train.index
                          )
df_prepared_test= pd.DataFrame(data_test.toarray(), columns= full_pipeline.
↪get_feature_names_out(),
                           index = X_test.index
                          )
```

[11]:
```
df_prepared.reset_index(drop=True, inplace=True)
df_prepared_test.reset_index(drop=True, inplace=True)
df_prepared.head()
```

[11]:

	nor__BonusMalus	cat__Area_A	cat__Area_B	cat__Area_C	cat__Area_D
0	0.000000	0.0	0.0	1.0	0.0
1	0.357143	0.0	0.0	0.0	1.0
2	0.040816	0.0	0.0	0.0	1.0
3	0.102041	0.0	0.0	1.0	0.0
4	0.000000	0.0	0.0	0.0	1.0

	cat__Area_E	cat__Area_F	cat__Region_Alsace	cat__Region_Aquitaine
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0

	cat__Region_Auvergne	cat__Region_Basse-Normandie	cat__Region_Bourgogne
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

	cat__Region_Bretagne	cat__Region_Centre	cat__Region_Champagne-Ardenne
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	1.0	0.0
3	0.0	1.0	0.0
4	0.0	0.0	0.0

	cat__Region_Corse	cat__Region_Franche-Comte	cat__Region_Haute-Normandie
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

	cat__Region_Ile-de-France	cat__Region_Languedoc-Roussillon
0	0.0	1.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

	cat__Region_Limousin	cat__Region_Midi-Pyrenees

```
0                   0.0                         0.0
1                   0.0                         0.0
2                   0.0                         0.0
3                   0.0                         0.0
4                   0.0                         0.0

   cat__Region_Nord-Pas-de-Calais  cat__Region_Pays-de-la-Loire  \
0                             0.0                           0.0
1                             1.0                           0.0
2                             0.0                           0.0
3                             0.0                           0.0
4                             1.0                           0.0

   cat__Region_Picardie  cat__Region_Poitou-Charentes  \
0                   0.0                           0.0
1                   0.0                           0.0
2                   0.0                           0.0
3                   0.0                           0.0
4                   0.0                           0.0

   cat__Region_Provence-Alpes-Cotes-D'Azur  cat__Region_Rhone-Alpes  \
0                                      0.0                      0.0
1                                      0.0                      0.0
2                                      0.0                      0.0
3                                      0.0                      0.0
4                                      0.0                      0.0

   cat__VehGas_Diesel  cat__VehGas_Regular  cat__VehBrand_B1  \
0                 1.0                  0.0               0.0
1                 1.0                  0.0               0.0
2                 1.0                  0.0               1.0
3                 0.0                  1.0               1.0
4                 0.0                  1.0               0.0

   cat__VehBrand_B10  cat__VehBrand_B11  cat__VehBrand_B12  cat__VehBrand_B13  \
0                0.0                0.0                1.0                0.0
1                0.0                1.0                0.0                0.0
2                0.0                0.0                0.0                0.0
3                0.0                0.0                0.0                0.0
4                0.0                0.0                1.0                0.0

   cat__VehBrand_B14  cat__VehBrand_B2  cat__VehBrand_B3  cat__VehBrand_B4  \
0                0.0               0.0               0.0               0.0
1                0.0               0.0               0.0               0.0
2                0.0               0.0               0.0               0.0
3                0.0               0.0               0.0               0.0
4                0.0               0.0               0.0               0.0
```

```
   cat__VehBrand_B5  cat__VehBrand_B6  cat__VehPower_4  cat__VehPower_5  \
0               0.0               0.0              0.0              0.0
1               0.0               0.0              0.0              0.0
2               0.0               0.0              0.0              0.0
3               0.0               0.0              0.0              0.0
4               0.0               0.0              0.0              1.0

   cat__VehPower_6  cat__VehPower_7  cat__VehPower_8  cat__VehPower_9  \
0              0.0              0.0              0.0              1.0
1              1.0              0.0              0.0              0.0
2              1.0              0.0              0.0              0.0
3              0.0              1.0              0.0              0.0
4              0.0              0.0              0.0              0.0

   cat__VehAge_0-5  cat__VehAge_13+  cat__VehAge_6-12  cat__VehAge_nan  \
0              1.0              0.0               0.0              0.0
1              1.0              0.0               0.0              0.0
2              0.0              0.0               1.0              0.0
3              1.0              0.0               0.0              0.0
4              1.0              0.0               0.0              0.0

   cat__DrivAge_18-20  cat__DrivAge_21-30  cat__DrivAge_31-40  \
0                 0.0                 0.0                 0.0
1                 0.0                 1.0                 0.0
2                 0.0                 0.0                 0.0
3                 0.0                 0.0                 1.0
4                 0.0                 0.0                 0.0

   cat__DrivAge_41-50  cat__DrivAge_51-70  cat__DrivAge_71+  cat__DrivAge_nan  \
0                 0.0                 1.0               0.0               0.0
1                 0.0                 0.0               0.0               0.0
2                 0.0                 1.0               0.0               0.0
3                 0.0                 0.0               0.0               0.0
4                 1.0                 0.0               0.0               0.0

   remainder__Density
0            4.672829
1            7.484369
2            6.969791
3            5.973810
4            7.118016
```

[12]:
```
#Eliminamos las variables ruido {'cat__DrivAge_nan', 'cat__VehAge_nan'}
df_prepared_test = df_prepared_test.drop(['cat__DrivAge_nan',␣
 ↪'cat__VehAge_nan'], axis=1)
```

[13]:
```
df_prepared = df_prepared.drop(['cat__DrivAge_nan', 'cat__VehAge_nan'], axis=1)
df_prepared.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 532593 entries, 0 to 532592
Data columns (total 57 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   nor__BonusMalus                         532593 non-null  float64
 1   cat__Area_A                             532593 non-null  float64
 2   cat__Area_B                             532593 non-null  float64
 3   cat__Area_C                             532593 non-null  float64
 4   cat__Area_D                             532593 non-null  float64
 5   cat__Area_E                             532593 non-null  float64
 6   cat__Area_F                             532593 non-null  float64
 7   cat__Region_Alsace                      532593 non-null  float64
 8   cat__Region_Aquitaine                   532593 non-null  float64
 9   cat__Region_Auvergne                    532593 non-null  float64
 10  cat__Region_Basse-Normandie             532593 non-null  float64
 11  cat__Region_Bourgogne                   532593 non-null  float64
 12  cat__Region_Bretagne                    532593 non-null  float64
 13  cat__Region_Centre                      532593 non-null  float64
 14  cat__Region_Champagne-Ardenne           532593 non-null  float64
 15  cat__Region_Corse                       532593 non-null  float64
 16  cat__Region_Franche-Comte               532593 non-null  float64
 17  cat__Region_Haute-Normandie             532593 non-null  float64
 18  cat__Region_Ile-de-France               532593 non-null  float64
 19  cat__Region_Languedoc-Roussillon        532593 non-null  float64
 20  cat__Region_Limousin                    532593 non-null  float64
 21  cat__Region_Midi-Pyrenees               532593 non-null  float64
 22  cat__Region_Nord-Pas-de-Calais          532593 non-null  float64
 23  cat__Region_Pays-de-la-Loire            532593 non-null  float64
 24  cat__Region_Picardie                    532593 non-null  float64
 25  cat__Region_Poitou-Charentes            532593 non-null  float64
 26  cat__Region_Provence-Alpes-Cotes-D'Azur 532593 non-null  float64
 27  cat__Region_Rhone-Alpes                 532593 non-null  float64
 28  cat__VehGas_Diesel                      532593 non-null  float64
 29  cat__VehGas_Regular                     532593 non-null  float64
 30  cat__VehBrand_B1                        532593 non-null  float64
 31  cat__VehBrand_B10                       532593 non-null  float64
 32  cat__VehBrand_B11                       532593 non-null  float64
 33  cat__VehBrand_B12                       532593 non-null  float64
 34  cat__VehBrand_B13                       532593 non-null  float64
 35  cat__VehBrand_B14                       532593 non-null  float64
 36  cat__VehBrand_B2                        532593 non-null  float64
 37  cat__VehBrand_B3                        532593 non-null  float64
 38  cat__VehBrand_B4                        532593 non-null  float64
 39  cat__VehBrand_B5                        532593 non-null  float64
```

```
 40  cat__VehBrand_B6                        532593 non-null  float64
 41  cat__VehPower_4                         532593 non-null  float64
 42  cat__VehPower_5                         532593 non-null  float64
 43  cat__VehPower_6                         532593 non-null  float64
 44  cat__VehPower_7                         532593 non-null  float64
 45  cat__VehPower_8                         532593 non-null  float64
 46  cat__VehPower_9                         532593 non-null  float64
 47  cat__VehAge_0-5                         532593 non-null  float64
 48  cat__VehAge_13+                         532593 non-null  float64
 49  cat__VehAge_6-12                        532593 non-null  float64
 50  cat__DrivAge_18-20                      532593 non-null  float64
 51  cat__DrivAge_21-30                      532593 non-null  float64
 52  cat__DrivAge_31-40                      532593 non-null  float64
 53  cat__DrivAge_41-50                      532593 non-null  float64
 54  cat__DrivAge_51-70                      532593 non-null  float64
 55  cat__DrivAge_71+                        532593 non-null  float64
 56  remainder__Density                      532593 non-null  float64
dtypes: float64(57)
memory usage: 231.6 MB
```

1.4 Modelos Random Forest

[14]:
```
# Modelo 1
tic = time.time()
rf = RandomForestClassifier( random_state=33,
        n_jobs=-1)
#Entrenamiento

rf.fit(df_prepared, y_train)


tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
```

```
-----------------------------------------------
Tiempo de Calculo:  26.5905s
```

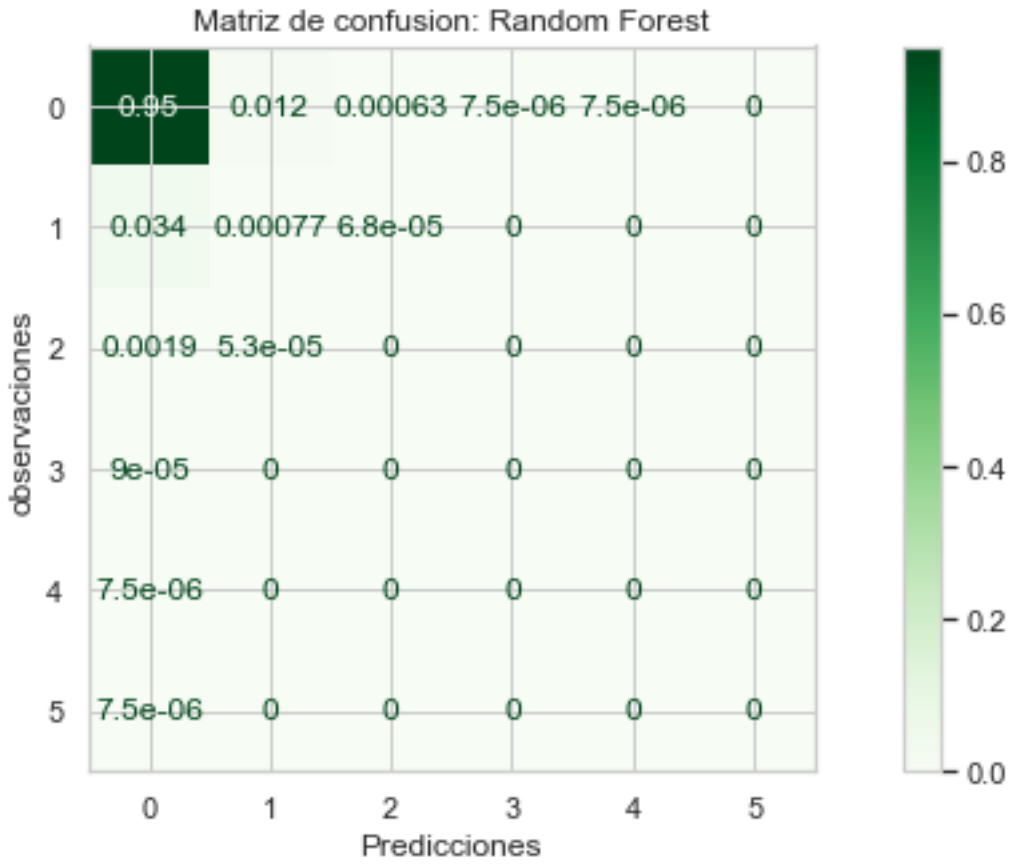
[15]:
```
rf_pred = rf.predict(df_prepared_test)
```

[16]:
```
rf_mc = confusion_matrix(y_true=y_test,
                         y_pred= rf_pred,

                        )
print(rf_mc)
```

```
[[126611   1535     84      1      1      0]
 [  4538    102      9      0      0      0]
 [   247      7      0      0      0      0]
```

```
[    12      0      0      0      0      0]
[     1      0      0      0      0      0]
[     1      0      0      0      0      0]]
```

[17]:
```
plot_confusion_matrix(rf, df_prepared_test, y_test, normalize='all',␣
↪cmap='Greens' )
plt.title('Matriz de confusion: Random Forest')
plt.xlabel('Predicciones')
plt.ylabel('observaciones')
plt.show()
```



[18]:
```
y_test.value_counts()
```

[18]:
```
0    128232
1      4649
2       254
3        12
5         1
4         1
```

```
Name: ClaimNb, dtype: int64
```

[19]:
```
rf_accu = accuracy_score(y_true = y_test,
                         y_pred = rf_pred,
                         normalize = True
                         )
print(rf_accu)
```

```
0.951663174338523
```

[20]:
```
print(classification_report(
            y_true= y_test,
            y_pred= rf_pred
        ))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98    128232
           1       0.06      0.02      0.03      4649
           2       0.00      0.00      0.00       254
           3       0.00      0.00      0.00        12
           4       0.00      0.00      0.00         1
           5       0.00      0.00      0.00         1

    accuracy                           0.95    133149
   macro avg       0.17      0.17      0.17    133149
weighted avg       0.93      0.95      0.94    133149
```

[322]:
```
rf_pred.mean()
```

[322]:
```
0.013796573763227663
```

1.4.1 Alternativas para Balanceo de los datos

1. class_weight='balanced_subsample'
2. class_weight='balanced'

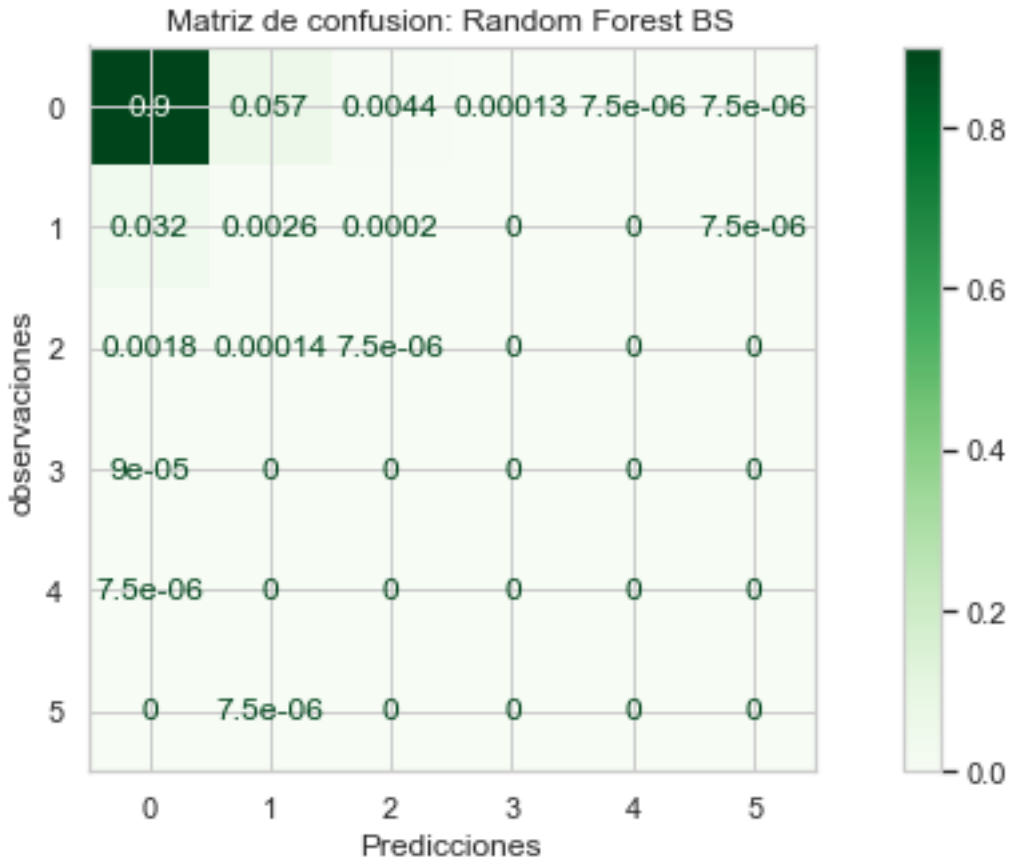
[25]:
```
# Modelo 2
tic = time.time()
rf_bs = RandomForestClassifier(class_weight='balanced_subsample',␣
 ↪random_state=33,
        n_jobs=-1)
#Entrenamiento

rf_bs.fit(df_prepared, y_train)
tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
```

```
-------------------------------------------
Tiempo de Calculo:  26.8804s
```

[22]:
```
rf_predbs = rf_bs.predict(df_prepared_test)
rf_mc0 = confusion_matrix(y_true=y_test,
                          y_pred= rf_predbs
                          )
print(rf_mc0)
print('_____________________________________________')
rf_accu = accuracy_score(y_true = y_test,
                          y_pred = rf_predbs,
                          normalize = True
                          )
print(rf_accu)
```

```
[[119987   7646    580     17      1      1]
 [  4273    349     26      0      0      1]
 [   235     18      1      0      0      0]
 [    12      0      0      0      0      0]
 [     1      0      0      0      0      0]
 [     0      1      0      0      0      0]]
-------------------------------------------
0.9037769716633245
```

[23]:
```
plot_confusion_matrix(rf_bs, df_prepared_test, y_test, normalize='all',␣
 ↪cmap='Greens' )
plt.title('Matriz de confusion: Random Forest BS')
plt.xlabel('Predicciones')
plt.ylabel('observaciones')
plt.show()
```



[24]:
```
rf_predbs.mean()
```

[24]: 0.06979399019143967

[26]:
```
# modelos 3
tic = time.time()
rf_b = RandomForestClassifier(class_weight='balanced', random_state=33,
        n_jobs=-1)
#Entrenamiento

rf_b.fit(df_prepared, y_train)
tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
```

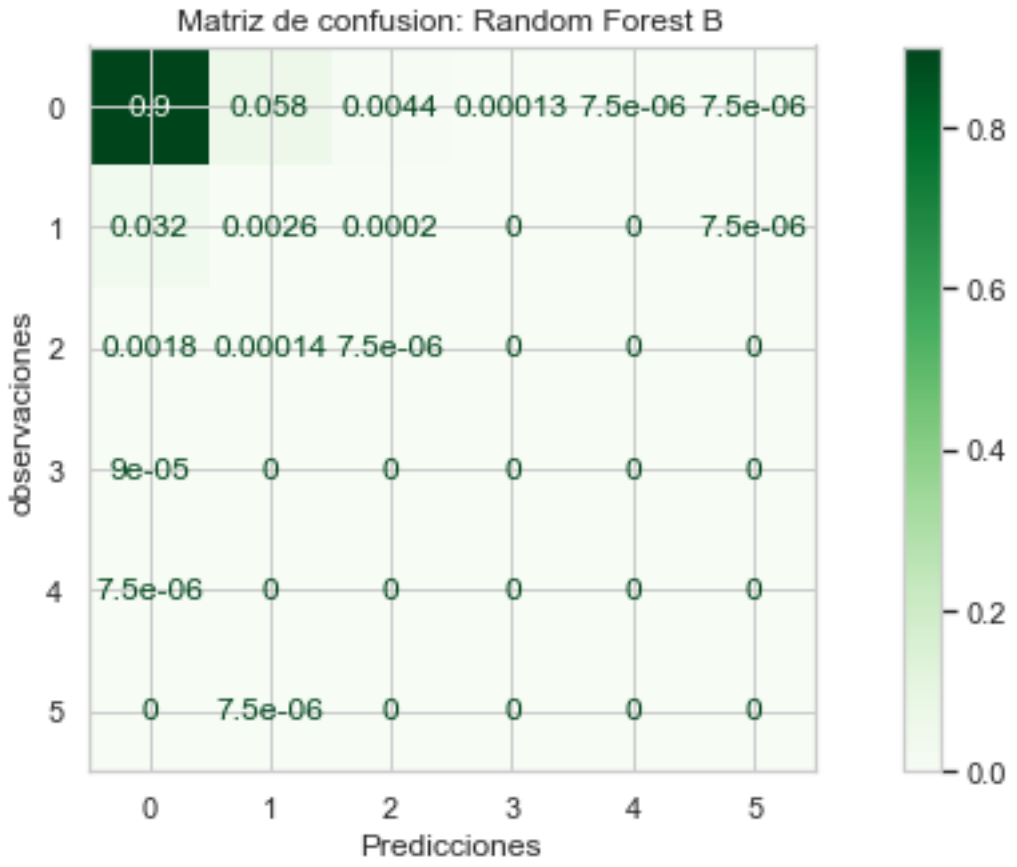
```
-----------------------------------------------
Tiempo de Calculo:  24.5242s
```

[327]:
```
rf_predb = rf_b.predict(df_prepared_test)
rf_mcb = confusion_matrix(y_true=y_test,
```

```
                        y_pred= rf_predb
                        )
print(rf_mcb)
print('_____________________________________________')
rf_accu = accuracy_score(y_true = y_test,
                         y_pred = rf_predb,
                         normalize = True
                         )
print(rf_accu)
```

```
[[119972   7660    581     17      1      1]
 [  4273    349     26      0      0      1]
 [   234     19      1      0      0      0]
 [    12      0      0      0      0      0]
 [     1      0      0      0      0      0]
 [     0      1      0      0      0      0]]

-----------------------------------------
0.9036643159167549
```

[330]:
```
plot_confusion_matrix(rf_b, df_prepared_test, y_test, normalize='all',␣
 ↪cmap='Greens' )
plt.title('Matriz de confusion: Random Forest B')
plt.xlabel('Predicciones')
plt.ylabel('observaciones')
plt.show()
```



[332]:
```
rf_predb.mean()
```

[332]: 0.06992166670421858

[347]:
```
print("Media de las predicciones: {:.5f}"  .format(rf_predb.mean()))
print("Media de las observaciones: {:.5f}"  .format(y_test.mean()))
```

```
Media de las predicciones: 0.06992
Media de las observaciones: 0.03907
```

1.5 Hiperparametros

¡Ejecutar la siguiente linea de codigo puede tardar muchas horas!

[ ]:
```
# Creamos los valores a evaluar


param_grid = ParameterGrid( {'n_estimators'   : [50, 100, 500, 1000],
              'max_features'   : ['auto', 'sqrt', 'log2'],
              'max_depth'      : [None, 5, 10, 20],
```

```
              'criterion'     : ['gini', 'entropy']
             })
#realizamos el entrenamiento de cada modelo por cada valor de hiperparametro
resultados = {'params': [], 'oob_accuracy': []}

for params in param_grid:

    rf_grid = RandomForestClassifier(
                oob_score    = True,
                n_jobs       = -1,
                random_state = 33,
                ** params)

    rf_grid.fit(df_prepared, y_train)

    resultados['params'].append(params)
    resultados['oob_accuracy'].append(rf_grid.oob_score_)
    print(f"rf_grid: {params} \u2713")

# Salidas
# ------------------------------------------------------------------------------
resultados = pd.DataFrame(resultados)
resultados = pd.concat([resultados, resultados['params'].apply(pd.Series)],␣
 ↪axis=1)
resultados = resultados.sort_values('oob_accuracy', ascending=False)
resultados = resultados.drop(columns = 'params')
resultados.head(4)
```

Se estima el *GridSearchCV* modificando solo el parametro numero de estimadores, puede tardar 73.530 minutos

[348]:
```
param_grid = {'n_estimators'  : [50, 100, 500, 1000]}

grid = GridSearchCV(
        estimator  = RandomForestClassifier(random_state = 33),
        param_grid = param_grid,
        scoring    = 'accuracy',
        n_jobs     = multiprocessing.cpu_count() - 1,
        cv         = RepeatedKFold(n_splits=3, n_repeats=1, random_state=33),
        refit      = True,
        verbose    = 0,
        return_train_score = True
       )
```

[349]:
```
tic = time.time()

grid.fit(X = df_prepared, y = y_train)
```

```
tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format((tac-tic)/60 ))
print('Terminado')
```

```
-----------------------------------------------
Tiempo de Calculo:  4411.8596s
Terminado
```

[350]:
```
#Resultados
resultados = pd.DataFrame(grid.cv_results_)
resultados.filter(regex = '(param*|mean_t|std_t)') \
    .drop(columns = 'params') \
    .sort_values('mean_test_score', ascending = False) \
    .head(4)
```

[350]:

	param_n_estimators	mean_test_score	std_test_score	mean_train_score	std_train_score
3	1000	0.952080	0.000515	0.978935	0.000125
2	500	0.952044	0.000533	0.978935	0.000125
1	100	0.951573	0.000437	0.978905	0.000126
0	50	0.951115	0.000461	0.978613	0.000107

[351]:
```
# Mejores hiperparámetros por validación cruzada
# ==============================================================================
print("----------------------------------------")
print("Mejores hiperparámetros encontrados (cv)")
print("----------------------------------------")
print(grid.best_params_, ":", grid.best_score_, grid.scoring)
```

```
----------------------------------------
Mejores hiperparámetros encontrados (cv)
----------------------------------------
{'n_estimators': 1000} : 0.9520797306761448 accuracy
```

[352]:
```
modelo_final_rf = grid.best_estimator_
```

[353]:
```
mean_rf = modelo_final_rf.predict(df_prepared_test)
```

[354]:
```
mean_rf.mean()
```

[354]:
```
0.012880307024461317
```

1.6 Modelo Final

[355]:
```
Claim_preds = modelo_final_rf.predict(df_prepared_test)
Claim_preds.mean()
```

[355]:
```
0.012880307024461317
```

[356]:
```
dfc = pd.DataFrame(Claim_preds)
```

[357]:
```
print(f'Clasificación del modelo \n {dfc.value_counts()} Clasificacion original:
 ↪\n {y_test.value_counts()}')
```

```
Clasificación del modelo
 0    131524
1      1538
2        85
3         1
4         1
dtype: int64 Clasificacion original:
 0    128232
1      4649
2       254
3        12
5         1
4         1
Name: ClaimNb, dtype: int64
```

[358]:
```
# METRICAS
confusion = confusion_matrix(
                    y_true    = y_test,
                    y_pred    = Claim_preds
                )

accuracy = accuracy_score(
             y_true    = y_test,
             y_pred    = Claim_preds,
             normalize = True
            )

print("Matriz de confusión")
print("--------------------")
print(confusion)
print("")
print(f"la precision de test es: {100 * accuracy} %")
```

```
Matriz de confusión
--------------------
[[126717   1437     76      1      1      0]
 [  4547     93      9      0      0      0]
```

```
[   246      8      0      0      0      0]
[    12      0      0      0      0      0]
[     1      0      0      0      0      0]
[     1      0      0      0      0      0]]
```

```
la precision de test es: 95.23916814996734 %
```

[359]:
```
# METRICAS
print(
    classification_report(
        y_true = y_test,
        y_pred = Claim_preds
    )
)
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98    128232
           1       0.06      0.02      0.03      4649
           2       0.00      0.00      0.00       254
           3       0.00      0.00      0.00        12
           4       0.00      0.00      0.00         1
           5       0.00      0.00      0.00         1

    accuracy                           0.95    133149
   macro avg       0.17      0.17      0.17    133149
weighted avg       0.93      0.95      0.94    133149
```

1.6.1 Importancia de los predictores

[360]:
```
importancia_predictores = pd.DataFrame(
                            {'predictor': df_prepared.columns,
                             'importancia': modelo_final_rf.feature_importances_}
                            )
print("Importancia de los predictores en el modelo")
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
importancia_predictores.sort_values('importancia', ascending=False)
```

```
Importancia de los predictores en el modelo
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

[360]:

	predictor	importancia
56	remainder__Density	0.348627
0	nor__BonusMalus	0.222792
47	cat__VehAge_0-5	0.020896
44	cat__VehPower_7	0.017705
42	cat__VehPower_5	0.016269

```
..                              ...          ...
7               cat__Region_Alsace     0.001276
6                     cat__Area_F      0.001203
15              cat__Region_Corse      0.001089
14  cat__Region_Champagne-Ardenne      0.000968
16      cat__Region_Franche-Comte      0.000625

[57 rows x 2 columns]
```

2 XGBoost

2.1 XGB Clasificación

[361]:
```
data_1 = xgb.DMatrix(data=data_0, label=y_train)
```

[362]:
```
model_1 = XGBClassifier(random_state = 33)
```

[363]:
```
model_1.fit(df_prepared, y_train)
```

[363]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, objective='multi:softprob',
              predictor='auto', random_state=33, reg_alpha=0, ...)
```

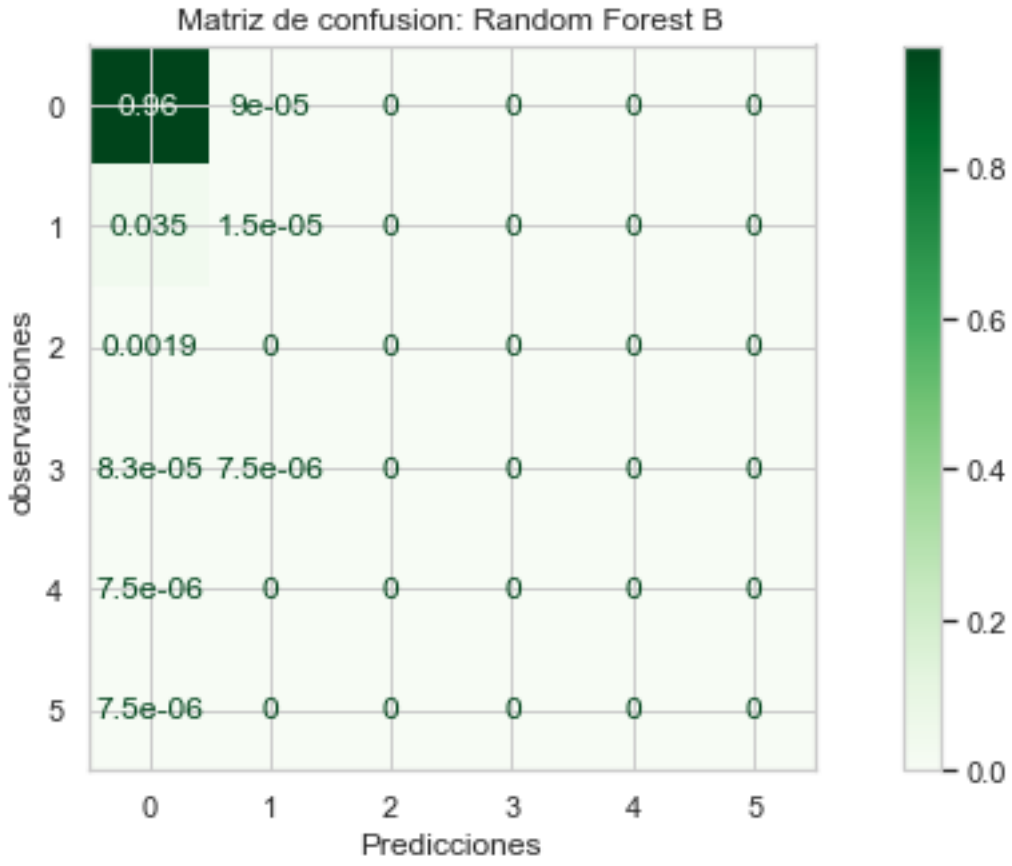
[364]:
```
preds_xgb = model_1.predict(df_prepared_test)
```

[365]:
```
confusion = confusion_matrix(
                    y_true    = y_test,
                    y_pred    = preds_xgb
                )

accuracy = accuracy_score(
            y_true    = y_test,
            y_pred    = preds_xgb,
            normalize = True
           )

print("Matriz de confusión")
print("--------------------")
print(confusion)
print("")
```

```
print(f"la precision de test es: {100 * accuracy} %")
```

```
Matriz de confusión
--------------------
[[128220     12      0      0      0      0]
 [  4647      2      0      0      0      0]
 [   254      0      0      0      0      0]
 [    11      1      0      0      0      0]
 [     1      0      0      0      0      0]
 [     1      0      0      0      0      0]]

la precision de test es: 96.2996342443428 %
```

[367]:
```
plot_confusion_matrix(model_1, df_prepared_test, y_test, normalize='all',␣
↪cmap='Greens' )
plt.title('Matriz de confusion: Random Forest B')
plt.xlabel('Predicciones')
plt.ylabel('observaciones')
plt.show()
```



[368]:
```
preds_xgb.mean()
```

[368]:
```
0.00011265574656963252
```

[ ]:
```

```

2.2 XGB Modelo de Regression

[369]:
```
#fijando el modelo, cambiando hiperparametros

model_xgb_poisson = XGBRegressor(random_state=33,
                                  objective='count:poisson',
                                 learning_rate= 0.01,
                                 max_depth = 5,
                                 alpha = 10
                                 )
```

[372]:
```
model_xgb_poisson.fit(df_prepared, y_train)
```

[372]:
```
XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.01, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0.699999988, max_depth=5, max_leaves=0,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='count:poisson', predictor='auto', random_state=33, ...)
```

Creamos una funcion que permita calcular las metricas

[373]:
```
# Creamos una funcion para calcular las metricas
def metricas(modelo, X_test, y_test):
    """metricas del modelo """
    y_pred = modelo.predict(X_test)

    print('Media de las predicciones:  %f' % y_pred.mean())
    print(
        "MSE: %.5f"
        % mean_squared_error(
            y_test, y_pred)
    )
    print(
        "MAE: %.5f"
        % mean_absolute_error(
            y_test, y_pred)
    )
```

```
    # Ignorar las Predicciones no positivas, para el calculo de la devianza
    mask = y_pred > 0
    if (~mask).any():
        n_masked, n_samples = (~mask).sum(), mask.shape[0]
        print(
            "Atensión: CAmpos de estimador invalidos, Predicciones no positivas "
            f" por {n_masked} ejemplos fuera de la muestra {n_samples}. estas␣
↪predicciones "
            "son ignoradas para cimputar la devianza de poisson."
        )

    print(
        "Desviance Media de Poisson: %.3f"
        % mean_poisson_deviance(
            y_test[mask],
            y_pred[mask],

        )
    )
```

[374]:
```
print("Evolucion del numero medio de reclamos:")
metricas(model_xgb_poisson, df_prepared_test, y_test)
```

```
Evolucion del numero medio de reclamos:
Media de las predicciones:  0.319746
MSE: 0.12065
MAE: 0.33488
Desviance Media de Poisson: 0.656
```

[376]:
```
preds_poisson = model_xgb_poisson.predict(df_prepared_test)
rmse = np.sqrt(mean_squared_error(y_test, preds_poisson))
print("RMSE: %f" % (rmse))
```

```
RMSE: 0.347346
```

[377]:
```
X_total = pd.concat([df_prepared, df_prepared_test], axis=0)
y_total = pd.concat([y_train, y_test], axis=0)
data_dmatrix = xgb.DMatrix(data=df_prepared, label= y_train)
data_dmatrix_ = xgb.DMatrix(data=X_total, label= y_total)
print(X_total.shape)
print(y_total.shape)
```

```
(665742, 57)
(665742,)
```

[378]:
```
#Validacion Cruzada
params = {"objective":"count:poisson",
          'colsample_bytree': 0.3,
```

```
          'learning_rate': 0.3,
          'max_depth': 5,
          'alpha': 10}
```

[379]:
```
#Analizamos el modelo
tic = time.time()#generando huella de tiempo
cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",␣
 ↪as_pandas=True, seed=33)
tac = time.time()#generando huella de tiempo
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format((tac-tic )/60))
print('Terminado')
```

```
-----------------------------------------------
Tiempo de Calculo:  0.1557s
Terminado
```

[380]:
```
cv_results
```

[380]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	0.446628	0.000192	0.446630	0.000269
1	0.398307	0.000319	0.398311	0.000332
2	0.358292	0.000365	0.358297	0.000486
3	0.325439	0.000468	0.325452	0.000593
4	0.298814	0.000543	0.298830	0.000729
..	...	...	...	...
45	0.203398	0.001073	0.203570	0.001853
46	0.203393	0.001074	0.203567	0.001851
47	0.203340	0.001046	0.203521	0.001879
48	0.203335	0.001046	0.203520	0.001881
49	0.203295	0.001001	0.203483	0.001929

[50 rows x 4 columns]

[335]:
```
print((cv_results["test-rmse-mean"]).tail(1)) #Modificando la tasa de␣
 ↪aprendisaje (0.01)
```

```
49    0.413812
Name: test-rmse-mean, dtype: float64
```

[340]:
```
print((cv_results["test-rmse-mean"]).tail(1)) #Modificando la tasa de␣
 ↪aprendisaje (0.1)
```

```
49    0.207392
Name: test-rmse-mean, dtype: float64
```

[471]:
```
print((cv_results["test-rmse-mean"]).tail(1)) #Modificando la tasa de␣
 ↪aprendisaje (0.03)
```

```
49    0.299436
Name: test-rmse-mean, dtype: float64
```

[381]:
```
print((cv_results["test-rmse-mean"]).tail(1)) #Modificando la tasa de␣
↪aprendisaje (0.3)
```

```
49    0.203483
Name: test-rmse-mean, dtype: float64
```

[382]:
```
tic = time.time()#generando huella de tiempo
model_xgb_final = XGBRegressor(random_state=33,
                               objective='count:poisson',
                               colsample_bytree= 0.3,
                               learning_rate= 0.3,
                               max_depth = 5,
                               alpha = 10
                                 )

tac = time.time()#generando huella de tiempo
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
print('Terminado')
```

```
-----------------------------------------------
Tiempo de Calculo:  0.0000s
Terminado
```

[383]:
```
tic = time.time()#generando huella de tiempo

model_xgb_final.fit(df_prepared, y_train)

tac = time.time()#generando huella de tiempo
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
print('Terminado')
```

```
-----------------------------------------------
Tiempo de Calculo:  5.8704s
Terminado
```

[384]:
```
preds_poisson_f = model_xgb_final.predict(df_prepared_test)

print('Media de las predicciones:  %f' % preds_poisson_f.mean())

print("Evolucion del numero medio constante:")
metricas(model_xgb_final, df_prepared_test, y_test)
```

```
Media de las predicciones:  0.039086
Evolucion del numero medio constante:
Media de las predicciones:  0.039086
```

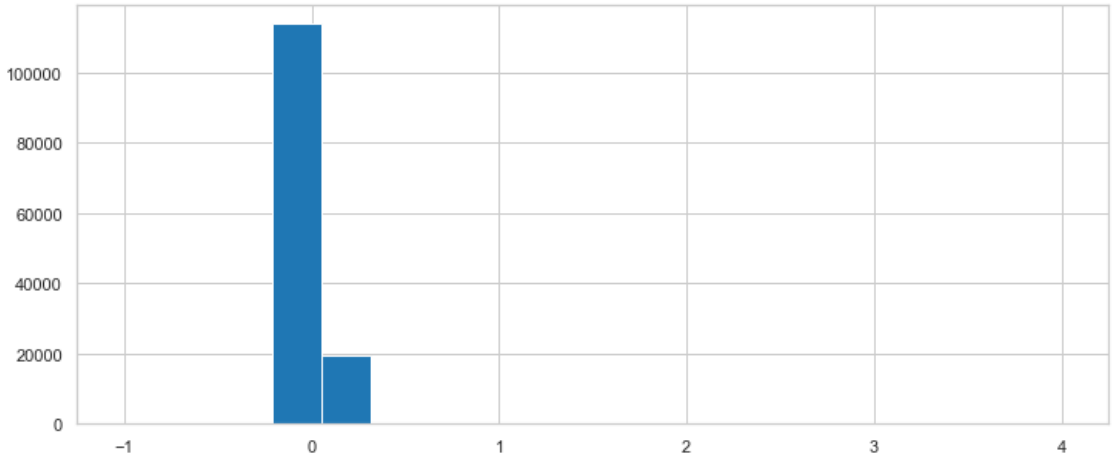
```
MSE: 0.04140
MAE: 0.07401
Desviance Media de Poisson: 0.246
```

[385]:
```
rmse_ = np.sqrt(mean_squared_error(y_test, preds_poisson_f))
print("RMSE: %f" % (rmse_))
```

```
RMSE: 0.203468
```

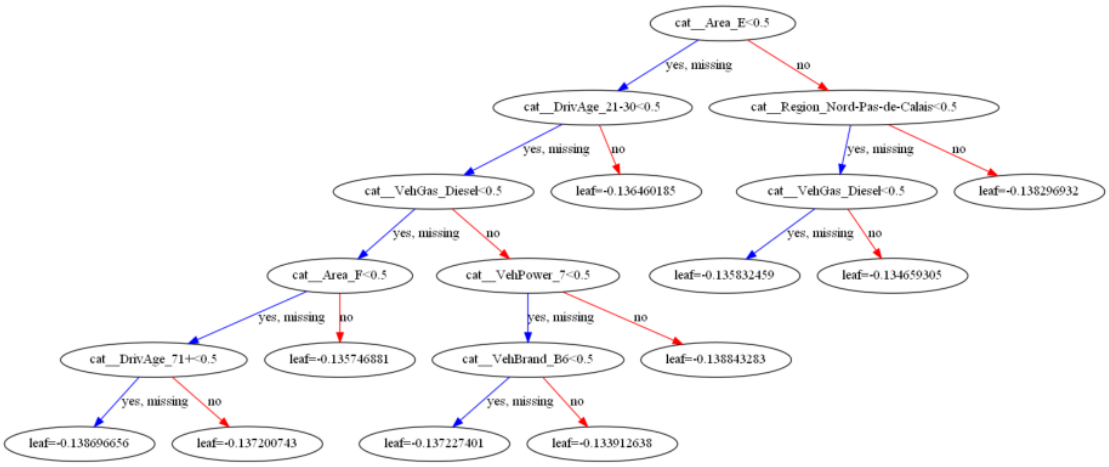
[386]:
```
pd.Series(preds_poisson_f).hist(
    bins=np.linspace(-1, 4, 20))
```

[386]: <AxesSubplot:>

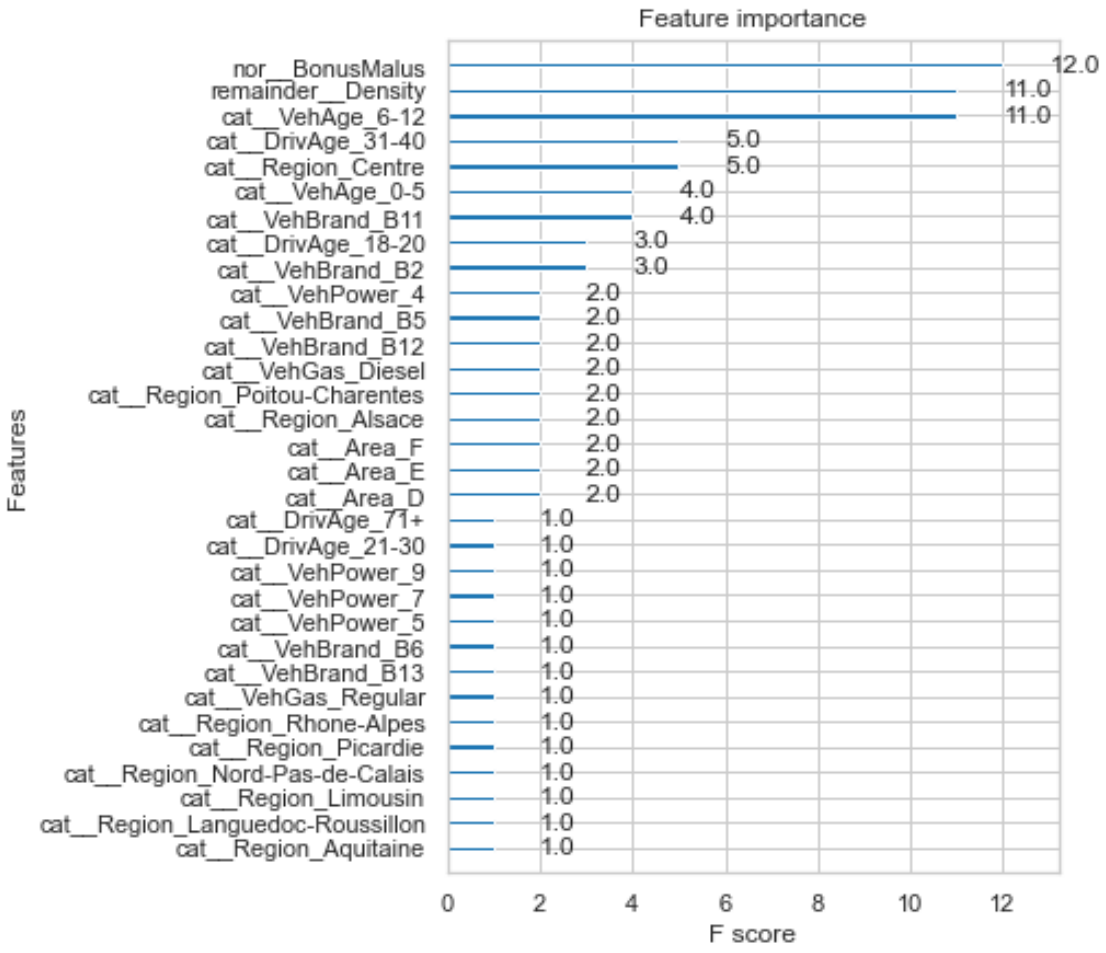


[387]:
```
xgb_reg = xgb.train(params=params, dtrain=data_dmatrix, num_boost_round=10)
```

[390]:
```
xgb.plot_tree(xgb_reg,num_trees=0)
plt.rcParams['figure.figsize'] = [5, 7]
plt.show()
```



[391]:
```
# Importancia de las variables en el train
xgb.plot_importance(xgb_reg[:5])
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```



2.2.1 Analisis de Caracteristicas sobre la Población Total

[392]:
```
preds_poisson_f.mean()
```

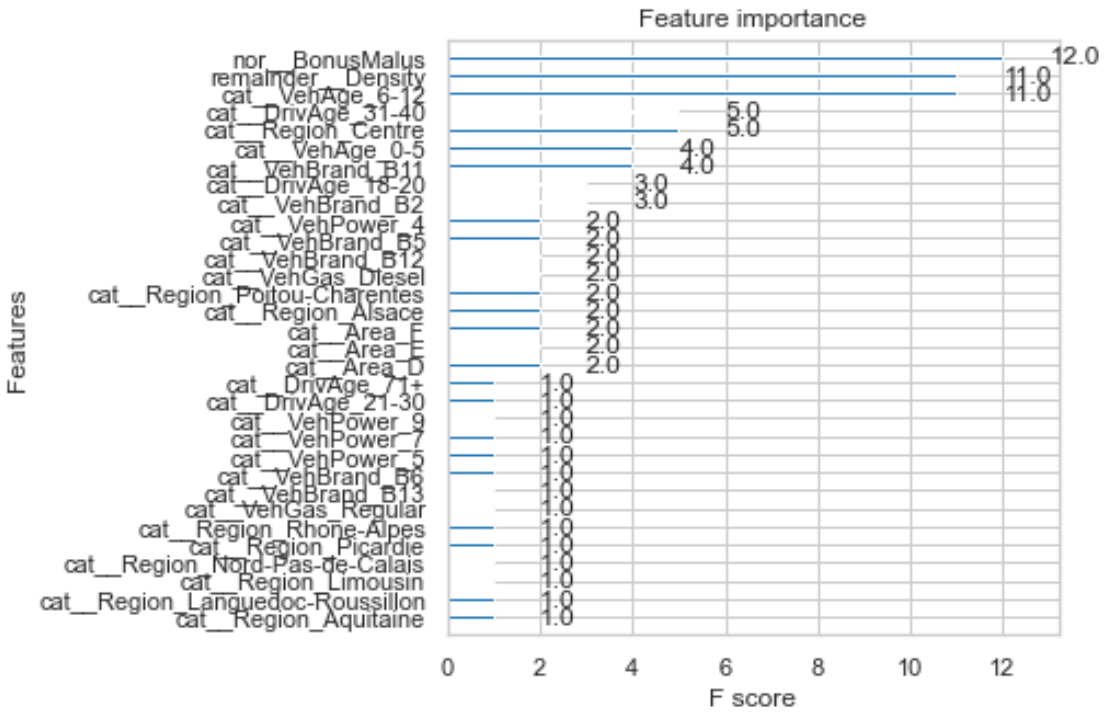
[392]:
```
0.039086424
```

[393]:
```
y_test.mean()# media de la data de prueba
```

[393]:
```
0.039069012910348554
```

[ ]:
```

```

[394]:
```
#Importancia total
xgb.plot_importance(xgb_reg[:5])
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```



[395]:
```
cv_results = xgb.cv(dtrain=data_dmatrix_, params=params, nfold=3,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",␣
↪as_pandas=True, seed=33)
```

[396]:
```
cv_results
```

[396]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	0.446655	0.000266	0.446656	0.000424
1	0.398331	0.000341	0.398333	0.000565
2	0.358309	0.000492	0.358313	0.000643
3	0.325452	0.000559	0.325467	0.000807
4	0.298831	0.000651	0.298847	0.000947
..	...	...	...	...
45	0.203494	0.001012	0.203640	0.002295
46	0.203489	0.001010	0.203638	0.002298
47	0.203436	0.001039	0.203590	0.002265
48	0.203430	0.001042	0.203587	0.002263
49	0.203398	0.001076	0.203560	0.002229

[50 rows x 4 columns]

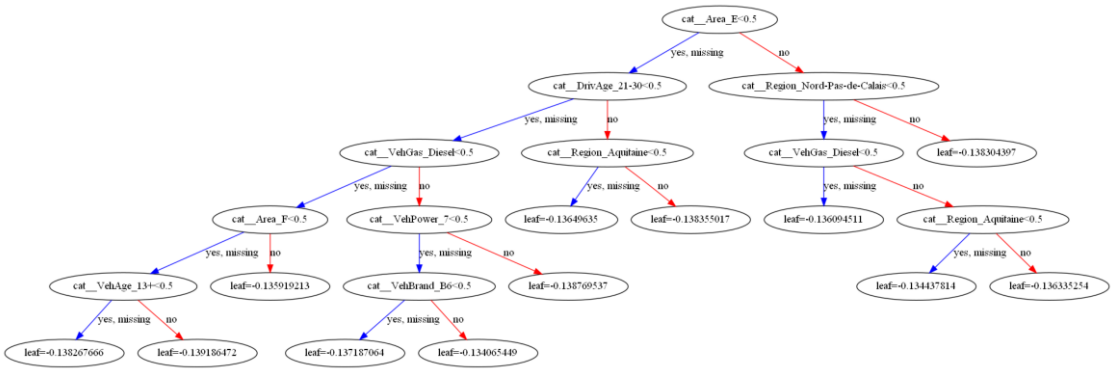
[397]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
49    0.20356
```

```
Name: test-rmse-mean, dtype: float64
```

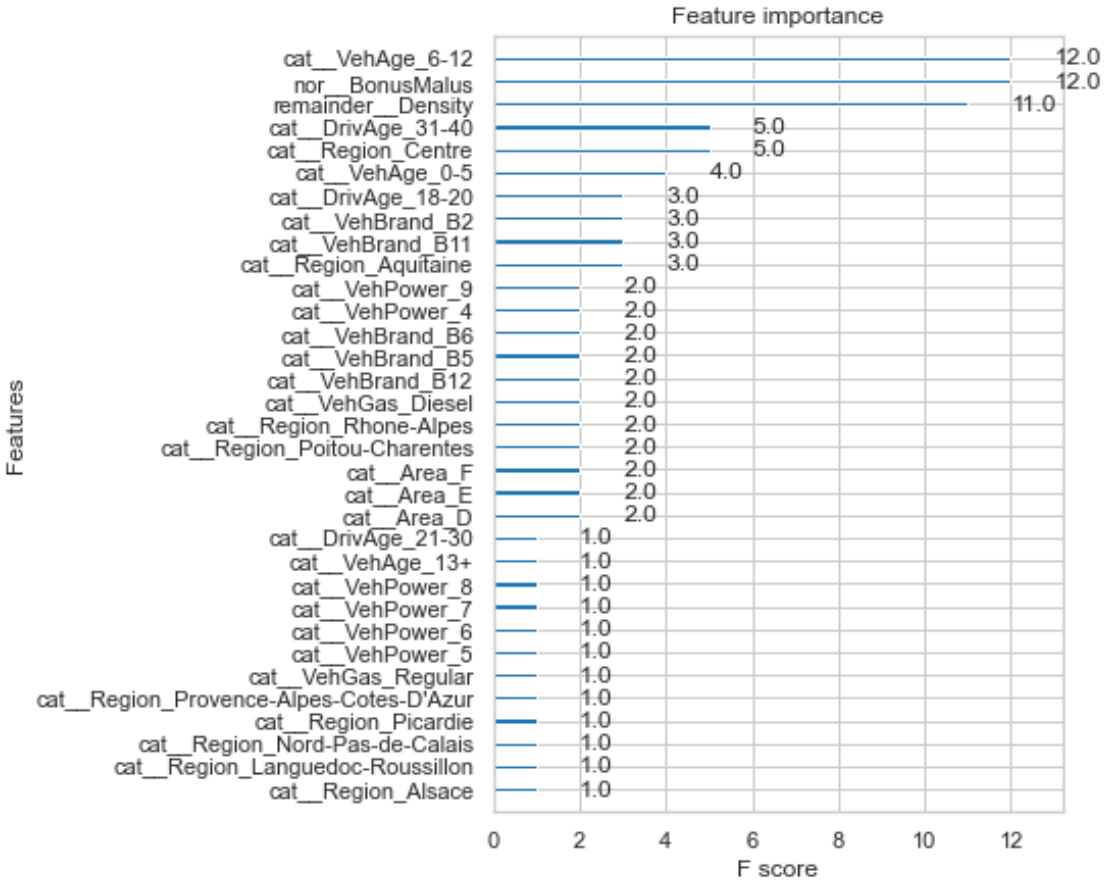
[398]:
```
xgb_reg_t = xgb.train(params=params, dtrain=data_dmatrix_, num_boost_round=10)
```

[400]:
```
xgb.plot_tree(xgb_reg_t,num_trees=0)
plt.rcParams['figure.figsize'] = [5, 7]
plt.show()
```

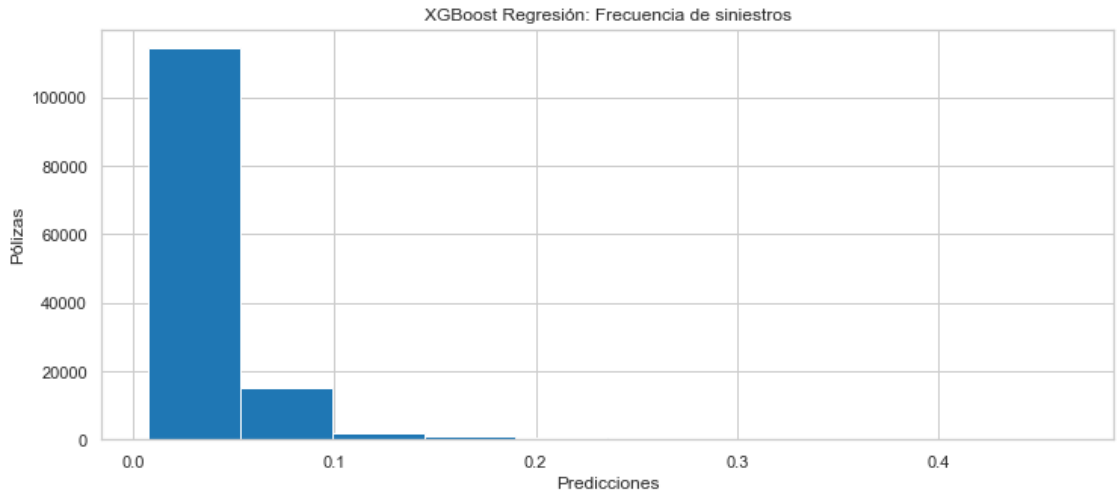


[ ]:
```
```

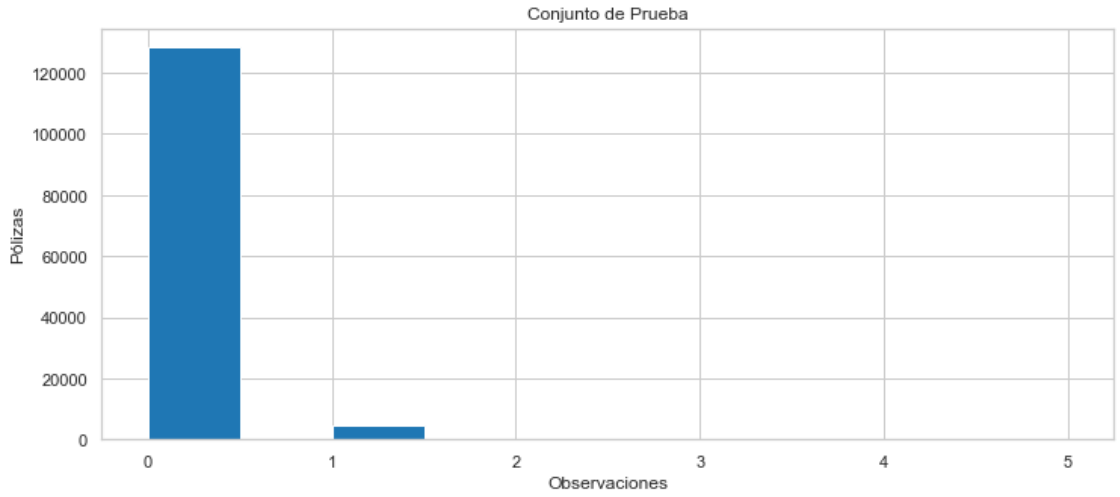
[417]:
```
# Importancia de las variables en el train
xgb.plot_importance(xgb_reg_t[:5])
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```



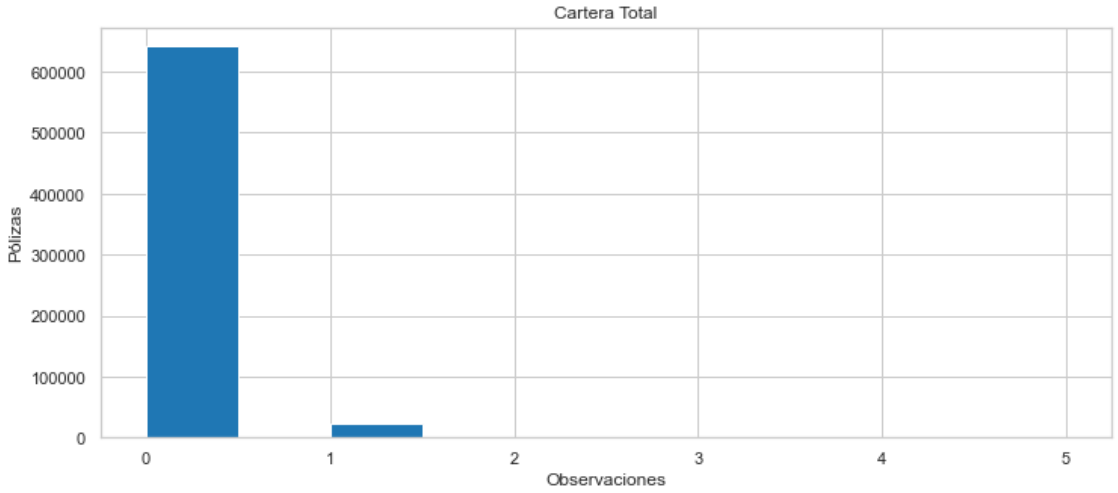
[414]:
```
pd.Series(preds_poisson_f).hist()
plt.title('XGBoost Regresión: Frecuencia de siniestros')
plt.xlabel('Predicciones')
plt.ylabel('Pólizas')
plt.show()
```



[406]:
```
y_test.hist()
plt.title('Conjunto de Prueba')
plt.xlabel('Observaciones')
plt.ylabel('Pólizas')
plt.show()
```



[421]:
```
df.ClaimNb.hist()
plt.title('Cartera Total')
plt.xlabel('Observaciones')
plt.ylabel('Pólizas')
plt.show()
```


Cartera Total
Pólizas
600000
500000
400000
300000
200000
100000
0
0
1
2
3
4
5
Observaciones

MLP_tfm

June 7, 2022

[2]:
```
%reset -f
%clear
```

[3]:
```
#importacion de librerias y modulos
import pandas as pd
import numpy as np
import matplotlib as mpl
import time
from sklearn.model_selection import StratifiedShuffleSplit
import tensorflow as tf
from tensorflow import feature_column
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.pipeline import make_pipeline
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_poisson_deviance
```

[4]:
```
import warnings
warnings.filterwarnings('ignore')

# configuracion de la visualizacion
import matplotlib.pylab as plt
%matplotlib inline
plt.rc('font', size=12)
plt.rc('figure', figsize = (12, 5))

import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook", font_scale=1, rc={"lines.linewidth": 2,'font.family':
 ↪ [u'times']})
```

```
# configuracion de la data
pd.set_option('display.max_rows', 25)
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_colwidth', 50)
```

[5]:
```
# importacion de datos

df = pd.read_table("C:/Users/JHONNY/OneDrive/TFM_CAF/df_Mof.txt", sep= ",")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 678007 entries, 0 to 678006
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Unnamed: 0    678007 non-null  int64
 1   IDpol         678007 non-null  float64
 2   Exposure      678007 non-null  float64
 3   VehPower      678007 non-null  int64
 4   VehAge        678007 non-null  int64
 5   DrivAge       678007 non-null  int64
 6   BonusMalus    678007 non-null  int64
 7   VehBrand      678007 non-null  object
 8   VehGas        678007 non-null  object
 9   Area          678007 non-null  object
 10  Density       678007 non-null  int64
 11  Region        678007 non-null  object
 12   ClaimTotal   678007 non-null  float64
 13  ClaimNb       678007 non-null  int64
dtypes: float64(3), int64(7), object(4)
memory usage: 72.4+ MB
```

[6]:
```
df = df.loc[df.VehAge<20]
df = df.loc[df.DrivAge<=90]
df = df.loc[df.BonusMalus<=150]

df.VehAge = np.where(df.VehAge>13, 14, df.VehAge)
df.DrivAge = np.where(df.DrivAge>70, 71, df.DrivAge)
df.VehPower = np.where(df.VehPower>9, 9, df.VehPower)
df.Density = np.log(df.Density)


df['VehAge'] = pd.cut(df.VehAge,
                              bins= [0., 5, 12,  np.inf],
                              labels=['0-5', '6-12', '13+'])

df['DrivAge'] = pd.cut(df.DrivAge,
```

```
                                bins= [18., 20, 30, 40, 50, 70, np.inf],
                                labels=['18-20', '21-30', '31-40', '41-50','51-70',␣
 ↪'71+'])
df.reset_index(drop=True, inplace=True)
```

[7]:
```
# Eliminamos las variables que no necesitamos
df = df.drop(columns=['Unnamed: 0','IDpol' ,' ClaimTotal ', 'Exposure'])
```

[8]:
```
#Set de entrenamiento(train) y prueba (test)
train, test = train_test_split(df, test_size=0.2, random_state=33)
train, val = train_test_split(train, test_size=0.2, random_state=33)
print((train.shape), 'train examples')
print((val.shape), 'validation examples')
print((test.shape), 'test examples')
```

```
(426074, 10) train examples
(106519, 10) validation examples
(133149, 10) test examples
```

[9]:
```
y_train = train.ClaimNb.copy()
y_test  = test.ClaimNb.copy()
y_val   = val.ClaimNb.copy()
print(y_train.shape, y_test.shape, y_val.shape )
```

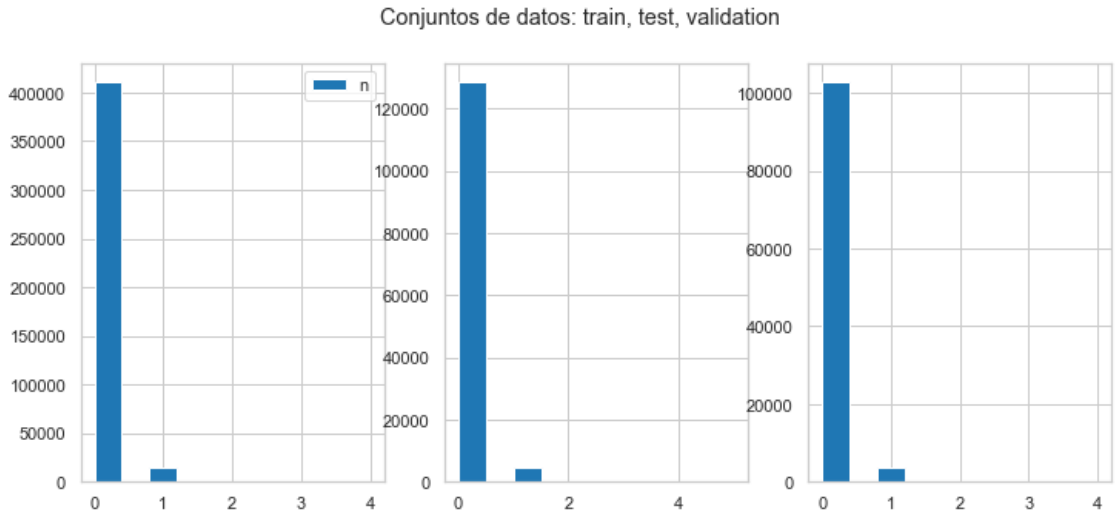
```
(426074,) (133149,) (106519,)
```

[10]:
```
y_train = pd.to_numeric(y_train, downcast='integer')
y_test  = pd.to_numeric(y_test, downcast='integer')
y_val   = pd.to_numeric(y_val, downcast='integer')
y_val.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 106519 entries, 647556 to 173203
Series name: ClaimNb
Non-Null Count   Dtype
--------------   -----
106519 non-null  int8
dtypes: int8(1)
memory usage: 936.2 KB
```

[11]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
fig.suptitle('Conjuntos de datos: train, test, validation')
ax1.hist(y_train)
ax2.hist(y_test)
ax3.hist(y_val)
ax1.legend('numero de polizas')
```

[11]:
```
<matplotlib.legend.Legend at 0x17e94a69540>
```



[12]:
```
y_train.value_counts(), y_val.value_counts(), y_test.value_counts()
```

[12]:
```
(0    410166
 1     15054
 2       807
 3        43
 4         4
 Name: ClaimNb, dtype: int64,
 0    102661
 1      3658
 2       192
 3         7
 4         1
 Name: ClaimNb, dtype: int64,
 0    128332
 1      4534
 2       270
 3        11
 5         2
 Name: ClaimNb, dtype: int64)
```

[13]:
```
train   = train.drop('ClaimNb', axis=1)
test    = test.drop('ClaimNb', axis=1)
val     = val.drop('ClaimNb', axis=1)
print(train.shape, test.shape, val.shape )
```

```
(426074, 9) (133149, 9) (106519, 9)
```

[14]:
```
#Procesamiento de la data
# SELECCION DE LA DATA PARA TRANSFORMAR
```

```
# Train
cat_item = train[[ 'Area', 'Region','VehGas', 'VehBrand','VehPower', 'VehAge', ␣
↪'DrivAge' ]] #CATEGORICAS
norm_item = train[[ 'BonusMalus']] #NORMALIZACION

# Test
cat_item_test = test[['Area', 'Region', 'VehGas', 'VehBrand','VehPower',␣
↪'VehAge',  'DrivAge' ]] #CATEGORICAS
norm_item_test = test[['BonusMalus']] #NORMALIZACION

cat_item_test = val[['Area', 'Region', 'VehGas', 'VehBrand','VehPower',␣
↪'VehAge',  'DrivAge' ]] #CATEGORICAS
norm_item_test = val[['BonusMalus']] #NORMALIZACION
#Damos formato a los datos para introducir a los calculos.

full_pipeline = ColumnTransformer([("nor", MinMaxScaler(), list(norm_item)),
                                   ("cat", OneHotEncoder(), list(cat_item)),
                                   ], remainder = 'passthrough')
```

[15]:
```
#Transformamos la data, a categorica con onehotencoder y minmaxscaler
data_train = full_pipeline.fit_transform(train)
data_test  = full_pipeline.transform(test)
data_val   = full_pipeline.fit_transform(val)
```

[16]:
```
#Convertimos a formato pandas.DataFrame
X_train = pd.DataFrame(data_train.toarray(), columns= full_pipeline.
↪get_feature_names_out(),
                           index = train.index
                          )
X_test  = pd.DataFrame(data_test.toarray(), columns= full_pipeline.
↪get_feature_names_out(),
                           index = test.index
                          )
X_val   = pd.DataFrame(data_val.toarray(), columns= full_pipeline.
↪get_feature_names_out(),
                           index = val.index
                          )
```

[17]:
```
X_train = X_train.drop(['cat__DrivAge_nan', 'cat__VehAge_nan'], axis=1)
X_test = X_test.drop(['cat__DrivAge_nan', 'cat__VehAge_nan'], axis=1)
X_val = X_val.drop(['cat__DrivAge_nan', 'cat__VehAge_nan'], axis=1)
print(X_train.head(4))
X_train.info()
```

```
        nor__BonusMalus  cat__Area_A  cat__Area_B  cat__Area_C  cat__Area_D  \
447970         0.060606          0.0          0.0          1.0          0.0
295691         0.000000          0.0          0.0          1.0          0.0
```

```
243354         0.454545          0.0          1.0          0.0          0.0
474452         0.303030          0.0          0.0          0.0          0.0

        cat__Area_E  cat__Area_F  cat__Region_Alsace  cat__Region_Aquitaine  \
447970          0.0          0.0                 0.0                    0.0
295691          0.0          0.0                 0.0                    0.0
243354          0.0          0.0                 0.0                    0.0
474452          1.0          0.0                 0.0                    0.0

        cat__Region_Auvergne  cat__Region_Basse-Normandie  \
447970                   0.0                          0.0
295691                   0.0                          0.0
243354                   0.0                          0.0
474452                   0.0                          0.0

        cat__Region_Bourgogne  cat__Region_Bretagne  cat__Region_Centre  \
447970                    0.0                   0.0                 1.0
295691                    0.0                   1.0                 0.0
243354                    0.0                   0.0                 1.0
474452                    0.0                   0.0                 0.0

        cat__Region_Champagne-Ardenne  cat__Region_Corse  \
447970                            0.0                0.0
295691                            0.0                0.0
243354                            0.0                0.0
474452                            0.0                0.0

        cat__Region_Franche-Comte  cat__Region_Haute-Normandie  \
447970                        0.0                          0.0
295691                        0.0                          0.0
243354                        0.0                          0.0
474452                        0.0                          0.0

        cat__Region_Ile-de-France  cat__Region_Languedoc-Roussillon  \
447970                        0.0                                0.0
295691                        0.0                                0.0
243354                        0.0                                0.0
474452                        1.0                                0.0

        cat__Region_Limousin  cat__Region_Midi-Pyrenees  \
447970                   0.0                         0.0
295691                   0.0                         0.0
243354                   0.0                         0.0
474452                   0.0                         0.0

        cat__Region_Nord-Pas-de-Calais  cat__Region_Pays-de-la-Loire  \
447970                             0.0                           0.0
295691                             0.0                           0.0
```

```
243354                           0.0                           0.0
474452                           0.0                           0.0

        cat__Region_Picardie  cat__Region_Poitou-Charentes  \
447970                   0.0                           0.0
295691                   0.0                           0.0
243354                   0.0                           0.0
474452                   0.0                           0.0

        cat__Region_Provence-Alpes-Cotes-D'Azur  cat__Region_Rhone-Alpes  \
447970                                      0.0                      0.0
295691                                      0.0                      0.0
243354                                      0.0                      0.0
474452                                      0.0                      0.0

        cat__VehGas_Diesel  cat__VehGas_Regular  cat__VehBrand_B1  \
447970                 1.0                  0.0               0.0
295691                 0.0                  1.0               1.0
243354                 0.0                  1.0               1.0
474452                 1.0                  0.0               0.0

        cat__VehBrand_B10  cat__VehBrand_B11  cat__VehBrand_B12  \
447970                0.0                0.0                0.0
295691                0.0                0.0                0.0
243354                0.0                0.0                0.0
474452                0.0                0.0                0.0

        cat__VehBrand_B13  cat__VehBrand_B14  cat__VehBrand_B2  \
447970                0.0                0.0               0.0
295691                0.0                0.0               0.0
243354                0.0                0.0               0.0
474452                0.0                0.0               0.0

        cat__VehBrand_B3  cat__VehBrand_B4  cat__VehBrand_B5  \
447970               1.0               0.0               0.0
295691               0.0               0.0               0.0
243354               0.0               0.0               0.0
474452               0.0               0.0               0.0

        cat__VehBrand_B6  cat__VehPower_4  cat__VehPower_5  cat__VehPower_6  \
447970               0.0              0.0              0.0              0.0
295691               0.0              0.0              0.0              1.0
243354               0.0              1.0              0.0              0.0
474452               1.0              0.0              0.0              0.0

        cat__VehPower_7  cat__VehPower_8  cat__VehPower_9  cat__VehAge_0-5  \
447970              1.0              0.0              0.0              1.0
295691              0.0              0.0              0.0              0.0
```

```
243354              0.0              0.0              0.0              0.0
474452              1.0              0.0              0.0              1.0

        cat__VehAge_13+  cat__VehAge_6-12  cat__DrivAge_18-20  \
447970              0.0               0.0                 0.0
295691              0.0               1.0                 0.0
243354              0.0               1.0                 1.0
474452              0.0               0.0                 0.0

        cat__DrivAge_21-30  cat__DrivAge_31-40  cat__DrivAge_41-50  \
447970                 0.0                 0.0                 0.0
295691                 0.0                 0.0                 1.0
243354                 0.0                 0.0                 0.0
474452                 0.0                 1.0                 0.0

        cat__DrivAge_51-70  cat__DrivAge_71+  remainder__Density
447970                 1.0               0.0            4.653960
295691                 0.0               0.0            4.770685
243354                 0.0               0.0            4.077537
474452                 0.0               0.0            7.731053
<class 'pandas.core.frame.DataFrame'>
Int64Index: 426074 entries, 447970 to 497390
Data columns (total 57 columns):
 #   Column                                 Non-Null Count   Dtype
---  ------                                 --------------   -----
 0   nor__BonusMalus                        426074 non-null  float64
 1   cat__Area_A                            426074 non-null  float64
 2   cat__Area_B                            426074 non-null  float64
 3   cat__Area_C                            426074 non-null  float64
 4   cat__Area_D                            426074 non-null  float64
 5   cat__Area_E                            426074 non-null  float64
 6   cat__Area_F                            426074 non-null  float64
 7   cat__Region_Alsace                     426074 non-null  float64
 8   cat__Region_Aquitaine                  426074 non-null  float64
 9   cat__Region_Auvergne                   426074 non-null  float64
 10  cat__Region_Basse-Normandie            426074 non-null  float64
 11  cat__Region_Bourgogne                  426074 non-null  float64
 12  cat__Region_Bretagne                   426074 non-null  float64
 13  cat__Region_Centre                     426074 non-null  float64
 14  cat__Region_Champagne-Ardenne          426074 non-null  float64
 15  cat__Region_Corse                      426074 non-null  float64
 16  cat__Region_Franche-Comte              426074 non-null  float64
 17  cat__Region_Haute-Normandie            426074 non-null  float64
 18  cat__Region_Ile-de-France              426074 non-null  float64
 19  cat__Region_Languedoc-Roussillon       426074 non-null  float64
 20  cat__Region_Limousin                   426074 non-null  float64
 21  cat__Region_Midi-Pyrenees              426074 non-null  float64
 22  cat__Region_Nord-Pas-de-Calais         426074 non-null  float64
```

```
 23  cat__Region_Pays-de-la-Loire             426074 non-null  float64
 24  cat__Region_Picardie                     426074 non-null  float64
 25  cat__Region_Poitou-Charentes             426074 non-null  float64
 26  cat__Region_Provence-Alpes-Cotes-D'Azur  426074 non-null  float64
 27  cat__Region_Rhone-Alpes                  426074 non-null  float64
 28  cat__VehGas_Diesel                       426074 non-null  float64
 29  cat__VehGas_Regular                      426074 non-null  float64
 30  cat__VehBrand_B1                         426074 non-null  float64
 31  cat__VehBrand_B10                        426074 non-null  float64
 32  cat__VehBrand_B11                        426074 non-null  float64
 33  cat__VehBrand_B12                        426074 non-null  float64
 34  cat__VehBrand_B13                        426074 non-null  float64
 35  cat__VehBrand_B14                        426074 non-null  float64
 36  cat__VehBrand_B2                         426074 non-null  float64
 37  cat__VehBrand_B3                         426074 non-null  float64
 38  cat__VehBrand_B4                         426074 non-null  float64
 39  cat__VehBrand_B5                         426074 non-null  float64
 40  cat__VehBrand_B6                         426074 non-null  float64
 41  cat__VehPower_4                          426074 non-null  float64
 42  cat__VehPower_5                          426074 non-null  float64
 43  cat__VehPower_6                          426074 non-null  float64
 44  cat__VehPower_7                          426074 non-null  float64
 45  cat__VehPower_8                          426074 non-null  float64
 46  cat__VehPower_9                          426074 non-null  float64
 47  cat__VehAge_0-5                          426074 non-null  float64
 48  cat__VehAge_13+                          426074 non-null  float64
 49  cat__VehAge_6-12                         426074 non-null  float64
 50  cat__DrivAge_18-20                       426074 non-null  float64
 51  cat__DrivAge_21-30                       426074 non-null  float64
 52  cat__DrivAge_31-40                       426074 non-null  float64
 53  cat__DrivAge_41-50                       426074 non-null  float64
 54  cat__DrivAge_51-70                       426074 non-null  float64
 55  cat__DrivAge_71+                         426074 non-null  float64
 56  remainder__Density                       426074 non-null  float64
dtypes: float64(57)
memory usage: 188.5 MB
```

[18]:
```
print(X_test.shape, X_train.shape, X_val.shape)
```

```
(133149, 57) (426074, 57) (106519, 57)
```

0.0.1 Funcion para Calculo de Metricas

[19]:
```
# Creamos una funcion para calcular las metricas
def metricas(modelo, X_test, y_test):
    """metricas del modelo """
    y_pred = modelo.predict(X_test)
```

```
    print('Media de las Observaciones:  %f' % y_test.mean())
    print('Media de las Predicciones:  %f' % y_pred.mean())
    print(
        "MSE: %.5f"
        % mean_squared_error(
            y_test, y_pred)
    )
    print(
        "MAE: %.5f"
        % mean_absolute_error(
            y_test, y_pred)
    )
    # Ignorar las Predicciones no positivas, para el calculo de la devianza
    mask = y_pred > 0
    if (~mask).any():
        n_masked, n_samples = (~mask).sum(), mask.shape[0]
        print(
            "Atensión: Campos de estimador invalidos, Predicciones no positivas "
            f" por {n_masked} ejemplos fuera de la muestra {n_samples}. estas␣
↪predicciones "
            "son ignoradas para computar la devianza de poisson."
        )

    print(
        "Desviance Media de Poisson: %.3f"
        % mean_poisson_deviance(
            y_test[mask],
            y_pred[mask],

        )
    )
```

1

2 TensorFlow y Keras

2.0.1 Modelado Suponiendo una funcion de perdida Binaria

[20]:
```
entreda  = layers.Dense(10, activation='relu')
oculta_1 = layers.Dense(20, activation='relu')
oculta_2 = layers.Dense(15, activation='relu')
overf    = layers.Dropout(.1)
salida   = layers.Dense(1)

#configuracion del modelo
model = tf.keras.Sequential([entreda, oculta_1, oculta_2, salida
])
```

```
#compilacion
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy','mean_squared_error', 'mean_absolute_error' ])
```

[21]:
```
#ajuste del modelo
tic = time.time()
historial = model.fit(X_train, y_train,
              validation_data=(X_val, y_val),
              epochs=30, verbose=0)

tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}m' .format((tac-tic)/60 ))
print('Terminado')
```

```
-----------------------------------------------
Tiempo de Calculo:  3.9394m
Terminado
```

[22]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                580

 dense_1 (Dense)             (None, 20)                220

 dense_2 (Dense)             (None, 15)                315

 dense_3 (Dense)             (None, 1)                 16

=================================================================
Total params: 1,131
Trainable params: 1,131
Non-trainable params: 0
_________________________________________________________________
```

[23]:
```
loss, accuracy, MSE, MAE = model.evaluate(X_test, y_test, verbose=1)
print("Accuracy", accuracy)
```

```
4161/4161 [==============================] - 2s 452us/step - loss: 0.1594 -
accuracy: 0.9638 - mean_squared_error: 11.1912 - mean_absolute_error: 3.2631
Accuracy 0.9638224840164185
```

[24]:
```
preds = model.predict(X_test)
```

[25]:
```
np.exp(preds.mean())
```

[25]:
```
0.03976758
```

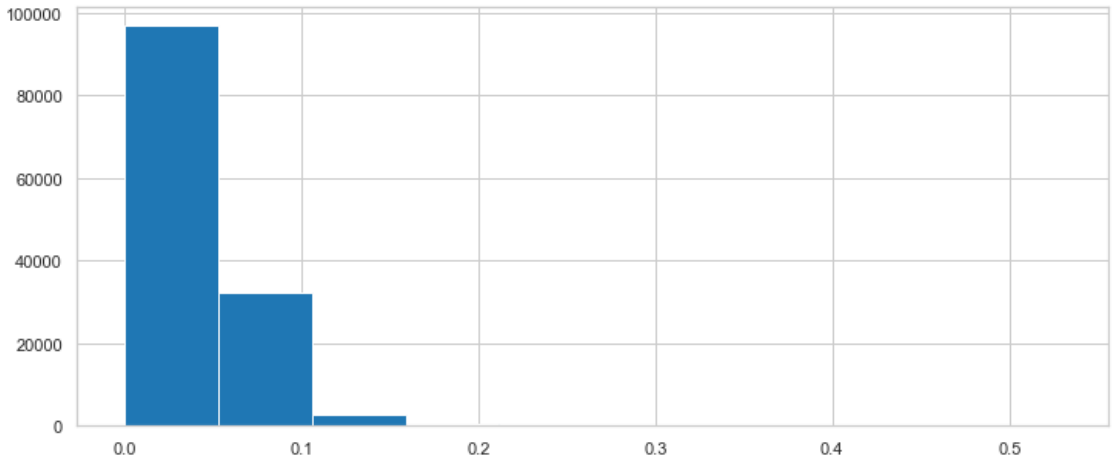
[28]:
```
exp_preds = np.exp(preds)
```

[29]:
```
print('Observaciones: {}' .format(y_test.mean()))
print('Prediccion: {}' .format(exp_preds.mean()))
print("La diferencia entre el las preds y las obser es : {}" .format(exp_preds.
 ↪mean()-y_test.mean()))
```

```
Observaciones: 0.03843063034645397
Prediccion: 0.046455904841423035
La diferencia entre el las preds y las obser es : 0.008025274494969065
```

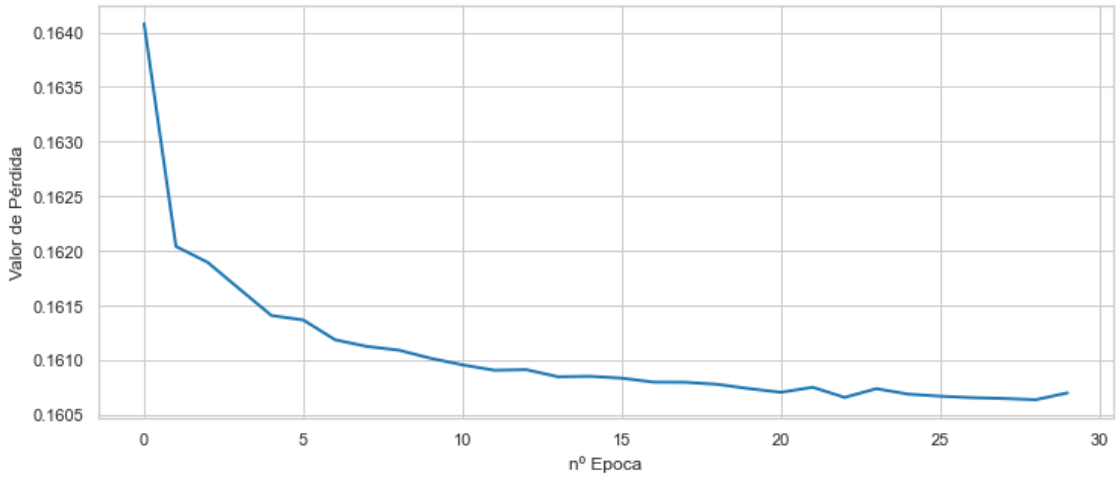
[30]:
```
plt.hist(exp_preds)
```

[30]:
```
(array([9.6781e+04, 3.2268e+04, 2.7430e+03, 5.9300e+02, 2.9800e+02,
        1.4500e+02, 1.2000e+02, 5.8000e+01, 4.3000e+01, 1.0000e+02]),
 array([3.6545851e-22, 5.2950136e-02, 1.0590027e-01, 1.5885042e-01,
        2.1180055e-01, 2.6475069e-01, 3.1770083e-01, 3.7065098e-01,
        4.2360109e-01, 4.7655123e-01, 5.2950138e-01], dtype=float32),
 <BarContainer object of 10 artists>)
```



[31]:
```
plt.xlabel('nº Epoca')
plt.ylabel('Valor de Pérdida')
plt.plot(historial.history['loss'])
```

[31]:
```
[<matplotlib.lines.Line2D at 0x17ea0cace80>]
```



3

3.0.1 Modelo multiclases (CategoricalCrossentropy)

[35]:
```
entreda  = layers.Input(shape=(57,),dtype= 'float64' ,name='Xtest')
oculta_1 = layers.Dense(33, activation='relu')
oculta_2 = layers.Dense(20, activation='relu')
oculta_3 = layers.Dense(10, activation='relu')
overf    = layers.Dense(.01)
salida   = layers.Dense(1 )

#configuracion del modelo
model = tf.keras.Sequential([entreda, oculta_1, oculta_2, oculta_3, salida
])

#compilacion
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy','mean_squared_error', 'mean_absolute_error' ])
```

[36]:
```
#ajuste del modelo
tic = time.time()
historial = model.fit(X_train, y_train ,
              validation_data=(X_val,y_val ),
              epochs=15, verbose=0)

tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}m' .format((tac-tic)/60 ))
print('Terminado')
```

```
-----------------------------------------
Tiempo de Calculo:  1.9983m
Terminado
```

[37]:
```
model.summary()
```

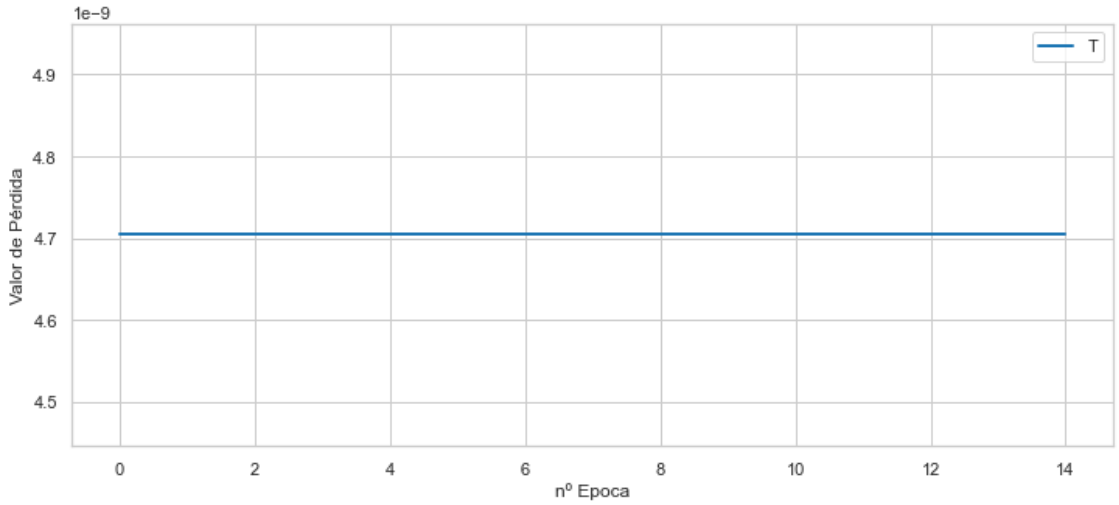
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 33)                1914

 dense_9 (Dense)             (None, 20)                680

 dense_10 (Dense)            (None, 10)                210

 dense_12 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,815
Trainable params: 2,815
Non-trainable params: 0
_________________________________________________________________
```

[38]:
```
loss, accuracy, MSE, MAE = model.evaluate(X_test, y_test, verbose=2)
print("Accuracy", accuracy)
```

```
4161/4161 - 2s - loss: 4.5813e-09 - accuracy: 0.9638 - mean_squared_error:
0.0632 - mean_absolute_error: 0.1761 - 2s/epoch - 363us/step
Accuracy 0.9638149738311768
```

[39]:
```
plt.xlabel('nº Epoca')
plt.ylabel('Valor de Pérdida')
plt.plot(historial.history['loss'])
plt.legend('Test')
#plt.plot(historial.epoch)
#plt.legend('Train')
```

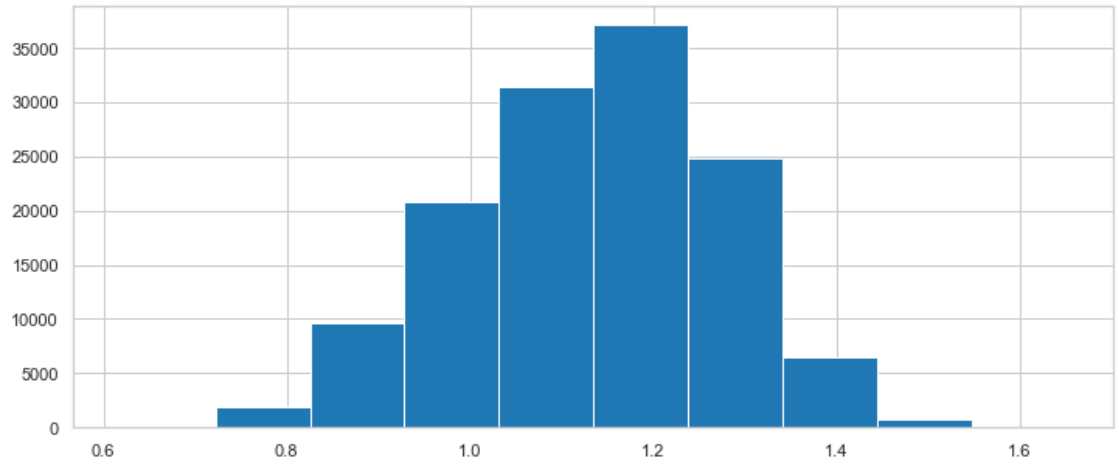
[39]:
```
<matplotlib.legend.Legend at 0x163f956aef0>
```



[46]:
```
preds = model.predict(X_test)
```

[47]:
```
plt.hist(np.exp(preds))
```

[47]:
```
(array([   90.,  1937.,  9692., 20768., 31440., 37056., 24787.,  6523.,
          796.,    60.]),
 array([0.6190393 , 0.72220665, 0.825374  , 0.9285413 , 1.0317087 ,
        1.134876  , 1.2380433 , 1.3412107 , 1.444378  , 1.5475454 ,
        1.6507127 ], dtype=float32),
 <BarContainer object of 10 artists>)
```



Lo anterior es señal de que la funcion objetivo o funcion de perdida no es la apropiada para evaluar los datos, el resultado de la frecuencia tiene una apariencia aproximada a

una distribucion normal.

3.0.2

3.1 Modelo Poisson, Perdida Poisson, Metrica Poisson Activación tanh y salida exponencial.

[42]:
```
# Modelo 4 capaas densas totalmente conectadas, funcion de activacion Tanh y␣
↪exponencial.
model = tf.keras.Sequential([
  layers.Dense(20, activation='tanh'),
  layers.Dense(15, activation='tanh'),
  layers.Dropout(.1),
  layers.Dense(1, activation='exponential')
])

model.compile(optimizer='adam',
              loss = tf.keras.losses.poisson,
              metrics=['accuracy','mean_squared_error', 'mean_absolute_error' ])
```

[43]:
```
# Entrenamiento del modelo  Poisson, Perdida Poisson, Activación tanh y salida␣
↪exponencial
tic = time.time()# huella de tiempo de entrada
historial = model.fit(X_train, y_train,
              validation_data=(X_val, y_val),
              epochs=30, verbose=1)

tac = time.time() # huella de tiempo de salida
print('------------------------------------------------')
print('Tiempo de Calculo:  {:.4f}m' .format((tac-tic)/60 ))
print('Terminado')
```

```
Epoch 1/30
13315/13315 [==============================] - 9s 690us/step - loss: 0.1688 -
accuracy: 0.9608 - mean_squared_error: 0.0471 - mean_absolute_error: 0.0800 -
val_loss: 0.1594 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0690
Epoch 2/30
13315/13315 [==============================] - 10s 722us/step - loss: 0.1643 -
accuracy: 0.9626 - mean_squared_error: 0.0421 - mean_absolute_error: 0.0760 -
val_loss: 0.1597 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0657
Epoch 3/30
13315/13315 [==============================] - 10s 723us/step - loss: 0.1638 -
accuracy: 0.9626 - mean_squared_error: 0.0421 - mean_absolute_error: 0.0758 -
val_loss: 0.1595 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0812
Epoch 4/30
```

```
13315/13315 [==============================] - 9s 711us/step - loss: 0.1634 -
accuracy: 0.9626 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0758 -
val_loss: 0.1589 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0686
Epoch 5/30
13315/13315 [==============================] - 9s 701us/step - loss: 0.1630 -
accuracy: 0.9627 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0756 -
val_loss: 0.1591 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0783
Epoch 6/30
13315/13315 [==============================] - 9s 696us/step - loss: 0.1628 -
accuracy: 0.9627 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0756 -
val_loss: 0.1598 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0406 -
val_mean_absolute_error: 0.0825
Epoch 7/30
13315/13315 [==============================] - 9s 702us/step - loss: 0.1626 -
accuracy: 0.9627 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0755 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0728
Epoch 8/30
13315/13315 [==============================] - 9s 697us/step - loss: 0.1624 -
accuracy: 0.9626 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0754 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0725
Epoch 9/30
13315/13315 [==============================] - 9s 700us/step - loss: 0.1622 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0752 -
val_loss: 0.1589 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0779
Epoch 10/30
13315/13315 [==============================] - 9s 699us/step - loss: 0.1620 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0752 -
val_loss: 0.1586 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0716
Epoch 11/30
13315/13315 [==============================] - 9s 702us/step - loss: 0.1620 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0752 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0723
Epoch 12/30
13315/13315 [==============================] - 9s 699us/step - loss: 0.1618 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0751 -
val_loss: 0.1595 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0674
Epoch 13/30
13315/13315 [==============================] - 9s 699us/step - loss: 0.1618 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0750 -
val_loss: 0.1587 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
```

```
val_mean_absolute_error: 0.0771
Epoch 14/30
13315/13315 [==============================] - 9s 691us/step - loss: 0.1616 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0750 -
val_loss: 0.1588 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0781
Epoch 15/30
13315/13315 [==============================] - 9s 700us/step - loss: 0.1616 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0750 -
val_loss: 0.1586 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0698
Epoch 16/30
13315/13315 [==============================] - 9s 699us/step - loss: 0.1614 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0753
Epoch 17/30
13315/13315 [==============================] - 9s 696us/step - loss: 0.1613 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0748 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0761
Epoch 18/30
13315/13315 [==============================] - 9s 696us/step - loss: 0.1612 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0748 -
val_loss: 0.1587 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0768
Epoch 19/30
13315/13315 [==============================] - 9s 700us/step - loss: 0.1611 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1583 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0718
Epoch 20/30
13315/13315 [==============================] - 9s 698us/step - loss: 0.1611 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0748 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0726
Epoch 21/30
13315/13315 [==============================] - 9s 695us/step - loss: 0.1610 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0748 -
val_loss: 0.1582 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0717
Epoch 22/30
13315/13315 [==============================] - 9s 697us/step - loss: 0.1609 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0716
Epoch 23/30
13315/13315 [==============================] - 9s 698us/step - loss: 0.1609 -
```

```
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1583 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0760
Epoch 24/30
13315/13315 [==============================] - 9s 702us/step - loss: 0.1608 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0715
Epoch 25/30
13315/13315 [==============================] - 9s 696us/step - loss: 0.1608 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1584 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0728
Epoch 26/30
13315/13315 [==============================] - 9s 706us/step - loss: 0.1607 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1588 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0681
Epoch 27/30
13315/13315 [==============================] - 9s 697us/step - loss: 0.1607 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1583 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0725
Epoch 28/30
13315/13315 [==============================] - 9s 700us/step - loss: 0.1606 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1587 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0743
Epoch 29/30
13315/13315 [==============================] - 9s 705us/step - loss: 0.1606 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1584 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0742
Epoch 30/30
13315/13315 [==============================] - 9s 703us/step - loss: 0.1605 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1582 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0732
--------------------------------------------
Tiempo de Calculo:  4.6716m
Terminado
```

[45]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 dense_11 (Dense)            (None, 20)                1160

 dense_12 (Dense)            (None, 15)                315

 dropout_3 (Dropout)         (None, 15)                0

 dense_13 (Dense)            (None, 1)                 16

=================================================================
Total params: 1,491
Trainable params: 1,491
Non-trainable params: 0
_________________________________________________________________
```

[46]:
```
# Modelo tanh & exponencial
loss, accuracy, MSE, MAE = model.evaluate(X_test, y_test, verbose=2)
print("Accuracy", accuracy)
```

```
4161/4161 - 2s - loss: 0.1595 - accuracy: 0.9638 - mean_squared_error: 0.0414 -
mean_absolute_error: 0.0735 - 2s/epoch - 369us/step
Accuracy 0.9638224840164185
```

[47]:
```
preds = model.predict(X_test)
preds.mean()
```

[47]:
```
0.03865309
```

[48]:
```
y_test.mean()
```

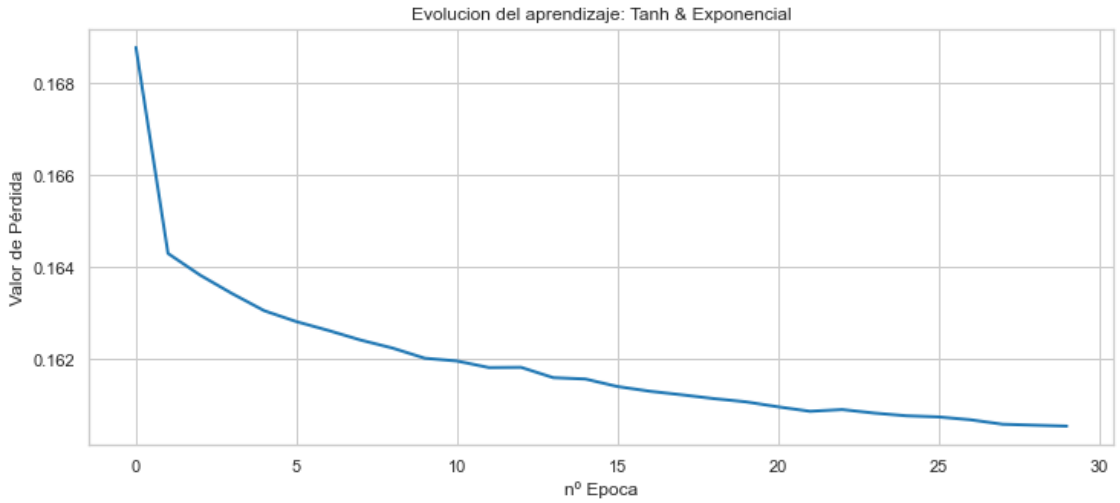
[48]:
```
0.03843063034645397
```

[49]:
```
    print(
         "Desviance Media de Poisson: %.3f"
        % mean_poisson_deviance(
            y_test,
            preds,
        )

        )
```

```
Desviance Media de Poisson: 0.249
```

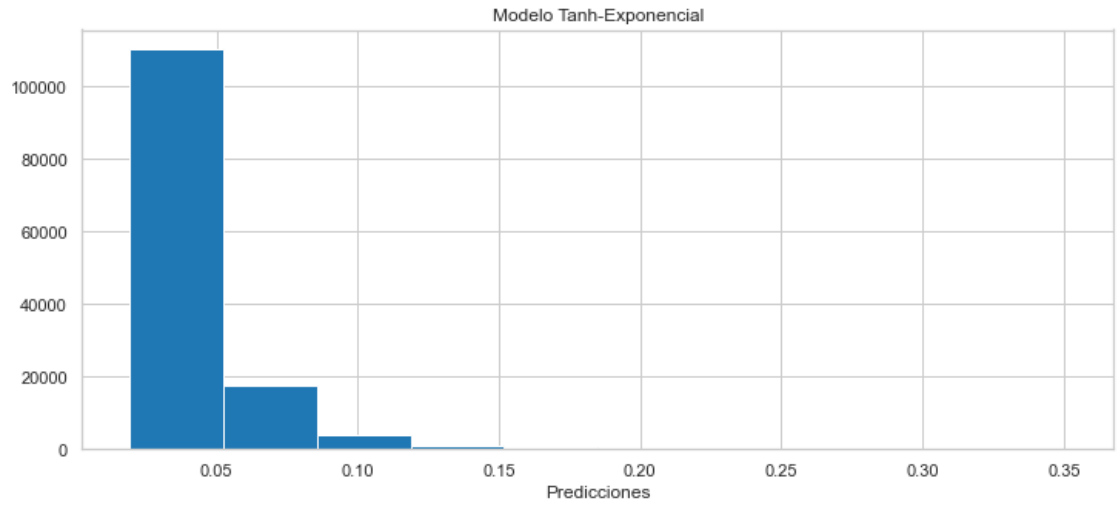
[50]:
```
plt.xlabel('nº Epoca')
plt.ylabel('Valor de Pérdida')
plt.plot(historial.history['loss'])
plt.title('Evolucion del aprendizaje: Tanh & Exponencial')
```

[50]:
```
Text(0.5, 1.0, 'Evolucion del aprendizaje: Tanh & Exponencial')
```



[51]:
```
plt.hist(preds)
plt.xlabel('Predicciones')
plt.title('Modelo Tanh-Exponencial')
```

[51]: Text(0.5, 1.0, 'Modelo Tanh-Exponencial')



3.1.1 Modelo de Perdida Poisson, Funcion de Activacion ReLu y Exponencial, Adicion de Capa Oculta

[52]:
```
entreda  = layers.Input(shape=(57,),dtype= 'float64' ,name='Xtest')
oculta_1 = layers.Dense(33, activation='relu')
oculta_2 = layers.Dense(20, activation='relu')
oculta_3 = layers.Dense(10, activation='relu')
overf    = layers.Dropout(.1)
salida   = layers.Dense(1,  activation='exponential' )

#configuracion del modelo
model = tf.keras.Sequential([entreda, oculta_1, oculta_2, oculta_3, salida
])

#compilacion
model.compile(optimizer='adam',
              loss=tf.keras.losses.poisson,
              metrics=['accuracy','mean_squared_error', 'mean_absolute_error' ])
```

[53]:
```
# Entrenamiento del modelo ReLU & Exponencial
tic = time.time()
historial = model.fit(X_train, y_train,
              validation_data=(X_val, y_val),
              epochs=30, verbose=1)

tac = time.time()
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}m' .format((tac-tic)/60 ))
print('Terminado')
```

```
Epoch 1/30
13315/13315 [==============================] - 10s 696us/step - loss: 0.1655 -
accuracy: 0.9613 - mean_squared_error: 0.0430 - mean_absolute_error: 0.0771 -
val_loss: 0.1594 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0669
Epoch 2/30
13315/13315 [==============================] - 9s 690us/step - loss: 0.1631 -
accuracy: 0.9627 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0755 -
val_loss: 0.1589 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0762
Epoch 3/30
13315/13315 [==============================] - 9s 690us/step - loss: 0.1628 -
accuracy: 0.9627 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0754 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0744
Epoch 4/30
13315/13315 [==============================] - 9s 695us/step - loss: 0.1626 -
accuracy: 0.9627 - mean_squared_error: 0.0420 - mean_absolute_error: 0.0753 -
```

```
val_loss: 0.1598 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0406 -
val_mean_absolute_error: 0.0826
Epoch 5/30
13315/13315 [==============================] - 9s 688us/step - loss: 0.1625 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0753 -
val_loss: 0.1596 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0651
Epoch 6/30
13315/13315 [==============================] - 9s 693us/step - loss: 0.1623 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0752 -
val_loss: 0.1587 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0770
Epoch 7/30
13315/13315 [==============================] - 9s 696us/step - loss: 0.1622 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0752 -
val_loss: 0.1590 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0789
Epoch 8/30
13315/13315 [==============================] - 9s 701us/step - loss: 0.1621 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0752 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0770
Epoch 9/30
13315/13315 [==============================] - 9s 700us/step - loss: 0.1619 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0751 -
val_loss: 0.1589 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0774
Epoch 10/30
13315/13315 [==============================] - 9s 711us/step - loss: 0.1619 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0751 -
val_loss: 0.1584 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0747
Epoch 11/30
13315/13315 [==============================] - 9s 702us/step - loss: 0.1618 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0751 -
val_loss: 0.1583 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0725
Epoch 12/30
13315/13315 [==============================] - 9s 709us/step - loss: 0.1618 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0751 -
val_loss: 0.1592 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0697
Epoch 13/30
13315/13315 [==============================] - 9s 701us/step - loss: 0.1617 -
accuracy: 0.9627 - mean_squared_error: 0.0419 - mean_absolute_error: 0.0750 -
val_loss: 0.1586 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0753
Epoch 14/30
```

```
13315/13315 [==============================] - 9s 704us/step - loss: 0.1616 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0750 -
val_loss: 0.1585 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0764
Epoch 15/30
13315/13315 [==============================] - 9s 706us/step - loss: 0.1616 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0750 -
val_loss: 0.1586 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0721
Epoch 16/30
13315/13315 [==============================] - 9s 713us/step - loss: 0.1615 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0750 -
val_loss: 0.1593 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0796
Epoch 17/30
13315/13315 [==============================] - 9s 711us/step - loss: 0.1614 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1595 - val_accuracy: 0.9635 - val_mean_squared_error: 0.0408 -
val_mean_absolute_error: 0.0801
Epoch 18/30
13315/13315 [==============================] - 10s 730us/step - loss: 0.1613 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1590 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0760
Epoch 19/30
13315/13315 [==============================] - 10s 729us/step - loss: 0.1613 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1594 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0797
Epoch 20/30
13315/13315 [==============================] - 10s 729us/step - loss: 0.1612 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1593 - val_accuracy: 0.9637 - val_mean_squared_error: 0.0406 -
val_mean_absolute_error: 0.0797
Epoch 21/30
13315/13315 [==============================] - 10s 717us/step - loss: 0.1611 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1596 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0406 -
val_mean_absolute_error: 0.0762
Epoch 22/30
13315/13315 [==============================] - 9s 712us/step - loss: 0.1611 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0748 -
val_loss: 0.1588 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0737
Epoch 23/30
13315/13315 [==============================] - 9s 704us/step - loss: 0.1610 -
accuracy: 0.9627 - mean_squared_error: 0.0418 - mean_absolute_error: 0.0749 -
val_loss: 0.1594 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
```

```
val_mean_absolute_error: 0.0678
Epoch 24/30
13315/13315 [==============================] - 9s 699us/step - loss: 0.1610 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1590 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0731
Epoch 25/30
13315/13315 [==============================] - 9s 703us/step - loss: 0.1609 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1589 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0404 -
val_mean_absolute_error: 0.0716
Epoch 26/30
13315/13315 [==============================] - 9s 703us/step - loss: 0.1609 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1593 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0712
Epoch 27/30
13315/13315 [==============================] - 9s 689us/step - loss: 0.1609 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1590 - val_accuracy: 0.9637 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0753
Epoch 28/30
13315/13315 [==============================] - 9s 693us/step - loss: 0.1609 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1590 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0406 -
val_mean_absolute_error: 0.0745
Epoch 29/30
13315/13315 [==============================] - 9s 699us/step - loss: 0.1608 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0748 -
val_loss: 0.1593 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0406 -
val_mean_absolute_error: 0.0787
Epoch 30/30
13315/13315 [==============================] - 9s 696us/step - loss: 0.1608 -
accuracy: 0.9627 - mean_squared_error: 0.0417 - mean_absolute_error: 0.0747 -
val_loss: 0.1599 - val_accuracy: 0.9638 - val_mean_squared_error: 0.0405 -
val_mean_absolute_error: 0.0743
------------------------------------------------
Tiempo de Calculo:  4.6911m
Terminado
```

[66]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 33)                1914
```

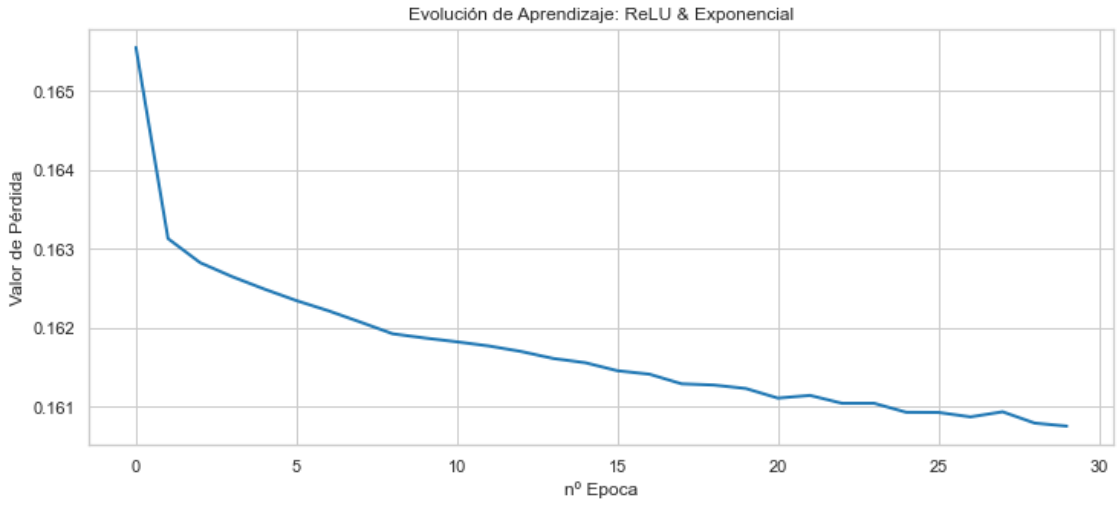
```
 dense_15 (Dense)            (None, 20)                680

 dense_16 (Dense)            (None, 10)                210

 dense_17 (Dense)            (None, 1)                 11

=================================================================
Total params: 2,815
Trainable params: 2,815
Non-trainable params: 0
_________________________________________________________________
```

[62]:
```
# ReLU Exponencial
loss, accuracy, MSE, MAE = model.evaluate(X_test, y_test, verbose=1)
print("Accuracy", accuracy)
```

```
4161/4161 [==============================] - 2s 461us/step - loss: 0.1606 -
accuracy: 0.9638 - mean_squared_error: 0.0415 - mean_absolute_error: 0.0746
Accuracy 0.9638149738311768
```

[56]:
```
plt.xlabel('nº Epoca')
plt.ylabel('Valor de Pérdida')
plt.plot(historial.history['loss'])
plt.title('Evolución de Aprendizaje: ReLU & Exponencial')
```

[56]: Text(0.5, 1.0, 'Evolución de Aprendizaje: ReLU & Exponencial')



[63]:
```
preds = model.predict(X_test)
preds.mean()
```

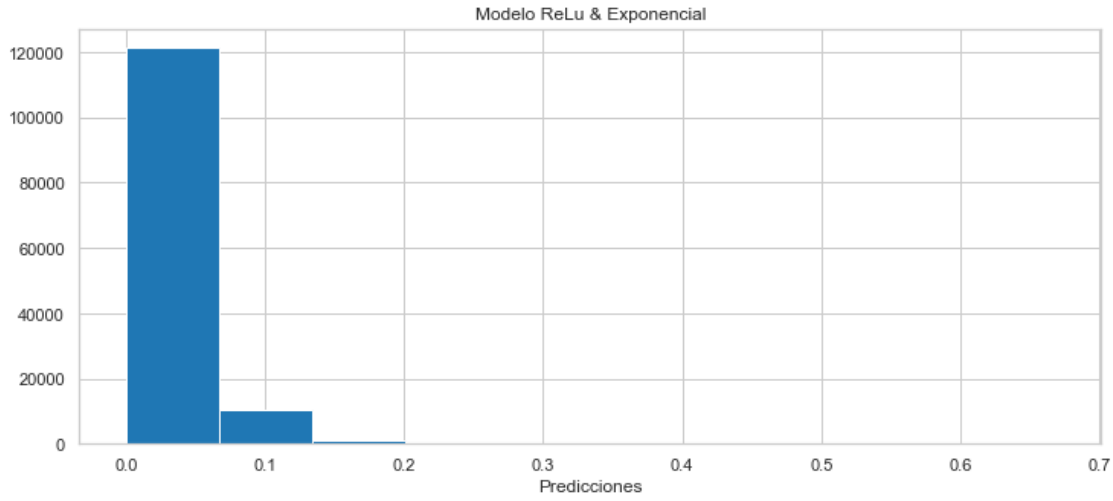
[63]: 0.039912976

[65]:
```
    print(
         "Desviance Media de Poisson: %.6f"
        % mean_poisson_deviance(
            y_test,
            preds,
        )

        )
```

```
Desviance Media de Poisson: 0.250901
```

[67]:
```
plt.hist(preds)
plt.xlabel('Predicciones')
plt.title('Modelo ReLu & Exponencial')
```

[67]:
```
Text(0.5, 1.0, 'Modelo ReLu & Exponencial')
```



4

5 ScikitLearn

5.0.1 Modelo MLPRegressor model_1: ReLu

[129]:
```
# Modelo MLPRegressor model_1

tic = time.time()# huella de tiempo de entrada

mlp_reg = MLPRegressor(hidden_layer_sizes=[20, 15, 10, ], random_state=33,␣
 ↪activation = "relu", beta_1=0.01)
```

```
pipeline = make_pipeline(mlp_reg)
pipeline.fit(X_train, y_train)

tac = time.time() # huella de tiempo de salida
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
```

```
-----------------------------------------------
Tiempo de Calculo:  12.6159s
```

[130]:
```
metricas(modelo=pipeline, X_test= X_test, y_test=y_test)
# Modelo MLPRegressor model_1
```

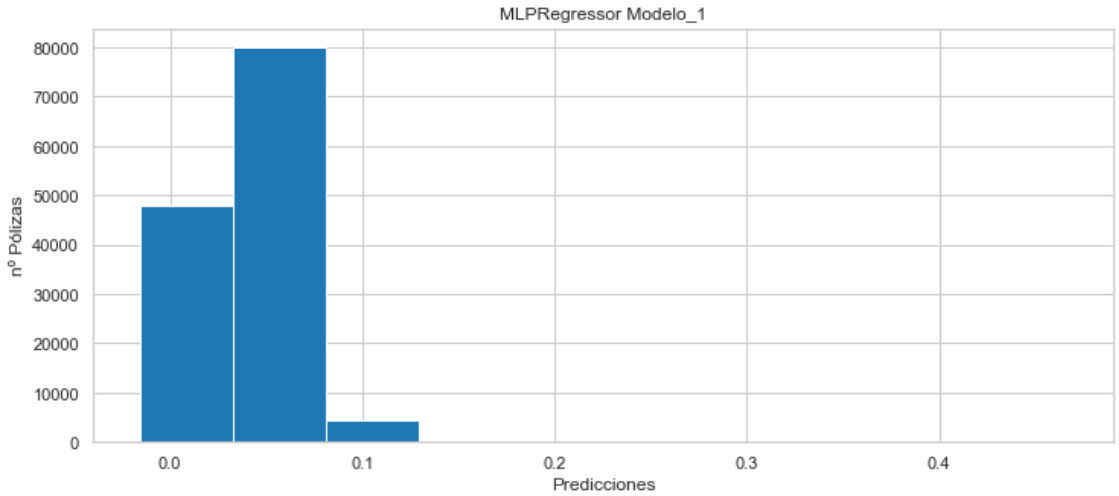
```
Media de las Observaciones:  0.038431
Media de las Predicciones:  0.042157
MSE: 0.04148
MAE: 0.07679
Atensión: Campos de estimador invalidos, Predicciones no positivas  por 27
ejemplos fuera de la muestra 133149. estas predicciones son ignoradas para
computar la devianza de poisson.
Desviance Media de Poisson: 0.249
```

[109]:
```
y_pred.mean()
```

[109]:
```
0.042156947941526496
```

[136]:
```
plt.hist(y_pred)
plt.title('MLPRegressor Modelo_1')
plt.xlabel('Predicciones')
plt.ylabel('nº Pólizas')
```

[136]:
```
Text(0, 0.5, 'nº Pólizas')
```



5.0.2 MLPRegressor Modelo_2 Tanh

[140]:
```
# MLPRegressor modelo_2

tic = time.time()# huella de tiempo de entrada

mlp_reg = MLPRegressor(hidden_layer_sizes=[20, 15, 10,], random_state=33,␣
 ↪activation = "tanh", beta_1=0.01)
pipeline = make_pipeline(mlp_reg)
pipeline.fit(X_train, y_train)


tac = time.time() # huella de tiempo de salida
print('-----------------------------------------------')
print('Tiempo de Calculo:  {:.4f}s' .format(tac-tic ))
```

```
-----------------------------------------------
Tiempo de Calculo:  14.0178s
```

[128]:
```
pipeline.feature_names_in_
pipeline.get_params
```

[128]:
```
<bound method Pipeline.get_params of Pipeline(steps=[('mlpregressor',
                 MLPRegressor(activation='tanh', beta_1=0.01,
                              hidden_layer_sizes=[20, 15, 10],
                              random_state=33))])>
```

[141]:
```
metricas(modelo=pipeline, X_test= X_test, y_test=y_test)
```

```
Media de las Observaciones:  0.038431
Media de las Predicciones:  0.041151
MSE: 0.04145
MAE: 0.07592
Atensión: Campos de estimador invalidos, Predicciones no positivas  por 43
ejemplos fuera de la muestra 133149. estas predicciones son ignoradas para
computar la devianza de poisson.
Desviance Media de Poisson: 0.249
```