Universidad de Valladolid

Facultad de Ciencias Económicas y Empresariales

Trabajo de Fin de Grado

Grado en Administración y Dirección de Empresas

Comparación de Diferentes Técnicas para Predecir la Pobreza

Presentado por:

Álvaro Curto Merino

Tutelado por:

Mercedes Prieto Alaiz

Valladolid, 10 de Julio de 2023

RESUMEN

La capacidad de predecir la pobreza es un elemento fundamental en el estudio de este fenómeno. La falta de información fiable para determinadas regiones genera la necesidad de poder clasificar qué parte de su población se encuentra en situación de pobreza o no a partir de determinadas variables. Tradicionalmente, los métodos más usados para lograrlo han sido los modelos propuestos desde el ámbito de la Econometría. No obstante, en los últimos tiempos se ha visto la popularización de los nuevos métodos de predicción del Aprendizaje Automático.

El objetivo de este trabajo es comparar el comportamiento de diferentes métodos, tanto de la Econometría como del Aprendizaje Automático, para predecir si una persona es pobre o no. En concreto, se analizan la regresión logística, el árbol de clasificación y una red neuronal de una sola capa, utilizando los datos de la Encuesta de Condiciones de Vida del año 2022. Los resultados muestran que todas las técnicas tienen un comportamiento similar en términos de capacidad de predicción.

Palabras clave:

Pobreza, Clasificación, Econometría, Aprendizaje Automático

ABSTRACT

The ability to predict poverty is a key element in the study of this phenomenon. The lack of reliable information for certain regions generates the need to be able to classify which part of the population is in a situation of poverty or not based on certain variables. Traditionally, the most commonly used methods to achieve this have been the models proposed in the field of econometrics. However, in recent times we have seen the new prediction methods of Machine Learning come into the spotlight.

In this paper, the Spanish Living Conditions Survey of the year 2022 is used as a data source to compare the performance of different forecasting methods from both Econometrics and Machine Learning. The models to be considered within econometrics are the logit and probit, while from machine learning we take logistic regression, classification tree and a single-layer neural network. The results show that all the techniques have a similar behavior in terms of predictive capacity.

Keywords:

Poverty, Classification, Econometrics, Machine Learning

Clasificación JEL:

I32 Bienestar y pobreza, Medición y análisis de la pobreza

C53 Modelización econométrica. Predicción y otras aplicaciones de modelos

C45 Redes neuronales y temas relacionados

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

1.1. Justificación

Combatir la pobreza es uno de los mayores retos a los que se enfrenta la humanidad para su progreso. Así, el primer Objetivo de Desarrollo Sostenible adoptado por los principales líderes del mundo en las Naciones Unidas es la erradicación de la pobreza extrema para todas las personas del mundo para el año 2030 (Belver García, 2020).

El estudio de la pobreza es un campo extenso y complejo que trata de solucionar un problema en el que no existe una clara causalidad. Dentro de este campo, resulta importante ser capaces de predecir si una persona es pobre o no a partir de ciertas variables que generalmente se han asociado con la pobreza. En este sentido, las técnicas de predicción estadística juegan un papel clave, siendo los modelos de elección discreta procedentes de la Econometría, la metodología más empleada hasta ahora en el análisis de la pobreza (Haughton & Khandker, 2009).

De modo paralelo, durante los últimos años, el Aprendizaje Automático o *Machine Learning* (ML) ha cobrado un gran protagonismo en la literatura académica, llegando incluso a dejar en segundo plano a los modelos econométricos tradicionales en la mayoría de las aplicaciones. Si bien la comunidad estadística ya ha integrado los modelos de Aprendizaje Automático junto con sus métodos estadísticos más tradicionales (véase Hastie et al., 2009; Efron & Hastie, 2016), su adopción en el ámbito de la economía ha sido más lenta y es actualmente objeto de una literatura metodológica en rápido crecimiento (Athey & Imbens, 2019).

Dentro de la investigación económica moderna, una de las áreas donde menos introducido se encuentra el Aprendizaje Automático es en la investigación de la pobreza. Para poder explicar la actual carencia de literatura sobre pobreza que emplee el Aprendizaje Automático, se hace necesario hacer un estudio que compare estas nuevas técnicas de predicción con los presentes modelos econométricos.

1.2. Interés y relevancia

El tema del Aprendizaje Automático, ya sea por sus aplicaciones en medicina y salud, en banca y finanzas o en asistentes virtuales, es un tema ahora más vigente que nunca. No obstante, y pese al actual furor en implementar aspectos de inteligencia artificial en todos los aspectos de la sociedad, existen áreas, como el estudio de la pobreza, donde su efectividad no está aun debidamente probada.

La predicción de la pobreza, tradicionalmente hecha mediante técnicas econométricas de clasificación, puede ver mejorar sus resultados con las técnicas de predicción del Aprendizaje Automático. Aunque ambas, Econometría y Aprendizaje Automático, se han ido desarrollando en paralelo, las dos permiten construir modelos predictivos para servir al mismo fin. Es por ello que, antes de sustituir la Econometría por el Aprendizaje Automático, hay que estudiar las fortalezas y debilidades de cada enfoque, y comprobar si realmente el cambio supone una mejora notable.

1.3. Antecedentes

Uno de los primeros ejemplos de incorporación de algoritmos de ML al servicio del estudio de la pobreza fue el concurso convocado por el Banco Mundial para la elaboración del mejor algoritmo para la predicción de pobreza en una base de datos de tres países (DrivenData, 2018). Otras investigaciones similares al respecto en materia de pobreza son comparativas entre diversos modelos de ML para analizar el impacto del COVID-19 sobre la pobreza (Kumar Satapathy et al., 2023); para estudiar la pobreza combinando diferentes fuentes de datos (Zixi, 2021) o para evaluar el nivel de pobreza en Costa Rica (Min et al., 2022).

Todas estas investigaciones parten directamente del ámbito del Aprendizaje Automático y simplemente emplean datos relacionados con la pobreza sin tener en cuenta la investigación ya realizada sobre la pobreza hecha con métodos estadísticos econométricos. El debate entre Econometría y Aprendizaje Automático sí está bien extendido y documentado, especialmente en el análisis de predicción crediticia (véase Lessman et al., 2015; Gunnarsson et al., 2021). Otros trabajos de referencia en la transición entre Econometría y Aprendizaje Automático en la economía han sido: Jarmulska (2020), Charpentier et al. (2018), Athey & Imbens (2019) y Varian (2014).

1.4. Objetivos

Este trabajo de fin de grado tiene como propósito principal comparar las técnicas clásicas de Econometría y las más recientes de Aprendizaje Automático para predecir si una persona es pobre o no. En concreto, los modelos logit y probit de la Econometría; y el árbol de clasificación y la red neuronal de una sola capa del ML. Los resultados muestran que no hay una clara superioridad de una técnica sobre otra.

Para lograr este objetivo, en la Sección 2 se clarificará el concepto de pobreza utilizado y se analizarán las variables que en la literatura han estado asociadas con la pobreza. La Sección 3 busca explicar algunas de las diferencias primordiales entre el campo de la Econometría y el Aprendizaje Automático, para acto seguido describir las técnicas cuya efectividad se va a estudiar en la predicción de datos. En la siguiente Sección, se describe la muestra de datos

empleada para el estudio, procedente de la Encuesta de Condiciones de Vida del año 2022, además de los factores finalmente escogidos para predecir la pobreza. La Sección 5 muestra los resultados obtenidos al predecir la pobreza con los diferentes modelos y la última Sección muestra las conclusiones más relevantes.

2. LA POBREZA Y SUS DETERMINANTES

2.1. ¿Qué es la pobreza?

La pobreza ha sido siempre una manifestación de desigualdad e injusticia entre la población cuya mera existencia es uno de los mayores problemas que la humanidad arrastra para desarrollar todo su potencial. Representa un fenómeno complejo para el cual no existe consenso ni en su definición ni en las dimensiones que realmente abarca.

Una de las definiciones más aceptadas para la pobreza es la del Banco Mundial (Haughton & Khandker, 2009), que afirma que la pobreza consiste en la privación pronunciada de bienestar de las personas. Dependiendo de la forma en la que se aproxime el bienestar, se distinguen dos enfoques para analizar la pobreza.

El enfoque unidimensional se basa en aproximar el bienestar mediante una sola variable, denominada genéricamente renta, que refleje la posición económica de los individuos (Foster et al., 1984). Variables como los ingresos, los gastos o la riqueza generalmente se han asociado con el término renta. Por otro lado, el enfoque multidimensional de la pobreza considera el bienestar como un fenómeno que depende no sólo de la renta, sino de otras dimensiones como la salud, la educación, o el acceso a diferentes servicios públicos. Si bien el análisis unidimensional de la pobreza a menudo pasa por alto la heterogeneidad de las experiencias y necesidades de los individuos (Bourguignon, 2003); su sencillez, versatilidad y uso extendido tal y como resalta Ravallion (2011) hacen que en este trabajo el enfoque a seguir sea el unidimensional.

Dentro del enfoque unidimensional, una de las cuestiones metodológicas más discutidas es la fijación de una renta, denominada línea o umbral de pobreza, por debajo de la cual se considera que una persona es pobre. Se pueden diferenciar dos tipos de umbral. Por un lado, la línea de pobreza absoluta separa pobres y “no pobres” a partir de un nivel crítico de renta por debajo del cual las necesidades básicas del individuo no se encuentran cubiertas y existe una carencia de bienes y servicios elementales (Addison & Hulme, 2005). Una de las características de las líneas de pobreza absoluta es que sus resultados son sensibles al desarrollo económico, es decir, un aumento de la renta en la población supone una reducción en el porcentaje de pobres, incluso si la distribución de ingresos se hace de forma totalmente homogénea (Hulme, 2010).

Otro tipo de umbral utilizado es la línea de pobreza relativa, que se fija en función de la distribución de la renta y, a diferencia de los umbrales absolutos,

sitúa a los individuos en relación con la sociedad que les rodea. Así, una persona se considera en situación de pobreza relativa cuando sus ingresos o recursos se encuentran significativamente por debajo de los estándares o niveles medios de vida de la sociedad en la que se encuentra (Hulme, 2010). En la pobreza relativa a medida que aumenta la renta mediana de un país, el umbral de pobreza tiende a aumentar también, ya que "los recursos mínimos necesarios para participar plenamente en la sociedad probablemente aumenten con el tiempo" (Haughton & Khandker, 2009).

Una de las líneas de pobreza absoluta más usadas es la establecida en 1$ diario de ingresos medido a precios de 1985 y ajustado al poder de compra local (Ravallion et al., 1991), línea hoy en día actualizada a 2,15$ diarios según el poder de compra de 2017 (Castaneda Aguilar et al., 2022). Las líneas de pobreza relativa en cambio se suelen establecer como un porcentaje de la media o la mediana de los ingresos o gastos (Foster et al., 1984). Ejemplos destacados de umbral relativo son el de la OCDE, establecido en el 50% de la mediana de los ingresos de la población de un país (OECD, 2008), o el utilizado en la Unión Europea, en el 60% de la mediana nacional de renta disponible por unidad de consumo equivalente (Eurostat, 2021).

Este último es el umbral que va a ser usado en este trabajo para clasificar a los pobres de los no pobres. La UE emplea esta línea de pobreza para construir el indicador de la tasa de riesgo de pobreza relativa (o “At-risk-of-poverty rate”), que mide la proporción de personas de un país que se encuentran por debajo de dicha línea de pobreza (Eurostat, 2021).

2.2. Los determinantes de la pobreza

En la predicción de la pobreza, las técnicas de clasificación tanto econométricas como de Aprendizaje Automático permiten analizar la asociación que existe entre diferentes factores y el hecho de si una persona es pobre o no (Haughton & Khandker, 2009). De acuerdo con la literatura en pobreza, se pueden distinguir tres conjuntos de factores que se vinculan con la pobreza:

- Características regionales: la región donde habita una persona afecta a sus posibilidades en la vida y en algunos casos, las diferentes condiciones de vida entre localizaciones geográficas pueden explicar la pobreza de algunos individuos.
- Características del hogar: aparte del emplazamiento geográfico, variables como la cantidad de habitantes por habitación, o la distribución de ingresos entre los miembros del hogar son síntomas de una peor calidad de vida.

- Características personales: existen múltiples variables propias al individuo que pueden ser usadas para predecir si es o no pobre. Variables como su salud, su edad, su sexo o sus estudios terminados son algunas de ellas.

Es importante señalar que a menudo la elección de los determinantes a usar en la predicción de la pobreza queda limitada por los datos disponible, ya sea por su calidad o por la dificultad de ser obtenidos. En la Sección 4, se señalan los determinantes finalmente escogidos en el trabajo en función de su disponibilidad en la Encuesta de Condiciones de Vida, base de datos utilizada.

Del mismo modo, es importante reiterar que el objetivo de este trabajo es predecir si una persona es pobre o no en función de una serie de variables. Por lo tanto, no se trata de realizar un análisis causal que precisaría de una especificación diferente de los modelos econométricos y de Aprendizaje Automático.

3. METODOLOGÍA

Las técnicas disponibles actualmente en la predicción comprenden tanto las del campo de la Econometría como las del Aprendizaje Automático (o *Machine Learning*, ML). Se trata de dos perspectivas diferentes desarrolladas en paralelo que comparten el mismo objetivo, que es la creación de modelos predictivos para una variable de interés a partir una serie de variables explicativas (Charpentier et al., 2018), pero con diferentes preocupaciones.

La Econometría trata de cuantificar las relaciones que se dan entre variables económicas y financieras con tres objetivos: 1) estimar dichas relaciones, 2) predecir el valor de una variable a partir de los valores de otras y 3) analizar las relaciones de causalidad entre las variables (Wooldridge, 2010). Dada una muestra aleatoria de la población de interés, los modelos econométricos que se plantean para alcanzar estos objetivos parten de una hipótesis sobre la distribución conjunta de un conjunto de variables y se buscan estimadores de los parámetros con buenas propiedades en términos de sesgo y varianza (Athey & Imbens, 2019). Un ejemplo sencillo de modelo econométrico puede ser la descripción de la esperanza de una variable dependiente Y_i condicionada a un conjunto de k variables regresoras $X_{1i}, X_{2i}, \dots, X_{ki}$. Se asume que:

$$Y_i / X_{1i}, X_{2i}, \dots, X_{ki} \sim N(X_i'\beta, \sigma^2)\ , i = 1, \dots, N$$

Siendo $X_i' = (1, X_{1i}, X_{2i}, \dots, X_{ki})$ el vector de la observación i de las variables independientes; $\beta' = \beta_0, \beta_2, \dots, \beta_k$ los coeficientes del modelo econométrico y N el conjunto de observaciones. Un criterio para estimar los coeficientes es el método de mínimos cuadrados ordinarios (MCO).

$$\min_{\beta} \sum_{i=1}^{N} (Y_i - X_i'\beta)^2$$

El Aprendizaje Automático, en cambio, profundiza en la construcción de modelos no necesariamente paramétricos construidos casi exclusivamente a partir de datos (sin hipótesis de distribución), centrados en la predicción de unas variables dadas a partir de otras (Charpentier et al., 2018). Los algoritmos de Aprendizaje Supervisado, es decir, aquellos dentro del *Machine Learning* en los que dentro del conjunto de variables hay una variable dependiente y otras son independientes (Abu-Mostafa et al., 2012), son especialmente eficaces para predecir y capturar relaciones no lineales en los datos (Flachaire et al., 2022). En los últimos años los modelos de Aprendizaje Supervisado están dando lugar mejores resultados que los modelos econométricos tradicionales y, sobre todo, una mayor eficiencia al manejar volúmenes de datos enormes con gran número de variables independientes (Varian, 2014).

Un ejemplo sencillo de modelo de Aprendizaje Supervisado es la predicción del valor Y_{N+1} de un individuo los valores regresores X_{N+1} mediante un predictor lineal, de modo que tenemos:

$$\hat{Y}_{N+1} = X'_{N+1}\hat{\beta}$$

Siendo $X'_{N+1} = (1, X_{1N+1}, X_{2N+1}, \dots, X_{kN+1})$ el vector de variables regresoras del individuo $N+1$, y $\hat{\beta}' = \hat{\beta}_0, \hat{\beta}_2, \dots, \hat{\beta}_k$ los coeficientes del estimador de ML. La estimación de los mejores coeficientes para cada modelo de Aprendizaje Automático se denomina entrenamiento, y se realiza de acuerdo con una determinada función de pérdidas, como puede ser la minimización del error cuadrático medio.

$$\min_{\hat{\beta}'} (Y_{N+1} - \hat{Y}_{N+1})^2$$

Tanto en Econometría como en Aprendizaje Supervisado, cuando la variable dependiente es cuantitativa se habla de problema de regresión, mientras que cuando es discreta se habla de clasificación (Charpentier et al., 2018). En este trabajo, la variable dependiente es si una persona es pobre o no, variable claramente discreta con dos valores posibles, por lo que las técnicas de predicción de Econometría y ML a comparar en este estudio van a consistir en modelos econométricos de clasificación binaria. En concreto, los modelos a comparar van a ser el logit, el probit, el árbol de clasificación y las redes neuronales.

3.1. Modelos logit y probit

En la Econometría, los modelos de clasificación más utilizados son el modelo logístico o logit y el modelo probit, los cuales constituyen la forma estándar de analizar los factores que determinan la pobreza (Haughton & Khandker, 2009).

El precursor de los modelos logit y probit es el Modelo de Probabilidad Lineal (MPL), en el que la variable dependiente Y_i solo puede tomar los valores cero y uno (Wooldridge, 2010). El MLP viene dado por la siguiente ecuación:

$$Y_i = X'_i\beta + e_i \quad i = 1 \dots n$$

donde Y_i toma el valor 1 o 0, X'_i es la observación iésima del conjunto de regresores y β es el vector de coeficientes. Asumiendo que la esperanza del error condicionado es nula, $E(e_i|X'_i) = 0$, se demuestra que la probabilidad de que la variable dependiente sea igual a uno es:

$$E(Y_i|X'_i) = P(Y_i = 1|X'_i) = p_i = X'_i\beta$$

La estimación de los coeficientes β puede realizarse por MCO, sin embargo, este modelo presenta una serie de problemas (Parra, 2019).

- La perturbación aleatoria e_i no sigue una distribución normal sino una Bernoulli. Este problema se atenúa en tamaños de muestras grandes.
- La perturbación aleatoria e_i es heterocedástica al no tener una varianza constante. También es posible solucionarlo estimando por mínimos cuadrados ponderados.
- Las predicciones de la probabilidad pueden dar valores fuera del intervalo [0,1].

Los modelos logit y probit aparecen como solución a este último problema, haciendo uso de funciones de distribución $F(z)$ para acotar los posibles resultados del MPL entre 0 y 1.

$$Y_i = F(X_i'\beta) + e_i \quad i = 1 \dots n$$

En el modelo logit la función de distribución $F(z)$ es una distribución logística de parámetros α=0 y β=1 con esperanza 0 y desviación típica $\frac{\pi}{\sqrt{3}}$=1,814. Así, en el modelo logit,

$$E(Y_i|X_i') = P(Y_i = 1|X_i) = p_i = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

donde $z = X_i'\beta$. En el modelo probit, $F(z)$ es una función de distribución $\phi \sim N(0,1)$ lo que conduce a

$$E(Y_i|X_i') = P(Y_i = 1|X_i) = p_i = \phi(z) = \int_{s=-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{s^2}{2}} ds$$

donde $z = X_i'\beta$.

Las funciones de distribución normal y logística son muy similares, pero la distribución normal se acerca ligeramente más rápido a los ejes que la logística tal y como se puede apreciar en la Ilustración 3.1.

La aplicación de los modelos logit y probit da lugar a la estimación de la probabilidad de que los individuos tengan el valor 1 en vez de 0. Así, si la probabilidad estimada de que un individuo pertenezca al grupo 1 supera un determinado valor se clasificaría en el grupo 1 y en el resto de los casos se clasificaría en el grupo 0. En los modelos que cuentan con el mismo número de observaciones para el grupo 0 y el grupo 1 (muestras de datos equilibradas) y en los que la penalización por equivocación es idéntica para ambas clases se suele usar un umbral de 0.5 de probabilidad.

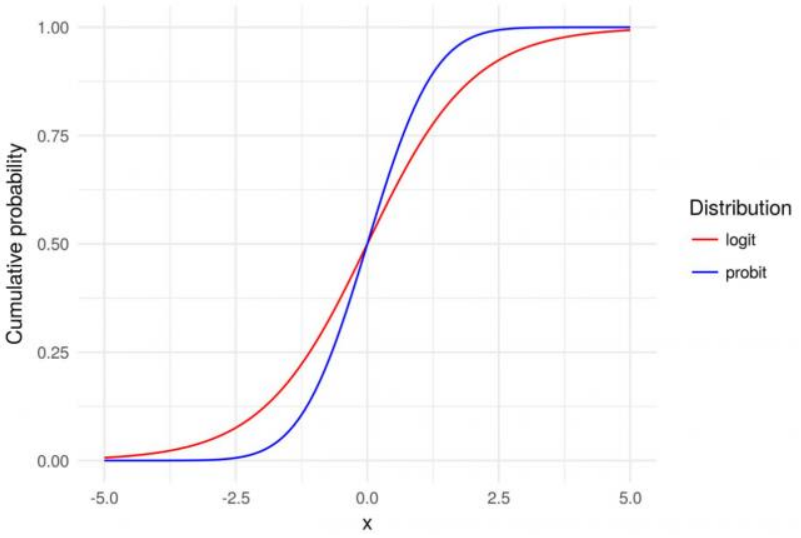


ILUSTRACIÓN 3.1. DIFERENCIAS ENTRE LA DISTRIBUCIÓN NORMAL Y LA DISTRIBUCIÓN LOGÍSTICA. FUENTE (KUMAR, 2023)

No obstante, cuando el coste de una clasificación errónea es diferente entre los grupos 1 y 0, es posible cambiar dicho umbral para que solamente se clasifiquen en el grupo 1 los individuos que presentan una mayor o menor probabilidad de serlo. En nuestro contexto, las clasificaciones de pobre o no pobre son igual de relevantes, por lo que en principio mantendremos el umbral en la probabilidad del 50%.

Debido a la no linealidad de este modelo, la estimación de sus parámetros no puede hacerse mediante MCO y se suele estimar por Máxima Verosimilitud (Wooldridge, 2010). Esta estimación da lugar al cálculo de estimadores eficientes (con la menor varianza) y consistentes (Stock & Watson, 2012).

Los modelos probit y logit con frecuencia llegan a resultados similares, siendo el modelo logit más usado en general debido a su simplicidad y ligeramente mayor robustez ante valores atípicos. No obstante, debido a que los economistas tienden a favorecer el supuesto de normalidad para la distribución de los errores e_i, en Econometría el modelo probit es más popular que el logit (Wooldridge, 2010).

Comentar por último que el modelo logit es también uno de los más populares dentro del Aprendizaje Automático, donde es conocido como el modelo de regresión logística (Abu-Mostafa et al., 2012). El entrenamiento de los modelos de regresión logística en el ML se realiza mediante técnicas de optimización numérica que permiten el uso de diferentes algoritmos de optimización (*solvers*) y fuerzas de penalización (hiperparámetro C) (scikit-learn, 2023).

Los hiperparámetros como la C mencionada, son atributos de configuración de los modelos de Aprendizaje Automático empleados en el

proceso de entrenamiento, cuyo valor no se obtiene de los datos. La elección de los mejores valores para los hiperparámetros se hace de forma previa al entrenamiento del modelo, luego es necesario calcular diferentes entrenamientos para cada posible valor de los hiperparámetros y comparar qué valores dan los mejores resultados (Raschka & Mirjalili, 2019).

3.2. Árboles de clasificación

Los árboles de decisión son modelos de predicción no paramétricos en forma de diagramas de múltiples bifurcaciones anidadas en forma de árbol. Cada nodo del árbol representa una decisión basada en una característica del conjunto de datos y las ramas los posibles resultados de esta decisión. Los árboles de decisión se conocen como árboles de clasificación si al final de cada rama se obtiene el grupo al que pertenece cada individuo, o árboles de regresión si se obtiene un valor escalar que aproxima el valor de la variable dependiente (Parra, 2019).

Los algoritmos de creación de árboles de decisión o de “segmentación recursiva” son principalmente el CHAID (Chi-Square Automatic Interaction Detector), el QUEST (Quick Unbiased Effiient Statistical Tree) y el CART (Classification and Regression Trees), siendo este último el que se va a emplear en este trabajo debido a su mayor sencillez y facilidad de implementación. El algoritmo CART, diseñado por Breiman y Friedman (1984), permite la generación de árboles de clasificación binarios, de modo que cada nodo se divide en exactamente dos ramas.

Los árboles de clasificación con el algoritmo CART se componen de nodos, ramas y hojas. Los nodos, comenzando desde el nodo raíz, consisten en diferentes condiciones de respuesta binaria (yes-no, if-else, true-false) sobre las variables independientes para determinar la siguiente rama a seguir. Cada rama da lugar a nuevos nodos y al final del árbol de clasificación, las últimas ramas seguidas dan lugar a las hojas, que representan las posibles clasificaciones para el individuo estudiado.

Para la creación de cada nodo de los árboles de clasificación, se estudian posibles subdivisiones de la muestra a partir de diferentes combinaciones de variables y umbrales, para elegir aquella que produzca las mejores subdivisiones según una función de impureza. La cantidad de opciones que se pueden comparar en la elección de cada nodo es tan elevada que por lo general se toman para comparar un conjunto de posibles variables y umbrales de forma aleatoria, luego es raro que dos predicciones consecutivas generen el mismo árbol de clasificación.

Las funciones de impureza en los árboles de clasificación sirven para medir la calidad de las subdivisiones de la muestra en cada nodo y la

homogeneidad de los conjuntos de datos generados. La impureza hace referencia a la probabilidad de que un individuo sea clasificado de forma incorrecta tras pasar por un nodo, por lo que interesa que las subdivisiones generadas den lugar a valores pequeños de la función de impureza. Las funciones de impureza más comunes para árboles de clasificación son la impureza de Gini, la entropía y el índice de ganancia de información. Las tres dan generalmente resultados similares, pero en el algoritmo CART se suele usar la impureza de Gini (Jarmulska, 2020). La impureza de Gini para un nodo w viene dada por

$$Gini(w) = \sum_{k=1}^{K} p_{wk}(1 - p_{wk}) = 1 - \sum_{k=1}^{K} (p_{wk})^2$$

donde p_{wk} se corresponde con la proporción de individuos pertenecientes a una clase k en el nodo w (en nuestro caso k sería pobre o no pobre), para K clases (dos en nuestro caso). Un valor del índice más próximo a cero indica una mayor homogeneidad en las dos ramas del árbol generadas y, por lo tanto, una mejor clasificación en el nodo.

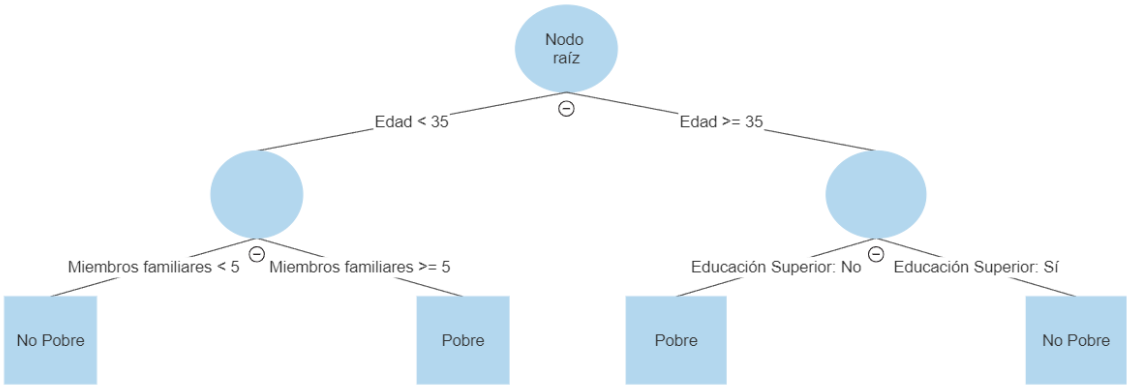


ILUSTRACIÓN 3.2. ÁRBOL DE CLASIFICACIÓN CART

En la Ilustración 3.2**¡Error! No se encuentra el origen de la referencia.** se muestra un ejemplo de árbol de clasificación CART creado para este trabajo en el que se determina si un individuo es pobre o no en función de su edad, sus miembros familiares y su nivel educativo. En la primera etapa del árbol, se comparan diferentes posibilidades y se obtiene que la variable edad con el umbral de 35 años es aquella que da una menor impureza de Gini, es decir, es la variable y umbral de entre los probados que mejor separa los individuos de la muestra en los dos grupos que recoge la variable dependiente (pobre y no pobre). En la siguiente etapa del árbol, se repite el proceso probando diferentes alternativas para obtener que, para los menores de 35 años, la mejor división se produce dependiendo de si el número de miembros familiares es menor o

mayor/igual que 5, y para los mayores/igual a 35 años la mejor clasificación de pobreza se estima según el nivel educativo.

El tamaño del árbol de clasificación puede extenderse hasta el infinito dando lugar a un problema conocido como sobreajuste. Este es un fenómeno que se da cuando un modelo se ciñe demasiado a los datos con los que se ha entrenado y no es capaz de generalizar correctamente con nuevos datos (Varian, 2014). Para la detención del proceso se definen diferentes hiperparámetros como un tamaño máximo de etapas del árbol, o un número mínimo de individuos por rama.

Los árboles de clasificación CART son especialmente eficaces en problemas con fuertes relaciones no lineares e interacciones entre variables, además de tener una gran interpretabilidad, pudiendo fácilmente ver y entender las decisiones tomadas por el modelo. No obstante, sus principales desventajas son que tienden a un rápido sobreajuste del modelo a los datos de entrenamiento y que las predicciones no siempre son demasiado precisas. Para solventar estos problemas, surgen los métodos de combinación de modelos, siendo los más conocidos el *boosting* y el *bagging*, dentro del cual se encuentran los Random Forest (Parra, 2019). En este trabajo no se van a abordar estos métodos, pero es interesante reseñar que numerosos trabajos en Economía recientes encuentran los mejores resultados para una predicción de propósito general en variantes de *boosting* y *bagging* (Gunnarsson et al., 2021; Min et al., 2022).

3.3. Redes neuronales

Las redes neuronales son un modelo de predicción semiparamétrico basado en un sistema de nodos o “neuronas” interconectadas entre sí con el objetivo de aprender y reconocer patrones en los datos mediante ensayos repetidos para organizarse mejor a sí mismas y mejorar las predicciones.

Existen actualmente más de 40 paradigmas de redes neuronales artificiales, pero el modelo más utilizado es el Modelo Perceptrón Multicapa (MLP). El MLP abarca el 70% de las aplicaciones prácticas de redes neuronales debido a su demostrada validez como aproximación universal de funciones (Parra, 2019). No obstante, para poder entender cómo opera el MLP, es necesario ver en primer lugar el funcionamiento de la red neuronal más simple que existe: el perceptrón de una sola capa.

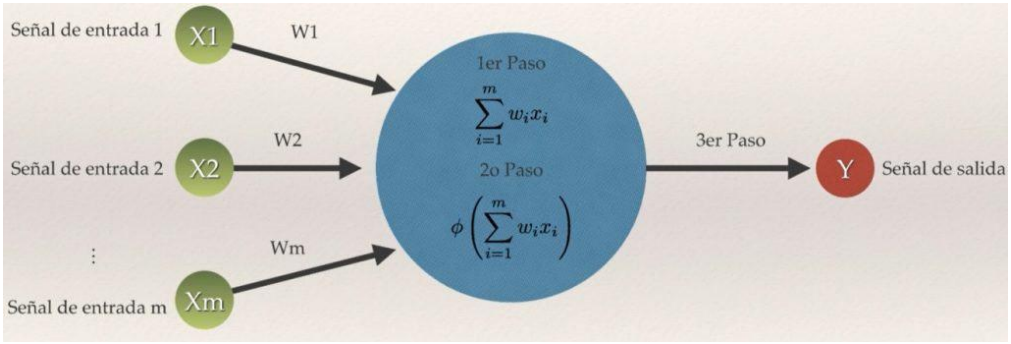


ILUSTRACIÓN 3.3. PERCEPTRÓN DE UNA SOLA CAPA (VILLANUEVA GARCÍA, 2020)

El perceptrón es un modelo de regresión o clasificación de dos etapas donde a partir de una serie de señales de entrada se obtiene una señal de salida (Hastie et al., 2009). Tal y como se muestra en la Ilustración 3.3, las señales de entrada o *inputs* $(x_1, \dots x_m)$ son en primer lugar agregadas en una suma ponderada mediante una serie de pesos sinápticos ajustables para cada *input* $(w_1, \dots w_m)$. A la combinación lineal de las señales y sus pesos sinápticos se le aplica una operación matemática o función de activación $\phi(.)$, para dar lugar a una señal de salida (Caparrini, 2022).

Existen un gran número de diferentes funciones de activación que se pueden utilizar dependiendo del tipo de salida que se desee. No obstante, las cuatro funciones más típicas son las mostradas en la **¡Error! No se encuentra el origen de la referencia.**.

Función escalón (threshold)	Devuelve un 0 si la combinación lineal es negativa, y 1 si es positiva o igual a cero	$\phi(x) = \begin{cases} 0 \; si \; x < 0 \\ 1 \; si \; x \geq 0 \end{cases}$
Función sigmoide	Idéntica a la utilizada en los modelos logit o de regresión logística. Acota los valores entre 1 y 0	$\phi(x) = \frac{1}{1 + e^{-x}}$
Función rectificadora (ReLU)	Devuelve el valor de la combinación lineal si esta es mayor o igual a 0, y devuelve 0 si esta tiene un valor negativo	$\phi(x) = \max(x, 0)$
Función tangente hiperbólica	Similar a la sigmoide, pero para acotar los valores entre 1 y -1	$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$

TABLA 3.1. FUNCIONES DE ACTIVACIÓN MÁS EMPLEADAS. FUENTE: (VILLANUEVA GARCÍA, 2020)

El MLP o perceptrón multicapa corresponde al encadenamiento de la salida de sucesivos nodos o positrones en diferentes capas, habiendo como mínimo 3: la capa de entrada con las variables de entrada de la muestra, la capa de salida con el resultado final y una o varias capas intermedias u ocultas, tal y como se muestra en la Ilustración 3.4.

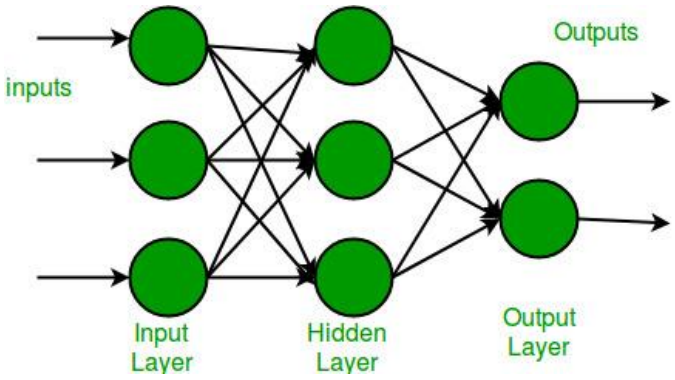


ILUSTRACIÓN 3.4. RED NEURONAL MLP DE UNA SOLA CAPA OCULTA (SWARNIMRAI, 2021)

Una de las principales cualidades de las redes neuronales es su capacidad de aprender y corregirse a sí mismas para minimizar el error de sus clasificaciones (Hinton et al., 1986). El método más usado para minimizar el error es el descenso de gradiente, también llamado retropropagación en el contexto de redes neuronales. En este método, se parte de una configuración de pesos aleatorios para todos los nodos del MLP y se calcula el error obtenido en la salida para una observación (error 1 o 0 en nuestro caso de clasificación binaria). El error se propaga hacia atrás en el MLP ajustando los pesos sinápticos de los diferentes nodos y con estos pesos actualizados se repite el proceso para la siguiente observación hasta entrenar con todas (Goodfellow et al., 2016).

Cuando las redes neuronales poseen más de una capa oculta se habla de Deep Learning. La complejidad de estos modelos aumenta exponencialmente a medida que el número de capas intermedias aumenta. Si bien un número superior de capas intermedias puede ofrecer mejores resultados, el entrenamiento de estos modelos requiere de tiempo y recursos computacionales muy elevados, los cuales sobrepasan el enfoque de este trabajo, por lo que nos limitaremos a la red neuronal MLP de una sola capa, comparando la efectividad de los hiperparámetros relativos al tamaño (número de nodos) de la capa oculta y a la fuerza del término de regularización L2 (scikit-learn, 2023).

4. DATOS Y DESCRIPCIÓN DE LAS VARIABLES EMPLEADAS

Los datos utilizados en este trabajo provienen de la Encuesta de Condiciones de Vida (ECV) del INE del año 2022 con base al año 2013. La muestra contiene observaciones acerca de un total de 24313 hogares y de 50147 personas. La ECV, de carácter anual, supone la principal fuente de referencia en España para estudiar la distribución de ingresos y la exclusión social (Instituto Nacional de Estadística, 2023). Esta encuesta es la adaptación española de la encuesta EU-SILC (European Union Statistics on Income and Living Conditions), de obligado cumplimiento para los países de la Unión Europea para asegurar la comparabilidad de datos entre los estados miembros y evaluar los objetivos de la política social europea relacionados con la pobreza. La ECV establece una metodología común y con criterios armonizados para todos los países europeos, actualizados por última vez en 2019 (Eurostat, 2023).

La ECV por ordenanza de la EU-SILC comprende una serie de variables recogidas anualmente englobadas en temas centrales como la renta, la actividad económica, la demografía o la educación, además de algunas recogidas cada cierto periodo en módulos (Eurostat, 2023). Por ejemplo, en 2022, se incluyeron el módulo trianual relativo a la salud y el módulo sexenal de la calidad de vida (Eurostat, 2023).

Los datos de la ECV están organizados en cuatro ficheros: el fichero D, con información básica del hogar; el H, con datos detallados del hogar; el R, con datos básicos de las personas y el P, con datos detallados de los adultos. De entre estos ficheros, el R no va a ser usado debido a que sus variables no resultan de interés para el estudio.

Los ficheros D y H hacen referencia a los mismos hogares representados mediante una variable de identificación transversal del hogar (‘DB030’ y ‘HB030’ respectivamente). De igual manera, el fichero H incluye una variable que muestra el identificador transversal de la primera persona responsable de la vivienda (‘HB080’), identificador que se corresponde con el identificador de cada individuo de la variable (‘PB030’) en el fichero P.

El estudio de la pobreza en los hogares va a ser realizado según las características tanto de los hogares como del principal adulto proveedor de la vivienda. Por ello, las variables de identificación son de especial importancia en la preparación de los datos y van a servir para fusionar los datos relativos al mismo hogar y su adulto responsable en un mismo fichero.

Para determinar si un hogar se encuentra en estado de pobreza o no, se calcula su renta disponible por unidad de consumo mediante el cociente entre la renta disponible total del hogar (variable ‘HY020’) y el número de unidades de

consumo del hogar medidas en la escala OCDE modificada ('HX240')[1], ambas variables presentes en el fichero H. Los hogares clasificados como pobres (con valor 1 en una nueva variable "**Pobre**") son aquellos cuya renta disponible por unidad de consumo se encuentra por debajo de un umbral de pobreza establecido en el 60% de la mediana de dicha renta. Todos aquellos hogares con una renta por encima del umbral son clasificados como "no pobres" y poseen el valor 0 en la variable "Pobre".

De acuerdo con los determinantes vistos en la Sección 2.2 y teniendo en cuenta los datos disponibles en la ECV, el resto de los factores a usar en los modelos de predicción son:

- **Familia_Numerosa**: Variable ficticia que toma el valor 1 si en el hogar conviven más de 4 personas, y 0 en el resto de los casos. Calculada mediante la variable 'HB120' del fichero H referente al número de habitantes del hogar.
- **Viv_PagadaOCedida**: Variable ficticia que vale 1 si el hogar está en propiedad completamente pagada o en cesión gratuita, y 0 si está en régimen de alquiler o supone gastos de hipoteca. Se calcula mediante la variable 'HH021' del fichero H.
- **Hacinamiento**: Ratio entre el número de habitantes del hogar (variable 'HB120') y las habitaciones disponibles ('HH030').
- **Ninos_Dep**: Variable ficticia que resume las diferentes opciones de la variable 'HX060' distinguiendo si el hogar tiene niños dependientes económicamente (1) o no (0).
- **Region_Andalucía-Region_Valencia**: Conjunto de 18 variables ficticias correspondientes a las 17 Comunidades Autónomas y a las ciudades autónomas de Ceuta y Melilla de forma conjunta. Las variables son generadas a partir de la variable 'DB040' que indican si el hogar se encuentra en una determinada comunidad autónoma (1) o no (0). En el caso de los modelos econométricos, se suprime la variable "Region_Andalucia" quedando como categoría base.
- **Urbanizacion**: Variable ficticia con información de la variable 'DB100' sobre el grado de urbanización de la zona donde se encuentra la vivienda.

[1] El concepto de unidades de consumo de un hogar sirve para tener en cuenta las economías de escala que se producen según el número y las edades de las personas que comparten los gastos del hogar. Existen múltiples posibles escalas de equivalencia para el cálculo del valor de cada unidad de consumo, pero la empleada por la UE es la de la OCDE modificada. Su funcionamiento de acuerdo con el INE (Instituto Nacional de Estadística, 2016b) es el siguiente:

$$Nº\ de\ u.c. = 1 + 0{,}5 * (A - 1) + 0{,}3 * M$$

El número de unidades de consumo de un hogar se corresponde con la suma ponderada de los diferentes miembros del hogar valiendo 1 el sustentador principal o primer adulto del hogar, 0,5 el resto de mayores de 13 años del hogar (A) y 0,3 los menores de 13 años (M).

Distingue con 1 a las zonas urbanas y con 0 a aquellas mediana o escasamente pobladas.

- **Edad**: Variable numérica calculada sustrayendo el año de nacimiento del principal adulto del hogar (obtenible en variable 'PB140') de la fecha en que se realizó la ECV.
- **Sexo**: Variable ficticia de valor 0 si el adulto responsable del hogar es varón y 1 si es mujer. A partir de la variable 'PB150'.
- **Estado_Civil_Casado, Estado_Civil_Soltero, Estado_Civil_Viudo y Estado_Civil_Divorciado/Separado**: Variables ficticias sobre el estado civil del principal adulto del hogar calculadas a partir de la variable 'PB190'. En los modelos econométricos la categoría base es 'Estado_Civil_Casado'.
- **En_Formación**: Variable ficticia con valor 1 si el adulto responsable del hogar se encuentra activamente en estado de formación y 0 si no lo está. Este estado de formación es independiente de si el adulto está trabajando o no y se obtiene de la variable 'PE010'.
- **Niv_Estudios_Bajo, Niv_Estudios_Medio y Niv_Estudios_Alto**: Variables ficticias con el nivel de estudios del principal adulto del hogar. El nivel bajo corresponde a las personas con una formación inferior a la educación secundaria completa, el nivel alto a aquellas con formación superior, y el nivel medio al resto. Las clasificaciones se realizan a partir de los niveles ISCED incluidos en la variable 'PE041', y la categoría base es Niv_Estudios_Alto.
- **Sit_Laboral_Ocupado, Sit_Laboral_Parado, Sit_Laboral_Jubilado y Sit_Laboral_Otros_inactivos**: Variables ficticias que indican la situación laboral del adulto responsable del hogar partir de la variable 'PL032'. La categoría base es Sit_Laboral_Jubilado.
- **Salud_Grave**: Variable ficticia con valor 1 en caso de que el principal adulto responsable del hogar posea una enfermedad o problema de salud grave, y 0 en caso contrario. Se obtiene a partir de la variable 'PH020'.

5. RESULTADOS

En este trabajo, la preparación de los datos, el entrenamiento o estimación de modelos y la elaboración de predicciones se han hecho mediante el lenguaje de programación Python a través de documentos de tipo "*Jupyter Notebook*". Los *Jupyter Notebooks* permiten mezclar campos de texto explicativo con bloques de código donde ejecutar las diferentes instrucciones de Python en tiempo real, de una forma visual y sencilla de entender. Las principales librerías usadas en el proyecto han sido *statsmodels* para la aplicación de modelos econométricos, y *sklearn* para los modelos de Aprendizaje Automático. En el Anexo, se adjuntan los dos *Notebooks* empleados para la preparación de datos y comparación de modelos.

El preprocesado de los datos realizado previo al análisis ha incluido los siguientes pasos (Raschka & Mirjalili, 2019):

1. Limpieza de datos. Consiste en la eliminación de las entradas donde falten valores para alguna de las variables o en la imputación de valores en ellas. Por suerte, la información eliminada ha sido mínima, ya que la ECV para las variables escogidas estaba suficientemente completo. Las entradas desechadas se corresponden únicamente con el 1.51% de la muestra total.
2. Codificación de variables categóricas. La forma de convertir las variables con valor no numérico o con K categorías en variables binarias es una de las principales diferencias entre la librería *statsmodels* de los modelos logit y probit y la librería *sklearn* de los modelos de Aprendizaje Automático. Para *statsmodels*, es recomendable introducir entre los parámetros un término constante asociado a una variable que es igual a 1 en todas las observaciones. Con el propósito de evitar la trampa de las variables ficticias, se genera un número de variables ficticias igual al número de categorías que tengan menos 1 (K-1). Por lo tanto, una de las categorías de cada variable cualitativa no tiene asociada una variable ficticia y sirve como la categoría base o de referencia (Wooldridge, 2010). Los modelos de Aprendizaje Automático admiten tantas variables binarias como categorías haya. Este proceso de codificación se conoce como "*one-hot encoding*" (Raschka & Mirjalili, 2019).
3. Partición de la muestra en datos de entrenamiento y test. La proporción seguida es de 70% para entrenamiento y 30% para test o prueba. Si tras la limpieza de datos contamos con 23944 hogares,

16760 van a ser usados para el entrenamiento y 7184 para probar los modelos.[2]

5.1. Análisis descriptivo de los datos

Hecha la preparación de los datos, el siguiente paso fue hacer un análisis preliminar de los datos con los que trabajar. En la Tabla 5.1 se muestran una serie de estadísticos descriptivos para cada una de las variables usadas.

	Media aritmética	**Desviación estándar**	**Coeficiente de variación (Desviación entre la media)**
Pobre	20.74%	0.4055	1.9549
Familia_Numerosa	4.74%	0.2126	4.4809
Viv_PagadaOCedida	57.13%	0.4949	0.8664
Hacinamiento	0.5385 habitaciones por habitante	0.3290	0.6109
Ninos_Dep	35.47%	0.4784	1.3490
Urbanización	53.35%	0.4989	0.9352
Edad	58.08 años	14.7645	0.2542
Sexo Femenino	42.74%	0.4947	1.1576
En_Formacion	2.52%	0.1569	6.2112
Salud_Grave	45.24%	0.4977	1.1002
Region_Andalucia	10.58%	0.3076	2.9074
Region_Aragon	4.04%	0.1969	4.8746
Region_Asturias	3.28%	0.1781	5.4317
Region_Baleares	2.81%	0.1652	5.8849
Region_Canarias	3.22%	0.1764	5.4861
Region_Cantabria	2.93%	0.1687	5.7541
Region_CastillaLaMancha	4.29%	0.2027	4.7215
Region_CastillaYLeon	6.06%	0.2387	3.9359
Region_Catalunya	21.88%	0.4134	1.8896
Region_CeutaYMelilla	1.52%	0.1225	8.0376
Region_Extremadura	3.96%	0.1949	4.9280
Region_Galicia	5.43%	0.2267	4.1719
Region_Madrid	9.77%	0.2969	3.0493
Region_Murcia	3.50%	0.1837	5.2543
Region_Navarra	2.66%	0.1608	6.0539
Region_Pais Vasco	4.38%	0.2046	4.6742
Region_Rioja	2.58%	0.1586	6.1438
Region_Valencia	7.12%	0.2572	3.6116

[2] Se ha realizado un análisis de sensibilidad utilizando el 80% de los datos disponibles como muestra de entrenamiento y el 20% como muestra de test. Los resultados alcanzados entre la proporción 70:30 y 80:20 son muy similares.

Estado_Civil_Casado	54.77%	0.4977	0.9087
Estado_Civil_Divorciado	11.90%	0.3238	2.7212
Estado_Civil_Soltero	20.67%	0.4049	1.9592
Estado_Civil_Viudo	12.66%	0.3325	2.6268
Niv_Estudios_Alto	34.89%	0.4766	1.3662
Niv_Estudios_Medio	22.08%	0.4148	1.8783
Niv_Estudios_Bajo	43.03%	0.4951	1.1507
Sit_Laboral_Jubilado	28.04%	0.4492	1.6020
Sit_Laboral_Ocupado	53.09%	0.4991	0.9401
Sit_Laboral_Otros inact.	11.18%	0.3151	2.8192
Sit_Laboral_Parado	7.70%	0.2666	3.4630

TABLA 5.1. ESTADÍSTICOS DESCRIPTIVOS DE LAS VARIABLES USADAS

Prestando atención a los valores medios de las variables, podemos ver como la mayoría de los adultos responsables de los hogares están casados, poseen empleo, tienen una edad media cercana a los 60 años y algo menos de la mitad de ellos poseen un nivel de estudios bajo y alguna enfermedad o problema de salud. La mayoría de los hogares de la muestra están en régimen de propiedad totalmente pagada, se ubican en las comunidades de Cataluña, Andalucía, Madrid y Valencia, y se encuentran en zonas urbanas.

Observando los coeficientes de variación podemos apreciar como varias de las variables de la muestra tienen una disparidad significativa debido a lo desigualmente distribuidas que están, en especial las variables ficticias obtenidas de variables categóricas, y las variables *En_Formacion*, *Familia_Numerosa* y la variable objetivo. Y es que los hogares en condición de pobreza son alrededor del 20.74%, lo que indica que la muestra está bastante desbalanceada.

Aparte de estos estadísticos descriptivos, con la librería de Python *ydata_profiling* se han analizados los datos para descubrir diversas correlaciones entre variables. Las más significativas son las altas correlaciones directas de la variable *Ninos_Dep* con las variables *Hacinamiento* y *Edad* y la de la variable *Sit_Laboral* también con *Edad.*

5.2. Resultados de los modelos logit y probit.

Tras hacer el análisis previo de los datos, podemos comenzar con las primeras comparaciones entre modelos. En primer lugar, se van a comparar en la Tabla 5.2 las tasas de acierto de los modelos logit y probit de la Econometría y del modelo de regresión logística del Aprendizaje Automático tanto en la muestra de entrenamiento como en la de prueba. Se distinguen dos tipos de tasas de acierto. Por un lado, las tasas globales que se corresponden con el porcentaje total de observaciones clasificadas de forma correcta. Por otro lado, las tasas de acierto de cada clase que se corresponden con el porcentaje de observaciones clasificadas correctamente dentro de cada grupo.

	Logit		**Probit**		**Regresión logística**	
Valores reales	*Datos Entrenamiento*	*Datos de Prueba*	*Datos de Entrenamiento*	*Datos de Prueba*	*Datos de Entrenamiento*	*Datos de Prueba*
Tasa de acierto con pobres	23.65%	21.95%	22.97%	21.26%	23.65%	21.95%
Tasa de acierto con no pobres	96.37%	96.39%	96.58%	96.55%	96.37%	96.39%
Tasa global de acierto	81.09%	81.43%	81.11%	81.42%	81.09%	81.43%

TABLA 5.2. TASAS DE ACIERTO DE MODELOS LOGIT, PROBIT Y REGRESIÓN LOGÍSTICA

De los resultados de la Tabla 5.2 cabe extraer una serie de conclusiones. La primera es verificar, tal y como se dijo en la Sección 3.1, que no existen apenas diferencias entre el desempeño de los modelos logit y probit. Otra conclusión es que los resultados de la regresión logística son exactamente los mismos que los del modelo Logit, comprobando que efectivamente ambos modelos son equivalentes, a pesar de ser calculados mediante librerías diferentes y con un distinto preprocesado de datos (en la en la Sección 5.3 se encuentran más detalles sobre el preprocesamiento de los datos de los modelos de ML).

La tercera conclusión es que las tasas de acierto, tanto global como de cada grupo, con los datos de entrenamiento son semejantes a las conseguidas con los datos de test, luego los modelos generalizan adecuadamente, pareciendo que no se incurre en un problema de sobreajuste.

Finalmente, la última conclusión que se puede extraer de la Tabla 5.2 es que, independientemente del modelo, la tasa de acierto de la pobreza es pésima, con un valor inferior al 25% de aciertos. En contraposición, la tasa de acierto de los hogares no pobres es excelente, rondando el 100%, lo que hace que la tasa global de acierto se quede en torno al 80%. Este resultado puede estar relacionado con que el número de hogares pobres es comparativamente muy inferior al número de hogares no pobres tal y como se pudo apreciar Subsección 5.1.

Con el fin de profundizar en este comportamiento asimétrico de los modelos en los hogares que son pobres y en los que no lo son, la Tabla 5.3 presenta las tablas de confusión con los datos de test de los diferentes modelos. Estas tablas contienen el número de hogares correcta e incorrectamente clasificados a partir de las predicciones de cada uno de los modelos (entre paréntesis, aparecen los porcentajes).

Se comprueba de nuevo que el número de hogares pobres que se predice es mucho menor que el de hogares no pobres. De hecho, los modelos tienden a predecir más hogares de los que se deberían como no pobres (alrededor de 93%

individuos de prueba son clasificados como no pobres, pese a que realmente son el 79.90%). Esto se debe a que al ser los no pobres la categoría mayoritaria predecir un hogar como no pobre tiene una mayor probabilidad de acierto.

	Predicciones Logit		**Predicciones Probit**		**Predicciones Regresión logística**		
Valores reales	*No pobre*	*Pobre*	*No pobre*	*Pobre*	*No pobre*	*Pobre*	***Total***
No pobre	5533 (77.02%)	207 (2.88%)	5542 (77.14%)	198 (2.76%)	5533 (77.02%)	207 (2.88%)	5740 (79.90%)
Pobre	1127 (15.69%)	317 (4.41%)	1137 (15.83%)	307 (4.27%)	1127 (15.69%)	317 (4.41%)	1444 (20.10%)
Total	6660 (92,71%)	524 (7.29%)	6679 (92.97%)	505 (7.03%)	6660 (92,71%)	524 (7.29%)	7184 (100%)

TABLA 5.3. MATRIZ DE CONFUSIÓN DE MODELOS LOGIT, PROBIT Y REGRESIÓN LOGÍSTICA

Aunque el objetivo de este trabajo es realizar predicciones, nos gustaría hacer un breve comentario sobre la importancia relativa de las variables independientes. Así, las variables más significativas de acuerdo con, con el p-valor más pequeño, por debajo de 0.05, y que presentan un estadístico de significación individual más elevado (valor z) por orden de importancia han sido *Niv_Estudios_Bajo, Sit_Laboral_Parado*, *Estado_Civil_Divorciado/Separado*, *Estado_Civil_Divorciado/Separado* y *Niv_Estudios_Medio*. En contraposición, las variables más irrelevantes con un mayor p-valor, superior a 0.05, han sido: *Viv_PagadaOCedida*, *Edad*, *En_Formacion* y las regiones de Canarias, Castilla La Mancha, Ceuta y Melilla, Murcia y Valencia.

En los modelos logit y probit, existen diversas acciones con las que intentar mejorar el resultado de las estimaciones hechas. En primer lugar, es posible probar con la inclusión de variables para recoger comportamientos no lineales, como puede ser la edad al cuadrado (Alkire & Fang, 2018). También, se puede modificar el umbral que especifica a partir de qué probabilidad se es pobre. El umbral óptimo para lograrlo se puede obtener como veremos más adelante en la comparación de los diferentes modelos de Aprendizaje Automático.

Finalmente, como el uso el modelo logit de la librería *statsmodels* produce los mismos resultados a la regresión logística de *sklearn*, de aquí en adelante, solamente se va a usar el modelo de regresión logística para hacer el resto de las comparaciones de modelos de Aprendizaje Automático

5.3. Resultados de las técnicas de Aprendizaje Automático

En este apartado se van a comparar los resultados de predicción de los modelos de Aprendizaje Automático. En concreto, se comparan la regresión logística, el árbol de clasificación y la red neuronal. Para el entrenamiento de estos modelos, se ha trabajado con variables tipificadas para facilitar la convergencia de los algoritmos. Asimismo, se realiza también un proceso simultáneo de optimización de hiperparámetros y de selección de variables[3].

Recordemos que los hiperparámetros son valores de configuración de los modelos no obtenidos de los datos, pero cuya elección es crucial en la definición del comportamiento de los algoritmos. Los hiperparámetros manipulados en este trabajo son:

- En la regresión logística, la fuerza de regularización (C).
- En el árbol de clasificación, la profundidad máxima y el número mínimo de individuos por rama.
- En la red neuronal, el tamaño máximo de la capa oculta y la fuerza de regularización (alpha).

Para reducir el número de variables utilizadas en el entrenamiento, la función *SelectKBest* de la librería *sklearn* permite seleccionar las k mejores características de acuerdo con su comportamiento individual en una determinada función de puntuación (scikit-learn, 2023). Estas funciones de puntuación son, por ejemplo, los estadísticos de contraste de la Chi-cuadrado ('chi2') o del ANOVA ('f_classif'), siendo este último el escogido para este trabajo.

En nuestro caso, en la reducción de la dimensionalidad, se han estudiado los resultados de cada modelo con las 15, 20, 25 y 30 variables más relevantes de acuerdo con el criterio 'f_classif'. Por lo general, los modelos obtienen mejor tasa de acierto con el mayor número de variables, aunque las mejoras son mínimas a partir de cierto número. El mejor número de variables para cada modelo se encuentra entre las 25 o 30, dependiendo de la aleatoriedad del entrenamiento de los atributos del modelo. El hecho de que todos los modelos hayan dado alguna vez 25 como mejor número de variables demuestra la escasa suma de información que aportan las variables descartadas.

De las 38, las 8 variables con peor puntuación en el test 'f_classif' han sido: *Viv_PagadaOCedida*, *Ninos_Dep*, *En_Formacion*, *Region_Asturias*, *Region_Cantabria*, *Region_Galicia*, *Region_Rioja* y *Niv_Estudios_Medio*. Cuando el mejor número de variables es 25, las 5 siguientes desechadas son: *Region_Baleares*, *Region_CastillaYLeon*, *Region_Pais_Vasco,*

[3] Dado que las variables no cuentan con la misma importancia en la predicción, es recomendable no incluirlas todas en los modelos de Aprendizaje Automático dado que su inclusión puede llegar a entorpecer la convergencia de los algoritmos y ralentizar los entrenamientos.

Region_Valencia, *Estado_Civil_Viudo*. Y *Niv_Estudios_Medio.* Se puede ver como salvo en el caso de *Edad*, las variables más irrelevantes son las mismas que se obtuvieron al ver los p-valores en los modelos de elección discreta.

El entrenamiento de los modelos de Aprendizaje Automático -para cada combinación de hiperparámetros y número de variables- se realiza mediante validación cruzada con la técnica de *k-folding* (scikit-learn, 2023). El *k-folding* consiste en dividir los datos de entrenamiento en k subconjuntos e ir empleando k-1 de ellos para el entrenamiento del modelo y el restante para validación. Este proceso se realiza k veces para cada uno de los subconjuntos y el valor final de los parámetros del modelo se calcula como el promedio de los calculados en cada iteración. En nuestro caso, k va a ser igual a 10.

Con el fin de mostrar el proceso de selección conjunta del número de variables y del valor de los hiperparámetros, la Ilustración 5.1 representa, para la regresión logística, la variación de la tasa de acierto media por cada combinación del número de variables y valor del hiperparámetro C. En este caso se comprueba que la mejor combinación se produjo para las 25 variables más relevantes y el hiperparámetro C con valor 0.1. Se realiza un procedimiento análogo de la selección conjunta de número de variables e hiperparámetros para el resto de los modelos, pero con una visualización más difícil al contar con un mayor número de hiperparámetros.

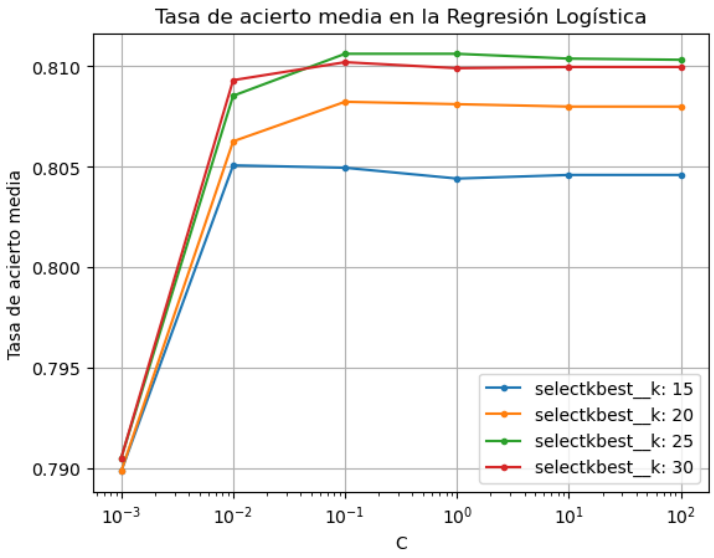


ILUSTRACIÓN 5.1. TASA DE ACIERTO MEDIA EN LA REGRESIÓN LOGÍSTICA SEGÚN EL NÚMERO DE VARIABLES Y EL HIPERPARÁMETRO C

El entrenamiento de la red neuronal mediante validación cruzada tardó alrededor de 25 minutos en finalizar, mientras que el de la regresión logística finalizó en menos de 10 segundos y el del árbol de clasificación 40 segundos. La gran cantidad de entrenamientos para la selección de los hiperparámetros y el

número de variables alarga el cálculo de parámetros que es casi instantáneo en los modelos de Econometría de la librería *statsmodels*.

Como resultado del entrenamiento de los modelos, las cinco variables más relevantes para cada modelo en orden de importancia han sido:

- Regresión Logística: *Sit_Laboral_Parado*, *Estado_Civil_Divorciado/ Separado, Niv_Estudios_Alto*, *Sit_Laboral_Ocupado* y *Hacinamiento*.
- Árbol de Clasificación: *Sit_Laboral_Parado*, *Niv_Estudios_Bajo*, *Sit_Laboral_Otros inactivos*, *Estado_Civil_Casado* y *Niv_Estudios_Alto*.
- Red Neuronal: *Sit_Laboral_Parado*, *Sit_Laboral_Otros inactivos*, *Region_Canarias*, *Sit_Laboral_Ocupado* y *Niv_Estudios_Alto*.

Como se ve, la variable más importante para todos los modelos es la situación laboral de parado. Otra variable que se encuentra entre las 5 más relevantes de todos los modelos es el nivel de estudios alto.

El primer criterio que se ha utilizado para comparar los diferentes modelos de Aprendizaje Automático está basado en la distribución de la tasa de acierto para las 10 iteraciones de la validación cruzada con la mejor combinación de hiperparámetros y número de variables (Ilustración 5.2). El modelo que parece ofrecer los peores resultados en la validación cruzada es el árbol de clasificación, mientras que los mejores los da la red neuronal quedando ligeramente por encima de la regresión logística.

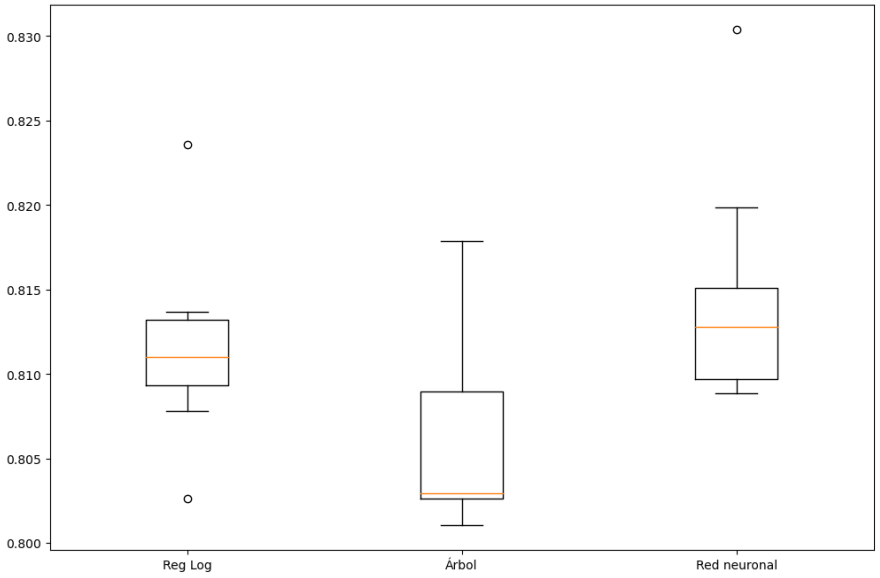


ILUSTRACIÓN 5.2. DIAGRAMAS DE CAJAS Y BIGOTES CON LOS RESULTADOS DE LA VALIDACIÓN CRUZADA

Si bien la validación cruzada sirve para medir la eficacia de predicción de los modelos de manera robusta, otra forma de analizar la capacidad predictiva de cada modelo es empleando los modelos ya entrenados para realizar

predicciones sobre el 30% de datos de prueba aún no utilizados. La tasa global de acierto se muestra en la Tabla 5.4. El comportamiento de los modelos es muy similar, quedando la red neuronal en primera posición y el árbol de clasificación en última.

	Regresión logística	**Árbol de Clasificación**	**Red Neuronal**
Tasa global de acierto	81.47%	81.32%	81.50%

TABLA 5.4. TASAS DE ACIERTO DE MODELOS DE REGRESIÓN LOGÍSTICA, ÁRBOL DE CLASIFICACIÓN Y RED NEURONAL

A pesar de que la tasa global de aciertos puede parecer el indicador más importante, este no resulta muy adecuado en muestras de datos no balanceadas, como la que tenemos. Con el objetivo de solucionar este problema, la Tabla 5.5 analiza la tasa de acierto en cada grupo mediante las matrices de confusión de los diferentes procedimientos. De las matrices de confusión podemos concluir que la red neuronal es el modelo que predice un mayor porcentaje de hogares como pobre, con un 7.64% de hogares pobres, frente al 7% y 5.48% de la regresión logística y el árbol de clasificación respectivamente, y frente al 20.10% de hogares pobres observados. Vemos que el árbol de clasificación es el que más tiende a predecir los hogares como no pobres al ser esta la categoría mayoritaria por lo que la utilidad de este modelo para predecir quien es realmente pobre pierda valor.

	Predicciones Regresión Logística		**Predicciones Árbol de Clasificación**		**Predicciones Red Neuronal**		
Valores reales	*No pobre*	*Pobre*	*No pobre*	*Pobre*	*No pobre*	*Pobre*	***Total***
No pobre	5545 (77.19%)	195 (2.71%)	5594 (77.87%)	146 (2.03%)	5523 (76.88%)	217 (3.02%)	5740 (79.90%)
Pobre	1136 (15.81%)	308 (4.29%)	1196 (16.65%)	248 (3.45%)	1112 (15.48%)	332 (4.62%)	1444 (20.10%)
Total	6681 (93%)	503 (7%)	6790 (94.52%)	394 (5.48%)	6635 (92,36%)	549 (7.64%)	7184 (100%)

TABLA 5.5. MATRICES DE CONFUSIÓN DE MODELOS DE REGRESIÓN LOGÍSTICA, ÁRBOL DE CLASIFICACIÓN Y RED NEURONAL

Relacionados con la matriz de confusión, se definen unos coeficientes que también permiten comparar los modelos analizados. El primer coeficiente es la especificidad (también llamada precisión en *sklearn*) que se define como

$$especifidad = \frac{TP}{(TP + FP)}$$

donde TP(True Positives) son los individuos clasificados como 1 correctamente y FP (False Positives) los clasificados como 1 incorrectamente. Por ejemplo,

para la regresión logística, la especificidad de la pobreza se obtiene como el cociente entre 308 y 503 y hace referencia a la proporción de hogares predichos como pobres correctamente.

El segundo coeficiente usado es la sensibilidad o *recall*, que se define como

$$sensibilidad = \frac{TP}{(TP + FN)}$$

donde FN (False Negatives) son los individuos clasificados como 0 incorrectamente. Por ejemplo, volviendo a la regresión logística, la sensibilidad de la pobreza se obtiene dividiendo 308 entre 1444 e indica la proporción de hogares pobres clasificados como pobres. Es equivalente a la tasa de acierto de la pobreza usada en la Sección 5.2.

Mientras que la especificidad permite medir la habilidad del modelo de no clasificar un hogar no pobre como pobre, la sensibilidad mide la habilidad del modelo para clasificar correctamente a los hogares pobres (scikit-learn, 2023). El tercer coeficiente es el valor F1, que es igual a la media armónica de la especificidad y la sensibilidad y sirve como término medio de ambas.

Dado que las etiquetas 0 y 1 se asignan de forma arbitraria, se pueden definir los tres anteriores coeficientes intercambiando la etiqueta de los grupos. De este modo, en nuestro caso tendremos la especificidad, la sensibilidad y el valor F1 tanto para los pobres como los no pobres (véase Tabla 5.6).

	Regresión Logística		**Árbol de Clasificación**		**Red Neuronal**	
	No pobre	*Pobre*	*No Pobre*	*Pobre*	*No pobre*	*Pobre*
Especificidad	82.99%	61.23%	82.39%	62.94%	83.24%	60.47%
Sensibilidad	96.60%	21.33%	97.46%	17.17%	96.22%	22.99%
Valor F1	89.28%	31.64%	89.29%	26.98%	89.26%	33.31%

TABLA 5.6. PRECISIÓN, SENSIBILIDAD Y VALOR F1 PARA LOS MODELOS DE REGRESIÓN LOGÍSTICA, ÁRBOL DE CLASIFICACIÓN Y RED NEURONAL

Todos los métodos tienen buena precisión y sensibilidad en la predicción de si un hogar no es pobre, pero su capacidad de predecir cuándo un hogar es pobre es mucho menor. Fijándonos en la sensibilidad, se ve como en el modelo con mejores resultados, la red neuronal, menos de la cuarta parte de los hogares pobres se predicen correctamente. Esta proporción se va reduciendo hasta llegar a menos de la quinta parte en el árbol de clasificación.

El último criterio para comparar los modelos va a ser la curva ROC (Receiver Operating Characteristic). Esta curva es la representación gráfica de la relación entre la especificidad y la sensibilidad para cada umbral de decisión posible (Raschka & Mirjalili, 2019). En el caso de modelos sin umbrales de clasificación, como los árboles de clasificación, la curva ROC se construye generando probabilidades de pertenencia a cada clase y representando la tasa de verdaderos positivos (True Positive Rate, TPR) y la tasa de falsos positivos (False Positive Rate, FPR). La diagonal de la gráfica ROC corresponde con la elección aleatoria y si el modelo va por debajo de ella es que sus estimaciones son peores que la elección aleatoria. Cuanto mayor sea el área entre la curva ROC y la diagonal o el AUC (Area Under the Curve, con valores de 0,5 a 1) mejor clasificador es el modelo (Perrier, 2022).

Las Ilustración 5.3, la Ilustración 5.4 y la Ilustración 5.5 corresponden a la gráfica ROC y su AUC de la regresión logística, el árbol de clasificación y la red neuronal. Estas gráficas avalan nuevamente como, aunque todas las curvas obtenidas y sus AUC son muy semejantes, el modelo que mejor clasifica es la red neuronal, mientras que el árbol de clasificación ofrece los peores resultados. El desempeño de la regresión logística (o su equivalente econométrico, el logit) queda muy ligeramente por debajo de la red neuronal, siendo no obstante un cálculo notablemente más rápido.

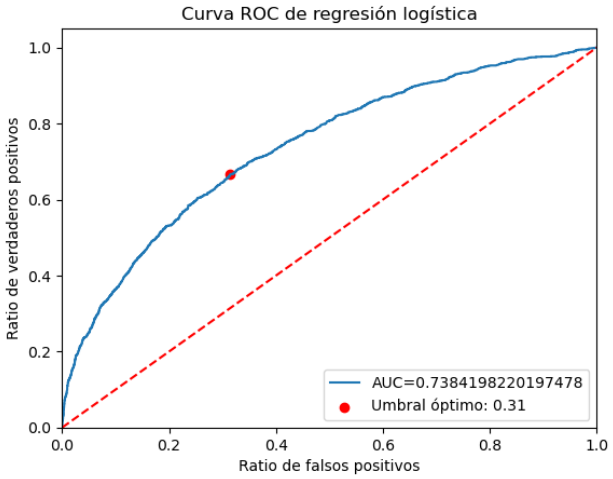


ILUSTRACIÓN 5.3. CURVA ROC DE LA REGRESIÓN LOGÍSTICA

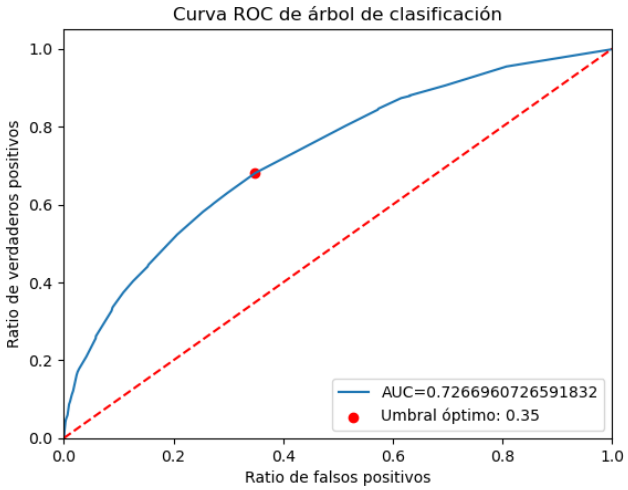


ILUSTRACIÓN 5.4. CURVA ROC DEL ÁRBOL DE CLASIFICACIÓN

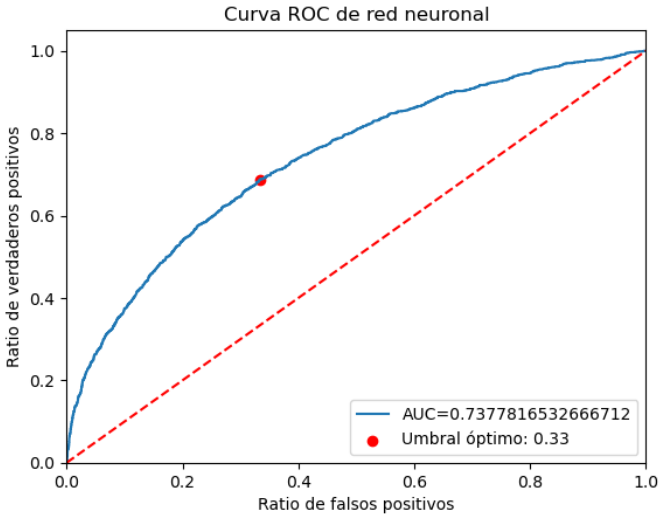


ILUSTRACIÓN 5.5. CURVA ROC DE LA RED NEURONAL

Otra de las utilidades de la curva ROC es la de servir para identificar el umbral de decisión más cercano a la esquina superior izquierda. Este es, el umbral de decisión que ofrece la mejor clasificación de variables para la muestra. Tomando los umbrales óptimos de cada modelo, hago una nueva predicción de datos y comparo la tasa de acierto global, las matrices de confusión y los parámetros de clasificación en la Tabla 5.7, la Tabla 5.8 y la Tabla 5.9 respectivamente.

Se puede apreciar cómo, si bien se produce un ligero empeoramiento en la tasa de acierto global de los modelos, la tasa de acierto de la pobreza o sensibilidad de esta variable mejora considerablemente. La cantidad de individuos clasificados en cada categoría es mucho más próxima a la cantidad real de individuos pobres o no pobres. Desde luego, para muestras de datos desbalanceadas como estas, la optimización del umbral de decisión mediante la ROC es un paso altamente recomendado.

	Regresión logística	Árbol de Clasificación	Red Neuronal
Tasa global de acierto	77.66%	78.76%	77.78%

TABLA 5.7. TASAS DE ACIERTO DE MODELOS DE REGRESIÓN LOGÍSTICA, ÁRBOL DE CLASIFICACIÓN Y RED NEURONAL CON EL UMBRAL DE DECISIÓN ÓPTIMO

	Predicciones Regresión Logística		**Predicciones Árbol de Clasificación**		**Predicciones Red Neuronal**		
Valores reales	*No pobre*	*Pobre*	*No pobre*	*Pobre*	*No pobre*	*Pobre*	***Total***
No pobre	4944 (68.82%)	796 (11.08%)	5118 (71.24%)	622 (8.66%)	4954 (68.96%)	786 (10.94%)	5740 (79.90%)
Pobre	809 (11.26%)	635 (8.84%)	904 (12.58%)	540 (7.52%)	810 (11.28%)	634 (8.83%)	1444 (20.10%)
Total	5753 (80.08%)	1431 (19.92%)	6022 (83.83%)	1162 (16.17%)	5764 (80,23%)	1420 (19.77%)	7184 (100%)

TABLA 5.8. MATRICES DE CONFUSIÓN DE MODELOS DE REGRESIÓN LOGÍSTICA, ÁRBOL DE CLASIFICACIÓN Y RED NEURONAL CON EL UMBRAL DE DECISIÓN ÓPTIMO

	Regresión Logística		**Árbol de Clasificación**		**Red Neuronal**	
	No pobre	*Pobre*	*No pobre*	*Pobre*	*No pobre*	*Pobre*
Precisión	85.93%	44.37%	84.99%	46.55%	85.95%	44.65%
Sensibilidad	86.13%	43.98%	89.16%	37.40%	86.31%	43.91%
Valor F1	86.03%	44.17%	87.03%	41.48%	86.13%	44.28%

TABLA 5.9. PRECISIÓN, SENSIBILIDAD Y VALOR F1 PARA LOS MODELOS DE REGRESIÓN LOGÍSTICA, ÁRBOL DE CLASIFICACIÓN Y RED NEURONAL CON EL UMBRAL DE DECISIÓN ÓPTIMO

6. CONCLUSIONES

Este trabajo compara diferentes técnicas para predecir si un hogar es pobre o no. Para ello se parte de una aproximación unidimensional a la pobreza, teniendo en cuenta la renta de los hogares, y considerando los factores más empleados en la literatura de la predicción de la pobreza. Estos factores han sido características regionales y características propias del hogar y de la persona de referencia; disponibles en la Encuesta de Condiciones de Vida del año 2022, que ha sido la fuente de datos empleada.

Las técnicas de predicción utilizadas proceden tanto del ámbito de la Econometría como del Aprendizaje Automático e incluyen el modelo logit o regresión logística, el modelo probit, el árbol de clasificación y la red neuronal. En su aplicación, se han empleado las librerías de *Python statsmodels* para los modelos econométricos y *sklearn* en los provenientes del ML.

Los resultados han servido para comprobar que los modelos logit y probit ofrecen prácticamente los mismos resultados. También han servido para mostrar que no existe ninguna diferencia entre los resultados del modelo logit y su homólogo del Aprendizaje Automático, la regresión logística.

Para el entrenamiento de los modelos de Aprendizaje Automático se ha hecho uso de la tipificación de variables, reducción de la dimensionalidad, selección de hiperparámetros y validación cruzada. Se han utilizado diferentes criterios para comparar los resultados. En concreto, la tasa de acierto media de la validación cruzada; la tasa de acierto global; la matriz de confusión; diferentes medidas obtenidas a partir de la matriz de confusión como la sensibilidad, la especificidad o el valor F1 y la curva ROC y su AUC.

Todos los modelos estudiados han obtenido tasas de acierto globales relativamente similares, algo superiores en la red neuronal. Sin embargo, ningún modelo es capaz de obtener buenos datos al clasificar los hogares como pobres debido al desbalance en la distribución de la variable dependiente utilizando un umbral de decisión que considera que hay el mismo número de hogares pobres como no pobres. La solución a esta última arbitrariedad en la selección del umbral pasa por el ajuste del umbral de decisión gracias a la curva ROC, obteniendo unos resultados en los que la tasa de acierto de la pobreza mejora hasta unos niveles igualmente insatisfactorios, pero considerablemente mejores.

Vistos los resultados obtenidos, el modelo logit o de regresión logística sigue siendo una de las mejores opciones para predecir si un hogar es pobre o no, tanto en la Econometría como en el Aprendizaje Automático. Sus principales ventajas son la rapidez de su cálculo y la gran interpretabilidad que ofrecen sus coeficientes.

La red neuronal ofrece los mejores resultados en casi todas las pruebas, luego parece la opción superior. No obstante, algunos de los inconvenientes de este modelo son su gran tiempo de cálculo y su empleo de considerables recursos computacionales. Es también un modelo opaco o de caja negra, en el que no es posible interpretar en detalle cómo es que se toman las decisiones. En el polo opuesto se encuentran los árboles de clasificación, los cuales si bien han dado los peores resultados en la predicción son los modelos que ofrecen una mayor interpretabilidad gracias al diagrama de árbol que construyen.

Como conclusión, si bien los modelos de Aprendizaje Automático son los más indicados para predicciones de gran exactitud para grandes muestras de datos (Varian, 2014), en problemas donde la interpretabilidad juega un papel relevante y las muestras de datos son relativamente pequeñas, los métodos econométricos aún son de gran utilidad. La mejora en la predicción de las redes neuronales no compensa la pérdida de interpretabilidad que suponen frente a los resultados que ya ofrecía el modelo logit; y los árboles de clasificación, si bien suponen otra forma de interpretar los datos, dan resultados peores que los de la regresión logística.

Una idea importante del Aprendizaje Automático es que el promedio de muchos modelos pequeños tiende a ofrecer mejores predicciones fuera de muestra que la elección de un único modelo. Así, en los últimos años están surgiendo diversos modelos diseñados desde una perspectiva econométrica para aprovechar el mejor resultado de predicción de los modelos del *Machine Learning* que mantienen la interpretabilidad clásica de los modelos econométricos. Ejemplos de estos modelos son el GAMMLA de Flachaire (2022) o el PLTR de Dumitrescu (2020), que pueden abrir nuevas líneas de investigación en la predicción de la pobreza.

BIBLIOGRAFÍA

Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H.-T. (2012). *Learning From Data. A Short Course.* AMLBook.

Addison, T., & Hulme, D. (2005). *Poverty Dynamics: Interdisciplinary Perspectives.* Nueva York: Oxford University Press.

Alkire, S., & Fang, Y. (2018). Dynamics of Multidimensional Poverty and Uni-dimensional Income Poverty: An Evidence of Stability Analysis from China. *Social Indicators Research 142*, 25-64.

Athey, S., & Imbens, G. W. (2019). Machine Learning Methods That Economisis Should Know About. *Anual Review of Economics*, 685-725.

Belver García, M. (2020). *Pobreza - Desarrollo Sostenible*. Obtenido de un.org: https://www.un.org/sustainabledevelopment/es/poverty/

Bourguignon, F. (2003). The Measurement of Multidimensional Poverty. *The Journal of Economic Inequality*, 25-49.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. (1984). *Classification and Regression Trees.* Taylor & Francis.

Caparrini, S. (16 de Marzo de 2022). *Redes Neuronales: una visión superficial*. Obtenido de cs.us.es: http://www.cs.us.es/~fsancho/?e=72

Castaneda Aguilar, R. A., Diaz-Bonilla, C., Fujs, T., Lakner, C., Gerszon Mahler, D., Nguyen, M. C., . . . Yonzan, N. (14 de Septiembre de 2022). *September 2022 global poverty update from the World Bank: 2017 PPPs and new data for India*. Obtenido de World Bank Blogs: https://blogs.worldbank.org/opendata/september-2022-global-poverty-update-world-bank-2017-ppps-and-new-data-india

Charpentier, A., Flachaire, E., & Ly, A. (2018). Econometrics and Machine Learning. *Economie et Statistique / Economics and Statistics, Institut National de la Statistique et des Etudes Economiques (INSEE), issue 505-506*, 147-169.

DrivenData. (2018). *Pover-T Tests: Predicting Poverty*. Obtenido de drivendata.org: https://www.drivendata.org/competitions/50/worldbank-poverty-prediction/page/99/

Efron, B., & Hastie, T. (2016). *Computer Age Statistical Inference, Vol. 5.* Cambridge: Cambridge University Press.

Eurostat. (13 de Abril de 2021). *Glossary: At-risk-of-poverty rate*. Obtenido de Eurostat: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:At-risk-of-poverty_rate

Eurostat. (28 de Mayo de 2023). *INCOME AND LIVING CONDITIONS: Methodology*. Obtenido de europa.eu: https://ec.europa.eu/eurostat/web/income-and-living-conditions/methodology

Eurostat. (28 de Mayo de 2023). *INCOME AND LIVING CONDITIONS: Modules*. Obtenido de europa.eu: https://ec.europa.eu/eurostat/web/income-and-living-conditions/database/modules

Flachaire, E., Hacheme, G., Hué, S., & Laurent, S. (2022). GAM(L)A: An econometric model for interpretable Machine Learning. *arXiv preprint arXiv:2203.11691*.

Foster, J., Greer, J., & Thorbecke, E. (1984). A Class of Decomposable Poverty Measures. *The Econometric Society Vol. 52, No. 3*, 761-766.

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

Gunnarsson, B. R., vanden Broucke, S., Baesens, B., Óskarsdóttir, M., & Lemahieu, W. (2021). Deep Learning for Credit Scoring: Do or Don't? *European Journal of Operational Research*, 292-305.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* Berlin: Springer.

Haughton, J., & Khandker, S. R. (2009). *Handbook on poverty and inequality.* Washington, DC: World Bank Publications.

Hinton, G. E., Rumelhart, D. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 533-536.

Hulme, D. (2010). *Global Poverty: How Global Governance is Failing the Poor.* Londres: Routledge.

Instituto Nacional de Estadística. (2016a). *Renta mediana por unidad de consumo*. Obtenido de ine.es: https://www.ine.es/DEFIne/es/concepto.htm?c=5780&op=30471&p=2&n=20

Instituto Nacional de Estadística. (2016b). *Escalas de equivalencia*. Obtenido de ine.es: https://www.ine.es/DEFIne/es/concepto.htm?c=5228&op=30458

Instituto Nacional de Estadística. (24 de Abril de 2023). *Encuesta de condiciones de vida. Resultados.* Obtenido de ine.es:

https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736176807&menu=ultiDatos&idp=1254735976608

Jarmulska, B. (2020). *Random Forest Versus Logit Models: Which Offers Better Early Warning of Fiscal Stress?* Fráncfort del Meno: European Central Bank (ECB).

Kumar Satapathy, S., Saravanan, S., Mishra, S., & Nandan Mohanty, S. (2023). A Comparative Analysis of Multidimensional COVID-19 Poverty Determinants: An Observational Machine Learning Approach. *New Generation Computing*, 155-184.

Kumar, A. (2 de Mayo de 2023). *Logit vs Probit Models: Differences, Examples.* Obtenido de vitalflux.com: https://vitalflux.com/logit-vs-probit-models-differences-examples/

Lessman, S., Baesens, B., Hsin-Vonn, S., & Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: A ten-year update. *European Journal of Operational Research*, 124-136.

Min, P. P., Wen Gan, Y., Binti Hamzah, S. N., Song Ong, T., & Sayeed, S. (2022). POVERTY PREDICTION USING MACHINE LEARNING APPROACH. *JOURNAL OF SOUTHWEST JIAOTONG UNIVERSITY*.

OECD. (2008). *Growing Unequal?: Income Distribution and Poverty in OECD Countries. Summary in Spanish.* Obtenido de https://www.oecd.org/els/soc/41547484.pdf

Parra, F. (25 de Enero de 2019). *Estadística y Machine Learning con R.* Obtenido de bookdown.org: https://bookdown.org/content/2274/modelos-con-variables-cualitativas.html

Perrier, A. (23 de Junio de 2022). *Evaluate Classification Models*. Obtenido de openclassrooms.com: https://openclassrooms.com/en/courses/5873596-design-effective-statistical-models-to-understand-your-data/6233091-evaluate-classification-models

Poverty Analysis Discussion Group. (2012). *Understanding poverty and wellbeing - A note with implications for research and policy.* Londres: Department for International Development.

Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning. Third Edition.* Birmingham: Packt Publishing Ltd.

Ravallion, M. (2011). On multidimensional indices of poverty. *The Journal of Economic Inequality 9*, 235-248.

Ravallion, M., Datt, G., & van de Walle, D. (1991). Quantifying Absolute Poverty in The Developing World. *Review of Income and Wealth*, 345-361.

scikit-learn. (15 de Junio de 2023). *Cross-validation: evaluating estimator performance*. Obtenido de scikit-learn.org: https://scikit-learn.org/stable/modules/cross_validation.html

scikit-learn. (24 de Enero de 2023). *sklearn.feature_selection.SelectKBest.* Obtenido de scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

scikit-learn. (05 de Julio de 2023). *sklearn.linear_model.LogisticRegression.* Obtenido de scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

scikit-learn. (15 de Junio de 2023). *sklearn.metrics.precision_recall_fscore_support*. Obtenido de scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support

scikit-learn. (05 de Julio de 2023). *sklearn.neural_network.MLPClassifier.* Obtenido de scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Stock, J., & Watson, M. (2012). *Introducción a la Econometría, 3ª Edición.* Madrid: Pearson Educación, S.A.

swarnimrai. (5 de Noviembre de 2021). *Multi-Layer Perceptron Learning in Tensorflow.* Obtenido de geeksforgeeks.org: https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/

Varian, H. R. (2014). Big Data: New Tricks for Econometrics. *journal of Economic Perspectives-Volume 28, Number 2*, 3-28.

Villanueva García, J. D. (23 de Octubre de 2020). *Redes neuronales desde cero (I) – Introducción.* Obtenido de iartificial.net: https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/

Wooldridge, J. M. (2010). *Introducción a la econometría: Un enfoque moderno. 4ª Edición.* México, D.F.: Cengage Learning Editores.

Zixi, H. (2021). Poverty Prediction Through Machine Learning. *2021 2nd International Conference on E-Commerce and Internet Technology*, 314-324.

ANEXO

Preparación de datos de la ECV 2022

Importar librerías empleadas

```
import pandas as pd # para manipulación de datasets en formato "dataframes"
import numpy as np  # para el procesado de valores "NaN" (Not a Number)
```

Importar los datasets D,H y P en formato CSV con separación de comas del año 2022 y seleccionar únicamente las variables de interés.

```
# Datasets 2022
anoECV = 2022

dbasicos_hogar = pd.read_csv("datos_2022/esudb22d.csv",low_memory=False)
dbasicos_hogar = dbasicos_hogar[['DB030','DB040','DB100']]

ddetallados_hogar = pd.read_csv("datos_2022/esudb22h.csv",low_memory=False)
ddetallados_hogar = ddetallados_hogar[['HB030','HB080','HB120','HY020', \
                                       'HH021','HH030','HX060', 'HX240']]

ddetallados_adultos = pd.read_csv("datos_2022/esudb22p.csv",low_memory=False)
ddetallados_adultos = ddetallados_adultos[['PB030','PB140','PB150','PB190', \
                                       'PE010', 'PE041', 'PL032', 'PH020']]
```

Combinar los dos dataframes de hogar y eliminar el identificador del hogar

```
datosFinal = ddetallados_hogar.merge(dbasicos_hogar.rename(columns={"DB030": "HB030"}))
datosFinal = datosFinal.drop(['HB030'],axis=1)
```

Revisar valores vacíos en HB080 para ver si hay hogares sin identificador de adulto responsable. Un hogar sin este identificador es inservible para el análisis. Eliminar esta fila si existe

```
for index, adulto in enumerate(datosFinal['HB080']):
    if adulto == ' ':
        display(datosFinal.iloc[[index]])
```

```
### ¡¡¡Sólo para cada fila con identificador vacío!!!
# En los datos de 2021 por ejemplo, esta fila es la 13526.

#datosFinal = datosFinal.drop(13526).convert_dtypes()
```

Combinar el dataframe de hogares con el de los adultos responsables y eliminar identificador.

```
# Formateo identificadores como números (int64). Antes se detectaban como como texto (string)
datosFinal['HB080'] = pd.to_numeric(datosFinal['HB080'])
datosFinal = datosFinal.merge(ddetallados_adultos.rename(columns={"PB030": "HB080"}).convert_dtypes(),\
                how = "inner")
datosFinal = datosFinal.drop(['HB080'],axis=1)
```

Para calcular el umbral de la pobreza, obtener el 60% de la mediana de la renta total por unidad de consumo

Las unidades de consumo se calculan como: renta disponible total del hogar/unidades de consumo del hogar (escala OCDE modificada)

```
Renta_UdConsumo = datosFinal['HY020']/datosFinal['HX240']
mediana = Renta_UdConsumo.median()
umbral = 0.6*mediana

# Elimino variables no necesarias tras calcular la renta por ud. consumo
datosFinal = datosFinal.drop(['HY020','HX240'],axis=1)
```

Insertar nueva columna en primera posición que indique si cada hogar/adulto es pobre o no

```
# Reutilizo variable Renta_UdConsumo para mostrar si hogar es pobre o no
for indice, valor in Renta_UdConsumo.items():
    if valor < umbral:
        Renta_UdConsumo[indice] = 1   #pobre
    else:
        Renta_UdConsumo[indice] = 0   #no pobre

datosFinal.insert(0,"Pobre",Renta_UdConsumo.astype('int64'))
```

```
datosFinal
```

9]:

	Pobre	HB120	HH021	HH030	HX060	DB040	DB100	PB140	PB150	PB190	PE010	PE041	PL032	PH020
0	0	4	2	4	12	ES21	1	1973	1	2	2	200	1	1
1	0	4	2	5	9	ES21	1	1960	1	2	2	340	4	1
2	1	1	2	4	2	ES21	1	1987	1	1	2	500	1	2
3	0	2	1	5	7	ES21	1	1941	1	2	2	200	3	2
4	0	3	1	5	9	ES21	1	1956	1	2	2	200	3	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
24308	0	1	1	2	6	ES51	3	1936	2	4	2	000	3	1
24309	0	4	2	4	12	ES51	3	1972	2	2	2	500	1	2
24310	0	2	1	4	8	ES51	3	1971	1				1	
24311	0	2	1	5	7	ES51	3	1941	2	4	2	100	3	1
24312	0	2	2	8	8	ES51	3	1961	1	2	2	350	1	1

24313 rows × 14 columns

Conversión de "missing values" a NaN y limpieza de datos

En primer lugar imputo el valor NaN (Not a Number) sobre los datos vacíos como para facilitar su procesado.

A continuación elimino las filas con valores NaN (no suponen un gran número, no compensa el esfuerzo de la imputación de datos en variables categóricas).

```
datosFinal.replace(" ", np.nan, inplace = True)    #transformo los valores " " en NaN
datosFinal.isna().sum()
```

3]:
```
Pobre      0
HB120      0
HH021      0
HH030     91
HX060     52
DB040      0
DB100      0
PB140      0
PB150      0
PB190    158
PE010    175
PE041    160
PL032     17
PH020    196
dtype: int64
```

```
filasAntes = len(datosFinal.index)
datosFinal = datosFinal.copy().dropna().reset_index(drop=True)
```

```
display(f'La proporción de datos desechados es tan solo del { 100 - len(datosFinal.index)*100/filasAntes}%')
```

```
'La proporción de datos desechados es tan solo del 1.5177065767285%'
```

Renombrar variables

Familia Numerosa

Creación de nueva variable binaria si en un hogar conviven más de 4 personas

```
numMiembrosHogar = datosFinal['HB120'].copy() #si no pongo copy, simplemente estoy referenciando la columna
for indice, valor in enumerate(numMiembrosHogar):
    if valor > 4:
        numMiembrosHogar[indice] = 1   #familia numerosa
    else:
        numMiembrosHogar[indice] = 0   #familia no numerosa
# Inserto nueva variable en la segunda posición
datosFinal.insert(1,"Familia_Numerosa",numMiembrosHogar)
```

Vivienda pagada o en cesión gratuita

Creación de nueva variable binaria si el hogar debe pagar por su vivienda

```
Viv_PagadaOCedida = datosFinal['HH021'].copy()
#1 propiedad sin hipoteca, 2 propiedad con hipoteca, 3 alquiler a precio mercado, 4 a precio inferior, 5 cesión gratuita
for indice, valor in enumerate(Viv_PagadaOCedida):
    if ((valor == 1) or (valor ==5)):
        Viv_PagadaOCedida[indice] = 1   #Vivienda pagada o con cesión gratuita
    else:
        Viv_PagadaOCedida[indice] = 0   #Vivienda en hipoteca o en alquiler.

# Inserto nueva variable en la tercera posición
datosFinal.insert(2,"Viv_PagadaOCedida",Viv_PagadaOCedida)
# Elimino variable que no se va a usar
datosFinal = datosFinal.drop(['HH021'],axis=1)
```

Hacinamiento

Creación de nueva variable cociente del número de habitantes del hogar entre las habitaciones disponibles.

```
Hacinamiento = datosFinal['HB120']/datosFinal['HH030'].astype(int) # Para que interprete HH030 como un número entero
# Hacinamiento.hist(bins=100) # Función para visualizar variable generada.
# La mayoría de valores ronda entre 0 y 1 como es lógico.

# Inserto nueva variable en la cuarta posición
datosFinal.insert(3,"Hacinamiento",Hacinamiento)
# Elimino variables que no se van a usar
datosFinal = datosFinal.drop(['HB120','HH030'],axis=1)
```

Con niños dependientes

Creación de nueva variable binaria que indique si en el hogar hay niños dependientes económicamente

```
NinosDep = datosFinal['HX060'].astype(int) #con catorce categorías sobre la estructura del hogar.
NinosDep.value_counts()

for indice, valor in enumerate(NinosDep):
    if valor > 9 :
        NinosDep[indice] = 1   #Hogar con niños dependientes ecómicamente
    else:
        NinosDep[indice] = 0   #Hogar sin niños dependientes económicamente

# Inserto nueva columna en la quinta posición
datosFinal.insert(4,"Ninos_Dep",NinosDep)
# Elimino variable que no se va a usar
datosFinal = datosFinal.drop(['HX060'],axis=1)
```

Región

Renombrar variable DB040 y sus valores con el nombre de las comunidades autónomas

```
Region = datosFinal["DB040"].copy()

for indice, valor in enumerate(Region):
    if valor == "ES11": Region[indice] = "Galicia"
    elif valor == "ES12": Region[indice] = "Asturias"
    elif valor == "ES13": Region[indice] = "Cantabria"
    elif valor == "ES21": Region[indice] = "Pais Vasco"
    elif valor == "ES22": Region[indice] = "Navarra"
    elif valor == "ES23": Region[indice] = "Rioja"
    elif valor == "ES24": Region[indice] = "Aragon"
    elif valor == "ES30": Region[indice] = "Madrid"
    elif valor == "ES41": Region[indice] = "CastillaYLeon"
    elif valor == "ES42": Region[indice] = "CastillaLaMancha"
    elif valor == "ES43": Region[indice] = "Extremadura"
    elif valor == "ES51": Region[indice] = "Catalunya"
    elif valor == "ES52": Region[indice] = "Valencia"
    elif valor == "ES53": Region[indice] = "Baleares"
    elif valor == "ES61": Region[indice] = "Andalucia"
    elif valor == "ES62": Region[indice] = "Murcia"
    elif valor == "ES70": Region[indice] = "Canarias"
    else: Region[indice] = "CeutaYMelilla"

# Inserto nueva variable en la sexta posición
datosFinal.insert(5,"Region",Region)
# Elimino variable que no se va a usar
datosFinal = datosFinal.drop(['DB040'],axis=1)
```

Urbanización

Renombrar variable DB100 e invertir el órden de importancia de sus valores.

De este modo, los valores de la variable "Urbanización" van a ser: (0 zona escasa o medianamente poblada, y 1 zona muy poblada)

```
Urbanizacion = datosFinal['DB100'].copy()
for indice, valor in enumerate(Urbanizacion):
    if valor > 1: Urbanizacion[indice] = 0

# Inserto nueva variable en la séptima posición
datosFinal.insert(6,"Urbanizacion",Urbanizacion)
# Elimino variable que no se va a usar
datosFinal = datosFinal.drop(['DB100'],axis=1)
```

Edad

Creación de nueva variable "Edad" a partir del año de crecimiento del adulto responsable de la vivienda

```
#Calculo la edad del responsable de la vivienda restando el año de la encuesta menos el año de su nacimiento
Edad = anoECV - datosFinal["PB140"].copy()
datosFinal.insert(7,"Edad",Edad)
datosFinal = datosFinal.drop(['PB140'],axis=1)
```

Sexo

Creación de nueva variable "Sexo" pasando los valores a tipo booleano (0 y 1)

```
#Mover y renombrar la variable "Sexo" (1 Varón, 2 Mujer) -> (0 Varón, 1 Mujer)
Sexo = datosFinal["PB150"] -1
datosFinal.insert(8,"Sexo",Sexo)
datosFinal = datosFinal.drop(['PB150'],axis=1)
```

Estado Civil

Renombrar variable PB190 y reduzco sus posibles valores a 4

```
EstadoCiv = datosFinal["PB190"].copy()
#Fusiono las categorías separado (3) y divorciado (5) y renombro valores.
for indice, valor in enumerate(EstadoCiv):
    if valor == '1': EstadoCiv[indice] = "Soltero"
    elif valor == '2': EstadoCiv[indice] = "Casado"
    elif valor == '4': EstadoCiv[indice] = "Viudo"
    else: EstadoCiv[indice] = "Divorciado/Separado"

datosFinal.insert(9,"Estado_Civil",EstadoCiv)
datosFinal = datosFinal.drop(['PB190'],axis=1)
```

En Formación

Creación de variable "En Formación" y pasar sus valores a tipo booleano (0 y 1)

```
# Conversión de valores (1 Sí, 2 No) -> (1 Sí, 0 No) con el cálculo: valor=|valor-2|
En_Formacion = abs(datosFinal["PE010"].astype(int) -2)
datosFinal.insert(10,"En_Formacion",En_Formacion)
datosFinal = datosFinal.drop(['PE010'],axis=1)
```

Nivel de estudios

Clasificación de niveles de estudios como hace eurostat: Educación baja (Niveles ISCED 0-2), media (Niveles 3-4) y alta (5-8)

```
Estudios = datosFinal["PE041"].astype(int)
#Estudios.sort_values().hist()    #La mayoría con estudios universitarios, después con sólo la educación secundaria básica.

for indice, valor in enumerate(Estudios):
    if valor < 300 :
        Estudios[indice] = "Bajo"
    elif valor < 500:
        Estudios[indice] = "Medio"
    else:
        Estudios[indice] = "Alto"

datosFinal.insert(11,"Niv_Estudios",Estudios)
datosFinal = datosFinal.drop(['PE041'],axis=1)
```

Situación laboral

Renombrado de posibles situaciones a Ocupado, Parado, Jubilado y Otros inactivos

```
# Valores actuales son trabajando (1), parado (2), jubilado (3), incapacitado, estudiante, labores del hogar... (4-8)
Trabajo = datosFinal["PL032"].copy()
for indice, valor in enumerate(Trabajo):
    if valor == '1': Trabajo[indice] = "Ocupado"
    elif valor == '2': Trabajo[indice] = "Parado"
    elif valor == '3': Trabajo[indice] = "Jubilado"
    else: Trabajo[indice] = "Otros inactivos"

datosFinal.insert(12,"Sit_Laboral",Trabajo)
datosFinal = datosFinal.drop(['PL032'],axis=1)
```

Enfermedad Crónica

Variable Salud_Grave binaria indica si el adulto responsable posee enfermedad o problema de salud grave. Reconversión de variables a binarias de tipo bool (0 o 1)

```
# Conversión de valores (1 Sí, 2 No) -> (1 Sí, 0 No) con el cálculo: valor=|valor-2|
Enfermedad = abs(datosFinal['PH020'].astype(int) -2)
datosFinal.insert(13,"Salud_Grave",Enfermedad)
datosFinal = datosFinal.drop(['PH020'],axis=1)
```

Guardar base de datos procesada

```
datosFinal.to_csv("datos_Pobreza_" + str(anoECV) + ".csv",index=False)
```

Comparación de modelos de predicción en Python

Este cuaderno va a programar la comparación de diversos modelos de predicción sobre una base de datos procesada del ECV. A partir de las distintas variables vamos a intentar predecir si un individuo es pobre o no.

```
import pandas as pd              # Manipulación de datasets como dataframes
import numpy as np               # Manipulación de arrays
import statsmodels.api as sm     # Modelos de econometría en python
import matplotlib.pyplot as plt  # Plotting en python
```

Preprocesado de datos

Carga de datos

Comenzamos cargando los datos a partir de un fichero csv:

```
anoECV = 2022
datos = pd.read_csv("datos_Pobreza_" + str(anoECV) + ".csv").convert_dtypes()
#datos.info()
```

Existen dos tipos de variables que requieren diferente tratamiento: las categóricas (binarias o con más categorías) y las numéricas.

- Las variables Region, Estado_Civil, Niv_Estudios y Sit_Laboral son variables categóricas no binarias.
- Las variables Hacinamiento y Edad son variables numéricas.
- El resto son variables categóricas binarias o dummie.

```
# Agrupación de variables que requieren procesamiento
var_cat = ['Region', 'Estado_Civil', 'Niv_Estudios', 'Sit_Laboral' ]
var_num = ['Hacinamiento', 'Edad', ]
```

Codificación de variables categóricas

Para statsmodels

Los modelos de statsmodels mejoran con la inclusión de un término constante y la creación de N-1 dummies por cada variable categórica de N categorías para evitar colinearidad.

```
# Función de pandas para automatizar la generación de dummies
datos_EC = pd.get_dummies(datos, columns=var_cat, drop_first= True).astype("float")

# Statsmodels requiere de la presencia de columna para el término constante
datos_EC.insert(1,"Const",1.0)
```

Para sklearn

Por cada variable categórica de N categorías, insertar N variables dummy

```
datos_ML = pd.get_dummies(datos, columns=var_cat, drop_first= False).astype("float")
```

División de datos en datos de enternamiento y datos de test

```
from sklearn.model_selection import train_test_split

X_train_EC, X_test_EC, Y_train_EC, Y_test_EC = train_test_split(datos_EC.drop(columns = ['Pobre']), \
                                                                datos_EC['Pobre'], \
                                                                train_size=0.7, \
                                                                random_state=45)
# 0.8) 23944 datos: 19155 datos para entrenar, 4789 datos para test
# 0.7) 23944 datos: 16760 datos para entrenar, 7184  datos para test
```

```
X_train_ML, X_test_ML, Y_train_ML, Y_test_ML = train_test_split(datos_ML.drop(columns = ['Pobre']), \
                                                                datos_ML['Pobre'], \
                                                                train_size=0.7, \
                                                                random_state=45)
```

Análisis descriptivo de los datos

```
import ydata_profiling
pd.set_option('display.max_columns', None)
display(datos.describe())
display(pd.DataFrame(datos_ML.std() / datos_ML.mean()).transpose())
profile_ML = ydata_profiling.ProfileReport(datos) # generamos el perfil de datos
#profile_ML
```

Modelos econométricos. Modelos Logit y Probit ¶

Implementación modelos logit y probit

```
# Crear el modelo y entrenarlo con los datos de entrenamiento
# Entrenamiento por defecto mediante método de máxima similitud (MLE)

logit = sm.Logit(Y_train_EC,X_train_EC).fit()
probit = sm.Probit(Y_train_EC,X_train_EC).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.433186
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.433078
         Iterations 6
```

Descomentar líneas del siguiente bloque de código para ver información sobre el modelo entrenado.

Se puede apreciar alguna variable con P-Valor superior a 0.05, es decir, poco influyente:

- Viv_PagadaOCedida
- Edad
- En_Formacion
- Algunas dummies (Regiones de Canarias, CastillaLaMancha, CeutaYMelilla y Valencia)

Se utiliza el método de estimación de Máxima Verosimilitud (MLE) y El valor de R-cuadrado es de 0.1548. Sólo el 15.48% de la variabilidad de la variable dependiente queda explicada por el modelo.

```
# Resumen descriptivo de resultados de regresión
#display(logit.summary())

# Resumen adicional con más resultados de la regresión
#display(logit.summary2())     #Otro resumen detalllado

# Tabla con listado de únicamente los coeficientes de cada variable
#pd.DataFrame(logit.params,columns=['coef']).sort_values('coef', ascending=False)
```

Aplicación de modelos sobre los datos de entrenamiento y datos test. La utilidad de comprobar la capacidad de predicción del modelo sobre los datos de entrenamiento es para evitar sobreajuste, las predicciones deben tener una puntería similar en ambos conjuntos de datos

```
# El método "predict" del modelo logit entrenado devuelve la probabilidad predicha.
prediccion_logit_train_prob  = logit.predict(X_train_EC)
prediccion_logit_test_prob   = logit.predict(X_test_EC)
prediccion_probit_train_prob = probit.predict(X_train_EC)
prediccion_probit_test_prob  = probit.predict(X_test_EC)

# Para la predicción, utilizar iterador MAP para "redondear" la probabilidad a  0 o 1.
# Umbral de decisión efectivo de 0.5
prediccion_logit_train  = list(map(round, prediccion_logit_train_prob))
prediccion_logit_test   = list(map(round, prediccion_logit_test_prob))
prediccion_probit_train = list(map(round, prediccion_probit_train_prob))
prediccion_probit_test  = list(map(round, prediccion_probit_test_prob))
```

Implementación de regresión logística como si fuera logit

Normalización de variables numéricas

Para que los modelos de sklearn converjan con mayor facilidad, es necesario normalizar las variables numéricas del dataset de entrenamiento para que tengan media = 0 y desviación típica = 1. Para evitar sesgo, se debe utilizar el tipificador entrenado con los datos de entrenamiento para la normalización del dataset de test.

El tipificador va a estar dentro de un "make_column_transformer" para afectar únicamente a las variables numéricas

```
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_transformer

normalizador = make_column_transformer((StandardScaler(), var_num),remainder='passthrough')
```

Cálculo de predicciones de regresión logística

Rehacer cálculo de predicción regresión logística para demostrar que a pesar de la diferente disciplina académica, es exactamente lo mismo que logit.

La regresión logística de sklearn admite varios parámetros ajustables como la penalización (innecesaria al ya haber normalizado), el parámetro C (1.0 por defecto) y el solver (lbfgs por defecto)

```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

# Pipeline con regresión logística
reg_log = make_pipeline(normalizador, LogisticRegression(penalty = None))

### Solver:
# For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones;
# For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss;
# 'liblinear' is limited to one-versus-rest schemes.

reg_log.fit(X_train_ML, Y_train_ML)

# Predicción de probabilidades
prediccion_logreg_train_prob = reg_log.predict_proba(X_train_ML)[:,1]
prediccion_logreg_test_prob  = reg_log.predict_proba(X_test_ML)[:,1]

# Predicción de valores
prediccion_logreg_train = reg_log.predict(X_train_ML)
prediccion_logreg_test  = reg_log.predict(X_test_ML)

# Comprobar que ahora las variables tienen media cero, desviación 1
#display( pd.DataFrame(pipero.fit_transform(X_train[var_num])).describe() )
```

Comparación de modelos

```
from sklearn.metrics import accuracy_score

display(f'Los datos de entrenamiento logit aciertan {sum(prediccion_logit_train == Y_train_EC)} de {len(Y_train_EC)} veces '
        + f'con una tasa de acierto de {accuracy_score(Y_train_EC, prediccion_logit_train)*100:.4f}%')

display(f'Los datos de test logit aciertan {sum(prediccion_logit_test == Y_test_EC)} de {len(Y_test_EC)} veces ' \
        + f'con una tasa de acierto de {accuracy_score(Y_test_EC, prediccion_logit_test)*100:.4f}%')

display(f'Los datos de entrenamiento probit aciertan {sum(prediccion_probit_train == Y_train_EC)} de {len(Y_train_EC)} veces
        + f'con una tasa de acierto de {accuracy_score(Y_train_EC, prediccion_probit_train)*100:.4f}%')

display(f'Los datos de test probit aciertan {sum(prediccion_probit_test == Y_test_EC)} de {len(Y_test_EC)} veces ' \
        + f'con una tasa de acierto de {accuracy_score(Y_test_EC, prediccion_probit_test)*100:.4f}%')

display(f'Los datos de entrenamiento de regresión logística aciertan {sum(prediccion_logreg_train == Y_train_ML)} de {len(Y_t
        + f'con una tasa de acierto de {reg_log.score(X_train_ML, Y_train_ML)*100:.4f} % ')

### Tasa de acierto calculable también como accuracy_score(Y_test, yhat_test)
display(f'Los datos de test de regresión logística aciertan {sum(prediccion_logreg_test == Y_test_ML)} de {len(Y_test_ML)} ve
        + f'con una tasa de acierto de {reg_log.score(X_test_ML, Y_test_ML)*100:.4f} % ')

### Ver coeficientes de cada variable del modelo
#print(reg_log[1].coef_)
```

```
'Los datos de entrenamiento logit aciertan 13591 de 16760 veces con una tasa de acierto de 81.0919%'

'Los datos de test logit aciertan 5850 de 7184 veces con una tasa de acierto de 81.4310%'

'Los datos de entrenamiento probit aciertan 13594 de 16760 veces con una tasa de acierto de 81.1098%'
```

'Los datos de test probit aciertan 5849 de 7184 veces con una tasa de acierto de 81.4170%'

'Los datos de entrenamiento de regresión logística aciertan 13591 de 16760 veces con una tasa de acierto de 81.0919 % '

'Los datos de test de regresión logística aciertan 5850 de 7184 veces con una tasa de acierto de 81.4310 % '

```
### Para matrices de confusión mucho más visuales
#from sklearn.metrics import ConfusionMatrixDisplay
#display(ConfusionMatrixDisplay(confusion_matrix(Y_testEC, prediccion_logit_test)).plot())
#display(ConfusionMatrixDisplay(confusion_matrix(Y_testEC, prediccion_probit_test)).plot())

print ("Matriz de confusión de entrenamiento Logit:")
display(pd.crosstab(Y_train_EC, prediccion_logit_train, colnames = ["Predicciones"], margins = True))

print ("Matriz de confusión de test Logit:")
display(pd.crosstab(Y_test_EC, prediccion_logit_test, colnames = ["Predicciones"], margins = True))

print ("Matriz de confusión de entrenamiento Probit:")
display(pd.crosstab(Y_train_EC, prediccion_probit_train, colnames = ["Predicciones"], margins = True))

print ("Matriz de confusión de test Probit:")
display(pd.crosstab(Y_test_EC, prediccion_probit_test, colnames = ["Predicciones"], margins = True))

print ("Matriz de confusión de entrenamiento de Regresión Logística:")
display(pd.crosstab(Y_train_ML, prediccion_logreg_train, colnames = ["Predicciones"], margins = True))

print ("Matriz de confusión de test de Regresión Logística:")
display(pd.crosstab(Y_test_ML, prediccion_logreg_test, colnames = ["Predicciones"], margins = True))
```

Matriz de confusión de entrenamiento Logit:

Predicciones	0	1	All
Pobre			
0.0	12758	480	13238
1.0	2689	833	3522
All	15447	1313	16760

Matriz de confusión de test Logit:

Predicciones	0	1	All
Pobre			
0.0	5533	207	5740
1.0	1127	317	1444
All	6660	524	7184

Matriz de confusión de entrenamiento Probit:

Predicciones	0	1	All
Pobre			
0.0	12785	453	13238
1.0	2713	809	3522
All	15498	1262	16760

Matriz de confusión de test Probit:

Predicciones	0	1	All
Pobre			
0.0	5542	198	5740
1.0	1137	307	1444
All	6679	505	7184

Matriz de confusión de entrenamiento de Regresión Logística:

Predicciones	0.0	1.0	All
Pobre			
0.0	12758	480	13238
1.0	2689	833	3522
All	15447	1313	16760

Matriz de confusión de test de Regresión Logística:

Predicciones	0.0	1.0	All
Pobre			
0.0	5533	207	5740
1.0	1127	317	1444
All	6660	524	7184

Las métricas del reporte de clasificación sirven para medir la calidad de las predicciones de un algoritmo de predicción:

- "precision" es la ratio de acierto al predecir positivo: tp / (tp + fp)
- "recall" o exhaustividad es la ratio de positivos correctamente clasificados: tp / (tp + fn)
- "f-beta" es la media armónica de la precisión y el recall. A mayor factor beta mayor peso tiene el recall. Beta=1 otorga igual importancia a ambas
- "support" es el nº de real de valores de cada tipo en Y_test

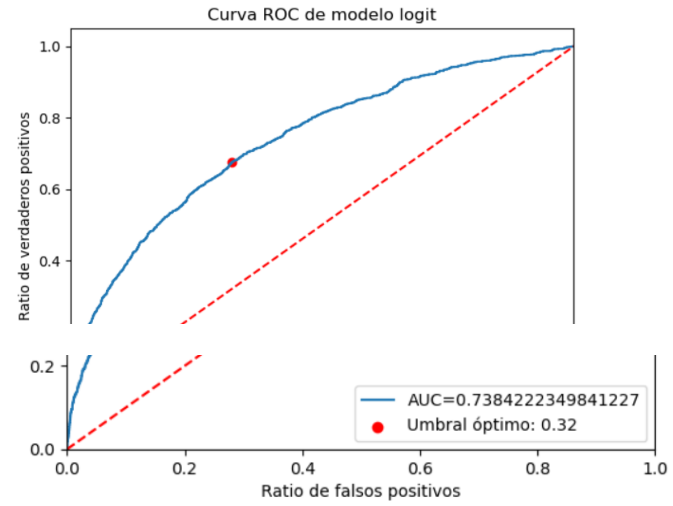
```
from sklearn.metrics import classification_report

print ("Reporte de clasificación de entrenamiento Logit:")
print(classification_report(Y_train_EC, prediccion_logit_train))
print ("Reporte de clasificación de test Logit:")
print(classification_report(Y_test_EC, prediccion_logit_test))
print ("Reporte de clasificación de entrenamiento Probit:")
print(classification_report(Y_train_EC, prediccion_probit_train))
print ("Reporte de clasificación de test Probit:")
print(classification_report(Y_test_EC, prediccion_probit_test))
print ("Reporte de clasificación de entrenamiento de Regresión Logística:")
print(classification_report(Y_train_ML, prediccion_logreg_train))
print ("Reporte de clasificación de test de Regresión Logística:")
print(classification_report(Y_test_ML, prediccion_logreg_test))
```

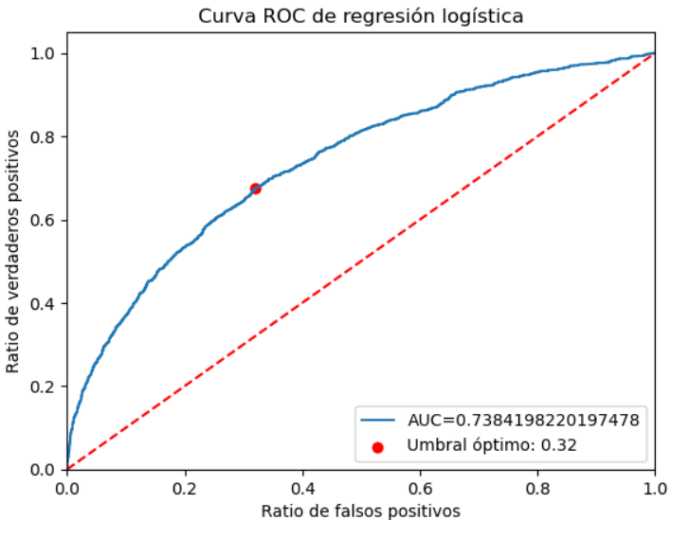
```
Reporte de clasificación de entrenamiento Logit:
              precision    recall  f1-score   support

         0.0       0.83      0.96      0.89     13238
         1.0       0.63      0.24      0.34      3522

    accuracy                           0.81     16760
   macro avg       0.73      0.60      0.62     16760
weighted avg       0.79      0.81      0.78     16760


Reporte de clasificación de test Logit:
              precision    recall  f1-score   support

         0.0       0.83      0.96      0.89      5740
         1.0       0.60      0.22      0.32      1444

    accuracy                           0.81      7184
   macro avg       0.72      0.59      0.61      7184
weighted avg       0.79      0.81      0.78      7184

Reporte de clasificación de entrenamiento Probit:
              precision    recall  f1-score   support

         0.0       0.82      0.97      0.89     13238
         1.0       0.64      0.23      0.34      3522

    accuracy                           0.81     16760
   macro avg       0.73      0.60      0.61     16760
weighted avg       0.79      0.81      0.77     16760

Reporte de clasificación de test Probit:
              precision    recall  f1-score   support

         0.0       0.83      0.97      0.89      5740
         1.0       0.61      0.21      0.32      1444

    accuracy                           0.81      7184
   macro avg       0.72      0.59      0.60      7184
weighted avg       0.79      0.81      0.78      7184


Reporte de clasificación de entrenamiento de Regresión Logística:
              precision    recall  f1-score   support

         0.0       0.83      0.96      0.89     13238
         1.0       0.63      0.24      0.34      3522

    accuracy                           0.81     16760
   macro avg       0.73      0.60      0.62     16760
weighted avg       0.79      0.81      0.78     16760

Reporte de clasificación de test de Regresión Logística:
              precision    recall  f1-score   support

         0.0       0.83      0.96      0.89      5740
         1.0       0.60      0.22      0.32      1444

    accuracy                           0.81      7184
   macro avg       0.72      0.59      0.61      7184
weighted avg       0.79      0.81      0.78      7184
```

Por último, visualización de la curva ROC

```python
def plot_roc(fpr, tpr, AUC, title, save=False):
    plt.figure()
    plt.plot(fpr, tpr, label="AUC="+str(AUC))
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.])
    plt.ylim([0, 1.05])
    plt.xlabel('Ratio de falsos positivos')
    plt.ylabel('Ratio de verdaderos positivos')
    plt.title(title)
    plt.legend(loc="lower right")

    # Encontrar el umbral óptimo
    optimal_threshold = fpr[np.argmax(tpr - fpr)]

    # Mostrar el umbral óptimo en el gráfico
    plt.scatter(optimal_threshold, tpr[np.argmax(tpr - fpr)], c='red', label="Umbral óptimo: {:.2f}".format(optimal_threshold
    plt.legend(loc="lower right")

    plt.savefig('ROC_logit') if save else None
    plt.show()
```

```python
from sklearn.metrics import roc_curve, roc_auc_score

#true positive rate es la sensitividad, false positive rate es specificity
fpr_logit, tpr_logit, _   = roc_curve(Y_test_EC, prediccion_logit_test_prob)
fpr_logreg, tpr_logreg, _ = roc_curve(Y_test_ML, prediccion_logreg_test_prob)

#AUC es área bajo la curva ROC, cuanto más alta mejor, de 0 a 1.
AUC_logit  = roc_auc_score(Y_test_EC, prediccion_logit_test_prob)
AUC_logreg = roc_auc_score(Y_test_ML, prediccion_logreg_test_prob)

plot_roc(fpr_logit, tpr_logit, AUC_logit, "Curva ROC de modelo logit")
plot_roc(fpr_logreg, tpr_logreg, AUC_logreg, "Curva ROC de regresión logística")
```





Predicción con el umbral óptimo adaptado

La ROC nos muestra como debido a la desigual distribución de la variable objetivo, es recomendable tener un umbral diferente a 0,5

```
# Umbral óptimo de 0.3
prediccion_logit_test_opt = np.where(prediccion_logit_test_prob >= 0.32, 1, 0)
prediccion_logreg_test_opt = np.where(prediccion_logreg_test_prob >= 0.32, 1, 0)

display(f'Los datos de test logit aciertan {sum(prediccion_logit_test_opt == Y_test_EC)} de {len(Y_test_EC)} veces ' \
        + f'con una tasa de acierto de {accuracy_score(Y_test_EC, prediccion_logit_test_opt)*100:.4f}%')

display(f'Los datos de test logreg aciertan {sum(prediccion_logreg_test_opt == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto de {accuracy_score(Y_test_ML, prediccion_logreg_test_opt)*100:.4f}%')

print ("Matriz de confusión de test Logit:")
display(pd.crosstab(Y_test_EC, prediccion_logit_test_opt, colnames = ["Predicciones"], margins = True))

print ("Matriz de confusión de regresión logística:")
display(pd.crosstab(Y_test_ML, prediccion_logreg_test_opt, colnames = ["Predicciones"], margins = True))

print(classification_report(Y_test_EC, prediccion_logit_test_opt))
print(classification_report(Y_test_ML, prediccion_logreg_test_opt))
```

```
'Los datos de test logit aciertan 5624 de 7184 veces con una tasa de acierto de 78.2850779510222%'

'Los datos de test logreg aciertan 5625 de 7184 veces con una tasa de acierto de 78.2989777728285%'
```

Matriz de confusión de test Logit:

Predicciones	0	1	All
Pobre			
0.0	5011	729	5740
1.0	831	613	1444
All	5842	1342	7184

Matriz de confusión de regresión logística:

Predicciones	0	1	All
Pobre			
0.0	5012	728	5740
1.0	831	613	1444
All	5843	1341	7184

```
              precision    recall  f1-score   support

         0.0       0.86      0.87      0.87      5740
         1.0       0.46      0.42      0.44      1444

    accuracy                           0.78      7184
   macro avg       0.66      0.65      0.65      7184
weighted avg       0.78      0.78      0.78      7184


              precision    recall  f1-score   support

         0.0       0.86      0.87      0.87      5740
         1.0       0.46      0.42      0.44      1444

    accuracy                           0.78      7184
   macro avg       0.66      0.65      0.65      7184
weighted avg       0.78      0.78      0.78      7184
```

Modelos de aprendizaje automático.

Regresión logística con ajuste de hiper-parámetros y K-folding

Al separar el dataset en los conjuntos de entrenamiento y test puedo estar entrenando sobre datos con un sesgo involuntario. Con kfolding hago la media de los resultados de aplicar diferentes segmentaciones para mitigar dicho sesgo.

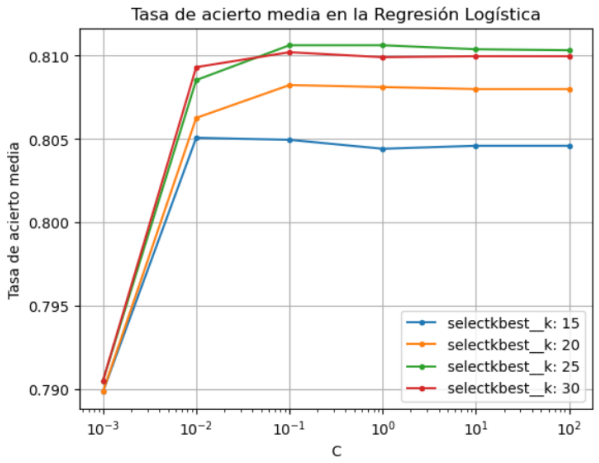
También es adecuado iterar sobre un subconjunto de validación (parte de los datos de entrenamiento) para calcular los mejor hiper-parámetros de ajuste a cada modelo

```
### Función para visualizar la puntería de cada interación del Cross-Validation
def get_cv_results_for_best_params(search_object):

    cv_results = search_object.cv_results_

    return [val[search_object.best_index_] for key, val in cv_results.items() if key.startswith("split")]
```

```
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif

reductor = SelectKBest(score_func=f_classif)

# Pipeline con regresión logística, esta vez con penalización l2 (la default)
reg_log_CV = make_pipeline(normalizador,
                           reductor,
                           LogisticRegression(penalty ='l2', random_state=0))

# La validación cruzada en red para comprobar hiper-parámetros
param_log = {
    'selectkbest__k': [ 15, 20, 25, 30],
#    'selectkbest__k': list(range(20, 30)),
    'logisticregression__C':[0.001, 0.01, 0.1, 1, 10, 100]}

busqueda_log = GridSearchCV(estimator = reg_log_CV,
                        param_grid = param_log,
                        scoring = 'accuracy',
                        cv = KFold(n_splits=10),
                        error_score='raise')

_ = busqueda_log.fit(X_train_ML, Y_train_ML)
```

```
# Función de antes para la tasa de acierto del CV
array_log = get_cv_results_for_best_params(busqueda_log)
media_log = np.asarray(array_log).mean()

display('Los mejores parámetros para la regresión logística dado el dataset son:', busqueda_log.best_params_)
display('Tasa de acierto en cada iteración de la CV con los mejores parámetros:', array_log)
display('Tasa de acierto media del de iteraciones de CV con los mejores parámetros:', media_log)

logreg_entrenado = busqueda_log.best_estimator_[2]

for num_vars in param_log['selectkbest__k']:
    mask = busqueda_log.cv_results_['param_selectkbest__k'] == num_vars
    C_values = busqueda_log.cv_results_['param_logisticregression__C'][mask]
    scores = busqueda_log.cv_results_['mean_test_score'][mask]
    plt.semilogx(C_values, scores, marker = ".",label = 'selectkbest__k: ' + str(num_vars))

plt.title('Tasa de acierto media en la Regresión Logística')
plt.ylabel('Tasa de acierto media')
plt.xlabel('C')
plt.grid(True)
plt.legend(loc="best")
plt.show()
```

```
'Los mejores parámetros para la regresión logística dado el dataset son:'

{'logisticregression__C': 0.1, 'selectkbest__k': 25}

'Tasa de acierto en cada iteración de la CV con los mejores parámetros:'

[0.7941527446300716,
 0.8233890214797136,
 0.7983293556085919,
 0.8114558472553699,
 0.815035799522673,
 0.801909307875895,
 0.8162291169451074,
 0.8269689737470167,
 0.8060859188544153,
 0.8126491646778043]

'Tasa de acierto media del de iteraciones de CV con los mejores parámetros:'

0.8106205250596659
```




```
selected_logreg_features = busqueda_log.best_estimator_[1].get_support()
nombre_variables_logreg = []
for feature_name, selected in zip(X_train_ML.columns, selected_logreg_features):
    if selected:
        nombre_variables_logreg.append(feature_name)

# DataFrame ordenado según el coeficiente de cada variable
importancia_predictores_logreg = pd.DataFrame(
                            {'Variable predictora': nombre_variables_logreg,
                             'Importancia': logreg_entrenado.coef_[0]}
                            )
display(importancia_predictores_logreg.sort_values('Importancia', ascending=False))

# DataFrame ordenado según el valor absoluto del coeficiente de cada variable
importancia_predictores_logreg['Importancia_abs'] = importancia_predictores_logreg['Importancia'].abs()
importancia_predictores_logreg_sorted = importancia_predictores_logreg.sort_values('Importancia_abs', ascending=False)
importancia_predictores_logreg_sorted = importancia_predictores_logreg_sorted.drop('Importancia_abs', axis=1)
display(importancia_predictores_logreg_sorted)

print("Características desechadas:")
for feature_name, selected in zip(X_train_ML.columns, selected_logreg_features):
    if not selected:
        print(feature_name)
```

	Variable predictora	Importancia
24	Sit_Laboral_Parado	0.880376
17	Estado_Civil_Divorciado/Separado	0.800719
1	Hacinamiento	0.705536
18	Estado_Civil_Soltero	0.676303
2	Urbanizacion	0.543738
12	Region_Extremadura	0.535501
20	Niv_Estudios_Bajo	0.494244
9	Region_CastillaLaMancha	0.460681
6	Region_Andalucia	0.416698
14	Region_Murcia	0.378074
23	Sit_Laboral_Otros inactivos	0.352764
11	Region_CeutaYMelilla	0.337964
8	Region_Canarias	0.272345
4	Sexo	0.125982
0	Familia_Numerosa	0.058128
5	PH020	-0.031743
3	Edad	-0.172404
16	Estado_Civil_Casado	-0.186904
13	Region_Madrid	-0.243806
10	Region_Catalunya	-0.279822
7	Region_Aragon	-0.308148
15	Region_Navarra	-0.459950
21	Sit_Laboral_Jubilado	-0.471496
22	Sit_Laboral_Ocupado	-0.766057
19	Niv_Estudios_Alto	-0.770611

	Variable predictora	Importancia
24	Sit_Laboral_Parado	0.880376
17	Estado_Civil_Divorciado/Separado	0.800719
19	Niv_Estudios_Alto	-0.770611
22	Sit_Laboral_Ocupado	-0.766057
1	Hacinamiento	0.705536
18	Estado_Civil_Soltero	0.676303
2	Urbanizacion	0.543738
12	Region_Extremadura	0.535501
20	Niv_Estudios_Bajo	0.494244
21	Sit_Laboral_Jubilado	-0.471496
9	Region_CastillaLaMancha	0.460681
15	Region_Navarra	-0.459950
6	Region_Andalucia	0.416698
14	Region_Murcia	0.378074
23	Sit_Laboral_Otros inactivos	0.352764
11	Region_CeutaYMelilla	0.337964
7	Region_Aragon	-0.308148
10	Region_Catalunya	-0.279822
8	Region_Canarias	0.272345
13	Region_Madrid	-0.243806
16	Estado_Civil_Casado	-0.186904
3	Edad	-0.172404
4	Sexo	0.125982
0	Familia_Numerosa	0.058128
5	PH020	-0.031743

```
Características desechadas:
Viv_PagadaOCedida
Ninos_Dep
En_Formacion
Region_Asturias
Region_Baleares
Region_Cantabria
Region_CastillaYLeon
Region_Galicia
Region_Pais Vasco
Region_Rioja
Region_Valencia
Estado_Civil_Viudo
Niv_Estudios_Medio
```

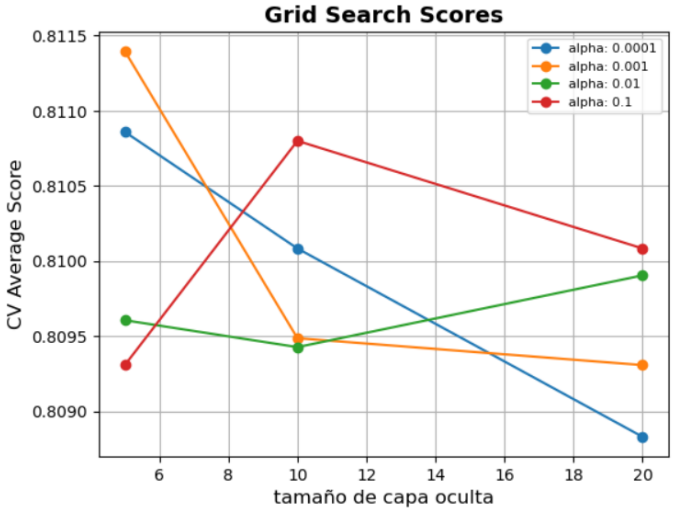
Red Neuronal (Clasificador MLP, MultiLayerPerceptron) de una sola capa

```python
# Función para visualización de posibles tamaños de capa oculta
def plot_grid_search(cv_results, grid_param_1, grid_param_2, xlabel, hiper_param):
    # Get Test Scores Mean and std for each grid search
    # Function obtained from:
    # https://stackoverflow.com/questions/37161563/how-to-graph-grid-scores-from-gridsearchcv
    scores_mean = cv_results['mean_test_score']
    scores_mean = np.array(scores_mean).reshape(len(grid_param_2),len(grid_param_1))

    scores_sd = cv_results['std_test_score']
    scores_sd = np.array(scores_sd).reshape(len(grid_param_2),len(grid_param_1))

    # Plot Grid search scores
    _, ax = plt.subplots(1,1)

    # Param1 is the X-axis, Param 2 is represented as a different curve (color line)
    for idx, val in enumerate(grid_param_2):
        ax.plot(grid_param_1, scores_mean[idx,:], '-o', label= hiper_param + ': ' + str(val))

    ax.set_title("Grid Search Scores", fontsize=14, fontweight='bold')
    ax.set_xlabel(xlabel, fontsize=12)
    ax.set_ylabel('CV Average Score', fontsize=12)
    ax.legend(loc="best", fontsize=8)
    ax.grid('on')
```

```python
from sklearn.neural_network import MLPClassifier
import warnings
from sklearn.exceptions import ConvergenceWarning

# Pipeline con perceptrón de una sola capa oculta
neuronal = make_pipeline(normalizador,
                         reductor,
                         MLPClassifier(random_state=None,max_iter=100))
# Solver = adam(default) debería ir bien, activation = relu(defaul) es un decisor linear pero tb puede ser logit

param_neuro = {
    'selectkbest__k': [15, 20, 25, 30],
    'mlpclassifier__hidden_layer_sizes':[(5,),(10,),(20,)],  # neuronas en la capa oculta
    'mlpclassifier__alpha':[1e-4, 1e-3, 0.01, 0.1]}          # fuerza del termino de regularizacion l2. analogo a C2

busqueda_neuro = GridSearchCV(estimator = neuronal,
                        param_grid = param_neuro,
                        scoring = 'accuracy',
                        cv = KFold(n_splits=10))

# Esto es para que no me salgan warnings por doquier
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=ConvergenceWarning)
    busqueda_neuro.fit(X_train_ML, Y_train_ML)
```

```python
# Función de antes para la tasa de acierto del CV
array_neuro = get_cv_results_for_best_params(busqueda_neuro)
media_neuro = np.asarray(array_neuro).mean()

display('Los mejores parámetros para la red neuronal dado el dataset son:', busqueda_neuro.best_params_)
display('Tasa de acierto en cada iteración de la CV con los mejores parámetros:', array_neuro)
display('Tasa de acierto media del de iteraciones de CV con los mejores parámetros:', media_neuro)

# Obtener el mejor modelo entrenado
neur_entrenado = busqueda_neuro.best_estimator_[2]

mask = busqueda_neuro.cv_results_['param_selectkbest__k'] == busqueda_neuro.best_params_['selectkbest__k']
cv_results_filtrado = {k: [v[i] for i in range(len(v)) if mask[i]] for k, v in busqueda_neuro.cv_results_.items()}

plot_grid_search(cv_results_filtrado,
                 param_neuro["mlpclassifier__hidden_layer_sizes"],
                 param_neuro["mlpclassifier__alpha"],
                 "tamaño de capa oculta",
                 "alpha")
```

```
'Los mejores parámetros para la red neuronal dado el dataset son:'

{'mlpclassifier__alpha': 0.001,
 'mlpclassifier__hidden_layer_sizes': (5,),
 'selectkbest__k': 25}

'Tasa de acierto en cada iteración de la CV con los mejores parámetros:'

[0.798926014319809,
 0.8233890214797136,
 0.7995226730310262,
 0.8108591885441527,
 0.818019093078759,
 0.7941527446300716,
 0.8215990453460621,
 0.8299522673031027,
 0.8025059665871122,
 0.815035799522673]

'Tasa de acierto media del de iteraciones de CV con los mejores parámetros:'

0.8113961813842483
```




```python
# Entender que en una red neuronal, los coeficientes son los pesos asociados a cada input (columnas)
# para cada neurona (fila)
# La segunda lista es la última capa, la cual tiene tantos pesos como neuronas haya en la capa oculta
display(neur_entrenado.coefs_)

# Calculo la importancia de cada variable calculando la media de los pesos asociados a cada variable
# en las neuronas de la capa oculta
importancia_predictores_neuro = []
for layer in neur_entrenado.coefs_:
    layer_importance = np.abs(layer).mean(axis=1)
    importancia_predictores_neuro.append(layer_importance)
```

```
[array([[-0.0300235 ,  0.29951079,  0.17493516, -0.01681862, -0.09641072],
        [-0.67847456,  0.61044649,  0.12493645, -0.10938713, -0.15884383],
        [-0.21229967,  0.23340917, -0.37297956, -0.01803394, -0.16524622],
        [ 0.14563394,  0.17289262, -0.21148244,  0.45054604, -0.27558326],
        [ 0.33472441,  0.43536229, -0.11790698, -0.00351632,  0.13560823],
        [ 0.17954789,  0.66027459,  0.45596579, -0.1019978 , -0.20203399],
        [-0.41650181, -0.01951704,  0.06409699, -0.31193364,  0.1905643 ],
        [-0.01364674,  0.13453595,  0.32341434,  0.41556963,  0.70143973],
        [-0.38521679,  0.3771618 ,  0.14668813, -0.00387247, -0.9058201 ],
        [ 0.05662245,  0.29807039, -0.53547927,  0.06312886, -0.08438877],
        [ 0.1067579 , -0.20187694,  0.01165858,  0.09032546,  0.01735942],
        [ 0.50791863,  0.86508763, -0.05701087, -0.24383481,  0.12281491],
        [-0.0744439 ,  0.02113902, -0.52805754, -0.12386116,  0.05290216],
        [ 0.35534049,  0.14057303,  0.37949929, -0.38294843, -0.19116452],
        [ 0.02726322,  0.25203497, -0.42630871,  0.10187311,  0.0410163 ],
        [ 0.51252841, -0.15263161,  0.02516604,  0.33364894,  0.63906478],
        [ 0.08947264, -0.04095002, -0.2683269 ,  0.37806103, -0.21883779],
        [-0.22170379,  0.35358708, -0.0885069 , -0.54870247,  0.2452563 ],
        [ 0.0348722 ,  0.28954931, -0.1721297 , -0.50490831,  0.3983635 ],
        [ 0.1542362 , -0.52439674,  0.63585589, -0.15566689,  0.33484138],
        [-0.36507361,  0.64253506, -0.01202834,  0.15950406, -0.38817804],
        [ 0.24064917, -0.24828408,  0.1402865 ,  0.38667904, -0.35163895],
        [ 0.77696863, -0.24820457,  0.14622384,  0.24939301, -0.39236848],
        [-0.4076533 ,  0.39839555,  0.22627314,  0.32520732,  0.46437021],
        [-0.13705584, -0.27240022, -0.5491975 , -0.17522896,  0.85887624]]),
 array([[-0.78582399],
        [ 0.60373255],
        [-0.86888444],
        [-0.75724386],
        [ 1.11082725]])]
```

```python
selected_neuro_features = busqueda_neuro.best_estimator_[1].get_support()
nombre_variables_neuro = []
for feature_name, selected in zip(X_train_ML.columns, selected_neuro_features):
    if selected:
        nombre_variables_neuro.append(feature_name)

importancia_predictores_neuro = pd.DataFrame(
                            {'Variable predictora': nombre_variables_neuro,
                             'Importancia': importancia_predictores_neuro[0]}
                            )
display(importancia_predictores_neuro.sort_values('Importancia', ascending=False))


print("Características desechadas:")
for feature_name, selected in zip(X_train_ML.columns, selected_neuro_features):
    if not selected:
        print(feature_name)
```

	Variable predictora	Importancia
24	Sit_Laboral_Parado	0.398552
23	Sit_Laboral_Otros inactivos	0.364380
8	Region_Canarias	0.363752
22	Sit_Laboral_Ocupado	0.362632
19	Niv_Estudios_Alto	0.360999
11	Region_CeutaYMelilla	0.359333
1	Hacinamiento	0.336418
15	Region_Navarra	0.332608
5	PH020	0.319964
7	Region_Aragon	0.317721
20	Niv_Estudios_Bajo	0.313464
17	Estado_Civil_Divorciado/Separado	0.291551
13	Region_Madrid	0.289905
18	Estado_Civil_Soltero	0.279965
21	Sit_Laboral_Jubilado	0.273508
3	Edad	0.251228
9	Region_CastillaLaMancha	0.207538
4	Sexo	0.205424
6	Region_Andalucia	0.200523
2	Urbanizacion	0.200394
16	Estado_Civil_Casado	0.199130
14	Region_Murcia	0.169699
12	Region_Extremadura	0.160081
0	Familia_Numerosa	0.123540
10	Region_Catalunya	0.085596

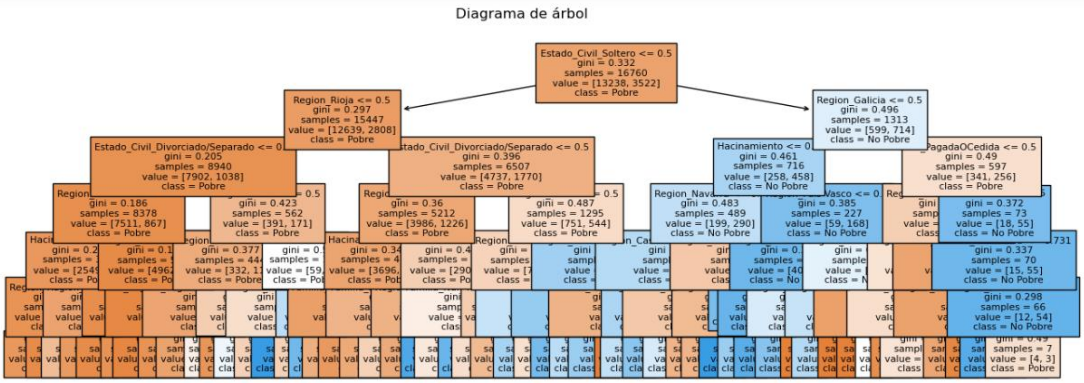
```
Características desechadas:
Viv_PagadaOCedida
Ninos_Dep
En_Formacion
Region_Asturias
Region_Baleares
Region_Cantabria
Region_CastillaYLeon
Region_Galicia
Region_Pais Vasco
Region_Rioja
Region_Valencia
Estado_Civil_Viudo
Niv_Estudios_Medio
```

Árbol de clasificación

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV #con prueba de hiperparámetros continua, pruebo al azar.

# Pipeline con modelo de árbol CART. El kernel default ya es el RBF
arbol = make_pipeline(normalizador,
                      reductor,
                      DecisionTreeClassifier())

param_arb = {
    'selectkbest__k': [20, 25, 30, 35],
    'decisiontreeclassifier__max_depth': [2, 4, 6, 8, 10],
    'decisiontreeclassifier__min_samples_split': [2, 3, 4, 5]}

# Se prueban "n_iter" (por defecto 10) combinaciones de los parámetros de "param_arb" en cada validación cruzada.
busqueda_arb = GridSearchCV(estimator = arbol,
                            param_grid = param_arb,
                            scoring = 'accuracy',
                            cv = KFold(n_splits=10))

with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=ConvergenceWarning)
    busqueda_arb.fit(X_train_ML, Y_train_ML)
```

```python
from sklearn.tree import plot_tree

# Función de antes para la tasa de acierto del CV
array_arb = get_cv_results_for_best_params(busqueda_arb)
media_arb = np.asarray(array_arb).mean()

display('Los mejores parámetros para el árbol de clasificación dado el dataset son:', busqueda_arb.best_params_)
display('Tasa de acierto en cada iteración de la CV con los mejores parámetros:', array_arb)
display('Tasa de acierto media del de iteraciones de CV con los mejores parámetros:', media_arb)

# Obtener el mejor modelo entrenado
arbol_entrenado = busqueda_arb.best_estimator_[2]

fig = plt.figure(figsize=(15,5))
fig.suptitle('Diagrama de árbol')
_ = plot_tree(arbol_entrenado, #el _ es porque suelta mucho spam por consola
              class_names= ['Pobre', 'No Pobre'],
              feature_names = X_train_ML.columns,
              filled=True,
              fontsize=8) # Modificar el fontsize para mayor o menor legibilidad
```

```
'Los mejores parámetros para el árbol de clasificación dado el dataset son:'

{'decisiontreeclassifier__max_depth': 6,
 'decisiontreeclassifier__min_samples_split': 5,
 'selectkbest__k': 30}

'Tasa de acierto en cada iteración de la CV con los mejores parámetros:'

[0.7899761336515513,
 0.818019093078759,
 0.7977326968973747,
 0.7977326968973747,
 0.8054892601431981,
 0.8013126491646778,
 0.8186157517899761,
 0.8227923627684964,
 0.8102625298329356,
 0.8048926014319809]

'Tasa de acierto media del de iteraciones de CV con los mejores parámetros:'

0.8066825775656324
```




```python
selected_arb_features = busqueda_arb.best_estimator_[1].get_support()
nombre_variables_arb = []
for feature_name, selected in zip(X_train_ML.columns, selected_arb_features):
    if selected:
        nombre_variables_arb.append(feature_name)

importancia_predictores_arb = pd.DataFrame(
                            {'Variable predictora': nombre_variables_arb,
                             'Importancia': arbol_entrenado.feature_importances_}
                            )
display(importancia_predictores_arb.sort_values('Importancia', ascending=False))

print("Características desechadas:")
for feature_name, selected in zip(X_train_ML.columns, selected_arb_features):
    if not selected:
        print(feature_name)
```

	Variable predictora	Importancia
29	Sit_Laboral_Parado	0.337578
25	Niv_Estudios_Bajo	0.194852
28	Sit_Laboral_Otros inactivos	0.120642
20	Estado_Civil_Casado	0.051464
24	Niv_Estudios_Alto	0.045666
2	Urbanizacion	0.044115
0	Familia_Numerosa	0.043827
21	Estado_Civil_Divorciado/Separado	0.038717
22	Estado_Civil_Soltero	0.029044
1	Hacinamiento	0.023903
4	Sexo	0.015830
3	Edad	0.010795
9	Region_Canarias	0.008656
13	Region_CeutaYMelilla	0.007247
23	Estado_Civil_Viudo	0.007175
15	Region_Madrid	0.005766
12	Region_Catalunya	0.004752
11	Region_CastillaYLeon	0.004510
19	Region_Valencia	0.002164
5	PH020	0.001847
16	Region_Murcia	0.001451
27	Sit_Laboral_Ocupado	0.000000
26	Sit_Laboral_Jubilado	0.000000
7	Region_Aragon	0.000000
6	Region_Andalucia	0.000000
8	Region_Baleares	0.000000
17	Region_Navarra	0.000000
14	Region_Extremadura	0.000000
10	Region_CastillaLaMancha	0.000000
18	Region_Pais Vasco	0.000000

```
Características desechadas:
Viv_PagadaOCedida
Ninos_Dep
En_Formacion
Region_Asturias
Region_Cantabria
Region_Galicia
Region_Rioja
Niv_Estudios_Medio
```

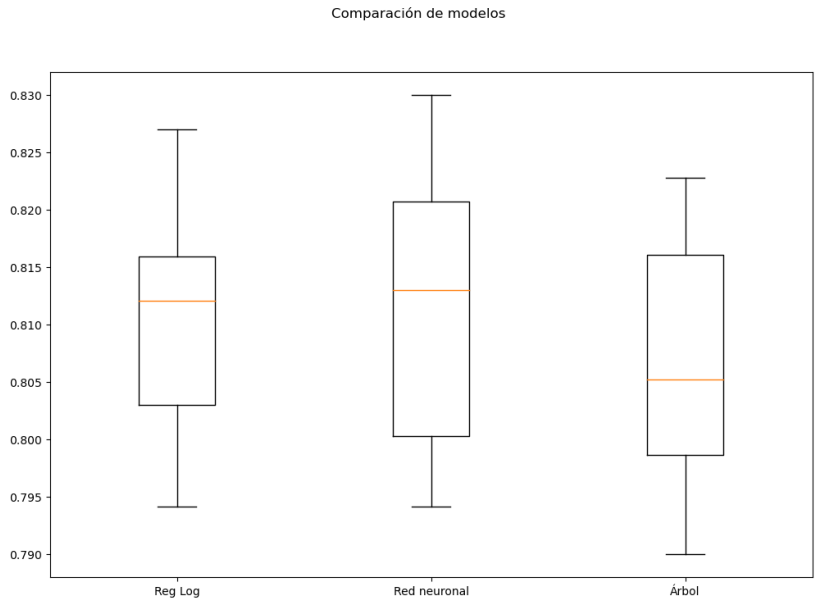
```
##### Si tengo graphviz instalado en el ordenador, puedo crear el árbol de decisión con esta herramienta gráfica
##### Si no lo tengo, ignorar. Los umbrales de decisión siguen sinedo los de variables numéricas normalizadas
### Descomentar y ejecutar al menos una vez por ordenador
###! pip install graphviz

import graphviz
from sklearn.tree import export_graphviz

dot_data_arb = export_graphviz(arbol_entrenado, out_file=None,
                               class_names= ['Pobre', 'No Pobre'],
                               feature_names=nombre_variables_arb,
                               filled=True)

### Draw graph
graph_arb = graphviz.Source(dot_data_arb, filename="arbol_decision.png", format="png")
display(graph_arb)
graph_arb.render(filename='arbol_decision')
```

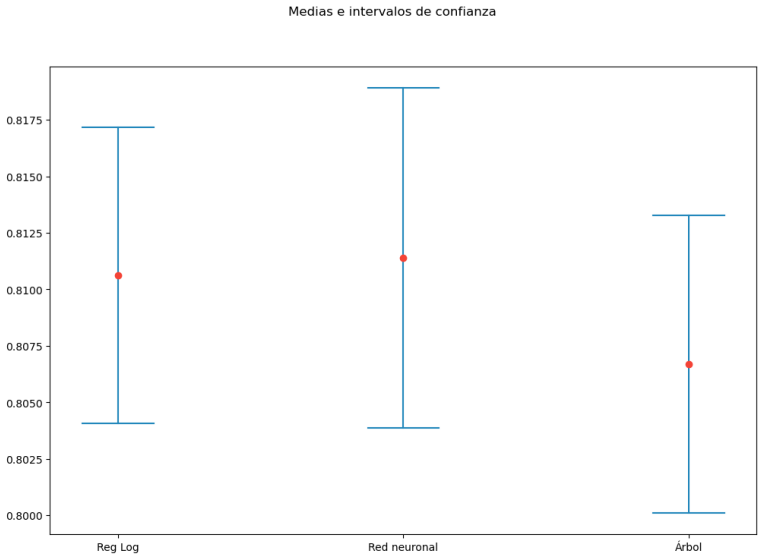
COMPARATIVA

```
import statistics
from math import sqrt

#función que he encontrado para hacer plot de intervalos de confianza
def plot_confidence_interval(x, values, z=1.96, color='#2187bb', horizontal_line_width=0.25):
    mean = statistics.mean(values)
    stdev = statistics.stdev(values)
    confidence_interval = z * stdev / sqrt(len(values))

    left = x - horizontal_line_width / 2
    top = mean - confidence_interval
    right = x + horizontal_line_width / 2
    bottom = mean + confidence_interval
    plt.plot([x, x], [top, bottom], color=color)
    plt.plot([left, right], [top, top], color=color)
    plt.plot([left, right], [bottom, bottom], color=color)
    plt.plot(x, mean, 'o', color='#f44336')
    return mean, confidence_interval
```

Intervalos de confianza de los resultados de las validaciones cruzadas

```
resultados = [array_log, array_neuro, array_arb]
modelos = ['Reg Log', 'Red neuronal', 'Árbol']

# diagrama de caja y bigotes
fig = plt.figure(figsize=(12,8))
fig.suptitle('Comparación de modelos')
ax = fig.add_subplot(111)   #para definir el eje inferior
plt.boxplot(resultados)
ax.set_xticklabels(modelos)
plt.show()
#aparentemente el modelo de árbol de decisión tiene una mejor mediana, además de un máximo de éxito superior.

fig = plt.figure(figsize=(12,8))
fig.suptitle('Medias e intervalos de confianza')
plt.xticks([1, 2, 3], modelos)
plot_confidence_interval(1, resultados[0], z=1.96) #valor crítico 1.95 equivale al 95% de confianza
plot_confidence_interval(2, resultados[1], z=1.96)
plot_confidence_interval(3, resultados[2], z=1.96)
plt.show()
#de aquí parece que el modelo random forest tiene una media ligeramente superior, al igual que su intervalo de confianza
```





Predicción mediante datos test

```
### Predicciones
prediccion_log   = busqueda_log.predict(X_test_ML)
prediccion_arb   = busqueda_arb.predict(X_test_ML)
prediccion_neuro = busqueda_neuro.predict(X_test_ML)

### Probabilidades
prediccion_log_prob   = busqueda_log.predict_proba(X_test_ML)[:,1]
prediccion_arb_prob   = busqueda_arb.predict_proba(X_test_ML)[:,1]
prediccion_neuro_prob = busqueda_neuro.predict_proba(X_test_ML)[:,1]
```

```
display(f'La regresión logística acierta {sum(prediccion_log == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_log)*100:.4f}% ')

display(f'El árbol de clasificación a acierta {sum(prediccion_arb == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_arb)*100:.4f}% ')

display(f'La red neuronal acierta {sum(prediccion_neuro == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_neuro)*100:.4f}% ')
```

```
'La regresión logística acierta 5853 de 7184 veces con una tasa de acierto media de 81.47271714922049% '

'El árbol de clasificación a acierta 5842 de 7184 veces con una tasa de acierto media de 81.3195991091314% '

'La red neuronal acierta 5855 de 7184 veces con una tasa de acierto media de 81.50055679287304% '
```

```
print ("Matriz de confusión de regresión logística:")
display(pd.crosstab(Y_test_ML, prediccion_log, colnames = ["Predicciones"], margins = True))
print ("Matriz de confusión de árbol de clasificacióin:")
display(pd.crosstab(Y_test_ML, prediccion_arb, colnames = ["Predicciones"], margins = True))
print ("Matriz de confusión de red neuronal:")
display(pd.crosstab(Y_test_ML, prediccion_neuro, colnames = ["Predicciones"], margins = True))
```

```
Matriz de confusión de regresión logística:
```

Predicciones	0.0	1.0	All
Pobre			
0.0	5545	195	5740
1.0	1136	308	1444
All	6681	503	7184

```
Matriz de confusión de árbol de clasificacióin:
```

Predicciones	0.0	1.0	All
Pobre			
0.0	5594	146	5740
1.0	1196	248	1444
All	6790	394	7184

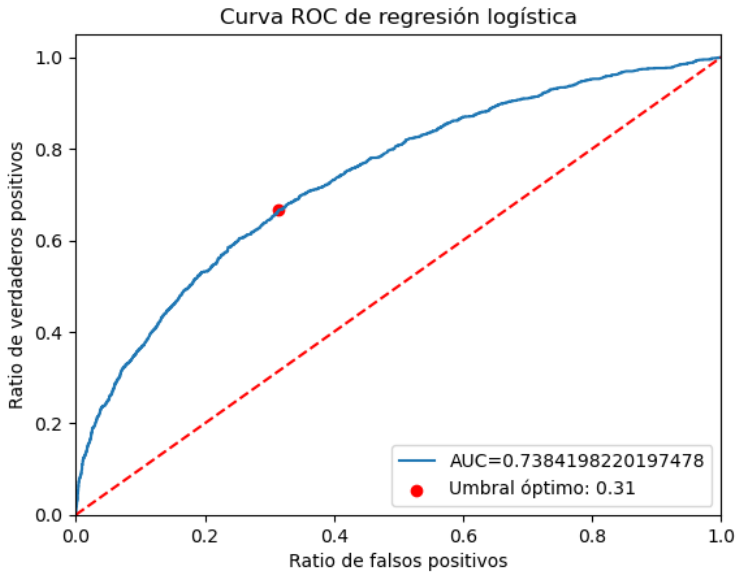
Matriz de confusión de red neuronal:

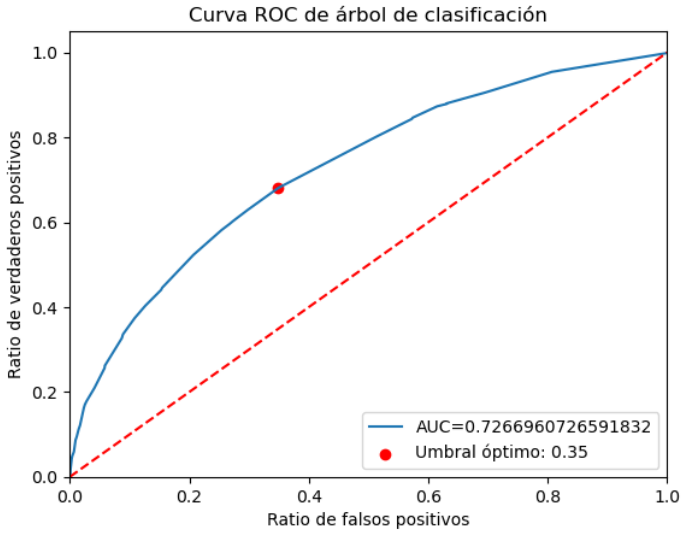
Predicciones	0.0	1.0	All
Pobre			
0.0	5523	217	5740
1.0	1112	332	1444
All	6635	549	7184

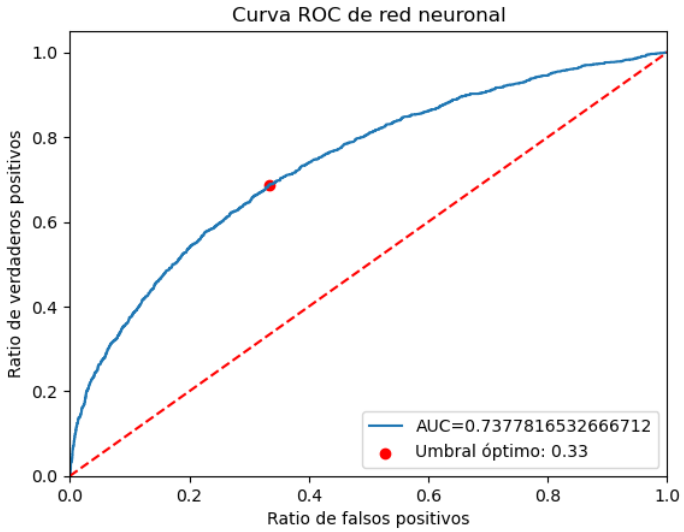
```
print ("Reporte de clasificación de regresión logística:")
print(classification_report(Y_test_ML, prediccion_log))
print ("Reporte de clasificación de árbol de clasificacióin:")
print(classification_report(Y_test_ML, prediccion_arb))
print ("Reporte de clasificación de red neuronal:")
print(classification_report(Y_test_ML, prediccion_neuro))
```

```
Reporte de clasificación de regresión logística:
              precision    recall  f1-score   support

         0.0       0.83      0.97      0.89      5740
         1.0       0.61      0.21      0.32      1444

    accuracy                           0.81      7184
   macro avg       0.72      0.59      0.60      7184
weighted avg       0.79      0.81      0.78      7184

Reporte de clasificación de árbol de clasificacióin:
              precision    recall  f1-score   support

         0.0       0.82      0.97      0.89      5740
         1.0       0.63      0.17      0.27      1444

    accuracy                           0.81      7184
   macro avg       0.73      0.57      0.58      7184
weighted avg       0.78      0.81      0.77      7184


Reporte de clasificación de red neuronal:
              precision    recall  f1-score   support

         0.0       0.83      0.96      0.89      5740
         1.0       0.60      0.23      0.33      1444

    accuracy                           0.82      7184
   macro avg       0.72      0.60      0.61      7184
weighted avg       0.79      0.82      0.78      7184
```

```
#true positive rate es la sensitividad, false positive rate es specificity
fpr_log, tpr_log, _     = roc_curve(Y_test_ML, prediccion_log_prob)
fpr_arb, tpr_arb, _     = roc_curve(Y_test_ML, prediccion_arb_prob)
fpr_neuro, tpr_neuro, _ = roc_curve(Y_test_ML, prediccion_neuro_prob)

#AUC es área bajo la curva ROC, cuanto más alta mejor, de 0 a 1.
AUC_log   = roc_auc_score(Y_test_ML, prediccion_log_prob)
AUC_arb   = roc_auc_score(Y_test_ML, prediccion_arb_prob)
AUC_neuro = roc_auc_score(Y_test_ML, prediccion_neuro_prob)

plot_roc(fpr_log, tpr_log, AUC_logreg, "Curva ROC de regresión logística")
plot_roc(fpr_arb, tpr_arb, AUC_arb, "Curva ROC de árbol de clasificación")
plot_roc(fpr_neuro, tpr_neuro, AUC_neuro, "Curva ROC de red neuronal")
```


Curva ROC de regresión logística
Ratio de verdaderos positivos
Ratio de falsos positivos
AUC=0.7384198220197478
Umbral óptimo: 0.31


Curva ROC de árbol de clasificación
Ratio de verdaderos positivos
Ratio de falsos positivos
AUC=0.7266960726591832
Umbral óptimo: 0.35


Curva ROC de red neuronal
Ratio de verdaderos positivos
Ratio de falsos positivos
AUC=0.7377816532666712
Umbral óptimo: 0.33

```
# Umbrales óptimos de cada modelo
prediccion_log_opt   = np.where(prediccion_log_prob >= 0.31, 1, 0)
prediccion_arb_opt   = np.where(prediccion_arb_prob >= 0.35, 1, 0)
prediccion_neuro_opt = np.where(prediccion_neuro_prob >= 0.33, 1, 0)

display(f'La regresión logística acierta {sum(prediccion_log_opt == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_log_opt)*100}% ')

display(f'El árbol de clasificación a acierta {sum(prediccion_arb_opt == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_arb_opt)*100}% ')

display(f'La red neuronal acierta {sum(prediccion_neuro_opt == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_neuro_opt)*100}% ')

print ("Matriz de confusión de regresión logística:")
display(pd.crosstab(Y_test_ML, prediccion_log_opt, colnames = ["Predicciones"], margins = True))
print ("Matriz de confusión de árbol de clasificacióin:")
display(pd.crosstab(Y_test_ML, prediccion_arb_opt, colnames = ["Predicciones"], margins = True))
print ("Matriz de confusión de red neuronal:")
display(pd.crosstab(Y_test_ML, prediccion_neuro_opt, colnames = ["Predicciones"], margins = True))

print ("Reporte de clasificación de regresión logística:")
print(classification_report(Y_test_ML, prediccion_log_opt))
print ("Reporte de clasificación de árbol de clasificacióin:")
print(classification_report(Y_test_ML, prediccion_arb_opt))
print ("Reporte de clasificación de red neuronal:")
print(classification_report(Y_test_ML, prediccion_neuro_opt))
```

```
'La regresión logística acierta 5579 de 7184 veces con una tasa de acierto media de 77.6586859688196% '

'El árbol de clasificación a acierta 5658 de 7184 veces con una tasa de acierto media de 78.75835189309576% '

'La red neuronal acierta 5588 de 7184 veces con una tasa de acierto media de 77.78396436525613% '
```

Matriz de confusión de regresión logística:

Predicciones	0	1	All
Pobre			
0.0	4944	796	5740
1.0	809	635	1444
All	5753	1431	7184

Matriz de confusión de árbol de clasificacióin:

Predicciones	0	1	All
Pobre			
0.0	5118	622	5740
1.0	904	540	1444
All	6022	1162	7184

Matriz de confusión de red neuronal:

Predicciones	0	1	All
Pobre			
0.0	4954	786	5740
1.0	810	634	1444
All	5764	1420	7184

```
Reporte de clasificación de regresión logística:
              precision    recall  f1-score   support

         0.0       0.86      0.86      0.86      5740
         1.0       0.44      0.44      0.44      1444

    accuracy                           0.78      7184
   macro avg       0.65      0.65      0.65      7184
weighted avg       0.78      0.78      0.78      7184

Reporte de clasificación de árbol de clasificacióin:
              precision    recall  f1-score   support

         0.0       0.85      0.89      0.87      5740
         1.0       0.46      0.37      0.41      1444

    accuracy                           0.79      7184
   macro avg       0.66      0.63      0.64      7184
weighted avg       0.77      0.79      0.78      7184

Reporte de clasificación de red neuronal:
              precision    recall  f1-score   support

         0.0       0.86      0.86      0.86      5740
         1.0       0.45      0.44      0.44      1444

    accuracy                           0.78      7184
   macro avg       0.65      0.65      0.65      7184
weighted avg       0.78      0.78      0.78      7184
```

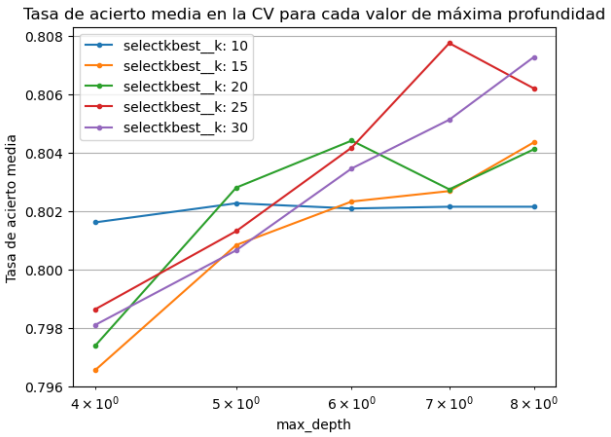
Otros modelos probados

Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

# Pipeline con modelo Random Forest. Se utilizan 10 árboles de clasificación
bosque = make_pipeline(normalizador,
                       reductor,
                       RandomForestClassifier(n_estimators = 10))

param_bosq = {'selectkbest__k': [10, 15, 20, 25, 30],
              'randomforestclassifier__max_depth': [4, 5, 6, 7, 8]}

busqueda_bosq = GridSearchCV(estimator = bosque,
                             param_grid = param_bosq,
                             scoring = 'accuracy',
                             cv = KFold(n_splits=10),
                             n_jobs=-1)

with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=ConvergenceWarning)
    busqueda_bosq.fit(X_train_ML, Y_train_ML)
```

```python
# Función de antes para la tasa de acierto del CV
array_bosq = get_cv_results_for_best_params(busqueda_bosq)
media_bosq = np.asarray(array_bosq).mean()

display('Los mejores parámetros para el Random Forest dado el dataset son:', busqueda_bosq.best_params_)
display('Tasa de acierto en cada iteración de la CV con los mejores parámetros:', array_bosq)
display('Tasa de acierto media del de iteraciones de CV con los mejores parámetros:', media_bosq)

# Obtener el mejor modelo entrenado
bosque_entrenado = busqueda_bosq.best_estimator_[2]

for num_vars in param_bosq['selectkbest__k']:
    mask = busqueda_bosq.cv_results_['param_selectkbest__k'] == num_vars
    C_values = busqueda_bosq.cv_results_['param_randomforestclassifier__max_depth'][mask]
    scores = busqueda_bosq.cv_results_['mean_test_score'][mask]
    plt.semilogx(C_values, scores, marker = ".",label = 'selectkbest__k: ' + str(num_vars))

plt.title('Tasa de acierto media en la CV para cada valor de máxima profundidad')
plt.ylabel('Tasa de acierto media')
plt.xlabel('max_depth')
plt.grid(True)
plt.legend(loc="best")
plt.show()
```

```
'Los mejores parámetros para el Random Forest dado el dataset son:'

{'randomforestclassifier__max_depth': 7, 'selectkbest__k': 25}

'Tasa de acierto en cada iteración de la CV con los mejores parámetros:'

[0.7875894988066826,
 0.8215990453460621,
 0.7941527446300716,
 0.8054892601431981,
 0.8102625298329356,
 0.8013126491646778,
 0.8162291169451074,
 0.8210023866348448,
 0.8138424821002387,
 0.8060859188544153]

'Tasa de acierto media del de iteraciones de CV con los mejores parámetros:'

0.8077565632458235
```




```python
### Aunque utilizar Random Forest sacrifica la explicabilidad, aún es posible observar qué variables de las seleccionadas son
selected_bosq_features = busqueda_bosq.best_estimator_[1].get_support()
nombre_variables = []
for feature_name, selected in zip(X_train_ML.columns, selected_bosq_features):
    if selected:
        nombre_variables.append(feature_name)

importancia_predictores = pd.DataFrame(
                            {'Variable predictora': nombre_variables,
                             'Importancia': bosque_entrenado.feature_importances_}
                            )
display(importancia_predictores.sort_values('Importancia', ascending=False))

print("Características desechadas:")
for feature_name, selected in zip(X_train_ML.columns, selected_bosq_features):
    if not selected:
        print(feature_name)
```

	Variable predictora	Importancia
24	Sit_Laboral_Parado	0.226135
20	Niv_Estudios_Bajo	0.133152
19	Niv_Estudios_Alto	0.098484
23	Sit_Laboral_Otros inactivos	0.074304
0	Familia_Numerosa	0.069656
22	Sit_Laboral_Ocupado	0.050625
17	Estado_Civil_Divorciado/Separado	0.047751
16	Estado_Civil_Casado	0.045882
2	Urbanizacion	0.037452
21	Sit_Laboral_Jubilado	0.036007
1	Hacinamiento	0.031110
18	Estado_Civil_Soltero	0.022645
4	Sexo	0.020403
6	Region_Andalucia	0.015460
3	Edad	0.015434
12	Region_Extremadura	0.013873
10	Region_Catalunya	0.013009
5	PH020	0.009808
13	Region_Madrid	0.008633
9	Region_CastillaLaMancha	0.007901
8	Region_Canarias	0.007391
14	Region_Murcia	0.004624
7	Region_Aragon	0.004382
11	Region_CeutaYMelilla	0.003722
15	Region_Navarra	0.002157

```
Características desechadas:
Viv_PagadaOCedida
Ninos_Dep
En_Formacion
Region_Asturias
Region_Baleares
Region_Cantabria
Region_CastillaYLeon
Region_Galicia
Region_Pais Vasco
Region_Rioja
Region_Valencia
Estado_Civil_Viudo
Niv_Estudios_Medio
```

Máquinas de Vector Soporte (SVM)

Importar clasificador (Support Vector Classifier) de la rama de métodos SVM. El kernel default de este modelo es RBF.

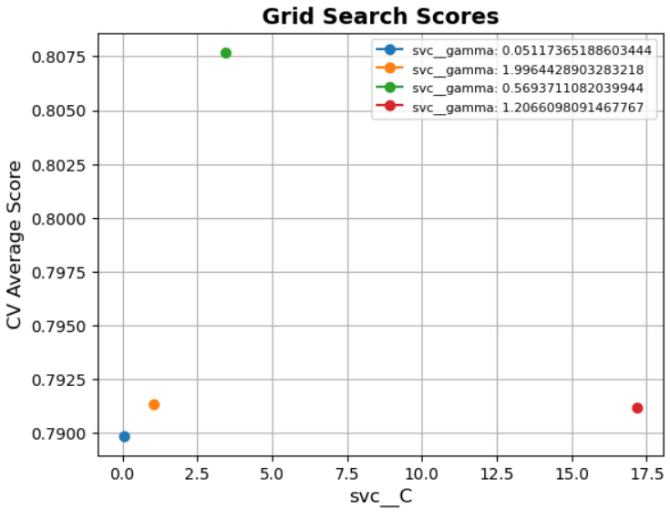
Los parámetros a estudiar son el regularizador C, y el coeficiente de kernel (RBF) gamma.

```
from sklearn.svm import SVC
from scipy.stats import loguniform

# Pipeline con modelo SVM
vector = make_pipeline(normalizador,
                       reductor,
                       SVC(probability = True))

param_vect = {'selectkbest__k': [10, 15, 20, 25, 30],
              'svc__C':loguniform(2**-5,2**5),
              'svc__gamma':loguniform(2**-5,2**5)}

busqueda_vect = RandomizedSearchCV(estimator = vector,
                        param_distributions = param_vect,
                        n_iter = 15,
                        scoring = 'accuracy',
                        cv = KFold(n_splits=5),
                        n_jobs=-1)

with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=ConvergenceWarning)
    busqueda_vect.fit(X_train_ML, Y_train_ML)
```

```
# Función de antes para la tasa de acierto del CV
array_vect = get_cv_results_for_best_params(busqueda_vect)
media_vect = np.asarray(array_vect).mean()

display('Los mejores parámetros para la SVM dado el dataset son:', busqueda_vect.best_params_)
display('Tasa de acierto en cada iteración de la CV con los mejores parámetros:', array_vect)
display('Tasa de acierto media del de iteraciones de CV con los mejores parámetros:', media_vect)

# Obtener el mejor modelo entrenado
svm_entrenado = busqueda_bosq.best_estimator_[2]
```

```
'Los mejores parámetros para la SVM dado el dataset son:'

{'selectkbest__k': 20,
 'svc__C': 1.0285300965348227,
 'svc__gamma': 0.5693711082039944}

'Tasa de acierto en cada iteración de la CV con los mejores parámetros:'

[0.8084725536992841,
 0.7950477326968973,
 0.8072792362768496,
 0.8201073985680191,
 0.8075775656324582]

'Tasa de acierto media del de iteraciones de CV con los mejores parámetros:'

0.8076968973747016
```

```
mask = busqueda_vect.cv_results_['param_selectkbest__k'] == busqueda_vect.best_params_['selectkbest__k']
vect_results_filtrado = {k: [v[i] for i in range(len(v)) if mask[i]] for k, v in busqueda_vect.cv_results_.items()}

# Obtener los valores de puntuación promedio para cada combinación de parámetros
scores_mean = vect_results_filtrado['mean_test_score']

# Crear un diccionario para almacenar los valores de puntuación promedio por combinación de parámetros
scores_dict = {}
for param1, param2, score in zip(vect_results_filtrado['param_svc__C'], vect_results_filtrado['param_svc__gamma'], scores_mea
    scores_dict[(param1, param2)] = score

# Plot Grid search scores
_, ax = plt.subplots(1,1)

# Parametros de la gráfica
x_values = vect_results_filtrado['param_svc__C']
x_values.sort()
y_values = list(scores_dict.values())

# Graficar los valores
for i, linea in enumerate( vect_results_filtrado['param_svc__gamma']):
    ax.plot(x_values[i], y_values[i], marker='o', linestyle='-',
            label='svc__gamma: ' + str(linea))

# Configurar los ejes y leyendas
ax.set_title("Grid Search Scores", fontsize=14, fontweight='bold')
ax.set_xlabel('svc__C', fontsize=12)
ax.set_ylabel('CV Average Score', fontsize=12)
ax.legend(loc="best", fontsize=8)
ax.grid('on')
```



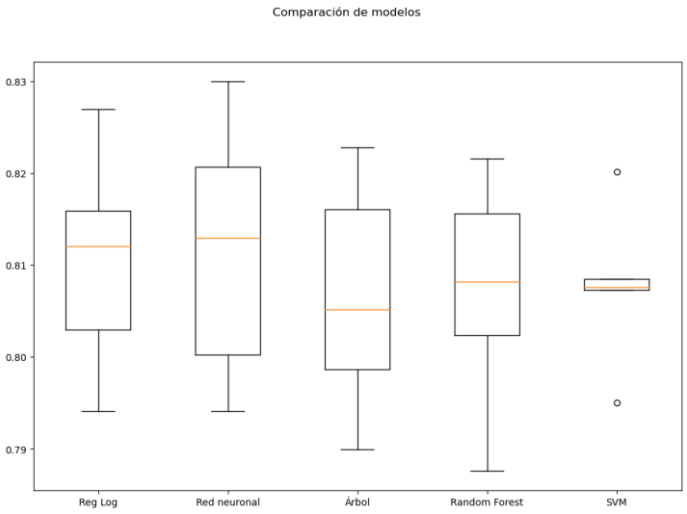
```
selected_vect_features = busqueda_vect.best_estimator_[1].get_support()
print("Características seleccionadas:")
for feature_name, selected in zip(X_train_ML.columns, selected_vect_features):
    if not selected:
        print(feature_name)
```

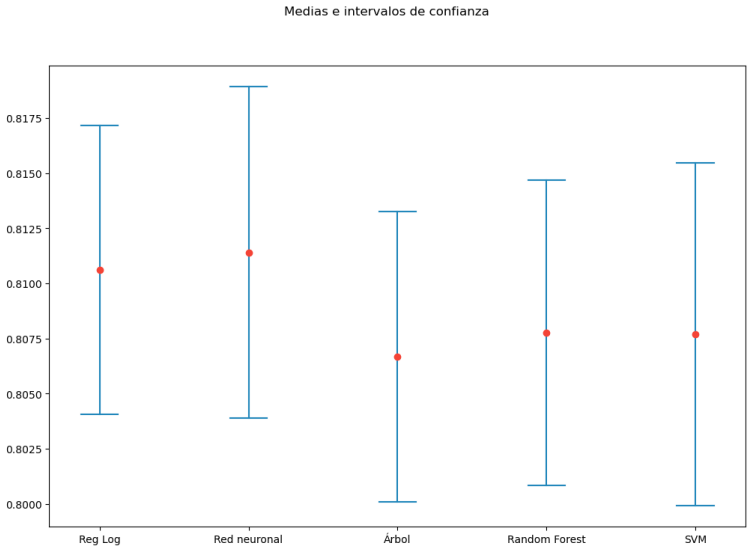
```
Características seleccionadas:
Familia_Numerosa
Viv_PagadaOCedida
Ninos_Dep
En_Formacion
Region_Aragon
Region_Asturias
Region_Baleares
Region_Cantabria
Region_CastillaYLeon
Region_CeutaYMelilla
Region_Galicia
Region_Murcia
Region_Navarra
Region_Pais Vasco
Region_Rioja
Region_Valencia
Estado_Civil_Viudo
Niv_Estudios_Medio
```

Comparativa

Intervalos de confianza de los resultados de las validaciones cruzadas

```
resultados = [array_log, array_neuro, array_arb, array_bosq, array_vect]
modelos = ['Reg Log', 'Red neuronal', 'Árbol', 'Random Forest', 'SVM']

# diagrama de caja y bigotes
fig = plt.figure(figsize=(12,8))
fig.suptitle('Comparación de modelos')
ax = fig.add_subplot(111)   #para definir el eje inferior
plt.boxplot(resultados)
ax.set_xticklabels(modelos)
plt.show()
#aparentemente el modelo de árbol de decisión tiene una mejor mediana, además de un máximo de éxito superior.

fig = plt.figure(figsize=(12,8))
fig.suptitle('Medias e intervalos de confianza')
plt.xticks([1, 2, 3, 4, 5], modelos)
plot_confidence_interval(1, resultados[0], z=1.96) #valor crítico 1.95 equivale al 95% de confianza
plot_confidence_interval(2, resultados[1], z=1.96)
plot_confidence_interval(3, resultados[2], z=1.96)
plot_confidence_interval(4, resultados[3], z=1.96)
plot_confidence_interval(5, resultados[4], z=1.96)
plt.show()
#de aquí parece que el modelo random forest tiene una media ligeramente superior, al igual que su intervalo de confianza
```





Predicción mediante datos test

```
### Predicciones
prediccion_bosq  = busqueda_bosq.predict(X_test_ML)
prediccion_vect  = busqueda_vect.predict(X_test_ML)
### Probabilidades
prediccion_bosq_prob  = busqueda_bosq.predict_proba(X_test_ML)[:,1]
prediccion_vect_prob  = busqueda_vect.predict_proba(X_test_ML)[:,1]
```

```
display(f'El Random Forest acierta {sum(prediccion_bosq == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_bosq)*100}% ')

display(f'La SVM acierta {sum(prediccion_vect == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_vect)*100}% ')
```

```
'El Random Forest acierta 5862 de 7184 veces con una tasa de acierto media de 81.59799554565701% '

'La SVM acierta 5846 de 7184 veces con una tasa de acierto media de 81.37527839643653% '
```

```
print ("Matriz de confusión de Random Forest:")
display(pd.crosstab(Y_test_ML, prediccion_bosq, colnames = ["Predicciones"], margins = True))
print ("Matriz de confusión de SVM:")
display(pd.crosstab(Y_test_ML, prediccion_vect, colnames = ["Predicciones"], margins = True))
```

```
Matriz de confusión de Random Forest:
```

Predicciones Pobre	0.0	1.0	All
0.0	5643	97	5740
1.0	1225	219	1444
All	6868	316	7184

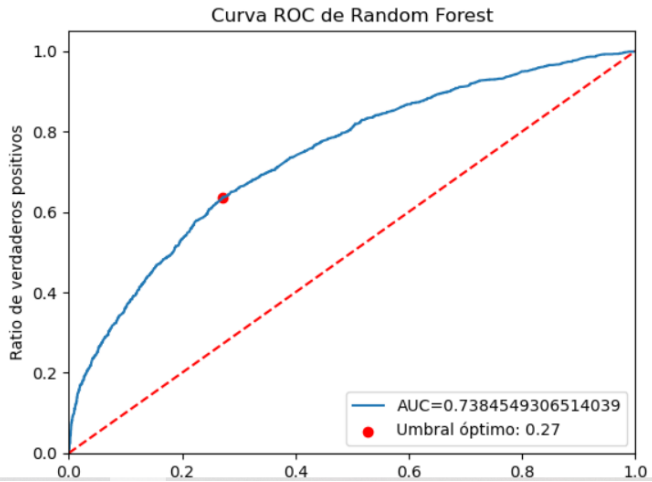
```
Matriz de confusión de SVM:
```

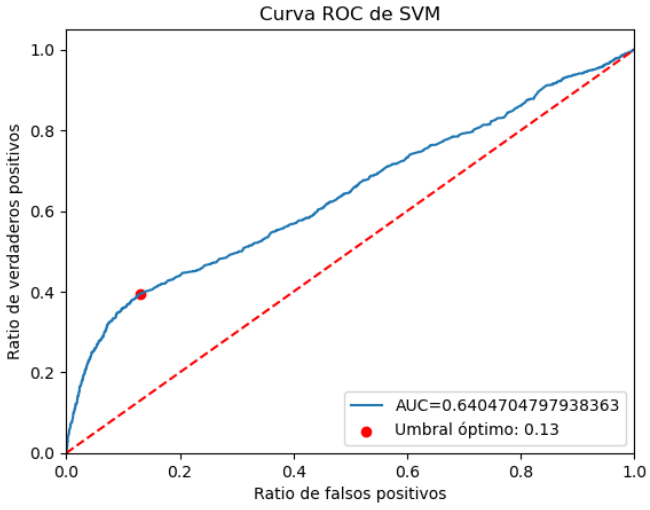
Predicciones Pobre	0.0	1.0	All
0.0	5518	222	5740
1.0	1116	328	1444
All	6634	550	7184

```
print ("Reporte de clasificación de Random Forest:")
print(classification_report(Y_test_ML, prediccion_bosq))
print ("Reporte de clasificación de SVM:")
print(classification_report(Y_test_ML, prediccion_vect))
```

```
Reporte de clasificación de Random Forest:
              precision    recall  f1-score   support

         0.0       0.82      0.98      0.90      5740
         1.0       0.69      0.15      0.25      1444

    accuracy                           0.82      7184
   macro avg       0.76      0.57      0.57      7184
weighted avg       0.80      0.82      0.77      7184

Reporte de clasificación de SVM:
              precision    recall  f1-score   support

         0.0       0.83      0.96      0.89      5740
         1.0       0.60      0.23      0.33      1444

    accuracy                           0.81      7184
   macro avg       0.71      0.59      0.61      7184
weighted avg       0.78      0.81      0.78      7184
```

```
#true positive rate es la sensitividad, false positive rate es specificity
fpr_bosq, tpr_bosq, _   = roc_curve(Y_test_ML, prediccion_bosq_prob)
fpr_vect, tpr_vect, _   = roc_curve(Y_test_ML, prediccion_vect_prob)

#AUC es área bajo la curva ROC, cuanto más alta mejor, de 0 a 1.
AUC_bosq  = roc_auc_score(Y_test_ML, prediccion_bosq_prob)
AUC_vect  = roc_auc_score(Y_test_ML, prediccion_vect_prob)

plot_roc(fpr_bosq, tpr_bosq, AUC_bosq, "Curva ROC de Random Forest")
plot_roc(fpr_vect, tpr_vect, AUC_vect, "Curva ROC de SVM")
```





```python
# Umbrales óptimos de cada modelo
prediccion_bosq_opt   = np.where(prediccion_bosq_prob >= 0.27, 1, 0)
prediccion_vect_opt   = np.where(prediccion_vect_prob >= 0.13, 1, 0)

display(f'El Random Forest acierta {sum(prediccion_bosq_opt == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_bosq_opt)*100}% ')

display(f'La SVM acierta {sum(prediccion_vect_opt == Y_test_ML)} de {len(Y_test_ML)} veces ' \
        + f'con una tasa de acierto media de {accuracy_score(Y_test_ML, prediccion_vect_opt)*100}% ')

print ("Matriz de confusión de Random Forest:")
display(pd.crosstab(Y_test_ML, prediccion_bosq_opt, colnames = ["Predicciones"], margins = True))
print ("Matriz de confusión de SVM:")
display(pd.crosstab(Y_test_ML, prediccion_vect_opt, colnames = ["Predicciones"], margins = True))

print ("Reporte de clasificación de Random Forest:")
print(classification_report(Y_test_ML, prediccion_bosq_opt))
print ("Reporte de clasificación de SVM:")
print(classification_report(Y_test_ML, prediccion_vect_opt))
```

```
'El Random Forest acierta 5446 de 7184 veces con una tasa de acierto media de 75.80734966592428% '
```

```
'La SVM acierta 1494 de 7184 veces con una tasa de acierto media de 20.79621380846325% '
```

Matriz de confusión de Random Forest:

Predicciones	0	1	All
Pobre			
0.0	4741	999	5740
1.0	739	705	1444
All	5480	1704	7184

Matriz de confusión de SVM:

Predicciones	0	1	All
Pobre			
0.0	63	5677	5740
1.0	13	1431	1444
All	76	7108	7184

```
Reporte de clasificación de Random Forest:
              precision    recall  f1-score   support

         0.0       0.87      0.83      0.85      5740
         1.0       0.41      0.49      0.45      1444

    accuracy                           0.76      7184
   macro avg       0.64      0.66      0.65      7184
weighted avg       0.77      0.76      0.77      7184

Reporte de clasificación de SVM:
              precision    recall  f1-score   support

         0.0       0.83      0.01      0.02      5740
         1.0       0.20      0.99      0.33      1444

    accuracy                           0.21      7184
   macro avg       0.52      0.50      0.18      7184
weighted avg       0.70      0.21      0.08      7184
```