Técnicas de aprendizaje automático aplicadas a la estimación del Valor a Riesgo y otras medidas de riesgo financiero

Hugo Naudín López

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Junio de 2025

Summary

Risk management has become a crucial aspect of financial institutions and corporations, especially in the wake of the 2008 financial crisis. In order to avoid the same mistakes that led to the crisis, new regulations were put in place, which required financial institutions to improve their risk management practices and develop more accurate risk prediction models.

The meaning of risk measure was formally defined and the characteristics that these measures should satisfy were established. Value at Risk (VaR) and Expected Shortfall (ES) are two widely used risk measures in this field. The main goal of these is to quantify the potential losses (or returns lower than expected) of an investment, which is essential for making informed decisions.

The VaR is a general risk measure related to the quantiles of the distribution of the returns of an investment. Let G_t be the random variable that represents the returns of an investment at time t, the VaR at a confidence level α is defined as the value that satisfies the following equation:

$$\mathbb{P}\{G_t \leq VaR_{\alpha,t}\} = 1 - \alpha$$

This means VaR represents the pessimistic scenario of the returns of an investment, which is the maximum loss that can be expected with a certain confidence. On the other hand, the ES is the expected return given that the return is below the VaR, which is formally defined as:

$$ES_{\alpha,t} = \mathbb{E}(G_t | G_t \leq VaR_{\alpha,t})$$

The joint use of these two measures provides a more complete picture of the risk associated with an investment.

Because of the importance of these measures, the ways to estimate them have been studied and improved throughout the years, as the precision and accuracy of the estimates are crucial. Linear regression is a basic statistical method that allows us to estimate the relationship between variables based on observed data. However, this technique has limitations, as it assumes linear relationships and can struggle with complex, nonlinear patterns present in real-world financial data.

To overcome these limitations, more sophisticated methods, as machine learning techniques, have been explored throughout this work. These techniques are theoretically more capable of capturing more complex relationships in the data, which can lead to more accurate estimations.

Many ML techniques can be used, but, in this work, the focus will be on two of them: Boosting and Neural Networks. Boosting is an ensemble learning technique that combines the predictions of several weak learners to create a strong learner. It is particularly effective for regression tasks, as it can capture complex relationships in the data, which will be helpful for the issue at hand. On the other hand, Neural Networks are a powerful class of models that can learn complex patterns in data. They are particularly useful for high-dimensional data and can capture nonlinear relationships, which can be beneficial for estimating risk measures.

This work provides a theoretical approach to the concepts of VaR and ES, as well as the basis of estimation methods just mentioned. Finally, the performance of these methods will be compared in terms of their accuracy and reliability in estimating the VaR and ES of a set of financial assets.

In the first chapter, the concepts of risk and risk measures are introduced, along with their characteristics and the reason why they are important. After that, the VaR and ES are defined, along with their advantages and disadvantages, also giving a general perspective on its use cases.

In the second chapter, the focus shifts to the models used to estimate these measures. Firstly, statistical basics are introduced, as the relationship between quantiles and expectiles and risk measures is the basis of the estimation methods. Then, we introduce the basic regression techniques, giving a theoretical overview. After that, the main theme centers on the machine learning techniques used to estimate the VaR and ES.

Finally, the third chapter presents a practical application of the methods presented. First, the aim is to determine the best method among all and then, a use case is presented, where the VaR and ES of a real state asset are estimated, which could help an investor decide whether the investment is profitable or not.

This work includes appendices where Regression Trees techniques, which are used in the Boosting method, the optimization techniques used to improve the performance of the estimation methods and the code used to implement the methods and the use case are explained.

Índice general

Capítulo 1

Introducción: el riesgo en las inversiones

A la hora de invertir, el interés de las corporaciones financieras es maximizar el beneficio y minimizar las pérdidas. Para ello, las compañías utilizan herramientas entre las que se encuentra la gestión del riesgo. El riesgo es la incertidumbre sobre la evolución de un activo, e indica la posibilidad de que una inversión ofrezca un rendimiento distinto del esperado, por lo que la gestión y el control de este puede ayudar a minimizar las pérdidas y optimizar las ganancias.

Durante la década de los 80, la Comisión de Bolsa y Valores de los Estados Unidos (SEC) comenzó a divulgar cómo las empresas estaban expuestas a la volatilidad de los mercados financieros, introduciendo y dando gran importancia al concepto de riesgo en el sistema, debido a los acontecimientos vividos en aquel entonces (Lunes Negro en 1987 y la Crisis de Ahorros y Préstamos). Más adelante, a partir de la crisis financiera del año 2008 y el acuerdo de Basilea III [**?**], las políticas del riesgo que podían correr los bancos a la hora de conceder créditos y realizar inversiones se endurecieron, lo que hizo que los modelos de predicción y gestión de riesgo se optimizaran y mejoraran de manera considerable.

Por otra parte, el uso de la inteligencia artificial (IA) y los algoritmos basados en el aprendizaje automático se han incrementado en los últimos años, llegando a introducirse también en el campo financiero y en la implementación de modelos alternativos a la hora de gestionar el riesgo.

1.1. Medidas de riesgo y sus características

Las medidas de riesgo son funciones que asignan a una posición financiera (representada por una variable aleatoria X que representa las ganancias) un número real que cuantifica su nivel de riesgo. Existen varias clases de medidas de riesgo según las características que cumple. Sean X e Y dos variables aleatorias que representan las ganancias de dos posiciones financieras y sea λ una medida de riesgo. De acuerdo a [**?**], las principales características son las siguientes:

1. **Monotonía** Si $X \leq Y$ casi seguro, entonces $\lambda(X) \geq \lambda(Y)$.

2. **Invarianza ante el efectivo** Sea $c \in \mathbb{R}$ una cantidad de efectivo, entonces se cumple que $\lambda(X + c) = \lambda(X) - c$.

3. **Convexidad** Sea $w \in [0,1]$ una proporción de inversión, entonces $\lambda(wX + (1-w)Y) \leq w\lambda(X) + (1-w)\lambda(Y)$.

4. **Cuasiconvexidad** Sea $w \in [0,1]$ una proporción de inversión, entonces $\lambda(wX + (1-w)Y) \leq \text{máx}\{\lambda(X), \lambda(Y)\}$.

5. **Homogeneidad** Sea $a > 0$ un escalar, entonces $\lambda(aX) = a\lambda(X)$.

6. **Subaditividad** $\lambda(X+Y) \leq \lambda(X)+\lambda(Y)$.

Definición 1. *Una medida de riesgo λ se dice general si cumple que es monótona e invariante ante el efectivo.*

Definición 2. *Una medida de riesgo λ se dice convexa si cumple que es general y cumple la característica de convexidad.*

Definición 3. *Una medida de riesgo λ se dice coherente si cumple que es general, homogénea, convexa y subaditiva.*

Se van a estudiar dos medidas ampliamente utilizadas en el ámbito financiero. Estas son el Valor a Riesgo (VaR) y el Expected Shortfall (ES). La primera es una medida de riesgo general, ya que no es subaditiva [**?**], mientras que la segunda es una medida de riesgo coherente. Ambas medidas se usan para cuantificar el riesgo en inversiones y su estimación es fundamental para la toma de decisiones financieras.

1.2. Valor a Riesgo (VaR)

Si bien el concepto del VaR existe desde mucho antes, la definición del término y su difusión se debió a la publicación de un manual de referencia de J.P: Morgan en los años 90 acerca de la medición de riesgos [**?**], lo cual hizo que se popularizara rápidamente.

Definición 4. *Sea G_t una variable aleatoria que representa las ganancias de una inversión en un tiempo t. Sea F la función de distribución de G_t y α un nivel de confianza específico. Entonces, el Valor a Riesgo (VaR) de G_t en el tiempo t con un nivel de confianza α se define como el valor que cumple lo siguiente:*

$$\mathbb{P}\{G_t \leq VaR_{\alpha,t}\} = 1-\alpha$$

Paralelamente, se puede definir de las siguientes maneras:

$$VaR_{\alpha,t} = inf\{x \in \mathbb{R} : F(x) \geq 1-\alpha\} = F^{-1}(1-\alpha)$$

1.2.1. Ventajas y desventajas del VaR

La medida VaR tiene una gran utilidad a la hora de limitar el riesgo y controlar las potenciales pérdidas (o retornos inferiores a lo esperado) de una inversión, pero hay otros aspectos en los que esta medida no es completa.

Por una parte, es una medida muy simple e intuitiva. Su significado y lo que representa es fácilmente comprensible, lo que hace que su utilización esté muy extendida. Además, la comparabilidad de dos distribuciones se puede realizar fácilmente mediante esta medida, lo que puede resultar útil para evaluar diferentes carteras o estrategias de inversión en términos de su riesgo potencial. Por ejemplo, al comparar dos activos, el VaR permite identificar cuál de ellos tiene una mayor probabilidad de incurrir en pérdidas superiores a un cierto umbral en un horizonte de tiempo definido. Esto facilita la toma de decisiones por parte de gestores de riesgo e inversores. Finalmente, hay en ocasiones en las que interesa la estabilidad de las medidas de riesgo. A la hora de hallar el VaR de una distribución, cualquier cuantil inferior al $1-\alpha$ puede tener cualquier valor, pero el VaR no variará, ya que es independiente de estos valores, siempre y cuando sean menores.

Sin embargo, que el VaR no varíe aunque los cuantiles menores sí puede ser negativo desde otro punto de vista. Esto significaría que una mínima variación del nivel de confianza significaría una gran

variación del VaR. Por tanto, esta medida estaría ocultando un riesgo mayor del que parece en un principio.

Esta desventaja hace del VaR una medida que, a pesar de su gran utilidad, presenta una notable incompletitud. Por ello, existen otras medidas que complementan esta primera, ofreciendo una extensión a la información del VaR, sobre todo, dando información del cuantil inferior. Entre estas variables se encuentra el Expected Shortfall (ES) o Valor a Riesgo Condicional (CVaR).

1.3. Expected Shortfall (ES) o Valor a Riesgo Condicional (CVaR)

El Expected Shortfall (ES) o Valor a Riesgo Condicional (CVar) son dos términos relacionados con el mismo concepto. Para no sobrecargar la notación, se referirá a este concepto como ES de aquí en adelante. Su definición y explicación fueron presentadas por Carlos Acerbi y Dirk Tasche en 2002 [**?**] [**?**].

Definición 5. *Sea G_t una variable aleatoria que representa las ganancias de una inversión en un tiempo t. Sea α un nivel de confianza específico. Entonces, el ES de G_t en el tiempo t con un nivel de confianza α se define como la esperanza condicional de G_t dado que G_t es menor o igual que el VaR en ese tiempo y nivel de confianza:*

$$ES_{\alpha,t} = \mathbb{E}(G_t | G_t \leq VaR_{\alpha,t})$$

El ES siempre será menor que el VaR, ya que el ES toma en cuenta los beneficios menores a él y, por lo tanto, refleja una evaluación más conservadora del riesgo. Si el VaR se considera como el valor máximo de pérdida en un nivel de confianza α, el ES estima el valor promedio de las pérdidas cuando las pérdidas ya superan ese valor. Por lo tanto, el ES proporciona una medida más completa del riesgo asociado a eventos extremos.

1.3.1. Ventajas y desventajas del ES

La medida ES, al contrario del VaR, es una medida continua respecto a α, es decir, cambios pequeños en el nivel de confianza no suponen cambios grandes en el ES. Si se toma un conjunto de inversiones $(X_i)_{i=1}^n$ y $(w_i)_{i=1}^n$ la proporción de cada inversión en nuestra cartera ($\sum_{i=1}^n w_i = 1$), entonces la medida ES en un tiempo t con una confianza α es convexa [**?**] respecto a la combinación de estas variables:

$$ES_{\alpha,t}(w_1X_1 + \ldots w_nX_n) \leq w_1ES_{\alpha,t}(X_1) + \ldots w_nES_{\alpha,t}(X_n)$$

lo cual es coherente con la idea de la diversificación de activos para la reducción del riesgo, ya que el ES de la cartera nunca excede el promedio ponderado de los ES individuales

Por otro lado, esta medida es muy sensible a los errores de estimación. Normalmente, no hay muchos datos en los cuantiles bajos, por lo que es necesario plantear un modelo para obtener más observaciones. Existen casos en los que estos modelos son erráticos y las estimaciones en las colas pueden desviarse de la realidad.

Por tanto, la utilización y corrección de esta medida depende en gran parte de la corrección y confianza que se tenga en los datos. Sin embargo, la combinación de esta medida con el VaR ofrece una visión muy completa del riesgo en la inversión.

1.4. Interés de las medidas de riesgo

Las medidas de riesgo, como el Valor en Riesgo (VaR) y Expected Shortfall (ES), son herramientas fundamentales en la gestión de riesgos, ya que permiten cuantificar y gestionar la incertidumbre asociada a las pérdidas potenciales en diferentes contextos. Su utilidad varía dependiendo de si se aplican a carteras de valores o al ámbito corporativo.

1.4.1. VaR y ES orientados a carteras de valores

En el ámbito de la gestión de carteras de inversión [**?**], el VaR y el ES son cruciales para entender y controlar la exposición al riesgo.

En el caso del VaR, estima la pérdida máxima de una cartera de valores en un horizonte temporal específico y con un nivel de confianza dado. Permite a los gestores de carteras evaluar riesgos acumulativos, identificando el impacto potencial de eventos de mercado extremos sobre la cartera, tomar decisiones informadas, comparando diferentes estrategias de inversión en función de su perfil de riesgo y cumplir requisitos regulatorios (normativas de Basilea).

El ES complementa al VaR al estimar la pérdida promedio en los casos en que esta exceda el VaR. Es especialmente útil porque ofrece una visión más completa y optimiza carteras.

Las dos medidas juntas son esenciales para medir y gestionar el riesgo de mercado, mejorar la asignación de activos y proteger a los inversores frente a pérdidas inesperadas.

1.4.2. VaR y ES orientados a inversiones inmobiliarias

La utilización de estas medidas de riesgo en las inversiones inmobiliarias se enfoca en la predicción del precio de los alquileres de los inmuebles. Dado el precio de una vivienda, el VaR y el ES permiten estimar escenarios negativos de precios a los que puede ser alquilado, permitiendo estimar si la inversión es suficientemente rentable.

Al mismo tiempo, el VaR y el ES se pueden utilizar para calcular una cota inferior del valor intrínseco de una vivienda dadas unas características específicas, como la ubicación, el tamaño y las condiciones del mercado. Esto permite a los inversores evaluar si el precio de compra de una propiedad es razonable en relación con su valor potencial de alquiler y su riesgo asociado.

1.4.3. VaR y ES orientados a corporaciones

En el contexto corporativo [**?**], estas métricas ayudan a evaluar el riesgo financiero y operativo que enfrentan las empresas en sus actividades.

El VaR se utiliza para medir riesgos asociados a las fluctuaciones en tipos de cambio, tasas de interés o precios de materias primas y a la toma de decisiones estratégicas.

Por su parte, el ES proporciona una perspectiva más amplia, considerando las pérdidas extremas que pueden no ser capturadas por el VaR. Ofrece un análisis más detallado de los escenarios de mayor impacto financiero y permite a las empresas diseñar planes de mitigación y estrategias para afrontar eventos inesperados.

En el ámbito corporativo, estas medidas de riesgo facilitan la identificación de vulnerabilidades clave, la planificación financiera y la protección de los recursos de la empresa frente a incertidumbres del

entorno operativo y del mercado.

1.4.4. VaR y ES orientados a predicciones de impago en créditos

Estas variables económicas también se emplean en la predicción de impago de créditos [**?**], lo que ayuda a los bancos y entidades de préstamos en sus decisiones. El VaR se utiliza para estimar la cantidad que con un nivel de confianza dado el cliente impagará, considerando factores como el historial crediticio, la situación financiera, las condiciones del mercado y determinadas características del cliente.

Por otra parte, el ES, al igual que en el caso de las carteras de valores, se utiliza para evaluar la pérdida promedio en los casos en que el cliente no pague, lo que da más información a la entidad acerca de la rentabilidad de dar el crédito.

Estas medidas permiten a las entidades evaluar el riesgo asociado a la concesión de créditos y ajustar sus políticas de préstamo en consecuencia.

Capítulo 2

Modelos de riesgo

Con el objetivo de aproximar las medidas de riesgo anteriormente descritas, a lo largo de la historia se han ido desarrollando diferentes modelos que emplean técnicas distintas para ayudar a predecir el riesgo dadas distintas situaciones.

2.1. Modelos tradicionales

Los métodos tradicionales están fundamentados en la econometría y en métodos estadísticos. Sin embargo, algunos de estos enfoques pueden presentar limitaciones cuando se enfrentan a condiciones de mercado altamente volátiles o a eventos inesperados o se basan en suposiciones no siempre ciertas (distribuciones normales, independencia de errores, etc.). Sin embargo, existen otras técnicas que tienen una mayor robustez.

2.1.1. Regresión cuantil

Para introducir los conceptos que se van a presentar, se toma X una variable aleatoria continua con función de distribución $F : \mathbb{R} \to [0,1]$ y función de densidad f_X. Se fija un nivel $\alpha \in (0,1)$.

Definición 6. *El cuantil de orden α de la variable aleatoria X, denotado $c_\alpha(X)$, se define como:*

$$c_\alpha(X) := F^{-1}(\alpha) = \text{ínf}\{x \in \mathbb{R} : F(x) \geq \alpha\}$$

La regresión cuantil [**?**] es un método estadístico que permite estimar la relación entre variables explicativas con los cuantiles de la distribución de la variable dependiente. A diferencia de la regresión lineal usual, que se centra en la media condicional, la regresión cuantil proporciona una visión más completa de la relación entre variables al considerar los cuantiles de la distribución, de donde se puede obtener una información mayor.

Para introducir la regresión cuantil, conviene partir de una formulación general de los modelos de regresión. Se considera un conjunto de variables explicativas $\mathbf{X}_i = \{X_{i1}, X_{i2}, \ldots, X_{ik}\}$ y una variable dependiente Y_i. La regresión lineal modela la esperanza condicionada de la variable dependiente en función de las variables explicativas mediante la utilización de los parámetros $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1 \ldots, \hat{\beta}_k)$ de la siguiente manera:

$$\hat{Y}_i = \mathbb{E}(Y_i|\mathbf{X_i}) = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \hat{\beta}_2 X_{i2} + \cdots + \hat{\beta}_k X_{ik}$$

Los coeficientes $\hat{\beta}_j$ se denominan coeficientes de regresión y cuantifican la influencia de cada variable explicativa sobre la respuesta. El objetivo de la regresión es estimar los parámetros $\hat{\beta}_j$ que mejor aproximen la relación entre Y_i y $\mathbf{X_i}$.

La regresión cuantil intenta modelar el cuantil α de la variable dependiente Y_i dado un conjunto de variables explicativas $\mathbf{X_i}$, en vez de centrarse en la media. La forma general del modelo es:

$$c_\alpha(Y_i|\mathbf{X}_i) = \dot{\beta}_0(\alpha) + \dot{\beta}_1(\alpha)X_{i1} + \dot{\beta}_2(\alpha)X_{i2} + \cdots + \dot{\beta}_k(\alpha)X_{ik}$$

donde $c_\alpha(Y_i|\mathbf{X}_i)$ es el cuantil α de la variable dependiente Y_i dado el vector de variables explicativas $\mathbf{X}_i$.

Para hallar los parámetros $\dot{\beta}$, se utiliza una función de pérdida, la cual se debe intentar minimizar. En el caso de la regresión cuantil, la función de pérdida empleada es la función de pérdida cuantílica o pinball.

Definición 7. *La función de pérdida cuantílica de orden $\alpha \in (0,1)$, se define como:*

$$\rho_\alpha(r) := r(\alpha - \mathbb{I}\{r < 0\}), \quad r \in \mathbb{R}. \tag{2.1}$$

Proposición 1. *El valor de $\theta \in \mathbb{R}$ que minimiza el valor esperado de la función de pérdida cuantílica es el cuantil de orden α de X, es decir:*

$$\arg\min_{\theta \in \mathbb{R}} \mathbb{E}_X[\rho_\alpha(X-\theta)] = c_\alpha(X)$$

Demostración. Se considera la función objetivo:

$$g(\theta) := \mathbb{E}_X[\rho_\alpha(X-\theta)] = \int_{-\infty}^{\infty} (x-\theta)(\alpha - \mathbb{I}\{x < \theta\}) f_X(x)\,dx$$

Bajo la hipótesis de que f_X es continua, esta función es diferenciable en θ, y se puede calcular su derivada teniendo en cuenta que los extremos de integración dependen de θ:

$$g'(\theta) = \frac{d}{d\theta}\left[\int_{-\infty}^{\theta} (x-\theta)(\alpha-1) f_X(x)\,dx + \int_{\theta}^{\infty} (x-\theta)\alpha f_X(x)\,dx\right]$$

Aplicando la regla de Leibniz:

$$\begin{aligned}
g'(\theta) &= (\alpha-1)\left[(x-\theta)f_X(x)\big|_{x=\theta} + \int_{-\infty}^{\theta} \frac{d}{d\theta}[(x-\theta)] f_X(x)\,dx\right] \\
&\quad + \alpha\left[(x-\theta)f_X(x)\big|_{x=\theta} + \int_{\theta}^{\infty} \frac{d}{d\theta}[(x-\theta)] f_X(x)\,dx\right] \\
&= (\alpha-1)(0 \cdot f_X(\theta)) - (\alpha-1)\int_{-\infty}^{\theta} f_X(x)\,dx \\
&\quad + \alpha(0 \cdot f_X(\theta)) - \alpha\int_{\theta}^{\infty} f_X(x)\,dx \\
&= -(\alpha-1)F(\theta) - \alpha(1-F(\theta)) \\
&= F(\theta) - \alpha
\end{aligned}$$

Como se quiere hallar el mínimo, se iguala la derivada a 0:

$$g'(\theta) = 0 \iff F(\theta) = \alpha \iff \theta = F^{-1}(\alpha).$$

Por tanto, el valor que minimiza la pérdida esperada es el cuantil de orden α de X. □

La estimación del cuántil no es trivial y se utilizan varios algoritmos para hallar los coeficientes de regresión $\dot{\beta}$ [**?**, p. 11]. Estas técnicas serán también relevantes a la hora de estudiar la regresión expectil, cuya función de pérdida es asimétrica y cuadrática, la cual se presentará más adelante.

Una vez se estiman los coeficientes, se podrá hacer una predicción de los cuantiles de la variable dependiente Y_i dado un conjunto de variables explicativas $\mathbf{X_i}$.

La utilidad de la regresión cuantil reside, entre otras, en la facilidad para el cálculo del VaR, ya que permite estimar beneficios extremadamente bajos (incluso negativos) y relacionarlos con otros factores como las tasas de interés, los precios de las materias primas o las condiciones del mercado, todo ello sin depender de la media ni de supuestos de normalidad.

2.1.2. Regresión expectil

Además de los cuantiles, existe una medida estadística de gran utilidad para la estimación del riesgo llamada expectil [**?**]. Para introducir los conceptos que se van a presentar, se toma X una variable aleatoria continua con función de distribución $F : \mathbb{R} \to [0,1]$ y función de densidad f_X. Se fija un nivel $\tau \in (0,1)$.

Definición 8. *El expectil de orden τ (e_τ) de la variable aleatoria X es aquel valor que minimiza la función $\phi(X - e_\tau)$, donde:*

$$\phi(r) = \begin{cases} \tau \cdot r^2 & si\ r > 0 \\ (1-\tau) r^2 & si\ r \leq 0 \end{cases}$$

Esta función se denomina función de pérdida expectílica de orden τ.

Los expectiles se pueden utilizar como una aproximación de los cuantiles, aunque no son exactamente lo mismo. Mientras que los cuantiles dividen la masa de probabilidad en proporciones fijas, los expectiles ponderan las pérdidas de manera diferente, dando más peso a pérdidas extremas. Por tanto, la aproximación no es la manera correcta de usarlos.

La regresión expectil es un método estadístico útil para estimar variables relacionadas con la cola de una distribución como, por ejemplo, el Expected Shortfall (ES). Mientras que la regresión cuantil minimiza una función de pérdida absoluta asimétrica, la regresión expectil minimiza una función de pérdida cuadrática asimétrica.

Definición 9. *La función de pérdida expectil de orden $\tau \in (0,1)$ se denota:*

$$\phi_\tau(r) = \begin{cases} \tau \cdot r^2 & si\ r > 0 \\ (1-\tau) r^2 & si\ r \leq 0 \end{cases} \tag{2.2}$$

Dadas unas variables explicativas $\mathbf{X_i} = \{X_{i1}, X_{i2}, \ldots, X_{ik}\}$ y una variable dependiente Y_i, la regresión expectil se define como:

$$e_\tau(Y_i|\mathbf{X_i}) = \bar{\beta}_0(\tau) + \bar{\beta}_1(\tau)X_{i1} + \bar{\beta}_2(\tau)X_{i2} + \cdots + \bar{\beta}_k(\tau)X_{ik} = \bar{\beta}(\tau)\mathbf{X_i}$$

y dado un conjunto de datos $(Y_i, \mathbf{X_i})_{i=1}^n$, la estimación de los coeficientes de regresión expectil $\bar{\beta}(\tau)$ se obtiene minimizando la función de pérdida expectílica:

$$\bar{\beta}(\tau) = argmin_\beta \sum_{i=1}^{n} \phi_\tau(Y_i - e_\tau(Y_i|\mathbf{X}_i))$$

Proposición 2. *Dado ES_α el Expected Shortfall de una variable aleatoria X de orden α, se puede expresar en funcion de un expectil de orden τ que cumple que su valor es igual al VaR_α ($e_\tau = VaR_\alpha \iff P(X \leq e_\tau) = 1 - \alpha$).*

Demostración. Se puede definir la función de pérdida expectílica como:

$$\begin{aligned}\phi_\tau(X-e_\tau) &= \tau(X-e_\tau)^2 I(X>e_\tau)+(1-\tau)(X-e_\tau)^2 I(X\le e_\tau)\\ &= (\tau I(X>e_\tau)+(1-\tau)I(X\le e_\tau))(X-e_\tau)^2\end{aligned}$$

Se va a desarrollar y obtener propiedades del expectil. Para ello, se va a tomar un parámetro θ en lugar del expectil y a ver cual es el valor de θ que minimiza la esperanza de la función.

$$g_{\tau,\theta}(X) = (\tau I(X>\theta)+(1-\tau)I(X\le\theta))(X-\theta)^2$$

Se calcula su esperanza:

$$\begin{aligned}E_X(g_{\tau,\theta}(X)) &= E_X((\tau I(X>\theta)+(1-\tau)I(X\le\theta))(X-\theta)^2) =\\ &= \int_{-\infty}^{\infty}(\tau I(x>\theta)+(1-\tau)I(x\le\theta))(x-\theta)^2 f_X(x)dx =\\ &= \int_{-\infty}^{\theta}((1-\tau)(x-\theta)^2 f_X(x)dx+\int_{\theta}^{\infty}(\tau(x-\theta)^2 f_X(x)dx =\\ &= (1-\tau)\int_{-\infty}^{\theta}(x-\theta)^2 f_X(x)dx+\tau\int_{\theta}^{\infty}(x-\theta)^2 f_X(x)dx\end{aligned}$$

Se deriva con respecto a θ y se iguala a 0:

$$\begin{aligned}\frac{d}{d\theta}E_X(g_{\tau,\theta}(X)) &= (1-\tau)\int_{-\infty}^{\theta}2(x-\theta)f_X(x)dx+(1-\tau)(\theta-\theta)^2 f_X(\theta)+\\ &\quad+\tau\int_{\theta}^{\infty}2(x-\theta)f_X(x)dx+\tau(\theta-\theta)^2 f_X(\theta) =\\ &= (1-\tau)\int_{-\infty}^{\theta}2(x-\theta)f_X(x)dx+2\tau\int_{\theta}^{\infty}(x-\theta)f_X(x)dx =\\ &= 2((1-\tau)E_X((X-\theta)\cdot 1_{X\le\theta})+\tau E_X((X-\theta)\cdot 1_{X>\theta}))=0\end{aligned}$$

Por tanto, simplificando, se tiene que:

$$E_X((X-\theta)\cdot 1_{X>\theta}) = -\frac{1-\tau}{\tau}E_X((X-\theta)\cdot 1_{X\le\theta})$$

y sustituyendo θ por e_τ, se tiene que:

$$E_X((X-e_\tau)\cdot 1_{X>e_\tau}) = -\frac{1-\tau}{\tau}E_X((X-e_\tau)\cdot 1_{X\le e_\tau})$$

Se denota $A = E_X((X-e_\tau)\cdot 1_{X>e_\tau})$ y $B = E_X((X-e_\tau)\cdot 1_{X\le e_\tau})$, por lo que:

$$A = -\frac{1-\tau}{\tau}B$$

Sabiendo que $1-\alpha = P(X\le e_\tau)$, por lo que:

$$\begin{aligned}E_X((X-e_\tau)\cdot 1_{X\le e_\tau}) &= \int_{-\infty}^{e_\tau}(x-e_\tau)f_X(x)dx = \int_{-\infty}^{e_\tau}xf_X(x)dx-e_\tau\int_{-\infty}^{e_\tau}f_X(x)dx =\\ &= \frac{1-\alpha}{1-\alpha}\int_{-\infty}^{e_\tau}xf_X(x)dx-e_\tau(1-\alpha) = (1-\alpha)(ES_\alpha-e_\tau)\end{aligned}$$

Por tanto, despejando ES_α:

$$ES_\alpha = \frac{1}{1-\alpha}E_X((X-e_\tau)\cdot 1_{X\le e_\tau})+e_\tau$$

Se desea una expresión más sencilla. Se suma A y B, calculados anteriormente:

$$\begin{aligned} A+B &= E_X((X-e_\tau)\cdot 1_{X>e_\tau}) + E_X((X-e_\tau)\cdot 1_{X\leq e_\tau}) = \\ &= E_X(X-e_\tau) = E_X(X) - e_\tau \end{aligned}$$

Al mismo tiempo,

$$A+B = -\frac{1-\tau}{\tau}B + B = \left(1-\frac{1-\tau}{\tau}\right)B = \frac{2\tau-1}{\tau}B$$

Por tanto, sustituyendo en la expresión anterior:

$$\begin{aligned} ES_\alpha &= \frac{1}{1-\alpha}B + e_\tau = \frac{1}{1-\alpha}E_X((X-e_\tau)\cdot 1_{X\leq e_\tau}) + e_\tau = \\ &= \frac{\tau}{(1-\alpha)\cdot(2\tau-1)}(E_X(X)-e_\tau) + e_\tau \end{aligned}$$

□

Por tanto, se utilizará la estimación de los expectiles para estimar el orden τ para el que el expectil e_τ coincide con el VaR ($P(X \leq e_\tau) = 1-\alpha \implies e_\tau = Var_\alpha$). Una vez se tenga el expectil, se podrá estimar el Expected Shortfall (ES) [**?**].

2.2. Modelos basados en Machine Learning

Conforme al avance de la tecnología, se han ido desarrollando nuevos métodos de estimación del riesgo, algunos basados en el aprendizaje máquina, que se ha desarrollado en los últimos años exponencialmente. Los métodos basados en Machine Learning (ML) son modelos de muchas dimensiones que se combinan con métodos de "regularización" y de reducción del sobreajuste, junto con algoritmos eficientes de búsqueda.

La base de los modelos de ML es parecida a la de los modelos econométricos, ya que están basados en métodos estadísticos. Sin embargo, existen diferencias entre los modelos que los vuelven muy diferentes.

Mientras que los modelos econométricos se centran en la inferencia causal y la interpretación de relaciones económicas, los modelos de ML se enfocan en la predicción y precisión del ajuste. Además, los modelos de ML no se basan en supuestos como la normalidad o la independencia, al contrario que muchos métodos econométricos, lo que los hace más fielmente aplicables a distintas situaciones.

Previo a la utilización de estos métodos, es necesario "entrenarlos", es decir, ajustar los parámetros del modelo a los datos. Para ello, se dividen los datos en dos conjuntos: el conjunto de entrenamiento y el conjunto de prueba. El conjunto de entrenamiento se utiliza para ajustar el modelo, mientras que el conjunto de prueba se utiliza para evaluar su rendimiento. Una vez se entrena el modelo, se podrá aplicar sobre otros datos, obteniendo predicciones sobre el riesgo asociado a los eventos.

Se van a presentar a continuación modelos utilizados para la estimación de cuantiles y expectiles [**?**], que posteriormente se utilizarán en un caso práctico.

2.2.1. Redes Neuronales

Las redes neuronales funcionan como cerebros humanos, en los que se encuentran neuronas conectadas entre si. Están formadas por varios conjuntos de neuronas, llamados capas. Hay tres tipos de capas,

llamadas la capas de entrada, ocultas y de salida. Para entrenar a la red neuronal, se introducen los datos por la capa de entrada. A partir de ahí, la información se transporta por las capas ocultas mediante funciones predeterminadas y pesos que determinan la influencia entre neuronas interconectadas. Este proceso se repite hasta que la información llega a la capa de salida. El proceso de razonamiento interno funciona como una caja negra y no se puede identificar realmente las relaciones entre la entrada y la salida.

Para hacer una explicación completa pero sencilla, se tomará una red neuronal con una sola capa oculta totalmente conectada. Las entradas $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ son transformadas mediante pesos y una función llamada función de activación y transportadas a la siguiente neurona, la cual está en la capa oculta. Acto seguido, se repetirá el proceso entre la capa oculta y la capa de salida pero utilizando Z_j $j = 1, 2, \ldots, M_1$ como entradas.

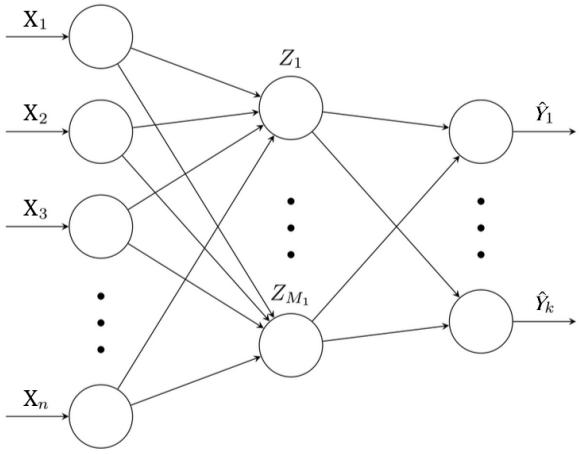

Figura 2.1: Esquema de una red neuronal artificial con una capa oculta.

Para cada neurona de la capa oculta, se transforman todos los valores de las neuronas de la capa de entrada de la siguiente manera:

$$a_j^{(1)} = \sum_{i=1}^{n} w_{j,i}^{(1)} X_i + b_j^{(1)}$$
$$Z_j^{(1)} = \sigma(a_j^{(1)}), j = 1, 2, \ldots M_1$$

con σ la función de activación y $w_{j,i}$ los pesos asociados a cada neurona, además de un peso de la propia neurona receptora, llamada sesgo b_j. En caso de haber más capas ocultas se repetiría este proceso:

$$Z_j^{(m)} = \sigma\left(\sum_{i=1}^{M_{(m-1)}} w_{j,i}^{(m)} Z_i^{(m-1)} + b_j^{(m)}\right)$$

En el caso de la figura **??**, al solamente tener una capa oculta, la salida de la red neuronal será:

$$\hat{Y}_k = \sigma\left(\sum_{i=1}^{M_1} w_{k,i}^{(2)} Z_i^{(1)} + b_k^{(2)}\right)$$

Tras obtener los resultados de la primera evaluación, se actualizan los pesos de la red intentando minimizar la siguiente función:

$$\mathscr{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = \sum_{i=1}^{n} \rho(Y_i - \hat{Y}_i)$$

Siendo $\hat{Y}_i$ el valor esperado. Es decir, se intentan modificar los pesos de manera que la predicción sea lo más parecida al valor que se busca. Se modificará la función ρ en función de cuál sea el objetivo de

cálculo de la red.

En cuanto a las funciones de activación, la más utilizada e implementada en la mayoría de métodos es la función ReLu:

$$\sigma(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases} = \text{máx}(0, x)$$

Una de sus principales ventajas es su simplicidad computacional y su capacidad para evitar el problema del gradiente desvanecido en regiones positivas, lo que mejora significativamente la eficiencia del entrenamiento mediante el método de descenso por gradiente. Además, su derivada es muy simple, lo que permite una propagación del gradiente más estable en comparación con funciones como la sigmoide.

Para ajustar los pesos, se utilizará el algoritmo de retropropagación (backpropagation), que se basa en el cálculo del gradiente de la función de pérdida con respecto a los pesos. Sin embargo, en lugar de utilizar el método del gradiente descendente clásico, cuyo uso es sencillo y fácil de comprender, se empleará el algoritmo *Adam* (Adaptive Moment Estimation) [**?**], que utiliza estimaciones adaptativas de la media y la varianza del gradiente para actualizar los parámetros de manera más eficiente.

Dado el gradiente de la función de pérdida con respecto a los pesos de la capa l, que se denota por $g^{(l)} = \frac{\partial \mathscr{L}}{\partial \theta} = -\sum_{i=1}^{n} \rho' \left(Y_i - \hat{Y}_i\right) \cdot \frac{\partial \hat{Y}_i}{\partial \theta}$, Adam calcula una estimación del primer momento (media) y del segundo momento (varianza) del gradiente mediante las siguientes ecuaciones:

$$\begin{aligned} m^{(l)} &\leftarrow \beta_1 m^{(l)} + (1-\beta_1) g^{(l)} \\ v^{(l)} &\leftarrow \beta_2 v^{(l)} + (1-\beta_2)(g^{(l)})^2 \\ m_b^{(l)} &\leftarrow \beta_1 m_b^{(l)} + (1-\beta_1) g_b^{(l)} \\ v_b^{(l)} &\leftarrow \beta_2 v_b^{(l)} + (1-\beta_2)(g_b^{(l)})^2 \end{aligned}$$

A continuación, se realiza una corrección del sesgo de estas estimaciones, importante en las primeras iteraciones:

$$\begin{aligned} \hat{m}^{(l)} &= \frac{m^{(l)}}{1-(\beta_1)^t} \quad \hat{v}^{(l)} = \frac{v^{(l)}}{1-(\beta_2)^t} \\ \hat{m}_b^{(l)} &= \frac{m_b^{(l)}}{1-(\beta_1)^t} \quad \hat{v}_b^{(l)} = \frac{v_b^{(l)}}{1-(\beta_2)^t} \end{aligned}$$

Finalmente, se actualizan los pesos y el sesgo

$$\begin{aligned} w^{(l)} &\leftarrow w^{(l)} - \zeta \cdot \frac{\hat{m}^{(l)}}{\sqrt{\hat{v}^{(l)}} + \varepsilon} \\ b^{(l)} &\leftarrow b^{(l)} - \zeta \cdot \frac{\hat{m}_b^{(l)}}{\sqrt{\hat{v}_b^{(l)}} + \varepsilon} \end{aligned}$$

donde ζ es la tasa de aprendizaje, β_1 y β_2 son los coeficientes de decaimiento para los momentos (típicamente 0.9 y 0.999), ε es un pequeño valor positivo para evitar divisiones entre 0 y t es la iteración actual.

Se supone, por tanto, que se tienen m datos disponibles $(\mathbf{X_1}, Y_1), (\mathbf{X_2}, Y_2), \ldots, (\mathbf{X_m}, Y_m)$, donde $\mathbf{X_i}$ son las variables explicativas y Y_i las variable dependientes. El algoritmo será:

Algorithm 1: Red neuronal

Inicializar aleatoriamente los pesos $w^{(l)}$, sesgos $b^{(l)}$, y momentos $m^{(l)}, v^{(l)}, m_b^{(l)}, v_b^{(l)}$ para $l = 1, \ldots, L$;
$t \leftarrow 0$;
// Bucle de entrenamiento;
for *epoch = 1* **to** T **do**
  for $i = 1$ **to** m **do**
    $t \leftarrow t + 1$;
    ; // 1. Propagación hacia adelante
    $Z^{(0)} \leftarrow \mathbf{X}_i$;
    for $l = 1$ **to** L **do**
      $a^{(l)} \leftarrow w^{(l)} Z^{(l-1)} + b^{(l)}$;
      $Z^{(l)} \leftarrow \sigma^{(l)}(a^{(l)})$;
    $\hat{\mathbf{Y}}_i \leftarrow Z^{(L)}$;
    ; // 2. Retropropagación (Cálculo de gradientes)
    $\delta^{(L)} \leftarrow \frac{\partial \mathscr{L}}{\partial \hat{y}_i} \cdot \sigma^{(L)'}(a^{(L)})$;
    $g^{(L)} \leftarrow \delta^{(L)} (Z^{(L-1)})^\top; \quad g_b^{(L)} \leftarrow \delta^{(L)}$;
    for $l = L - 1$ **to** 1 **do**
      $\delta^{(l)} \leftarrow ((w^{(l+1)})^\top \delta^{(l+1)}) \circ \sigma^{(l)'}(a^{(l)})$;
      $g^{(l)} \leftarrow \delta^{(l)} (Z^{(l-1)})^\top; \quad g_b^{(l)} \leftarrow \delta^{(l)}$;
    ; // 3. Actualización de parámetros (Optimizador Adam)
    for $l = 1$ **to** L **do**
      ; // Actualizar momentos
      $m^{(l)} \leftarrow \beta_1 m^{(l)} + (1 - \beta_1) g^{(l)}; \quad m_b^{(l)} \leftarrow \beta_1 m_b^{(l)} + (1 - \beta_1) g_b^{(l)}$;
      $v^{(l)} \leftarrow \beta_2 v^{(l)} + (1 - \beta_2)(g^{(l)})^2; \quad v_b^{(l)} \leftarrow \beta_2 v_b^{(l)} + (1 - \beta_2)(g_b^{(l)})^2$;
      ; // Corregir sesgos de los momentos
      $\hat{m}^{(l)} \leftarrow \frac{m^{(l)}}{1 - \beta_1^t}; \quad \hat{m}_b^{(l)} \leftarrow \frac{m_b^{(l)}}{1 - \beta_1^t}$;
      $\hat{v}^{(l)} \leftarrow \frac{v^{(l)}}{1 - \beta_2^t}; \quad \hat{v}_b^{(l)} \leftarrow \frac{v_b^{(l)}}{1 - \beta_2^t}$;
      ; // Actualizar parámetros
      $w^{(l)} \leftarrow w^{(l)} - \zeta \frac{\hat{m}^{(l)}}{\sqrt{\hat{v}^{(l)}} + \varepsilon}$;
      $b^{(l)} \leftarrow b^{(l)} - \zeta \frac{\hat{m}_b^{(l)}}{\sqrt{\hat{v}_b^{(l)}} + \varepsilon}$;

Las redes neuronales tienen una gran capacidad de aprendizaje y alta precisión en tareas complejas. Estas son las características que hacen que su utilización esté muy extendida en modelos predictivos.

2.2.2. Algoritmos de Boosting

Los algoritmos de boosting [**?**] son una técnica de aprendizaje automático que se basa en la existencia de algoritmos débiles que, dados ejemplos de entrenamiento etiquetados, produce unos predictores débiles que, después, se combinan produciendo un predictor fuerte de alta precisión.

El objetivo es generar estos predictores de forma iterativa. En cada iteración, el predictor débil intenta corregir los errores del predictor anterior. Para ello, se modifica la importancia de cada dato dentro de la muestra para que cada clasificador débil se ”especialice” en los datos cuya predicción ha sido errada

por el anterior. Para predecir correctamente, el predictor débil trata de minimizar una función de pérdida, que se definirá dependiendo del objetivo del algoritmo.

Para ajustar el modelo, se debe entrenar, al igual que ocurre con las redes neuronales. Por tanto, para la explicación del modelo, se toman $(\mathbf{X_1}, Y_1), (\mathbf{X_2}, Y_2), \ldots, (\mathbf{X_m}, Y_m)$, siendo $\mathbf{X_i}$ variables explicativas y $\mathbf{Y}_i$ las variables dependientes explicadas. Se define $\mathbf{X} = \{\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m}\}$.

Asociando al predictor débil la función $g : \mathbb{R}^n \to \mathbb{R}$, la función de pérdida que tratará de minimizar será:

$$\mathscr{L}(\mathbf{X}, \mathbf{Y}, g) = \sum_{i=1}^{m} \rho(Y_i - g(\mathbf{X}_i))$$

Para estimar cuantiles se utilizará la función de pérdida cuantil o *pinball loss* (**??**) y para calcular expectiles se usará la función de pérdida expectil (**??**).

Los predictores débiles que se utilizarán serán árboles de regresión, que son estructuras jerárquicas que dividen el espacio de las variables explicativas con el fin de minimizar el error de predicción en cada hoja. Los árboles utilizados en boosting son poco profundos, a menudo con una profundidad limitada, lo que los convierte en predictores débiles pero complementarios entre sí. Una descripción más detallada de los árboles de regresión se pueden encontrar en el **??**.

Para la optimización de los predictores se utilizará el gradiente descendente. En lugar de ajustar directamente parámetros numéricos, se busca en cada iteración una función (el predictor débil) que apunte en la dirección del gradiente negativo de la función de pérdida.

Dado $f_{m-1}(x)$ el predictor fuerte en una iteración dada, se calcula el gradiente negativo de la pérdida con respecto a las predicciones.:

$$\mathbf{u}_{i,m} = - \left. \frac{\partial \mathscr{L}(\mathbf{X}, \mathbf{Y}, f)}{\partial f(\mathbf{X}_i)} \right|_{f=f_{m-1}}$$

Estos valores representan la dirección en la que debe moverse para minimizar la función de pérdida. El predictor débil $g_m(x)$ se ajusta entonces para aproximar estos residuos, y se actualiza el modelo sumando la nueva función escalada por un factor de aprendizaje η:

$$f_m(x) = f_{m-1}(x) + \eta \cdot g_m(x)$$

Cuanto más pequeña sea la tasa de aprendizaje η, más lento será el proceso, pero también más estable y menos propenso al sobreajuste.

El algoritmo de boosting es teóricamente ideal para afrontar el problema ya que $\mathbf{X_i}$ es una variable multidimensional y el problema puede tornarse complejo. La eficiencia y precisión de los algoritmos de boosting los hacen muy útiles en la estimación de cuantiles y expectiles.

Algorithm 2: Boosting para cuantiles y expectiles

Inicializar $f_0(x) \leftarrow 0$ o $f_0(x) \leftarrow \bar{y}$ (la media de los valores y_i);
for $m = 1$ **to** M **do**
  for $i = 1$ **to** n **do**
    $u_{i,m} \leftarrow -\rho'_\tau(y_i - f_{m-1}(x_i))$;
  ; `// Ajustar un predictor débil` $g_m(x)$ `a los pares` $(x_i, u_{i,m})$
  $\{(x_i, u_{i,m})\}_{i=1}^{n} \longrightarrow g_m(x)$;
  $f_m(x) \leftarrow f_{m-1}(x) + \eta \cdot g_m(x)$;

Capítulo 3

Aplicaciones de modelos a inversiones inmobiliarias

En este capítulo se va a presentar una aplicación de los modelos presentados en el capítulo anterior a un caso de inversión inmobiliaria. El objetivo va a ser la predicción de medidas de riesgo asociadas a inversiones inmobiliarias, modelando la mínima rentabilidad esperada de la inversión.

Un tipo de inversión inmobiliaria es la compra de inmuebles para su posterior alquiler. Siendo este el caso, el comprador debe estimar si merece la pena el precio de compra del inmueble, teniendo en cuenta los gastos asociados a la compra y, más importante, los ingresos esperados de ese alquiler. A la hora de acometer una inversión, se debe fijar un retorno mínimo anual esperado, al cual se referirá como rentabilidad mínima esperada, que se denotará r_{min}. En términos teóricos, se puede identificar el VaR con la renta mínima esperada, la cual se utilizará para hallar después r_{min} y ayudará a decidir si una inversión es rentable o no.

Por otro lado, también es interesante estimar cuál sería la rentabilidad esperada en caso de ser menor que r_{min}, ya que ayudará a decidir si, pese a no alcanzar la rentabilidad mínima esperada, merece la pena asumir dicho riesgo.

Por tanto, el objetivo de este capítulo es la estimación de la renta mínima esperada y la rentabilidad esperada en caso de ser menor que la mínima esperada con un nivel de confianza dado mediante los modelos presentados, utilizando datos de viviendas en la Comunidad de Madrid [**?**] mediante web scraping. Se han tomado todos los anuncios (aproximadamente 15000) de dicha web de viviendas para obtener una muestra relativamente grande que ayude a una más correcta estimación de los modelos.

Las variables utilizadas para su estimación han sido escogidas en base a su relevancia teórica y su disponibilidad en los anuncios de Idealista y son las siguientes:

- **Precio de la vivienda:** Alquiler mensual al que se oferta
- **Superficie:** Tamaño de la vivienda en metros cuadrados
- **Número de habitaciones:** Número de habitaciones de la vivienda
- **Número de baños:** Número de baños de la vivienda
- **Planta:** Planta en la que se encuentra la vivienda (se considera que los bajos y entreplantas corresponden a la planta 0)
- **Latitud:** Coordenada geográfica norte-sur de la vivienda
- **Longitud:** Coordenada geográfica este-oeste de la vivienda

- **Ascensor:** Si la vivienda dispone de ascensor o no
- **Obra nueva:** Si la vivienda es de obra nueva o no
- **Piscina:** Si la vivienda dispone de piscina o no
- **Terraza:** Si la vivienda dispone de terraza o no
- **Garaje:** Si la vivienda dispone de garaje o no
- **Garaje incluido en el precio:** Si el garaje está incluido en el precio del alquiler
- **Aire acondicionado:** Si la vivienda dispone de aire acondicionado o no
- **Trastero:** Si la vivienda dispone de trastero o no
- **Jardín:** Si la vivienda dispone de jardín o no
- **Disponibilidad de planta:** Si la información acerca de la planta en la que está la vivienda está disponible o no.

Una variable que se ha querido incluir pero no ha sido posible ha sido la fecha de construcción de la vivienda, que es esperable que influya con cierta importancia en el valor de alquiler de la vivienda. Sin embargo, dicho dato no está disponible en los anuncios de la web, por lo que se ha tenido que prescindir de él.

Al observar los datos, se ve que existen valores atípicos en el precio de alquiler de las viviendas, que pueden influir negativamente en la estimación de los modelos. Por ello, se realizará en primer lugar un modelado con los datos originales y posteriormente se eliminarán los valores atípicos. Los valores atípicos se han identificado siguiendo un criterio conservador, tomando todos los datos por encima de $q_{0,75} + 1{,}5 * (q_{0,75} - q_{0,25})$, utilizando el criterio de rango intercuartílico (IQR).

Se va a entrenar a los modelos cuantílicos y se compararán los resultados obtenidos con la regresión cuantil y los modelos basados en machine learning. En primer lugar se separan los datos en dos conjuntos, uno de entrenamiento y otro de prueba, de forma aleatoria, tomando el 75 % de los datos para el entrenamiento y el 25 % restante para la prueba.

Modelo	**VaR Score (Pérdida Pinball)**
Regresión Cuantílica (QR)	59,70
Red Neuronal Cuantílica (QNN)	43,61
Boosting Cuantílico (QBR)	39,87

Tabla 3.1: Evaluación de los modelos de estimación del VaR ($\alpha = 0{,}95$) en el conjunto de prueba.

El *VaR Score* representa el error medio del modelo al predecir el cuantil 0,05, por lo que un valor inferior indica un mejor rendimiento. Los tres modelos tienen un error medio relativamente bajo (se desvían menos de 60€ de la media) y se puede observar que el modelo de Boosting Cuantílico obtiene el resultado más bajo sugiriendo que es el más preciso de los tres. La Regresión Cuantílica, al ser un modelo lineal, obtiene el peor resultado, lo que puede ser debido a su incapacidad para capturar las complejas no relaciones entre las variables del mercado inmobiliario.

Ahora, tras observar estos resultados, se realiza el mismo procedimiento eliminando los valores atípicos en el precio de alquiler, obteniendo los resultados:

Modelo	**VaR Score (Pérdida Pinball)**
Regresión Cuantílica (QR)	43,38
Red Neuronal Cuantílica (QNN)	40,59
Boosting Cuantílico (QBR)	27,18

Tabla 3.2: Evaluación de los modelos de estimación del VaR ($\alpha = 0{,}95$) en el conjunto de prueba habiendo eliminado datos atípicos.

Hay una gran mejora en los resultados obtenidos, especialmente en los modelos de regresión cuantílica, tanto lineal como boosting. Por tanto, se decide utilizar estos modelos para la estimación del VaR.

En la figura **??** se muestran los valores reales del precio del alquiler junto con las predicciones del VaR para $\alpha = 0{,}95$ realizadas por los distintos modelos. Para facilitar la comparación, se han ordenado los precios reales de mayor a menor. Esta visualización permite observar gráficamente que los modelos tienden a ajustarse mejor en la parte media de la distribución de precios, debido a la mayor cantidad de datos de entrenamiento que en las colas. Este detalle debe tenerse en cuenta a la hora de la aplicación de los modelos en caso de que se quiera estimar el VaR de una vivienda con un precio en estas zonas.

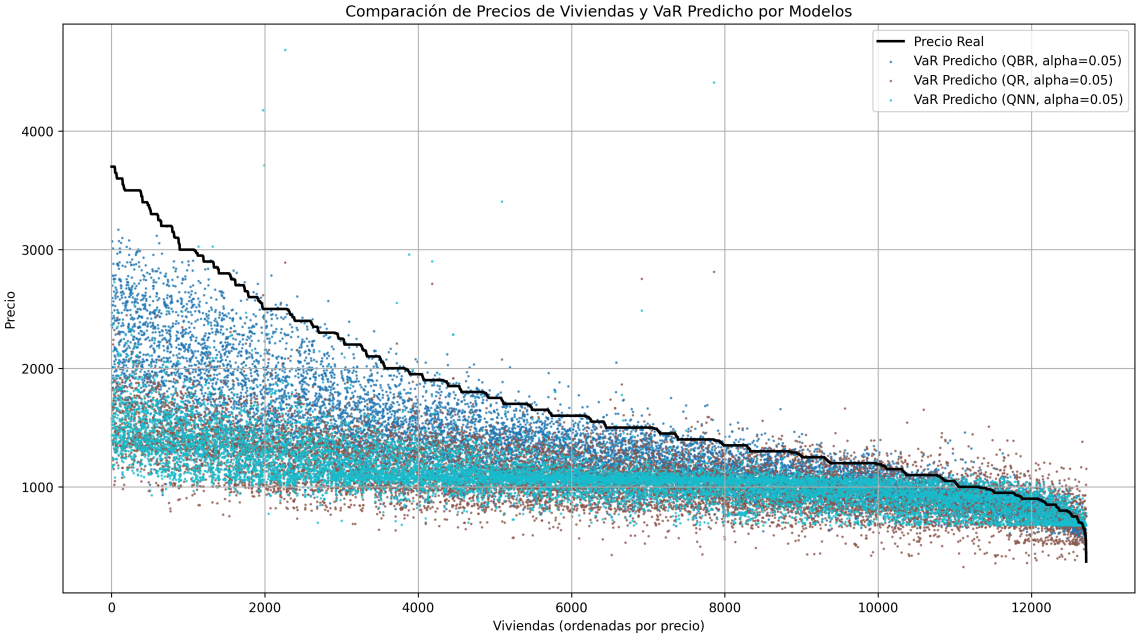

Figura 3.1: Precios reales y el VaR estimado por los modelos QR, QNN y QBR para $\alpha = 0{,}95$.

A pesar de que los tres modelos obtengan un resultado parecido, los resultados difieren. El modelo QBR, que ya se ha visto que era el más preciso, da la estimación más alta. Esto sugiere que ajusta de manera más óptima el riesgo específico de cada vivienda y es capaz de adaptarse a características particulares de la vivienda. En la figura **??** se muestra el error de cada modelo al predecir el VaR:

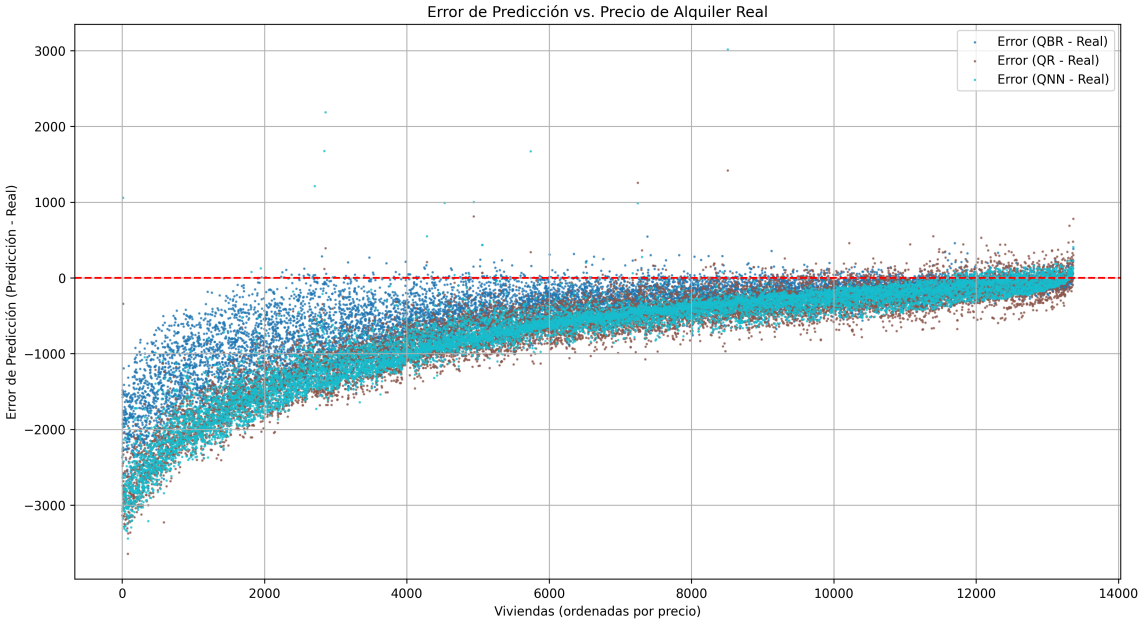

Figura 3.2: Error en la predicción VaR estimado por los modelos QR, QNN y QBR para $\alpha = 0{,}95$.

Se puede observar que el boosting es más preciso en las colas de la distribución y que las tres predicciones se ajustan bastante bien en la parte media y baja.

Una vez calculado el VaR, se va a calcular el Expected Shortfall (ES) para entender qué pasaría en el peor de los casos. En la figura **??** se observa la distribución de los ES en función del par de métodos utilizados (combinación entre el método para calcular el cuantil y el expectil necesarios):

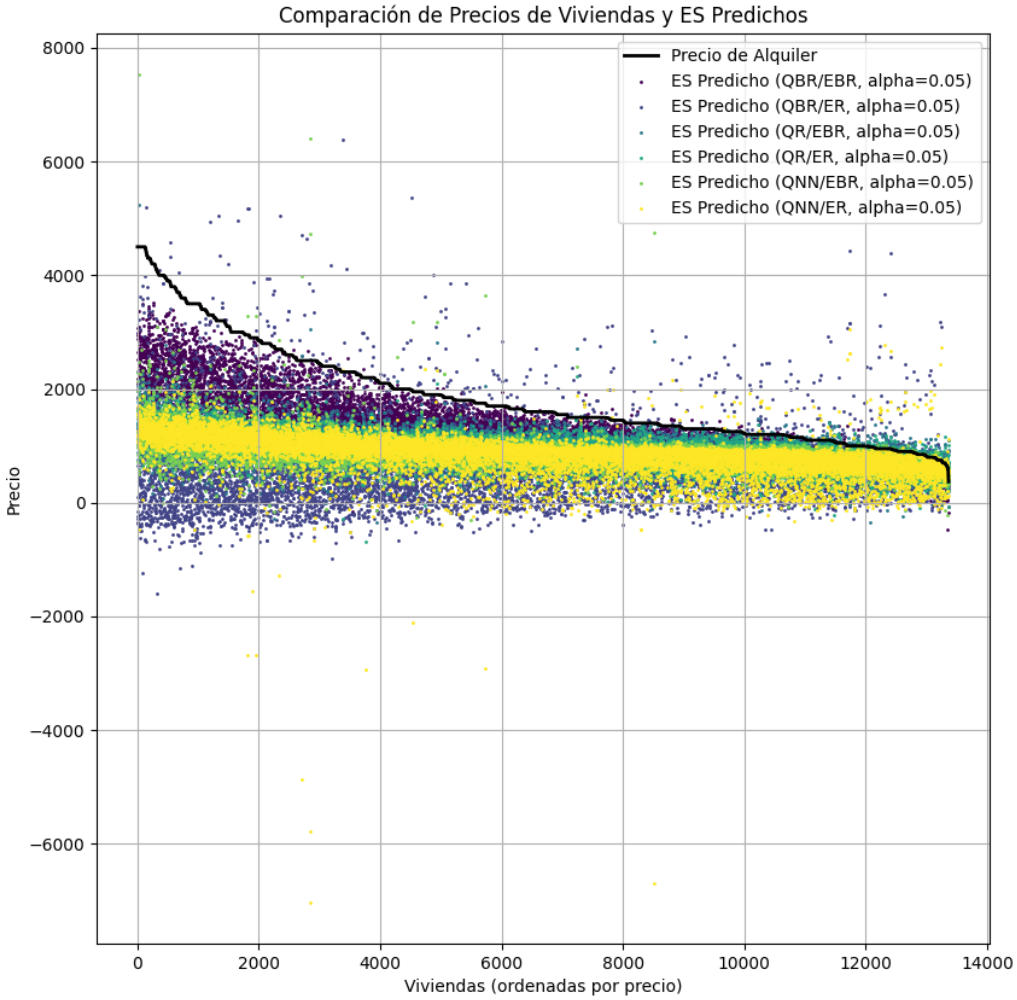

Figura 3.3: Precios reales y el ES estimado por los modelos $\alpha = 0{,}95$.

Se puede observar que los valores predichos están en su mayoría por debajo del precio del alquiler. Sin embargo, se puede ver alguna incorrección en algún cálculo, ya que esta inversión no puede tener retorno negativo y, sin embargo, se observa que el ES calculado en algunos casos es menor que 0. Esto se debe a que la precisión para calcular el cuantil es mayor que para calcular el expectil, lo cual hace que la estimación del ES se desvirtue.

3.1. Caso de uso real

Para ver cómo se comportan estos modelos en un caso práctico, se va a usar cada uno para predecir la renta mínima de una vivienda concreta tomada de otro portal de viviendas [**?**]. Dicha vivienda se vende por 625000€ y se va a comprobar cuál es la renta mínima esperada al poner al alquiler dicha vivienda. Los resultados se ven en la Tabla **??**.

Modelo	Predicción de VaR (Renta Mínima Estimada)
Regresión Cuantílica (QR)	1235,09 €
Red Neuronal Cuantílica (QNN)	1135,35 €
Boosting Cuantílico (QBR)	1637,66€

Tabla 3.3: Predicción del VaR ($\alpha = 0{,}95$) para la vivienda tomada como ejemplo.

Por otro lado, también se obtienen la estimaciones del ES para este caso de uso:

Modelo	Predicción de ES
QR & ER	1071,38 €
QR & EBR	1134,49 €
QBR & ER	233,81 €
QBR & EBR	1619,22 €
QNN & ER	951,29 €
QNN & EBR	1014,39 €

Tabla 3.4: Predicción del ES ($\alpha = 0{,}95$) para la vivienda tomada como ejemplo.

Una vez obtenidos estos resultados, se puede hacer un cálculo de la rentabilidad mínima esperada de la inversión. Para ello, se ha consultado una web de cálculo de hipotecas [**?**] y se han obtenido los siguientes datos:

- **Precio de la vivienda:** 625000€
- **Entrada:** 187500€ (30 % del precio de la vivienda)
- **Hipoteca:** 437500€ (70 % del precio de la vivienda)
- **Gastos aproximados de la compra:** 40228,31€
- **Intereses y gastos extra:** 2,74 %
- **Plazo:** 30 años
- **Cuota mensual:** 1728,45€
- **Intereses mensuales:** 513,37€

Para calcular la rentabilidad de una inversión inmobiliaria se debe tener en cuenta los ingresos netos que se van a percibir y los gastos totales asociados a la compra, sin considerar la parte de la hipoteca asociada a las amortizaciones del capital, ya que no representa un gasto real. Además del precio de compra/venta, se dispone de los valores de las variables explicativas utilizadas por los modelos predictivos, por lo que se usarán como entrada de los modelos ajustados para obtener una estimación del precio de alquiler. Por tanto, para calcular la rentabilidad anual, se aplica:

$$r_{min} = \frac{(\text{Ingresos mensuales} - \text{Gastos mensuales}) * 12}{\text{Entrada de la vivienda} + \text{Gastos asociados a la compra de la vivienda}}$$

Por tanto, la rentabilidad mínima esperada de la inversión según el modelo que se escoja es la si-

guiente:

$$r_{min}^{QBR} = \frac{(1637{,}66 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0592 = 5{,}92\,\%$$
$$r_{min}^{QR} = \frac{(1235{,}09 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0380 = 3{,}80\,\%$$
$$r_{min}^{QNN} = \frac{(1135{,}35 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0327 = 3{,}27\,\%$$

Por otro lado, también interesa saber la rentabilidad esperada en caso de que esta fuese menor a la mínima. En ese caso, se toma el ES:

$$r_{min}^{QBR,EBR} = \frac{(1619{,}22 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0583 = 5{,}83\,\%$$
$$r_{min}^{QR,EBR} = \frac{(1134{,}49 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0258 = 2{,}58\,\%$$
$$r_{min}^{QNN,EBR} = \frac{(1014{,}39 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0327 = 3{,}27\,\%$$
$$r_{min}^{QBR,ER} = \frac{(233{,}81 - 513{,}37) * 12}{187500 + 40228{,}31} = -0{,}0147 = -1{,}47\,\%$$
$$r_{min}^{QR,ER} = \frac{(1071{,}38 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0294 = 2{,}94\,\%$$
$$r_{min}^{QNN,ER} = \frac{(951{,}29 - 513{,}37) * 12}{187500 + 40228{,}31} = 0{,}0231 = 2{,}31\,\%$$

A pesar de no haberse tenido en cuenta gastos extra como la comunidad, el seguro de la vivienda y más, estos datos serían los que un inversor podría utilizar para valorar si merece la pena o no la inversión.

Bibliografía

[1] C. ACERBI Y D. TASCHE, *On the Coherence of Expected Shortfall*, Journal of Banking & Finance, 2002.

[2] C. ACERBI Y D. TASCHE, *Expected Shortfall: A Natural Coherent Alternative to Value at Risk*, Economic Notes, 2002.

[3] BANK FOR INTERNATIONAL SETTLEMENTS (BIS), *Basel III: A global regulatory framework for more resilient banks and banking systems*, rev. June 2011, https://www.bis.org/publ/bcbs189.pdf.

[4] BBVA, *Simulador de hipotecas*, BBVA.es, consultado el 7 de julio de 2025, disponible en https://www.bbva.es/personas/productos/hipotecas/simulador-hipotecas.html.

[5] L. BREIMAN, J. H. FRIEDMAN, R. A. OLSHEN Y C. J. STONE, *Classification and Regression Trees*, Wadsworth International Group, 1984.

[6] H. FÖLLMER Y A. SCHIED, *Convex and coherent risk measures*, Humboldt-Universität zu Berlin, Institut für Mathematik; versión del 8 de octubre de 2008, disponible en https://www.math.hu-berlin.de/~foellmer/papers/CCRM.pdf.

[7] FOTOCASA, *Anuncio de vivienda en venta en Madrid capital (ID: 185543795)*, Fotocasa.es, consultado el 6 de julio de 2025, disponible en https://www.fotocasa.es/es/comprar/vivienda/madrid-capital/aire-acondicionado-calefaccion-ascensor-no-amueblado/185543795/.

[8] P. I. FRAZIER, *A Tutorial on Bayesian Optimization*, 2018, consultado el 8 de julio de 2025, disponible en https://arxiv.org/pdf/1807.02811.

[9] T. HASTIE, R. TIBSHIRANI Y J. FRIEDMAN, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2017, disponible en https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12_toc.pdf.download.html.

[10] IDEALISTA, *Datos de viviendas en alquiler en la provincia de Madrid*, Idealista.com, consultado el 5 de julio de 2025, disponible en https://www.idealista.com/alquiler-viviendas/madrid-provincia/.

[11] S. IOFFE Y C. SZEGEDY, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015, disponible en http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43442.pdf.

[12] D. P. KINGMA Y J. BA, *Adam: A Method for Stochastic Optimization*, 2014 disponible en https://arxiv.org/pdf/1412.6980.

[13] R. KOENKER, *Quantile Regression*, Cambridge University Press, 2005.

[14] P. MARTÍNEZ-CAMBLOR, *Regresión Cuantilica: Estimación y Contrastes*, Universidad Autónoma de Madrid, 2020, disponible en https://www.uam.es/uam/media/doc/1606862082401/regresion-cuantilica-estimacion-y-contrastes.pdf.

[15] A. Y. NG, *Feature selection, L1 vs. L2 regularization, and rotational invariance*, Proceedings of the Twenty-First International Conference on Machine Learning (ICML), 2004, disponible en https://robotics.stanford.edu/~ang/papers/icml04-l1l2.pdf.

[16] J.P. MORGAN, *RiskMetrics — Technical Document*, 4ª edición, J.P. Morgan/Reuters, Nueva York, 1996, consultado el 9 de julio de 2025, disponible en https://www.msci.com/documents/10199/5915b101-4206-4ba0-aee2-3449d5c7e95a.

[17] W. K. NEWEY Y J. L. POWELL, *Asymmetric least squares estimation and testing*, Econometrica, 1987, https://doi.org/10.2307/1911031.

[18] K. K. OSMUNDSEN, *Using Expected Shortfall for Credit Risk Regulation*, University of Stavanger, Stavanger, 26 de febrero de 2017.

[19] M. OSORIO BUITRÓN, *Estimación del Valor en Riesgo (VaR) para la tasa de cambio (TRM) COP-USD mediante modelos ARCH*, Trabajo de grado en Estadística, Universidad del Valle, Santiago de Cali, Colombia, 2017, disponible en https://bibliotecadigital.univalle.edu.co/server/api/core/bitstreams/9ce73714-c4c7-45ec-82ac-c46591f78c2f/content.

[20] S. SARYKALIN, G. SERRAINO Y S. URYASEV, *Value-at-Risk vs. Conditional Value-at-Risk in Risk Management and Optimization*, Tutorials in Operations Research, INFORMS, 2008, https://pubsonline.informs.org/doi/epdf/10.1287/educ.1080.0052.

[21] R. E. SCHAPIRE Y Y. FREUND, *Boosting: Foundations and Algorithms*, MIT Press, 2012, https://direct.mit.edu/books/book/5342/Boosting-Foundations-and-Algorithms.

[22] N. SRIVASTAVA, G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER Y R. SALAKHUTDINOV, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014, disponible en https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[23] J.W. TAYLOR, *Estimating Value at Risk and Expected Shortfall Using Expectiles*, Journal of Financial Econometrics, vol. 6, n.º 2, 2008, pp. 231–252, disponible en https://users.ox.ac.uk/~mast0315/ExpectilesVaR&ES.pdf.

Apéndice 1: Árboles de regresión

Los árboles de regresión [**?**] son modelos predictivos no paramétricos que dividen el espacio de las variables de entrada en subregiones rectangulares disjuntas. Dentro de cada una de estas regiones, se ajusta un modelo simple para realizar predicciones. Este proceso de división se realiza de forma recursiva, creando una estructura jerárquica similar a un árbol.

El objetivo principal de un árbol de regresión es minimizar el error de predicción. Para ello, en cada paso de la construcción del árbol, se busca la mejor división posible de los datos. Una división se define por una variable predictora y un punto de corte y la "mejor"división es aquella que reduce al máximo una función de pérdida predefinida.

Se considera un conjunto de datos de entrenamiento con variables explicativas $\mathbf{X} = \{X_1, X_2, \ldots, X_p\}$ y una variable dependiente Y. El proceso de construcción del árbol implica encontrar divisiones binarias recursivas. Para cada nodo del árbol, se evalúan todas las posibles variables X_j y todos los posibles puntos de corte t para esa variable, con el fin de encontrar la división que optimice la función de pérdida. Por ejemplo, para la función de pérdida cuadrática, la división se realizaría de la siguiente manera

$$\mathscr{L}(X_j,t) = \sum_{i=1}^{n} I(X_{ij} \leq t)(Y_i - \bar{Y}_{X_{ij}\leq t})^2 + \sum_{i=1}^{n} I(X_{ij} > t)(Y_i - \bar{Y}_{X_{ij}>t})^2$$

donde $\bar{Y}_{X_i \leq t}$ y $\bar{Y}_{X_i > t}$ son las medias de Y en las regiones definidas por la división. Esta recursividad se repite hasta que se cumplen ciertos criterios de parada, como una profundidad máxima del árbol o un número mínimo de observaciones en un nodo.

Para los casos explorados en este trabajo, se han utilizado árboles de regresión binarios como predictores débiles del algoritmo de boosting. Estos árboles se han construido utilizando el proceso presentado, adaptando la función de pérdida. En el caso del boosting para cuantiles, se ha intentado minimizar la suma de las pérdidas pinball en cada región, es decir:

$$\mathscr{L}(X_j,t) = \sum_{i \in R_1(j,t)} \rho_\alpha(Y_i - c_1) + \sum_{i \in R_2(j,t)} \rho_\alpha(Y_i - c_2)$$

donde $R_1(j,t)$ y $R_2(j,t)$ son las dos regiones resultantes de la división por la variable X_j en el punto de corte t, y c_1 y c_2 son las predicciones óptimas (cuantiles condicionales) para cada región.

Mientras que para el caso de los expectiles, se ha intentado minimizar la suma de las pérdidas expectílicas en cada región:

$$\mathscr{L}(X_j,t) = \sum_{i \in R_1(j,t)} \phi_\tau(Y_i - e_1) + \sum_{i \in R_2(j,t)} \phi_\tau(Y_i - e_2)$$

donde e_1 y e_2 son las predicciones óptimas (expectiles condicionales) para cada región.

Una vez el árbol está construido, se puede realizar una predicción dados la introducción de un nuevo dato. El dato se introduce en el árbol desde el nodo raíz y, siguiendo las subdivisiones, llegará a una hoja. La predicción se realiza de forma distinta dependiendo si se quiere el cuantil o el expectil. Se supone el

conjunto de variables que caen en la hoja $\{Y_i\}_{i \in R}$, donde R es el conjunto de índices de las observaciones que caen en la hoja. Para calcular el cuantil de la hoja se toma el cuantil de las observaciones que caen en la hoja:

$$\hat{q}_\alpha = Y_{(\lceil \alpha \cdot n_h \rceil)}$$

Por otro lado, el expectil se calcula mediante el algoritmo Iteratively Reweighted Least Squares (IRLS) presentado por Newey y Powell [**?**].

A1.1. Ventajas y desventajas de los árboles de regresión

Los árboles de regresión son una herramienta muy poderosa y tiene unas grandes ventajas, pero también se pueden encontrar deficiencias:

Ventajas

- **Facilidad de Interpretación:** Son modelos muy fáciles de entender, ya que su estructura se parece a un diagrama de flujo de decisiones.
- **Manejo de Relaciones No Lineales:** Pueden capturar relaciones complejas y no lineales entre las variables sin necesidad de transformaciones previas de los datos.
- **No Requieren Normalización:** No es necesario escalar o normalizar las variables de entrada, simplificando la preparación de los datos.
- **Robustez ante Valores Atípicos:** Son robustos ante la presencia de valores atípicos en los datos.
- **Selección Automática de Características:** Realizan una selección automática de las características más relevantes al elegir las mejores divisiones.

Desventajas

- **Sobreajuste:** Son propensos al sobreajuste, sobre todo si los datos son ruidosos o el árbol muy profundo. Más adelante se exploran técnicas para evitarlo.
- **Inestabilidad:** Pequeñas variaciones en el conjunto de datos pueden dar lugar a árboles muy diferentes, lo que puede afectar a la generalización del modelo.
- **Sesgo hacia Divisiones con Muchas Características:** Pueden favorecer divisiones que involucran muchas características, lo que puede llevar a una interpretación errónea de la importancia de las variables.

Apéndice 2: Detalles de los algoritmos de optimización de los modelos

Con el objetivo de mejorar la precisión de los modelos, se han utilizado diferentes algoritmos de optimización para ajustar los parámetros asociados a cada modelo. Cada uno de ellos ha aportado una gran mejoría al rendimiento del modelo. Esto ha sido especialmente relevante en el caso de las redes neuronales, donde la optimización de los pesos y sesgos es crucial.

A2.1. Reescalado y normalización de los datos

Para mejorar la convergencia de los algoritmos de optimización, se ha optado por aplicar un reescalado y normalización de los datos, lo cual facilita el entrenamiento de todos los modelos. Para cada variable se realiza la siguiente transformación:

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

donde μ es la media de la variable y σ es su desviación estándar.

A2.2. Técnicas de regularización

Las regularizaciones son técnicas utilizadas para evitar el sobreajuste y hacer que el modelo se centre más en la tendencia general, en lugar de ajustar el ruido de los datos. Es muy frecuente observar en los modelos sobreajustados coeficientes o pesos con magnitudes elevadas, por esto una forma de mitigar el sobreajuste consiste en penalizar los modelos con coeficientes muy grandes. Se han utilizado técnicas de regularización L1 y L2 [**?**], que penalizan los pesos del modelo para evitar que se vuelvan demasiado grandes. Para lograrlo, se intenta minimizar una función de pérdida que incluye un término de regularización. Las funciones de pérdida incluyendo las regularizaciones L1 y L2 son las siguientes:

$$\mathscr{L}_{L1}(\mathbf{X_i}, \mathbf{Y}_i, g) = \sum_{i=1}^{m} \rho(\mathbf{Y}_i - f(\mathbf{X}_i)) + \upsilon \sum_{j=1}^{n} |w_j|$$

$$\mathscr{L}_{L2}(\mathbf{X_i}, \mathbf{Y}_i, g) = \sum_{i=1}^{m} \rho(\mathbf{Y}_i - f(\mathbf{X}_i)) + \lambda \sum_{j=1}^{n} w_j^2$$

$$\mathscr{L}_{L1+L2}(\mathbf{X_i}, \mathbf{Y}_i, g) = \sum_{i=1}^{m} \rho(\mathbf{Y}_i - f(\mathbf{X}_i)) + \upsilon \sum_{j=1}^{n} |w_j| + \lambda \sum_{j=1}^{n} w_j^2$$

con υ y λ los parámetros de regularización L1 y L2, respectivamente, y w_j todos los pesos del modelo. Al ser el objetivo minimizar la función de pérdida, el modelo trata de encontrar un equilibrio entre ajustar los datos y mantener los pesos con pequeños valores, lo cual ayuda a mejorar la sensibilidad a datos atípicos y la generalización del modelo.

Por otro lado, en las redes neuronales se utiliza una técnica llamada Dropout [**?**] . Por cada entrada de datos en el entrenamiento, se desactivarán de forma aleatoria un porcentaje de las neuronas de la red.

Con esto se consigue que la red, por una parte, reducir el sobreajuste y que determinadas neuronas no se vuelvan demasiado importantes y, por otra parte, simula un entrenamiento de múltiples modelos que, combinados, forma un modelo más robusto.

También, en las redes neuronales, se utiliza la normalización por lotes (Batch Normalization) [**?**], que consiste en normalizar las activaciones de cada capa de la red. Esto ayuda a estabilizar el entrenamiento y acelerarlo, ya que reduce la variación en la distribución de las entradas a cada capa a medida que cambian los pesos de las capas anteriores.

A2.3. Búsqueda de hiperparámetros: la Búsqueda Bayesiana

La correcta elección de los hiperparámetros es una de las claves para el éxito de un modelo de machine learning. Existen muchas técnicas para escogerlos. La que ofrece mejor rendimiento es la búsqueda de cuadrícula (Grid Search), que consiste en probar todas las combinaciones posibles de los hiperparámetros y escoger la que mejor rendimiento tenga. Sin embargo, es muy costosa computacional y temporalmente, por lo que se debe utilizar alguna otra técnica que pueda optimizarlos más eficientemente.

En este caso, se ha utilizado la búsqueda bayesiana [**?**]. Este método consiste en optimizar una función objetivo mediante la construcción de un modelo probabilístico que representa la función. Se ha optado por este método debido a que tiene una gran corrección cuando los hiperparámetros no superan un número razonable (en este caso, menos de 10).

Para ejemplificar el algoritmo de búsqueda bayesiana, se define un espacio de hiperparámetros $\mathscr{X}$, y un conjunto de combinaciones iniciales de hiperparámetros $\{\kappa_1, \kappa_2, \ldots, \kappa_{n_0}\} \subset \mathscr{X}$, generadas aleatoriamente.

Sea $f(\kappa)$ la función objetivo que se quiere optimizar, que evalúa la configuración de hiperparámetros $\kappa \in \mathscr{X}$. Además, se define una función de adquisición $a(\kappa \mid \mathscr{D})$ de la siguiente manera:

$$a(\kappa \mid \mathscr{D}) = \mathbb{E}\left[\text{máx}(f_{\text{mejor}} - f(\kappa), 0)\right]$$

con $\mathscr{D}$ el conjunto de observaciones de la función objetivo y f_{mejor} el mejor valor de la función objetivo encontrado hasta el momento.

Algorithm 3: Búsqueda Bayesiana

Inicializar el conjunto de observaciones $\mathscr{D} = \{\}$;
Definir una distribución a priori sobre la función objetivo $f(\kappa)$, que modela nuestra creencia inicial sobre su comportamiento (proceso gaussiano con media y covarianza predefinidas);
Generar n_0 configuraciones iniciales $\kappa_1, \ldots, \kappa_{n_0} \in \mathscr{X}$ aleatoriamente;
for $i = 1$ **to** n_0 **do**
    Evaluar $y_i = f(\kappa_i)$;
    Añadir (κ_i, y_i) a $\mathscr{D}$;
for $i = n_0 + 1$ **to** N **do**
    Actualizar la distribución de probabilidad a posteriori sobre f utilizando todos los datos disponibles $\mathscr{D}$;
    Construir la función de adquisición $a(\kappa \mid \mathscr{D})$;
    Seleccionar los siguientes hiperparámetros:

$$\kappa_i = \arg\max_{\kappa \in \mathscr{X}} a(\kappa \mid \mathscr{D})$$

    Evaluar $y_i = f(\kappa_i)$;
    Añadir (κ_i, y_i) a $\mathscr{D}$;

Apéndice 3: Código utilizado en Python y R

A3.1. Implementación de modelos en Python

```
   from ml.expectileboosting import EBR
from ml.quantileboosting import QBR
from ml.neuralnetwork import QuantileNeuralNetwork
from regression.quantileregression import QR
from regression.expectileregression import ER
import matplotlib.cm as cm
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import pandas as pd
import time


class ModelPredictor:
    def __init__(self, X, y):
        self.qbr = QBR(X, y)
        self.qr = QR(X, y)
        self.qnn  = QuantileNeuralNetwork(X, y)
        self.er = ER(X, y)
        self.ebr = EBR(X, y)

    def fit_quantile_models(self, alpha:float):
        self.qbr.fit(alpha)
        self.qr.fit(alpha)
        self.qnn.fit(alpha)

    def fit_expectile_models(self):
        for t in range(1, 100):
            t = t/100
            self.ebr.fit(t)
            self.er.fit(t)



    ###Hallamos el VaR con todas las tecnicas###
    def predict_VaR(self, method: str, X_scalar, alpha:float):
        if method == "QBR":
            return self.qbr.predict(alpha, X_scalar)
        elif method == "QR":
            return self.qr.predict(alpha, X_scalar)
        elif method == "QNN":
            return self.qnn .predict(alpha, X_scalar)

    def predict_ES_scalar(self, method: str, X_scalar:float, alpha: float,
    predicted_VaR_scalar:float, y):
        if method == "EBR":
```

```
            model = self.ebr
        elif method == "ER":
            model = self.er

        ##Hay que hallar el tau que corresponde al VaR
        min_diff = float("inf")
        expectile = None
        tau = None

        # Buscamos el tau que minimiza la diferencia entre el VaR predicho y expectil
        for t in range (1, 100):
            t = t/100
            exp = model.predict(t, X_scalar.reshape(1, -1))
            if abs(exp - predicted_VaR_scalar) < min_diff:
                min_diff = abs(exp - predicted_VaR_scalar)
                expectile = exp
                tau = t
            else:
                break
        if tau == 0.5:
            print("Warning: tau is  0.5, this might indicate an issue with the model or data.")
            return 0

        ES = predicted_VaR_scalar + tau/(alpha*(2*tau-1))*(model.predict(0.5, X_scalar.reshape(1, -1))-predicted_VaR_scalar)
        return ES


    def predict_ES(self, method: str, X, alpha: float, predicted_VaR:float, y):
        ES = np.zeros(len(X))
        for i in range(len(X)):
            ES[i] = self.predict_ES_scalar(method, X_scalar = X[i], alpha = alpha, predicted_VaR_scalar = predicted_VaR[i], y = y)
        return ES

    def VaR_Score(self, method: str, X, y, alpha: float):
        if method == "QBR":
            return self.qbr.score(alpha, X, y)
        elif method == "QR":
            return self.qr.score(alpha, X, y)
        elif method == "QNN":
            return self.qnn.score(alpha, X, y)

    def ES_Score(self, method: str, X, y, alpha: float):
        if method == "EBR":
            model = self.ebr
        elif method == "ER":
            model = self.er
        return model.score(alpha, X, y)




if __name__ == "__main__":
    # Campos del archivo cleaned_rental_data.csv:
    # precio: Precio del alquiler (variable objetivo)
    # metros_cuadrados_construidos: Metros cuadrados construidos de la propiedad
    # habitaciones: N mero de habitaciones
    # banos: N mero de ba os
    # planta: Planta en la que se encuentra la propiedad (0 para bajo, -1 para s tano , etc.)
```

```
    # latitud: Latitud de la ubicaci n de la propiedad
    # longitud: Longitud de la ubicaci n de la propiedad
    # ascensor: Si la propiedad tiene ascensor (1 si s  , 0 si no)
    # obra_nueva: Si la propiedad es de obra nueva (1 si s  , 0 si no)
    # piscina: Si la propiedad tiene piscina (1 si s  , 0 si no)
    # terraza: Si la propiedad tiene terraza (1 si s  , 0 si no)
    # parking: Si la propiedad tiene parking (1 si s  , 0 si no)
    # parking_incluido_en_el_precio: Si el parking est  incluido en el precio (1 si s  , 0 si no)
    # aire_acondicionado: Si la propiedad tiene aire acondicionado (1 si s  , 0 si no)
    # trastero: Si la propiedad tiene trastero (1 si s  , 0 si no)
    # jardin: Si la propiedad tiene jard n (1 si s  , 0 si no)
    # planta_was_nan: Si el valor de la planta original era NaN (1 si s  , 0 si no)

    new_house = [73, 3, 1, 5, 40.418395, -3.677125, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0]

    df = pd.read_csv("/home/hugonaudin/Documentos/UNI/24-25/TFG/codigo/credits/rental_data/cleaned_rental_data.csv", sep=',', encoding='utf-8-sig')
    y = df["precio"]
    X = df.drop(columns=["precio"]).values
#
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    modelpredictor = ModelPredictor(X_scaled,y)

    modelpredictor.fit_quantile_models(0.05)
    modelpredictor.fit_expectile_models()
#
    quantile_types = ["QBR", "QR", "QNN"]
    expectile_types = ["EBR", "ER"]
    VaR_score = {}
    ES_score = {}
    predictions_for_plot = {}
    predictions_ES_for_plot = {}
#
    for quantile_type in quantile_types:
        predicted_VaR = modelpredictor.predict_VaR(quantile_type, X_scaled, 0.05)
        predictions_for_plot[quantile_type] = predicted_VaR
        VaR_score[quantile_type] = modelpredictor.VaR_Score(quantile_type, X_scaled, y, 0.05)

    print("VaR Scores:")
    for quantile_type, score in VaR_score.items():
        print(f"{quantile_type}: {score}")


    df = pd.read_csv("rental_data/cleaned_rental_data.csv", sep=',', encoding='utf-8-sig')

    y = df["precio"]
    X = df.drop(columns=["precio"]).values
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    modelpredictor = ModelPredictor(X_scaled,y)

    modelpredictor.fit_quantile_models(0.05)
    modelpredictor.fit_expectile_models()

    for quantile_type in quantile_types:
```

```
        predicted_VaR = modelpredictor.predict_VaR(quantile_type, X_scaled, 0.05)
        predictions_for_plot[quantile_type] = predicted_VaR
        VaR_score[quantile_type] = modelpredictor.VaR_Score(quantile_type, X_scaled, y, 0.05)


    print("VaR Scores:")
    for quantile_type, score in VaR_score.items():
        print(f"{quantile_type}: {score}")

    # --- Creaci n de la gr fica ---
    plot_df = pd.DataFrame({'Rent Price': y})
    for model, predictions in predictions_for_plot.items():
        plot_df[f'{model} VaR'] = predictions

    # Ordenar para graficar
    plot_df_sorted = plot_df.sort_values(by='Rent Price', ascending=False).reset_index(drop=True)

    # Configurar figura
    plt.figure(figsize=(15, 8))

    # L nea de precios reales
    plt.plot(
        plot_df_sorted.index,
        plot_df_sorted['Rent Price'],
        label='Precio de Alquiler',
        color='black',
        linewidth=2
    )

    # Colormap para asignar colores distintos autom ticamente
    cmap = cm.get_cmap('tab10', len(predictions_for_plot))

    # Pintar cada modelo con color distinto
    for idx, model in enumerate(predictions_for_plot.keys()):
        plt.scatter(
            plot_df_sorted.index,
            plot_df_sorted[f'{model} VaR'],
            label=f'VaR Predicho ({model}, alpha=0.05)',
            color=cmap(idx),
            s=1, #puntos m s peque os
            alpha=0.7
        )

    plt.title('Comparaci n de Precios de Viviendas y VaR Predicho por Modelos')
    plt.xlabel('Viviendas (ordenadas por precio)')
    plt.ylabel('Precio')
    plt.legend()
    plt.grid(True)

    plt.savefig("grafico_var_vs_precio2.png", dpi=300, bbox_inches='tight')
    plt.show()
    # --- Fin de la gr fica ---


    # --- Creaci n de la gr fica de errores ---
    plot_df = pd.DataFrame({'Rent Price': y})
    for model, predictions in predictions_for_plot.items():
        plot_df[f'{model} VaR'] = predictions

    # Ordenar para graficar
```

```
    plot_df_sorted = plot_df.sort_values(by='Rent Price', ascending=False).reset_index(drop=True)

    # Configurar figura
    plt.figure(figsize=(15, 8))

    # Colormap para asignar colores distintos autom ticamente
    cmap = cm.get_cmap('tab10', len(predictions_for_plot))

    # Pintar cada modelo con color distinto
    for idx, model in enumerate(predictions_for_plot.keys()):
        error = plot_df_sorted[f'{model} VaR'] - plot_df_sorted['Rent Price']
        plt.scatter(
            plot_df_sorted.index,
            error,
            label=f'Error ({model} - Real)',
            color=cmap(idx),
            s=1,
            alpha=0.7
        )

    plt.title('Error de Predicci n vs. Precio de Alquiler Real')
    plt.xlabel('Viviendas (ordenadas por precio)')
    plt.ylabel('Error de Predicci n (Predicci n - Real)')
    plt.legend()
    plt.grid(True)
    plt.axhline(y=0, color='r', linestyle='--') # Add a horizontal line at y=0 to see the error bias

    plt.savefig("grafico_errores_prediccion.png", dpi=300, bbox_inches='tight')
    plt.show()
    # --- Fin de la gr fica ---


    ##predecir el precio de new_house
    new_house = np.array(new_house).reshape(1, -1)
    scaled_new_house = scaler.transform(new_house)

    print("Predicted VaR for new house at 0.05:", modelpredictor.predict_VaR("QBR", scaled_new_house, 0.05))
    print("Predicted VaR for new house at 0.05:", modelpredictor.predict_VaR("QR", scaled_new_house, 0.05))
    print("Predicted VaR for new house at 0.05:", modelpredictor.predict_VaR("QNN", scaled_new_house, 0.05))

    print("\n\n--- ES Calculation ---")
    for quantile_type in quantile_types:
        predicted_VaR = modelpredictor.predict_VaR(quantile_type, scaled_new_house, 0.05)
        print(f"\nCalculating ES for VaR from {quantile_type} (VaR = {predicted_VaR}")
        for expectile_type in expectile_types:
            predicted_ES = modelpredictor.predict_ES_scalar(
                method=expectile_type,
                X_scalar=scaled_new_house[0],
                alpha=0.05,
                predicted_VaR_scalar=predicted_VaR[0],
                y=y
            )
            score = modelpredictor.ES_Score(expectile_type, X_scaled, y, 0.05)
            print(f"  - With {expectile_type} and {quantile_type}:  (ES: {prediected_ES:.2f})")

```

```
274
275     # --- Calculation of ES for all data points ---
276     es_predictions_for_plot = {}
277     for quantile_type in quantile_types:
278         predicted_VaR = predictions_for_plot[quantile_type]
279         for expectile_type in expectile_types:
280             print(f"Calculating ES for {quantile_type} and {expectile_type}")
281             predicted_ES = modelpredictor.predict_ES(
282                 method=expectile_type,
283                 X=X_scaled,
284                 alpha=0.05,
285                 predicted_VaR=predicted_VaR,
286                 y=y
287             )
288             es_predictions_for_plot[(quantile_type, expectile_type)] =
    predicted_ES
289
290     # --- Create plot with ES ---
291     plot_df = pd.DataFrame({'Rent Price': y})
292     for model, predictions in predictions_for_plot.items():
293         plot_df[f'{model} VaR'] = predictions
294
295     for (q_model, e_model), predictions in es_predictions_for_plot.items():
296         plot_df[f'ES ({q_model}/{e_model})'] = predictions
297
298     # Sort to plot
299     plot_df_sorted = plot_df.sort_values(by='Rent Price', ascending=False).
    reset_index(drop=True)
300
301     # Configure figure
302     plt.figure(figsize=(15, 8))
303
304     # Real prices line
305     plt.plot(
306         plot_df_sorted.index,
307         plot_df_sorted['Rent Price'],
308         label='Precio de Alquiler',
309         color='black',
310         linewidth=2
311     )
312
313
314     # Colormap for ES
315     cmap_es = cm.get_cmap('viridis', len(es_predictions_for_plot))
316
317     # Plot each ES model
318     for idx, ((q_model, e_model), _) in enumerate(es_predictions_for_plot.items
    ()):
319         plt.scatter(
320             plot_df_sorted.index,
321             plot_df_sorted[f'ES ({q_model}/{e_model})'],
322             label=f'ES Predicho ({q_model}/{e_model}, alpha=0.05)',
323             color=cmap_es(idx),
324             s=1, # Make ES points larger to distinguish them
325         )
326
327     plt.title('Comparaci n de Precios de Viviendas y ES Predichos')
328     plt.xlabel('Viviendas (ordenadas por precio)')
329     plt.ylabel('Precio')
330     plt.legend()
331     plt.grid(True)
332
333     plt.savefig("grafico_es_vs_precio.png", dpi=300, bbox_inches='tight')
```

```
    plt.show()
    # --- End of ES plot ---
```

Listado A3.1: main.py

```
   from sklearn.model_selection import KFold
from sklearn.linear_model import QuantileRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import make_scorer, mean_pinball_loss




class QR:
    def __init__(self, X, y ):
        self.X_train, self.X_test,self. y_train, self.y_test = train_test_split(
    X, y, random_state=0)
        self.models = {}
        self.predictions_train = {}
        self.predictions_test = {}


    def neg_mean_pinball_loss_scorer(self, alpha):
        return make_scorer(mean_pinball_loss, alpha=alpha,greater_is_better=
    False, )

    #Entrenamiento del modelo
    def fit(self, alpha: float):
        print(f"----------------------------------------Training model for alpha:
     {alpha} ----------------------------------")

        param_grid = {
            'alpha': [0.0, 0.01, 0.1, 0.5, 1.0, 2.0, 5.0],  # Regularizaci n L1
            'solver': ['highs-ds', 'highs-ipm', 'highs'],     # solvers
            'fit_intercept': [True, False]                     #intercepto
        }
        model = QuantileRegressor(quantile=alpha)
        kf = KFold(n_splits=10, shuffle=True, random_state=42)

    #Busqueda de hiperpar metros con validaci n cruzada
        grid_search = GridSearchCV(
            estimator=model,
            param_grid=param_grid,
            cv=kf,
            scoring=self.neg_mean_pinball_loss_scorer(alpha),
            n_jobs=-1,
            verbose=1,  # Para ver el progreso
            return_train_score=True
        )

        # Entrenar con b squeda de hiperpar metros
        print("Buscando mejores hiperpar metros...")
        grid_search.fit(self.X_train, self.y_train)

        # Guardar el mejor modelo
        self.models[alpha] = grid_search.best_estimator_


    def predict(self, alpha, X):
        prediction = self.models[alpha].predict(X)
        return prediction
```

```
55
56      def score(self, alpha: float, X, y):
57          prediction = self.predict(alpha, X)
58          score = mean_pinball_loss(y, prediction, alpha=alpha)
59          return score
60
61
```

Listado A3.2: quantileregression.py

```
1     from scipy.optimize import minimize
2  import numpy as np
3  import pandas as pd
4  import matplotlib.pyplot as plt
5  from sklearn.model_selection import train_test_split, KFold, cross_val_score
6  from sklearn.metrics import make_scorer
7  from sklearn.base import BaseEstimator
8
9
10
11
12 ### Implementacion de Expectile Regression Lineal
13 class ExpectileRegression(BaseEstimator):
14
15     #Metodo necesario
16     def __init__(self, expectile: float =.5, **kwargs):
17         assert 0 < expectile < 1, \
18             f"expectile parameter must be stricly 0 < e < 1, but it equals: {
    expectile}"
19         self.expectile = expectile
20         super().__init__(**kwargs)
21
22     #Metodo necesario
23     def get_params(self, deep=True):
24         return {'expectile': self.expectile}
25
26     #Metodo necesario
27     def set_params(self, **params):
28         for key, value in params.items():
29             if hasattr(self, key):
30                 setattr(self, key, value)
31             else:
32                 raise ValueError(f'Invalid parameter {key} for estimator {self.
    __class__.__name__}')
33         return self
34
35     #Entrenamiento del modelo
36     def fit(self, X, y):
37
38         if type(X) is pd.DataFrame:
39             X = X.values
40
41         if type(y) in [pd.DataFrame, pd.Series]:
42             y = y.values
43
44         # Adding column of ones
45         n = X.shape[0]
46
47         #Se pone  una columna de unos para el intercepto
48         X = np.hstack([X, np.ones(shape=(n, 1))])
49
50         self.beta = np.random.rand(X.shape[1])
51
52         #Perdida
```

```
        def expectile_loss(beta, *args):
            y_hat = X @ beta
            errors = y.reshape(-1, 1) - y_hat.reshape(-1, 1)
            return (
                    np.where(
                        errors < 0, 1 - self.expectile , self.expectile
                    ) * errors ** 2
                   ).mean()

        # Gradiente de la perdida
        def expectile_grad(beta, *args):
            y_hat = X @ beta
            errors = y.reshape(-1, 1) - y_hat.reshape(-1, 1)
            return (
                    -2 * X.T @ (
                        np.where(
                            errors < 0, 1 - self.expectile , self.expectile
                        ) * errors
                      )
                    ) / X.shape[0]

        # Minimizacion
        res = minimize(
            fun    = expectile_loss,
            x0     = self.beta,
            args   = None,
            method = 'SLSQP' ,
            jac    = expectile_grad
        )

        self.beta = res.x

        return self

    #Prediccion del modelo
    def predict(self, X):
        n = X.shape[0]
        X_ = np.hstack([X, np.ones(shape=(n, 1))])
        return X_ @ self.beta


class ER:
    def __init__(self, X, y ):
        self.X_train, self.X_test,self. y_train, self.y_test = train_test_split(
    X, y, random_state=0)
        self.models = {}
        self.predictions_train = {}
        self.predictions_test = {}


    def neg_mean_expectile_loss_scorer(self, expectile):
        def score_fn(y_true, y_pred):
            errors = y_true - y_pred
            return -np.mean(np.where(errors < 0, 1 - expectile, expectile) *
    errors ** 2)

        return make_scorer(score_fn, greater_is_better=False)

    def fit(self, expectile: float):
        print(f"-------------------------------------Training model for
    expectile: {expectile} ----------------------------------")

```

```
        model = ExpectileRegression(expectile=expectile)
        kf = KFold(n_splits=5, shuffle=True, random_state=42)

        #Validacion cruzada
        scores = cross_val_score(
            model,
            self.X_train,
            self.y_train,
            cv=kf,
            scoring=self.neg_mean_expectile_loss_scorer(expectile),
            n_jobs=1
        )
        score = scores.mean()
        print(f"Score for expectile {expectile}: {score}")
        model.fit(self.X_train, self.y_train)

        self.models[expectile] = model

    def predict(self, expectile, X):
        prediction = self.models[expectile].predict(X)
        return prediction

    def score(self, alpha: float, X, y):
        prediction = self.predict(alpha, X)
        errors = y - prediction
        return np.mean(np.where(errors < 0, 1 - alpha, alpha) * errors ** 2)


```

Listado A3.3: expectileregression.py

```
   from sklearn.ensemble import GradientBoostingRegressor
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import train_test_split, HalvingRandomSearchCV
from sklearn.metrics import make_scorer, mean_pinball_loss
from sklearn.datasets import fetch_california_housing
import matplotlib.pyplot as plt
import numpy as np


class QBR:
    def __init__(self, X, y ):
        self.X_train, self.X_test,self. y_train, self.y_test = train_test_split(
    X, y, random_state=0)
        self.models = {}
        self.predictions_train = {}
        self.predictions_test = {}

    # Scorer para la funci n de p rdida pinball
    def neg_mean_pinball_loss(self, alpha):
        return make_scorer(mean_pinball_loss, alpha=alpha, greater_is_better=
    False, )

    def fit(self, alpha: float):
        learning_rate = np.logspace(-3, -0.5, 20)  # De 0.001 a ~0.316
        max_depth = np.arange(2, 12, 1)
        min_samples_leaf = np.arange(1, 31, 2)
        min_samples_split = np.arange(5, 101, 5)

        param_grid = dict(
            learning_rate=learning_rate,
            max_depth=max_depth,
            min_samples_leaf=min_samples_leaf,
            min_samples_split=min_samples_split,
```

```
        )

        neg_mean_pinball_loss_scorer= self.neg_mean_pinball_loss(alpha)

        gbr = GradientBoostingRegressor(loss="quantile", alpha=alpha,
    random_state=0)

        #busqueda de hiperpar metros aleatoria
        search_p = HalvingRandomSearchCV(
            gbr,
            param_grid,
            resource="n_estimators",
            max_resources=250,
            min_resources=50,
            scoring=neg_mean_pinball_loss_scorer,
            n_jobs=1,
            random_state=0,
            verbose = 0,
            cv=5
        ).fit(self.X_train, self.y_train)

        self.models[alpha] = search_p


    def predict(self, alpha:float, X):
        prediction = self.models[alpha].predict(X)
        return prediction

    def score(self, alpha: float, X, y):
        prediction = self.predict(alpha, X)
        score = mean_pinball_loss(y, prediction, alpha=alpha)
        return score

```

Listado A3.4: quantileboosting.py

```
   import numpy as np
from sklearn.model_selection import train_test_split
from skopt import BayesSearchCV
from skopt.space import Real, Integer
from sklearn.metrics import make_scorer
from lightgbm.sklearn import LGBMRegressor
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

## Funci n de p rdida expectile -> gradiente y hessiano
def expectile_loss(tau):
    def loss(y_true, y_pred):
        residual = y_true - y_pred
        grad = -2 * np.where(residual < 0, 1 - tau, tau) * residual
        hess = 2 * np.where(residual < 0, 1 - tau, tau)
        return grad, hess
    return loss

##Clase del expectile boosting con LightGBM
class ExpectileLGBM(LGBMRegressor):
    def __init__(self, tau=0.5, **kwargs):
        super().__init__(**kwargs)
        self.tau = tau

    def fit(self, X, y, **kwargs):
        self.set_params(objective=expectile_loss(self.tau))
        return super().fit(X, y, **kwargs)
```

```
29
30
31
32 class EBR:
33     def __init__(self, X, y):
34         self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
    X, y, random_state=0)
35         self.models = {}
36         self.predictions_train = {}
37         self.predictions_test = {}
38         self.tau = None
39
40     #Scorer para la funci n de p rdida expectile
41     def neg_mean_expectile_loss_scorer(self, expectile):
42         def score_fn(y_true, y_pred):
43             errors = y_true - y_pred
44             return -np.mean(np.where(errors < 0, 1 - expectile, expectile) *
    errors ** 2)
45         return make_scorer(score_fn, greater_is_better=False)
46
47
48     def fit(self, expectile: float):
49
50         print (f"Fitting Expectile Boosting model for expectile {expectile}...")
51         self.tau = expectile
52
53         param_grid = {
54             "learning_rate": Real(0.01, 0.3, prior="log-uniform"),
55             "max_depth": Integer(2, 12),
56             "min_data_in_leaf": Integer(5, 50),
57             "num_leaves": Integer(15, 150),
58             "subsample": Real(0.6, 1.0),
59             "colsample_bytree": Real(0.6, 1.0),
60             "reg_alpha": Real(1e-4, 1.0, prior="log-uniform"),
61             "reg_lambda": Real(1e-4, 1.0, prior="log-uniform"),
62             "min_split_gain": Real(0, 0.3),
63             "max_bin": Integer(63, 255)
64         }
65
66
67         neg_mean_expectile_loss_scorer = self.neg_mean_expectile_loss_scorer(
    expectile)
68
69
70         base_model = ExpectileLGBM(
71             tau=expectile,
72             random_state=0,
73             verbose=-1,
74             n_estimators=300  # En lugar de early stopping
75         )
76
77         # Busqueda bayesiana de hiperpar metros
78         search = BayesSearchCV(
79             estimator=base_model,
80             search_spaces=param_grid,
81             scoring=neg_mean_expectile_loss_scorer,
82             n_iter=40,
83             cv=3,
84             n_jobs=-1,
85             random_state=0,
86             verbose=0
87         )
88
```

```
        search.fit(self.X_train, self.y_train)

        self.models[expectile] = search.best_estimator_

        print(f"Best score for expectile {expectile}: {search.best_score_}")

    def predict(self, expectile, X):
        prediction = self.models[expectile].predict(X)
        return prediction

    def score(self, alpha: float, X, y):
        prediction = self.predict(alpha, X)
        errors = y - prediction
        return np.mean(np.where(errors < 0, 1 - alpha, alpha) * errors ** 2)

```

Listado A3.5: expectileboosting.py

```
   import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
import keras.backend as K
from keras import regularizers
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_pinball_loss
from skopt import BayesSearchCV
from skopt.space import Real,Integer
from scikeras.wrappers import KerasRegressor
import gc



@tf.keras.utils.register_keras_serializable()
class PinballLoss(tf.keras.losses.Loss):
    def __init__(self, tau, name="pinball_loss", **kwargs):
        super().__init__(name=name, **kwargs)
        self.tau = tau

    def call(self, y_true, y_pred):
        diff = y_true - y_pred
        return tf.reduce_mean(tf.maximum(self.tau * diff, (self.tau - 1) * diff)
    )

    def get_config(self):
        config = super().get_config()
        config.update({"tau": self.tau})
        return config


###Creacion el modelo segun hiperparametros
def create_quantile_model(quantile, input_shape, hidden_layers=2,
    neurons_per_layer=64,
                          learning_rate=0.01, dropout_rate=0.0, activation='relu
    ', l2_penalty=1e-4):
    model = keras.Sequential()
    model.add(layers.Dense(neurons_per_layer,
                           input_shape=input_shape,
                           kernel_regularizer=regularizers.l2(l2_penalty)))
    model.add(layers.BatchNormalization())
    model.add(layers.Activation(activation))
    if dropout_rate > 0:
        model.add(layers.Dropout(dropout_rate))
    for _ in range(hidden_layers - 1):
```

```
        model.add(layers.Dense(neurons_per_layer, activation=activation,
    kernel_regularizer = regularizers.l2(l2_penalty)))
        model.add(layers.BatchNormalization())
        model.add(layers.Activation(activation))
        if dropout_rate > 0:
            model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(1))
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
        loss=PinballLoss(quantile)
    )
    return model


class QuantileNeuralNetwork:
    def __init__(self, X, y, validation_split=0.2):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
    X, y, random_state=42)
        self.validation_split = validation_split
        self.quantiles = []
        self.predictions_train = {}
        self.predictions_test = {}
        self.best_params = {}
        self.optimization_results = {}

    def bayesian_search_optimization(self, quantile, n_iter=40, param_space=None
    ):
        print(f"Bayesian Search para quantile {quantile} ({n_iter} iteraciones)"
    )
        if param_space is None:
            param_space = {
                'model__hidden_layers': Integer(1, 7),
                'model__neurons_per_layer': Integer(16, 512),
                'model__learning_rate': Real(1e-4, 1e-1, prior='log-uniform'),
                'model__dropout_rate': Real(0.0, 0.6),
                'batch_size': Integer(8, 128),
                'epochs': Integer(50, 300)
            }

        reg = KerasRegressor(
            model=create_quantile_model,
            quantile=quantile,
            input_shape=(self.X_train.shape[1],)
        )

        opt = BayesSearchCV(
            reg,
            param_space,
            n_iter=n_iter,
            cv=3,
            n_jobs=-1,
            verbose=0,
            random_state=42
        )

        opt.fit(self.X_train, self.y_train)

        best_params = opt.best_params_
        best_score = opt.best_score_

        best_params = {k.replace('model__', ''): v for k, v in best_params.items
    ()}

```

```
        self.best_params[quantile] = best_params
        self.optimization_results[quantile] = {
            'best_score': best_score,
            'best_params': best_params,
            'all_results': opt.cv_results_
        }

        print(f"Mejores par metros para quantile {quantile}:")
        for key, value in best_params.items():
            print(f"  {key}: {value}")
        print(f"Score: {best_score:.4f}")

        return best_params, best_score

    def fit(self, quantile: float):
        print(f"-------------------------------------Entrenando modelo para quantile: {quantile} ----------------------------------")

        #Busqueda de hiperparametros optimos
        best_params, best_score = self.bayesian_search_optimization(quantile)

        model = create_quantile_model(
            quantile=quantile,
            input_shape=(self.X_train.shape[1],),
            hidden_layers=best_params['hidden_layers'],
            neurons_per_layer=best_params['neurons_per_layer'],
            learning_rate=best_params['learning_rate'],
            dropout_rate=best_params['dropout_rate']
        )

        early_stopping = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)
        reduce_lr = keras.callbacks.ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=5,
            min_lr=1e-4,
            verbose=0
        )

        model.fit(
            self.X_train, self.y_train,
            epochs=best_params['epochs'],
            batch_size=best_params['batch_size'],
            validation_split=self.validation_split,
            verbose=0,
            callbacks=[early_stopping, reduce_lr]
        )

        self.quantiles.append(quantile)
        model.save(f"ml/models/quantile_model_{quantile}.keras")

        del model
        model = None
        K.clear_session()
        gc.collect()
        print(f"Modelo para quantile {quantile} entrenado y guardado.")

    def predict(self, alpha, X):
        print(f"Prediciendo quantile {alpha}...")
        model = keras.models.load_model(f"ml/models/quantile_model_{alpha}.keras", custom_objects={'PinballLoss': PinballLoss(alpha)})
        prediction = model.predict(X, verbose=0)
```

```
        return prediction

    def score(self, alpha: float, X, y):
        prediction = self.predict(alpha, X)
        score = mean_pinball_loss(y, prediction, alpha=alpha)
        return score


```

Listado A3.6: neuralnetwork.py

A3.2. Limpieza de datos en R

```
   # Cargar el dataset
data <- read.csv("credits/rental_data/cleaned_rental_data.csv")
# Lista de variables num ricas
vars <- c("precio", "metros_cuadrados_construidos", "habitaciones",
          "banos", "planta", "latitud", "longitud")

# Funci n para contar outliers con criterio de IQR
count_outliers <- function(x) {
  x <- na.omit(x)
  Q1 <- quantile(x, 0.25)
  Q3 <- quantile(x, 0.75)
  IQR <- Q3 - Q1
  upper <- Q3 + 1.5 * IQR
  sum(x > upper)
}

# Funci n para detectar outliers (devuelve vector l gico)
get_outliers_logical <- function(x) {
  x <- na.omit(x)
  Q1 <- quantile(x, 0.25)
  Q3 <- quantile(x, 0.75)
  IQR <- Q3 - Q1
  upper <- Q3 + 1.5 * IQR
  return(x > upper)
}


# Aplicar a cada variable del data_sample
outlier_counts <- sapply(vars, function(var) {
  count_outliers(data[[var]])
})

# Mostrar resultados
outlier_counts


# Funci n auxiliar para alinear con el vector original (por si hay NAs)
align_outliers <- function(outliers_logical, original_vector) {
  full <- rep(NA, length(original_vector))
  full[!is.na(original_vector)] <- outliers_logical
  return(full)
}

# Detectar outliers
precio_outliers <- align_outliers(get_outliers_logical(data$precio), data$precio
    )


```

```r
# Eliminar esas filas del dataset
data_sample_clean <- data[!(precio_outliers), ]

# Guardar el dataset limpio en un archivo CSV
write.csv(data_sample_clean, "outlierless.csv", row.names = FALSE)

outlierless <- read.csv("credits/rental_data/outlierless.csv")
```

Listado A3.7: clean_data.R