Máster en Ciencias Actuariales y Financieras

TRABAJO FINAL DE MÁSTER

Título: Econometría versus Machine Learning: impacto de la disrupción tecnológica en la modelización de los seguros de vida

Autoría: Rubén García Tallante

Tutoría: Dr. Salvador Torra Porras

Curso académico: 2022/2023

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

Econometría versus Machine Learning: impacto de la disrupción tecnológica en la modelización de los seguros de vida

Autoría: Rubén García Tallante

Tutoría: Dr. Salvador Torra Porras

Agradecimientos

A los ilustres docentes del presente máster,
por enseñarme el camino de la profesión actuarial

A mi familia y seres queridos,
por enseñarme el camino de la vida

"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 han sido expresadas en el texto"

RESUMEN

Inmersos en un paradigma socioeconómico donde el valor de los datos es cada vez más notorio, existe una creciente demanda de técnicas avanzadas y precisas para su modelización. En el transcurso del presente análisis serán comparados los modelos econométricos tradicionales y los modelos basados en Machine Learning desarrollados durante la disrupción tecnológica de la que formamos parte.

Inicialmente, serán descritas las semejanzas y diferencias que presentan ambos enfoques desde un punto de vista teórico, prosiguiendo con un análisis empírico consistente en el desarrollo de un modelo predictivo para la clasificación del riesgo individual en los seguros de vida mediante distintas técnicas de modelización. Para concluir, será elaborada una comparativa enfatizando en las fortalezas y limitaciones de cada una de las alternativas consideradas, proporcionando información genuina y valiosa para las compañías que operan en la industria de los seguros de vida.

Palabras clave:

Econometría · Machine Learning · Seguros de Vida · Modelización del Riesgo · Python

ABSTRACT

Immersed within a socioeconomic paradigm where the significance of data is increasingly prominent, there is a mounting demand for advanced and precise techniques concerning its modeling. Within the scope of this analysis, a comparison shall be drawn between the traditional econometric models and the Machine Learning based models developed during the technological disruption we are presently experiencing.

First and foremost, the theoretical similarities and disparities between both approaches shall be described. Subsequently, an empirical analysis shall be conducted to develop a predictive model for individual risk classification in life insurance using various modeling techniques. Ultimately, a comparison shall be drawn, highlighting the strengths and limitations of each alternative considered, thus providing genuine and valuable information for companies operating within the life insurance industry.

Keywords:

Econometrics · Machine Learning · Life Insurance · Risk Modelling · Python

TABLA DE CONTENIDOS

TABLA DE ECUACIONES

TABLA DE FIGURAS

I) INTRODUCCIÓN

Durante los últimos años, la disrupción tecnológica ha generado cambios significativos en diversos sectores económicos entre los que se encuentra la industria aseguradora, que ha adoptado técnicas basadas en el aprendizaje automático para modelar riesgos. Estas técnicas han surgido como una alternativa a los modelos econométricos tradicionales y han abierto nuevas posibilidades para la valoración y diseño de productos en el sector. No obstante, todavía existe un debate abierto sobre cuál entre ambos enfoques es más eficaz para modelizar los riesgos asegurados, máxime tomando en consideración el binomio del rendimiento de los modelos frente al coste asociado a su implementación.

La modelización del riesgo mediante técnicas econométricas tradicionales ha sido durante décadas la metodología predominante, sin embargo, su uso está siendo cuestionado a causa de la creciente demanda de herramientas más avanzadas y precisas en un paradigma en el que el valor de los datos es cada vez más evidente y su disponibilidad más abundante. Como resultado, la aplicación de modelización mediante inteligencia artificial está ganando importancia propulsada por el aprendizaje automático, la traducción del término anglosajón Machine Learning. Mientras que en seguros de no vida la modelización mediante estas técnicas innovadoras ya cuenta con bases académicas consolidadas y potentes investigaciones, en los seguros de vida su progreso está siendo más tardío.

En España, el sector asegurador de vida cuenta con un elevado grado de desarrollo, acentuado tras experimentar un importante crecimiento en los últimos tiempos. A lo largo del año 2022, el volumen estimado de primas emitidas de seguro directo a nivel nacional alcanzó los 24.535 millones de euros, representando un crecimiento del 4,18% respecto del año anterior y el 37,88% del volumen total de primas de seguros en España (ICEA, 2023). Estas cifras no son para nada despreciables, pues en ese mismo año representaron un 1,85% del Producto Interior Bruto español (INE, 2023). Su oferta comercial está compuesta por seguros de vida riesgo, que cubren exclusivamente el fallecimiento del asegurado, y con mayor volumen de negocio liderando el ramo se encuentran los seguros de vida ahorro, que combinan una cobertura de fallecimiento con un componente de ahorro a largo plazo.

El objetivo primordial que será desarrollado a lo largo del presente informe es analizar el impacto de la disrupción tecnológica en la modelización de los seguros de vida, evaluando las semejanzas y diferencias que presentan los distintos modelos desde un punto de vista teórico, añadiendo valor cuantitativo mediante un análisis empírico basado en la modelización predictiva para la clasificación del riesgo a partir de datos de clientes reales publicados por la aseguradora estadounidense Prudential. Será evaluado el rendimiento de diez alternativas mediante diferentes métricas con la finalidad de optimizar la selección de clientes y la fijación de precios.

Finalmente, serán expuestas conclusiones sobre los distintos modelos predictivos del riesgo asociado a los seguros de vida, concretando sus fortalezas y limitaciones a nivel comparativo, siendo también valorado el potencial de la combinación entre la econometría tradicional y el aprendizaje automático para la optimización de la modelización, un ámbito de rigurosa actualidad. En definitiva, el presente informe contribuye a la comprensión del impacto de la disrupción tecnológica hacia la práctica aseguradora, enfatizando en sus efectos sobre la modelización de los seguros de vida.

II) FUNDAMENTOS TEÓRICOS

En la actualidad estamos inmersos en una revolución entorno a la forma en que se analizan y predicen los fenómenos económicos. Cabe remarcar que la modelización es una tarea crucial en la industria aseguradora, ya que proporciona la base para la selección de carteras y el cálculo de primas adecuadas, permitiendo la gestión de riesgos asociados a las pólizas aseguradas. En este sentido, existen dos enfoques principales para la modelización en el sector de los seguros de vida: los modelos econométricos tradicionales y los modelos basados en Machine Learning. Ambas opciones utilizan técnicas estadísticas para analizar y predecir el comportamiento de la variable objetivo, pero difieren en su enfoque teórico y metodológico. A continuación, serán descritos los fundamentos de ambos enfoques.

Comenzando por los modelos econométricos tradicionales, son aquellos desarrollados e implementados antes de la revolución computacional de los años noventa. Éstos emplean técnicas matemáticas y estadísticas para describir y explicar las relaciones económicas entre las variables analizadas, basándose en la teoría económica establecida a priori. Al estar fundamentados en una teoría consolidada, ofrecen resultados fácilmente interpretables y brindan información práctica sobre el comportamiento de las variables económicas analizadas.

En el presente estudio serán desarrollados cuatro de los modelos econométricos tradicionales más populares para la predicción multinomial[1]: la **regresión logística**, definida por estimar las probabilidades de pertenecer a cada categoría en comparación con una categoría de referencia mediante la función Logit con parámetros de regresión; el **análisis discriminante lineal**, cuyo objetivo es buscar combinaciones lineales de las variables explicativas que maximicen la separación entre las categorías de la variable respuesta; el modelo de **Naive Bayes**, el cual asume que las variables explicativas son independientes entre sí basándose en el popular teorema de Bayes; y los **árboles de decisión,** una herramienta econométrica no paramétrica la cual divide iterativamente el conjunto de datos en subconjuntos más pequeños en función de una variable de decisión para predecir una variable respuesta, generando la posibilidad de capturar relaciones no lineales dentro de los datos analizados.

Por otro lado, los modelos basados en Machine Learning pertenecen a una rama de la inteligencia artificial cuyo objetivo es el desarrollo de algoritmos que permitan a los algoritmos aprender y tomar decisiones basadas en datos, sin necesidad de programar los modelos explícitamente para cada una de las tareas que desarrollen durante el proceso de modelización. De esta forma se persigue la meta de crear sistemas capaces de aprender automáticamente, fundamentándose en algoritmos que aprenden de los propios datos, explotando patrones y relaciones directas. Sus orígenes se remontan a teorías de medianos del siglo veinte, pero su desarrollo no fue popularizado hasta inicios del siglo veintiuno gracias al aumento del poder computacional y a la disponibilidad de bases de datos más extensas tras su digitalización. Una gran ventaja estructural de estos modelos respecto a los econométricos tradicionales es que no se basan en una especificación previa, sino que descubren en los datos patrones ocultos que podrían pasar desapercibidos en la modelización tradicional. Además, los modelos basados en Machine Learning se caracterizan por su flexibilidad adaptativa, lo que los hace aplicables a una amplia variedad de situaciones.

---

[1] Enfoque estadístico para clasificar conjuntos de datos en múltiples categorías mediante la asignación de probabilidades para cada alternativa del espacio muestral, ampliando el enfoque binomial tradicional.

En el contexto de los modelos de aprendizaje automático basados en Machine Learning, se pueden distinguir tres categorías principales según su metodología. Los modelos de **aprendizaje supervisado** que son entrenados mediante ejemplos de datos etiquetados donde se conoce la relación entre las características de entrada y la variable objetivo. Dichos modelos aprenden a realizar predicciones precisas sobre nuevos datos y son extremadamente útiles cuando se dispone de un conjunto de datos con variables explicativas y una variable objetivo conocida, como es el caso analizado.

Alternativamente, los modelos de **aprendizaje no supervisado** son utilizados cuando no se dispone de ejemplos de datos etiquetados o una variable objetivo específica, su aspecto principal es el de descubrir patrones y estructuras ocultas en los datos sin una guía explícita, resultando de gran utilidad para el análisis exploratorio de datos y la identificación de grupos o perfiles similares en una población. Finalmente, los modelos de **aprendizaje profundo o Deep Learning** están basados en redes neuronales artificiales con múltiples capas ocultas, las cuales tienen la capacidad de aprender representaciones de alto nivel de los datos de entrada, lo que les permite capturar características complejas y no lineales en los datos, demostrando un rendimiento revolucionario en procesamiento del lenguaje natural y en tareas gráficas de reconocimiento por imágenes, aunque también demostrando un excelente desempeño como modelos predictores de clasificación (Bengio, Y. et al., 2015).

En busca de una confrontación empírica ante los modelos econométricos tradicionales, en el presente estudio serán desarrollados los siguientes modelos basados en Machine Learning: **Random Forest**, cuyo algoritmo emplea múltiples árboles de decisión para realizar predicciones, combinando sus resultados mediante un proceso de votación para mejorar la precisión y reducir el sobreajuste; **Gradient Boosting**, el cual construye un modelo de predicción en forma de conjunto de árboles de decisión donde cada árbol se ajusta a los errores residuales del anterior, buscando minimizar su función de pérdida; **XGBoost**, un algoritmo del mencionado Gradient Boosting pero optimizado mediante técnicas adicionales como la regularización y la paralelización de los datos posteriormente detalladas; y **Redes Neuronales Artificiales**, inspiradas en el funcionamiento del cerebro humano y construidas con capas de nodos interconectados llamados neuronas, las cuales logran aprender patrones complejos a partir de los datos de entrada mediante un proceso de entrenamiento iterativo.

Adicionalmente, también serán analizados dos alternativas combinatorias de modelos ya mencionados: **Voting Classifier**, cuyo enfoque integra múltiples modelos de clasificación para tomar decisiones basadas en la mayoría de votos de los modelos individuales; y **Stacking Classifier**, el cual combina múltiples clasificadores de base al entrenar un clasificador final que utiliza las predicciones de los clasificadores de base como características de entrada, permitiendo capturar interrelaciones de los datos más complejas. Mediante la combinación de técnicas de modelización se pretende valorar si la agregación de metodologías heterogéneas permite optimizar la precisión y la estabilidad de sus resultados.

A lo largo del presente capítulo, serán analizados en detalle los **diez modelos predictivos multinomiales de clasificación,** describiendo sus fundamentos y pilares, comparando exhaustivamente sus similitudes y diferencias a nivel teórico, examinando cómo la elección entre los distintos enfoques metodológicos puede afectar a la precisión y credibilidad de las predicciones generadas por cada uno de los modelos.

II.1) Modelos Econométricos Tradicionales

II.1.1) Regresión Logística

El modelo de regresión logística es una técnica estadística desarrollada por el estadístico David Roxbee Cox en el año 1958, convirtiéndose posteriormente en una de las herramientas más utilizadas en el análisis de datos. En su forma más básica, el modelo de regresión logística se aplica a situaciones en las que la variable dependiente tiene dos categorías mutuamente excluyentes, como es el caso del binomio de que ocurra un evento o no ocurra. Sus fundamentos se basan en la función Logit, que modela la probabilidad de que la variable respuesta o dependiente pertenezca a una de las categorías en relación con las variables explicativas o independientes, transformando la probabilidad en una escala logarítmica que permite una interpretación más sencilla y facilita el análisis de las relaciones entre las variables independientes y la probabilidad de pertenecer a una categoría.

Sin embargo, hay casos en los que no es aplicable la teoría binomial, aquellos en los que la variable respuesta tiene más de dos categorías. En tales situaciones, el modelo de regresión logística se generaliza para el caso multinomial, el cual fue desarrollado en el documento 'The Multinomial Logit Model' publicado por Albert Madansky en el año 1959 con el objetivo de analizar y predecir la pertenencia a múltiples categorías.

En lugar de estimar directamente la probabilidad de pertenecer a cada categoría, el enfoque multinomial en la regresión logística permite estimar y comparar múltiples ecuaciones logísticas simultáneamente, una para cada categoría de la variable dependiente, como se puede observar en la fórmula de la Ecuación 1 con múltiples parámetros predictores, donde i es el número de variables explicativas, las β_i representan los coeficientes asociados a cada variable explicativa x_i en la ecuación logística para la categoría A.

$$\ln\left(\frac{p_A}{1-p_A}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i$$

$$\text{logit}(p_A) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i$$

Ecuación 1 Función Logit de la Categoría A. Elaboración Propia (2023)

Se estima un conjunto de coeficientes para cada ecuación logística, y la elección de la categoría de referencia se realiza mediante la comparación de las probabilidades logarítmicas entre las diferentes categorías. De esta manera, el modelo de regresión logística multinomial permite analizar y predecir la probabilidad de pertenecer a cada una de las categorías en comparación con una categoría de referencia. Posteriormente, el valor de la probabilidad de pertenecer a la categoría A puede ser obtenido mediante la inversa del logaritmo natural, como se muestra en la Ecuación 2.

$$\text{P}(A) = \frac{e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i}}{1 + e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i}}$$

Ecuación 2. Probabilidad de Pertenecer a la Categoría A. Elaboración Propia (2023)

La estimación de los parámetros beta en el modelo de regresión logística multinomial se lleva a cabo mediante técnicas de máxima verosimilitud. En el caso del modelo logístico multinomial, la función de verosimilitud se construye considerando las probabilidades de pertenecer a cada categoría en comparación con la categoría de referencia, obteniendo la verosimilitud total al multiplicar estas probabilidades para cada observación, persiguiendo el objetivo de encontrar los valores de los coeficientes que maximicen la probabilidad conjunta de observar los datos que se obtendrían si el modelo fuera verdadero.

En cuanto a la interpretación de los coeficientes en el modelo de regresión logística multinomial, sigue siendo similar a la del modelo binario, donde coeficientes β indican cómo se comparan las probabilidades logarítmicas entre las diferentes categorías de la variable respuesta, en función de las variables explicativas.

Si el coeficiente β de una variable explicativa es positivo, indica que un aumento en esa variable está asociado con un aumento en la probabilidad relativa de pertenecer a una categoría en particular en comparación con la categoría de referencia. Por el contrario, si el coeficiente es negativo, implica que un aumento en esa variable está relacionado con una disminución en la probabilidad relativa de pertenecer a esa categoría. En este caso, todas las comparaciones se realizan con respecto a la categoría de referencia elegida.

Además de interpretar los coeficientes β, el rendimiento y precisión de los modelos empleados a lo largo del presente informe pueden ser evaluados mediante una gran variedad de métricas entre las que destaca la curva ROC, cuyas iniciales indican Receiver Operating Characteristic, y supone una representación gráfica de la sensibilidad frente a la especificidad, es decir, la tasa de verdaderos positivos en el eje de las coordenadas frente a la tasa de verdaderos negativos en el eje de las abscisas, todos ellos para diferentes puntos de corte de probabilidad permitiendo evaluar la capacidad del modelo para discriminar entre las diferentes categorías de la variable respuesta. De esta métrica deriva el AUC-ROC, acrónimo anglosajón del área bajo la curva ROC, indicando un mejor rendimiento del modelo en términos de clasificación y predicción cuanto mayor sea su área, siendo detallado exhaustivamente en el posterior apartado de resultados.

Es importante destacar que el modelo de regresión logística multinomial se basa en la suposición de linealidad en escala Logit entre las variables explicativas y las probabilidades de pertenencia a cada categoría, un supuesto cuya verosimilitud es ínfima al aplicarlo a casos reales, por lo que la precisión del modelo puede verse perjudicada.

A fin de cuentas, el modelo de regresión logística multinomial es una extensión del modelo de regresión logística binomial que permite abordar problemas de clasificación con más de dos categorías en la variable respuesta. A través de múltiples ecuaciones logísticas, se estima un conjunto de coeficientes para cada categoría exceptuando para la categoría referenciada en cada caso, modelando posteriormente la probabilidad de pertenecer a cada categoría en función de las variables explicativas. Sin embargo, es importante enfatizar en las limitaciones causadas por la suposición de linealidad del modelo, una suposición teórica que habitualmente no se cumple en las bases de datos.

II.1.2) Análisis Discriminante Lineal

El modelo de análisis discriminante es una técnica desarrollada por el estadístico y biólogo Ronald Aylmer Fisher en el año 1936, con el objetivo de encontrar combinaciones lineales de variables explicativas que maximizaran la separación entre grupos predefinidos en una variable respuesta categórica, buscando una función discriminante lineal que permita clasificar adecuadamente nuevas observaciones en función de sus rasgos característicos.

En su variante lineal, es asumida la hipótesis que considera que las variables explicativas siguen una distribución multivariante normal en cada grupo y que las matrices de covarianza son iguales entre los grupos. Bajo estas suposiciones, se estima una matriz de covarianza común entre todas las categorías y un vector de medias para cada categoría, que representa la ubicación relativa de cada grupo en el espacio de las variables explicativas.

En consecuencia, el principal objetivo de la metodología del análisis discriminante lineal es maximizar la relación entre la varianza entre grupos y la varianza dentro de los grupos. Para lograrlo, son empleados valores propios que representan la importancia relativa de las direcciones discriminantes, y vectores propios de la matriz de covarianza que definen las combinaciones lineales óptimas de las variables explicativas. En definitiva, el procedimiento se basa en la factorización de matrices para encontrar las direcciones de proyección óptimas que maximizan la separación entre las múltiples clases.

El modelo de análisis discriminante lineal multinomial es una extensión del análisis discriminante lineal generalizada para casos en los que la variable dependiente tiene más de dos categorías, desarrollada por el estadístico británico Bernard Lewis Welch en el año 1939. Su principal objetivo es encontrar una función lineal discriminante que permita clasificar las observaciones en una de las múltiples categorías posibles. El mismo utiliza una combinación lineal de las variables explicativas ponderadas por los coeficientes estimados para calcular la distancia de cada observación a los centroides de cada categoría, asumiendo la hipótesis de igualdad de covarianza entre categorías, estimando una única matriz de covarianza común para todas las categorías. Por otro lado, el vector de medias es estimado para cada categoría, cuyo objetivo es el de representar su ubicación relativa en el espacio de las variables explicativas.

El procedimiento para calcular los coeficientes del modelo de análisis discriminante lineal multinomial es iniciado mediante el cálculo de las matrices de dispersión dentro de cada clase de la variable respuesta y entre clases, representado sus respectivas variabilidades. Seguidamente, se obtiene la matriz inversa de la matriz de dispersión de cada clase, calculando posteriormente el vector de coeficientes del modelo mediante la multiplicación de la matriz inversa de la matriz de dispersión para cada clase por la diferencia entre las medias de las clases. Finalmente, se proyectan las observaciones sobre un subespacio definido por los coeficientes del modelo, obteniendo como resultado las puntuaciones discriminantes para aplicar a cada observación de la base de datos (Welch, B. L., 1939).

Si asignamos A para cada una de las posibles categorías de la variable respuesta, con media μ_i y covarianza Σ igual entre categorías, la dispersión de la variabilidad de las clases se define como la covarianza de las medias de las clases μ, siendo formulada su expresión en la Ecuación 3.

$$\Sigma_b = \frac{1}{A}\sum_{i=1}^{A}(\mu_i - \mu)\,(\mu_i - \mu)^T$$

Ecuación 3. Dispersión entre la Variabilidad de las Categorías. Elaboración Propia (2023)

Adicionalmente, la separación S de las clases medida por el vector direccional w es formulada en la expresión de la Ecuación 4.

$$S = \frac{\vec{w}^T\,\Sigma_b\,\vec{w}}{\vec{w}^T\,\Sigma\,\vec{w}}$$

Ecuación 4. Separación S de las Clases medida por el Vector Direccional w. Elaboración Propia (2023)

En consecuencia, el modelo de análisis discriminante lineal multinomial proporciona una herramienta útil para el análisis y la clasificación de datos con múltiples categorías. Sin embargo, a pesar de que existan variantes más flexibles que permiten relajar algunas de sus hipótesis, es esencial tener en cuenta que el modelo tradicional está fundamentado en suposiciones relacionadas con la distribución multivariante normal y la igualdad de matrices de covarianza entre las distintas categorías, unas hipótesis que difícilmente se cumplen en bases de datos reales, por lo que es necesario evaluar la validez y el impacto de estas suposiciones antes de aplicar el modelo.

También es importante destacar que asume linealidad en la relación entre las variables explicativas y la variable respuesta, una condición que limita su capacidad para capturar relaciones no lineales en los datos, uno de sus principales puntos débiles y, por el contrario, una ventaja competitiva de los modelos de aprendizaje automático que serán posteriormente detallados.

Los coeficientes son calculados mediante la estimación de los parámetros que maximizan la separación entre las clases, y en términos de interpretación, miden cuantitativamente en términos de dirección y magnitud el impacto de cada variable explicativa en el proceso de clasificación, siendo el caso de coeficiente positivo el indica que un aumento en el valor de la variable predictora está asociado con una mayor probabilidad de pertenencia a una clase en particular, mientras que un coeficiente negativo indica una menor probabilidad de pertenencia a esa misma clase en cuestión. En su conjunto, los resultados permiten comprender cómo las variables predictoras influyen en la asignación de las observaciones a cada clase.

En resumen, el análisis discriminante lineal multinomial permite clasificar observaciones en una variable respuesta con múltiples categorías mediante la búsqueda de combinaciones lineales óptimas de variables explicativas, con el objetivo de maximizar la separación entre las distintas categorías. Su mayor debilidad se basa en pasar por alto relaciones no lineales en los datos y en sus suposiciones sobre la distribución multivariante normal y la igualdad de matrices de covarianza, unas situaciones poco verosímiles que deben tomarse en consideración a pesar de que el modelo en sí tiene la capacidad de generar información valiosa en relación a la categorización multinomial de la base de datos.

II.1.3) Naive Bayes

El modelo clasificatorio predictivo de Naive Bayes, también conocido como clasificador bayesiano ingenuo, se basa en el popular teorema de Bayes desarrollado por el matemático y estadístico británico Thomas Bayes en el siglo XVIII, el cual establece una relación entre la probabilidad condicional inversa y la probabilidad conjunta de dos eventos (Bayes, T. y Price, R., 1763).

El mencionado teorema de Bayes establece que la probabilidad de que ocurra un evento llamado A_i dado que ha ocurrido un evento B es igual a la probabilidad de que ocurra el evento B dado que ha ocurrido el evento A_i, multiplicado por la probabilidad a priori de que ocurra el evento A_i, siendo todo dividido por la probabilidad a priori de que ocurra el evento B. Ésta es formulada en la Ecuación 5.

$$P(A_i|B) = \frac{P(B|A_i)\,P(A_i)}{P(B)}$$

Ecuación 5. Fórmula del Teorema de Bayes. Elaboración Propia (2023)

Sin embargo, el modelo de Naive Bayes también cuenta con hipótesis simplificadoras además de su base en el teorema mencionado. La suposición clave es la independencia condicional entre las variables predictoras dada la variable de respuesta, lo cual significa que se asume que las variables explicativas son independientes entre sí una vez que se conoce la categoría de la variable de respuesta.

Es importante destacar que la palabra Naive del nombre del modelo basado en el teorema de Bayes es traducida del inglés al español como ingenuo o simplista, un adjetivo calificativo derivado de la suposición del modelo de independencia condicional, puesto que habitualmente las variables explicativas suelen tener cierto grado de correlación entre sí.

Aunque esta suposición es muy limitante ya que habitualmente no se cumple en la práctica, el modelo de Naive Bayes suele proporciona resultados aceptables y simplifica en gran medida los cálculos agilizando el proceso de modelización, su mayor ventaja competitiva frente a los modelos basados en Machine Learning, cuyo requerimiento computacional suele ser mucho más exigente.

Extendido a su variante multinomial, el teorema de Bayes es utilizado para calcular la probabilidad de que una observación pertenezca a una determinada categoría dado un conjunto de variables explicativas que son usadas como predictoras. A partir de la base de datos, se estiman las probabilidades a priori y a posteriori necesarias para realizar las clasificaciones posteriores.

Las probabilidades a priori se refieren a la probabilidad de que una observación pertenezca a una categoría específica sin tener en cuenta ninguna información adicional, siendo calculadas mediante la división del número de observaciones en cada categoría por el número total de observaciones en la muestra.

Alternativamente, las probabilidades a posteriori se refieren a la probabilidad de que una observación pertenezca a una categoría dada la información proporcionada por las variables explicativas siendo calculadas mediante la regla de Bayes formulada en la Ecuación 6, que utiliza las probabilidades a priori y la verosimilitud condicional de las variables predictoras dada la categoría de la variable de respuesta para calcular la probabilidad condicional.

$$P(A_i|B) = \frac{P(B|A_i)\,P(A_i)}{\sum_{k=1}^{n} P(B|A_k)\,P(A_k)}$$

Ecuación 6. Fórmula de la Regla de Bayes. Elaboración Propia (2023)

Finalmente, el cálculo de la densidad de la probabilidad de un cierto valor x_i asumiendo la clase *y* queda formulado en la Ecuación 7, agregando el valor x_i a la ecuación de la distribución Normal con parámetros σ_y^2 y μ_y, donde σ_y^2 cuantifica la varianza del valor x_i y μ_y su la media, en ambos casos en relación a su asociación con la clase *y*.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_y^2}\right)$$

Ecuación 7. Fórmula de la Densidad de la Probabilidad del Valor x_i dada la Clase y. Elaboración Propia (2023)

Una vez que se han estimado las probabilidades a priori y a posteriori, el modelo clasifica nuevas observaciones calculando las probabilidades a posteriori para cada categoría y seleccionando la categoría con la probabilidad más alta. Esto se logra calculando la verosimilitud condicional para cada categoría y multiplicándola por la probabilidad a priori correspondiente. El proceso de clasificación se basa en la idea de que una observación es más probable que pertenezca a una categoría si la verosimilitud condicional es alta y la probabilidad a priori es alta.

En resumen, el modelo clasificatorio predictivo de Naive Bayes multinomial se basa en el teorema de Bayes y en la suposición de independencia condicional entre las variables predictoras. Utiliza una distribución multinomial para modelar las variables predictoras y estima las probabilidades a priori y a posteriori para clasificar observaciones en categorías discretas.

A pesar de que la suposición de independencia condicional sea simplista por escaso cumplimiento en bases de datos reales, el modelo de Naive Bayes multinomial ha demostrado empíricamente su efectividad y es ampliamente utilizado al optimizar el binomio de rendimiento del modelo frente a recursos empleados para la modelización, ya que Naive Bayes requiere mucho menor poder computacional que otras alternativas consideradas en el presente informe.

II.1.4) Árboles de Decisión

A pesar de que la estructura de árboles de decisión fue originada y popularizada en las ciencias sociales, durante el año 1984 los estadísticos estadounidenses Leo Breiman, Jerome Friedman, Richard Olshen y Charles Stone publicaron un trabajo de investigación titulado 'Classification and Regression Trees', dando lugar a su popularización en el ámbito de la modelización econométrica.

El modelo se fundamenta en la teoría de clasificación en base a múltiples variables explicativas, consolidando una estructura jerárquica en forma de árbol al estar compuesta por nodos y ramas, donde cada nodo de decisión representa una variable explicativa y cada rama representa una regla de decisión que guía la clasificación de las observaciones en las distintas categorías, tal y como se observa gráficamente en la Figura 1 mediante un esquema muy simplificado de un árbol de decisión que usa dos variables explicativas para clasificar la base de datos en tres clases de la variable respuesta.

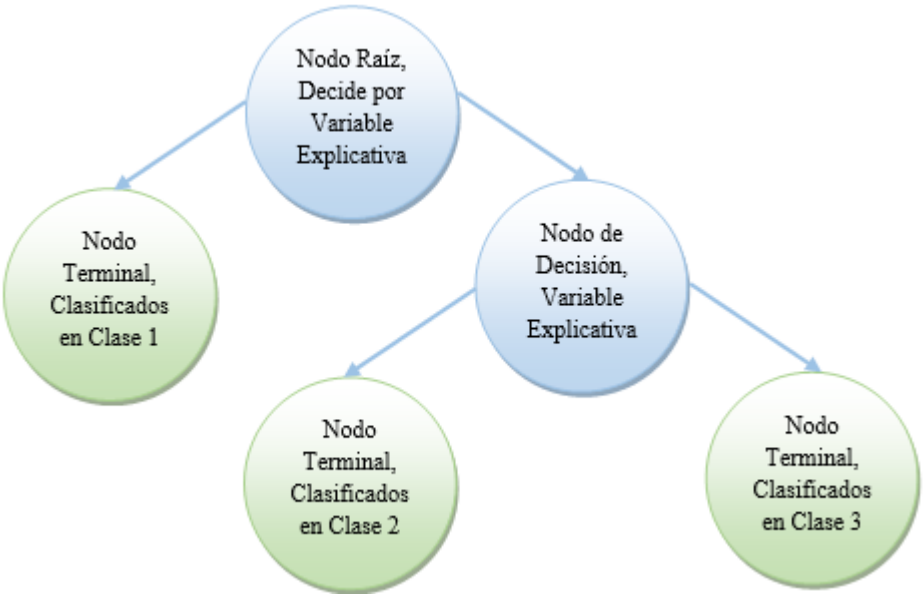

Figura 1. Esquema de Clasificación mediante Árbol de Decisión. Elaboración Propia (2023)

Como puede ser observado, los árboles de decisión cuyo objetivo es la segmentación se basan en la división recursiva de un conjunto de datos en función de variables explicativas. Una vez seleccionada la variable explicativa encargada de realizar la segmentación en cada nodo, se establece una regla de decisión basada en un conjunto de valores que divide las observaciones en términos de categorías de la variable respuesta creando subconjuntos más homogéneos entre sí. Este proceso de selección y división se repite de forma recursiva hasta que se alcanza un criterio de parada, como el tamaño mínimo del subconjunto o la profundidad máxima del árbol, llegando a los catalogados como nodos terminales, donde se ubica la clase de la variable respuesta a la que pertenece cada observación.

La creación de los árboles de decisión puede categorizarse en tres metodologías distintas: AID, iniciales de Automatic Interaction Detection, un método que se centra en la detección automática de interacciones entre variables al construir un árbol de decisión; CHAID, acrónimo de Chi-square Automatic Interaction Detection, una extensión de AID que utiliza pruebas de chi-cuadrado para evaluar la significancia de las interacciones entre variables categóricas en cada paso de construcción del árbol; y finalmente CART, las iniciales de Classification and Regression Trees, siendo esta la alternativa con mayor valor para el presente informe al ser óptima para problemas de clasificación, además de también ser adecuada ante problemas de regresión.

En términos estadísticos, el enfoque de división recursiva en un árbol de decisión utiliza medidas de información para seleccionar la variable explicativa que mejor discrimine las observaciones en cada nodo del árbol. Dos de las medidas de desigualdad más comunes son la entropía y el índice de Gini. La primera entropía mide la incertidumbre en un nodo dado, basándose en la distribución de categorías de las observaciones; por otro lado, el índice de Gini cuantifica la probabilidad de que dos observaciones seleccionadas aleatoriamente de una población sean clasificadas incorrectamente si se etiquetan al azar según la distribución de categorías en el nodo. Estas medidas son utilizadas para evaluar la calidad de la división de variables en el árbol de decisión y determinar la mejor forma de clasificar las observaciones en cada nodo, mostrando sus fórmulas en la Ecuación 8, donde *i* son los distintos nodos y *k* las distintas clases, siendo p_{ik} la proporción observada de la clase *k* dentro del nodo *i*.

$$Entropía_i = \sum_{i=1}^{C} -p_{i\,k}\ log_2\ p_{i\,k}$$

$$Índice\ de\ Gini_i = 1 - \sum_{k} p_{i\,k}^2$$

Ecuación 8. Fórmula de la Entropía y del Índice de Gini. Elaboración Propia (2023)

La popularidad de los árboles de decisión está fundamentada en su versatilidad, al manejar datos de variables categóricas y numéricas simultáneamente, pudiendo capturar interacciones entre ambos tipos de datos. Además, su interpretabilidad gráfica es muy simple, permitiendo visualizar el proceso de toma de decisiones completo de forma comprensiva y accesible a profesionales de otros sectores sin conocimientos avanzados de modelización. Otra gran ventaja competitiva es que su enfoque no paramétrico no requiere suposiciones sobre la distribución de los datos, una de las mayores limitaciones de los modelos detallados anteriormente.

Sin embargo, los árboles de decisión también presentan algunas limitaciones, ya que puede sufrir de sobreajuste cuando el árbol es demasiado complejo y se adapta demasiado a los datos con los que ha sido entrenado, lo que puede resultar en una baja generalización a nuevas bases de datos, por lo que será un factor a tener en cuenta en la hiperparametrización del modelo.

En resumen, el modelo multinomial basado en árboles de decisión logra clasificar observaciones en una variable respuesta con múltiples categorías posibles, persiguiendo el objetivo de encontrar la combinación óptima de variables explicativas y reglas de decisión que maximicen la precisión de la clasificación generada por la modelización.

II.2) Modelos Basados en Machine Learning

II.2.1) Random Forest

El modelo de Random Forest, traducido al español árboles aleatorios, es una técnica ampliamente utilizada en el campo del aprendizaje automático que combina la simplicidad y robustez de los árboles de decisión con el poder de la agregación y la aleatorización de los cálculos del modelo. Inicialmente fue propuesto por Tin Kam Ho en el año 1995, pero su popularización no llegaría hasta el año 2001 con la doctrina establecida por parte de Leo Breiman, el mismo estadístico estadounidense que desarrolló los árboles de decisión años atrás.

El concepto fundamental del modelo de Random Forest es la creación de un conjunto diverso de árboles de decisión, donde cada árbol se entrena de forma independiente en una muestra aleatoria de los datos utilizando un subconjunto aleatorio de variables predictoras, es decir, replicando múltiples veces la metodología de los árboles de decisión. A través de la agregación de los resultados individuales de los mismos, el modelo final de Random Forest es capaz de mejorar la precisión predictiva y mitigar el sobreajuste inherente en los árboles de decisión individuales, como ha sido comentado anteriormente. A continuación es detallado el procedimiento de construcción del modelo en cuestión:

Inicialmente, se lleva a cabo un procedimiento de Bootstraping o muestreo con reemplazo, con el que se consigue que cada árbol sea entrenado con una muestra única de datos seleccionados aleatoriamente. El modelo sigue con la selección aleatoria de variables explicativas, ya que en cada nodo de cada árbol se selecciona un subconjunto aleatorio de variables predictoras para tomarlas en consideración en la división de nodos, una medida que evita la correlación entre los distintos árboles.

Posteriormente se procede a la construcción de árboles de decisión para cada una de las muestras de datos seleccionadas aleatoriamente, construyéndolos mediante el algoritmo CART de división recursiva en el caso de árboles clasificatorios. Los árboles se construyen de manera independiente, como si se estuvieran elaborando varios modelos de árboles de decisión paralelamente, pudiendo crecer hasta que llegue a sus límites como el número mínimo de observaciones en un nodo o su máxima profundidad.

Finalmente, una vez que todos los árboles de decisión han sido calculados, las predicciones del modelo Random Forest son llevadas a cabo mediante la agregación de los resultados individuales. En el caso concreto de los Random Forest de regresión el resultado final es obtenido tras promediar las predicciones de los árboles de decisión, mientras que en aquellos cuyo objetivo es la clasificación predictiva multinomial es obtenida la clase final empleando una metodología de votación por mayoría, donde cada árbol emite un voto y la clase que más apoyo recibe es la que se considera como la clasificación final.

Todo el procedimiento que ha sido detallado en los párrafos anteriores, puede ser observado esquemáticamente en la Figura 2 mostrada a continuación.

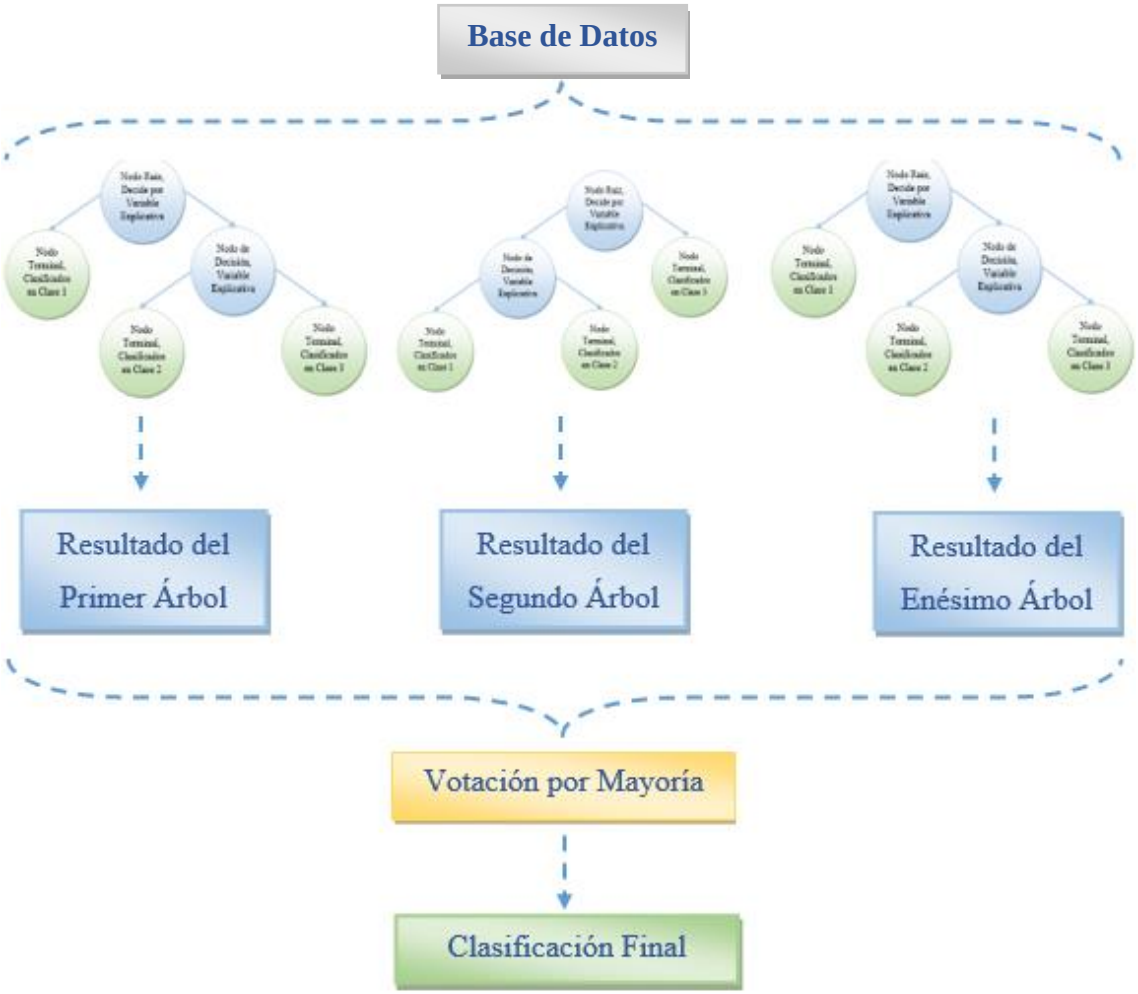

Figura 2. Esquema de la Metodología de Random Forest. Elaboración Propia (2023)

Una de las ventajas clave del Random Forest es su capacidad para manejar grandes bases de datos con multitud de variables explicativas, así como la resistencia al ruido de los datos manteniendo su fiabilidad ante datos atípicos juntamente con la capacidad de lidiar con variables irrelevantes, manejando adecuadamente las variables que no aportan información útil para la tarea de clasificación.

Sin embargo, también es importante destacar algunas de sus limitaciones, ya que a pesar de ser conocido por su robustez y capacidad para manejar datos ruidosos, puede tener dificultades para capturar relaciones no lineales complejas. Además, debido a los componentes mencionados de aleatorización y agregación, el modelo pierde por completo la facilidad de interpretación tan característica de los árboles de decisión.

En conclusión, el modelo Random Forest es una metodología poderosa y ampliamente utilizada al combinar la simplicidad de los árboles de decisión con la agregación y la aleatorización de los datos, lo que resulta en una mejora en la precisión predictiva y la capacidad para mitigar el sobreajuste. En consecuencia, Random Forest se ha convertido en un referente en la clasificación multinomial gracias a su enfoque de ensamble, el cual combina múltiples árboles de decisión y concluye mediante votación por mayoría, generando así una alternativa de modelización que permite capturar la complejidad de los datos y proporcionar resultados más robustos y fiables.

II.2.2) Gradient Boosting

El modelo de Gradient Boosting es una técnica de aprendizaje automático propuesta por el estadístico estadounidense Jerome Harold Friedman en el año 1999, la cual destaca por su eficacia en la resolución de problemas complejos de regresión y clasificación. El algoritmo en sí se fundamenta en la optimización de una función de pérdida mediante la que se representa la discrepancia entre las predicciones del modelo y los valores reales del conjunto de datos. En cada iteración se ajusta un modelo más simple, siendo habitualmente usados los árboles de decisión, y emplea un enfoque de descenso de gradiente para minimizar la función de pérdida. Esto implica calcular los gradientes de la función de pérdida con respecto a las predicciones del modelo y ajustar los parámetros del modelo en la dirección opuesta al gradiente, es decir, optimiza el modelo a través de ajustes sucesivos de sus parámetros.

El proceso de entrenamiento del modelo de Gradient Boosting se repite múltiples veces, ajustando los pesos de cada modelo en función de su contribución al rendimiento general del conjunto. De esta manera, se logra una combinación ponderada de modelos simples que se adapta de manera óptima a los datos de entrenamiento y generaliza bien a nuevos datos, el objetivo principal de la modelización predictiva de clasificación.

Además, el Gradient Boosting ofrece una serie de hiperparámetros que permiten ajustar y personalizar el modelo de acuerdo con las características específicas de cada base de datos. Estos hiperparámetros controlan aspectos como la tasa de aprendizaje y la profundidad de los árboles de decisión, detallados a continuación.

La tasa de aprendizaje es un hiperparámetro clave ya que controla la velocidad con la que el modelo aprende de los errores cometidos en cada iteración. Un valor bajo de la tasa de aprendizaje resulta en un aprendizaje más lento pero más preciso, mientras que un valor alto puede conducir a un aprendizaje más rápido pero a expensas de una mayor sensibilidad al ruido. En consecuencia, la selección adecuada de la tasa de aprendizaje es fundamental para encontrar un equilibrio óptimo entre la velocidad de convergencia y la precisión del modelo.

Alternativamente, la profundidad de los árboles de decisión es otro hiperparámetro importante al controlar la complejidad de los árboles de decisión y, por lo tanto, la capacidad de generalización del modelo. Un valor bajo de la profundidad del árbol resulta en árboles más simples y menos profundos, lo que puede llevar a un modelo más estable pero con menor capacidad de capturar relaciones complejas en los datos. Por otro lado, un valor alto de la profundidad del árbol permite capturar relaciones más complejas, pero también puede aumentar el riesgo de sobreajuste, especialmente si el conjunto de datos es pequeño. Por lo tanto, es necesario encontrar un equilibrio adecuado al seleccionar la profundidad de los árboles.

En consecuencia, la selección óptima de hiperparámetros es crucial para equilibrar sesgo y varianza en el modelo. Un modelo con alta varianza se ajusta demasiado a los datos de entrenamiento y tiene dificultades para generalizar, mientras que un modelo con alto sesgo simplifica demasiado y tiene una baja capacidad de predicción. Por lo tanto, es importante realizar una búsqueda sistemática de hiperparámetros y realizar validación cruzada parar configurar un modelo equilibrado y generalizable.

En pocas palabras, el enfoque del Gradient Boosting se basa en el concepto combinatorio de modelos más simples para así formar un modelo más fuerte y preciso. A diferencia de los métodos tradicionales de ensemble, como el Random Forest del apartado anterior que construye múltiples modelos independientes y luego los unifica mediante votación por mayoría, mostrado en el esquema de la Figura 2, el Gradient Boosting construye modelos en forma secuencial, donde cada modelo se ajusta a los errores residuales del modelo anterior. De esta manera, se busca mejorar continuamente el rendimiento del modelo en cada iteración, como se muestra en la Figura 3, donde los puntos rojos son predicciones incorrectas y los verdes son predicciones correctas.

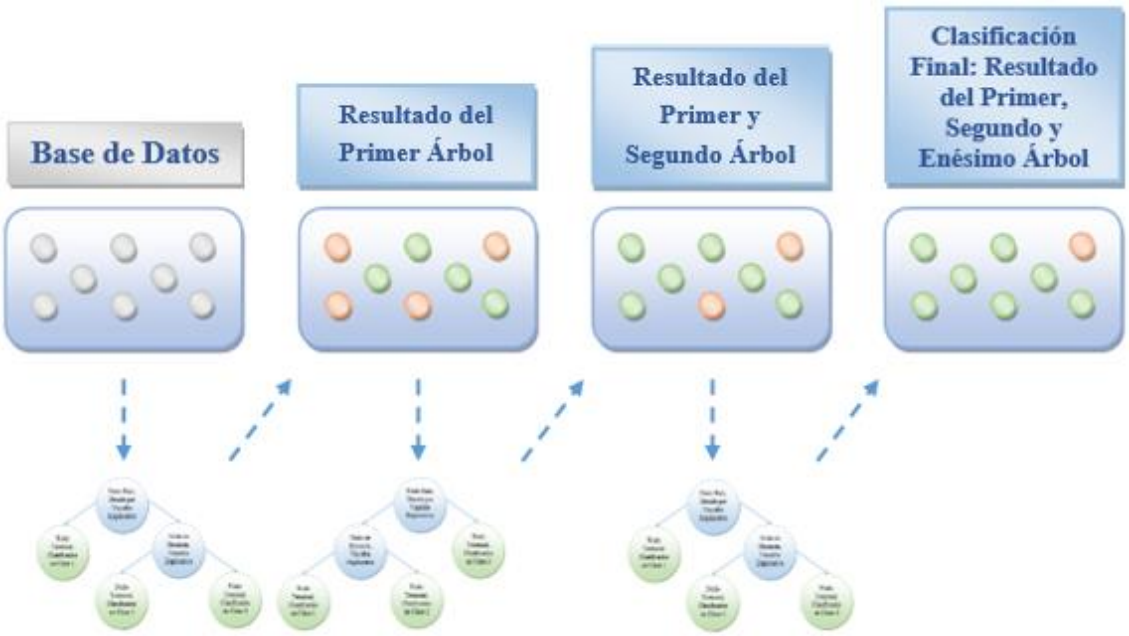

Figura 3. Esquema de la Metodología de Gradient Boosting. Elaboración Propia (2023)

Una de las ventajas clave del Gradient Boosting es su capacidad para capturar relaciones no lineales y características complejas en los datos, empleando un algoritmo altamente flexible que se desenvuelve sin problemas ante la necesidad de manejar conjuntos de datos grandes y de alta dimensionalidad, es decir, con un elevado número de variables explicativas. Sin embargo, su mayor limitación es causada por su naturaleza secuencial, ya que el proceso de entrenamiento y ajuste del modelo suele ser costoso y lento a nivel computacional respecto a las alternativas mencionadas anteriormente. Además, como el resto de modelos basados en Machine Learning, el Gradient Boosting está sujeto a problemas de sobreajuste si no se controlan adecuadamente los mencionados hiperparámetros.

En resumen, el modelo de Gradient Boosting es una alternativa de aprendizaje automático que utiliza el enfoque de descenso de gradiente para modelizar combinando múltiples modelos más sencillos, habitualmente árboles de decisión. A través de iteraciones sucesivas, el modelo se ajusta a los errores cometidos por los modelos previos, mejorando así su capacidad predictiva. Este enfoque permite construir modelos precisos y capaces de capturar relaciones complejas en los datos, siendo capaz de manejar grandes volúmenes de datos, lidiar con relaciones no lineales y capturar interacciones complejas entre variables, convirtiéndose en una herramienta valiosa para comprender y predecir el comportamiento de los datos analizados.

II.2.3) XGBoost

El modelo de Extreme Gradient Boosting, abreviado como XGBoost, es una técnica de aprendizaje automático desarrollada a partir del año 2014 por el ingeniero Tianqi Chen en colaboración con el equipo de investigación de Distributed Machine Learning Community. Desde su lanzamiento, ha ganado popularidad debido a su eficacia y capacidad para manejar problemas complejos de regresión y clasificación, siendo esta la alternativa de modelización más innovadora de las analizadas en el presente informe.

XGBoost es una implementación optimizada y escalable del algoritmo de Gradient Boosting detallado en el apartado anterior, el cual combina múltiples modelos más sencillos como árboles de decisión con el objetivo de consturuir un modelo más fuerte y preciso. De la misma manera que el modelo esquematizado en la Figura 3, el proceso de entrenamiento de XGBoost comienza con un árbol de decisión simple, y se van añadiendo nuevos árboles de decisión a medida que se van corrigiendo los errores de predicción del modelo anterior.

Sin embargo, la principal diferencia entre XGBoost y Gradient Boosting radica en su enfoque de optimización y en la implementación de técnicas aún más avanzadas. Aunque ambos métodos pertenecen a la familia de algoritmos de boosting, XGBoost introduce mejoras significativas que lo hacen más eficiente y preciso al utilizar el método de Newton-Raphson en el espacio de funciones, a diferencia del Gradient Boosting que se basa en el descenso de gradiente detallado anteriormente.

La conexión con el método de Newton-Raphson permite que XGBoost realice ajustes más precisos en cada iteración, ya que a pesar de que ambos métodos sean algoritmos de optimización utilizados para encontrar el mínimo de una función, Newton-Raphson tiene en cuenta tanto el gradiente mediante la pendiente como la curvatura mediante la derivada de segundo orden de la función de pérdida, resultando en un aprendizaje más preciso. Sin embargo, su mayor desventaja recae en el requerimiento de mayor poder computacional y en una mayor duración de su entrenamiento, siendo ambas consecuencias directas del aumento en la complejidad de su algoritmo.

Además, XGBoost incorpora una estrategia adicional llamada Regularized Learning Objective cuya función es la de agregar términos de penalización a la función objetivo, ayudando a controlar el sobreajuste del modelo y mejorando su capacidad de generalización a nuevos datos. En términos de rendimiento y escalabilidad, XGBoost también ofrece ventajas significativas, ya que ha sido específicamente diseñado para aprovechar al máximo la paralelización además de utilizar técnicas de compresión y particionamiento de datos para acelerar el proceso de entrenamiento y predicción.

En resumen, el modelo XGBoost se ha convertido en una de las técnicas más populares en el campo del aprendizaje automático debido a las mejoras que introduce sobre un modelo ya poderoso como es el caso del Gradient Boosting, optimizándolo a través de la implementación de técnicas más avanzadas en su algoritmo y mejorando su rendimiento y escalabilidad, provocando que a costa de mayores requisitos computacionales por su mayor complejidad, XGBoost sea una alternativa aún más poderosa y eficiente para la construcción de modelos predictivos de alta calidad y excelente rendimiento.

II.2.4) Redes Neuronales

El modelo de Redes Neuronales fue inicialmente propuesto por el neurólogo Warren Sturgis McCulloch conjuntamente con el matemático Walter Pitts en el año 1943, siendo considerado el primer modelo computacional inspirados en la biología humana. En la década de los cincuenta, las redes neuronales continuaron con su desarrollo gracias a trabajos pioneros del psicólogo estadounidense Frank Rosenblatt y su perceptrón, un modelo compuesto por una red neuronal de una sola capa, siendo prometedor pero avanzado a sus tiempos por la falta de poder de cálculo computacional, causando una disminución generalizada del interés en las redes neuronales durante los años siguientes.

La popularización de las redes neuronales tal y como las conocemos en la actualidad fue iniciada durante los últimos años del siglo XX, cuando se desarrollaron nuevas técnicas de entrenamiento y arquitecturas más profundas. Un hito que marco un antes y un después fue el descubrimiento del algoritmo de retropropagación con el cual existía la posibilidad de entrenar redes neuronales multicapa de manera escalable, impulsando la eficacia de las redes neuronales en la resolución de problemas como el reconocimiento de patrones o el procesamiento del lenguaje natural (Hinton, G. et al., 1986).

Posteriormente, el desarrollo de los modelos de redes neuronales se produjo en sincronía con el aumento significativo en la capacidad de procesamiento computacional y el acceso a grandes cantidades de datos, ambas características siendo elementos esenciales para entrenar redes neuronales más grandes y profundas, lo que sigue llevando hoy en día a mejoras en el rendimiento y la capacidad de generalización de esta alternativa de modelización.

El concepto de redes neuronales se fundamenta en la simulación computacional de las redes neuronales biológicas presentes en el cerebro humano, buscando imitar su capacidad de aprendizaje y resolución de problemas complejos a partir de ejemplos previamente percibidos. Técnicamente, las redes neuronales están compuestas por algoritmos de aprendizaje automático que consisten en unidades interconectadas conocidas como neuronas o nodos. Estas neuronas funcionan de manera similar a las células nerviosas del cerebro y trabajan en conjunto para abordar una tarea específica. Cada neurona recibe entradas, las procesa mediante una función de activación y produce una salida que puede ser transmitida a otras neuronas. A través de la combinación y modificación de los pesos de las conexiones entre las neuronas, las redes neuronales aprenden a reconocer patrones y realizar predicciones.

Las neuronas que forman el modelo están organizadas en capas, donde cada neurona se conecta con las neuronas de la capa siguiente a través de conexiones ponderadas. La primera es la capa de entrada, donde se introducen los datos seleccionados para su entrenamiento. Le siguen las capas intermedias, conocidas también como capas ocultas, donde se procesa y transforma la información introducida en la capa de entrada. Finalmente, la última es llamada capa de salida, cuya finalidad es la de proporcionar los resultados finales. Esta estructura en capas permite que las redes neuronales capturen patrones complejos y realicen predicciones precisas. Para una mayor comprensión, la estructura del modelo es graficada en la Figura 4 mostrada a continuación.

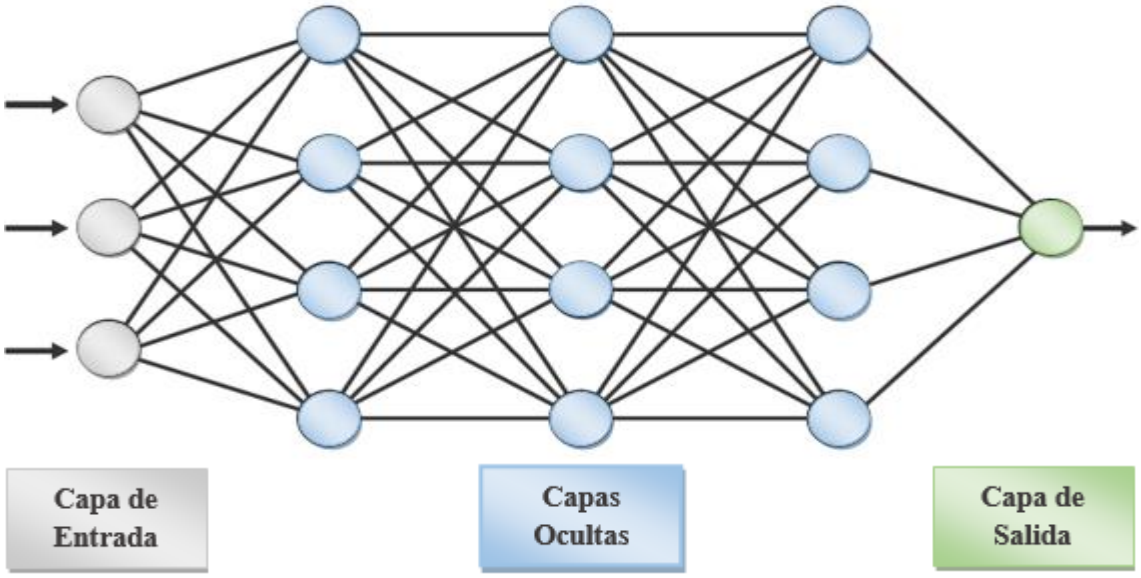

Figura 4. Esquema de la Estructura de Redes Neuronales. Elaboración Propia (2023)

Cabe remarcar que existen distintas alternativas para modelizar mediante redes neuronales, siendo cada una de ellas diseñadas para abordar diferentes tipos de problemas y tareas de aprendizaje. Las redes neuronales de alimentación directa son las más clásicas, ya que la información fluye desde las capas de entrada a través de una serie de capas ocultas hasta llegar a la capa de salida, sin que haya retroalimentación. En relación a la modelización predictiva destaca el clasificador de perceptrón multicapa, conocido como MLP por sus iniciales en inglés, el cual será el elegido para el análisis empírico desarrollado ulteriormente al ser caracterizado por su notable capacidad para aprender y modelizar tanto relaciones como patrones en los datos utilizando múltiples capas ocultas y funciones de activación no lineales para procesar la información.

Dentro de las alternativas más recientes destacan las redes neuronales recurrentes donde las conexiones entre neuronas forman ciclos, permitiendo que la información fluya hacia adelante y hacia atrás, habilitando la modelización de secuencias y datos con dependencias temporales como lenguaje natural o el reconocimiento de voz. Alternativamente, las redes neuronales tienen la capacidad de trabajar con datos estructurados en forma de grafos, donde los nodos representan entidades y las aristas representan relaciones entre ellas. La elección entre alternativas dependerá de las características de los datos y de la finalidad de la modelización.

A pesar de sus numerosas fortalezas, el modelo de redes neuronales también presenta algunas limitaciones. Entre ellas, destacan la necesidad de una gran cantidad de datos para su entrenamiento y los respectivos requisitos computacionales necesarios para volúmenes grandes de información. Además, la interpretación de sus resultados presenta una dificultad mucho mayor que las demás alternativas, ya que las redes neuronales no proporcionan una explicación clara de cómo generaron cada predicción. Sin embargo, actualmente ya se está trabajando en el cálculo de cómo cambios pequeños en los valores de entrada afectan a los resultados producidos por la red neuronal, una métrica de sensibilidad que ayudaría notoriamente a comprender mejor su comportamiento. En conclusión, a pesar de sus limitaciones marcadas por el elevado volumen de datos necesarios y su escasa interpretabilidad, los modelos basados en redes neuronales son excelentes herramientas para llevar a cabo la modelización y la predicción de patrones complejos en bases de datos, destacando por su versatilidad ante numerosos escenarios heterogéneos junto con un excelente desempeño robusto y preciso.

II.2.5) Voting Classifier

El modelo de Voting Classifier es una técnica de ensamble popularizada a partir de la década de los noventa por una gran variedad de autores. Su metodología combina múltiples modelos predictivos con el objetivo de realizar una predicción conjunta. Cada uno de los modelos genera de manera individual su propio resultado y, posteriormente, es aplicada una regla de votación para determinar la predicción final del Voting Classifier. Esta técnica se basa en la idea de heterogeneidad en las fortalezas y debilidades de los distintos modelos de clasificación, combinando sus predicciones para así obtener una clasificación resultante más robusta y precisa (Bartlett, P. et al, 1997).

Existen dos enfoques principales para el Voting Classifier: el Hard Voting y el Soft Voting. En el primero, cada modelo individual emite una única predicción y la clase que obtiene la mayoría de votos es seleccionada como la predicción final. Alternativamente, en el Soft Voting cada modelo individual emite una estimación de probabilidad para cada una de las clases de la variable respuesta, generando la predicción final al seleccionar la clase con mayor de probabilidad tras promediar los modelos individuales.

En la práctica, el Voting Classifier puede ser implementado con dos algoritmos del mismo modelo pero con diferentes hiperparámetros, por ejemplo, en árboles de decisión con distinto nivel de profundidad máxima. Sin embargo, habitualmente suele implementarse agregando distintos algoritmos de aprendizaje supervisado como es mostrado en la Figura 5, esquematizando la estructura de esta técnica de ensamble.

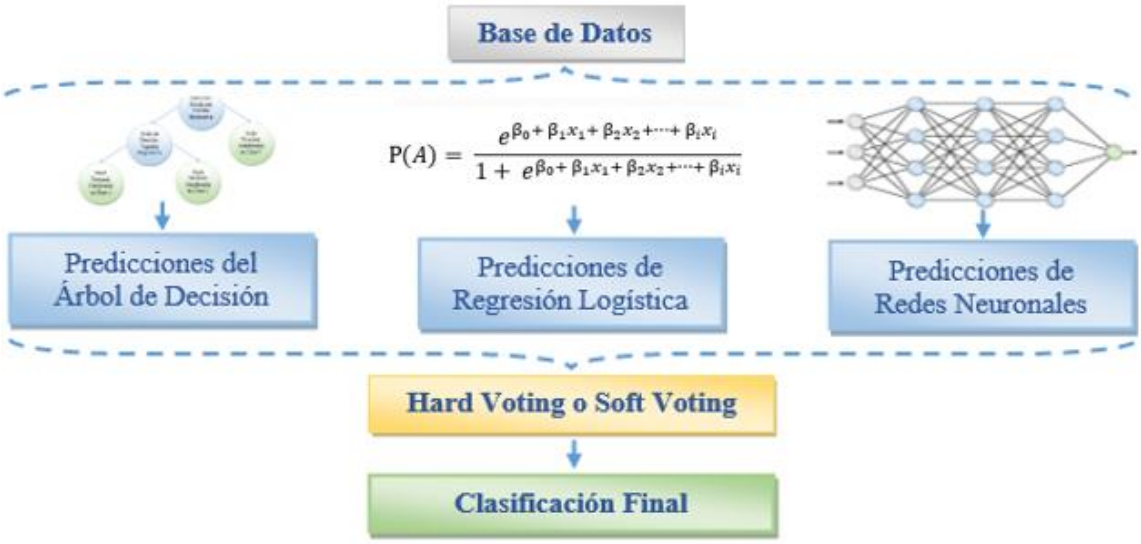

Figura 5. Esquema de la Estructura del Voting Classifier. Elaboración Propia (2023)

En términos generales, el Voting Classifier es un algoritmo de aprendizaje automático que combina el resultado de múltiples modelos de clasificación calculados individualmente, obteniendo así una clasificación final basada en la votación de los distintos modelos, donde se pretende que las debilidades de unos se compensen con las fortalezas de otros. A pesar de que su mayor inconveniente se centra en la necesidad de poder computacional suficiente para el cálculo de múltiples modelos, con el tiempo que ello comporta, este enfoque es de gran utilidad cuando un conjunto de datos complejos que no puede ser modelizado adecuadamente con un solo clasificador, convirtiendo al Voting Classifier en una alternativa robusta y precisa.

II.2.6) Stacking Classifier

El modelo de Stacking Classifier es una técnica de ensamble propuesta por el matemático David Hilton Wolpert a través de su paper 'Stacked Generalization', publicado en el año 1992. De la misma manera que el Voting Classifier detallado anteriormente, el enfoque de este modelo se basa la combinación de múltiples modelos predictivos para mejorar su precisión. Sin embargo, a diferencia del Voting Classifier, el Stacking Classifier incorpora un modelo adicional de nivel superior que combina las predicciones de los modelos base, también conocidos como clasificadores de nivel inferior. De esta manera, el Stacking Classifier utiliza tanto las predicciones individuales de los clasificadores base como las características de los datos de entrada para realizar una predicción final más precisa y robusta.

El proceso de entrenamiento del Stacking Classifier consta de dos etapas principales. Inicialmente, los clasificadores de nivel inferior son entrenados paralelamente de manera individual, utilizando la información procedente de la base de datos, generando cada clasificador de nivel inferior sus propias predicciones para las distintas clases de la variable respuesta. Seguidamente, estas predicciones son utilizadas como características de entrada para el clasificador de nivel superior, al que también se le conoce bajo el nombre de meta-clasificador, el cual se entrena con el objetivo de combinar las predicciones de los clasificadores de nivel inferior, generando así la predicción final.

Una de las mayores ventajas del Stacking Classifier es que las debilidades de ciertos modelos base son compensadas con las fortalezas de otros y viceversa, provocando una mejora significativa en la precisión y en la capacidad de generalización del modelo en comparación con los clasificadores base tratados individualmente. Al combinar las predicciones de múltiples modelos, el Stacking Classifier tiene la capacidad de capturar patrones más complejos y sutiles en los datos, lo que resulta en una mayor capacidad de discriminación y una reducción en el sesgo de las predicciones. Además, al incorporar un modelo de nivel superior que aprende a combinar los resultados de los modelos base, el Stacking Classifier tiene la capacidad de adaptarse y ajustarse mejor a los datos, lo que lo convierte en un modelo final más robusto y generalizable.

A pesar de sus ventajas, el Stacking Classifier también presenta algunas limitaciones a considerar. La incorporación de un nivel superior de modelización añade complejidad al proceso de entrenamiento, por lo que son requeridos más recursos computacionales y también un plazo temporal más elevado. A su vez, este aumento de complejidad provoca que el Stacking Classifier convierta modelos individualmente interpretables en un modelo agregado cuya interpretación es muchísimo más compleja.

Adicionalmente, es importante tener en cuenta que la elección de los modelos añadidos al Stacking Classifier es crucial para su rendimiento, dando prioridad a la selección de modelos que sean lo más heterogéneos posible en términos de sus características y comportamiento, para así poder beneficiarse del poder combinatorio. En la Figura 6 se esquematiza un Stacking Classifier formado por árboles de decisión, regresión logística y redes neuronales como modelos base, combinándolos en un nivel superior con Gradient Boosting como modelo meta-clasificador.

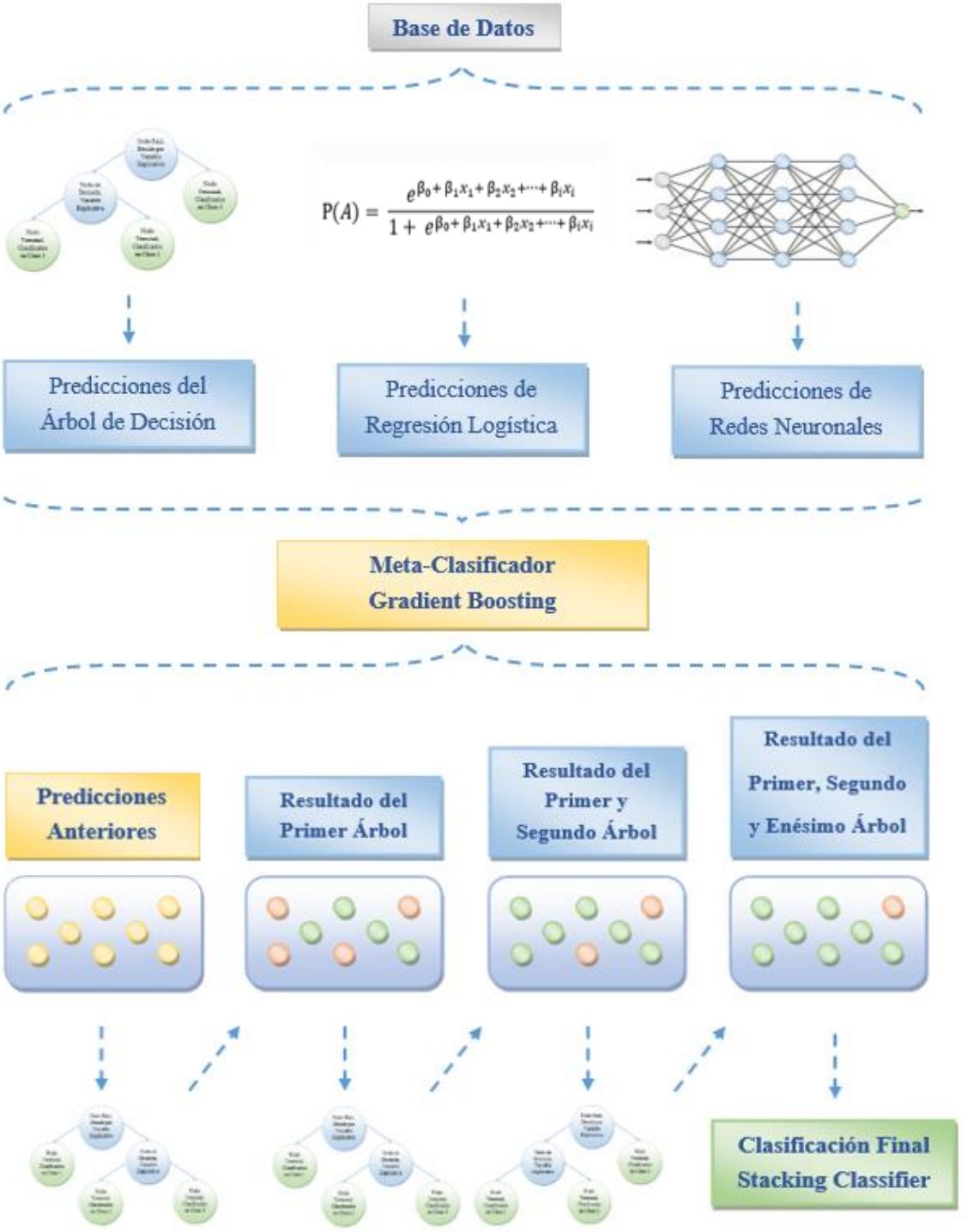

Figura 6. Esquema de la Estructura del Stacking Classifier. Elaboración Propia (2023)

En términos generales, el Stacking Classifier es una técnica de ensamble que combina múltiples modelos de clasificación para mejorar la precisión de las predicciones. Su metodología en dos etapas, utilizando clasificadores de nivel inferior y un clasificador de nivel superior, permite aprovechar las fortalezas individuales de cada uno de los modelos. Si se seleccionan y configuran adecuadamente, el Stacking Classifier puede convertirse en una técnica muy potente y optimizada para mejorar la precisión y la robustez de los modelos predictivos de clasificación.

III) ANÁLISIS EMPÍRICO

III.1) Presentación de los Objetivos

Con la finalidad de complementar cuantitativamente los aspectos teóricos detallados para cada modelo en los apartados anteriores y añadir valor empírico al presente trabajo, seguidamente será elaborado un análisis basado en la modelización predictiva para la clasificación del riesgo de cada uno de los individuos que conforman una cartera de seguros de vida.

El objetivo principal del mismo consiste en la evaluación del impacto de la disrupción tecnológica sobre la modelización de los seguros de vida mediante una comparativa de las técnicas econométricas tradicionales versus las herramientas de aprendizaje automático, analizando y evaluando el rendimiento de las distintas alternativas mediante métricas de precisión y de bondad de ajuste en base a sus resultados. Adicionalmente, serán identificadas las fortalezas y las limitaciones de cada una de las técnicas de modelización empleadas, interpretando los resultados generados y discutiendo sus implicaciones hacia la segmentación de los individuos que conforman la cartera analizada, descrita con detalle en el siguiente apartado.

En relación a la aplicación e interpretación de los modelos, como técnicas de modelización econométricas tradicionales serán empleados los modelos de Regresión Logística, Análisis Discriminante Lineal, Naive Bayes, y Árboles de Decisión. En cuanto a la modelización basada en Machine Learning, serán utilizados los modelos de Random Forest, Gradient Boosting, XGBoost, Redes Neuronales, Voting Classifier y Stacking Classifier. Para cada una de las técnicas predictivas de segmentación, será analizado su procedimiento de cálculo enfatizando en la interpretabilidad de sus resutados, aplicando las metodologías teóricas descritas anteriormente.

Tras llevar a cabo la modelización, la precisión de cada alternativa será evaluada a través del análisis de la Curva ROC, acrónimo de *Receiver Operating Characteristic*, empleando la metodología 'One vs All' caracterizada por evaluar cada clase objetivo frente a la combinación del resto de las clases, el enfoque adecuado al tratarse de un análisis clasificatorio con variable respuesta multinomial.

Complementariamente, también serán calculadas otras métricas como el F-Score, la precisión, la exactitud, la exhaustividad, y la matriz de confusión de cada uno de los modelos. La evaluación del rendimiento será valorada sobre el subconjunto estratificados de test, evaluando cada uno de los modelos con datos que no han sido utilizados en el su proceso de entrenamiento.

En definitiva, el presente análisis empírico tiene como objetivo contribuir a la comprensión del impacto de la disrupción tecnológica en la práctica aseguradora, enfocándose específicamente en la modelización de los seguros de vida, con el fin de ofrecer nuevas perspectivas marcando el camino hacia la optimización de la segmentación predictiva en procedimientos tan importantes como la selección de cartera y la fijación de primas adecuadas al nivel de riesgo de cada póliza.

III.2) Descripción de la Base de Datos

La base de datos[2] sobre la que se realiza el presente análisis empírico fue compartida públicamente en el año 2016 por la aseguradora de origen estadounidense Prudential Financial Incorporated, conocida de manera popular simplemente bajo el nombre de Prudential y caracterizada por su logo azul en el que figura el peñón de Gibraltar como símbolo de fortaleza, protección y seguridad tras la resistencia de la pequeña península ante el Gran Asedio sufrido a finales del siglo XVIII (Prudential Financial, 2023).

Tras su fundación en el año 1875 por John Fairfield Dryden en el Estado de Nueva Jersey, su expansión no ha cesado durante los más de 140 años dedicados al sector financiero y a los seguros de vida, operando actualmente en veintitrés países distribuidos por América, Asia y el continente europeo. Actualmente es una de las quinientas empresas incluidas en el índice bursátil Standard & Poor's 500, cotizando en el New York Stock Exchange con una capitalización bursátil superior a los treinta mil millones de dólares al cierre del ejercicio del año 2022 (Yahoo Finance, 2023). Su línea de negocio se encuentra diversificada en dos ramas principales: los servicios financieros de asesoramiento e inversión, y la comercialización de seguros de salud y de vida.

Dentro de esta última división de negocio nace la intención de hacer accesible al público general una base de datos donde son recogidas más de cien variables explicativas que describen los atributos de los solicitantes de sus seguros de vida, es decir, de cada uno de los individuos que conforman la muestra (Kaggle, 2023). Seguidamente será analizada en detalle, ya que es fundamental que los datos objeto de estudio sean relevantes, consistentes, precisos y representativos poblacionalmente porque la calidad de los resultados generados y de las conclusiones subsiguientes dependen directamente de la calidad de la base de datos.

Es importante remarcar que la información empleada ha sido debidamente anonimizada por Prudential antes de hacerla accesible al público, contando únicamente con una variable llamada ID como identificador individual. En la sociedad actual, la protección de los datos personales se ha convertido en una cuestión crucial, especialmente en un contexto en el que la tecnología y la digitalización han aumentado exponencialmente la cantidad y la complejidad de los datos personales que manejan las empresas.

En cuanto al resto de variables, ha sido llevada a cabo una normalización de sus valores transformándolos hacia una escala común con el objetivo de poder realizar comparativas y análisis más efectivos, reduciendo el impacto de las distintas escalas de valoración o de unidades de medida heterogéneas, asegurando que todos los datos de una misma variable estén expresados en una misma escala y, consecuentemente, cuenten con valor comparativo. Es de esencial importancia tener en cuenta que la mencionada normalización de variables en ningún caso impacta en la información que puede ser extraída de la muestra, simplemente la información es estandarizada sin alterar su representación fiel de los datos originales.

En consecuencia, tras aplicar la técnica de anonimizado mediante la variable ID y la normalización del resto de variables evitando la identificación individual de sujetos que cumplan ciertas características, es generada una base de datos que garantiza la protección de la privacidad de los datos personales, preservando su confidencialidad.

---

[2] Kaggle. (2016). Prudential Life Insurance Dataset.
https://www.kaggle.com/competitions/prudential-life-insurance-assessment/data

A modo descriptivo, seguidamente es definida la variable respuesta junto al resto de variables explicativas de la base de datos:

- **''Response''**: variable respuesta, de tipología cualitativa ordinal, cuyo espacio muestral está formado por los números enteros desde el 1 hasta el 8, distintos niveles que cuantifican de menor a mayor el riesgo asociado a cada uno de los individuos de la muestra.

- **''ID''**: variable numérica discreta que actúa como índice y otorga un identificador único para cada individuo que forma parte de la muestra.

- **''Product_Info_1-7''**: conjunto de variables normalizadas que se relacionan con la tipología y características del seguro de vida contratado. Todas ellas son de tipología categórica nominal, exceptuando "Product_Info_4" al ser una variable numérica continua.

- **''Ins_Age''**: variable numérica continua que determina la edad normalizada de cada individuo en el momento de la solicitud.

- **''Ht''**: variable numérica continua que recoge la altura individual normalizada.

- **''Wt''**: variable numérica continua que recoge el peso individual normalizado.

- **''BMI''**: variable numérica continua que recoge el índice de masa corporal normalizado de cada individuo.

- **''Employment_Info_1-6''**: conjunto de variables normalizadas que proporcionan información sobre el historial laboral de cada uno de los individuos. "Employment_Info_1,4,6" son de tipología numérica continua, mientras que "Employment_Info_2,3,5" son de tipología categórica nominal.

- **''InsuredInfo_1-7''**: conjunto de variables categóricas nominales normalizadas que proporcionan información personal sobre cada individuo.

- **''Insurance_History_1-9''**: conjunto de variables normalizadas sobre el historial asegurador del individuo. Todas son de tipología categórica nominal, exceptuando "Insurance_History_5" al ser una variable numérica continua.

- **''Family_Hist_1-5''**: conjunto de variables normalizadas relacionadas con el historial familiar de cada individuo. Todas ellas son de tipología numérica continua, exceptuando "Family_Hist_1" al ser una variable categórica nominal.

- **''Medical_History_1-41''**: conjunto de variables normalizadas relacionadas con el historial médico de cada individuo. Todas ellas son de tipología categórica nominal, exceptuando "Medical_History_1,10,15,24,32" al ser variables numéricas discretas.

- **''Medical_Keyword_1-48''**: conjunto de variables binarias ficticias o *dummy* que indican la presencia o ausencia de una palabra clave médica específica en el formulario de solicitud del seguro de vida elaborado por cada individuo.

En su totalidad, la base de datos analizada consta de 59.381 observaciones de individuos donde cada uno de ellos cuenta con un número ID de identificación único, 126 variables explicativas y una variable respuesta. Esta última, "Response", es una medida cualitativa ordinal de riesgo que cuenta con 8 niveles distintos, empezando por el nivel 1 adjudicado a los individuos cuyo riesgo asociado es menor y prosiguiendo de manera creciente hasta el nivel 8 que categoriza el segmento con mayor riesgo asociado. Por consiguiente, "Response" es la variable que será utilizada para evaluar el rendimiento y bondad de ajuste de los modelos de clasificación, cuya distribución es graficada a continuación.

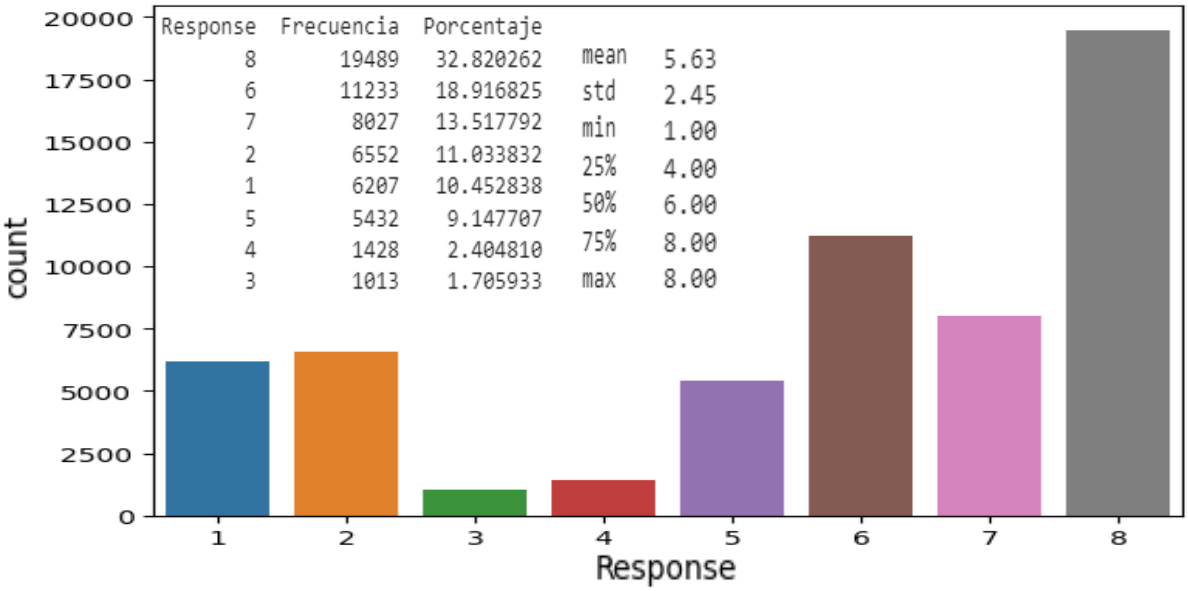

Figura 7. Gráfico de la Distribución y Estadísticos Básicos de la Variable Respuesta. Elaboración Propia (2023)

Visualmente, en la distribución de la variable respuesta destaca su desequilibrio al ser notoriamente sesgada hacia los segmentos cuyo número es más elevado, destacando el octavo segmento que es el que concentra mayor número de individuos. Esta característica es confirmada numéricamente mediante el valor de la mediana o percentil del 50%, ya que el hecho de que su valor para la variable respuesta sea ubicada en el sexto segmento significa que la mitad de las observaciones se encuentran distribuidas en los segmentos 1 al 6, por lo que la otra mitad se encuentra concentrada únicamente en los segmentos 6 al 8.

A pesar de ser una variable cualitativa, su característica de segmentación ordinal hace que pueda ser coherente el análisis de la asimetría de su distribución, siendo negativa hacia la izquierda además de contar con una media superior a la mediana a causa del gran número de observaciones pertenecientes al segmento 8 juntamente con el 6 y el 7, los tres segmentos que alojan el mayor número de individuos en ese mismo orden.

Por otro lado, también hay una proporción de datos notable en los segmentos 2 y 1, dejando a los segmentos 3 y 4 seguidos por el 5 como los que concentran un menor número de individuos. En los próximos apartados del presente análisis deberá ser valorada la precisión del ajuste de los distintos modelos empleados para esta distribución en concreto, generando así un indicador clave para cuantificar el rendimiento de las distintas técnicas de modelización.

Prosiguiendo con la descripción de la base de datos es generada la matriz de correlaciones entre variables, la cual permite entender la interrelación de las mismas cuantificando la magnitud y dirección de la relación lineal entre pares de variables.

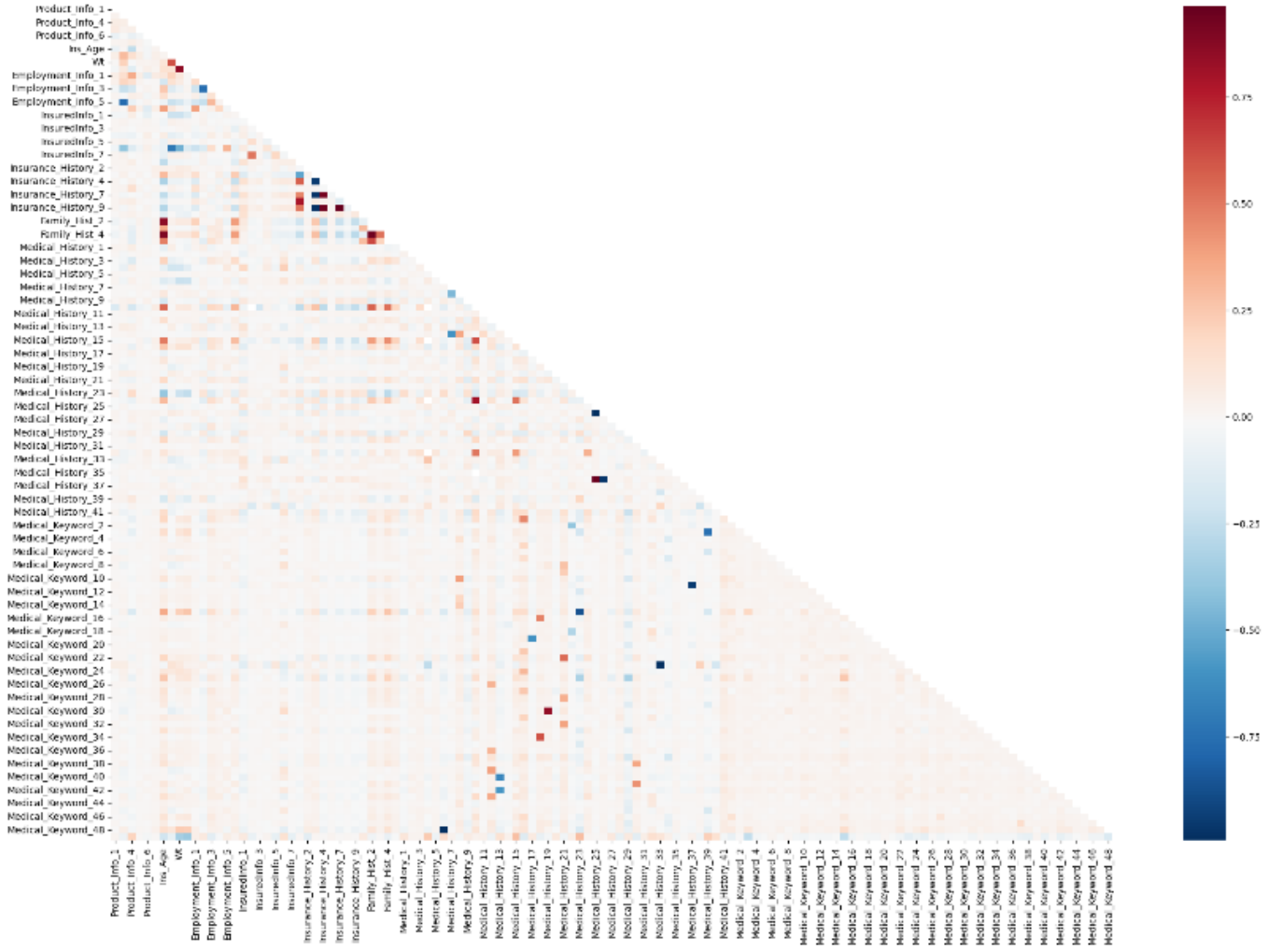

Figura 8. Gráfico de la Matriz de Correlaciones. Elaboración Propia (2023)

Como puede ser observado, las correlaciones con un mayor valor absoluto se encuentran entre las propias variables relacionadas con el historial asegurador juntamente con variables del historial médico y del historial familiar, existiendo una elevada correlación entre las variables que forman las tres categorías mencionadas. Alternativamente, existe poca correlación notoria entre variables de distintas categorías.

La identificación de variables altamente correlacionadas es un paso esencial previo a la modelización de los datos, ya que una correlación cercana a 1 tanto positiva como negativa es significativa y puede indicar que las variables están midiendo la misma característica del individuo y, por lo tanto, es necesario considerar cuidadosamente su inclusión en el modelo. Puede ser beneficioso realizar técnicas como la reducción de dimensionalidad o la selección de características para evitar problemas de multicolinealidad y mejorar la interpretación y eficiencia del modelo, las cuales serán aplicadas en el siguiente apartado.

Complementariamente, en la Figura 9 son presentados los binomios de variables cuya correlación es superior a 0,9 en valor absoluto, denotando una asociación notoria entre ellas al existir significancia correlativa. Un ejemplo de alta correlación en la base de datos analizada es el valor de +0,962528 entre Insurance_History_7 e Insurance_History_9, lo que sugiere una relación muy fuerte entre estas dos variables. También hay varias correlaciones negativas muy elevadas, como el valor de -0.993101 entre las variables Medical_History_6 y Medical_History_48. lo que sugiere una relación inversa muy significativa.

Variable 1	Variable 2	Correlación
Insurance_History_7	Insurance_History_9	0.962528
Medical_History_25	Medical_History_36	0.954110
Insurance_History_4	Insurance_History_9	0.938654
Family_Hist_2	Family_Hist_4	0.934615
Insurance_History_4	Insurance_History_7	0.919526
Insurance_History_3	Insurance_History_4	-0.949388
Medical_History_37	Medical_Keyword_11	-0.950069
Medical_History_26	Medical_History_36	-0.965349
Insurance_History_3	Insurance_History_7	-0.974910
Insurance_History_3	Insurance_History_9	-0.982598
Medical_History_25	Medical_History_26	-0.987910
Medical_History_33	Medical_Keyword_23	-0.993030
Medical_History_6	Medical_Keyword_48	-0.993101

Figura 9. Pares de Variables cuya Correlación es Superior a 0,9 en Valor Absoluto. Elaboración Propia (2023)

Adicionalmente es llevada a cabo la comprobación de valores faltantes para cada una de las variables de la base de datos, un paso de notoria importancia debido a que pueden afectar a la precisión y validez de los resultados, conduciendo a conclusiones sesgadas o inexactas en caso de modelizar mediante variables con un porcentaje muy elevado de valores faltantes. Como puede ser observado, en la Figura 10 son listadas las variables con una proporción de observaciones faltantes superior al 50%, es decir, más de la mitad de los individuos no cuentan con valores para las variables en cuestión. En el caso de Medical_History_10 faltan el 99,06% de los datos, denotando un poder estadístico extremadamente limitado, por lo que posteriormente serán eliminadas las variables con más del 50% de datos faltantes. Alternativamente, aquellas variables con menor porcentaje sí que podrían ser adecuadas para el análisis, por lo que sus valores faltantes deberán ser imputados mediante técnicas estadísticas.

```
Medical_History_10: 58824 (99.06%)
Medical_History_32: 58274 (98.14%)
Medical_History_24: 55580 (93.60%)
Medical_History_15: 44596 (75.10%)
Family_Hist_5: 41811 (70.41%)
Family_Hist_3: 34241 (57.66%)
```

Figura 10. Conteo y Porcentaje en Variables con Más del 50% de Valores Faltantes. Elaboración Propia (2023)

Finalmente, tras describir e interpretar exhaustivamente la base de datos, ha sido detectada la existencia de correlaciones muy significativas y proporciones elevadas de valores faltantes en ciertas variables. En consecuencia, es generada la necesidad de tratar los datos adecuadamente, por lo que en el apartado ulterior se procederá a la adaptación de la base de datos para evitar sesgos en los análisis posteriores.

III.3) Metodología

La metodología aplicada para elaborar el presente estudio consta de una serie de pasos cuyo objetivo es asegurar la calidad de los datos para desarrollar adecuadamente y de manera precisa los distintos modelos predictivos de clasificación. En consecuencia, la calidad de los datos es optimizada prosiguiendo con un proceso de depuración y transformación para asegurar la integridad de la base de datos.

Inicialmente, es llevado a cabo el tratamiento de aquellos pares de variables cuya correlación sea mayor a 0,9 en valor absoluto debido a los posibles perjuicios causados en el análisis de datos por la presencia de multicolinealidad. Para evitar que los binomios de variables de la Figura 9 proporcionen información similar o redundante al modelo, son eliminadas las siguientes variables explicativas: 'Insurance_History_3,4,9', 'Family_Hist_4', 'Medical_Keyword_11,23,48', y 'Medical_History_26,36'. Tras esta modificación, la tabla de pares de variables con correlación mayor a 0,9 en valor absoluto queda vacía, beneficiando así a la interrelación de las 117 variable explicativas restantes junto a la variable respuesta.

Posteriormente, también son eliminadas las variables cuyo porcentaje de datos faltantes sea superior al 50%, mostradas en la Figura 10. Este paso también es fundamental, ya que modelizar con variables cuya proporción de datos faltantes sea alta puede sesgar las estimaciones paramétricas y las inferencias obtenidas a partir de los datos, conduciendo directamente hacia conclusiones incorrectas y poco confiables. En consecuencia, son eliminadas de la base de datos las variables 'Medical_History_10,15,24,32' y 'Family_Hist_3,5', corroborando la desaparición de variables que les falten más de la mitad de sus valores entre las 111 variables explicativas restantes que serán usadas para predecir la variable respuesta.

En el resto de variables explicativas también hay algunas que cuentan con un cierto número de valores faltantes, aunque siempre menor al 50% establecido como punto de criba, como se muestra en la Figura 11.

```
La proporción de Valores Faltantes de la Variable Family_Hist_2 es del: 48.26 %.
La proporción de Valores Faltantes de la Variable Insurance_History_5 es del: 42.77 %.
La proporción de Valores Faltantes de la Variable Employment_Info_6 es del: 18.28 %.
La proporción de Valores Faltantes de la Variable Medical_History_1 es del: 14.97 %.
La proporción de Valores Faltantes de la Variable Employment_Info_4 es del: 11.42 %.
La proporción de Valores Faltantes de la Variable Employment_Info_1 es del: 0.03 %.
```

Figura 11. Porcentaje de Valores Faltantes en cada Variable. Elaboración Propia (2023)

Con el fin de garantizar la compatibilidad de la base de datos con la modelización a través del aprendizaje automático, preservando sus características estructurales y el valor informativo de las variables, se implementa un procedimiento de imputación iterativa. Éste consiste en llevar a cabo la estimación de los valores faltantes de las variables a partir de la información que proporcionan los datos disponibles, aplicando técnicas de regresión para predecir los valores desconocidos. Finalmente, la secuencia de predicciones es repetida múltiples veces hasta alcanzar una estimación completa y fiable, lo que da como resultado una base de datos exenta de valores faltantes.

Prosiguiendo con la adaptación, la variable alfanumérica Product_Info_2 es sustituida por una versión transformada de la misma pero convertida a numérica mediante One-Hot Encoding. Este proceso consiste en representar cada categoría única de la variable con una columna binaria separada, asignando el valor 1 a la presencia de esa categoría y el valor 0 a las demás categorías. El One-Hot Encoding permite que el modelo pueda interpretar y utilizar de manera efectiva la información categórica de la variable en el proceso de entrenamiento, ya que transforma las categorías en una representación numérica que puede ser fácilmente interpretada por los algoritmos de aprendizaje automático. Esto permite evitar problemas ocasionados por los caracteres alfabéticos en la modelización.

En el resto de variables se lleva a cabo un procedimiento de normalización por mínimos y máximos, el cual consiste en escalar los valores de cada variable para que se encuentren dentro del rango de la unidad, y cuya finalidad es la de asegurar que todas las variables tengan un peso similar y no se vean afectadas por la escala original en la que se encuentren. Al normalizar los datos, se facilita la comparación y el análisis de las variables, lo que permite realizar un tratamiento estadístico más preciso.

Tras a la eliminación de correlaciones significativas entre variables y de aquellas con gran proporción de datos faltantes, juntamente con la adaptación del resto de variables, la base de datos analizada empíricamente queda formada por la variable índice ID para un total de 111 variables explicativas sin ningún valor faltante juntamente con la variable respuesta detallada anteriormente, cuyo objetivo es la clasificación en ocho segmentos de distinto riesgo.

Ulteriormente, una vez se dispone de los datos correctamente adaptados y listos para modelizar, el último paso antes de calibrar los modelos consiste en dividir la base de datos en dos subconjuntos catalogados como Entrenamiento y Test, con el 80% y el 20% del total de individuos respectivamente, generando la oportunidad de evaluar la capacidad predictiva de cada alternativa de modelización en datos que no han sido utilizados en el proceso de entrenamiento del modelo cuyo desempeño y capacidad predictiva son puestos a prueba.

En pocas palabras, el subconjunto de Entrenamiento será el empleado para ajustar y validar los modelos de clasificación, mientras que el subconjunto de Test será el utilizado para evaluar el rendimiento de los modelos con datos que no se han utilizado en el proceso de entrenamiento. Esta característica ayuda a evitar el sobreajuste que ocurre cuando el modelo se ajusta demasiado a los datos del subconjunto de Entrenamiento y no puede generalizarse para otras observaciones, un efecto significativamente indeseado en la modelización al disminuir el rendimiento en caso de aplicarse sobre nuevos datos.

Adicionalmente, es imprescindible que ambos subconjuntos de datos estén estratificados ya que deben garantizar una representación adecuada y proporcionada de las características relevantes de la base de datos original. En la Figura 12 es mostrada la distribución de la variable respuesta en el subconjunto de Entrenamiento formado por el 80% de los datos, mientras que en la Figura 13 es mostrada la distribución de la variable respuesta esta vez en el subconjunto de Test donde son incluidos el 20% restante de individuos.

Response	Frecuencia	Porcentaje
8	15631	32.904598
6	9027	19.002610
7	6419	13.512546
2	5226	11.001179
1	4927	10.371758
5	4310	9.072920
4	1145	2.410323
3	819	1.724065

Figura 12. Distribución de Variable Respuesta en la Muestra de Entrenamiento. Elaboración Propia (2023)

Response	Frecuencia	Porcentaje
8	3858	32.482950
6	2206	18.573714
7	1608	13.538772
2	1326	11.164435
1	1280	10.777132
5	1122	9.446830
4	283	2.382757
3	194	1.633409

Figura 13. Distribución de Variable Respuesta en la Muestra de Test. Elaboración Propia (2023)

Como puede ser observado, la estratificación entre los subconjuntos de Entrenamiento y Test es claramente homogénea en relación a la distribución de la variable respuesta utilizada para evaluar el rendimiento y bondad de ajuste de los modelos de clasificación, manteniendo un gran número de observaciones pertenecientes al segmento 8 juntamente con el 6 y el 7, siguiendo con una proporción de datos menor pero notable en los segmentos 2 y 1, y dejando a los segmentos 3 y 4 seguidos por el 5 como los que concentran un menor número de individuos, por lo que puede afirmarse que ambos subconjuntos representan la base de datos original de manera adecuada y proporcionada.

Finalmente, como el número de observaciones es muy heterogéneo en cada uno de los ocho segmentos de la variable respuesta, es establecido un vector que contiene la ponderación del número de observaciones en cada segmento de la muestra de entrenamiento. Estas ponderaciones se asignan con el objetivo de garantizar una representación adecuada y equilibrada de todas las categorías de la variable respuesta durante el proceso de entrenamiento del modelo, ajustando la importancia relativa de cada observación en función de su segmento correspondiente.

Al establecer un vector de ponderaciones, se contrarresta el impacto desproporcionado que tendría el segmento 8 debido a su mayor número de observaciones, permitiendo así que los demás segmentos también tengan una influencia significativa en el proceso de entrenamiento. De esta manera, se logra evitar sesgos garantizando que el modelo aprenda de manera equitativa y precisa en relación a todos los subconjuntos.

A fin de cuentas, la base de datos descrita y adaptada será el sustento de la modelización elaborada a continuación con el objetivo de predecir su variable respuesta para clasificar en ocho grupos de distinto nivel de riesgo a cada uno de los individuos que forman parte de la misma, habilitando la posibilidad de comparar el rendimiento de los modelos econométricos frente a los basados en Machine Learning mediante distintas métricas para cuantificar su rendimiento y precisión.

III.4) Resultados

A lo largo del presente apartado serán detallados los resultados obtenidos en el análisis empírico sobre el impacto de la disrupción tecnológica en la modelización de los seguros de vida, llevado a cabo mediante los cuatro modelos econométricos tradicionales y los seis basados en Machine Learning que han sido presentados en el apartado de Fundamentos Teóricos. Todos ellos han sido entrenados a partir de la base de datos adaptada anteriormente y en el lenguaje de programación Python[3], siendo descrito en detalle el entorno de programación en el Anexo I y cuya secuencia de comandos se encuentra en las 1.788 filas del Anexo II, ambos al final presente informe.

Cabe remarcar que para mejorar los resultados de cada uno de los modelos ha sido elaborado un procedimiento de optimización mediante hiperparámetros, el cual consiste en realizar un cálculo exhaustivo de estos valores mediante la técnica de validación cruzada[4], evaluando el desempeño del modelo resultante de cada combinación de hiperparámetros y seleccionando las mejores opciones para el entrenamiento definitivo del modelo optimizado. Además, todos son entrenados de manera equitativa para los distintos segmentos de la variable respuesta, ya que durante el ajuste de los modelos es tomado en consideración el vector de ponderaciones mencionado anteriormente.

Con la finalidad de evaluar el rendimiento y la precisión de las distintas alternativas de modelización, han sido calculadas métricas como el F-Score, la precisión, la exactitud, la exhaustividad, y la matriz de confusión. Sin embargo, la comparativa será focalizada sobre la valoración e interpretación de la Curva ROC en su versión One vs All y del subsiguiente área bajo la curva ROC, al ser ampliamente considerada como una de las mejores técnicas para la evaluación del rendimiento en modelos predictivos de clasificación multinomial.

La Curva ROC, traducida del término anglosajón Receiver Operating Characteristic Curve, es una métrica caracterizada por representar gráficamente la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos de las predicciones de un modelo en concreto, es decir, la relación entre la sensibilidad y la unidad menos la especificidad de cada técnica de modelización.

A diferencia de la Curva ROC tradicional que se utiliza para problemas de clasificación binaria, cuando se trata de un problema de clasificación multinomial como es el caso presente, el enfoque One vs All habilita la construcción de una Curva ROC que evalúa cada conjunto de la variable respuesta individualmente comparándolo con el resto, entrenando un clasificador binario para cada segmento donde el de interés se considera como positivo y el resto se agrupan como negativos.

Finalmente, cada punto en la Curva ROC One vs All representa el desempeño del clasificador para una determinada sensibilidad y especificidad. Al unir todos sus puntos, se obtiene una curva que permite visualizar y comparar el rendimiento de los diferentes modelos clasificadores en términos de su capacidad para predecir correctamente la clasificación del segmento de interés frente a los demás, llevando a cabo este procedimiento para cada uno de los segmentos de la variable respuesta.

---

[3] Python. (2023). About Python Programing Language. *https://www.python.org/about/*

[4] Método que consiste en evaluar y validar un modelo utilizando múltiples subdivisiones del conjunto de datos, permitiendo una estimación más precisa de su rendimiento y reduciendo el riesgo de sobreajuste

Tanto la sensibilidad como la especificidad son métricas clave en la evaluación de la capacidad discriminativa del modelo, ya que la primera cuantifica su capacidad para identificar correctamente las muestras positivas de un segmento determinado, mientras que la segunda cuantifica la capacidad para clasificar correctamente las muestras negativas en esa misma clase. Adicionalmente, la Curva ROC que será calculada sobre los datos de Test, no solo permite evaluar el rendimiento global del modelo mediante la media del ajuste de sus distintos segmentos, sino que también permite identificar posibles desequilibrios en la clasificación de segmentos específicos.

El área bajo la curva ROC, habitualmente conocida como AUC-ROC por sus iniciales en inglés, es una medida numérica que cuantifica la información contenida en la Curva ROC One vs All, al ser calculada mediante la integral del área bajo la curva, lo que permite obtener un valor que representa la capacidad discriminativa del clasificador.

El valor de la métrica AUC-ROC varía en un rango entre 0 y 1, donde un valor de 0 significaría que el modelo ha clasificado incorrectamente todas las observaciones, mientras que un valor de 0,5 corresponde a un clasificador con un rendimiento aleatorio, y un valor de 1 correspondería a un modelo perfecto, cuyas predicciones fueran siempre certeras. Cuanto mayor sea el área bajo la curva, mejor será el rendimiento del modelo analizado en términos de su capacidad para distinguir correctamente los segmentos de interés del resto.

En el presente análisis multinomial, el valor del AUC-ROC será calculado para cada segmento en comparación con el resto, posteriormente calculando la media de las ocho áreas bajo la curva para obtener así un valor que representa el desempeño promedio de las predicciones de clasificación generadas por el modelo predictivo multinomial analizado, obteniendo una visión general del rendimiento del modelo que permite compararlo con el resto de alternativas de modelización.

Adicionalmente, para cada uno de los diez modelos también será calculada su matriz de confusión multinomial, la cual se compone de filas donde está representado el segmento real de cada individuo y columnas donde se representa el segmento en el que las predicciones del modelo clasifican a ese mismo individuo. En consecuencia, es otra alternativa para obtener una representación de las observaciones clasificadas correctamente en forma de verdaderos positivos y de las clasificadas incorrectamente en forma de falsos positivos y falsos negativos.

La matriz de confusión[5] deberá ser calculada en su opción ponderada, ya que en una muestra sesgada hacia ciertos segmentos como es este caso, es crucial ponderarla para realizar una evaluación justa del rendimiento del modelo de clasificación, evitando conclusiones sesgadas por el desequilibrio en la distribución de la variable respuesta.

De la misma manera que con la Curva ROC One vs All, la matriz de confusión multinomial permite obtener información detallada sobre el rendimiento del modelo en la clasificación de cada segmento, esta vez de forma matricial en vez de gráfica. Sin embargo, la matriz permite evaluar hacia qué segmentos en concreto se desvían los errores de clasificación, habilitando la posibilidad de sugerir mejoras y ajustes en las técnicas empleadas. Por esta razón, la matriz de confusión será calculada en los diez modelos y detallada en los que su información sea más significativa para la comprensión de la modelización.

[5] La matriz de confusión consiste en una tabla cuyo eje vertical representa las categorías reales de los datos, mientras que el eje horizontal representa las categorías predichas por el modelo de clasificación.

Empezando por los modelos econométricos tradicionales, inicialmente es valorado el rendimiento del modelo de Regresión Logística, graficando su Curva ROC en la Figura 14. Como puede ser observado, las predicciones relativas a los segmentos 4, 8 y 3 destacan por su excelente rendimiento, todas ellas cercanas al valor 0,9 de AUC-ROC. Mientras que el segmento 5 queda cercano a la media, el modelo sufre en relación a las predicciones del resto de segmentos, sobre todo destacando el sexto segmento con un área bajo la curva únicamente de 0,68, como se puede observar con la línea rosa.

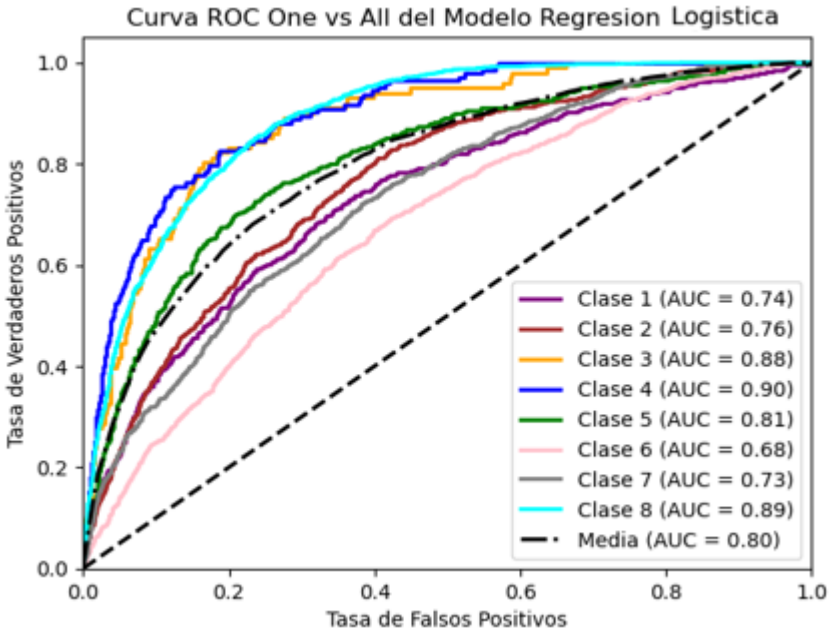

Figura 14. Curva ROC One vs All del Modelo de Regresión Logística sobre los datos de Test. Elaboración Propia (2023)

En este caso, la matriz de confusión del modelo de regresión logística corrobora las conclusiones del gráfico de la Curva ROC, aunque se observa como el predictor clasifica a muy pocas observaciones en la clase 6, de ahí su bajo nivel de verdaderos positivos. La diagonal de la matriz indica los casos donde la predicción es acertada, siendo positivo que resalte su color en relación al resto de celdas en la Figura 15.

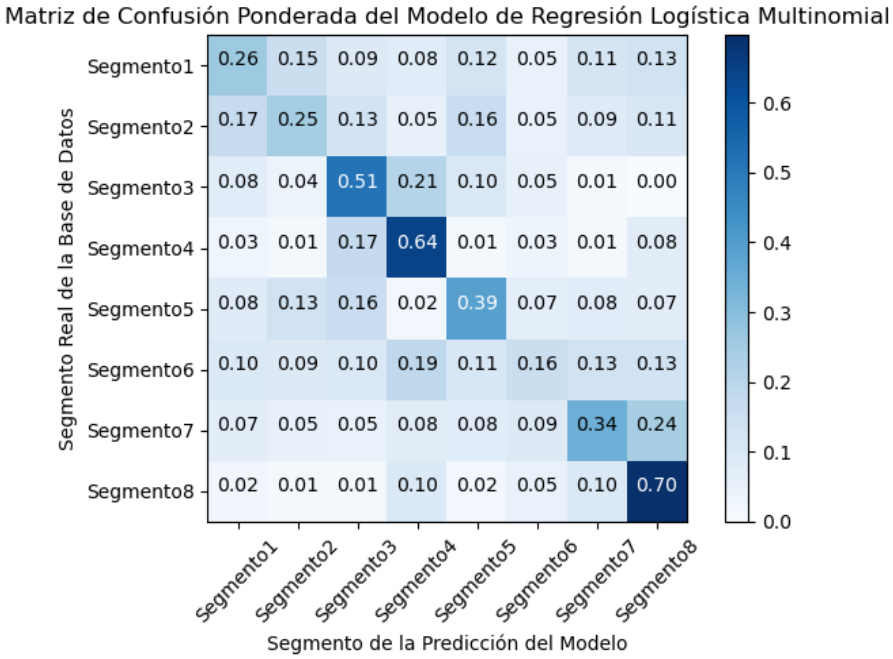

Figura 15. Matriz de Confusión del Modelo de Regresión Logística sobre los datos de Test. Elaboración Propia (2023)

Seguidamente, el modelo de Análisis Discriminante Lineal en la Figura 16 presenta un área bajo la curva con exactamente la misma media que el modelo de Regresión Logística, sin embargo, mejora el ajuste del segmento 6 a costa de ser ligeramente menos preciso en el resto de segmentos, resultando en una alternativa de modelización ligeramente más equilibrada a pesar de ser igual de precisa a nivel general.

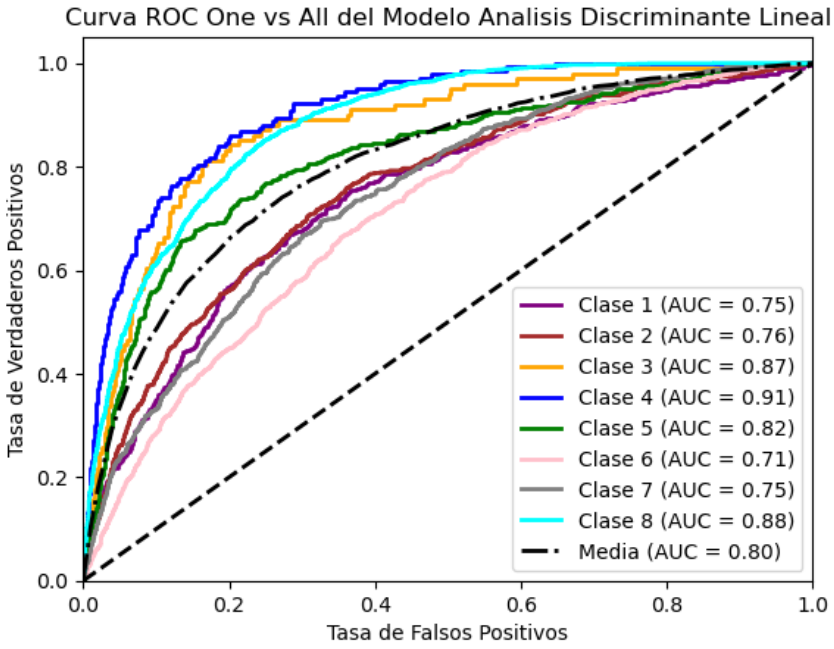

Figura 16. Curva ROC One vs All del Modelo de Análisis Discriminante Lineal sobre los datos de Test. Elaboración Propia (2023)

El modelo de Naive Bayes es, con diferencia, la alternativa que peor se ha adaptado a la base de datos analizada. La media de su área bajo la curva se sitúa en tan solo 0,74 conllevando un numero de predicciones incorrectas bastante mayor que el resto de modelos. Sin embargo, cabe remarcar que es el modelo cuyo cálculo ha resultado más rápido, tardando apenas segundos gracias a sus procedimientos sencillos en comparación con el resto de alternativas, como fue detallado en el apartado de fundamentos teóricos.

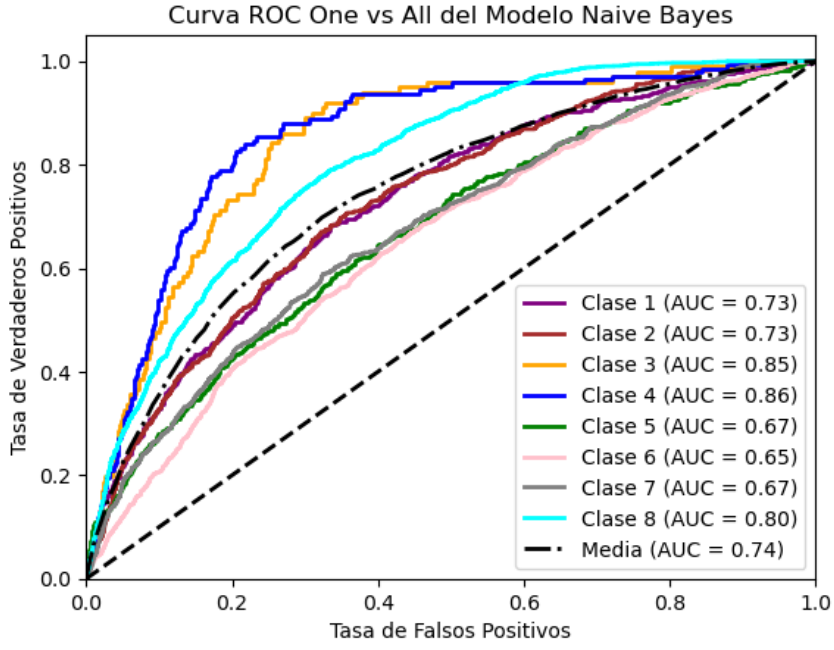

Figura 17. Curva ROC One vs All del Modelo de Naive Bayes sobre los datos de Test. Elaboración Propia (2023)

La cuarta alternativa de los modelos econométricos tradicionales son los Árboles de Decisión, cuya media de área bajo la Curva ROC de 0,76 es la segunda menor de las diez alternativas consideradas para la modelización. También es destacable su velocidad de cálculo, demorándose menos de un minuto para el entrenamiento del modelo optimizado mediante distintas alternativas de hiperparámetros.

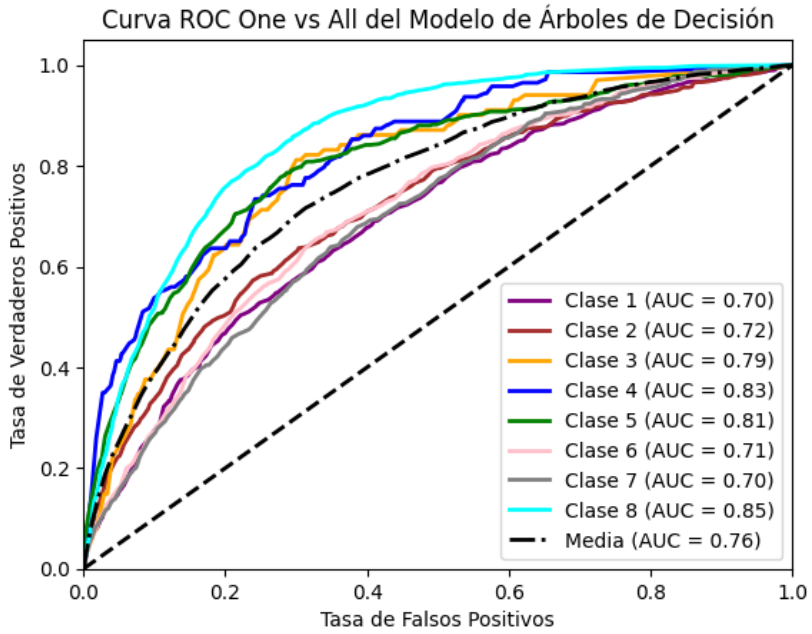

Figura 18. Curva ROC One vs All del Modelo de Árboles de Decisión sobre los datos de Test. Elaboración Propia (2023)

En lo referente a la modelización mediante técnicas basadas en Machine Learning, el primer modelo evaluado es el de Random Forest. En este ya puede ser apreciado un salto cualitativo notorio, con una media de 0,84 para el área bajo la curva y con cuatro clases cercanas al 0,9 que indica predicciones sobresalientes. Además, es destacable que la mejor área bajo la curva para la clase 6 en los modelos econométricos era únicamente de 0,71, siendo ahora superado hasta el valor de 0,78 mostrado en la Figura 19.

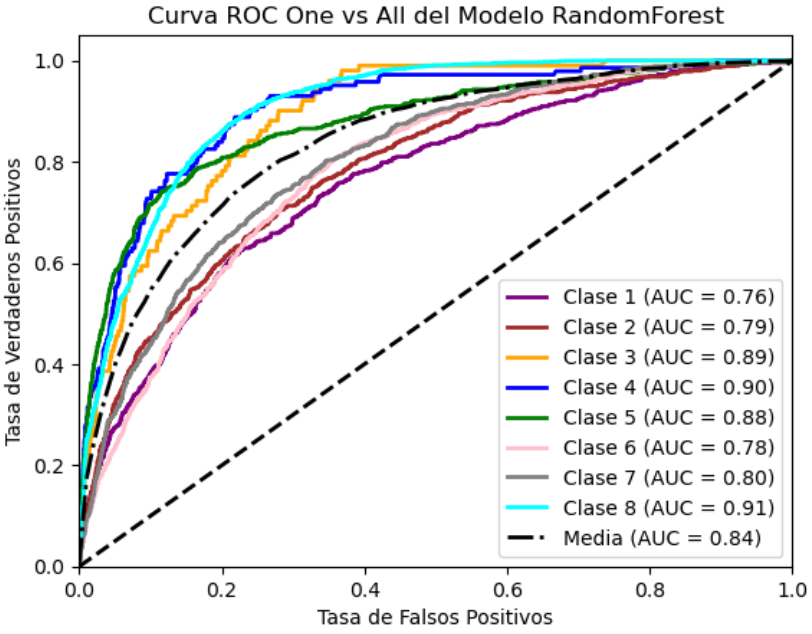

Figura 19. Curva ROC One vs All del Modelo de Random Forest sobre los datos de Test. Elaboración Propia (2023)

La segunda alternativa de aprendizaje automático considerada es el modelo de Gradient Boosting, cuya media de 0,84 en relación al área bajo la curva iguala al modelo de Random Forest. Sin embargo, con un valor de AUC mínimo de 0,77 en el segmento 1 y máximo de 0,92 en el segmento 4, Gradient Boosting consigue una mayor robustez y estabilidad que cualquiera de las opciones anteriores, mostrándolo en la Figura 20.

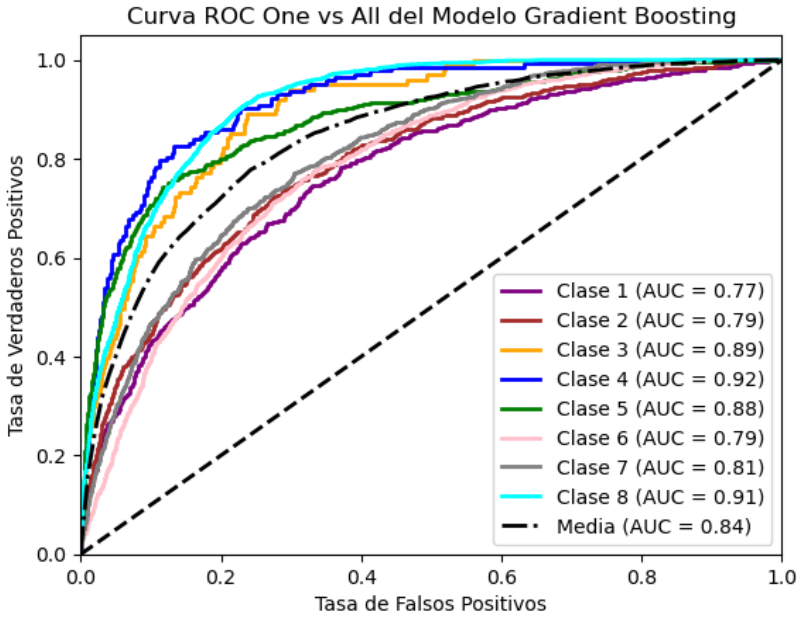

Figura 20. Curva ROC One vs All del Modelo de Gradient Boosting sobre los datos de Test. Elaboración Propia (2023)

Como ha sido mencionado en los apartados anteriores, el modelo de XGBoost es una técnica de modelización que pretende optimizar el modelo de Gradient Boosting, por lo que se esperaría que su rendimiento predictivo fuese aún mejor que el anterior. Efectivamente, XGBoost es el mejor modelo individual de los considerados en el informe, con un área bajo la curva ROC promediada en 0,85 y con un excelente rendimiento al optimizar ligeramente el modelo en el que se basa, mostrando en la Figura 21 sus excelentes resultados en las predicciones de clasificación por segmentos.

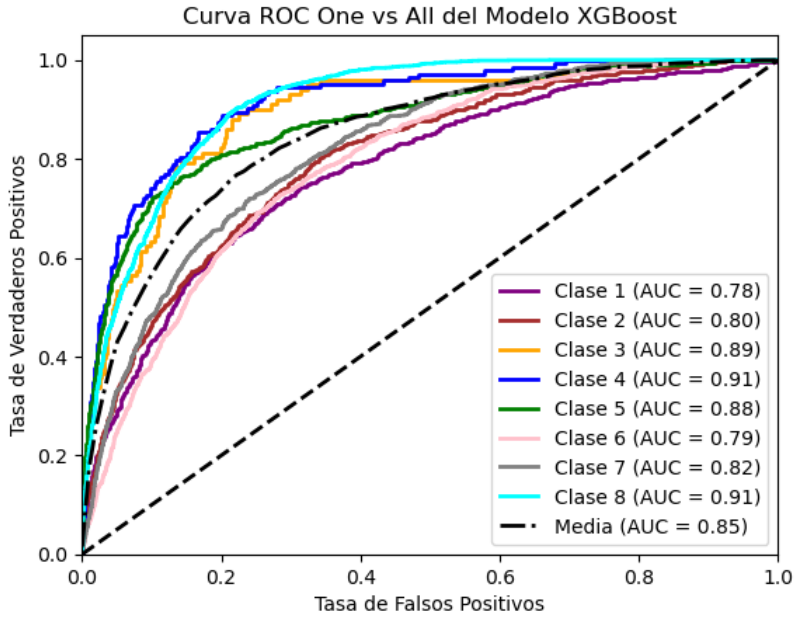

Figura 21. Curva ROC One vs All del Modelo de XGBoost sobre los datos de Test. Elaboración Propia (2023)

En cuanto al modelo de Redes Neuronales, concretamente el de perceptrón multicapa (MLP), cabe remarcar que incluso siendo la alternativa de aprendizaje automático con peor rendimiento, es más preciso que cualquiera de los modelos econométricas tradicionales analizados. Su área bajo la curva se sitúa en un valor promedio de 0,82, destacando su capacidad de predicción casi tan buena como los mejores modelos en relación a los segmentos 4 y 8, pero sin embargo, cuenta con grandes limitaciones para predecir el segmento 1, marcado en la Figura 22 de color rosa.

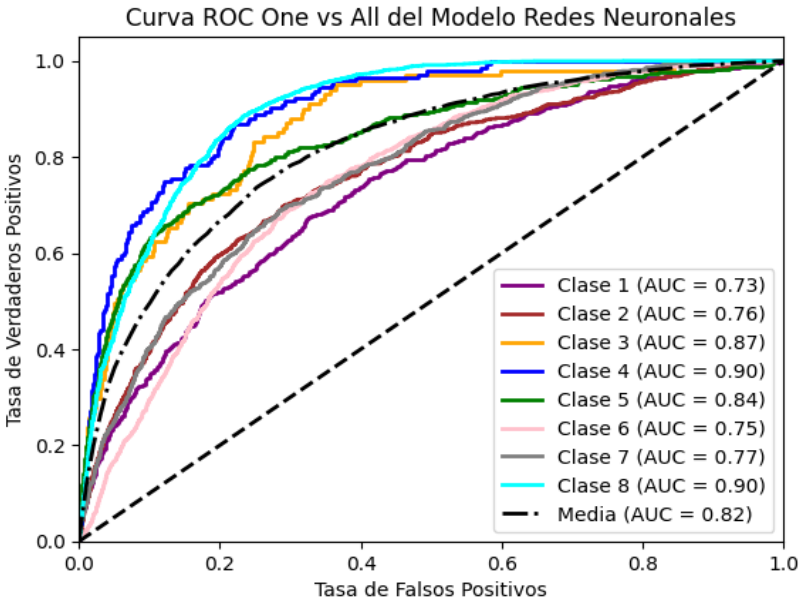

Figura 22. Curva ROC One vs All del Modelo de Redes Neuronales sobre los datos de Test. Elaboración Propia (2023)

La valoración de los modelos de ensamble es una de las claves del presente informe. En relación a un modelo Voting Classifier formado por la mejor alternativa econométrica tradicional y la mejor basada en Machine Learning, es decir, el modelo de Regresión Logística y el XGBoost, se obtiene una alternativa de modelización cuyo rendimiento es aún mejor que el de ambos modelos por separado, como puede ser observado en la Figura 23. Es el único modelo en el que todos sus segmentos tienen un área bajo la curva superior al valor de 0,8 además de que tres de ellos superan el 0,9 de AUC-ROC.

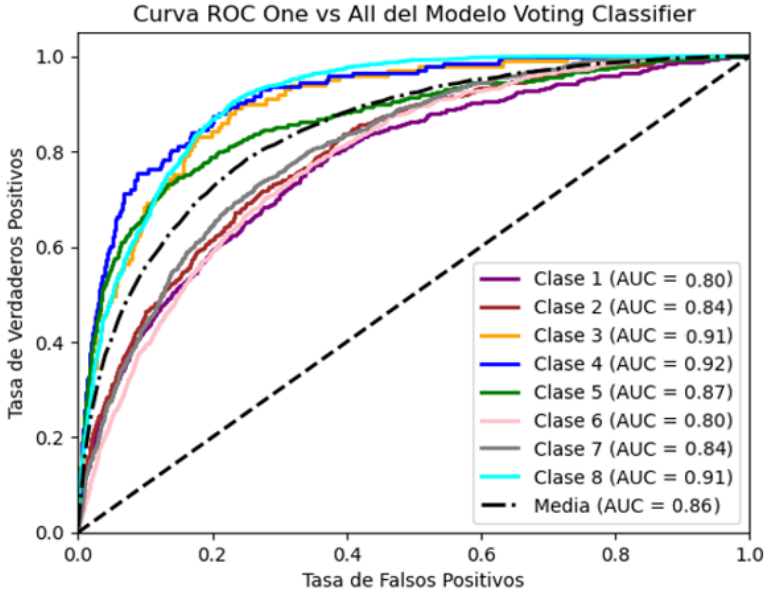

Figura 23. Curva ROC One vs All del Modelo de Voting Classifier sobre los datos de Test. Elaboración Propia (2023)

El modelo de Stacking Classifier seleccionado es construido mediante el modelo XGBoost como base y el de Regresión Logística como meta-clasificador, mostrando su resultado en la Figura 24. El rendimiento de esta técnica de ensamble es sobresaliente en comparación con el resto de modelos, con un área bajo la curva ROC cuyo promedio es cuantificado en 0,86, coincidiendo con el resultado del Voting Classifier y que les sitúa a ambos modelos de ensamble como las mejores alternativas para la modelización clasificatoria de la cartera de seguros de vida.

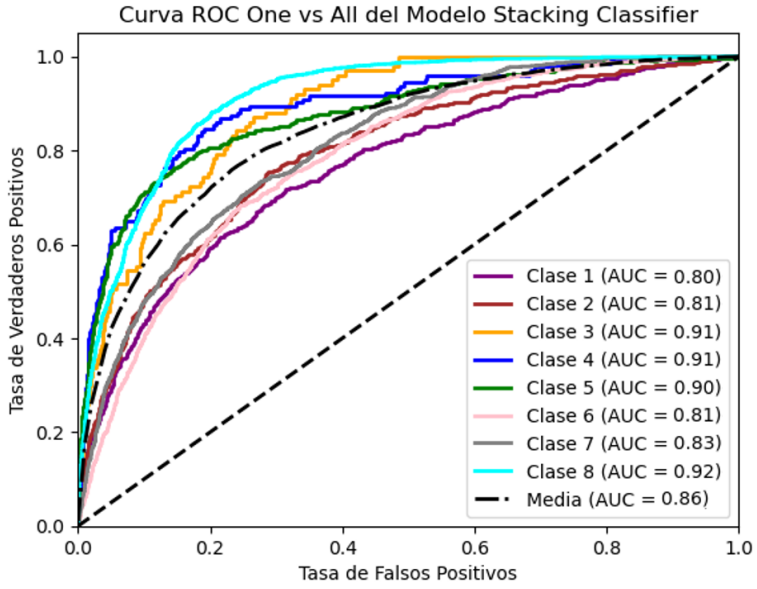

Figura 24. Curva ROC One vs All del Modelo de Stacking Classifier sobre los datos de Test. Elaboración Propia (2023)

Cabe mencionar que durante el procedimiento de modelización han sido calculadas métricas de rendimiento como el F-Score, la precisión, la exactitud, y la exhaustividad para cada uno de los modelos. Sin embargo, sus conclusiones iban en plena concordancia con las generadas mediante la Curva ROC y su AUC, por lo que no han sido mostradas todas las métricas calculadas para no caer en redundancia informativa. Finalmente, a modo recopilatorio es generada una Curva ROC compuesta por el valor promedio del rendimiento demostrado por cada uno de los modelos, juntamente con una leyenda en la que se especifica el área bajo la curva de cada alternativa, consiguiendo el objetivo de graficar y cuantificar el impacto de la disrupción tecnológica en la modelización de los seguros de vida, detallando sus conclusiones en el apartado ulterior.

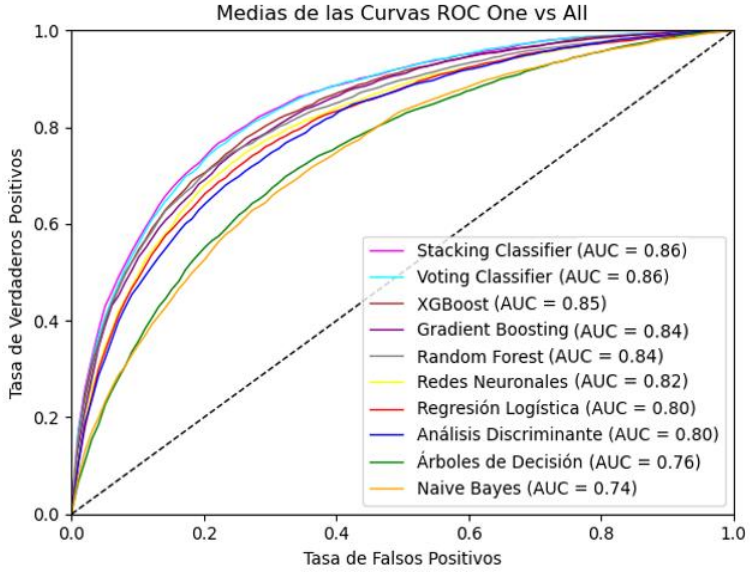

Figura 25. Curva ROC del Promedio de los Segmentos de Cada Modelo sobre los datos de Test. Elaboración Propia (2023)

IV) CONCLUSIONES Y RECOMENDACIONES

A lo largo del presente informe ha sido llevado a cabo un análisis exhaustivo de dos enfoques distintos para la modelización de los seguros de vida: los modelos econométricos tradicionales y los modelos computacionales basados en las disruptivas técnicas de Machine Learning. A través de la presentación de los fundamentos teóricos que consolidan los cimientos de cada modelo y un análisis empírico aplicado a una cartera real de seguros de vida, ha sido perseguido el objetivo de determinar cuál de ambos enfoques es el más efectivo en términos de rendimiento y precisión para la industria aseguradora de vida.

Las conclusiones generadas a partir del análisis de los fundamentos teóricos y los resultados obtenidos mediante en análisis empírico revelan que los modelos basados en el aprendizaje automático son inherentemente más potentes que los modelos econométricos tradicionales, los cuales cuentan con una estructuración más sencilla. Los primeros cuentan con la capacidad de capturar relaciones complejas y no lineales en las bases de datos, lo que les confiere una ventaja en términos de precisión predictiva, además de que han demostrado su eficacia en diversos contextos presentando una mayor flexibilidad y generalización hacia nuevos datos sin perjuicios en la captura de patrones mediante los que obtener información para la modelización.

Sin embargo, es de notoria importancia destacar que la complejidad asociada a estos modelos plantea desafíos considerables, como es el caso de la necesidad de una mayor capacidad computacional, lo cual implica mayores requisitos de hardware y tiempo de cálculo, con el sobrecoste que pueden producir ambas condiciones en el ámbito laboral. Durante el desarrollo analítico fueron anotados los tiempos de optimización hiperparamétrica y generación de predicciones para cada una de las diez alternativas de modelización consideradas: en el caso de los modelos econométricos tradicionales, las cuatro alternativas fueron calculadas en menos de diez minutos; sin embargo, los tiempos de procesamiento en los modelos basados en Machine Learning fueron considerablemente superiores, con técnicas como el Gradient Boosting o el XGBoost cuya modelización se demoró de manera ininterrumpida durante varios días.

Otro de los aspectos clave que debe tomarse en consideración es la interpretabilidad de los resultados y la escalabilidad de los modelos, ambos conceptos delimitados por la estructura de cada método de cálculo. Las nuevas técnicas basados en el aprendizaje automático tienden a ser más difíciles de interpretar debido a su complejidad y falta de transparencia en la toma de decisiones, las cuales son tomadas por algoritmos que incluyen modelos matemáticos altamente sofisticados y difícilmente comprensibles.

Mientras que los modelos econométricos tradicionales cuentan con una escalabilidad bastante limitada, esta misma característica es una de las mayores ventajas competitivas de las técnicas innovadoras. Este es un aspecto clave teniendo en cuenta que la tendencia del sector asegurador se dirige hacia un escenario en el que la disponibilidad de datos para modelizar será cada vez mayor, enriqueciendo las bases de datos tanto verticalmente como horizontalmente, es decir, aumentando en su número de observaciones y en el número de variables disponibles para cada individuo.

El término anglosajón Big Data, referido al uso masivo de datos, está destinado a ser uno de los grandes componentes innovadores en el futuro próximo, donde la escalabilidad de la modelización será un enorme rasgo diferencial en el que deberían centrarse los profesionales del sector asegurador. Sin embargo, en un paradigma social donde la preocupación en relación a la protección de datos va en aumento, será un gran reto la implementación de medidas para evitar tanto la discriminación algorítmica como el asedio hacia los clientes con el objetivo de conseguir nuevas variables predictoras, por ejemplo, información sensible sobre su estilo de vida, sus perfiles de redes sociales o el seguimiento de su ubicación geográfica sin el consentimiento expreso de la persona.

Es crucial establecer salvaguardias sólidas que protejan la privacidad y los derechos de los asegurados, promoviendo una transparencia total en el uso de datos y garantizando que las primas sean determinadas de manera justa y equitativa, basadas únicamente en información relevante y no sesgada. En consecuencia, de cara al futuro será primordial que las aplicaciones de los algoritmos en el sector asegurador siempre mantengan un enfoque ético y legalmente responsable.

En última instancia, puede concluirse que si bien es un hecho que los modelos basados en técnicas de Machine Learning son inherentemente más potentes y superan en poder predictivo a los modelos econométricos tradicionales, la mejor alternativa es tomar en consideración los modelos de ensamble que combinen ambas técnicas, como es el caso del Voting Classifier o el Stacking Classifier, los cuales cuentan con la gran ventaja de compensar las debilidades de ciertos modelos con las fortalezas de otros, provocando una mejora significativa en la precisión y en la capacidad de generalización del modelo en comparación con los modelos generados individualmente.

Al combinar múltiples predicciones obtenidas mediante técnicas heterogéneas entre sí, los modelos de ensamble tienen la capacidad de capturar patrones más complejos y sutiles en los datos, lo que resulta en una mayor capacidad de discriminación y una reducción sustancial en el sesgo de las predicciones, afianzándoles como una técnica de modelización robusta y precisa. A medida que avanza la disrupción tecnológica, es esencial seguir explorando nuevas metodologías y enfoques que permitan mejorar la capacidad predictiva y la eficiencia de los modelos en pro de afrontar los retos venideros en el campo de la ciencia de los datos.

A fin de cuentas, en base a las conclusiones generadas a partir del estudio de fundamentos teóricos de las distintas alternativas de modelización juntamente con los resultados obtenidos mediante el análisis empírico, el presente informe demuestra que la disrupción tecnológica ha tenido un impacto notorio en la modelización de los seguros de vida, respaldando la superioridad predictiva de los modelos basados en Machine Learning, pero también tomando en consideración factores tan relevantes como los requisitos de poder computacional junto con el tiempo dedicado a la modelización y la interpretabilidad de los resultados, donde la sencillez de los modelos econométricos tradicionales se convierte en su mayor riqueza.

La modelización mediante técnicas de ensamble que combinen la econometría tradicional con técnicas disruptivas basadas en Machine Learning se posiciona hoy en día como la mejor alternativa para aprovechar al máximo las cualidades y ventajas de ambos enfoques, brindando un equilibrio excepcional en el binomio formado por la precisión y la eficiencia, dos cualidades clave en la modelización de los seguros de vida.

ANEXO I – DESCRIPCIÓN DEL ENTORNO DE PROGRAMACIÓN

Como ha sido mencionado de manera reiterativa, la modelización mediante técnicas de Machine Learning lleva asociada de manera inherente una estructuración compleja y la necesidad de un poder computacional notorio, por lo que se deberá prestar especial atención en el momento de escoger un entorno de programación que se adecúe a las necesidades de cada caso.

Con la finalidad de llevar a cabo el análisis empírico desarrollado en el presente informe, el lenguaje de programación considerado como la opción más adecuada es Python, el cual se caracteriza por ser un software de código abierto cuya popularidad se debe a su excelente versatilidad, siendo usado en campos tan heterogéneos como la inteligencia artificial y el desarrollo web, entre otros. Fue presentado en el año 1991 por el informático neerlandés Guido van Rossum, el cual perseguía el objetivo de crear una sintaxis de programación cuya escritura fuese similar a la de las lenguas derivadas de alfabeto romano, como el inglés o el castellano. En consecuencia, el código programado en Python puede ser leído con un elevado nivel de interpretabilidad.

Dado que el análisis empírico de la base de datos de Prudential requiere de un poder computacional elevado, tanto por su dimensionalidad como por la complejidad de las técnicas de modelización aplicadas, la programación en Python ha sido llevada a cabo mediante Google Colaboratory[6], también conocido simplemente como Colab. Éste es un entorno de programación de código abierto de gran utilidad al estar basado en servicios 'Cloud Computing', un término anglosajón empleado para describir los servicios de computación en la nube, es decir, usando servidores ajenos al ordenador del usuario.

El 'Cloud Computing' proporciona una ventaja muy remarcable para la modelización basada en Machine Learning, ya que Google Colaboratory ofrece un cuaderno en el que se introduce el código programado, y al ejecutarlo se procesa en servidores de Google, por lo que permite realizar cálculos intensivos y acelerar el tiempo de ejecución de los programas sin la necesidad de invertir en ordenadores potentes, sin tener que realizar instalaciones ni configuraciones en los equipos locales, y pudiendo acceder a los códigos desde cualquier lugar del mundo que cuente con conexión a Internet. Además, este entorno de programación ofrece la posibilidad de integrarlo con herramientas como Google Drive, lo cual facilita la carga y el almacenamiento de las bases de datos.

Adicionalmente, para llevar a cabo la modelización del análisis empírico han sido empleadas bibliotecas de Python, las cuales son módulos de código predefinido que proporcionan funcionalidades adicionales para facilitar tareas específicas en la programación, como la implementación de algoritmos. En el presente estudio, las principales bibliotecas empleadas han sido Pandas para estructurar los datos de cara a su análisis, Numpy para facilitar los cálculos numéricos y las operaciones matriciales, Matplotlib y Seaborn para la visualización de los resultados en gráficos, Statsmodels para la estimación de modelos econométricos tradicionales, y Scikit-Learn para la modelización mediante algoritmos de aprendizaje automático.

En última instancia, cabe concluir que la utilización del lenguaje de programación Python en conjunto con las bibliotecas mencionadas y en los servidores de Google Colaboratory, confiere al presente estudio un entorno de trabajo sumamente robusto y eficiente, al mismo tiempo que optimiza de manera significativa su productividad.

---

[6] Google. (2023). Google Colaboratory. *https://colab.research.google.com/*

ANEXO II - PROGRAMACIÓN EN PYTHON: SECUENCIA DE COMANDOS

```
############################## TRABAJO FINAL DE MÁSTER ##############################
######################### UNIVERSITAT DE BARCELONA - UB #########################
################### MÁSTER CIENCIAS ACTUARIALES Y FINANCIERAS ###################
############## RUBÉN GARCÍA TALLANTE - NIUB 21116001 - CURSO 2022/2023 ###########

#################### ECONOMETRÍA VERSUS MACHINE LEARNING: ####################
# IMPACTO DE LA DISRUPCIÓN TECNOLÓGICA EN LA MODELIZACIÓN DE LOS SEGUROS DE VIDA #

#### Importación de Libterías Requeridas para el Desarrollo del Análisis Empírico ####

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

import xgboost as xgb

import graphviz

import pydotplus

import itertools

import time

from google.colab import drive

from google.colab import data_table
```

```
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.experimental import enable_iterative_imputer

from sklearn.impute import IterativeImputer

from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

from sklearn.feature_selection import mutual_info_classif

from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score,

confusion_matrix, classification_report, roc_curve, auc

from sklearn.naive_bayes import MultinomialNB

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.neural_network import MLPClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,

VotingClassifier, StackingClassifier



#### Importación de los Datos a Google Colaboratory ####

## Emparejar con Google Drive ##

drive.mount('/content/drive')
```

```
## Definir ruta de la carpeta donde se encuentran los archivos CSV ##

ruta = '/content/drive/MyDrive/'


#### Descripción de la Base de Datos ####

## Lectura del Archivo CSV ##

BaseDatos = pd.read_csv('/content/drive/MyDrive/Dataset_Life_Insurance_Prudential.csv',

index_col='Id')

data_table.enable_dataframe_formatter()


## Muestra de las primeras filas ##

print(BaseDatos.head())


## Muestra de las últimas filas ##

print(BaseDatos.tail())


## Dimensiones de la tabla de datos ##

print(BaseDatos.shape)


## Número Total de Observaciones ##

print("Número total de registros: ", len(BaseDatos))
```

```
## Número Total de Variables, 126 Explicativas y la Variable Respuesta ##
print("Número total de variables: ", len(BaseDatos.columns))

## Conteo y Porcentaje de la Distribución de la Variable Respuesta ##
TablaFrecuencias = pd.DataFrame(BaseDatos['Response'].value_counts())
TablaFrecuencias = TablaFrecuencias.reset_index()
TablaFrecuencias.columns = ['Response', 'Frecuencia']
TablaFrecuencias['Porcentaje'] = TablaFrecuencias['Frecuencia']/len(BaseDatos)*100
print(TablaFrecuencias)

## Gráfico de la Distribución de la Variable Respuesta ##
plt.figure(figsize=(8, 6))
sns.countplot(data=BaseDatos, x='Response')
plt.xlabel('Segmento de la Variable Respuesta')
plt.ylabel('Recuento de Observaciones')
plt.title('Distribución de la Variable Respuesta')
plt.show()

## Tabla de Estadísticos Básicos de la Variable Respuesta ##
BaseDatos['Response'].describe()
```

```
## Tabla de Estadísticos Básicos de cada Variable ##

BaseDatos.describe()


## Comando para habilitar la visualización de todas las variables en los próximos pasos ##

pd.set_option('display.max_columns', 200)

pd.set_option('display.max_rows', 200)


## Cálculo de la Matriz de Correlaciones entre Variables ##

Correlacion = BaseDatos.corr()

print(Correlacion)


## Gráfico de la Matriz de Correlaciones entre Variables ##

corrs = BaseDatos.corr()

mask = np.zeros_like(corrs)

mask[np.triu_indices_from(mask)] = True


plt.figure(figsize=(24,16))

sns.heatmap(corrs, cmap='RdBu_r', mask=mask)

plt.show()
```

```
## Cálculo de la Matriz de Correlaciones con Valor Absoluto Superior a 0,9 entre Variables ##

Correlacion90 = Correlacion.where(np.triu(np.abs(Correlacion)>=0.9, k=1))

Correlacion90 = Correlacion90.stack().reset_index()

Correlacion90.columns = ['Variable 1', 'Variable 2', 'Correlación']

Correlacion90 = Correlacion90.dropna()

Correlacion90 = Correlacion90.sort_values(by='Correlación', ascending=False)

print(Correlacion90)


## Comprobación de  Valores Faltantes mediante Conteo por Columnas ##

Valores_Faltantes = BaseDatos.isnull().sum()


## Comprobación de Valores Faltantes mediante Porcentaje por Columnas ##

Porcentaje_Valores_Faltantes = round((Valores_Faltantes / len(BaseDatos)) * 100, 2)


## Tabla del Conteo y Porcentaje por Columnas con más del 50% de Valores Faltantes ##

print("Comprobación de columnas con más del 50% de valores faltantes (ordenado de mayor a

menor):\n")

for col, val, pct in sorted(zip(BaseDatos.columns, Valores_Faltantes,

Porcentaje_Valores_Faltantes), key=lambda x: x[2], reverse=True):

    if pct > 50:

        print("{}: {} ({:.2f}%)".format(col, val, pct))
```

```
## Cuantificación de la Información Mútua ##

def Informacion_Mutua(X, y):

    mi_scores = mutual_info_classif(X, y)

    mi_scores = pd.Series(mi_scores, name="Valores de Información Mútua", index=X.columns)

    mi_scores = mi_scores.sort_values(ascending=False)

    return mi_scores


def Grafico_Informacion_Mutua(scores):

    scores = scores.sort_values(ascending=True)

    width = np.arange(len(scores))

    ticks = list(scores.index)

    plt.barh(width, scores)

    plt.yticks(width, ticks)

    plt.title("Valores de Información Mútua")

    plt.show()


Valores_Informacion_Mutua = Informacion_Mutua(VariablesExplicativas, VariableRespuesta)


plt.figure(dpi=100, figsize=(20,50))


Grafico_Informacion_Mutua(Valores_Informacion_Mutua)
```

Metodología

Eliminación de Una de las Variables de cada Binomio con Correlación Mayor a 0,9 en Valor Absoluto

```
VariablesCorrelacion90 = ['Insurance_History_9', 'Medical_History_36', 'Family_Hist_4',
'Insurance_History_4', 'Medical_Keyword_11', 'Insurance_History_3', 'Medical_History_26',
'Medical_Keyword_23', 'Medical_Keyword_48']

for column in VariablesCorrelacion90:
    try:
        BaseDatos.drop(column, axis=1, inplace=True)
        print(f"La variable '{column}' ha sido eliminada correctamente.")
    except KeyError:
        print(f"La columna '{column}' no existe en la base de datos.")
```

Comprobación de Desaparición de Pares de Variables con con Correlación Mayor a 0,9 en Valor Absoluto

```
Correlacion = BaseDatos.corr()
Correlacion90 = Correlacion.where(np.triu(np.abs(Correlacion)>=0.9, k=1))
Correlacion90 = Correlacion90.stack().reset_index()
```

```
Correlacion90.columns = ['Variable 1', 'Variable 2', 'Correlación']

Correlacion90 = Correlacion90.dropna()

Correlacion90 = Correlacion90.sort_values(by='Correlación', ascending=False)

print(Correlacion90)


## Comprobación del Nuevo Número Total de Variables, 117 Explicativas y la Variable

Respuesta ##

print("Número total de variables: ", len(BaseDatos.columns))


## Eliminación de las Variables con Porcentaje de Valores Faltantes Superior al 50% ##

BaseDatos.dropna(thresh=len(BaseDatos)*0.5, axis=1, inplace=True)


## Comprobación de Desaparición de Variables con Más del 50% de Valores Faltantes ##

Valores_Faltantes = BaseDatos.isnull().sum()

Porcentaje_Valores_Faltantes = round((Valores_Faltantes / len(BaseDatos)) * 100, 2)

print("Comprobación de columnas con más del 50% de valores faltantes (ordenado de mayor a

menor):\n")


for col, val, pct in sorted(zip(BaseDatos.columns, Valores_Faltantes,

Porcentaje_Valores_Faltantes), key=lambda x: x[2], reverse=True):

    if pct > 50:

        print("{}: {} ({:.2f}%)".format(col, val, pct))
```

```
## Comprobación del Nuevo Número Total de Variables ##

print("Número total de variables: ", len(BaseDatos.columns))


## División de la Base de Datos en Variables Explicativas y Variable Respuesta ##

VariablesExplicativas = BaseDatos.drop(labels='Response',axis=1)

VariableRespuesta = BaseDatos['Response']


## Tabla del Porcentaje de Valores Faltantes por Columnas ##

for col in sorted(VariablesExplicativas, key=lambda x: VariablesExplicativas[x].isna().sum() /
len(VariablesExplicativas[x].index), reverse=True):
    sum = VariablesExplicativas[col].isna().sum()
    length = len(VariablesExplicativas[col].index)
    ratio = sum/length
    if ratio != 0:
        print('La proporción de Valores Faltantes de la Variable', col, 'es del:', round(ratio*100,2),
'%.')


## Imputación Iterativa en Variables Explicativas con Valores Faltantes ##

VariablesExplicativas_Sin_Imputar = VariablesExplicativas.copy()

VariablesExplicativas_Con_Valores_Faltantes =
['Family_Hist_2','Insurance_History_5','Employment_Info_6','Medical_History_1','Employment
_Info_4', 'Employment_Info_1']
```

```
VariablesExplicativas_Sin_Valores_Faltantes =
VariablesExplicativas_Sin_Imputar.drop(VariablesExplicativas_Con_Valores_Faltantes,
axis=1)

VariablesExplicativas_Previas =
VariablesExplicativas_Sin_Imputar[VariablesExplicativas_Con_Valores_Faltantes]

Imputacion_Iterativa = IterativeImputer(random_state=0)

VariablesExplicativas_Post =
pd.DataFrame(Imputacion_Iterativa.fit_transform(VariablesExplicativas_Previas),
columns=VariablesExplicativas_Previas.columns)

VariablesExplicativas_Post.index = VariablesExplicativas_Previas.index

VariablesExplicativas=pd.concat([VariablesExplicativas_Sin_Valores_Faltantes,
VariablesExplicativas_Post], axis=1)

## Tabla del Porcentaje de Valores Faltantes por Columnas, tras la Imputación Iterativa los
Valores Faltantes Totales son del 0% ##
```

```
for col in sorted(VariablesExplicativas, key=lambda x: VariablesExplicativas[x].isna().sum() /
len(VariablesExplicativas[x].index), reverse=True):
    sum = VariablesExplicativas[col].isna().sum()
    length = len(VariablesExplicativas[col].index)
    ratio = sum/length
    if ratio != 0:
        print('La proporción de Valores Faltantes de la Variable', col, 'es del:', round(ratio*100,2),
'%.')

## Gráficos KDE Previos a la Imputación Iterativa ##
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(25,10))
x_limits = {}
y_limits = {}

for i, column in enumerate(BaseDatos[VariablesExplicativas_Post.columns].columns):
    sns.kdeplot(data=BaseDatos[VariablesExplicativas_Post.columns],
                x=column,
                fill=True, common_norm=True, alpha=0.05,
                ax=axes[i//3,i%3])

    x_limits[column] = axes[i//3,i%3].get_xlim()
    y_limits[column] = axes[i//3,i%3].get_ylim()
```

```
## Gráficos KDE Posteriores a la Imputación Iterativa ##
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(25,10))

for i, column in enumerate(VariablesExplicativas_Post.columns):
    sns.kdeplot(data=VariablesExplicativas_Post,
                x=column,
                fill=True, common_norm=True, alpha=0.05,
                ax=axes[i//3,i%3])

    axes[i//3,i%3].set_xlim(x_limits[column])
    axes[i//3,i%3].set_ylim(y_limits[column])

## One-Hot Encoding para la Variable 'Product_Info_2', al ser Alfanumérica podría Generar
Problemas ##
One_Hot_Encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
VariablesExplicativas_OHE = VariablesExplicativas.copy()
One_Hot_Encoding_Columnas = ['Product_Info_2']
Proceso_One_Hot_Encoding =
pd.DataFrame(One_Hot_Encoder.fit_transform(VariablesExplicativas_OHE[One_Hot_Encodin
g_Columnas]))
Proceso_One_Hot_Encoding.index = VariablesExplicativas_OHE.index
```

```
Proceso_One_Hot_Encoding.columns =

One_Hot_Encoder.get_feature_names_out(One_Hot_Encoding_Columnas)


VariablesExplicativas_Sin_One_Hot_Encoding =

VariablesExplicativas_OHE.drop(One_Hot_Encoding_Columnas, axis=1)


VariablesExplicativas = pd.concat([VariablesExplicativas_Sin_One_Hot_Encoding,

Proceso_One_Hot_Encoding], axis=1)


print(VariablesExplicativas.head())


## Normalización de todas las Variables Explicativas ##

Normalizacion_Minimos_Maximos = MinMaxScaler()

Normalizacion_Minimos_Maximos.fit(VariablesExplicativas)


VariablesExplicativas =

pd.DataFrame(Normalizacion_Minimos_Maximos.transform(VariablesExplicativas),

                    index=VariablesExplicativas.index,

                    columns=VariablesExplicativas.columns)

print(VariablesExplicativas.head())
```

```
## División de la Base de Datos en dos Subconjuntos: Entrenamiento con el 80% de los
Individuos y Test con el 20% Restante ##
VariablesExplicativas_Entrenamiento, VariablesExplicativas_Test,
VariableRespuesta_Entrenamiento, VariableRespuesta_Test =
train_test_split(VariablesExplicativas,VariableRespuesta,train_size=0.8, test_size=0.1,
random_state=0, stratify=VariableRespuesta)

## Conteo y Porcentaje de la Distribución de la Variable Respuesta del Subconjunto de Test ##
TablaFrecuenciasTest = pd.DataFrame(VariableRespuesta_Test.value_counts())
TablaFrecuenciasTest = TablaFrecuenciasTest.reset_index()
TablaFrecuenciasTest.columns = ['Response', 'Frecuencia']
TablaFrecuenciasTest['Porcentaje'] =
TablaFrecuenciasTest['Frecuencia']/len(VariableRespuesta_Test)*100
print(TablaFrecuenciasTest)

## Conteo y Porcentaje de la Distribución de la Variable Respuesta del Subconjunto de
Entrenamiento ##
TablaFrecuenciasEntrenamiento =
pd.DataFrame(VariableRespuesta_Entrenamiento.value_counts())
TablaFrecuenciasEntrenamiento = TablaFrecuenciasEntrenamiento.reset_index()
TablaFrecuenciasEntrenamiento.columns = ['Response', 'Frecuencia']
```

```
TablaFrecuenciasEntrenamiento['Porcentaje'] =

TablaFrecuenciasEntrenamiento['Frecuencia']/len(VariableRespuesta_Entrenamiento)*100

print(TablaFrecuenciasEntrenamiento)


## Ponderación para la Estratificación del Número de Observaciones para cada Segmento de la

Variable Respuesta de Entrenamiento ##

Segmentos = [1, 2, 3, 4, 5, 6, 7, 8]


Proporciones = [0.1045, 0.1103, 0.0171, 0.0240, 0.0915, 0.1892, 0.1352, 0.3282]


PesosPonderados = []


for Segmento in VariableRespuesta_Entrenamiento:

    PesosPonderadosSegmentos = 1 / Proporciones[Segmento - 1]

    PesosPonderados.append(PesosPonderadosSegmentos)


#### Resultados ####


#### Modelos Econométricos Tradicionales ####


## Modelo de Regresión Logística Multinomial ##
```

```
# Definir las Alternativas de Hiperparámetros #

InicioRegresionLogistica = time.time()

HiperparametrosRegresionLogistica = {

'solver': ['sag', 'lbfgs', 'newton-cg'],

'C': [0.1, 1.0, 10.0],

'max_iter': [10000, 15000, 20000],

'penalty': ['l1', 'l2']

}


# Crear una Instancia para la Optimización de Hiperparámetros #

OptimizacionRegresionLogistica =

GridSearchCV(LogisticRegression(multi_class='multinomial'),

HiperparametrosRegresionLogistica, cv=5)


# Cálculo Exhaustivo de Hiperparámetros #

OptimizacionRegresionLogistica.fit(VariablesExplicativas_Entrenamiento,

VariableRespuesta_Entrenamiento)


# Mejores Hiperparámetros Encontrados #

MejoresHiperparametrosRegresionLogistica = OptimizacionRegresionLogistica.best_params_
```

```
# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
RegresionLogistica = LogisticRegression(multi_class='multinomial',
**MejoresHiperparametrosRegresionLogistica)

# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
RegresionLogistica.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesRegresionLogistica = RegresionLogistica.predict(VariablesExplicativas_Test)

# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionRegresionLogistica =
classification_report(VariableRespuesta_Test, PrediccionesRegresionLogistica)
print("Reporte de Métricas de Clasificación:\n",
ReporteMetricasClasificacionRegresionLogistica)

# Calcular el F-score #
FScoreRegresionLogistica = f1_score(VariableRespuesta_Test,
PrediccionesRegresionLogistica, average='weighted')
print("F-Score:", FScoreRegresionLogistica)
```

```
# Calcular la Precisión #
MetricaPrecisionRegresionLogistica = precision_score(VariableRespuesta_Test,
PrediccionesRegresionLogistica, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionRegresionLogistica)

# Calcular la Exactitud o Accuracy #
MetricaExactitudRegresionLogistica = accuracy_score(VariableRespuesta_Test,
PrediccionesRegresionLogistica)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudRegresionLogistica)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadRegresionLogistica = recall_score(VariableRespuesta_Test,
PrediccionesRegresionLogistica, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadRegresionLogistica)

# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCRegresionLogistica =
RegresionLogistica.predict_proba(VariablesExplicativas_Test)
VariableRespuesta_Test_Ajustada = VariableRespuesta_Test - 1
FPRRegresionLogistica = dict()
TPRRegresionLogistica = dict()
AUCROCRegresionLogistica = dict()
```

```
n_classes = ProbabilidadesROCRegresionLogistica.shape[1]
for i in range(n_classes):
    FPRRegresionLogistica[i], TPRRegresionLogistica[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCRegresionLogistica[:, i])
    AUCROCRegresionLogistica[i] = auc(FPRRegresionLogistica[i], TPRRegresionLogistica[i])

# Calcular la Media de la Curva ROC #
MediaFPRRegresionLogistica = np.linspace(0, 1, 100)
MediaTPRRegresionLogistica = np.zeros_like(MediaFPRRegresionLogistica)
for i in range(n_classes):
    MediaTPRRegresionLogistica += np.interp(MediaFPRRegresionLogistica,
FPRRegresionLogistica[i], TPRRegresionLogistica[i])
MediaTPRRegresionLogistica /= n_classes
MediaAUCRegresionLogistica = auc(MediaFPRRegresionLogistica,
MediaTPRRegresionLogistica)

# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
```

```
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRRegresionLogistica[i], TPRRegresionLogistica[i], color=color, lw=2,
label='Clase {0} (AUC = {1:0.2f})'.format(i+1, AUCROCRegresionLogistica[i]))

plt.plot(MediaFPRRegresionLogistica, MediaTPRRegresionLogistica, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCRegresionLogistica))

plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo RegresionLogistica')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionRegresionLogistica = confusion_matrix(VariableRespuesta_Test,
PrediccionesRegresionLogistica)
print("Matriz de Confusión:\n", MatrizConfusionRegresionLogistica)
```

```
# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
de Regresión Logística', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionRegresionLogistica, classes=['Segmento1',
'Segmento2', 'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
title='Matriz de Confusión del Modelo de Regresión Logística Multinomial')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionRegresionLogistica, classes=['Segmento1',
'Segmento2', 'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo de Regresión Logística
Multinomial')
plt.show()

# Tiempo de Ejecución del Modelo #
FinRegresionLogistica = time.time()
TiempoEjecucionRegresionLogistica = FinRegresionLogistica - InicioRegresionLogistica
TiempoEjecucionRegresionLogistica = TiempoEjecucionRegresionLogistica / 60
print("Tiempo de ejecución:", TiempoEjecucionRegresionLogistica, "minutos")
```

```
## Modelo de Análisis Discriminante Lineal ##


# Definir las Alterntivas de Hiperparámetros #

InicioAnalisisDiscriminanteLineal = time.time()

HiperparametrosAnalisisDiscriminanteLineal = {

'solver': ['lsqr', 'lbfgs'],

'shrinkage': [None, 'auto']

}


# Crear una Instancia para la Optimización de Hiperparámetros #

OptimizacionAnalisisDiscriminanteLineal = GridSearchCV(LinearDiscriminantAnalysis(),

HiperparametrosAnalisisDiscriminanteLineal, cv=5)


# Cálculo Exhaustivo de Hiperparámetros #

OptimizacionAnalisisDiscriminanteLineal.fit(VariablesExplicativas_Entrenamiento,

VariableRespuesta_Entrenamiento)


# Mejores Hiperparámetros Encontrados #

MejoresHiperparametrosAnalisisDiscriminanteLineal =

OptimizacionAnalisisDiscriminanteLineal.best_params_


# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
```

```
AnalisisDiscriminanteLineal =

LinearDiscriminantAnalysis(**MejoresHiperparametrosAnalisisDiscriminanteLineal)


# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #

AnalisisDiscriminanteLineal.fit(VariablesExplicativas_Entrenamiento,

VariableRespuesta_Entrenamiento)


# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #

PrediccionesAnalisisDiscriminanteLineal =

AnalisisDiscriminanteLineal.predict(VariablesExplicativas_Test)


# Generación del Reporte de Métricas de Clasificación #

ReporteMetricasClasificacionAnalisisDiscriminanteLineal =

classification_report(VariableRespuesta_Test, PrediccionesAnalisisDiscriminanteLineal)

print("Reporte de Métricas de Clasificación:\n",

ReporteMetricasClasificacionAnalisisDiscriminanteLineal)


# Calcular el F-score #

FScoreAnalisisDiscriminanteLineal = f1_score(VariableRespuesta_Test,

PrediccionesAnalisisDiscriminanteLineal, average='weighted')

print("F-Score:", FScoreAnalisisDiscriminanteLineal)
```

```
# Calcular la Precisión #
MetricaPrecisionAnalisisDiscriminanteLineal = precision_score(VariableRespuesta_Test,
PrediccionesAnalisisDiscriminanteLineal, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionAnalisisDiscriminanteLineal)

# Calcular la Exactitud o Accuracy #
MetricaExactitudAnalisisDiscriminanteLineal = accuracy_score(VariableRespuesta_Test,
PrediccionesAnalisisDiscriminanteLineal)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudAnalisisDiscriminanteLineal)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadAnalisisDiscriminanteLineal = recall_score(VariableRespuesta_Test,
PrediccionesAnalisisDiscriminanteLineal, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadAnalisisDiscriminanteLineal)

# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCAnalisisDiscriminanteLineal =
AnalisisDiscriminanteLineal.predict_proba(VariablesExplicativas_Test)
FPRAnalisisDiscriminanteLineal = dict()
TPRAnalisisDiscriminanteLineal = dict()
AUCROCAnalisisDiscriminanteLineal = dict()
n_classes = ProbabilidadesROCAnalisisDiscriminanteLineal.shape[1]
```

```
for i in range(n_classes):
    FPRAnalisisDiscriminanteLineal[i], TPRAnalisisDiscriminanteLineal[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCAnalisisDiscriminanteLineal[:, i])
    AUCROCAnalisisDiscriminanteLineal[i] = auc(FPRAnalisisDiscriminanteLineal[i],
TPRAnalisisDiscriminanteLineal[i])

# Calcular la Media de la Curva ROC #
MediaFPRAnalisisDiscriminanteLineal = np.linspace(0, 1, 100)

MediaTPRAnalisisDiscriminanteLineal =
np.zeros_like(MediaFPRAnalisisDiscriminanteLineal)

for i in range(n_classes):
    MediaTPRAnalisisDiscriminanteLineal += np.interp(MediaFPRAnalisisDiscriminanteLineal,
FPRAnalisisDiscriminanteLineal[i], TPRAnalisisDiscriminanteLineal[i])

MediaTPRAnalisisDiscriminanteLineal /= n_classes
MediaAUCAnalisisDiscriminanteLineal = auc(MediaFPRAnalisisDiscriminanteLineal,
MediaTPRAnalisisDiscriminanteLineal)
```

```
# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRAnalisisDiscriminanteLineal[i], TPRAnalisisDiscriminanteLineal[i], color=color,
lw=2, label='Clase {0} (AUC = {1:0.2f})'.format(i+1,
AUCROCAnalisisDiscriminanteLineal[i]))

plt.plot(MediaFPRAnalisisDiscriminanteLineal, MediaTPRAnalisisDiscriminanteLineal,
color='black', lw=2, linestyle='-.', label='Media (AUC =
{0:0.2f})'.format(MediaAUCAnalisisDiscriminanteLineal))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo Analisis Discriminante Lineal')
plt.legend(loc="lower right")
plt.show()
```

```
# Calcular la Matriz de Confusión #
MatrizConfusionAnalisisDiscriminanteLineal = confusion_matrix(VariableRespuesta_Test,
PrediccionesAnalisisDiscriminanteLineal)

print("Matriz de Confusión:\n", MatrizConfusionAnalisisDiscriminanteLineal)

# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
de Análisis Discriminante Lineal', cmap=plt.cm.Blues):

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionAnalisisDiscriminanteLineal, classes=['Segmento1',
'Segmento2', 'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
title='Matriz de Confusión del Análisis Discriminante Lineal')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionAnalisisDiscriminanteLineal, classes=['Segmento1',
'Segmento2', 'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Análisis Discriminante Lineal')
plt.show()
```

```
# Tiempo de Ejecución del Modelo #
FinAnalisisDiscriminanteLineal = time.time()
TiempoEjecucionAnalisisDiscriminanteLineal = FinAnalisisDiscriminanteLineal -
InicioAnalisisDiscriminanteLineal
TiempoEjecucionAnalisisDiscriminanteLineal = TiempoEjecucionAnalisisDiscriminanteLineal
/ 60
print("Tiempo de ejecución:", TiempoEjecucionAnalisisDiscriminanteLineal, "minutos")



## Modelo de Naive Bayes Multinomial ##


# Definir las Alternativas de Hiperparámetros #
InicioNaiveBayes = time.time()
HiperparametrosNaiveBayes = {
    'alpha': [0.1, 0.25, 0.5, 0.75, 1.0],
    'fit_prior': [True, False],
}


# Crear una Instancia para la Optimización de Hiperparámetros #
OptimizacionNaiveBayes = GridSearchCV(MultinomialNB(), HiperparametrosNaiveBayes,
cv=5)
```

```
# Cálculo Exhaustivo de Hiperparámetros #
OptimizacionNaiveBayes.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)

# Mejores Hiperparámetros Encontrados #
MejoresHiperparametrosNaiveBayes = OptimizacionNaiveBayes.best_params_

# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
NaiveBayes = MultinomialNB(**MejoresHiperparametrosNaiveBayes)

# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
NaiveBayes.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesNaiveBayes = NaiveBayes.predict(VariablesExplicativas_Test)

# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionNaiveBayes = classification_report(VariableRespuesta_Test,
PrediccionesNaiveBayes)
print("Reporte de Métricas de Clasificación:\n", ReporteMetricasClasificacionNaiveBayes)
```

```
# Calcular el F-score #
FScoreNaiveBayes = f1_score(VariableRespuesta_Test, PrediccionesNaiveBayes,
average='weighted')
print("F-Score:", FScoreNaiveBayes)

# Calcular la Precisión #
MetricaPrecisionNaiveBayes = precision_score(VariableRespuesta_Test,
PrediccionesNaiveBayes, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionNaiveBayes)

# Calcular la Exactitud o Accuracy #
MetricaExactitudNaiveBayes = accuracy_score(VariableRespuesta_Test,
PrediccionesNaiveBayes)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudNaiveBayes)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadNaiveBayes = recall_score(VariableRespuesta_Test,
PrediccionesNaiveBayes, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadNaiveBayes)

# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCNaiveBayes = NaiveBayes.predict_proba(VariablesExplicativas_Test)
```

```
FPRNaiveBayes = dict()

TPRNaiveBayes = dict()

AUCROCNaiveBayes = dict()

n_classes = ProbabilidadesROCNaiveBayes.shape[1]

for i in range(n_classes):

    FPRNaiveBayes[i], TPRNaiveBayes[i], _ = roc_curve((VariableRespuesta_Test_Ajustada ==

i).astype(int), ProbabilidadesROCNaiveBayes[:, i])

    AUCROCNaiveBayes[i] = auc(FPRNaiveBayes[i], TPRNaiveBayes[i])


# Calcular la Media de la Curva ROC #

MediaFPRNaiveBayes = np.linspace(0, 1, 100)

MediaTPRNaiveBayes = np.zeros_like(MediaFPRNaiveBayes)

for i in range(n_classes):

    MediaTPRNaiveBayes += np.interp(MediaFPRNaiveBayes, FPRNaiveBayes[i],

TPRNaiveBayes[i])

MediaTPRNaiveBayes /= n_classes

MediaAUCNaiveBayes = auc(MediaFPRNaiveBayes, MediaTPRNaiveBayes)


# Visualizar la Curva ROC para cada Segmento #

plt.figure()

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
```

```
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRNaiveBayes[i], TPRNaiveBayes[i], color=color, lw=2, label='Clase {0} (AUC =
{1:0.2f})'.format(i+1, AUCROCNaiveBayes[i]))
plt.plot(MediaFPRNaiveBayes, MediaTPRNaiveBayes, color='black', lw=2, linestyle='-.',
label='Media (AUC = {0:0.2f})'.format(MediaAUCNaiveBayes))

plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo Naive Bayes')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionNaiveBayes = confusion_matrix(VariableRespuesta_Test,
PrediccionesNaiveBayes)
print("Matriz de Confusión:\n", MatrizConfusionNaiveBayes)
```

```
# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
de Naive Bayes', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionNaiveBayes, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión Ponderada de Naive Bayes Multinomial')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionNaiveBayes, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada de Naive Bayes Multinomial')
plt.show()

# Tiempo de Ejecución del Modelo #
FinNaiveBayes = time.time()
TiempoEjecucionNaiveBayes = FinNaiveBayes - InicioNaiveBayes
TiempoEjecucionNaiveBayes = TiempoEjecucionNaiveBayes / 60
print("Tiempo de ejecución:", TiempoEjecucionNaiveBayes, "minutos")
```

```
## Modelo de Árboles de Decisión ##


# Definir las Alterntivas de Hiperparámetros #

InicioArbolesDecision = time.time()

HiperparametrosArbolesDecision = {

'criterion': ['gini', 'entropy'],

'max_depth': [None, 10, 20],

'min_samples_split': [100,300,500],

'min_samples_leaf': [1, 2, 4],

'max_features': ['sqrt', 'log2']

}


# Crear una Instancia para la Optimización de Hiperparámetros #

OptimizacionArbolesDecision = GridSearchCV(DecisionTreeClassifier(),

HiperparametrosArbolesDecision, cv=5)


# Cálculo Exhaustivo de Hiperparámetros #

OptimizacionArbolesDecision.fit(VariablesExplicativas_Entrenamiento,

VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)


# Mejores Hiperparámetros Encontrados #

MejoresHiperparametrosArbolesDecision = OptimizacionArbolesDecision.best_params_
```

```
# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
ArbolesDecision = DecisionTreeClassifier(**MejoresHiperparametrosArbolesDecision)

# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
ArbolesDecision.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesArbolesDecision = ArbolesDecision.predict(VariablesExplicativas_Test)

# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionArbolesDecision = classification_report(VariableRespuesta_Test,
PrediccionesArbolesDecision)

print("Reporte de Métricas de Clasificación:\n", ReporteMetricasClasificacionArbolesDecision)

# Calcular el F-score #
FScoreArbolesDecision = f1_score(VariableRespuesta_Test, PrediccionesArbolesDecision,
average='weighted')

print("F-Score:", FScoreArbolesDecision)
```

```
# Calcular la Precisión #
MetricaPrecisionArbolesDecision = precision_score(VariableRespuesta_Test,
PrediccionesArbolesDecision, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionArbolesDecision)

# Calcular la Exactitud o Accuracy #
MetricaExactitudArbolesDecision = accuracy_score(VariableRespuesta_Test,
PrediccionesArbolesDecision)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudArbolesDecision)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadArbolesDecision = recall_score(VariableRespuesta_Test,
PrediccionesArbolesDecision, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadArbolesDecision)

# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCArbolesDecision =
ArbolesDecision.predict_proba(VariablesExplicativas_Test)
FPRArbolesDecision = dict()
TPRArbolesDecision = dict()
AUCROCArbolesDecision = dict()
n_classes = ProbabilidadesROCArbolesDecision.shape[1]
```

```
for i in range(n_classes):
    FPRArbolesDecision[i], TPRArbolesDecision[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCArbolesDecision[:, i])
    AUCROCArbolesDecision[i] = auc(FPRArbolesDecision[i], TPRArbolesDecision[i])

# Calcular la Media de la Curva ROC #
MediaFPRArbolesDecision = np.linspace(0, 1, 100)
MediaTPRArbolesDecision = np.zeros_like(MediaFPRArbolesDecision)
for i in range(n_classes):
    MediaTPRArbolesDecision += np.interp(MediaFPRArbolesDecision,
FPRArbolesDecision[i], TPRArbolesDecision[i])
MediaTPRArbolesDecision /= n_classes
MediaAUCArbolesDecision = auc(MediaFPRArbolesDecision, MediaTPRArbolesDecision)

# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRArbolesDecision[i], TPRArbolesDecision[i], color=color, lw=2, label='Clase {0}
(AUC = {1:0.2f})'.format(i+1, AUCROCArbolesDecision[i]))
```

```
plt.plot(MediaFPRArbolesDecision, MediaTPRArbolesDecision, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCArbolesDecision))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo de Árboles de Decisión')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionArbolesDecision = confusion_matrix(VariableRespuesta_Test,
PrediccionesArbolesDecision)
print("Matriz de Confusión:\n", MatrizConfusionArbolesDecision)

# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
de Árboles de Decisión', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
```

```
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionArbolesDecision, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo de Árboles de Decisión')
plt.show()
```

```
# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionArbolesDecision, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo de Árboles de Decisión')
plt.show()

# Tiempo de Ejecución del Modelo #
FinArbolesDecision = time.time()
TiempoEjecucionArbolesDecision = FinArbolesDecision - InicioArbolesDecision
TiempoEjecucionArbolesDecision = TiempoEjecucionArbolesDecision / 60
print("Tiempo de ejecución:", TiempoEjecucionArbolesDecision, "minutos")



#### Modelos Basados en Machine Learning ####



## Modelo de Random Forest ##


# Definir las Alterntivas de Hiperparámetros #
InicioRandomForest = time.time()
```

```
HiperparametrosRandomForest = {
'n_estimators': [50, 100, 200],
'criterion': ['gini', 'entropy'],
'max_depth': [None, 10, 20],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'max_features': ['sqrt', 'log2']
}

# Crear una Instancia para la Optimización de Hiperparámetros #
OptimizacionRandomForest = GridSearchCV(RandomForestClassifier(),
HiperparametrosRandomForest, cv=5)

# Cálculo Exhaustivo de Hiperparámetros #
OptimizacionRandomForest.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)

# Mejores Hiperparámetros Encontrados #
MejoresHiperparametrosRandomForest = OptimizacionRandomForest.best_params_

# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
RandomForest = RandomForestClassifier(**MejoresHiperparametrosRandomForest)
```

```
# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
RandomForest.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesRandomForest = RandomForest.predict(VariablesExplicativas_Test)

# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionRandomForest = classification_report(VariableRespuesta_Test,
PrediccionesRandomForest)
print("Reporte de Métricas de Clasificación:\n", ReporteMetricasClasificacionRandomForest)

# Calcular el F-score #
FScoreRandomForest = f1_score(VariableRespuesta_Test, PrediccionesRandomForest,
average='weighted')
print("F-Score:", FScoreRandomForest)

# Calcular la Precisión #
MetricaPrecisionRandomForest = precision_score(VariableRespuesta_Test,
PrediccionesRandomForest, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionRandomForest)
```

```
# Calcular la Exactitud o Accuracy #
MetricaExactitudRandomForest = accuracy_score(VariableRespuesta_Test,
PrediccionesRandomForest)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudRandomForest)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadRandomForest = recall_score(VariableRespuesta_Test,
PrediccionesRandomForest, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadRandomForest)

# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCRandomForest = RandomForest.predict_proba(VariablesExplicativas_Test)
FPRRandomForest = dict()
TPRRandomForest = dict()
AUCROCRandomForest = dict()
n_classes = ProbabilidadesROCRandomForest.shape[1]
for i in range(n_classes):
    FPRRandomForest[i], TPRRandomForest[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCRandomForest[:, i])
    AUCROCRandomForest[i] = auc(FPRRandomForest[i], TPRRandomForest[i])
```

```
# Calcular la Media de la Curva ROC #
MediaFPRRandomForest = np.linspace(0, 1, 100)
MediaTPRRandomForest = np.zeros_like(MediaFPRRandomForest)
for i in range(n_classes):
    MediaTPRRandomForest += np.interp(MediaFPRRandomForest, FPRRandomForest[i],
TPRRandomForest[i])
MediaTPRRandomForest /= n_classes
MediaAUCRandomForest = auc(MediaFPRRandomForest, MediaTPRRandomForest)


# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRRandomForest[i], TPRRandomForest[i], color=color, lw=2, label='Clase {0}
(AUC = {1:0.2f})'.format(i+1, AUCROCRandomForest[i]))
plt.plot(MediaFPRRandomForest, MediaTPRRandomForest, color='black', lw=2, linestyle='-.',
label='Media (AUC = {0:0.2f})'.format(MediaAUCRandomForest))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
```

```
plt.title('Curva ROC One vs All del Modelo RandomForest')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionRandomForest = confusion_matrix(VariableRespuesta_Test,
PrediccionesRandomForest)
print("Matriz de Confusión:\n", MatrizConfusionRandomForest)

# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
Random Forest', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionRandomForest, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo de Random Forest')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionRandomForest, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo de Random Forest')
plt.show()
```

```
# Tiempo de Ejecución del Modelo #

FinRandomForest = time.time()


TiempoEjecucionRandomForest = FinRandomForest – InicioRandomForest


TiempoEjecucionRandomForest = TiempoEjecucionRandomForest / 60


print("Tiempo de ejecución:", TiempoEjecucionRandomForest, "minutos")


## Modelo de Gradient Boosting ##


# Definir las Alternativas de Hiperparámetros #

InicioGradientBoosting = time.time()

HiperparametrosGradientBoosting = {

'learning_rate': [0.01, 0.1, 1],

'n_estimators': [50, 100, 200],

'max_depth': [3, 5, 7],

'min_samples_split': [2, 5, 10],

'min_samples_leaf': [1, 2, 4],

'max_features': ['sqrt', 'log2']

}
```

```
#Crear una Instancia para la Optimización de Hiperparámetros #
OptimizacionGradientBoosting = GridSearchCV(GradientBoostingClassifier(),
HiperparametrosGradientBoosting, cv=5)

# Cálculo Exhaustivo de Hiperparámetros #
OptimizacionGradientBoosting.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)

# Mejores Hiperparámetros Encontrados #
MejoresHiperparametrosGradientBoosting = OptimizacionGradientBoosting.best_params_

# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
GradientBoosting = GradientBoostingClassifier(**MejoresHiperparametrosGradientBoosting)

# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
GradientBoosting.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesGradientBoosting = GradientBoosting.predict(VariablesExplicativas_Test)
```

```
# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionGradientBoosting = classification_report(VariableRespuesta_Test,
PrediccionesGradientBoosting)
print("Reporte de Métricas de Clasificación:\n",
ReporteMetricasClasificacionGradientBoosting)

# Calcular el F-score #
FScoreGradientBoosting = f1_score(VariableRespuesta_Test, PrediccionesGradientBoosting,
average='weighted')
print("F-Score:", FScoreGradientBoosting)

# Calcular la Precisión #
MetricaPrecisionGradientBoosting = precision_score(VariableRespuesta_Test,
PrediccionesGradientBoosting, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionGradientBoosting)

# Calcular la Exactitud o Accuracy #
MetricaExactitudGradientBoosting = accuracy_score(VariableRespuesta_Test,
PrediccionesGradientBoosting)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudGradientBoosting)
```

```
# Calcular la Exhaustividad o Recall #

MetricaExhaustividadGradientBoosting = recall_score(VariableRespuesta_Test,

PrediccionesGradientBoosting, average='weighted')

print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadGradientBoosting)


# Calcular la Curva ROC para cada Segmento #

ProbabilidadesROCGradientBoosting =

GradientBoosting.predict_proba(VariablesExplicativas_Test)


FPRGradientBoosting = dict()

TPRGradientBoosting = dict()


AUCROCGradientBoosting = dict()


n_classes = ProbabilidadesROCGradientBoosting.shape[1]


for i in range(n_classes):

    FPRGradientBoosting[i], TPRGradientBoosting[i], _ =

roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),

ProbabilidadesROCGradientBoosting[:, i])

    AUCROCGradientBoosting[i] = auc(FPRGradientBoosting[i], TPRGradientBoosting[i])
```

```
# Calcular la Media de la Curva ROC #
MediaFPRGradientBoosting = np.linspace(0, 1, 100)
MediaTPRGradientBoosting = np.zeros_like(MediaFPRGradientBoosting)
for i in range(n_classes):
    MediaTPRGradientBoosting += np.interp(MediaFPRGradientBoosting,
FPRGradientBoosting[i], TPRGradientBoosting[i])
MediaTPRGradientBoosting /= n_classes
MediaAUCGradientBoosting = auc(MediaFPRGradientBoosting, MediaTPRGradientBoosting)


# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRGradientBoosting[i], TPRGradientBoosting[i], color=color, lw=2, label='Clase
{0} (AUC = {1:0.2f})'.format(i+1, AUCROCGradientBoosting[i]))
plt.plot(MediaFPRGradientBoosting, MediaTPRGradientBoosting, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCGradientBoosting))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
```

```
plt.title('Curva ROC One vs All del Modelo Gradient Boosting')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionGradientBoosting = confusion_matrix(VariableRespuesta_Test,
PrediccionesGradientBoosting)
print("Matriz de Confusión:\n", MatrizConfusionGradientBoosting)

# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
Gradient Boosting', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionGradientBoosting, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo de Gradient Boosting')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionGradientBoosting, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo de Gradient Boosting')
plt.show()
```

```
# Tiempo de Ejecución del Modelo #
FinGradientBoosting = time.time()
TiempoEjecucionGradientBoosting = FinGradientBoosting - InicioGradientBoosting
TiempoEjecucionGradientBoosting = TiempoEjecucionGradientBoosting / 60
print("Tiempo de ejecución:", TiempoEjecucionGradientBoosting, "minutos")



## Modelo de XGBoost ##


# Definir las Alternativas de Hiperparámetros #
InicioXGBoost = time.time()
HiperparametrosXGBoost = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200, 500],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}


# Restar 1 a los Valores de Entrenamiento de la Variable Respuesta #
VariableRespuesta_Entrenamiento -= 1
```

```
# Restar 1 a los Valores de Test de la Variable Respuesta #
VariableRespuesta_Test -= 1

# Crear una Instancia para la Optimización de Hiperparámetros #
OptimizacionXGBoost = GridSearchCV(xgb.XGBClassifier(), HiperparametrosXGBoost,
cv=5)

# Cálculo Exhaustivo de Hiperparámetros #
OptimizacionXGBoost.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)

# Mejores Hiperparámetros Encontrados #
MejoresHiperparametrosXGBoost = OptimizacionXGBoost.best_params_

# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
XGBoost = xgb.XGBClassifier(**MejoresHiperparametrosXGBoost)

# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
XGBoost.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesXGBoost = XGBoost.predict(VariablesExplicativas_Test)
```

```
# Sumar 1 a los Valores de las Predicciones para Obtener Segmentos del 1 al 8 #
PrediccionesXGBoost  += 1

# Restar 1 a los Valores de Entrenamiento de la Variable Respuesta para Volver a la Original #
VariableRespuesta_Entrenamiento += 1

# Restar 1 a los Valores de Test de la Variable Respuesta para Volver a la Original #
VariableRespuesta_Test += 1

# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionXGBoost = classification_report(VariableRespuesta_Test,
PrediccionesXGBoost)
print("Reporte de Métricas de Clasificación:\n", ReporteMetricasClasificacionXGBoost)

# Calcular el F-score #
FScoreXGBoost = f1_score(VariableRespuesta_Test, PrediccionesXGBoost,
average='weighted')
print("F-Score:", FScoreXGBoost)

# Calcular la Precisión #
MetricaPrecisionXGBoost = precision_score(VariableRespuesta_Test, PrediccionesXGBoost,
average='weighted')
```

```
print("Métrica de Precisión:", MetricaPrecisionXGBoost)


# Calcular la Exactitud o Accuracy #
MetricaExactitudXGBoost = accuracy_score(VariableRespuesta_Test, PrediccionesXGBoost)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudXGBoost)


# Calcular la Exhaustividad o Recall #
MetricaExhaustividadXGBoost = recall_score(VariableRespuesta_Test, PrediccionesXGBoost,
average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadXGBoost)


# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCXGBoost = XGBoost.predict_proba(VariablesExplicativas_Test)
FPRXGBoost = dict()
TPRXGBoost = dict()
AUCROCXGBoost = dict()
n_classes = ProbabilidadesROCXGBoost.shape[1]
for i in range(n_classes):
    FPRXGBoost[i], TPRXGBoost[i], _ = roc_curve((VariableRespuesta_Test_Ajustada ==
i).astype(int), ProbabilidadesROCXGBoost[:, i])
    AUCROCXGBoost[i] = auc(FPRXGBoost[i], TPRXGBoost[i])
```

```
# Calcular la Media de la Curva ROC #
MediaFPRXGBoost = np.linspace(0, 1, 100)
MediaTPRXGBoost = np.zeros_like(MediaFPRXGBoost)
for i in range(n_classes):
    MediaTPRXGBoost += np.interp(MediaFPRXGBoost, FPRXGBoost[i], TPRXGBoost[i])
MediaTPRXGBoost /= n_classes
MediaAUCXGBoost = auc(MediaFPRXGBoost, MediaTPRXGBoost)

# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRXGBoost[i], TPRXGBoost[i], color=color, lw=2, label='Clase {0} (AUC =
{1:0.2f})'.format(i+1, AUCROCXGBoost[i]))
plt.plot(MediaFPRXGBoost, MediaTPRXGBoost, color='black', lw=2, linestyle='-.',
label='Media (AUC = {0:0.2f})'.format(MediaAUCXGBoost))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
```

```
plt.title('Curva ROC One vs All del Modelo XGBoost')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionXGBoost = confusion_matrix(VariableRespuesta_Test, PrediccionesXGBoost)
print("Matriz de Confusión:\n", MatrizConfusionXGBoost)

# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
XGBoost', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
```

```
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionXGBoost, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo XGBoost')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionXGBoost, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo XGBoost')
plt.show()
```

```
# Tiempo de Ejecución del Modelo #
FinXGBoost = time.time()
TiempoEjecucionXGBoost = FinXGBoost - InicioXGBoost
TiempoEjecucionXGBoost = TiempoEjecucionXGBoost / 60
print("Tiempo de ejecución:", TiempoEjecucionXGBoost, "minutos")



## Modelo de Redes Neuronales ##


# Definir las Alterntivas de Hiperparámetros #
InicioRedesNeuronales = time.time()
HiperparametrosRedesNeuronales = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'activation': ['relu', 'tanh', 'logistic', 'identity'],
    'solver': ['adam', 'sgd', 'lbfgs'],
    'alpha': [0.0001, 0.001, 0.01],
    'max_iter': [200, 2000, 20000]
}


# Crear una Instancia para la Optimización de Hiperparámetros #
OptimizacionRedesNeuronales = GridSearchCV(MLPClassifier(),
HiperparametrosRedesNeuronales, cv=5)
```

```
# Cálculo Exhaustivo de Hiperparámetros #
OptimizacionRedesNeuronales.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento)

# Mejores Hiperparámetros Encontrados #
MejoresHiperparametrosRedesNeuronales = OptimizacionRedesNeuronales.best_params_

# Crear una Instancia del Modelo con los Mejores Hiperparámetros #
RedesNeuronales = MLPClassifier(**MejoresHiperparametrosRedesNeuronales)
RedesNeuronales = MLPClassifier()

# Modelización sobre los Datos de Entrenamiento con los Mejores Hiperparámetros #
RedesNeuronales.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento)

# Predicción sobre los Datos de Test con los Mejores Hiperparámetros #
PrediccionesRedesNeuronales = RedesNeuronales.predict(VariablesExplicativas_Test)

# Generación del Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionRedesNeuronales = classification_report(VariableRespuesta_Test,
PrediccionesRedesNeuronales)
print("Reporte de Métricas de Clasificación:\n",
ReporteMetricasClasificacionRedesNeuronales)
```

```
# Calcular el Valor F-score #
FScoreRedesNeuronales = f1_score(VariableRespuesta_Test, PrediccionesRedesNeuronales,
average='weighted')
print("F-Score:", FScoreRedesNeuronales)

# Calcular la Precisión #
MetricaPrecisionRedesNeuronales = precision_score(VariableRespuesta_Test,
PrediccionesRedesNeuronales, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionRedesNeuronales)

# Calcular la Exactitud o Accuracy #
MetricaExactitudRedesNeuronales = accuracy_score(VariableRespuesta_Test,
PrediccionesRedesNeuronales)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudRedesNeuronales)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadRedesNeuronales = recall_score(VariableRespuesta_Test,
PrediccionesRedesNeuronales, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadRedesNeuronales)
```

```
# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCRedesNeuronales =
RedesNeuronales.predict_proba(VariablesExplicativas_Test)
FPRRedesNeuronales = dict()
TPRRedesNeuronales = dict()
AUCROCRedesNeuronales = dict()
n_classes = ProbabilidadesROCRedesNeuronales.shape[1]
for i in range(n_classes):
    FPRRedesNeuronales[i], TPRRedesNeuronales[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCRedesNeuronales[:, i])
    AUCROCRedesNeuronales[i] = auc(FPRRedesNeuronales[i], TPRRedesNeuronales[i])

# Calcular la Media de la Curva ROC #
MediaFPRRedesNeuronales = np.linspace(0, 1, 100)
MediaTPRRedesNeuronales = np.zeros_like(MediaFPRRedesNeuronales)
for i in range(n_classes):
    MediaTPRRedesNeuronales += np.interp(MediaFPRRedesNeuronales,
FPRRedesNeuronales[i], TPRRedesNeuronales[i])
MediaTPRRedesNeuronales /= n_classes
MediaAUCRedesNeuronales = auc(MediaFPRRedesNeuronales, MediaTPRRedesNeuronales)
```

```
# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRRedesNeuronales[i], TPRRedesNeuronales[i], color=color, lw=2, label='Clase
{0} (AUC = {1:0.2f})'.format(i+1, AUCROCRedesNeuronales[i]))
plt.plot(MediaFPRRedesNeuronales, MediaTPRRedesNeuronales, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCRedesNeuronales))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo Redes Neuronales')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionRedesNeuronales = confusion_matrix(VariableRespuesta_Test,
PrediccionesRedesNeuronales)
print("Matriz de Confusión:\n", MatrizConfusionRedesNeuronales)
```

```
# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
de Redes Neuronales', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
  plt.tight_layout()
   plt.ylabel('Segmento Real de la Base de Datos')
   plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionRedesNeuronales, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo de Redes Neuronales')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionRedesNeuronales, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo de Redes Neuronales')
plt.show()

# Tiempo de Ejecución del Modelo #
FinRedesNeuronales = time.time()
TiempoEjecucionRedesNeuronales = FinRedesNeuronales - InicioRedesNeuronales
TiempoEjecucionRedesNeuronales = TiempoEjecucionRedesNeuronales / 60
print("Tiempo de ejecución:", TiempoEjecucionRedesNeuronales, "minutos")
```

```
## Modelo Voting Classifier con XGBoost y Regresión Logística Multinomial, ambos con
Hiperparámetros Optimizados ##

# Crear el Voting Classifier #
InicioVotingClassifier = time.time()
Voting_Classifier = VotingClassifier(
    estimators=[('xgb', XGBoost), ('lr', RegresionLogistica)],
    voting='soft'
)

# Entrenar el Voting Classifier #
Voting_Classifier.fit(VariablesExplicativas_Entrenamiento, VariableRespuesta_Entrenamiento,
sample_weight=PesosPonderados)

# Predecir con el Voting Classifier #
PrediccionesVotingClassifier = Voting_Classifier.predict(VariablesExplicativas_Test)

# Generar el Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionVotingClassifier = classification_report(VariableRespuesta_Test,
PrediccionesVotingClassifier)
print("Reporte de Métricas de Clasificación:\n", ReporteMetricasClasificacionVotingClassifier)
```

```
# Calcular el F-score #
FScoreVotingClassifier = f1_score(VariableRespuesta_Test, PrediccionesVotingClassifier,
average='weighted')
print("F-Score:", FScoreVotingClassifier)

# Calcular la Precisión #
MetricaPrecisionVotingClassifier = precision_score(VariableRespuesta_Test,
PrediccionesVotingClassifier, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionVotingClassifier)

# Calcular la Exactitud o Accuracy #
MetricaExactitudVotingClassifier = accuracy_score(VariableRespuesta_Test,
PrediccionesVotingClassifier)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudVotingClassifier)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadVotingClassifier = recall_score(VariableRespuesta_Test,
PrediccionesVotingClassifier, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadVotingClassifier)
```

```
# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCVotingClassifier =
Voting_Classifier.predict_proba(VariablesExplicativas_Test)
FPRVotingClassifier = dict()
TPRVotingClassifier = dict()
AUCROCVotingClassifier = dict()
n_classes = ProbabilidadesROCVotingClassifier.shape[1]
for i in range(n_classes):
   FPRVotingClassifier[i], TPRVotingClassifier[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCVotingClassifier[:, i])
   AUCROCVotingClassifier[i] = auc(FPRVotingClassifier[i], TPRVotingClassifier[i])

# Calcular la Media de la Curva ROC #
MediaFPRVotingClassifier = np.linspace(0, 1, 100)
MediaTPRVotingClassifier = np.zeros_like(MediaFPRVotingClassifier)
for i in range(n_classes):
   MediaTPRVotingClassifier += np.interp(MediaFPRVotingClassifier, FPRVotingClassifier[i],
TPRVotingClassifier[i])
MediaTPRVotingClassifier /= n_classes
MediaAUCVotingClassifier = auc(MediaFPRVotingClassifier, MediaTPRVotingClassifier)
```

```
# Visualizar la Curva ROC para cada Segmento #
plt.figure()
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRVotingClassifier[i], TPRVotingClassifier[i], color=color, lw=2, label='Clase {0}
(AUC = {1:0.2f})'.format(i+1, AUCROCVotingClassifier[i]))
plt.plot(MediaFPRVotingClassifier, MediaTPRVotingClassifier, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCVotingClassifier))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo Voting Classifier')
plt.legend(loc="lower right")
plt.show()

# Calcular la Matriz de Confusión #
MatrizConfusionVotingClassifier = confusion_matrix(VariableRespuesta_Test,
PrediccionesVotingClassifier)
print("Matriz de Confusión:\n", MatrizConfusionVotingClassifier)
```

```
# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
Voting Classifier', cmap=plt.cm.Blues):

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionVotingClassifier, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo Voting Classifier')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionVotingClassifier, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo Voting Classifier')
plt.show()

# Tiempo de Ejecución del Modelo #
FinVotingClassifier = time.time()
TiempoEjecucionVotingClassifier = FinVotingClassifier - InicioVotingClassifier
TiempoEjecucionVotingClassifier = TiempoEjecucionVotingClassifier / 60
print("Tiempo de ejecución:", TiempoEjecucionVotingClassifier, "minutos")
```

```
## Modelo Stacking Classifier con XGBoost y Regresión Logística Multinomial, ambos con
Hiperparámetros Optimizados ##

# Crear el Stacking Classifier #
InicioStackingClassifier = time.time()
EstimadoresStackingClassifier = [
    ('xgb', XGBClassifier())
]
Stacking_Classifier = StackingClassifier(
    estimators=EstimadoresStackingClassifier,
    final_estimator=LogisticRegression(max_iter=10000)
)
# Entrenar el Stacking Classifier #
Stacking_Classifier.fit(VariablesExplicativas_Entrenamiento,
VariableRespuesta_Entrenamiento, sample_weight=PesosPonderados)

# Predecir con el Stacking Classifier #
PrediccionesStackingClassifier = Stacking_Classifier.predict(VariablesExplicativas_Test)

# Generar el Reporte de Métricas de Clasificación #
ReporteMetricasClasificacionStackingClassifier =
classification_report(VariableRespuesta_Test, PrediccionesStackingClassifier)
```

```
print("Reporte de Métricas de Clasificación:\n",
ReporteMetricasClasificacionStackingClassifier)

# Calcular el F-score #
FScoreStackingClassifier = f1_score(VariableRespuesta_Test, PrediccionesStackingClassifier,
average='weighted')
print("F-Score:", FScoreStackingClassifier)

# Calcular la Precisión #
MetricaPrecisionStackingClassifier = precision_score(VariableRespuesta_Test,
PrediccionesStackingClassifier, average='weighted')
print("Métrica de Precisión:", MetricaPrecisionStackingClassifier)

# Calcular la Exactitud o Accuracy #
MetricaExactitudStackingClassifier = accuracy_score(VariableRespuesta_Test,
PrediccionesStackingClassifier)
print("Métrica de Exactitud o Accuracy:", MetricaExactitudStackingClassifier)

# Calcular la Exhaustividad o Recall #
MetricaExhaustividadStackingClassifier = recall_score(VariableRespuesta_Test,
PrediccionesStackingClassifier, average='weighted')
print("Métrica de Exhaustividad o Recall:", MetricaExhaustividadStackingClassifier)
```

```
# Calcular la Curva ROC para cada Segmento #
ProbabilidadesROCStackingClassifier =
Stacking_Classifier.predict_proba(VariablesExplicativas_Test)

FPRStackingClassifier = dict()
TPRStackingClassifier = dict()

AUCROCStackingClassifier = dict()
n_classes = ProbabilidadesROCStackingClassifier.shape[1]

for i in range(n_classes):
    FPRStackingClassifier[i], TPRStackingClassifier[i], _ =
roc_curve((VariableRespuesta_Test_Ajustada == i).astype(int),
ProbabilidadesROCStackingClassifier[:, i])
    AUCROCStackingClassifier[i] = auc(FPRStackingClassifier[i], TPRStackingClassifier[i])

# Calcular la Media de la Curva ROC #
MediaFPRStackingClassifier = np.linspace(0, 1, 100)
MediaTPRStackingClassifier = np.zeros_like(MediaFPRStackingClassifier)
for i in range(n_classes):
    MediaTPRStackingClassifier += np.interp(MediaFPRStackingClassifier,
FPRStackingClassifier[i], TPRStackingClassifier[i])
```

```
MediaTPRStackingClassifier /= n_classes

MediaAUCStackingClassifier = auc(MediaFPRStackingClassifier,

MediaTPRStackingClassifier)


# Visualizar la Curva ROC para cada Segmento #

plt.figure()

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']

for i, color in zip(range(n_classes), colors):

    plt.plot(FPRStackingClassifier[i], TPRStackingClassifier[i], color=color, lw=2, label='Clase

{0} (AUC = {1:0.2f})'.format(i+1, AUCROCStackingClassifier[i]))

plt.plot(MediaFPRStackingClassifier, MediaTPRStackingClassifier, color='black', lw=2,

linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCStackingClassifier))

plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo Stacking Classifier')

plt.legend(loc="lower right")

plt.show()
```

```
# Calcular la Matriz de Confusión #
MatrizConfusionStackingClassifier = confusion_matrix(VariableRespuesta_Test,
PrediccionesStackingClassifier)
print("Matriz de Confusión:\n", MatrizConfusionStackingClassifier)


# Visualizar la Matriz de Confusión #
def plot_confusion_matrix(cm, classes, normalize=False, title='Matriz de Confusión del Modelo
Stacking Classifier', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, format(cm[i, j], '.2f'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
```

```
        else:
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('Segmento Real de la Base de Datos')
    plt.xlabel('Segmento de la Predicción del Modelo')
plot_confusion_matrix(MatrizConfusionStackingClassifier, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], title='Matriz de
Confusión del Modelo Stacking Classifier')
plt.show()

# Visualizar la Matriz de Confusión Ponderada #
plot_confusion_matrix(MatrizConfusionStackingClassifier, classes=['Segmento1', 'Segmento2',
'Segmento3', 'Segmento4', 'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'],
normalize=True, title='Matriz de Confusión Ponderada del Modelo Stacking Classifier')
plt.show()
# Tiempo de Ejecución del Modelo #
FinStackingClassifier = time.time()
TiempoEjecucionStackingClassifier = FinStackingClassifier - InicioStackingClassifier
TiempoEjecucionStackingClassifier = TiempoEjecucionStackingClassifier / 60
print("Tiempo de ejecución:", TiempoEjecucionStackingClassifier, "minutos")
```

```
#### Comparativa entre las Alternativas de Modelización ####

## Tabla Comparativa de Métricas de Clasificación ##

TablaComparativaMetricas = {

    'Modelo': [

        'Regresión Logística',

        'Análisis Discriminante Lineal',

        'Naive Bayes',

        'Árboles de Decisión',

        'Random Forest',

        'Gradient Boosting',

        'XGBoost',

        'Redes Neuronales',

        'Voting Classifier',

        'Stacking Classifier'

    ],

    'F-Score': [

        FScoreRegresionLogistica,

        FScoreAnalisisDiscriminanteLineal,

        FScoreNaiveBayes,

        FScoreArbolesDecision,

        FScoreRandomForest,

        FScoreGradientBoosting,
```

```
    FScoreXGBoost,
    FScoreRedesNeuronales,
    FScoreVotingClassifier,
    FScoreStackingClassifier
],
'Precisión': [
    MetricaPrecisionRegresionLogistica,
    MetricaPrecisionAnalisisDiscriminanteLineal,
    MetricaPrecisionNaiveBayes,
    MetricaPrecisionArbolesDecision,
    MetricaPrecisionRandomForest,
    MetricaPrecisionGradientBoosting,
    MetricaPrecisionXGBoost,
    MetricaPrecisionRedesNeuronales,
    MetricaPrecisionVotingClassifier,
    MetricaPrecisionStackingClassifier
],
'Exactitud': [
    MetricaExactitudRegresionLogistica,
    MetricaExactitudAnalisisDiscriminanteLineal,
    MetricaExactitudNaiveBayes,
    MetricaExactitudArbolesDecision,
```

```
        MetricaExactitudRandomForest,
        MetricaExactitudGradientBoosting,
        MetricaExactitudXGBoost,
        MetricaExactitudRedesNeuronales,
        MetricaExactitudVotingClassifier,
        MetricaExactitudStackingClassifier
    ],
    'Exhaustividad': [
        MetricaExhaustividadRegresionLogistica,
        MetricaExhaustividadAnalisisDiscriminanteLineal,
        MetricaExhaustividadNaiveBayes,
        MetricaExhaustividadArbolesDecision,
        MetricaExhaustividadRandomForest,
        MetricaExhaustividadGradientBoosting,
        MetricaExhaustividadXGBoost,
        MetricaExhaustividadRedesNeuronales,
        MetricaExhaustividadVotingClassifier,
        MetricaExhaustividadStackingClassifier
    ]
}

TablaComparativaMetricas = pd.DataFrame(TablaComparativaMetricas)
```

```
TablaComparativaMetricas = TablaComparativaMetricas.sort_values(by='F-Score',
ascending=False)
print(TablaComparativaMetricas)


## Comparativa entre las Matrices de Confusión ##
MatricesConfusion = [
    (MatrizConfusionRegresionLogistica, 'Regresión Logística'),
    (MatrizConfusionAnalisisDiscriminanteLineal, 'Análisis Discriminante Lineal'),
    (MatrizConfusionNaiveBayes, 'Naive Bayes'),
    (MatrizConfusionArbolesDecision, 'Árboles de Decisión'),
    (MatrizConfusionRandomForest, 'Random Forest'),
    (MatrizConfusionGradientBoosting, 'Gradient Boosting'),
    (MatrizConfusionXGBoost, 'XGBoost'),
    (MatrizConfusionRedesNeuronales, 'Redes Neuronales'),
    (MatrizConfusionVotingClassifier, 'Voting Classifier'),
    (MatrizConfusionStackingClassifier, 'StackingClassifier')
]
fig, axs = plt.subplots(5, 2, figsize=(12, 20))
plt.subplots_adjust(hspace=0.5)
for i, (matriz, titulo) in enumerate(MatricesConfusion):
    plt.subplot(5, 2, i + 1)
```

```
    plot_confusion_matrix(matriz, classes=['Segmento1', 'Segmento2', 'Segmento3', 'Segmento4',
'Segmento5', 'Segmento6', 'Segmento7', 'Segmento8'], normalize=True, title=titulo)
plt.show()
## Comparativa entre las Curvas ROC One vs All ##
plt.figure(figsize=(15, 20))
plt.subplot(5, 2, 1)
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRRegresionLogistica[i], TPRRegresionLogistica[i], color=color, lw=2,
label='Clase {0} (AUC = {1:0.2f})'.format(i+1, AUCROCRegresionLogistica[i]))
plt.plot(MediaFPRRegresionLogistica, MediaTPRRegresionLogistica, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCRegresionLogistica))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo RegresionLogistica')
plt.legend(loc="lower right")
plt.subplot(5, 2, 2)
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
```

```
    plt.plot(FPRAnalisisDiscriminanteLineal[i], TPRAnalisisDiscriminanteLineal[i], color=color,
lw=2, label='Clase {0} (AUC = {1:0.2f})'.format(i+1,
AUCROCAnalisisDiscriminanteLineal[i]))
plt.plot(MediaFPRAnalisisDiscriminanteLineal, MediaTPRAnalisisDiscriminanteLineal,
color='black', lw=2, linestyle='-.', label='Media (AUC =
{0:0.2f})'.format(MediaAUCAnalisisDiscriminanteLineal))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo Analisis Discriminante Lineal')
plt.legend(loc="lower right")
plt.subplot(5, 2, 3)
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRNaiveBayes[i], TPRNaiveBayes[i], color=color, lw=2, label='Clase {0} (AUC =
{1:0.2f})'.format(i+1, AUCROCNaiveBayes[i]))
plt.plot(MediaFPRNaiveBayes, MediaTPRNaiveBayes, color='black', lw=2, linestyle='-.',
label='Media (AUC = {0:0.2f})'.format(MediaAUCNaiveBayes))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.0])

plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo Naive Bayes')

plt.legend(loc="lower right")


plt.subplot(5, 2, 4)

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']

for i, color in zip(range(n_classes), colors):

    plt.plot(FPRArbolesDecision[i], TPRArbolesDecision[i], color=color, lw=2, label='Clase {0}

(AUC = {1:0.2f})'.format(i+1, AUCROCArbolesDecision[i]))

plt.plot(MediaFPRArbolesDecision, MediaTPRArbolesDecision, color='black', lw=2,

linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCArbolesDecision))

plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo RandomForest')

plt.legend(loc="lower right")

plt.subplot(5, 2, 5)

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
```

```
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRRandomForest[i], TPRRandomForest[i], color=color, lw=2, label='Clase {0}
(AUC = {1:0.2f})'.format(i+1, AUCROCRandomForest[i]))
plt.plot(MediaFPRRandomForest, MediaTPRRandomForest, color='black', lw=2, linestyle='-.',
label='Media (AUC = {0:0.2f})'.format(MediaAUCRandomForest))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC One vs All del Modelo RandomForest')
plt.legend(loc="lower right")
plt.subplot(5, 2, 6)
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRGradientBoosting[i], TPRGradientBoosting[i], color=color, lw=2, label='Clase
{0} (AUC = {1:0.2f})'.format(i+1, AUCROCGradientBoosting[i]))
plt.plot(MediaFPRGradientBoosting, MediaTPRGradientBoosting, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCGradientBoosting))
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
```

```
plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo Gradient Boosting')

plt.legend(loc="lower right")

plt.subplot(5, 2, 7)

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']

for i, color in zip(range(n_classes), colors):

    plt.plot(FPRXGBoost[i], TPRXGBoost[i], color=color, lw=2, label='Clase {0} (AUC =

{1:0.2f})'.format(i+1, AUCROCXGBoost[i]))

plt.plot(MediaFPRXGBoost, MediaTPRXGBoost, color='black', lw=2, linestyle='-.',

label='Media (AUC = {0:0.2f})'.format(MediaAUCXGBoost))


plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])


plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo XGBoost')

plt.legend(loc="lower right")

plt.subplot(5, 2, 8)

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
```

```
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRRedesNeuronales[i], TPRRedesNeuronales[i], color=color, lw=2, label='Clase
{0} (AUC = {1:0.2f})'.format(i+1, AUCROCRedesNeuronales[i]))
plt.plot(MediaFPRRedesNeuronales, MediaTPRRedesNeuronales, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCRedesNeuronales))

plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])

plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo Redes Neuronales')
plt.legend(loc="lower right")
plt.subplot(5, 2, 9)
colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']
for i, color in zip(range(n_classes), colors):
    plt.plot(FPRStackingClassifier[i], TPRStackingClassifier[i], color=color, lw=2, label='Clase
{0} (AUC = {1:0.2f})'.format(i+1, AUCROCStackingClassifier[i]))
plt.plot(MediaFPRStackingClassifier, MediaTPRStackingClassifier, color='black', lw=2,
linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCStackingClassifier))
```

```
plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')

plt.title('Curva ROC One vs All del Modelo Stacking Classifier')

plt.legend(loc="lower right")

plt.subplot(5, 2, 10)

colors = ['purple', 'brown', 'orange', 'blue', 'green', 'pink', 'gray', 'cyan']

for i, color in zip(range(n_classes), colors):

    plt.plot(FPRStackingClassifier[i], TPRStackingClassifier[i], color=color, lw=2, label='Clase

{0} (AUC = {1:0.2f})'.format(i+1, AUCROCStackingClassifier[i]))

plt.plot(MediaFPRStackingClassifier, MediaTPRStackingClassifier, color='black', lw=2,

linestyle='-.', label='Media (AUC = {0:0.2f})'.format(MediaAUCStackingClassifier))


plt.plot([0, 1], [0, 1], color='black', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])


plt.xlabel('Tasa de Falsos Positivos')

plt.ylabel('Tasa de Verdaderos Positivos')
```

```
plt.title('Curva ROC One vs All del Modelo Stacking Classifier')
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()


## Comparativa entre las Medias de las Curvas ROC One vs All ##
CurvasROC = [
    (MediaFPRRegresionLogistica, MediaTPRRegresionLogistica,
MediaAUCRegresionLogistica, 'blue', 'Regresión Logística'),
    (MediaFPRAnalisisDiscriminanteLineal, MediaTPRAnalisisDiscriminanteLineal,
MediaAUCAnalisisDiscriminanteLineal, 'red', 'Análisis Discriminante'),
    (MediaFPRNaiveBayes, MediaTPRNaiveBayes, MediaAUCNaiveBayes, 'green', 'Naive
Bayes'),
    (MediaFPRArbolesDecision, MediaTPRArbolesDecision, MediaAUCArbolesDecision,
'orange', 'Arboles de Decisión'),
    (MediaFPRRandomForest, MediaTPRRandomForest, MediaAUCRandomForest, 'purple',
'Random Forest'),
    (MediaFPRGradientBoosting, MediaTPRGradientBoosting, MediaAUCGradientBoosting,
'cyan', 'Gradient Boosting'),
(MediaFPRXGBoost, MediaTPRXGBoost, MediaAUCXGBoost, 'magenta', 'XGBoosting'),
(MediaFPRRedesNeuronales, MediaTPRRedesNeuronales, MediaAUCRedesNeuronales,
'yellow', 'Redes Neuronales'),
```

```
(MediaFPRVotingClassifier, MediaTPRVotingClassifier, MediaAUCVotingClassifier, 'brown',
'Voting Classifier'),
(MediaFPRStackingClassifier, MediaTPRStackingClassifier, MediaAUCStackingClassifier,
'gray', 'Stacking Classifier')
]
CurvasROCOrdenadas = sorted(CurvasROC, key=lambda x: x[2], reverse=True)
for roc_curve in CurvasROCOrdenadas:
fpr, tpr, auc, color, label = roc_curve
plt.plot(fpr, tpr, color=color, lw=1, label='{0} (AUC = {1:0.2f})'.format(label, auc))
plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Medias de las Curvas ROC One vs All')
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

################################ FIN DEL SCRIPT ################################

REFERENCIAS BIBLIOGRÁFICAS

Bayes, T. y Price, R. (1763). An Essay towards solving a Problem in the Doctrine of Chances. Philosophical Transactions of the Royal Society of London, 53, 370-418.

Bengio, Y. et al. (2015). Deep Neural Networks for Classification: A Review. Nature, 521(7553), 436-444.

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

Breiman, L. et al. (1984). Classification and Regression Trees. Taylor & Francis Group.

Chen, T. y Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. ACM, 785-794.

Finance Yahoo. (2023). Prudential Financial Summary. Recuperado el 10 de abril del año 2023, de *https://finance.yahoo.com/quote/PRU/*

Friedman, J. H. (1999a). Greedy Function Approximation: A Gradient Boosting Machine. Stanford University.

Friedman, J. H. (1999b). Stochastic Gradient Boosting. Stanford University.

Hinton, G. et al. (1986). Learning representations by back-propagating errors. Nature, 323, 533-536.

Ho, T. K. (1995). Random Decision Forest. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14-16 August 1995, 278-282.

ICEA. (2023). Estadísticas y estudios de seguros. Recuperado el 22 de abril del año 2023, de *https://www.icea.es/es-es/informaciondelseguro/*

INE. (2023). Producto Interior Bruto por el Instituto Nacional de Estadística. Recuperado el 22 de abril del año 2023, de *https://www.ine.es/*

Kaggle. (2016). Prudential Life Insurance Dataset. Recuperado 8 de abril de 2023, de *https://www.kaggle.com/competitions/prudential-life-insurance-assessment/data/*

Madansky, A. (1959). The Multinomial Logit Model. Journal of the American Statistical Association, 54(287), 153-160.

McCulloch, W. y Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, 5(4), 115-133.

Prudential Financial. (2023). About Us, Prudential Financial. Recuperado el 18 de abril del año 2023, de *https://www.prudential.com/about/*

Python. (2023). About Python Programing Language. Recuperado el primero de mayo del año 2023, de *https://www.python.org/about/*

Schapire, R. E. et al. (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. In D. Fisher (Ed.), Machine Learning: Proceedings of the Fourteenth International Conference (pp. 322–330). Morgan Kaufmann.

Welch, B. L. (1939). Some Useful Properties of the Multivariate t Distribution, with Applications to Discriminant Analysis. Biometrics, 35(1), 110-117.

Wolpert, D. (1992). Stacked Generalization. Neural Networks: The Official Journal of the International Neural Network Society, 5(2), 241-259.