ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

Máster en Big Data: Tecnología y Analítica Avanzada

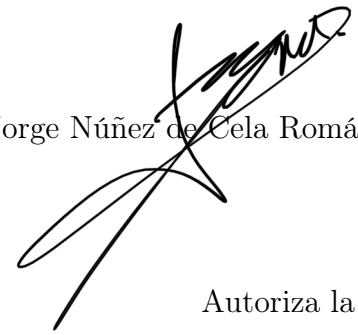
Modelo para la predicción de precios de renting de vehículos.

Autor
Jorge Núñez de Cela Román

Dirigido por
Domingo Guinea García-Alegre

Madrid
Enero 2025

Jore Núñez de Cela Román, declara bajo su responsabilidad, que el Proyecto con título **Estimador de precios de renting de vehículos** presentado en la ETS de Ingeniería (ICAI) de la Universidad Pontificia Comillas en el curso académico 2024/25 es de su autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.



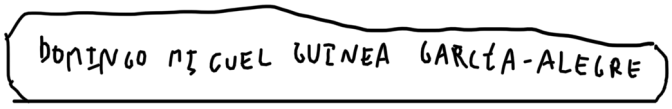
Fdo.: Jorge Núñez de Cela Román Fecha: 13 / 01 / 2025

Autoriza la entrega:

EL DIRECTOR DEL PROYECTO

Domingo Guinea García-Alegre

Fdo.: Domingo Guinea García-Alegre Fecha: 13 / 05 / 2021



V. B. del Coordinador de Proyectos

Nombre del Coordinador

Fdo.: ..... Fecha: ..... / ..... / .....

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Jorge Núñez de Cela Román **DECLARA** ser el titular de los derechos de propiedad intelectual de la obra: Estimador de precios de renting de vehículos, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, los derechos de digitalización, de archivo, de reproducción, de distribución y de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

(a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.

(b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.

(c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.

(d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.

(e) Asignar por defecto a estos trabajos una licencia Creative Commons.

(f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

(a) Que la Universidad identifique claramente su nombre como autor de la misma

(b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.

(c) Solicitar la retirada de la obra del repositorio por causa justificada.

(d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

(a) El autor se compromete a:

(b) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.

(c) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.

(d) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

(e) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

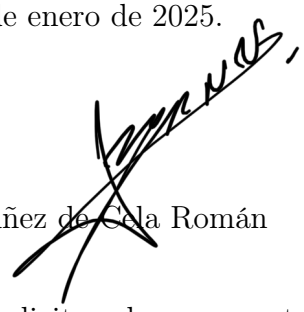
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 13 de enero de 2025.

ACEPTA



Fdo.: Jorge Núñez de Cela Román

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
(ICAI)

Máster en Big Data: Tecnología y Analítica Avanzada

Modelo para la predicción de precios de renting de vehículos.

Autor
Jorge Núñez de Cela Román

Dirigido por
Domingo Guinea García-Alegre

Madrid
Enero 2025

Resumen

Durante este trabajo se entrena e implementa un modelo de ML para la predicción del precio de renting de vehículos. Para ello, se comienza viendo el modelo anterior utilizado por la compañía. Después, una serie de modelos se entrenan con los datos disponibles para ver qué tipo de modelos son los mejores, concluyendo que los modelos más complejos son mejores pero se tiene la necesidad de reducir su varianza. Se construye un scrapeador web para recopilar más datos y se realiza una clusterización para reducir el número de variables. Tanto la sección de los modelos como la del clustering van acompañadas de una breve explicación teórica. Finalmente, se entrena un XGB, y se concluye que este es el mejor modelo de todos los probados. Diferentes tipos de posibles implementaciones del modelo en la BBDD de la empresa son discutidos.

Abstract

During this work, a machine learning model is trained and implemented to predict vehicle renting prices. The process begins by reviewing the previous model used by the company. Then, several models are trained with the available data to determine which types of models are the best, concluding that more complex models are superior but require variance reduction. A web scraper is constructed to gather more data, and a clusterization is performed to reduce the number of variables. Both the modeling and clustering sections are completed by brief theoretical explanations. Finally, an XGB model is trained, and it is concluded that this is the best model among those tested. Different possible implementations of the model in the company's database are discussed.

Índice general

Índice de figuras

Índice de cuadros

Acrónimos

TFM	Trabajo de Fin de Máster
ML	Machine Learning
MAE	Mean Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
CV	Cross Validation
MLP	Multilayer Perceptron
SVR	Support Vector Regression
LR	Linear Regression
DT	Decision Tree
XGB	Extreme Gradient Boosting
SaaS	Software as a Service
BBDD	Base de datos

Capítulo 1

Introducción

El objetivo de este TFM es mostrar un caso de uso el cual se resuelve utilizando modelos de Machine Learning. Se verán todas las partes del proceso, desde la obtención de datos hasta la puesta en producción. Este caso de uso ha sido realizado para la compañía Ozone Drive.

Ozone Drive es una start-up que se encarga de renovar y gestionar flotas de vehículos desde un punto de vista ecológico. Con este objetivo, se desarrollan algoritmos para decidir cuales son los mejores vehículos para cada cliente particular. Estas decisiones pueden estar motivadas por diversos temas, siendo los principales objetivos el económico y el ecológico. Los datos que se ingestan en estos algoritmos provienen de una base de datos central, la cual es imprescindible que se mantenga actualizada, ya que el mercado automovilístico varía mucho y las recomendaciones que demos a nuestros clientes tiene que ser actuales. Actualmente, la empresa se encuentra en una transición de vender productos, en forma de informes, a alquilar SaS (Software as a Service), lo cual también tendrá un impacto importante en nuestro proyecto.

La base de datos se alimenta de fuentes muy diversas, siendo la principal forma de recopilar datos para alimentar la base de datos el scrapeo web. Esto tiene una serie de inconvenientes, entre los cuales podemos destacar la dificultad técnica o la dependencia de una página web, con todo lo que esto conlleva, y que veremos más adelante en este trabajo. Por otro lado, hay datos de suma importancia para la empresa que no se encuentran disponibles en ninguna de las principales fuentes de información de la empresa y hay que intentar predecirlos con la información de la que disponemos. Aquí es donde cobra importancia este proyecto.

Un valor clave para la fiabilidad de las recomendaciones basadas en los algoritmos es la predicción de los precios de renting de los vehículos. Esto es importante por dos razones, la primera es enriquecer el informe final que se le pasa al cliente con esta información, la segunda es poder utilizar la estimación en sí para poder comparar el precio con el de otros vehículos y dar una recomendación más eco-

nómica. En el contexto de SaS, cobra especial importancia que un cliente pueda consultar esta información al momento para cualquier vehículo que quiera. Además, también se podría hacer un cálculo del total del precio de la nueva flota y compararlo con la antigua, siempre y cuando ambas sean de renting, de manera que el cliente sepa cual es el ahorro total, tanto anual como mensual.

Así, el objetivo de este TFM es desarrollar y poner en producción un modelo que sea capaz de predecir este valor de una manera relativamente fiable. A lo largo del TFM se verá una comparativa de todos los modelos probados, así como todo el camino hasta llegar hasta el modelo que mejores resultados obtenía y su correspondiente implementación en el código generado.

Capítulo 2

Situación Inicial. La necesidad del proyecto.

Esta primera sección está dedicada a dar contexto al trabajo. En ella se habla de la situación antes de comenzar el proyecto y de los conocimientos básicos que serán utilizados a lo largo del proyecto. Además se da más información acerca de la necesidad del proyecto.

2.1. Algunos Conceptos básicos

Hay muchos conceptos sobre machine learning y estadística que aparecen a lo largo del trabajo. Con el objetivo de esclarecer estos conceptos y no tener que definirlos cada vez que se mencionan, existe esta sección.

2.1.1. Métricas

Para saber cual es el mejor modelo, utilizaremos el error cuadrático medio (RMSE) y el error absoluto medio (MAE) como valores de referencia. El MAE simplemente es una media de de los valores absolutos del error, su fórmula es la siguiente:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.1}$$

Si elevamos los errores al cuadrado tenemos:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

Para la optimización, casi siempre es mejor utilizar la Raíz del Error Cuadrático Medio (Root Mean Squared Error, RMSE), cuya fórmula es la siguiente:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.3}$$

La principal diferencia es que el MSE es mucho más sensible a outliers. Generalmente, y en este caso concreto, nos interesa penalizar los outliers, pues sería un problema para la empresa equivocarse en la predicción de un valor más de 50 euros, aunque todos los demás valores estén muy cercanos al 0. Este error se verá reflejado en el MSE, pero no tanto en el MAE. El RMSE es simplemente el MSE escalado.

Finalmente, también nos interesará ver el coeficiente de determinación o R^2. Este coeficiente nos dice que cantidad de la variación puede ser explicada por el modelo, o dicho de otra manera, que capacidad tiene el modelo de replicar los resultados. Dicho de otra manera, tener un R^2 alto es necesario para tener un buen modelo, pero no suficiente, pues no implica que los errores vayan a estar dentro del rango que consideramos como aceptables. El cálculo de este coeficiente es el siguiente:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

Para más información sobre las medidas de rendimiento para objetivos continuos consultar [8, Sección 9.4.5].

Cuando hablemos de MAE o RMSE negativos, significará que estos valores se multiplican por -1 para convertirlos en métricas de maximización en lugar de minimización, lo que significa que valores más altos son mejores. Esto aparecerá en el uso de paquetes de python como scikit-learn, donde las funciones de puntuación por defecto están diseñadas para maximizar el rendimiento del modelo.

2.1.2. Grid Search y Cross Validation

Otros conceptos que también aparecerán a lo largo del trabajo son los de Grid Search, hiperparámetro y validación cruzada (CV). Grid Search es una técnica para optimizar hiperparámetros de un modelo de machine learning. Define un conjunto de valores posibles para cada hiperparámetro y evalúa todas las combinaciones para encontrar la mejor. Los hiperparámetros son parámetros que se establecen antes del entrenamiento y controlan el proceso de aprendizaje y la estructura del modelo, como la tasa de aprendizaje o la profundidad de un árbol de decisión. Varían dependiendo de cada modelo. La validación cruzada (CV o Cross Validation) es

una técnica para evaluar la capacidad de generalización de un modelo dividiendo los datos en varios subconjuntos, entrenando el modelo en algunos subconjuntos y validándolo en los restantes. Esto permite una evaluación más robusta y ayuda a evitar el sobreajuste, proporcionando una estimación más fiable del rendimiento del modelo. Para más información consultar *Over-Fitting and Model Tuning* [9, Chapter 4.2].

2.1.3. Sesgo y Varianza

Por último, cuando hablemos de sesgo nos referiremos al error que introduce un modelo cuando simplifica demasiado los datos y no captura las relaciones subyacentes, resultando en una baja precisión tanto en el entrenamiento como en el conjunto de prueba. La varianza, en cambio, es el error que introduce un modelo cuando es demasiado complejo y se ajusta demasiado a los datos de entrenamiento, capturando ruido y variaciones que no se generalizan bien a nuevos datos. En *The Variance-Bias Trade-off* [9, Chapter 5.2] se puede encontrar una explicación muy interesante entre la necesidad de encontrar una buena relación entre el sesgo y la varianza de un modelo, además de la siguiente relación:

$$E[MSE] = \sigma^2 + (\text{Sesgo del modelo})^2 + \text{Varianza del modelo}$$

donde estamos asumiendo que los residuos tienen una distribución teórica con media constante de cero y varianza de σ^2.

2.2. Situación Inicial

Antiguamente se utilizaba en la empresa una recta de regresión para predecir los datos de la cuota de renting de un vehículo. Este modelo esta implementado en excel y esta entrenado con unos 100 datos aproximadamente. Además, las marcas con las que está entrenado solo son Toyota y Audi, y los meses para los que está entrenado son 36 y 48 meses solamente. En la figura 2.1 se puede observar con más detalle.

Esta variación de la recta de regresión seguía esta ecuación:

$$Y = \beta_0 + \frac{X}{\beta_1} \tag{2.4}$$

donde:

- Y es la variable dependiente, en este caso la cuota estimada de renting.

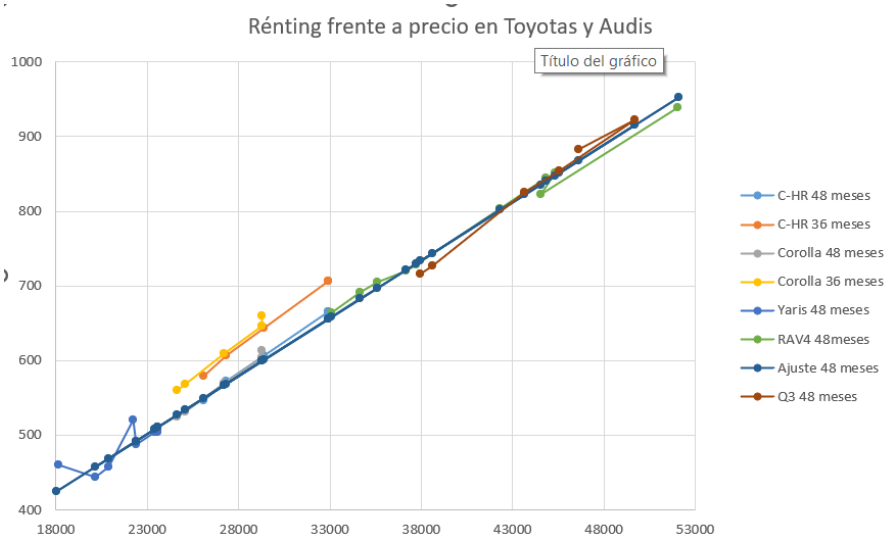


Figura 2.1: Modelo Inicial.

- X es la variable independiente. En este caso representa el precio con el impuesto de matriculación. Para calcularla se ha seguido la siguiente fórmula:

$$X = X_1 \dot{(} 1 + \frac{X_2}{100})$$

  donde

  - X_1 es el precio del vehículo sin impuestos
  - X_2 es el impuesto de matriculación en porcentaje

- β_0 es el intercepto (el valor de Y cuando todas las variables independientes son cero), en este caso valía 145,27

- β_1 es el coeficiente de regresión que representa la relación entre las variables independientes y la variable dependiente, en este caso valía 64,51

Tanto los valores de β_0 y β_1 se calcularon utilizando unas rectas de regresión aplicadas a valores antiguos, donde los valores que se enfrentaban para calcular la recta era precios de vehículos y cuota de renting.

Este modelo presentaba una serie de problemas bastante evidentes, lo que supuso necesariamente un cambio en la manera de predecir este dato. A continuación se enumeran los más relevantes:

- Capacidad de generalización: El modelo estaba entrenado con pocos datos y utilizando solo unas pocas marcas, lo que hacía que fuera difícil generalizar a datos muy distintos a los que habíamos utilizado.

- Meses de renting: Como se ha mencionado antes, los datos de los que se disponía era contratos de renting solo a 36 y 48 meses. Los casos más normales para las ofertas de renting suelen valorar entre 36 y 72 meses. Es decir, hay ofertas que simplemente no podremos predecir.
- Dificultad de calcular el error: No existía una manera clara de saber cuanto nos estábamos equivocando al calcular ofertas para coches de los que no disponíamos un precio de oferta de renting, por lo que el modelo solo estaba testeado y validado con un conjunto reducido de datos.
- Dificultad de implementación con el resto del core de la empresa: El objetivo final de este proyecto era tener un modelo que se pudiera implementar en el pipeline de creación de base de datos de la empresa y que se reentrenara periódicamente, cosa bastante difícil con este modelo.

Por estas razones, se decide cambiar este modelo por un modelo de machine learning que fuera capaz de corregir parcial o totalmente estos errores. Este es, en resumen, el objetivo de este TFM.

Capítulo 3

Entendiendo el problema. Una primera aproximación.

Uno de los principales problemas que tuvimos, es que se tuvo que empezar desde cero en el contexto de machine learning. No había ningún código existente que pudiéramos reutilizar ni ningún modelo que nos pudiera dar una idea de que tipo de modelos se adaptarían mejor a los datos que teníamos. Así, esta primera sección se dedica a entender el problema: En particular, entender los datos de los que disponemos actualmente y probar modelos para ver que nivel de complejidad es el adecuado a la hora de modelarlos.

3.1. Datos disponibles

Decidimos hacer una primera prueba de modelos con un conjunto de datos bastante limitados. El objetivo de esta prueba era ver qué modelos se adaptaban mejor a nuestros datos y cuáles iban a ser nuestros mayores problemas a solventar durante la realización del caso de uso.

El origen de los datos a utilizar es muy importante en este tipo de casos de uso. Hay que tener en cuenta que el precio de renting de los vehículos es algo subjetivo, y no depende solo del coche en sí, sino también de de la empresa que ofrezca el renting. Cada empresa tiene sus propias ofertas en función de sus relaciones contractuales con los diferentes fabricantes de vehículos. De esta forma, a nosotros sólo nos interesan las ofertas expedidas por cierta empresa, que es con lo que trabajamos en la empresa.

Esto quiere decir que la obtención de datos es difícil, no sirve scrapear cualquier sitio web o utilizar cualquier conjunto de datos de ofertas de renting. En particular, solo tenemos una fuente de datos que nos sirva para realizar este modelo, y la obtención de los datos es costosa debido a que el proceso es manual.

Así, conseguimos recopilar unas 750 observaciones con 4 variables, cuya distribución se puede observar en la figura 3.1. Cabe mencionar que los datos con los que se entreno la recta mencionada en la sección 1 están excluidos de estos datos. La razón es que son datos con años de antigüedad y el mercado automovilístico fluctúa mucho, así que no tenía sentido utilizarlos ya que iban a sesgar nuestro modelo. No observamos nada especialmente relevante en los datos, más allá de que existe un clara relación entre la cuota de renting y el precio del vehículo con cierto parecido a una regresión lineal.

Las variables con las que contamos son:

- cuota_sin_iva: Nuestro target. El precio de la cuota de renting del vehículo sin iva.
- plazo: El plazo de meses que dura el renting: Puede tomar 3 valores: 36, 48 y 60.
- precio_inicial: El precio de compra del vehículo sin extras
- Km_anuales: Los kilómetros máximos permitidos por año que puede recorrer el vehículo. Por encima de este valor se cobrará una penalización a los clientes.

Todas las variables predictoras son tratadas como categóricas, menos precio_inicial.

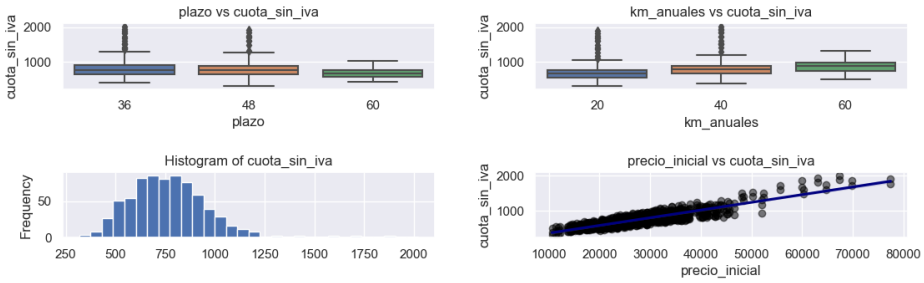


Figura 3.1: Visualización de los datos disponibles

Estos datos presentan dos problemas a simple vista. El primero es que no se están considerando todas las posibles combinaciones de km_anuales y plazo.

El segundo es que no se está considerando el factor marca del vehículo, lo cual intuitivamente parece que puede influir en la cuota, pero no tenemos ninguna evidencia para respaldar esta afirmación.

Se han probado varios modelos con el objetivo de ver cuáles son los que mejor se adaptan a los datos que tenemos. El entrenamiento se ha realizado con la librería scikit-learn de Python. Para la prueba se ha ido aumentando la complejidad de los modelos y se ha visto una tendencia clara, cuanta mayor complejidad menos error y más sobreaprendizaje, como era de esperar. Antes de exponer los resultados de cada método se dará una breve explicación teórica de cada uno de ellos para poner un poco de contexto.

3.2. Regresión lineal

Para la parte teórica de esta sección me baso en *Linear Regression and Its Cousins* [9, Chapter 6]. La regresión lineal es un método estadístico utilizado para modelar la relación entre una variable dependiente Y y una o más variables independientes X. A continuación veremos unas cuantas variantes.

3.2.1. Regresión lineal

En su forma más simple, la regresión lineal asume una relación lineal entre las variables, representada por la ecuación:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon$$

Donde:

- Y es la variable dependiente (la que se está tratando de predecir).
- $X_1, X_2, \ldots, X_n$ son las variables independientes (predictoras).
- β_0 es el intercepto (el valor medio de Y cuando todas las variables independientes son cero).
- $\beta_1, \beta_2, \ldots, \beta_n$ son los coeficientes de regresión que representan la relación entre las variables independientes y la variable dependiente.
- ε es el término de error, que representa la variabilidad no explicada por el modelo.

El objetivo de la regresión lineal es estimar los coeficientes $\beta_0, \beta_1, \ldots, \beta_n$ que minimicen la suma de los cuadrados de los residuos (diferencias entre los valores observados y los valores predichos por el modelo).

Como para todos los modelos de esta sección, se ha utilizado validación cruzada para estimar los errores de prueba. En la figura 3.2 podemos ver el resultado de este método aplicado a los datos que tenemos.

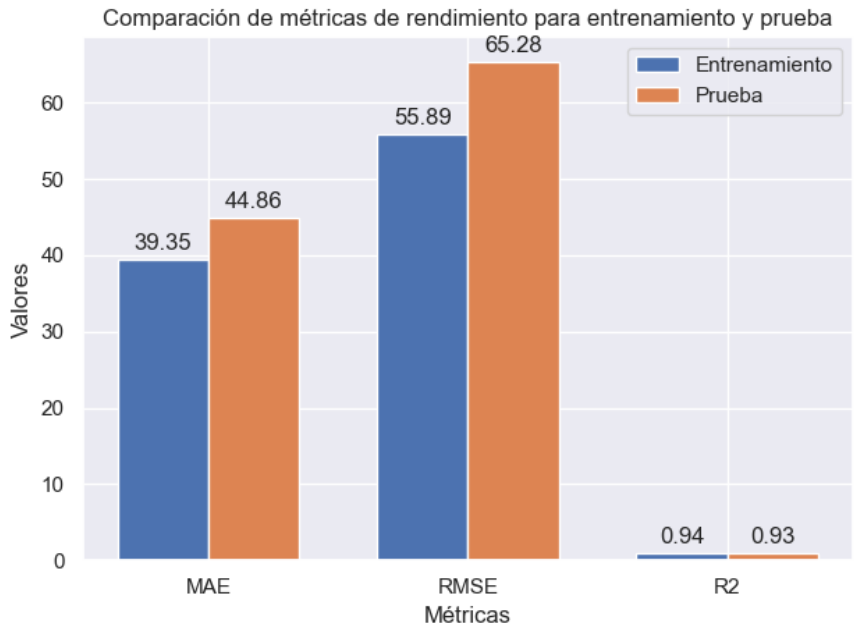


Figura 3.2: Métricas Regresión Lineal

3.2.2. Ridge Regresión

La regresión Ridge es una técnica de regresión que se utiliza para abordar los problemas de multicolinealidad y sobreajuste en modelos de regresión lineal. Se basa en la regresión lineal pero con una penalización adicional en los coeficientes de regresión para evitar que tomen valores extremadamente grandes. La regresión Ridge busca minimizar la siguiente función de coste:

$$\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Donde:

- y_i es el valor observado para la observación i.
- $\beta_0, \beta_1, \ldots, \beta_p$ son los coeficientes de regresión.
- x_{ij} es el valor de la variable predictora j para la observación i.

- p es el número de variables predictoras.
- λ es el parámetro de penalización, que controla la fuerza de la penalización sobre los coeficientes de regresión.

El término $\lambda \sum_{j=1}^{p} \beta_j^2$ es la penalización de Ridge, que se agrega a la función de coste de la regresión lineal ordinaria. Aumentar el valor de λ reduce la influencia de las variables predictoras menos importantes al contraer los coeficientes de regresión hacia cero.

La elección del parámetro de penalización λ suele influir en el modelo. Habitualmente, un λ demasiado grande produciría un sesgo excesivo, mientras que uno demasiado pequeño produciría varianza excesiva. Normalmente se elige mediante gridsearch y CV, en este caso no ha sido diferente, encontrando una minimización del error cuando $\lambda = 0{,}54$.

En la figura 3.3 podemos ver los resultados para este método. Es curioso observar como los errores son casi iguales que para la regresión lineal, lo cual indica que este método no ha supuesto apenas mejoría respecto a la regresión normal.

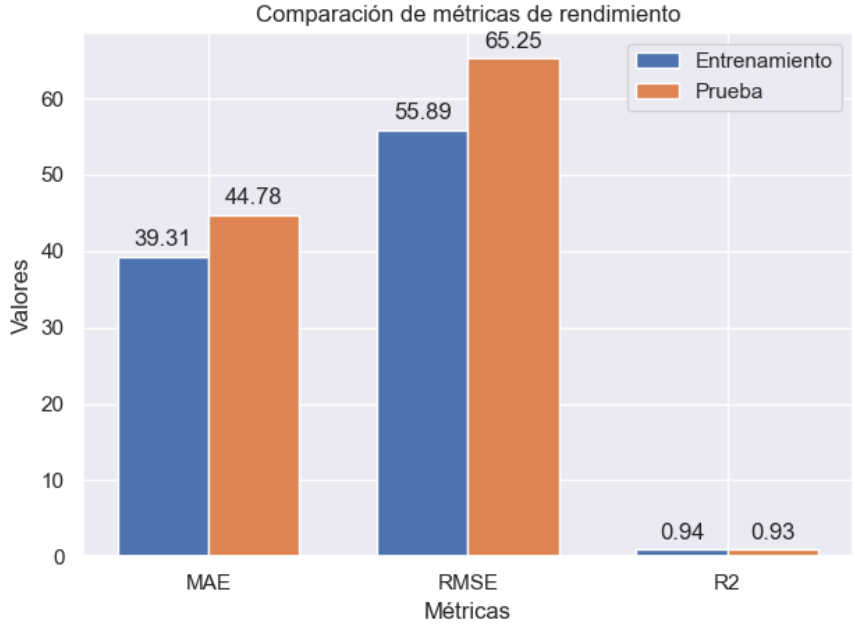


Figura 3.3: Métricas Regresión Ridge

3.2.3. Lasso Regresion

Otra variante de la regresión convencional es la regresuón Lasso (Least Absolute Shrinkage and Selection Operator), que es una técnica de regresión similar a la

regresión Ridge que se utiliza para abordar el problema de la multicolinealidad y realizar selección de variables al mismo tiempo. Al igual que la regresión Ridge, la regresión Lasso también agrega una penalización a los coeficientes de regresión, pero en lugar de usar la norma euclidiana al cuadrado como en Ridge, utiliza la norma $L1$, lo que puede llevar a que algunos coeficientes sean exactamente cero.

La función de coste de la regresión Lasso se define como:

$$\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}))^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

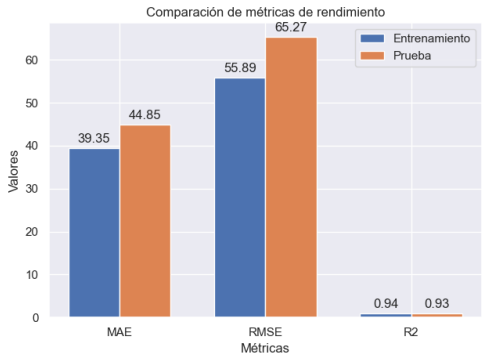
Donde:

- y_i es el valor observado para la observación i.
- $\beta_0, \beta_1, \ldots, \beta_p$ son los coeficientes de regresión.
- x_{ij} es el valor de la variable predictora j para la observación i.
- p es el número de variables predictoras.
- λ es el parámetro de penalización, que controla la fuerza de la penalización sobre los coeficientes de regresión.

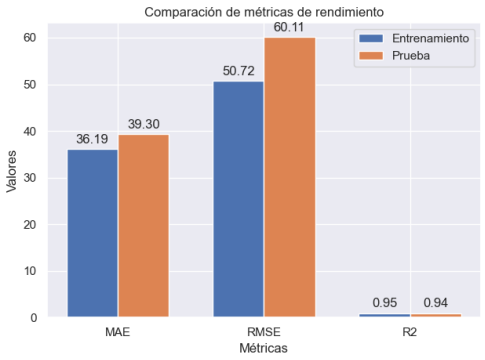
El término $\lambda \sum_{j=1}^{p} |\beta_j|$ es la penalización de Lasso. Al aumentar el valor de λ, algunos coeficientes de regresión se vuelven exactamente cero, lo que conduce a la selección de variables y al ajuste de un modelo más simple.

Básicamente, aquí estamos considerando todas las características más interesantes que nos da la regresión lineal: la reducción de parámetros y la penalización de estos. Para terminar de observar todos los escenarios posibles, también incluimos interacciones entre las variables, es decir, incluimos nuevas variables que vienen a ser productos entre las variables actuales. Al igual que antes, el parámetro lambda ha sido elegido mediante gridsearch y CV, obteniendo un valor de $\lambda = 0{,}01$.

De los resultados observados en la figura 3.4 podemos sacar conclusiones interesantes. Obviamente al hacer eliminación de variables no hemos eliminado ninguna, ya que tenemos muy pocas y todas son importantes. Sin embargo, al incluir interacciones entre las mismas si que hemos mejorado algo el modelo. De todas formas, esto parece que es lo mejor que puede hacer una regresión, quizás un modelo demasiado simple, ya que no tenemos mucha varianza viendo las diferencias entre training y test, pero el sesgo es evidente.



(a) Sin interacciones



(b) Con interacciones

Figura 3.4: Regresión Lasso

Existen muchas más variaciones de la regresión lineal, como la elasticnet o la PCR. Sin embargo, no tenía mucho sentido probar más por lo explicado anteriormente. De todas formas, para más información se puede consultar [9, Chapter 6].

3.3. Modelos no lineales

Como ya hemos mencionado, los modelos no lineales son modelos más complejos que los modelos lineales. Se pierde explicabilidad pero normalmente ofrecen mejores resultados. Como en nuestro caso la explicabilidad no es importante, no tenemos ningún problema en utilizarlos. Sin embargo, hay que tener cuidado con el sobreaprendizaje, ya que este tipo de modelos suele tener más varianza.

3.3.1. k-Nearest Neighbours (kNN)

El k-Nearest Neighbours (kNN) Regressor es un algoritmo de aprendizaje supervisado utilizado para problemas de regresión. A diferencia de los modelos paramétricos como la regresión lineal, kNN no asume una estructura específica para los datos, sino que realiza predicciones basadas en la similitud entre las características de los puntos de datos.

Para realizar una predicción utilizando kNN Regressor, se siguen los siguientes pasos:

1. Calcular la distancia entre el punto de consulta y todos los puntos de datos en el conjunto de entrenamiento utilizando una métrica de distancia, como la euclidiana.

2. Seleccionar los k puntos más cercanos al punto de consulta.

3. Calcular la predicción promediando los valores de la variable objetivo correspondientes a los k puntos más cercanos.

Para una explicación más exhaustiva consultar *K-Nearest Neighbors* [9, Section 7.4].

La principal decisión que se debe tomar al usar kNN Regressor es la elección del valor de k. Un valor más bajo de k puede llevar a un modelo más flexible y propenso a sobreajustar los datos, mientras que un valor más alto de k puede dar como resultado un modelo menos agresivo y menos sensible al ruido en los datos, lo que está estrictamente relacionado con un mayor sesgo al no captar tan bien fenómenos locales. En nuestro caso el valor de k ha sido elegido mediante gridsearch, obtiendo que el mejor valor posible es 7.

Es importante tener en cuenta que kNN Regressor no proporciona coeficientes interpretables como en la regresión lineal, ya que no hay una función explícita que describa la relación entre las características y la variable objetivo. En cambio, realiza predicciones basadas en la similitud con los puntos de datos cercanos en el espacio de características. En otras palabras, perdemos explicabilidad para ganar precisión.

Los resultados visibles en la figura 3.5 son curiosas, ya que en MAE tenemos más error en training que en test, esto se debe normalmente a que el conjunto de datos de entrenamiento es bastante escaso, así que no son unos scores muy fiables. De todas formas si que podemos afirmar que este modelo tiene menos sesgo que los anteriores.

3.3.2. Decision Tree - Regression

Según *Regression Trees and Rule-Based Models* [9, Chapter 8], el árbol de decisión para regresión es un algoritmo de aprendizaje supervisado utilizado para resolver problemas de regresión.

El proceso de construcción de un árbol de decisión para regresión implica dividir el espacio de características en regiones rectangulares, y en cada región, el modelo predice un valor específico. Esto se logra mediante la división recursiva del espacio de características en subconjuntos más pequeños, utilizando características y umbrales de división que maximizan la reducción en la varianza de los valores de la variable objetivo.

La predicción de un árbol de decisión para regresión se realiza utilizando la media de los valores de la variable objetivo en la hoja correspondiente a la región en la que cae la observación.

El árbol de decisión para regresión puede ser representado matemáticamente como:

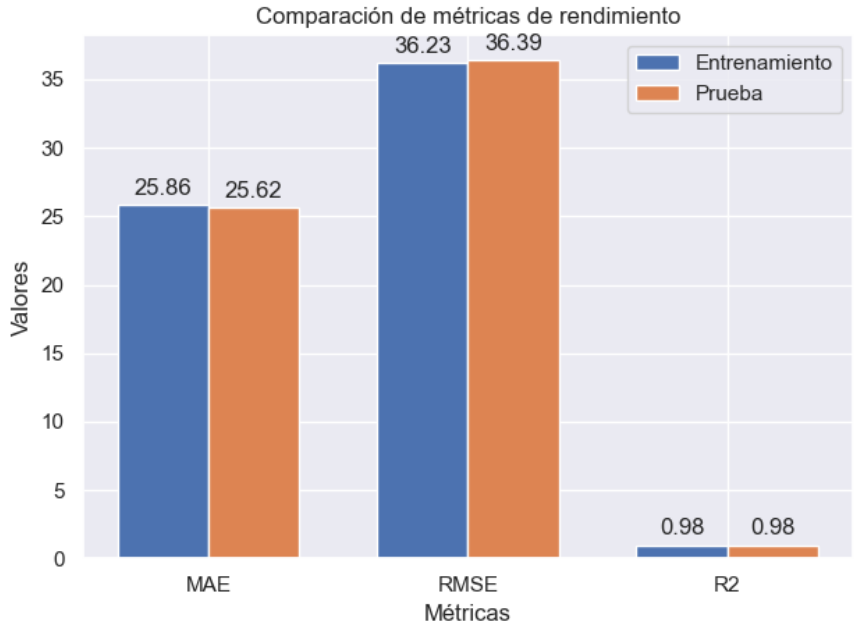


Figura 3.5: Métricas kNN

$$\hat{y}_i = \sum_{j=1}^{J} c_j \cdot \mathbb{I}(x_i \in R_j)$$

Donde:

- $\hat{y}_i$ es la predicción para la observación i.
- c_j es el valor promedio de la variable objetivo en la región R_j.
- $\mathbb{I}(x_i \in R_j)$ es una función indicadora que toma el valor de 1 si la observación i cae en la región R_j, y 0 en caso contrario.
- J es el número total de regiones en el árbol.

El árbol de decisión tiene muchos más parámetros para ajustar que el resto de modelos vistos hasta ahora, para más información ver [11]. Como cuantos más parámetros más tiempo de entrenamiento, hay que elegir bien los parámetros que nos interesan. Nosotros obtamos por un enfoque bastante estándar, que es fijar min_sample_split (número mínimo de muestras necesarias para dividir un nodo) y min_sample_leaf(número mínimo de muestras que debe tener un nodo por hoja) a 5 cada uno, y solo realizar Grid Search sobre min_impurity_decrese, que es el

mínimo valor de impureza que tiene que tener un nodo para que se haga un split, minimizando el error cuando toma el valor 0.25.

El árbol de decisión es fácil de entender e interpretar, y puede capturar relaciones no lineales y complejas entre las características y la variable objetivo. Sin embargo, tiende a sobreajustarse a los datos de entrenamiento, especialmente en conjuntos de datos de alta dimensionalidad o con ruido. De hecho, es lo que pasa en este caso. Basta fijarse en la figura 3.6 para ver que no es un buen modelo.

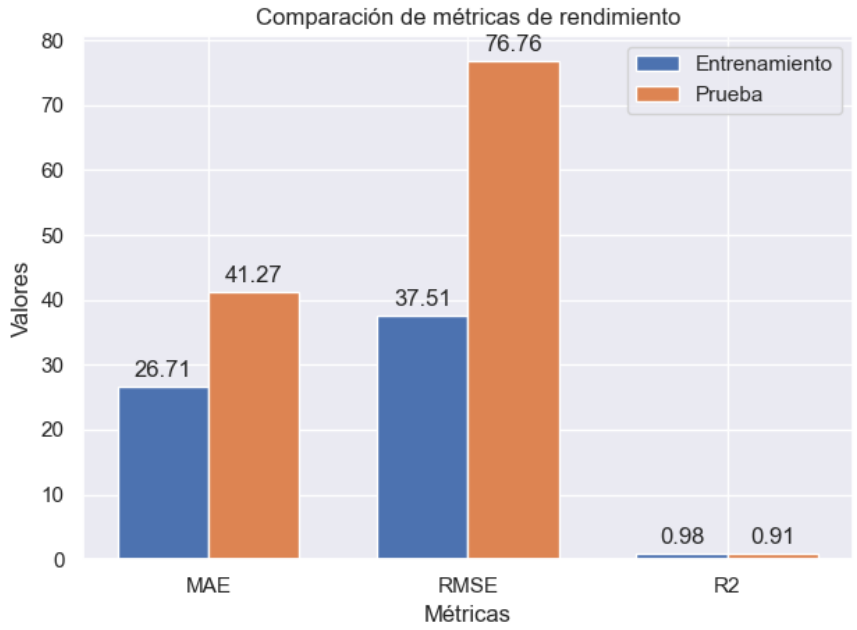


Figura 3.6: Métricas decision tree

3.3.3. Support Vector Regression

Según *Support Vector Machines* [9, Section 7.4], el SVR es un algoritmo de regresión basado en máquinas de vectores de soporte (SVM) que se utiliza para problemas de regresión.

El objetivo de SVR es encontrar una función $f(x)$ que tenga un margen máximo de tolerancia (ϵ) alrededor de los puntos de datos de entrenamiento. Esta función se define como:

$$f(x) = \langle w, x \rangle + b$$

Donde:

- w son los pesos asignados a las características.

- b es el sesgo.
- $\langle \cdot, \cdot \rangle$ denota el producto punto entre w y x.

En lugar de trabajar directamente en el espacio original de características, se asocian los datos de entrada x a un espacio de características de mayor dimensión mediante una función $\phi(x)$. Sin embargo, en lugar de computar $\phi(x)$ explícitamente, se utiliza una función kernel $K(x, x')$ que calcula el producto interno en el espacio de características mapeado:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

Las bases, o funciones de mapeo, $\phi(x)$ pueden ser cualquier conjunto de funciones que transformen los datos de entrada a un espacio de características de mayor dimensión, como polinomios. Para encontrar la función $f(x)$ que maximiza el margen de tolerancia, SVR resuelve el siguiente problema de optimización:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*)$$

Sujeto a las restricciones:

$$\begin{gathered} y_i - \langle w, x_i \rangle - b \leq \epsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{gathered}$$

Donde:

- C es un parámetro de regularización que controla el equilibrio entre la maximización del margen y la minimización del error.
- ξ_i y ξ_i^* son variables de holgura que permiten que algunos puntos de datos estén dentro del margen de tolerancia.

Por defecto, la librería scikit-learn utiliza un kernel gaussiano (RBF), pero puede admitir como parámetro la función kernel que utilizar. El parámetro C ha sido obtenido mediante gridsearch, consiguiendo un valor óptimo de 1000. El parámetro γ, referente a la anchura de las bases gaussianas también ha sido obtenido mediante gridsearch, tomando el valor de 0.1. Usar un kernel gaussiano es lo estándar y es el que mejores resultados suele dar. Para más información sobre las posibles bases a utilizar y parámetros disponibles consultar [14].

En la figura 3.7 podemos observar los resultados del SVR aplicado a nuestro problema, similares a modelos anteriores.

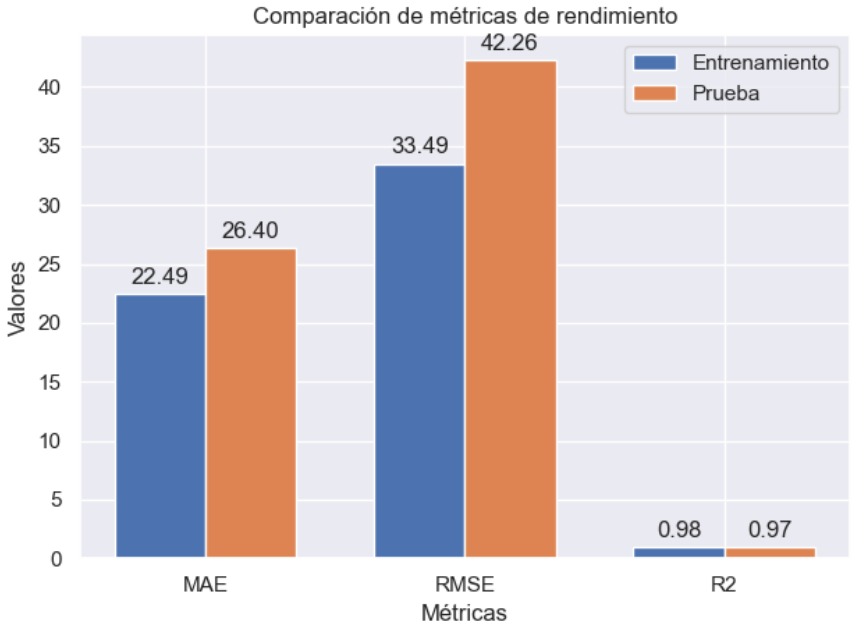


Figura 3.7: Métricas SVR

3.3.4. Multi-Layer Perceptron (MLP)

Finalmente, la Multi-Layer Perceptron (MLP) es una arquitectura de red neuronal artificial utilizada para resolver problemas de regresión y clasificación. Consiste en múltiples capas de neuronas, cada una conectada con las neuronas de las capas adyacentes. En su forma más simple, un MLP consta de una capa de entrada, una o más capas ocultas y una capa de salida.

Cada neurona en una capa oculta y de salida realiza una combinación lineal de las salidas de las neuronas de la capa anterior, seguida de una función de activación no lineal. Las funciones de activación introducen no linealidades en la red, permitiendo que los MLP capturen relaciones complejas en los datos.

La salida de una neurona en la capa l está dada por:

$$a^{(l)} = \sigma\left(\sum_{i=1}^{n^{(l-1)}} w_i^{(l)} a_i^{(l-1)} + b^{(l)}\right)$$

Donde:

- $a^{(l)}$ es la salida de la neurona en la capa l.
- $n^{(l-1)}$ es el número de neuronas en la capa anterior.

- $w_i^{(l)}$ son los pesos asociados con las conexiones entre la neurona actual y las neuronas en la capa anterior.

- $a_i^{(l-1)}$ son las salidas de las neuronas en la capa anterior.

- $b^{(l)}$ es el sesgo de la neurona en la capa l.

- σ es la función de activación aplicada a la suma ponderada de las entradas.

El entrenamiento de un MLP implica ajustar los pesos y sesgos de las conexiones entre las neuronas para minimizar una función de coste. Normalmente se suele tardar mucho para entrenar una MLP, pero los resultados suelen ser bastante buenos, aunque a veces, como se puede ver en la figura 3.8, la varianza es elevada. En este caso se han tomando dos decisiones, el número de capas ocultas, que en este caso son 13, y el valor del parámetro α, referente a la función de regularización, que en este caso es 0.001. Como siempre, estas decisiones se han tomado utilizando grid search y quedándonos con los valores que minimizan el error de validación cruzada. Normalmente estos son las hiperparámetros más importantes, pero la MLP puede admitir muchos más parámetros. Hay que elegir muy bien sobre que hiperparámetros realizar Grid Search porque el tiempo de entramiento puede ser muy elevado. Para ver todos los posibles parámetros consultar [13].

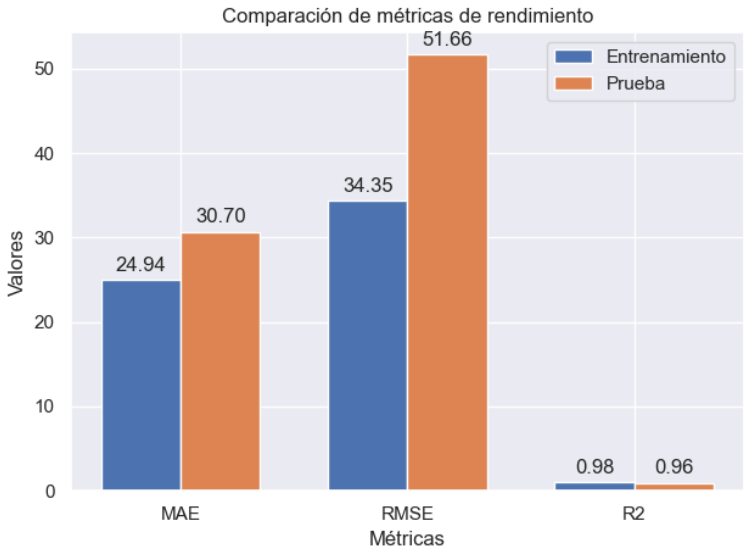


Figura 3.8: Métricas MLP

3.4. Conclusiones sobre los modelos probados

Para comparar los modelos de una manera más visual de la vista anteriormente se ha utilizado también la estrategia de cross-validation. Para cada modelo se calcula la puntuación de error cuadrático medio negativo (Neg RMSE), error absoluto medio negativo (Neg MAE) y coeficiente de correlación (R^2) utilizando validación cruzada con 10 particiones. La función de python *cross_val_score* [10] evalúa el rendimiento del modelo dividiendo el conjunto de datos de entrenamiento en 10 subconjuntos, entrenando el modelo en 9 subconjuntos y validándolo en el subconjunto restante, repitiendo este proceso 10 veces. Cuando se tienen los 10 resultados, se hace la media. Los resultados de estas evaluaciones se guardan y podemos ver la distribución de las métricas en las figuras 3.9, 3.10 y 3.11.

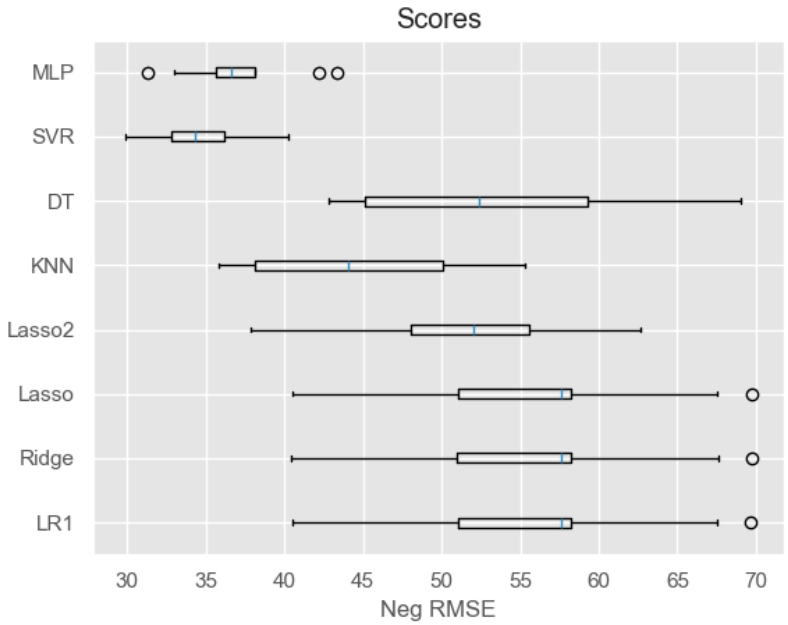


Figura 3.9: Comparación RMSE

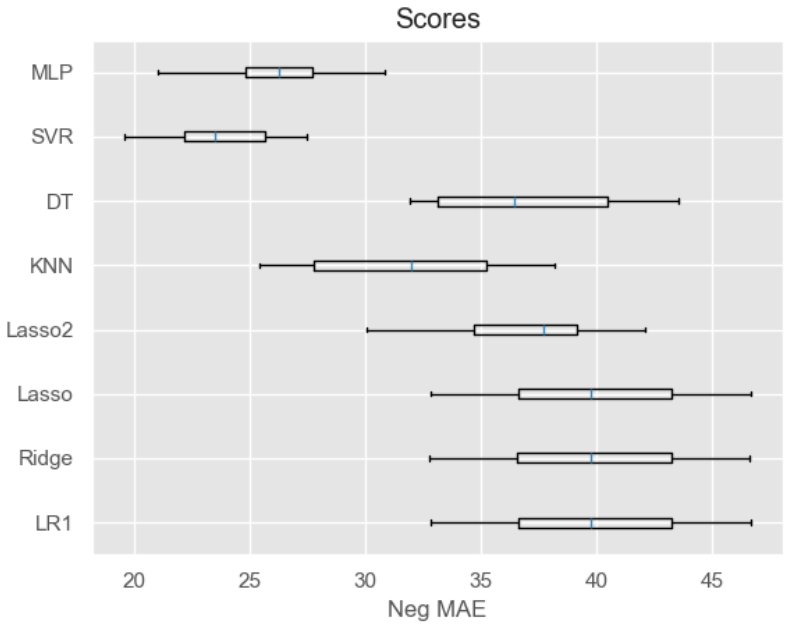


Figura 3.10: Comparación MAE

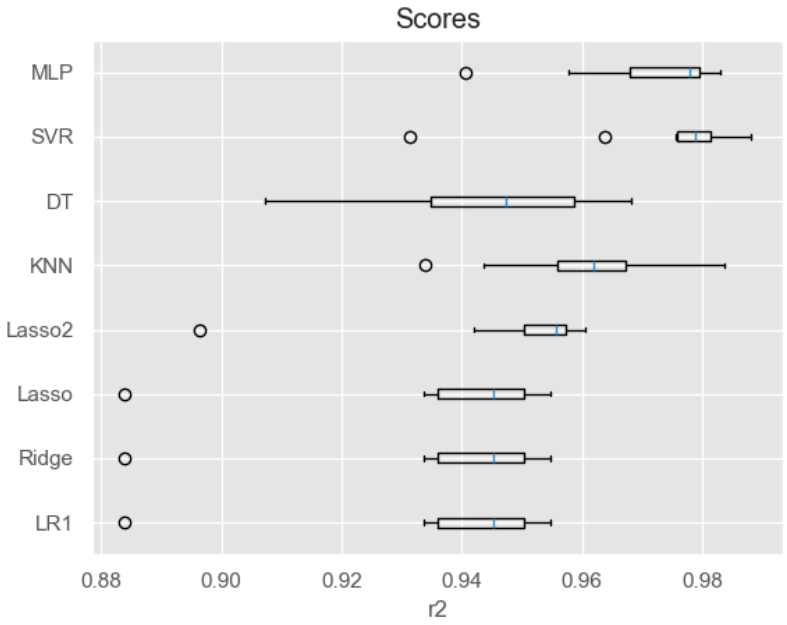


Figura 3.11: Comparación R2

Observamos un par de conclusiones bastante llamativas. La primera es que los modelos que mejor predicen los datos son la MLP y el SVR, una conclusión que nos hubiera costado remarcar solo fijándonos en la sección anterior. Por eso es mucho más interesante calcular los errores de esta forma. Sin embargo, podemos ver como los errores son más altos de lo que esperábamos y el valor de R^2 es muy cercano a uno en todos los casos, es decir, estamos consiguiendo explicar gran parte de la variabilidad del modelo, pero seguimos teniendo bastante sesgo (recordemos que un valor alto del coeficiente de correlación es necesario pero no suficiente para tener un buen modelo). Dicho de otra forma, es necesario obtener mejores datos para conseguir un modelo que reduzca el sesgo, y entonces probar cuales son los modelos que consiguen reducir más la varianza de las predicciones.

Otra cosa que llama la atención, es que los mejores modelos son los modelos no lineales, es decir, los modelos más complejos. Cuando digo que llama la atención me refiero a que con el estudio anterior de los datos no daba la sensación de que los datos tuvieran relaciones complejas, y podría parecer que una regresión iba a ser el mejor modelo. Sin embargo, hemos comprobado que al contrario, cuanto más complejo es el modelo mejor, eso sí, con algo de sobreaprendizaje. Además, estamos sacrificando explicabilidad y tiempo, ya que la diferencia de tiempo de entrenamiento es muy notable, sin embargo, por la naturaleza del proyecto, ninguna de estas dos cosas nos importa demasiado. Finalmente, como posteriormente añadiremos más variables y categorías, es de esperar que los modelos complejos aumenten su mejora respecto a los lineales.

Esta última afirmación se basa en que las variables y categorías que se añadirán tendrán relaciones complejas con las salidas, y por esto es de esperar que los modelos complejos nos den un mejor resultados.

Capítulo 4

Mejora de los datos del Modelo

Como vimos en el capítulo anterior para mejorar el modelo que probamos necesitábamos más datos. Hay dos formas de conseguir más datos: podemos conseguir un mayor número de observaciones para alimentar el modelo y podemos intentar añadir variables predictoras al modelo. Ninguna de las dos es normalmente fácil, ya que los datos de los que dispones normalmente tienen una razón de ser, ya sea que por temas de privacidad o dificultad de obtención no se puedan utilizar más datos o simplemente no existen más datos que nos sean útiles. Aquí es interesante recordar que las ofertas depende normalmente de las empresas que las emita, así que no vale coger cualquier conjunto de datos o de cualquier fecha que podemos encontrar. Por suerte, nosotros disponíamos de un cotizador online de la empresa que nos interesaba. De esta forma, nuestra siguiente parte del proyecto fue realizar un scrapeador que nos permitiera obtener datos de ofertas de vehículos actualizadas.

4.1. Scrapeador mediante python

Sin embargo, no todo es tan bueno como parece. La página web a scrapear en sí requiería un alto nivel de interacción con un humano, así que ya está claro que necesitamos utilizar la librería *selenium* de python. *Selenium* es una popular biblioteca de python para la automatización de navegadores web. Es utilizada principalmente para pruebas automatizadas de aplicaciones web y para tareas de scraping web [4].

Así, el siguiente objetivo es scrapear datos para aumentar nuestro número de observaciones. Para ello tenemos que decidir una forma de salida de los datos para poder transformarla con facilidad a la tabla de entrada de nuestro modelo, pero de forma que también sea compatible con el scrapeador y sea lo más rápida posible. Por tanto, decidimos utilizar la librería *jsonlines* de python, que sirve para generar archivos con la extensión .jsonl. La principal diferencia es que los archivos

.jsonl consisten en múltiples líneas, cada una de las cuales es un objeto JSON independiente. Esto permite procesar los datos línea por línea sin necesidad de cargar todo el archivo en memoria, lo cual es muy útil para trabajar con grandes volúmenes de datos, que es justo lo que necesitamos [15].

Siguiendo con lo anterior, desarrollamos un scrapeador que en cada interacción devuelve un diccionario donde las claves son: marca, modelo, versión, meses, km_anuales, precio_sin_iva y cuota sin iva. El problema de esto es que por los puntos de espera y las interacciones con la web necesarias, cada vehículo tarda en scrapearse más de un minuto, y si quisiéramos obtener información de todos los vehículos y versiones de cada marca tendríamos que tener el scrapeador 6 meses funcionando aproximadamente. Dado que no disponíamos de ese tiempo ni de los recursos para tener tanto tiempo en funcionamiento, se decidió coger solamente datos de un par de vehículos con todas sus versiones por marca, para así tener datos de todas las marcas, que era un problema que teníamos antes. El criterio para que coches elegir no fue otro que el orden en el que aparecían en la página web.

También hay que mencionar que para cada vehículo y versión se recopila un dato de cuota por cada combinación posible de kilómetros máximos permitidos y duración del renting en meses, teniendo en cuenta que las duraciones posibles son: 36, 48, 60 y 72 meses y los kilómetros anuales máximos permitidos varían entre 5000 y 75000 kilómetros en intervalos de 5000 en 5000, tenemos para cada coche y versión únicos un total de 60 combinaciones posibles.

En definitiva, se hizo un scraper utilizando la librería *selenium* de python. Una vez hecho el scraper, se decidió dockerizar el código con el objetivo de automatizar la ejecución del scraper cada cierto tiempo. El uso de docker supuso bastantes dificultades, ya que la librería *selenium* necesita de una "pantalla"para poder interaccionar con el navegador. La solución final consistió en modificar el dockerfile de manera que instalará en el contedor paquetes de linux capaces de simular el uso de un navegador en el contenedor de docker.

4.2. Clustering

Antes no podíamos usar la marca de los coches como variable predictora, pero ahora que tenemos datos de todas las marcas podemos utilizar esta variable. Sin embargo, pasarla como variable categórica tal cual al modelo no parece la mejor de las ideas. Primero, teninedo en cuenta que existen un gran número de marcas de coches que se comercializan en España, aproximadamente 40, y por como funciona el one hot encoding, estaríamos creando aproximadamente 40 nuevas variables binarias para el modelo. Por otro lado, el mercado en España cambia más de lo que alguien podría pensar, hay marcas que se descatalogan y marcas nuevas

que irrumpen en el mercado, sobre todo con los nuevos coches eléctricos, lo que supondría la necesidad de volver a obtener datos y reentrenar el modelo en caso de la aparición de una nueva marca. La solución que planteamos aquí es la de clusterizar las marcas de coches en función de como se deprecian a lo largo de los años.

Para hacer el clustering no hay una forma clara de hacerlo, así que vamos a intentar identificar cada marca con un punto tridimensional. La idea es ver si las cuotas de renting se parecen entre las marcas a precios bajos, medios y altos de los vehículos. Se cogen tres rangos de precios, porque aunque el comportamiento en dos rangos define la recta de regresión, es posible que dos vehículos tengan cuotas distintas a precios bajos y altos, y que a precios medios sean similares porque se crucen sus rectas. Con tres puntos ya podemos capturar el efecto de los cruces en una de las zonas o entre dos zonas.

Siguiendo con esta idea, para cada marca se predicen 3 puntos, a los que denotamos p_20, p_50 y p_80. La predicción de estos puntos es sencilla, se ajusta una curva por marca, cogiendo como variable x el precio del vehículo y como variable y la cuota del vehículo. En la figura 4.1 se puede ver la curva trazada para cada marca. Una vez que tenemos una curva para cada marca, calculamos la predicción usando la recta para los percentiles 20, 50 y 80 sobre el total de vehículos de la variable precio, obteniendo el punto $p = (p\_20, p\_50, p\_80) \in \mathbb{R}^3$. Los puntos elegidos se pueden ver en la figura 4.2. De esta forma, tenemos un punto en $\mathbb{R}^3$ para cada marca y solo queda agrupar estos puntos.

El algoritmo de clustering K-means es un método iterativo que particiona un conjunto de datos en k clusters. Según [8, Sección 10.3], el algoritmo en nuestro caso particular sería el siguiente:

1. Dado un conjunto de datos $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, donde cada $\mathbf{x}_i \in \mathbb{R}^3$ es un vector de características, y un número de clusters k, seleccionar aleatoriamente k puntos en el espacio $\mathbb{R}^3$ como los centros iniciales de los clusters: $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$.

2. Asignar cada punto $\mathbf{x}_i$ al cluster cuyo centro esté más cercano, utilizando la distancia euclidiana. Esto se puede expresar como:

$$C_i = \{\mathbf{x}_j : \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2 \leq \|\mathbf{x}_j - \boldsymbol{\mu}_l\|^2, \, \forall \, l = 1, 2, \ldots, k\}$$

   donde $\| \cdot \|^2$ representa la distancia euclidiana al cuadrado.

3. Actualizar los centros de los clusters calculando la media de los puntos asignados a cada cluster:

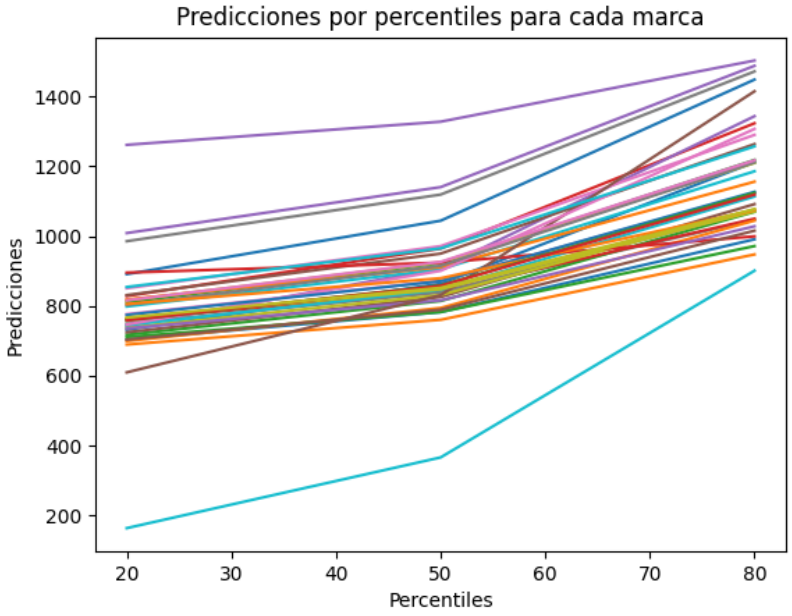


Figura 4.1: Rectas trazadas por marca para obtener los puntos

$$\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$$

donde $|C_i|$ es el número de puntos en el cluster C_i.

4. Repetir los pasos de asignación de clusters y actualización de centros hasta que los centros de los clusters no cambien significativamente o hasta que se alcance un número máximo de iteraciones.

En resumen, el objetivo del algoritmo K-means es minimizar la suma de las distancias al cuadrado entre cada punto y el centro de su cluster correspondiente. Esta función objetivo se puede expresar como:

$$J = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

Ahora, para elegir el número óptimo de cluster se suele seguir siempre la misma estrategia, que es la que hacemos nosotros. Ejecutar en bucle el siguiente algoritmo aumentando el valor de k, en nuestro caso empezamos con $k = 2$ y terminamos con $k = 10$. El método de clustering *kmeans* puede aceptar muchos parámetros de

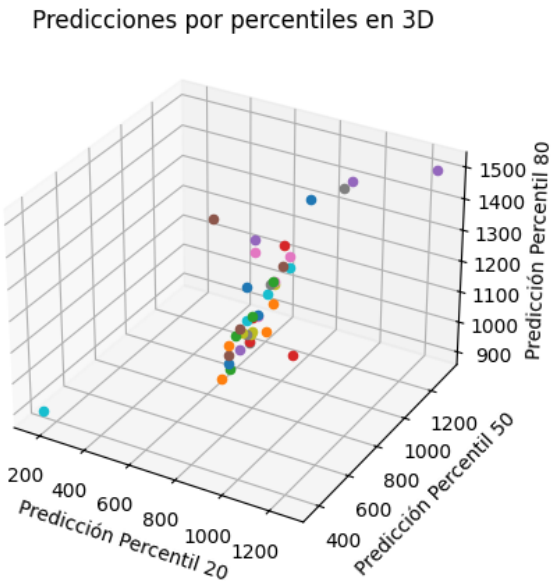


Figura 4.2: Puntos obtenidos para clusterizar

entrada, aunque ninguno es obligatorio [12]. En nuestro caso solo le pasamos uno, que como ya hemos mencionado es el número de clusters sobre el que va iterando el bucle. Una vez hecho esto analizamos dos métricas para elegir el número óptimo de clusters: el método del codo y el coeficiente de silhouette.

4.2.1. Método del Codo

El método del codo es una técnica utilizada para determinar el número óptimo de clusters en un conjunto de datos al realizar un análisis de clustering. Este método se basa en la suma de los errores cuadráticos SSE dentro de cada cluster. El procedimiento es el siguiente:

1. Realizar el clustering de los datos para diferentes valores de k.
2. Calcular el SSE para cada valor de k. El SSE es la suma de las distancias al cuadrado de cada punto de datos a su centroide más cercano.
3. Representar el SSE en función de k. La gráfica suele tener una forma de codo.
4. Determinar el punto donde la tasa de disminución de SSE se reduce significativamente, es decir, donde se forma un "codo" en la gráfica. Este punto

indica el número óptimo de clusters.

La idea principal es que agregar más clusters después del punto del codo no proporciona una mejora sustancial en la reducción del SSE, lo que sugiere que el número óptimo de clusters se encuentra en ese punto. Para más información consultar *Determining the Number of Clusters* [7, Section 10.6.2].

4.2.2. Coeficiente de Silhouette

El coeficiente de silhouette es una medida utilizada para evaluar la calidad de un clustering. Este coeficiente calcula qué tan similar es un punto a los puntos de su propio cluster en comparación con los puntos de otros clusters. Para cada punto i en el conjunto de datos, el coeficiente de silhouette $s(i)$ se define como:

$$s(i) = \frac{b(i) - a(i)}{\text{máx}(a(i), b(i))}$$

donde:

- $a(i)$ es la distancia promedio entre i y todos los demás puntos en el mismo cluster.
- $b(i)$ es la distancia promedio entre i y todos los puntos en el cluster más cercano del cual i no es miembro.

El coeficiente de silhouette toma valores en el rango de -1 a 1:

- Un valor cercano a 1 indica que el punto está bien agrupado.
- Un valor cercano a 0 indica que el punto está en el límite de dos clusters.
- Un valor negativo indica que el punto podría estar mal agrupado.

El promedio de los coeficientes de silhouette para todos los puntos proporciona una medida de cuán bien están separados los clusters y cuán compactos son. *Measuring Clustering Quality* [7, Section 10.6.3].

4.2.3. Resultados Finales de cluster

En la figura 4.3 podemos ver los resultados de silhouette y del método del codo. Ya hemos explicado antes como se interpreta cada uno de los métodos, ahora necesitamos elegir el número óptimo de grupos. Observamos que lo mejor sería formar 3 grupos diferentes. Sin embargo, al ser un número reducido de grupos, podríamos estar perdiendo información. De hecho, si nos fijamos en la figura 4.2

podemos observar como los 3 grupos diferentes seguramente serían considerando los 2 puntos más alejados como puntos individuales, y el resto de puntos todos en un grupo, con lo cual estamos perdiendo mucha información. Por ello, elegir 5 clusters parece una opción mucho mejor, siendo este valor de k el segundo mejor después de 3. Los resultados se pueden ver en la figura 4.4. Se ha aplanado la figura a dos dimensiones para una mejor visualización.

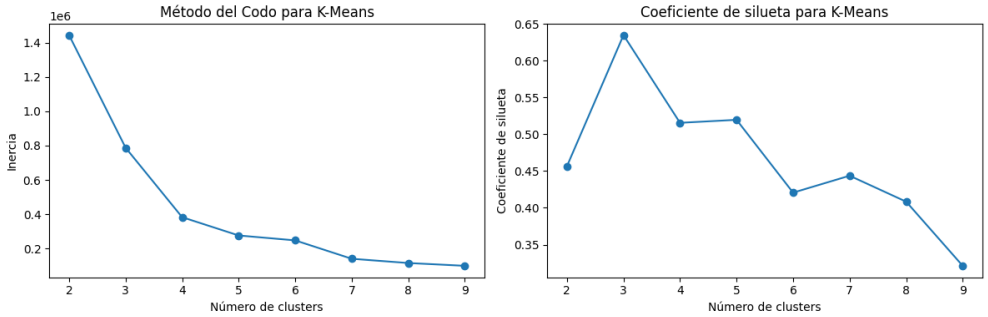


Figura 4.3: Métricas para decidir el número de clusters

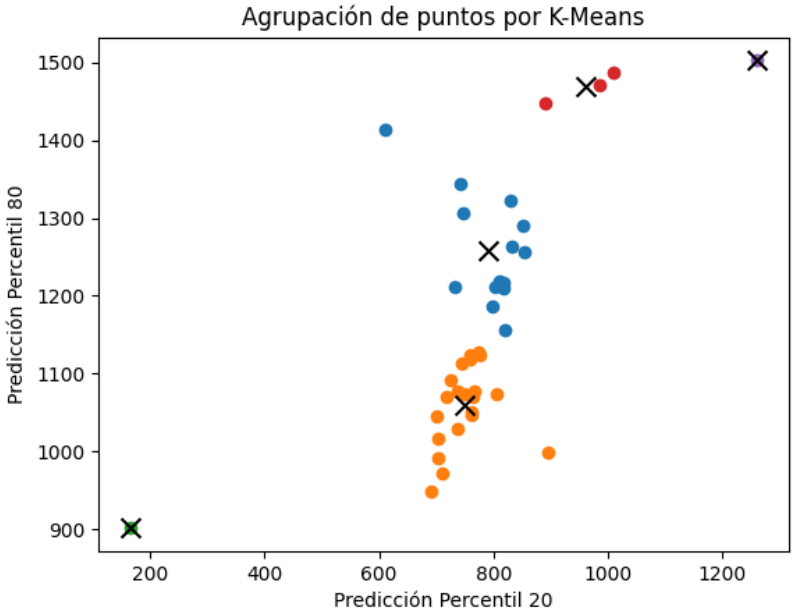


Figura 4.4: Cluster final en dos dimensiones

4.2.4. Medición del impacto de la nueva variable

Ahora toca comprobar que lo que hemos hecho sirve de algo, es decir, tenemos que ver que la variable cluster mejora el modelo y también tendremos que ver que no estamos perdiendo información respecto de la variable marca. Para ello vamos a entrenar dos nuevos modelos, una red neuronal y un SVR, ya que como vimos en la sección anterior eran los que mejor accuracy conseguían. Tampoco hemos implementado muchos cambios respecto a los modelos de la última sección, simplemente hemos añadido la variable cluster. Es importante mencionar aquí que no se están añadiendo los nuevos datos scrapeados para entrenar el modelo, se han utilizado para clusterizar los datos y se ha añadido la variable cluster a los datos utilizados en la sección anterior. Esto es importante tenerlo en cuenta porque lo que estamos intentando comprobar es que la clusterización mejora los modelos anteriores. Se pueden ver los resultados en las siguientes figuras:

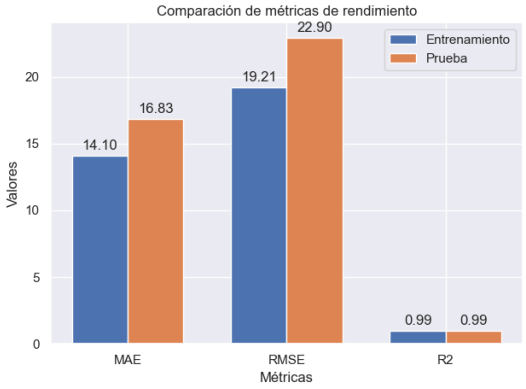


Figura 4.5: Métricas de rendimiento de MLP con los nuevos datos y variable cluster

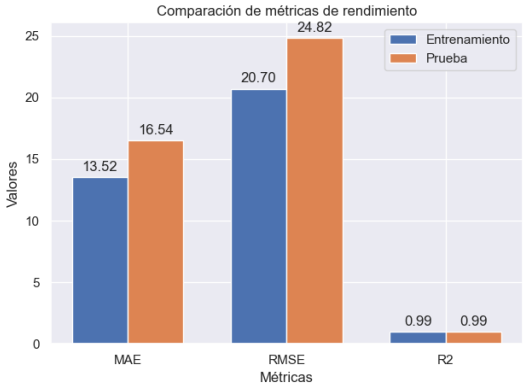
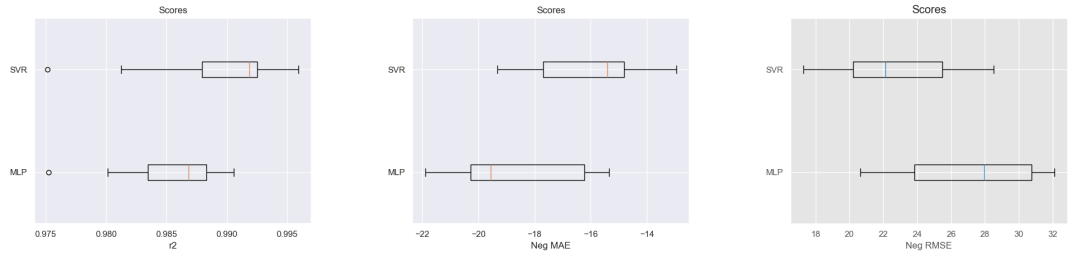


Figura 4.6: Métricas de rendimiento de SVR con los nuevos datos y variable cluster

Observamos como la mejora es evidente respecto a los modelos de la sección anterior, por lo que podemos concluir que hemos hecho un buen trabajo. Ahora nos quedaría probar entre utilizar la variable cluster o utilizar directamente la variable marca como una variable categórica. En otras palabras, ver si estamos recogiendo toda la información respectiva a las marcas en la agrupación que hemos hecho.

Los resultados utilizando la variable marca como variable categórica directamente se pueden ver en las figuras 4.8 y 4.9. Si los comparamos con las figuras 4.5 y 4.6, podemos ver como no solo no estamos perdiendo información, sino que la mejoría es evidente. Así pues, ya tenemos una variable que va a mejorar nuestros futuros modelos. En las figuras 4.7 y 4.10 podemos ver los resultados de promediar los errores al entrenar 10 modelos escogiendo para cada modelo de manera alea-

toria los conjuntos de training y de test. Hay dos hechos interesantes, el primero, que ya habíamos mencionado, es que la varibla cluster es mejor que la variable marca, y el segundo es que en culquiera de los casos, el SVR parece mejor modelo que la red neuronal.



(a) Comparación R^2 (b) Comparación NMAE (c) Comparación RMSE

Figura 4.7: Comparación de métricas variable cluster

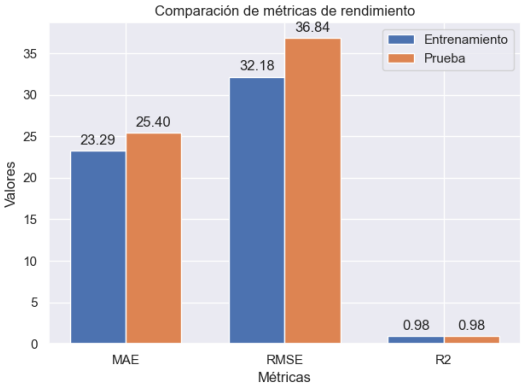


Figura 4.8: Métricas de rendimiento de MLP con los nuevos datos y variable marca

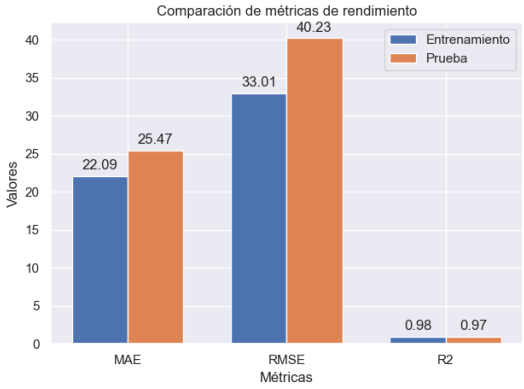
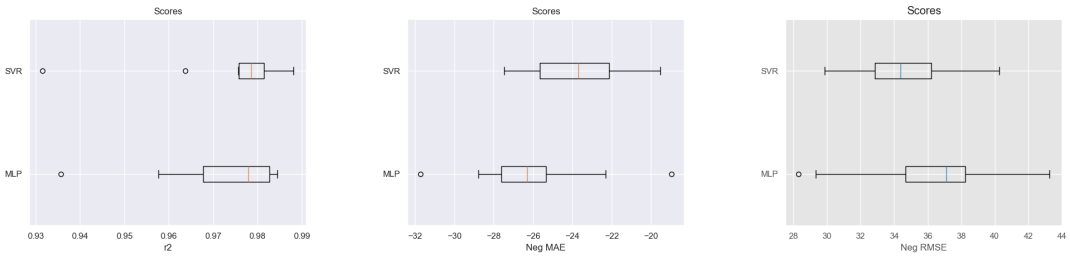


Figura 4.9: Métricas de rendimiento de SVR con los nuevos datos y variable marca



(a) Comparación R^2 (b) Comparación NMAE (c) Comparación RMSE

Figura 4.10: Comparación de métricas variable marca

Capítulo 5

Modelos basados en árboles. El modelo final

Hasta ahora, tenemos dos hechos importantes probados. El primero es que el mejor modelo hasta el momento es el SVR, y el segundo es que tenemos claro cuales son las mejores variables para entrenar los modelos. Sin embargo, todavía no hemos probado modelos basados en árboles. Podemos diferenciar diferentes tipos de modelos basados en árboles, los principales son: Bagging, Boosting y Randomization. Según [2], está claro que cualquiera de estos métodos supone una mejora del decission tree. Sin embargo, entre estos tres métodos puede haber diferencias, principalmente basadas en el ruido presente en la muestra de la población con la que se trabaja. Como en nuestro caso los datos tienen muy poco ruido, boosting es la mejor opción. Dentro de boosting, el mejor método es el Gradient Boosting.

5.1. The Xtreme Gradient Boosting (XGB)

Según *Gradient Boosting* [8, Section 4.4.5.3], Gradient Boosting es una técnica de aprendizaje automático que se utiliza para construir un modelo predictivo robusto a partir de un conjunto de modelos débiles, típicamente árboles de decisión. El principio básico detrás del Gradient Boosting es el ensamblaje secuencial de modelos, donde cada modelo sucesivo trata de corregir los errores de los modelos anteriores. Este enfoque permite crear un modelo fuerte y preciso combinando múltiples modelos débiles.

Como hemos mencionado, el proceso de entrenamiento de Gradient Boosting implica la construcción de árboles de decisión de forma secuencial. Cada nuevo árbol se enfoca en los errores cometidos por los árboles anteriores, ajustándose a los residuos de las predicciones anteriores. Para lograr esto, se utiliza el método de descenso de gradientes, que minimiza una función de coste específica que mide la

discrepancia entre las predicciones del modelo y los valores reales.

La función de coste utilizada en Gradient Boosting generalmente se define como:

$$\sum_{i=1}^{n} L(y_i, F(x_i))$$

Donde:

- $L(y_i, F(x_i))$ es una función de pérdida que mide la discrepancia entre la predicción del modelo $F(x_i)$ y el valor real y_i.
- n es el número de muestras en el conjunto de datos de entrenamiento.

Para prevenir el overfiting y controlar la complejidad del modelo, el GB puede incorporar técnicas de regularización. El nuevo modelo que surge se conoce como Xtreme Gradient Boosting.

El XGB es una implementación optimizada de Gradient Boosting. La principal mejora que incluye sobre el algoritmo tradicional de Gradient Boosting es el uso de una función de regularización adicional para controlar la complejidad del modelo y evitar el sobreajuste. Así, la función de coste en XGB se extiende para incluir una función de regularización $\Omega(F)$, que penaliza la complejidad del modelo:

$$\sum_{i=1}^{n} L(y_i, F(x_i)) + \Omega(F)$$

Donde:

- $L(y_i, F(x_i))$ es la función de pérdida.
- n es el número de muestras en el conjunto de datos de entrenamiento.
- $\Omega(F)$ es una función de regularización que penaliza la complejidad del modelo F.

Además, XGB no solo se enfoca en minimizar los errores de predicción mediante el ajuste de los árboles de decisión sucesivos, sino que también incorpora técnicas como el manejo de valores vacíos y la paralelización para mejorar la eficiencia y el rendimiento del modelo [3].

El XGB es otro modelo que puede aceptar muchos hiperparámetros. Es muy importante decidir qué hipérparametros elegir y sobre cuáles hacer Grid Search, ya que es un modelo que tarda mucho en entrenarse. Algo interesante es que a diferencia del resto de modelos mencionados, XGB es una librería aparte de python, fuera de scikit-learn. El hiperparámetro más importante que ajustar es la

profundidad de los árboles (max_depth), ya que necesitamos árboles simples, por lo que debe de ser un número relativamente bajo, en nuestro caso tras el ajuste con Grid Search toma un valor de 3. De igual manera, el lerning_rate, que sirve para prevenir el sobreaprendizaje, toma un valor de 0.2. Finalmente, n_stiamtors, que es el número de arboles entrenados, se fija en 500. Este parámetro es muy fácil de entender pero a la vez muy importante, ya que siempre hay que elegirlo de manera particular para el problema, pues un número inferior de árboles resultará en underfitting, mientras que un número muy alto de árboles en overfitting. Los resultados los podemos ver en la figura 5.1.

5.1.1. Comparando XGB y SVR

Como tenemos claro que el mejor modelo de los capítulos anteriores es el SVR, tendremos que comparar el XGB con el SVR. La verdad que los resultados son muy interesantes, porque no está claro cual es el mejor modelo. Si nos fijamos en la figura 5.3, parece que el SVR es algo mejor, aunque por muy poco. Recordar que en esta figura se obtienen los errores mediante GS. Por otro lado, si nos fijamos en las figuras 5.1 y 5.2, parece que el XGB es mejor, tiene menos sesgo, aunque presenta bastante overfitting. Básicamente, lo que está pasando, es que el XGB tiene menos sesgo pero más varianza, mientras que el SVR tiene más sesgo, pero menos varianza. Como no se puede decidir cual es mejor, habrá que entrenar un modelo de cada tipo con los datos scrapeados y ver que métricas podemos obtener de ahí.

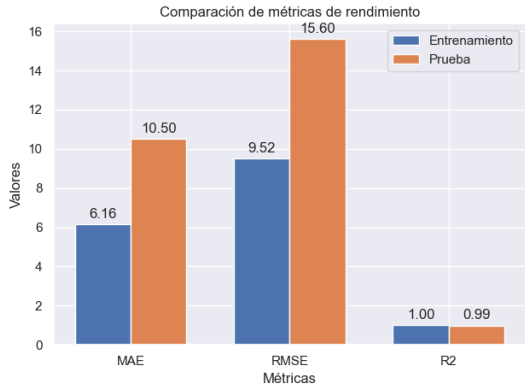


Figura 5.1: Métricas de rendimiento de XGB

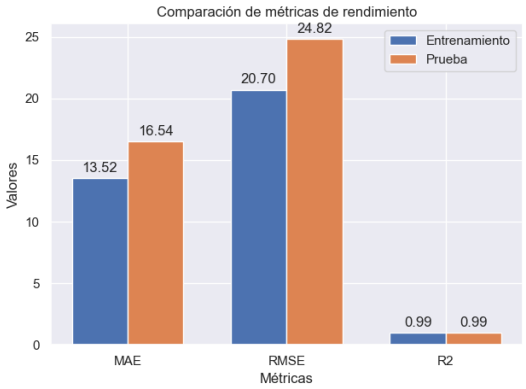
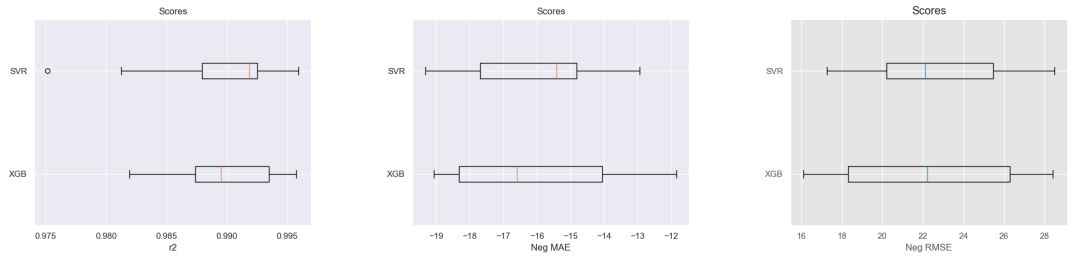


Figura 5.2: Métricas de rendimiento de SVR



(a) Comparación R^2 (b) Comparación NMAE (c) Comparación RMSE

Figura 5.3: Comparación de métricas entre SVR y XGB

5.2. Modelos utilizando los datos obtenidos por el Scrapeo

5.2.1. Datos Utilizados

. Hasta ahora no habíamos utilizado los datos recopilados mediante nuestro scrapeador aparte de para el clustering. La razón es que ya habíamos empezado a trabajar con los otros datos y además el scrapeador seguía funcionando para recopilar datos de vehículos. La distribución de estos datos finales se puede observar en la figura 5.4. Cabe mencionar que los datos con los que se entrenaron los anteriores modelos están incluidos en estos datos, ya que la distancia temporal entre la toma de las distintas observaciones no era muy grande, a lo sumo 2 meses. Evidentemente fueron obtenidos de la misma fuente, ya hemos hablado antes de la importancia de esto en el tema de renting de vehículos. Las variables con las que contamos son:

- cuota_sin_iva: Nuestro target. El precio de la cuota de renting del vehículo sin iva.
- plazo: El plazo de meses que dura el renting: Puede tomar 3 valores: 36, 48, 60 y 72.
- precio_inicial: El precio de compra del vehículo sin extras
- Km_anuales: Los kilómetros máximos permitidos por año que puede recorrer el vehículo, sin incurrir en penalizaciones.
- Cluster: Agrupación de marcas de vehículos en función de su depreciación.

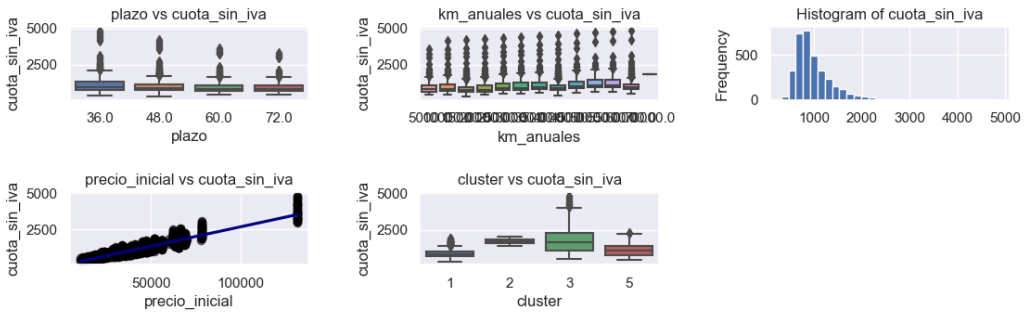


Figura 5.4: Visualización de los datos disponibles tras el scrapeo

Aquí se incluye un cambio bastante importante respecto a todos los modelos anteriores, que es considerar las variables de Km_anuales y plazo como variables numéricas en vez de variables categóricas. Hay varias razones para ello. Por un lado, con los nuevos datos, tenemos muchos más valores que pueden tomar estas variables, así que por como funciona el *one hot encoder*, estaríamos creando una gran cantidad de variables binarias. Además, estamos añadiendo dimensionalidad al problema, pues lo vectores creados por el *one hot encoder* serían vectores linealmente independientes entre si, esto conlleva una perdida de información entre la relación que puede existir entre los valores que toma una variable categórica. Por ejemplo, cogiendo la variable plazo, 48 meses queda entre 36 meses y 60 meses (y además a la misma distancia de ambos), luego si de 36 meses a 48 meses la cuota cae, es de esperar que de 48 a 60 meses la cuota también caiga. Si los vectores asociados a dichos valores de las variables son linealmente independientes, entonces no se puede captar esa relación. Finalmente, nos da la opción de generalizar los valores que pueden tomar las variables. Si entendiéramos estas variables como categóricas, entonces podrían tomar solo los valores con los que ha sido entrenado el modelo. Como el contexto de este modelo es desplegarlo como una funcionalidad de un SaS, un cliente podría introducir el valor que quisiera, y sería interesante que nosotros pudiéramos darle un resultado.

En la siguiente sección se ve que en ningún caso este cambio supone una peor precisión del modelos. De hecho, las pruebas que se hicieron considerando los dos escenarios dejaron ver una mejora de la precisión utilizando variables numéricas. Eso sí, la mejora en la precisión era muy moderada.

5.2.2. Comparación final de los modelos

Hemos entrenado un XGB y un SVR con estos datos. La forma de entrenarlos ha sido igual que en el caso anterior, es decir, hemos elegido los mismos parámetros antes mencionados mediante GridSearch. Estos parámetros han sido los siguientes:

- XGBoost: learning_rate = 0.4, elegido entre 0.1 y 0.5; max_depth = 2, elegido entre 1 y 5; n_estimators = 3500, elegido entre 500 y 5000 [5].
- SVR: C = 1000, elegido entre 0.00001 y 1000; $\gamma = 1$, elegido entre 0.0001 y 10 [14].

Como podemos ver en las figura 5.7, la diferencia es abismal. El XGB es mucho mejor modelo que el SVR con los datos finales, de hecho, no solo observamos la gran diferencia entre la comparación, si no vemos que el XGB es un modelo muy bueno, de hecho, parece dificilmente mejorable. Sin embargo, en la figura 5.5, podemos ver como el error es bastante bajo, pero sigue teniendo un poco de overfitting, lo cual puede dar un poco de miedo a la hora de ponerlo en producción, para ello habrá que utilizar un conjunto de validación completamente diferente a la población con la que se ha entrenado hasta ahora. La gran diferencia en los resultados que observamos seguramente se deba a la disminución de variación que ha podido presentar el XGB manteniendo poco sesgo. Sin embargo, en el SVR, al incorporar más variables y más datos, es posible que no haya sido capaz de predecir tan bien como antes y haya aumentado su sesgo. En resumen, el XGB es el mejor modelo que tenemos con todo lo probado.

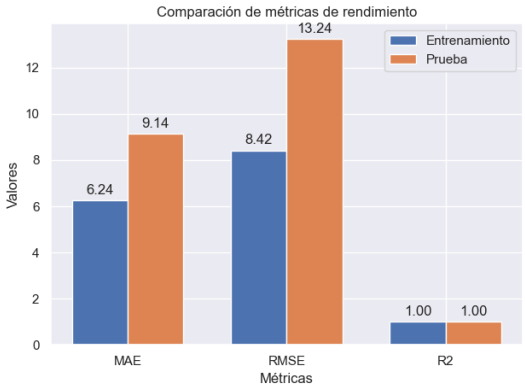


Figura 5.5: Métricas de rendimiento de XGB con el dataset final

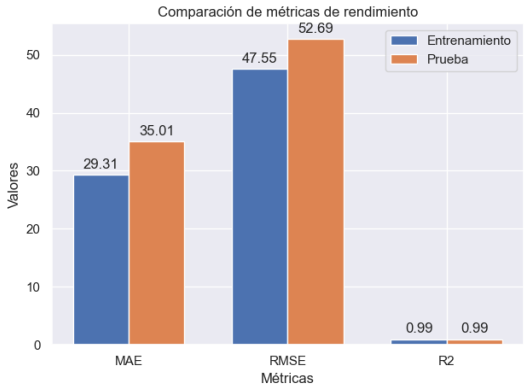
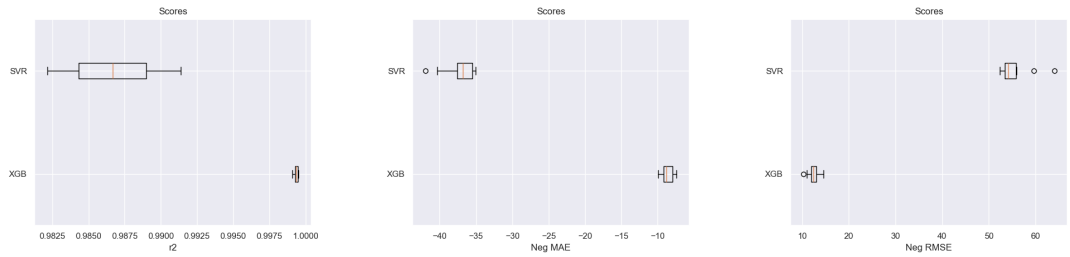


Figura 5.6: Métricas de rendimiento de SVR con el dataset final



(a) Comparación R^2 (b) Comparación NMAE (c) Comparación NMSE

Figura 5.7: Comparación de métricas entre SVR y XGB con los datos finales

5.3. Validación

Como hemos comentado antes, es necesario validar el modelo. Ya hemos decidido cual va a ser el modelo final y con que hiperparámetros. Ahora lo que vamos a hacer es pasarle un conjunto de datos que no haya visto nunca y ver si el error se mantiene cercano al observado en test en el apartado anterior.

5.3.1. Conjunto de Validación

Para crear el conjunto de validación, hemos hecho una cosa muy sencilla. Recordemos que las posibles combinaciones entre Kilómetros máximos por año y meses del renting eran 60, cogimos los 60 coches más vendidos del año 2023 desde una página de internet. Ahora, para estos coches seleccionamos para cada uno una posible combinación, empezando por 5000 kilómetros y 36 meses, y terminando por 75000 kilómetros y 72 meses. Así tenemos coches de marcas distintas, con precios distintos, considerando todos los posibles valores de las variables categóricas.

5.3.2. Resultados de la validación

Los resultados de la validación se pueden ver en la tabla 5.1. Efectivamente, vemos que hay un poco de sobreaprendizaje, sin embargo, consideramos que podemos asumirlo. Al fin y al cabo, la diferencia entre la validación y el test no es tanta, y las métricas de validación están por debajo de lo que nos habíamos propuesto como un modelo exitoso. Así, solo falta poner el modelo en producción.

	RMSE	MAE	R^2
Validación	20.484	16.768	0.979
Training	8.429	6.248	0.99
Test	13.242	9.144	0.99

Cuadro 5.1: Métricas de rendimiento para el modelo final

5.4. Implementación actual y futura

En un primer momento, y en la forma actual que funciona, la puesta en producción se hizo mediante la librería de python Joblib. Joblib es un conjunto de herramientas para proporcionar pipelines en Python. En particular, permite computación paralela simple y fácil, caché de disco de funciones y re-evaluación del modelo. Además, Joblib está optimizado para ser rápido y robusto en grandes catidades de datos, y tiene optimizaciones específicas para arreglos de numpy [6].

En un principio, tampoco se utilizan todas las funcionalidades que ofrece Joblib. Solamente, una vez que se tuvo el modelo entrenado, se guardó en un archivo .plk, es decir, un archivo de "pickle", utilizado para serializar y deserializar objetos en Python. La serialización es el proceso de convertir un objeto en una secuencia de bytes que puede ser almacenada en un archivo, y la deserialización es el proceso inverso: convertir la secuencia de bytes de nuevo en un objeto Python.

Joblib permite guardar modelos ya entrenados en archivos .plk, y cargar archivos .plk en cualquier otro script de python. Una vez que hemos cargado el archivo .plk, para poder predecir el target de nuevos datos solo necesitaremos utilizar el método predict de scikit-learn. De esta manera, la primera puesta en producción del modelo se basó simplemente en incluir el modelo cargado en un archivo .plk en el código de generación de BBDD de nuestra empresa, de forma que cada vez que se actualice la BBDD, esta incluya una nueva columna con el valor de renting estimado para cada coche. Recordemos que esta valor es para una empresa de renting concreta.

Actualmente se trabaja en una implementación mucho más profesional, mediante un proceso de extracción, transformación y carga (ETL). Este es el proceso consistente en combinar datos de diferentes orígenes en un gran repositorio central llamado almacenamiento de datos.La ETL utiliza un conjunto de reglas para limpiar y organizar datos en bruto y prepararlos para el almacenamiento, el análisis de datos y el machine learning (ML) [16]. La herramienta encargada de hacer esto es Apache Airflow. Apache Airflow es una plataforma de código abierto para desarrollar, programar y monitorear flujos de trabajo orientados a batch. El marco extensible de Python de Airflow te permite crear flujos de trabajo que se conec-

tan prácticamente con cualquier tecnología. Airflow se puede desplegar de muchas formas, desde un solo proceso en tu portátil hasta una configuración distribuida para soportar incluso los flujos de trabajo más grandes [1].

Con estas herramientas, el objetivo de la implementación sería como sigue. En un servidor, se deja ejecutando el scrapeador realizado mediante airflow. Una vez que se tienen los datos recopilados, se procesan para poder tener un formato igual al formato de entrada que admite nuestro modelo. Una vez que se tienen los datos procesados, por un lado se almacenan en un bucket de S3 (el almacenamiento en la nube que ofrece AWS) destinado a los datos scrapeados, con la fecha en que se termina el scrapeo como nomnbre del archivo. Por otro lado, los datos procesados se dividen en training y test, teniendo una proporción de 80-20 respectivamente. Con los nuevos datos de trainig se entrena un modelo y este nuevo modelo se compara con el que actualmente hay en producción. La manera de compararlo es sencilla, se evalúa cada uno con los nuevos datos de test. Si el modelo que está en producción obtiene mejores resultados, no se hace nada, pero si el nuevo modelo obtiene mejores resultados, este se despliega en producción y el modelo antiguo deja de usarse.

Capítulo 6

Conclusiones

A lo largo de este trabajo se ha explicado un caso de uso de analítica avanzada. El caso propuesto ha consistido en desarrollar un modelo de Machine Learning desde la recopilación de los datos hasta la implementación en el pipeline de la empresa. A lo largo del proyecto se han puesto en práctica muchas técnicas diferentes, desde la recopilación de datos de diversas formas, como el scrapeo web, el análisis de datos, el desarrollo de un modelo, con sus técnicas y pruebas correspondientes y la implementación final. Todo este proceso, como es evidente, ha dado lugar a una serie de conclusiones, que trataremos de resumir en este último capítulo.

En primer lugar, hemos podido observar como en la vida real, el principal problema para realizar un modelo de Machine Learning viene de los datos. En este caso concreto teníamos unos datos muy limitados, dando resultado a modelos sesgados y con una varianza relativamente alta, y hasta que no hemos conseguido mejorar nuestros datos, no hemos podido obtener un modelo que quedara dentro de los límites propuestos. Seguramente, con más datos, se pudiera haber hecho un modelo con una capacidad de generalización mayor al que tenemos, pues hay que recordar que simplemente contábamos con 5 variables, y aun así hemos conseguido un coeficiente de correlación muy alto.

Siguiendo con este tema, es importante mencionar dos temas importantes sobre los datos a utilizar. Normalmente al conjunto de todos los datos los llamamos población, y a los que tenemos nosotros para realizar nuestros modelos predictivos, muestra de la población. Es muy importante que la muestra de la población no este sesgada, es decir, no tener una muestra que comparta una característica determinada o que la muestra presente una característica no presente en la población. En nuestro caso esto ha sido un problema en varias ocasiones. Recordemos que solo podíamos coger ofertas de vehículos actuales y además, emitidas por una empresa determinada. Cualquier otro dato fuera de estos, hubiera sesgado nuestro modelo.

El segundo tema importante que sacamos de aquí, es que obtener datos de calidad puede ser muy difícil. En nuestro caso el proceso manual dificultó mucho

avanzar en un tiempo razonable al principio, y automatizar el proceso de recolección de datos requirió un conocimiento técnico bastante elevado. Además, tampoco mejoramos en tiempo con la recolección automática, pero si que pudimos dedicarnos a hacer otras cosas mientras los datos se recopilaban.

En cuanto a la primera fase del caso de uso, observamos bastantes cosas interesantes. La primera conclusión curiosa es que muchas veces unos datos que a priori parecen simples no se modelan bien con los modelos más simples, como fue nuestro caso. Así, siempre es interesante probar modelos de diferentes tipos y diferentes complejidades, dentro de las posibilidades temporales y de procesamiento de cada uno. Por otro lado, observamos como a veces hay que volver hacia atrás para recoger mejores datos o simplemente más datos. Este fue nuestro caso, y es que antes de hacer el modelo no teníamos claro cuantos datos iban a ser suficientes, algo importante en nuestro caso ya que como ya he dicho, la obtención era difícil.

La automatización de la recolección de datos no trajo conclusiones tan interesantes como las anteriores, más allá de que la librería que usamos es seguramente la peor opción de las que ofrece python en cuanto a tiempo se refiere, pero la única que te permite interactuar de una manera clara con la web, que era lo que necesitábamos en nuestro caso.

Ahora, los modelos probados fueron muchos, y no hemos hablado de ellos todavía. En un primer lugar, la regresión lineal y todos sus derivados no parecieron en ningún momento una buena opción, así que las descartamos rápido, seguramente debido a las diferentes combinaciones posibles entre variables categóricas. Esto dio paso a que probáramos modelos no lineales y altamente más complejos. De aquí surgió que los mejores modelos en un primer lugar era la red neuronal y el SVR. Modelos altamente complejos y no explicables.

Sin embargo, estos modelos tenían mucho sesgo y poca varianza. Aquí es cunado concluimos que el problema seguramente no fueran los modelos, sino los datos de los que disponíamos, ya que más datos nos permitían entrenar un modelo más complejo sin aumentar la varianza, reduciendo así el sesgo. Así que un hecho muy importante de los aprendidos es volver hacia detrás si las cosas no salen como esperamos, y no seguir avanzando confiando en que los problemas ya se arreglarán. Además, siempre se pueden sacar conclusiones del trabajo ya hecho, aunque no se vaya a utilizar.

Finalmente, el mejor modelo resultó ser el XGB, un modelo muy potente basado en sucesivos árboles muy simples que van aprendiendo a corregir los errores del anterior. Este es un modelo que resultó ser una mejora muy importante en nuestro proyecto, y es que es capaz de mejorar casi cualquier modelo, como en nuestro caso el SVR, por lo que es un modelo que habría que probar siempre en cualquier caso de uso que incorpore modelos predictivos, siempre y cuando no sea importante la explicabilidad.

En resumen, hemos podido extraer muchas conclusiones de este trabajo, donde hemos tardado mucho tiempo en realizar el proyecto por todos los problemas surgidos, pero el aprendizaje adquirido será de ayuda en futuros proyectos para desarrollarlos con más rapidez y eficacia.

Bibliografía

[1] Apache Software Foundation. Apache airflow documentation. https://airflow.apache.org/docs/apache-airflow/stable/index.html, 2024. Accessed: 2024-06-04.

[2] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[4] Selenium Contributors. Selenium with python. https://selenium-python.readthedocs.io/, 2024. Accessed: 2024-06-02.

[5] XGB Contributors. Xgboost with python. https://xgboost.readthedocs.io/en/stable/, 2024. Accessed: 2024-06-02.

[6] Joblib developers. Joblib with python. https://joblib.readthedocs.io/, 2021. Accessed: 2024-06-04.

[7] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, Burlington, MA, 3rd edition, 2011.

[8] John D. Kelleher, Brian Mac Namee, and Aoife D'Arcy. *Fundamentals of Machine Learning for Predictive Data Analytics*. The MIT Press, Cambridge, MA, 2nd edition, 2020.

[9] Kuhn Max & Johnson Kjell. *Applied Predictive Modeling*. Springer, New York, 2013.

[10] Scikit learn Maintainers Team. cross val score. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html, 2024. Accessed: 2024-06-02.

[11] Scikit learn Maintainers Team. Decision trees. https://scikit-learn.org/dev/modules/tree.html#tree, 2024. Accessed: 2024-06-02.

[12] Scikit learn Maintainers Team. Kmeans. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, 2024. Accessed: 2024-06-02.

[13] Scikit learn Maintainers Team. Mlpregressor. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html, 2024. Accessed: 2024-06-02.

[14] Scikit learn Maintainers Team. Support vector regression (svr). https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html, 2024. Accessed: 2024-06-02.

[15] Jason Long and Ian Ward. Jsonlines documentation. https://jsonlines.org/, 2024. Accessed: 2024-06-04.

[16] Amazon Web Services. Etl description. https://aws.amazon.com/es/what-is/etl/#:~:text=es%20AWS%20Glue%3F-,%C2%BFQu%C3%A9%20es%20ETL%3F,central%20llamado%20almacenamiento%20de%20datos., 2023. Accessed: 2024-06-04.