Universidad Torcuato Di Tella

Master in Management + Analytics

Tesis de Maestría

Métodos de Aprendizaje Estadístico y Automático para la Estimación del Load Factor y Asignación de Capacidad para una Aerolínea

de

Leandro Bello

Directora

Dra. Magdalena Cornejo

Julio 2023

RESUMEN

En cualquier industria, conocer con antelación la demanda permite una asignación de recursos óptima. En la industria aeronáutica, uno de los tantos recursos a asignar es la capacidad; es decir la cantidad de asientos que se van a destinar a la venta. Cualquier asiento que no se venda en una ruta, es un asiento que podría haberse vendido en otra. Es por este motivo que tener estimaciones de demanda precisas tiene mucho valor, permitiendo que las decisiones de último momento afecten lo menor posible a la compañía.

En el presente trabajo se utilizan múltiples técnicas de aprendizaje estadístico y automático para estimar la demanda de pasajeros para 10 rutas de cabotaje en 3 meses de una aerolínea argentina. Posteriormente, con la información de demanda estimada, se reasignará la capacidad total de la que se dispone, en pos de maximizar los ingresos totales durante esos meses.

El mejor resultado de predicciones se obtuvo con un modelo de redes neuronales *encoder-decoder* con mecanismo de atención. Se ajustaron múltiples modelos estadísticos lineales y basados en árboles, pero ninguno lo hizo mejor que los modelos de redes neuronales.

Finalmente, la asignación de capacidad que maximizaba el ingreso se logró mediante el planteo de un problema de programación lineal entera.

ABSTRACT

In any industry, anticipating demand in advance allows for optimal resource allocation. In the aviation industry, one of the many resources to allocate is capacity, which refers to the number of seats to be made available for sale. Any unsold seat on one route is a seat that could have been sold on another. Therefore, accurate demand estimations are highly valuable, as they minimize the impact of last-minute decisions on the company.

This study employs multiple statistical and machine learning techniques to estimate passenger demand for 10 domestic routes over a span of 3 months for an Argentine airline. Subsequently, based on the estimated demand information, the total available capacity will be reassigned to maximize overall revenue during those months.

The best prediction results were obtained using an encoder-decoder neural network model with an attention mechanism. Multiple linear and tree-based models were also adjusted, but none performed better than the neural network models.

Finally, the capacity allocation that maximized revenue was achieved through an integer linear programming approach.

ÍNDICE DE TEMAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 Contexto

El *Revenue Management* es una estrategia utilizada por determinadas industrias para gestionar la asignación de su capacidad a diferentes clases de tarifa con el fin de maximizar los ingresos. Esta técnica de trabajo no es aplicable en todas las industrias, ya que depende de la naturaleza de éstas. Las condiciones que debe cumplir dicha industria son las siguientes:

- El vendedor ofrece un stock fijo de capacidad perecedera.
- Los clientes realizan reservas antes de utilizar el producto o servicio.
- El vendedor gestiona un conjunto de clases de tarifas, cada una de las cuales tiene un precio fijo (al menos en el corto plazo).
- El vendedor puede cambiar la disponibilidad de las clases de tarifas en el tiempo.

La industria aeronáutica reúne todas estas condiciones, y es con la que se va a trabajar en esta tesis.

Para poder determinar las estrategias correctas que permitan una maximización de ingresos, es necesario conocer la demanda. Es por ello que saber de antemano el *load factor* con el que volará cada ruta en cada mes es crucial.

En la empresa donde trabajo, durante mucho tiempo se estuvo utilizando un modelo muy básico que proyectaba el *load factor* con una simple operación aritmética de sustracción. Este modelo no arrojaba resultados muy confiables y presentaba diferencias con la realidad, además de carecer de rigurosidad académica.

Existen varios trabajos académicos que proponen soluciones a este problema de predicción de demanda, como el trabajo de Gaset (2022) de predicción agregada de pasajeros en las principales rutas regionales en Argentina. La autora sugiere la utilización de redes neuronales recurrentes, principalmente las de largo plazo (*Long short-term memory* LSTM). Estos modelos, como indica la tesis, demostraron ser muy robustos y excelentes para capturar tendencias en temporadas altas y bajas. Como contrapartida, para que tengan buen desempeño, es necesario una gran cantidad de datos.

Por su parte, el trabajo de Al-Sultan et al. (2021) concluye que, para su correspondiente caso de estudio, el modelo que mejor resultados arrojó fue la serie de tiempo bayesiana, seleccionada entre otras cinco técnicas de aprendizaje estadístico. Asimismo, sugiere la incorporación de variables económicas al modelo para mejorar las proyecciones, tales como el producto bruto interno, tasa de desempleo y precio de boletos.

En un estudio académico presentado por Adetayo et al. (2018) sobre la predicción de la demanda de tráfico aéreo en Nigeria, se utilizaron dos modelos ampliamente conocidos: el modelo de media móvil (*moving average*) y el modelo de suavización exponencial (*exponential smoothing*). El objetivo de dicho estudio era determinar qué modelo se aproximaba mejor a la demanda real. Los investigadores se basaron únicamente en la demanda de años anteriores como variable para predecir la demanda futura. Al comparar los resultados obtenidos, se encontró que el modelo de media móvil fue el que mejor se ajustó a la demanda real de tráfico aéreo en Nigeria. Esto sugiere que la tendencia histórica de la demanda puede proporcionar una buena estimación para pronosticar el comportamiento futuro.

En contraposición a este último, se puede citar el trabajo de Weatherford et al (2002) en donde se comparan los modelos más tradicionales de pronóstico de series de tiempo de media móvil y suavización exponencial con una red neuronal básica. El autor concluyó que el modelo de red neuronal *feed-forward* utilizado fue el que obtuvo los mejores resultados en el corto plazo, y fue el segundo mejor en el largo plazo. El hecho de haber obtenido los mejores resultados se lo adjudica a la capacidad del modelo de combinar información previa y capturar patrones de manera superior a los modelos tradicionales. Al final de su trabajo, agrega que el desafío de las redes neuronales es encontrar la arquitectura y los hiperparámetros perfectos para conseguir el desempeño adecuado. Según enuncia este trabajo, ésta fue la primera publicación en la utilización de redes neuronales en la industria aérea.

El presente trabajo pretende mostrar una solución alternativa que requiere otra estructura de datos. Se evaluarán modelos estadísticos más tradicionales y modelos de redes neuronales de diferentes arquitecturas. Además, la incorporación de capas de *embedding* que se utilizó permite no solamente mejorar la precisión de las estimaciones a través de un problema de aprendizaje supervisado, sino también permite la *clusterización* de ruta aéreas a través de un problema de aprendizaje no supervisado.

Para las predicciones, se utilizará un conjunto de variables más amplio: además de la demanda histórica, se incorporaron otras variables relevantes como la tarifa media, la capacidad de los vuelos, el período de vuelo, las semanas previas al despegue, la cabina utilizada y la ruta aérea específica. Además de estas variables específicas del sector aéreo, también se tuvieron en cuenta factores macroeconómicos que pueden influir en la demanda, como las tasas de desempleo y los índices de inflación. Estas variables se agregaron durante el desarrollo del trabajo para intentar conseguir predicciones más precisas.

Como primer abordaje, se utilizarán modelos estadísticos lineales para intentar conseguir interpretabilidad, luego se utilizarán algoritmos que logren capturar relaciones no lineales, como los modelos basados en árboles y redes neuronales. Finalmente, se escogerá el modelo que menor error presente en 10 rutas y 3 períodos seleccionados.

1.2 Problema

El problema que se intentará resolver es el de conocer con anticipación la demanda de los vuelos, es decir la cantidad de boletos que se venderán en cada mes para cada ruta de cierta aerolínea que opera en Argentina. Al contar con esta información con anterioridad, se reajustará la capacidad ofertada para esas rutas y se reasignarán vuelos de manera tal que maximicen los ingresos totales, variando la tarifa.

1.3 Objetivo

Se pretende predecir el load factor a fin de mes para 10 rutas para los meses de febrero, marzo y abril de 2023, valiéndose de técnicas de aprendizaje estadístico y automático. La calidad de predicción de los modelos se medirá utilizando la medida usual de desempeño del error cuadrático medio, usando como conjunto de validación el 30% de los datos. Además, para ajustar los modelos, se utilizarán técnicas de regularización: ridge (L2), lasso (L2) y *elastic nets* para modelos estadísticos lineales; *bagging*, L1 y L2 para modelos de árboles; y *dropout* y *weight decay* para la red neuronal. Posteriormente, una vez obtenidas las predicciones de demanda, se podrá predecir también el *load factor* y los ingresos. Dicho esto, el objetivo final estribará en realizar una reasignación de capacidad que maximice los ingresos asociados a cada una de esas rutas y cada mes, ajustando las tarifas, para lo cual se considerará utilizar un modelo de optimización lineal.

2 DATOS

Se utilizarán dos bases de datos provistas por la aerolínea donde trabajo. La primera posee la siguiente estructura:

- RED_COMERCIAL: si corresponde a la red de vuelos internacionales, regionales o de cabotaje.
- METRO_RT: alude a "*metro round trip*", que es una variable contempla el aeropuerto de origen y el de destino, incluyendo tanto los vuelos de ida como de vuelta.
- LEG: esta variable contempla el aeropuerto de origen y el de destino.
- CAB: sección destinada a los asientos de los pasajeros. Puede ser W (*premium)* o Y (*economy*).
- ANIO: año en el que parte el vuelo.
- MES: mes en el que parte el vuelo.
- SEMANAS: semanas que faltan para que finalice el período de vuelo.
- RPK: *revenue passenger kilometers*; por sus siglas en inglés. Métrica que se obtiene al multiplicar el número de pasajeros que pagan tarifa por la distancia de vuelo. Esta es una medida que refleja la demanda real de transporte aéreo que genera ingresos para la aerolínea. Contar simplemente el número de pasajeros no tomaría en cuenta cuánto están volando y, por lo tanto, no proporcionaría una imagen precisa de la demanda. Una aerolínea podría tener muchos pasajeros en vuelos cortos, pero su rentabilidad se vería afectada si los pasajeros no están volando distancias largas que generen ingresos significativos.
- ASK: *available seats kilometers*; por sus siglas en inglés. Métrica que se obtiene al multiplicar el número de asientos disponibles a la venta por la distancia de vuelo. Esta medida refleja la capacidad de la aerolínea para transportar a los pasajeros a lo largo de una cierta distancia. Dos vuelos con el mismo número de asientos pueden tener capacidades muy diferentes si uno es un vuelo corto y el otro es un vuelo largo. Por lo tanto, la capacidad se mide en términos de la distancia que se puede cubrir.
- CAP: cantidad de asientos totales destinados a la venta.

Y la segunda, además de compartir varias columnas con la base anterior, contiene los siguientes campos:

- FECHA_VISTO_EMISION: fecha en la que se tomó registro de los datos.

- INC: refiere a "*income*". Son los ingresos acumulados en dólares estadounidenses.
- PAX: suma de boletos vendidos.

Se pretende lograr una única base de datos a partir de la fusión de ambas. El propósito fundamental de este proceso radica en la necesidad de consolidar la información contenida en ambas fuentes en un único conjunto de datos, con el fin de obtener una visión más completa y holística de los fenómenos analizados. Algunas de las columnas compartidas entre ambas bases de datos actúan como puntos de convergencia, permitiendo la correlación de datos y la creación de una base de datos compuesta. Esta unión de tablas se explica en el apartado siguiente llamado "Transformaciones al dataset".

Ambas bases de datos contienen aproximadamente 868.000 registros. Cada registro contiene información de RPK, ASK y CAP (primera base de datos), e INC y PAX (segunda base de datos); ambas segmentadas a nivel de RED_COMERCIAL, METRO_RT, LEG, CAB, ANIO, MES y SEMANAS. A continuación, se ofrece una aproximación de la cantidad de observaciones.

Las base de datos tiene información de 122 rutas, separadas por cabina, para los 12 meses del año y 50 semanas, desde 2019 hasta 2023. Además, la información está dividida en aeropuertos de origen y destino. Mínimamente, cada ruta tiene una ida y una vuelta, pero hay otras que tienen dos aeropuertos, como en el caso de Buenos Aires, que tiene Aeroparque y Ezeiza. En este caso, estas rutas tienen dos idas y dos vueltas. Entonces, el tamaño inicial del *dataset* es:

$$(12\ rutas\ \times\ 4\ aerop. + 110\ rutas\ \times\ 2\ aerop.)\ \times\ 2\ cab.\ \times\ 12\ meses\ \times\ 27\ sem.\ \times\ 5\ años$$
$$\approx 868.000\ observaciones^{1}$$

A los fines de responder a la pregunta de investigación planteada, no interesa tener la información segmentada por idas y vueltas, por lo que se va a colapsar todo a nivel ruta. Tampoco se van a tener en cuenta varios meses correspondientes a los años 2020 y 2021 para evitar los valores atípicos del período de pandemia.

[1] Si bien se dijo que se cuenta con 50 semanas para cada ruta, lo cierto es que no todas las rutas operan con las 2 cabinas, ni en los 5 años, ni están abiertas las 50 semanas. Por este motivo, para aproximar el cálculo, se penalizó a las semanas, de 50 a 27, para ofrecer un número aproximado de registros.

Se dijo que la mayoría de las rutas poseían registros a 50 semanas en avance. Esto quiere decir que se empiezan a tomar datos con casi un año de anticipación. Para los vuelos de cabotaje, no es frecuente que se vendan boletos con tanto adelanto, por lo que, en los registros correspondientes a valores de semana altos, las variables de TM_SEM *(tarifa media semanal)* y PAX_SEM (boletos vendidos esa semana), van a ser todas cero. Por este motivo es que se van a tomar semanas menores a 40, para no tener muchos valores nulos. Estas variables son campos calculados, no son parte de la base de datos original, y se calcularán en el apartado de "Transformaciones al dataset".

Finalmente, se retirarán 12 rutas del análisis por ser atípicas y extraordinarias. Con todos estos recortes y filtros, ambos *datasets* quedan con 141.000 registros.

$$98\ rutas\ \times\ 2\ cabinas\ \times\ 12\ meses\ \times\ 15\ semanas\ \times\ 4\ años$$
$$\approx 141.000\ observaciones^{2}$$

Por cuestiones de confidencialidad, los datos se presentarán en forma anonimizada.

Los modelos se entrenarán con 98 rutas de cabotaje, desde 2019 hasta octubre de 2023, en frecuencia semanal (descontando los meses de pandemia). Es importante remarcar que se cuenta con registros para meses en adelante (que todavía no volaron) porque la venta de boletos se abre aproximadamente 48 semanas en avance.

Se omitieron los registros del año 2020 a partir de marzo, y los primeros meses de 2021 para algunas rutas por ser una época tan atípica. Las predicciones se harán para los meses de febrero, marzo y abril de 2023.

Se entrenarán y evaluarán los modelos en todas las rutas, todos los años, todos los meses y ambas cabinas. Esto se logrará dividiendo al *dataset* en dos conjuntos: entrenamiento (70%) y testeo (30%), estando ambos balanceados en todas las variables para conseguir mejores mediciones y predicciones.

[2] Por el mismo motivo explicado anteriormente es que para la cuenta se consideran 20 semanas. En realidad, la mayoría de las rutas tienen 40 semanas, pero no todas operan todos los meses, en las dos cabinas y todo el año. Hay incluso rutas que dejaron de operar.

2.1 Transformaciones al *dataset*

Se comenzó agregando un campo calculado en la segunda tabla: TM (tarifa media), definida como el cociente entre INC (ingresos) y PAX (reservas). Esta variable busca medir la elasticidad de la demanda. Se espera que cuanto más elevada sea la tarifa, menor sea la entrada de reservas.

Posteriormente, a partir de las dos bases de datos se generó una única, combinando la información de ambas. A continuación, se realizó una serie de transformaciones al *dataset* de manera que sea apto para poder correr una regresión lineal. No interesaba tener la tabla segmentada por LEG (ida y vuelta), por lo que se agrupó por METRO_RT, sumando los valores de RPK, ASK, CAP y PAX. Luego se calculó la columna LF (*load factor*), que se define como el cociente entre RPK y ASK.

Es de interés saber, además, la tarifa media y la demanda semanales. Para la tarifa media se calculó el cociente entre las diferencias de los elementos consecutivos de los vectores INC y PAX. Es decir:

$$tm\_sem_i = \frac{inc_i - inc_{i+1}}{pax_i - pax_{i+1}} \qquad (01)$$

Y para la demanda simplemente se tomó el numerador de la ecuación anterior y se la denominó PAX_SEM.

La columna semanas funciona como un registro de evolución de la ruta, e indica cuántas semanas faltan para que termine el mes en el que van a partir todos los vuelos. Por ejemplo, si en la semana 7, el valor de la variable PAX es 20.000, es esperable que en la semana 6 este valor aumente, porque transcurrió una semana y las reservas van incrementando conforme se va acercando la fecha de vuelo.

De esta manera, la tabla queda estructuradas para poder correr una regresión. A continuación, se muestra la estructura final:

- METRO_RT: esta variable contempla el aeropuerto de origen y el de destino, incluyendo tanto los vuelos de ida como de vuelta. Variable categórica que puede adoptar el nombre de 122 rutas aéreas diferentes.

- CAB: variable destinada a la cabina de asientos de los pasajeros. Variable categórica que puede adoptar W (*premium)* o Y (*economy*).
- ANIO: año en que se voló, o va a volar. Variable discreta ordinal que puede adoptar valores de 2019 hasta 2023.
- MES: mes en que se voló, o va a volar. Variable discreta ordinal que puede adoptar valores de 1 hasta 12.
- SEMANAS: semanas que faltan para que finalice el período de vuelo. Variable discreta ordinal que puede adoptar valores de 0 hasta 40.
- RPK: *revenue passenger kilometers*; por sus siglas en inglés. Métrica que se obtiene al multiplicar el número de pasajeros que pagan tarifa por la distancia de vuelo. Variable continua que puede adoptar valores de 0 hasta infinito.
- ASK: *available seats kilometers*; por sus siglas en inglés. Métrica que se obtiene al multiplicar el número de asientos disponibles a la venta por la distancia de vuelo. Variable continua que puede adoptar valores de 0 hasta infinito.
- CAP: cantidad de asientos totales destinados a la venta. Variable continua que puede adoptar valores de 0 hasta infinito.
- LF: métrica definida como el cociente entre RPK sobre ASK. Puede entenderse como la ocupación del vuelo expresada en porcentaje. Variable continua que puede adoptar valores de 0 hasta 1.
- PAX: suma de boletos vendidos hasta ese momento. Variable discreta positiva que puede adoptar valores de 0 hasta infinito.
- TM_SEM: tarifa media semanal. Variable continua que puede adoptar valores de 0 hasta infinito.
- INC: son los ingresos acumulados en dólares estadounidenses. Variable continua que puede adoptar valores de 0 hasta infinito.
- TM: tarifa media acumulada. Variable continua que puede adoptar valores de 0 hasta infinito.
- PAX_SEM: cantidad de boletos vendidos en esa semana. Variable discreta positiva que puede adoptar valores de 0 hasta infinito.

2.2 Análisis exploratorio de datos

Se omitió la columna METRO_RT correspondiente a cada una de las rutas porque contenía muchas categorías y en el histograma no aportaba ningún valor.

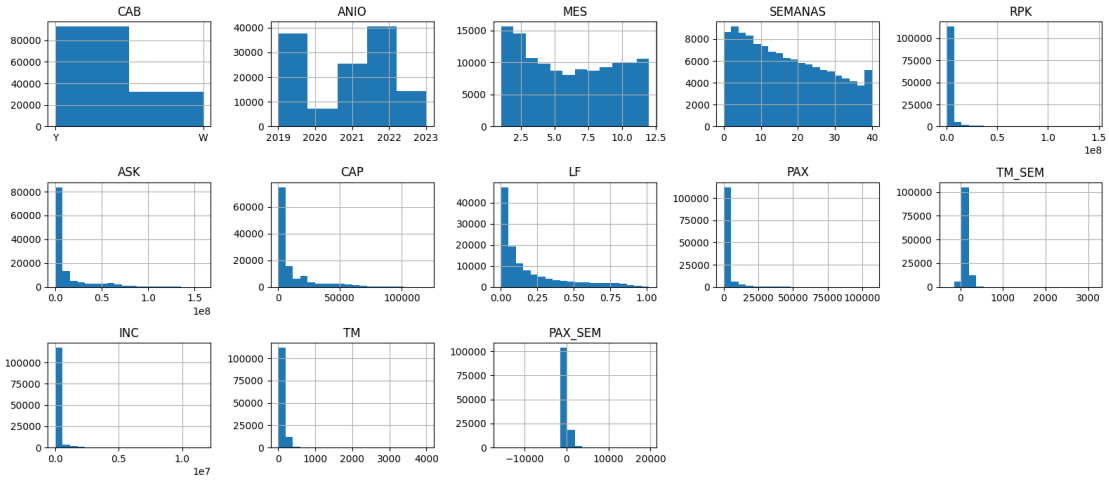


Figura No. (01) Histograma de frecuencias de las variables del *dataset*.

Observando estos histogramas, se puede ver que se tiene más registros de la cabina Y que de la W, que hay más información de los años 2019, 2021 y 2022, que hay más vuelos en los meses de enero y febrero, y que la distribución de las semanas tiene una forma trapecial. Esto último es evidente, puesto que no todas las rutas se abren con 40 semanas de anticipación.

El valor de PAX_SEM, que corresponde a la entrada de reserva semanal, evidencia la presencia de *outliers*; porque el eje de abscisas muestra valores de -10.000 y 20.000. Esta diferencia de valores comprime la escala y es por este motivo que la gran mayoría de los valores están representados en torno al cero. Lo mismo ocurre con la variable TM_SEM.

Por último, la distribución de la variable *load factor* indica que la mayoría de los registros muestran una ocupación cercana a 0% y muy pocos cercanos al 100%. Esto es esperable por el mismo motivo de que los vuelos empiezan a llenarse a semanas próximas al período de vuelos, es decir semanas cercanas a cero.

También se realizó un gráfico *pairplot* para visualizar la relación entre pares de variables.

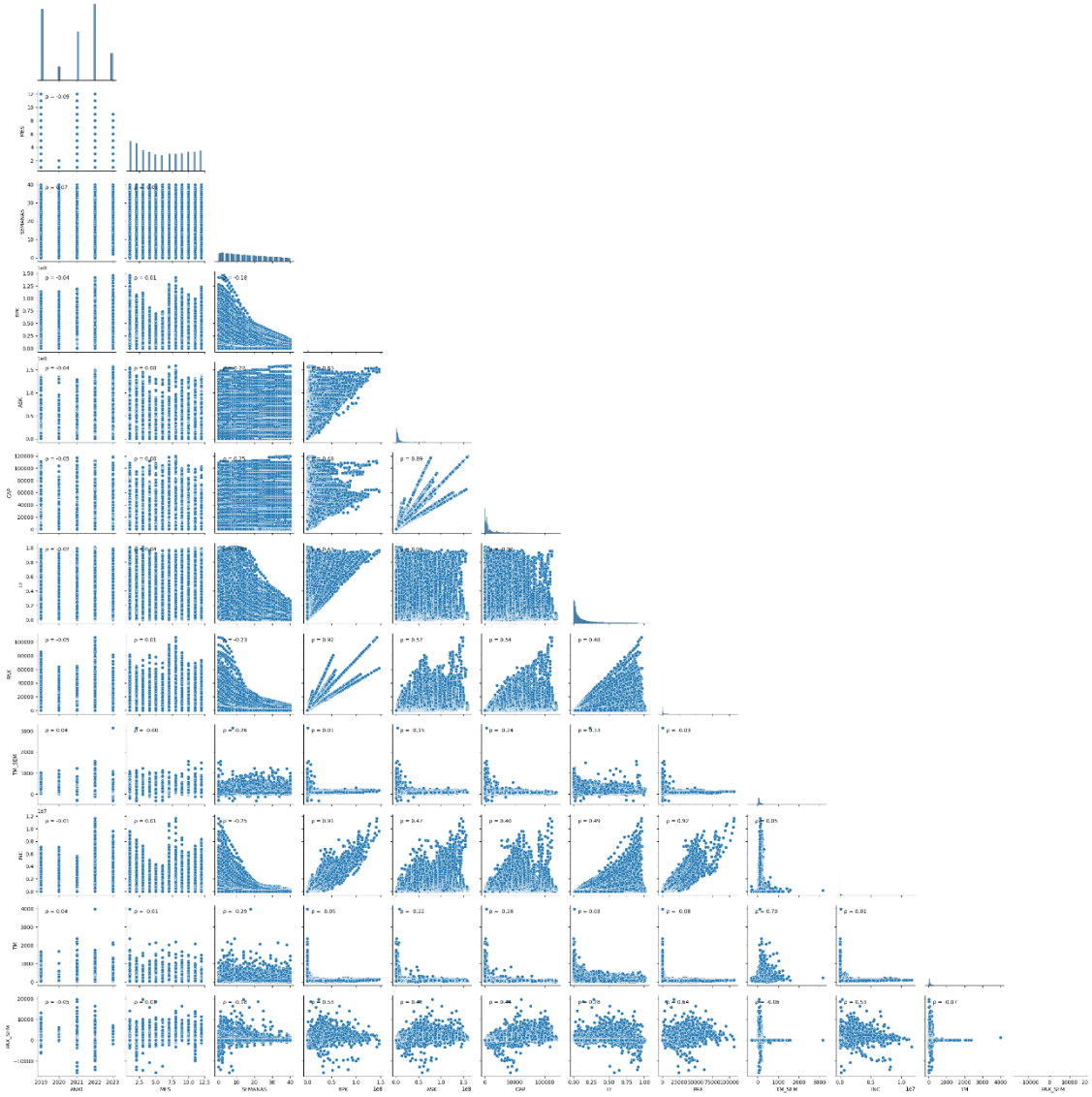


Figura No. (02) *Pairplot* de las variables del *dataset*.

Intencionalmente se ordenó el *dataset* dejando como última variable a la variable objetivo para de esta manera observar únicamente la última fila y ver cómo correlaciona ésta con las demás covariables. Se puede ver cómo las variable RPK y PAX guardan una alta correlación entre sí, al igual que las variables ASK y CAP. Esto es porque por definición, las variables RPK y ASK son las variables PAX y CAP multiplicadas por una constante de distancia (distancia del aeropuerto de origen al aeropuerto de destino).

Las variables que mayor correlación tienen son INC (ingresos) y PAX (reservas), con 0,92. Esto es lógico, puesto que los ingresos aumentan conforme lo hacen las reservas.

Hay varios puntos aislados que parecen ser *outliers*. Sin embargo, es difícil concluir eso con solamente ver el *pairplot* porque el gráfico muestra relaciones entre pares de variables, no la relación conjunta.

Para analizar la temporalidad y detectar datos atípicos, se puede graficar la cantidad de reservas en los períodos de enero de 2019 hasta septiembre de 2023 para todas las rutas de la red doméstica.

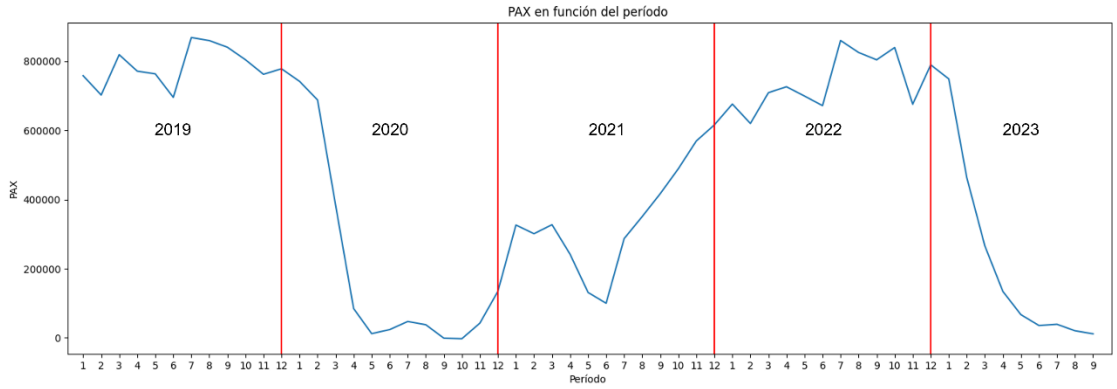


Figura No. (03) Gráfico de reservas de toda la red doméstica en función de los períodos de enero de 2019 hasta septiembre de 2023.

En el gráfico anterior se pueden identificar varias cuestiones. En primer lugar, los períodos más estables e inafectados por los efectos de la pandemia son los años 2019 y 2022. En estos años se ven los picos de reservas que corresponden a los períodos de vuelo de alta demanda, que son siempre en julio, debido a las vacaciones de invierno; y los períodos de vuelo de baja demanda, que suelen ser en mayo y junio. En 2020, a partir de marzo se puede ver la caída en las reservas, recuperándose muy levemente a fines de ese año. Posteriormente los valores vuelven a caer en los meses de abril, mayo y junio, que fue cuando hubo un rebrote del virus. Finalmente, la caída que se ve en el año 2023 ocurre porque la base de datos tiene información actualizada hasta fines de enero de 2023. Esto quiere decir que los vuelos programados en los meses próximos están aún vendiéndose y no alcanzaron su máxima capacidad. El objetivo de este trabajo es, precisamente, determinar la capacidad ideal para determinadas rutas en los meses de febrero, marzo y abril de 2023, luego de haber estimado la demanda para esos períodos.

El próximo gráfico muestra la cantidad de reservas para no para toda la red doméstica, sino para las 10 rutas seleccionadas para las predicciones.

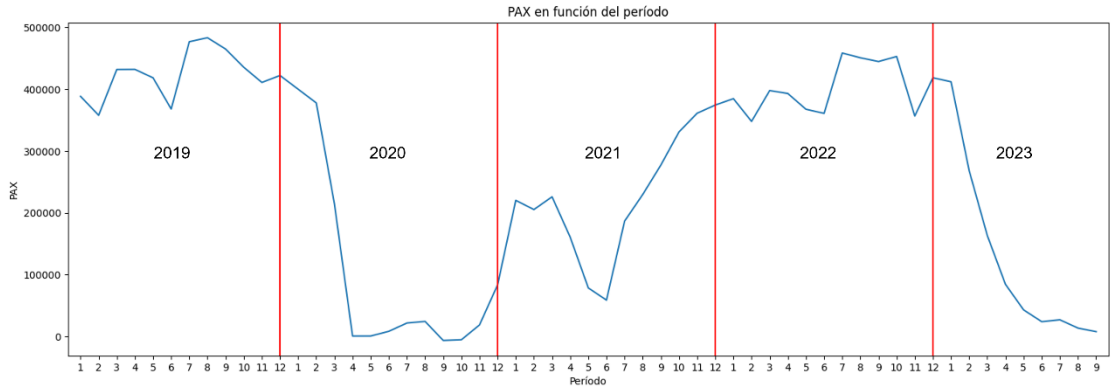


Figura No. (04) Gráfico de reservas de 10 de las rutas más significativas en función de los períodos de enero de 2019 hasta septiembre de 2023.

Se puede ver cómo la curva es muy similar a la de las reservas de toda la red. Esto es porque las rutas seleccionadas son muy significativas y representan más de la mitad de los pasajeros. El gráfico de barras a continuación refleja esta situación.

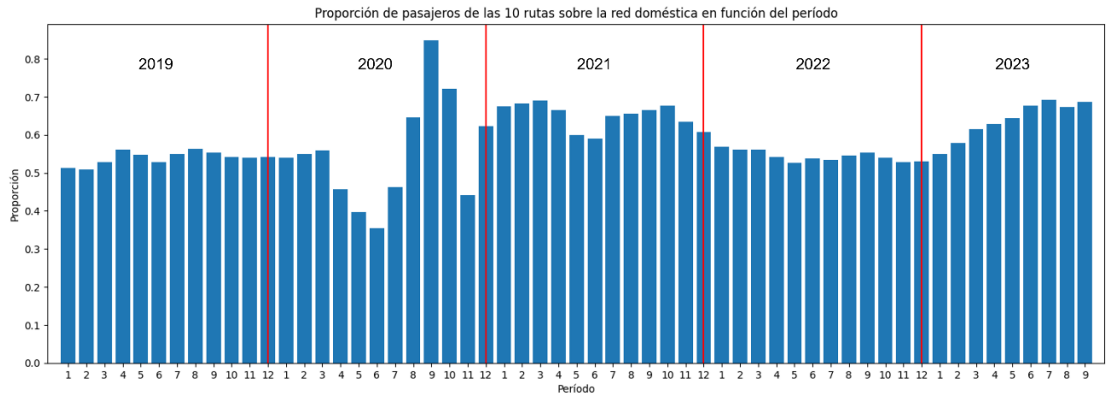


Figura No. (05) Gráfico de barras de la proporción de pasajeros de las 10 rutas en función de los períodos de enero de 2019 hasta septiembre de 2023.

Luego de observar los gráficos de reservas, se excluyeron del *dataset* todos los meses del año 2020, a excepción de enero y febrero. También se retiraron los primeros 5 meses de 2021.

Otra métrica que puede analizarse es la tarifa media de cada período volado. En el gráfico siguiente se visualiza esta variable sin mostrar el año 2023 porque, como se mencionó anteriormente, la información es hasta fines de enero, por lo que no existe ningún mes finalizado de ese año como para graficar esta métrica.

Figura No. (06) Gráfico de tarifa media a mes volado de cada período.

Se puede ver que el año más irregular es, nuevamente, el año de la pandemia. Como la tarifa media es calculada como el cociente entre los ingresos mensuales y la cantidad de reservas, al ser muy pequeño el denominador, la tarifa media alcanza valores más elevados. En períodos más estables, se puede ver que las temporadas bajas (abril, mayo y junio) son las que menor tarifa media tienen.

3 METODOLOGÍA

La metodología utilizada en este trabajo se basa en el desarrollo de un algoritmo de predicción de demanda de tráfico aéreo y en la posterior asignación de capacidad en las diferentes rutas consideradas. Para lograr este objetivo, se emplearán diversas técnicas de aprendizaje estadístico y automático:

.

- Regresión lineal múltiple
- Regresión lineal regularizada L1, L2 y *elastic nets*
- Regresión ponderada
- Regresión con componentes principales
- Regresión con mínimos cuadrados parciales
- Regresión con *Random Forest*
- Regresión con *XGBoost*
- Red neuronal *feed-forward*
- Red neuronal recurrente ordinaria
- Red neuronal LSTM
- Red neuronal *encoder-decoder*

Este trabajo inicia con una detallada explicación teórica de los modelos que se utilizarán a lo largo del estudio. Durante el desarrollo, el trabajo se presenta de manera progresiva y explicativa, con el objetivo de guiar al lector en un recorrido gradual hacia la construcción de modelos más precisos y complejos. Conforme se avanza en el trabajo y se obtienen resultados, se brinda una minuciosa explicación de cada paso y de los análisis realizados. Se presta especial atención a ofrecer una comprensión detallada de los hallazgos y los métodos utilizados, asegurando que no solo se muestren los resultados finales, sino también la lógica y los fundamentos detrás de ellos.

Luego de entrenados todos los modelos, se seleccionará el mejor de ellos en base al error cuadrático medio obtenido en el conjunto de validación. En caso de que dos modelos presenten valores similares en la métrica de selección, se realizará un test de hipótesis para determinar que la diferencia sea estadísticamente significativa. Si no se pudiese rechazar la hipótesis, se tomarán ambos modelos para realizar las predicciones sobre un conjunto de testeo y medir nuevamente el erro cuadrático medio. Si hasta esa instancia, las diferencias no son significativas, el modelo se elegirá en base a su complejidad e interpretabilidad.

Los modelos de redes neuronales utilizados en este trabajo incluyen una capa de *embeddings* de rutas. Los *embeddings* son representaciones numéricas densas que capturan características y relaciones entre las rutas. Para visualizarlos en el plano y analizar la similitud entre distintas rutas, se aplicarán diversas técnicas, como t-SNE (*t-distributed Stochastic Neighbor Embedding*) y PCA (*Principal Component Analysis*) sobre el modelo de redes neuronales que menor error obtenga.

Para agrupar las rutas en *clusters*, se utilizará el algoritmo de *K-means*. Este algoritmo agrupa los datos en K *clusters*, donde K representa la cantidad de grupos definidos. La selección de la cantidad óptima de grupos se realizará mediante diversas técnicas, como la inspección visual de los gráficos de los *embeddings* y el análisis de métricas de calidad de agrupamiento, como el coeficiente de silueta.

Una vez escogido el mejor modelo y obtenidas las predicciones de demanda de tráfico aéreo, se utilizará un modelo de optimización lineal para calcular la mejor asignación de capacidad de una selección de rutas. El modelo de optimización tiene en cuenta las predicciones de demanda y las restricciones de capacidad. Mediante la resolución de este modelo, se determinará la asignación óptima de capacidad, maximizando ingresos asociados.

Para los modelos lineales, de árboles y red neuronal *feed-forward*, la variable objetivo será TM_SEM, que corresponde a los pasajeros transportados en la próxima semana. Y para los modelos de redes neuronales recurrentes y *encoder-decoder*, la variable objetivo será PAX en el próximo paso de tiempo.

Cuando se desea estimar la demanda del final del período de interés, se requiere una aproximación especial. En este punto, se realiza una serie de estimaciones secuenciales, en las cuales el resultado de una predicción se convierte en el input de la siguiente estimación. Esto se asemeja a una concatenación de estimaciones, donde la predicción anterior se utiliza como punto de partida para la predicción subsiguiente.

Las siguientes tablas muestran un ejemplo de la estructura de la base de datos para cada caso, así también como los datos de entrada y valor objetivo.

BASE DE DATOS

METRO_RT	SEMANAS	ANIO	MES	CAB	PAX	PAX_SEM
RUTA A	3	2022	4	Y	7.000	800
RUTA A	2	2022	4	Y	7.800	1.100
RUTA A	1	2022	4	Y	8.900	1.100
RUTA A	0	2022	4	Y	10.000	700
RUTA B	3	2022	4	Y	6.300	800
RUTA B	2	2022	4	Y	7.100	900

INPUT

METRO_RT	SEMANAS	ANIO	MES	CAB	PAX
RUTA A	3	2022	4	Y	7.000

VARIABLE OBJETIVO

PAX_SEM
800

Tabla No. (01) Ejemplo de estructura de base de datos, input (variables explicativas) y variable objetivo de los modelos lineales, de árboles y red neuronal *feed-forward*.

BASE DE DATOS

METRO_RT	SEMANAS	ANIO	MES	PAX	CAP	LF
RUTA A	3	2022	4	7.000	15.000	0,46
RUTA A	2	2022	4	7.800	15.000	0,52
RUTA A	1	2022	4	8.900	16.000	0,55
RUTA A	0	2022	4	10.000	16.000	0,62

INPUT DE ENTRADA (SECUENCIA)

PAX	CAP	LF	INC	TM	...	SALARIO
7.000	15.000	0,46	546.000	78	...	120
7.800	15.000	0,52	616.200	79	...	120
8.900	16.000	0,55	720.900	81	...	120

INPUT DE ENTRADA (ESTÁTICO)

METRO_RT	ANIO	MES	CAB
RUTA A	2022	4	Y

VARIABLE OBJETIVO

PAX
10.000

Tabla No. (02) Ejemplo de estructura de base de datos, input (secuencia y vector estático) y variable objetivo de los modelos de redes neuronales recurrentes *y encoder-decoder*.

A modo de resumen, se pretende seguir el siguiente orden:

- Explicación teórica de los modelos
- Ajuste de modelos
    - Modelos lineales
    - Modelos basados en árboles
    - Modelos de redes neuronales
- Análisis posterior
    - Selección del mejor modelo
    - Interpretación de *embeddings*
- Cierre de trabajo
    - Planteo de problema de optimización lineal para la toma de decisión (análisis prescriptivo)
    - *Conclusiones*.

4 EXPLICACIÓN TEÓRICA DE LOS MODELOS

4.1 Modelos estadísticos lineales

4.1.1 Regresión lineal

Un modelo de regresión es una función matemática que describe una relación entre la variable de respuesta y las variables explicativas. La expresión generalizada del modelo en cuestión es la siguiente:

$$E[Y|X_1 = x_1, X_2 = x_{2,} \ \dots \ X_k = x_k] = \Phi(x_1, x_2, \dots, x_k) \qquad (02)$$

Si se asume linealidad en los parámetros, el modelo puede expresarse de la siguiente manera:

$$\Phi(x_1, x_2, \dots, x_k) = \ \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k \qquad (03)$$

o, lo que es lo mismo, pero cambiando el nombre de algunas variables y subíndices:

$$\eta = \ \beta_0 + \beta_1 x_1 + \cdots + \beta_{p-1} x_{p-1} \qquad (04)$$

A este modelo se lo conoce como regresión lineal. Si se tiene una sola variable predictora, se dice que la regresión lineal es simple, mientras que, si el modelo se ajusta con más de una covariable, se lo denomina regresión lineal múltiple. El parámetro η es desconocido, las variables explicativas $x_1, x_2, \dots, x_{p-1}$ son constantes conocidas y $\beta_j \ (j = 0, 1, \dots, p-1)$ son parámetros desconocidos que serán estimados. Si los x_j son variados y los n valores, $Y_1, Y_2, \dots Y_n$, son observados, entonces:

$$Y_i = \ \beta_0 + \beta_1 x_{i,1} + \cdots + \beta_{p-1} x_{i,p-1} + \varepsilon_i \qquad (i = 1, 2, \dots, n) \qquad (05)$$

En la ecuación anterior, Y es una variable aleatoria que fluctúa alrededor de un parámetro desconocido η, y ε es el error aleatorio. La misma puede expresarse en forma matricial de la siguiente manera:

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,p-1} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,p-1} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}, \qquad (06)$$

o lo que es lo mismo:

$$Y = X\beta + \varepsilon, \qquad (07)$$

donde $x_{1,0} = x_{2,0} = \cdots = 1$. La matriz X de dimensiones $n \times p$ es llamada matriz de regresión y sus columnas son generalmente escogidas de manera tal que sean todas linealmente independientes para que la matriz tenga rango p, es decir que sea de rango completo.

Un método para hallar una estimación de los parámetros β es el llamado *método de mínimos cuadrados* que consiste en minimizar $\Sigma\, \varepsilon_i^2$ respecto de β. Al definir $\theta = X\beta$, se minimiza $\varepsilon^T \varepsilon = \|Y - \theta\|^2$ sujeto a $\theta \in C(X) = \Omega$, donde Ω es el espacio columna de X. Si se deja θ variar en Ω, $\|Y - \theta\|^2$ será un mínimo para $\theta = \hat{\theta}$ cuando $(Y - \hat{\theta}) \perp \Omega$. En definitiva, el método de mínimos cuadrados consiste en encontrar A tal que AB sea un mínimo.

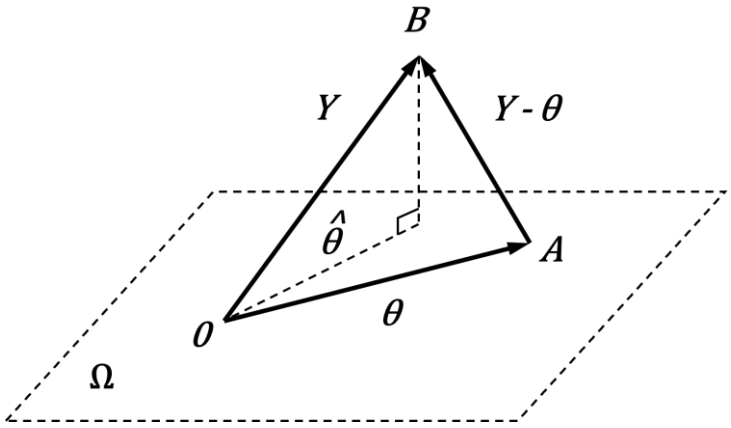


Figura No. (07) Solución gráfica del problema de regresión lineal como método de proyección.

La solución al problema de optimización es $\hat{\theta} = X\hat{\beta}$, con:

$$\hat{\beta} = (X^T X)^{-1} X^T y \quad (08)$$

Esta es la ecuación[3] para hallar los parámetros β que minimizan la suma del cuadrado de las distancias de la recta de estimación y el valor observado.

Para aplicar correctamente el modelo lineal, es necesario que se cumplan las siguientes condiciones:

- Linealidad: la relación entre la variable dependiente y las variables explicativas es lineal.
- Independencia: los errores de la regresión son independientes entre sí y no muestran patrones o correlaciones.
- Homocedasticidad: la varianza de los errores es constante en todos los niveles de las variables explicativas.
- Normalidad: los errores siguen una distribución normal.
- Ausencia de multicolinealidad perfecta: las variables explicativas no están perfectamente correlacionadas entre sí.

4.1.2 Regularización (métodos de encogimiento de parámetros)

Al ajustar una regresión lineal mediante el método de mínimos cuadrados ordinario, se hallan los parámetros que minimizan el error empírico. Son los mejores valores para el conjunto de entrenamiento con el que se ajustó. Sin embargo, si se evalúa sobre otro conjunto, es posible que no se desempeñe correctamente. En este caso, se dice que el modelo presenta poco sesgo pero mucha varianza, porque su calidad predictiva cambia sustancialmente cuando se aplica sobre otro conjunto de datos.

Una solución a este problema son las regresiones regularizadas. En este proyecto se va a trabajar con tres: Ridge, Lasso y una combinación de ambas denominadas Redes Elásticas.

[3] La deducción de la fórmula y no está dentro del alcance de esta tesis. Para más información puede consultar la obra de Seber y Lee, (2003) *Linear Regression Analysis*. Wiley

4.1.2.1 Ridge

El método de Ridge es similar el de mínimos cuadrados ordinario, solo que, en lugar de minimizar la suma de residuos al cuadrado, se le agrega un sumando a la expresión, quedando:

$$\hat{\beta}^{ridge} = argmin\left\{\sum_{n=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2\right\} \qquad (09)$$

El parámetro $\lambda > 0$ controla la complejidad del modelo. A mayor valor, mayor regularización, por lo tanto, menor complejidad. A medida λ que va creciendo, los coeficientes van acercándose asintóticamente a cero. Otra manera equivalente de escribir el problema ridge es la siguiente:

$$\hat{\beta}^{ridge} = argmin \sum_{n=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 \qquad s.t. \sum_{j=1}^{p} \beta_j^2 \leq t \qquad (10)$$

En esta forma puede apreciarse directamente la restricción de presupuesto que se le aplica. La suma de los parámetros al cuadrado no es más que la distancia euclídea (también llamada distancia L2), en donde en la fórmula se explicita que debe ser menor a un valor t. Gráficamente, el problema puede visualizarse en la siguiente imagen:

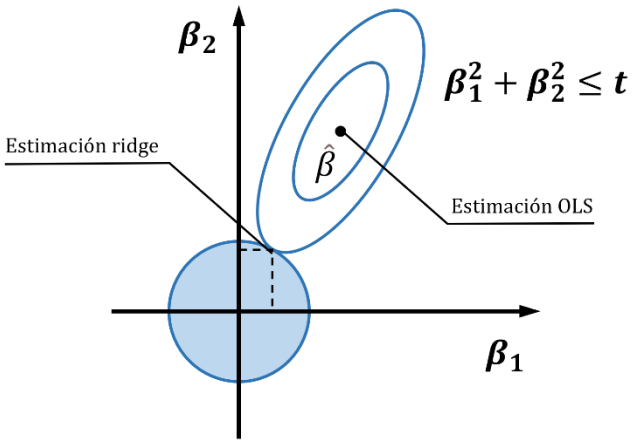


Figura No. (08) Estimación gráfica para el problema de la regresión ridge.

Por cuestiones de simplificación, se ejemplifica con un problema en el plano, con dos variables β_1 y β_2. El radio de la circunferencia es el valor t y las elipses representan el error empírico. Existe una correspondencia uno a uno entre los parámetros t y λ .En la fórmula (09), si se setea el parámetro λ en cero, se está trabajando con el método de mínimos cuadrados, y su estimación es el punto negro $\hat{\beta}$. La elipse va aumentando conforme lo hace λ; y la estimación ridge será el punto de contacto de la circunferencia con la elipse.

Existe otra expresión más, en forma matricial. Resolver la operación siguiente equivale a resolver los problemas de optimización mostrados anteriormente:

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T y \qquad (11)$$

Puede notarse que es muy similar a la fórmula de mínimos cuadrados ordinarios, con un término adicional, que es el parámetro λ multiplicado por la matriz identidad.

4.1.2.2 Lasso

El método lasso es similar al de ridge, con la diferencia de que en la restricción se utiliza la sumatoria de los valores absolutos de los parámetros (distancia L1). La estimación lasso se define, entonces, de la siguiente manera:

$$\hat{\beta}^{lasso} = argmin \sum_{n=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 \qquad s.t. \sum_{j=1}^{p} |\beta_j| \leq t \tag{12}$$

Para un problema en el plano, la imagen a continuación muestra cómo es posible para un parámetro alcanzar el valor cero. Esto se debe a que ahora la región generada por la restricción de presupuesto conforma un rombo, en lugar de una circunferencia. Es probable que la elipse haga contacto en un vértice, que es donde la distancia L1 es máxima de valor t para un parámetro, y valor necesariamente nulo para el otro.

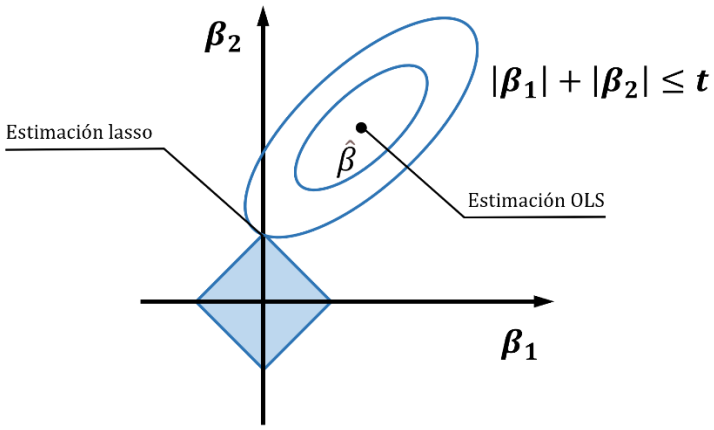


Figura No. (09) Estimación gráfica para el problema de la regresión lasso.

Esta es la principal diferencia entre ridge y lasso, y es la que permite a este último a ser utilizado como selector de variables.

4.1.2.3 Redes elásticas

Los métodos de regularización recién mencionados podrían expresarse de manera más generalizada de la siguiente manera:

$$\hat{\beta} = argmin \left\{ \sum_{n=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\}, \tag{13}$$

donde $q = 1$ corresponde a la regularización lasso y $q = 2$ a la regularización ridge. Al hacer variar el hiperparámetro q entre 1 y 2, se puede pensar al método como un compromiso entre ambos enfoques. Sin embargo, esto no es del todo cierto porque para un valor de $q > 1$, la función de optimización es diferenciable en cero y, por lo tanto, carece de la capacidad que tiene lasso de llevar los coeficientes a exactamente cero. Las redes elásticas vienen a solucionar este problema y se presentan como una combinación de ambas regularizaciones. Se introduce un nuevo hiperparámetro α que varía entre 0 y 1, y que asigna la proporción de cada método. La función resulta, entonces, la siguiente:

$$\hat{\beta}^{enet} = argmin\left\{\sum_{n=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda \sum_{j=1}^{p}(\alpha\beta_j^2 + (1-\alpha)|\beta_j|)\right\} \quad (14)$$

4.1.3 Regresión de componentes principales (PCR)

La regresión de componentes principales (PCR) es una técnica basada en el análisis de componentes principales (PCA, por sus siglas en inglés *Principal Component Analysis*). Tiene como objetivo estimar los valores de la variable dependiente sobre la base de componentes principales seleccionados de las variables explicativas. Como se está trabajando sobre las componentes principales, la dimensionalidad de los regresores se reduce solo a un subconjunto de ellas, lo que reduce el error estándar en las estimaciones de los coeficientes. Es por este motivo que se suele utilizar para solucionar los problemas asociados a la multicolinealidad.

El análisis de componentes principales es un proceso mediante el cual las componentes principales son calculadas y analizadas posteriormente. Es un abordaje de aprendizaje no supervisado porque prescinde de una variable de respuesta asociada a las variables explicativas. Las componentes principales pueden calcularse de diferentes maneras: un problema de optimización, descomposición de valores propios de matriz de covarianza, o descomposición en valores singulares de la matriz de regresión. En este trabajo se va a explicar el último método.

Cualquier matriz A de dimensiones $m\ x\ n$ se puede descomponer en la multiplicación de tres matrices de distintas dimensiones:

$$A = U\,\Sigma\,V^T \tag{15}$$

La matriz U, de dimensiones $m\ x\ m$, contiene en sus columnas los vectores singulares de A por izquierda, que son perpendiculares en $\mathbb{R}^m$; la matriz V^T, de dimensiones $n\ x\ n$, contiene en sus columnas los vectores singulares de A por derecha, que son ortogonales en $\mathbb{R}^n$; y la matriz central Σ, de dimensiones $m\ x\ n$, contiene en su diagonal los valores singulares de la matriz original A. La relación entre ambos grupos de vectores viene dada por la siguiente ecuación:

$$A\,\vec{v} = \sigma\vec{u} \tag{16}$$

Los valores singulares de la matriz Σ se hallan calculando la raíz cuadrada de los autovalores de la matriz $A^T A$, los vectores singulares a derecha se hallan calculando los autovectores de la misma matriz $A^T A$; y los vectores singulares por izquierda se pueden calcular despejando de la ecuación de arriba el vector $\vec{u}$, lo que daría entonces $\overrightarrow{u_i} = \frac{1}{\sigma} A\overrightarrow{v_i}$.

Para hallar las componentes principales de una matriz de datos, primero se centran las columnas, es decir se le sustrae la media, y luego se le aplica la descomposición en valores singulares. Para la regresión de componentes principales, la matriz que importa es la de vectores singulares a derecha, puesto que los n vectores columna de V (o los n vectores fila de V^T) lo que indican es la dirección de mayor varianza, o sea, la dirección de las componentes principales.

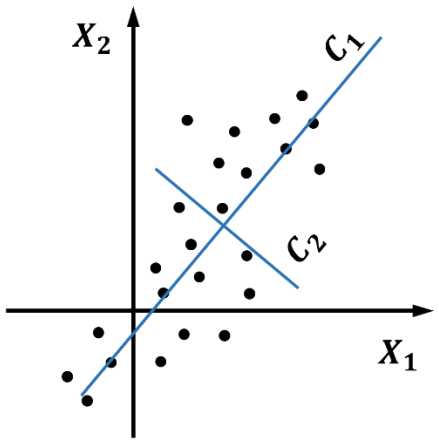


Figura No. (10) Gráfico de dispersión de variables X1 y X2 y visualización de componentes principales C1 y C2.

La imagen anterior muestra una nube de puntos ordenados con coordenadas $(X1_i, X2_i)$, y sus componentes principales en azul. Si se multiplican los vectores singulares a derecha por la transpuesta de la matriz A, se obtienen las componentes de cada vector de A en dirección de las componentes principales. Lo que se logra es otra matriz de las mismas dimensiones que la original, pero ordenadas desde la componente más importante (la que presenta más varianza) hasta la menos importante. A partir de esta nueva matriz se corre un modelo de regresión lineal. Las estimaciones de los coeficientes ahora no van a explicar el comportamiento de las variables originales $X_1, X_2, \dots, X_p$, sino la de las componentes principales $C_1, C_2, \dots, C_p$.

Calcular las componentes principales y trabajar sobre éstas es una solución al problema de multicolinealidad entre covariables. Los coeficientes estimados sobre estas componentes pierden explicabilidad porque cada dirección está compuesta por una proporción de todas las variables explicativas. Es por ello que, luego de correr el modelo de regresión, los coeficientes se multipliquen por los elementos de las componentes principales, para volver a expresar la regresión en términos de las variables originales, y no de sus componentes principales. La ventaja de utilizar este modelo es que de cada predictor se extrae su parte de mayor varianza y se elimina lo más posible la independencia entre variables, lo que resulta en un modelo con menores errores estándar, mejores intervalos de confianza y mejor poder de inferencia.

4.1.4 Regresión con mínimos cuadrados parciales (PLS)

Este método es similar al anterior en la medida que reduce el número de variables explicativas a un menor conjunto de variables no correlacionadas. Sin embargo, el problema que soluciona respecto del método de regresión con componentes principales es que la reducción de dimensión sí contempla la variable independiente.

Las variables predictoras son combinadas en p variables, denominadas las componentes de mínimos cuadrados parciales, o simplemente variables latentes. Cada una de ellas es una combinación lineal de las variables explicativas originales. Es decir, la variable latente i se define como la multiplicación entre la matriz de variables X y la matriz A, compuesta de coeficientes. La expresión matemática se muestra a continuación:

$$LV_i = X\,A = \begin{pmatrix} x_{1,1} & \cdots & x_{1,p} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,p} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_p \end{pmatrix} \tag{17}$$

Los valores de los coeficientes α serán los que maximicen la covarianza entre la variable latente y la variable dependiente (o variable objetivo), sujetas a una restricción, es decir:

$$\widehat{\alpha_i}^{PLS} = argmax\ Cov(LV_i, y) \qquad s.t. \sum_{j=1}^{p} \alpha_j^2 = 1 \tag{18}$$

En caso de que se desee extraer una segunda variable latente, a la ecuación anterior se le agrega una restricción adicional de ortogonalidad respecto de las variables halladas anteriormente. El método continúa ajustando una regresión lineal utilizando como variables predictivas a las variables latentes. Como el número de estas variables es siempre menor (o igual) al número de variables originales, al ajustar una regresión lineal se calculan menos coeficientes; de ahí el nombre de mínimos cuadrados parciales. Posteriormente, se multiplican los coeficientes hallados por los coeficientes α para obtener una expresión lineal en función de las variables originales.

4.2 Modelos basados en árboles

Los árboles de decisión son modelos de aprendizaje supervisado que se utilizan tanto para problemas de clasificación como de regresión. Estos modelos toman decisiones basadas en reglas if-then que se organizan en una estructura de árbol. Cada nodo del árbol representa una característica o atributo, y las ramas representan las posibles salidas o resultados. Los árboles de decisión son populares debido a su interpretabilidad y capacidad para capturar relaciones no lineales y complejas en los datos.

Los árboles de decisión pueden ser construidos utilizando diferentes algoritmos. Un de ellos es l algoritmo de partición binaria recursiva, que es uno de los enfoques más comunes para construir árboles de decisión. Comienza con un nodo raíz que contiene todo el conjunto de datos y realiza divisiones binarias recursivas en función de alguna medida de impureza, como la ganancia de

información o el índice Gini. En cada paso, se selecciona una característica y un umbral que dividen el conjunto de datos en dos subconjuntos más puros. Este proceso se repite de forma recursiva en cada subconjunto hasta que se cumpla un criterio de parada, como alcanzar un número máximo de niveles o una pureza mínima en las hojas.

Otro algoritmo es la partición generalizada, que se refiere a un enfoque más flexible donde no se limita a divisiones binarias en cada nodo. En lugar de eso, se permiten particiones múltiples o más complejas en función de las características y los umbrales seleccionados. Esto permite capturar relaciones más complejas y no lineales en los datos, lo que puede mejorar el rendimiento del modelo.

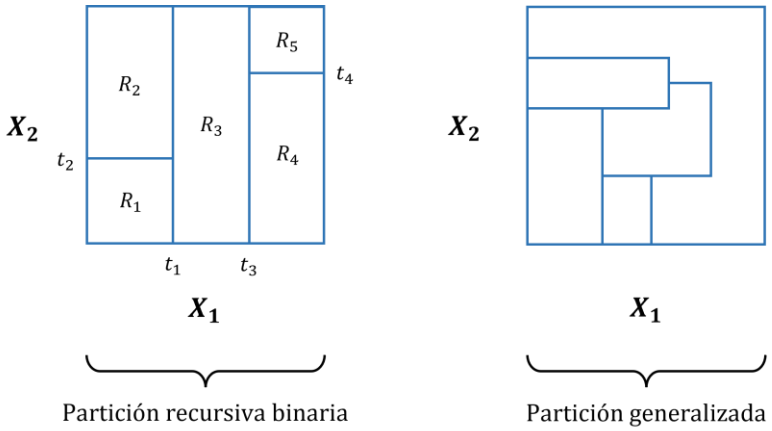


Figura No. (11) Partición recursiva (izquierda) y partición generalizada (derecha).

Los árboles de decisión tienden a ser propensos al sobreajuste, especialmente cuando se construyen árboles profundos. Esto significa que pueden memorizar los datos de entrenamiento y tener dificultades para generalizar a nuevos datos.

4.2.1 Random Forest

Una solución al problema de sobreajuste de árboles es el algoritmo *Random Forest*, que combina múltiples árboles de decisión independientes para tomar una decisión final. El enfoque que utiliza este algoritmo se llama *bagging* (*Bootstrap Aggregating*). *Bagging* es una técnica que combina múltiples modelos independientes entrenados en subconjuntos aleatorios del conjunto de datos de entrenamiento. El proceso de *bagging* implica crear múltiples muestras de entrenamiento mediante muestreo con reemplazo (*bootstrap*) de los datos originales. Cada muestra se utiliza para entrenar

un modelo individual, y las predicciones de estos modelos se promedian para obtener la predicción final.

En el caso de *Random Forest*, se utilizan árboles de decisión como modelos base y, además de entrenarse cada árbol con un subconjunto de los datos originales, también se utilizan un subconjunto aleatorio de variables. El número de variables a utilizar constituye un hiperparámetro a seleccionar. Durante el proceso de construcción de cada árbol, se selecciona una muestra aleatoria de los datos de entrenamiento con reemplazo, lo que significa que una misma instancia puede aparecer múltiples veces en una muestra y algunas instancias pueden quedar excluidas. Esto crea árboles independientes que capturan diferentes patrones en los datos. Luego, las predicciones de todos los árboles se promedian (en el caso de regresión) o se realiza una votación (en el caso de clasificación) para obtener la predicción final.

El *bagging* reduce el sobreajuste al promediar las predicciones de múltiples modelos independientes y tiene la ventaja adicional de proporcionar una medida de importancia de las características. Sin embargo, cada modelo individual en el *bagging* puede no ser tan preciso como se desearía, ya que no se corrigen los errores cometidos por los modelos anteriores.

4.2.2 XGBoost

Existe otro algoritmo, denominado *XGBoost* (*Extreme Gradient Boosting*) que utiliza el enfoque de *boosting*. Este enfoque construye un conjunto de modelos de manera secuencial, donde cada modelo se enfoca en corregir los errores cometidos por los modelos anteriores. El proceso de *boosting* implica asignar pesos a las instancias de datos en función de su dificultad de predicción y ajustar los modelos para centrarse en las instancias más difíciles.

En el caso de *XGBoost*, se utilizan árboles de decisión como modelos base. Durante el proceso de construcción de cada árbol, se ajusta el modelo a los errores residuales de los árboles anteriores. En cada iteración, se actualizan los pesos de las instancias de datos en función de los errores cometidos, y se ajusta el modelo para minimizar estos errores. El conjunto de árboles se construye de manera secuencial, centrándose cada vez más en los casos difíciles de predecir.

El *boosting* tiene la capacidad de construir un modelo final más fuerte y preciso al enfocarse en los errores cometidos por los modelos anteriores. Además, técnicas como la regularización y el ajuste

de hiperparámetros se utilizan en este proceso para controlar la complejidad del modelo y prevenir el sobreajuste.

4.3 Redes neuronales

Una red neuronal puede definirse como un método que busca aproximar una función que mapea un valor de entrada x a un valor de salida y. Corresponden a una rama de la inteligencia artificial denominada aprendizaje profundo porque la técnica se base en la concatenación de funciones, una dentro de la otra, lo que le otorga la cualidad de abstraer información de los datos que recibe.

4.3.1 Redes neuronales *feed-forward*

A continuación, se muestra la arquitectura de una red neuronal *feed-forward*, también llamadas perceptrones multicapa:

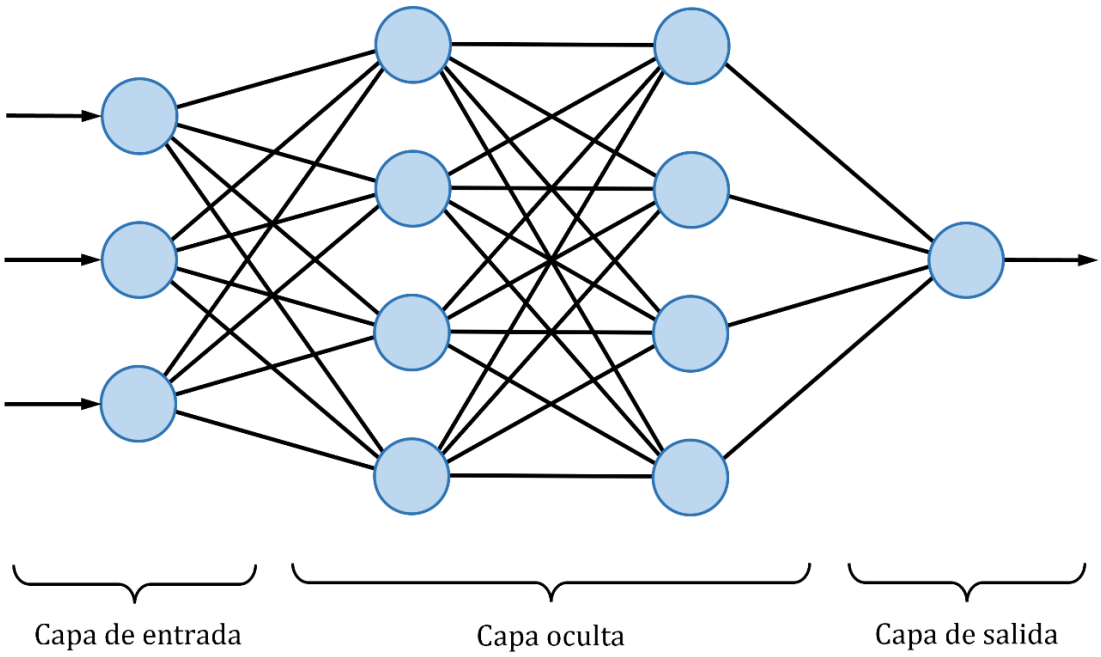


Figura No. (12) Esquema de red neuronal *feed-forward* (perceptrón multicapa).

Los datos ingresan por la capa de entrada y fluyen por toda la red hasta la capa de salida. En cada neurona (círculos azules) se ejecuta una función, denominada función de activación, que tiene como argumento a una regresión lineal:

$$r = f(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k) \tag{19}$$

Puede observarse cómo el *output* de una neurona constituye el *input* de la siguiente, consolidando la idea de concatenación de funciones.

La calidad de la predicción se mide con funciones especiales llamadas funciones de costo. Para problemas de regresión se utiliza como función de costo al error cuadrático. El objetivo es minimizar esta función, ajustando los parámetros de todas las neuronas en el proceso. El algoritmo encargado de llevar a cabo esta optimización es el *backpropagation*, que lo que calcula son los gradientes de la función de costo respecto de cada uno de los parámetros de cada neurona. El valor del gradiente (con signo opuesto) indica la dirección en que debe avanzar el parámetro en la siguiente iteración para conseguir una mejor predicción.

4.3.2 Redes neuronales recurrentes ordinarias (RNN)

Existe otra familia de redes neuronales, llamadas recurrentes, que se diferencian de las explicadas recientemente por el hecho de que, en lugar de avanzar hacia adelante con los datos, las neuronas se retroalimentan del *output* que ellas mismas generan. El esquema a continuación corresponde al de una red neuronal recurrente:

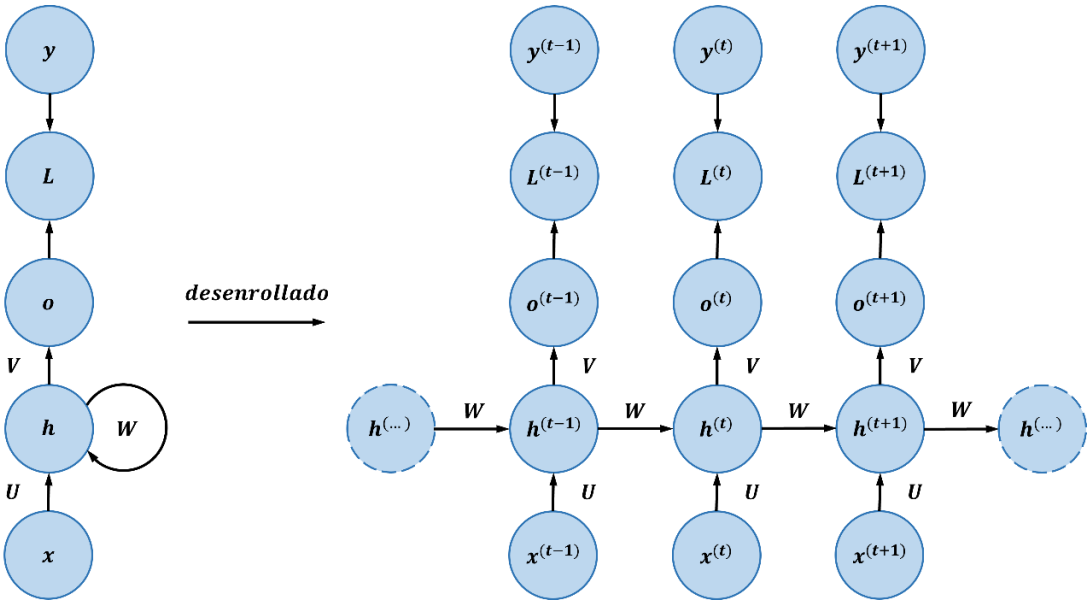


Figura No. (13) Esquema de red neuronal recurrente compacta (izquierda) y desenrollada (derecha).

A la izquierda se muestra la arquitectura simplificada, en donde ingresa una matriz de datos X, se multiplica por una matriz de pesos U, y se la hace pasar por una función de activación. Al resultado de esta operación se lo denomina *hidden state*. Luego, este resultado se multiplica por una matriz de pesos W, cuyo resultado vuelve a ingresar a la misma neurona, para ser multiplicado nuevamente por la misma matriz. Al mismo tiempo que ocurre esto, el *hidden state* se multiplica por una matriz de pesos V y constituye el resultado de la neurona o. Posteriormente, en la neurona L, se calcula la función de costo, en la cual ingresan la predicción y el valor real, para finalmente calcular el gradiente y minimizar el error en la próxima iteración. En lado derecho se muestra la misma arquitectura, pero "desenrollada" o "expandida".

A modo de ejemplo, se muestran las operaciones en cada neurona para una instancia t:

- $a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$ (20)

- $h^{(t)} = tanh\,(a^{(t)})$ (21)

- $o^{(t)} = c + Vh^{(t)}$ (22)

- $\hat{y}^{(t)} = softmax\left(o^{(t)}\right)$ (23)

4.3.3 Redes neuronales recurrentes *Long Short-Term Memory* (LSTM)

Las redes neuronales recurrentes ordinarias enfrentan problemas relacionados con gradientes que desaparecen o explotan durante el entrenamiento. Esto ocurre porque la multiplicación sucesiva por la matriz de pesos Wh puede volverse inestable, llevando a que el gradiente desaparezca durante la retropropagación o se vuelva extremadamente grande de manera incontrolable. Esta inestabilidad es causada por la multiplicación repetitiva con la matriz de pesos recurrentes en diferentes momentos.

Una forma de entender este problema es que una red neuronal que utiliza solo actualizaciones multiplicativas es eficaz para aprender en secuencias cortas, lo que significa que tiene buena memoria a corto plazo, pero dificulta el almacenamiento de información a largo plazo. Para resolver este problema, se utiliza una solución que implica modificar la ecuación recurrente para el vector oculto mediante el uso de LSTM (Memoria a Corto y Largo Plazo). El LSTM se diseñó específicamente para tener un control detallado sobre los datos que se almacenan en la memoria a largo plazo, lo que ayuda a superar las dificultades de aprendizaje a largo plazo en las RNN.

A continuación, se muestra el esquema de una celda LSTM.

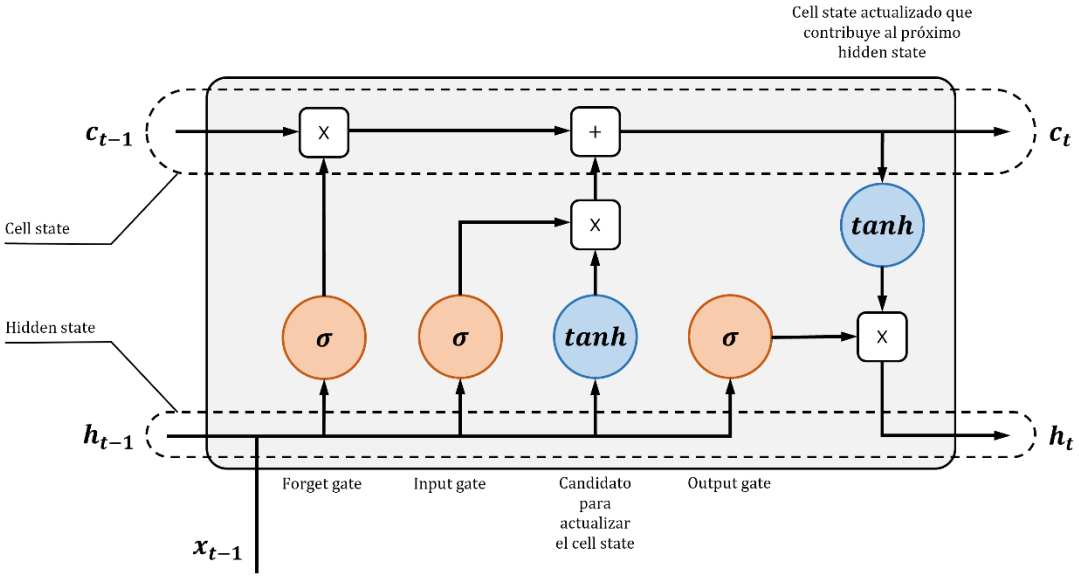


Figura No. (14) Esquema de celda LSTM.

En este caso, las operaciones en cada neurona para una instancia t, son enumeradas a continuación.

- $i^{(t)} = \sigma(b^{(i)} + W^{(i)} x^{(t)} + U^{(i)} h^{(t-1)})$ (24)
- $f^{(t)} = \sigma(b^{(f)} + W^{(f)} x^{(t)} + U^{(f)} h^{(t-1)})$ (25)
- $o^{(t)} = \sigma(b^{(o)} + W^{(o)} x^{(t)} + U^{(o)} h^{(t-1)})$ (26)
- $\tilde{c}^{(t)} = tanh(b^{(c)} + W^{(c)} x^{(t)} + U^{(c)} h^{(t-1)})$ (27)
- $c^{(t)} = f^{(t)} * c^{(t-1)} + i^{(t)} * \tilde{c}^{(t)}$ (28)
- $h^{(t)} = o^{(t)} * \tanh(c^{(t)})$ (29)

En la *input gate*, dado el último estado $h^{(t-1)}$, se evalúa cuánto de la nueva entrada x se incluirá en la memoria a largo plazo $c^{(t)}$. En la *forget gate*, dada la entrada x, se evalúa cuánto del estado de memoria anterior $c^{(t-1)}$ tiene importancia en el nuevo estado. Y en la *output gate*, se evalúa qué parte de estado de la memoria $c^{(t)}$ pasa al próximo estado de memoria $h^{(t)}$.

4.3.4 Redes neuronales *encoder-decoder*

Las arquitecturas *encoder-decoder*, también conocidas como codificador-decodificador, son una clase de modelos que constan de un codificador y un decodificador que trabajan en conjunto para realizar una tarea específica. Se usan usualmente en tareas de procesamiento de secuencias, como la generación de texto y predicción de series de tiempo.

El objetivo principal de las arquitecturas *encoder-decoder* es aprender una representación intermedia o un espacio latente en el *encoder* y luego utilizar esa representación para generar una salida adecuada en el *decoder*. Las RNN y las LSTM son adecuadas para esta etapa de codificación, ya que pueden procesar secuencias de longitud variable y capturar dependencias temporales.

Las RNN son capaces de mantener una memoria interna y propagarla a través de la secuencia. Sin embargo, las RNN tradicionales pueden tener dificultades con gradientes que desaparece, como se explicó anteriormente. Es por ello que también pueden utilizarse las LSTM para una mayor robustez, puesto que éstas incluyen mecanismos de compuerta que les permiten aprender a olvidar o recordar información de manera selectiva.

El *decoder* (decodificador) toma el estado oculto del *encoder* y lo utiliza como entrada para generar la secuencia de salida deseada. En cada paso de tiempo, el *decoder* toma como entrada la salida generada en el paso anterior y el estado oculto actual para generar la siguiente salida. Este proceso se lo conoce como autorregresión, y se puede repetir hasta que se haya generado toda la secuencia deseada.

5 DESARROLLO

5.1 Modelos estadísticos lineales

Antes de estimar un modelo lineal de regresión, es necesario transformar el *dataset* para que se puedan hacer estimaciones sobre variables no numéricas. Se comenzó aplicando el método de codificación denominado *one-hot-encoding*, que consiste en crear una nueva columna binaria por cada categoría. Se decidió codificar las variables CAB y METRO_RT, por ser no numérica, y ANIO y MES, por ser numéricas nominales. El resultado de esta operación aumentó el número de variables de la base de datos, pasando de 14 a 122. Posteriormente, se separó al *dataset* de manera aleatoria en un conjunto de entrenamiento, que guardaba el 70% de los registros, y otro conjunto de testeo, con el restante 30%. Sobre el conjunto de entrenamiento se ajustó un modelo de regresión lineal múltiple, mediante el método de mínimos cuadrados, que incluía todos los covariables del *dataset*.

$$\begin{aligned} PAX\_SEM = {} & METRO\_RT * \beta_{1-97} + CAB * \beta_{98} + ANIO * \beta_{99-102} + MES \\ & * \beta_{103-113} + SEMANAS * \beta_{114} + RPK * \beta_{115} + ASK * \beta_{116} \\ & + CAP * \beta_{117} + LF * \beta_{118} + PAX * \beta_{119} + TM\_SEM * \beta_{120} \\ & + INC * \beta_{121} + TM * \beta_{122} + \beta_0 \end{aligned} \tag{30}$$

Para analizar la calidad del ajuste y de las predicciones, se calcularon tres métricas: el error cuadrático medio, el coeficiente de determinación R^2 y el R^2 ajustado. El error cuadrático medio MSE (por sus siglas en inglés: *mean squared error*) mide la cantidad promedio del error que el modelo comete al predecir la variable objetivo, es definida por la siguiente fórmula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{31}$$

Donde n es la cantidad de observaciones (en este caso son 124.700), y_i son los valores reales de la variable objetivo, y $\hat{y}$ son las predicciones del modelo. La otra métrica es el R^2, que es una medida estadística que se utiliza para medir la calidad de ajuste de un modelo de regresión. Este coeficiente indica cuánta variación de la variable objetivo puede explicarse por las variables predictoras incluidas en el modelo. Es definido por la fórmula:

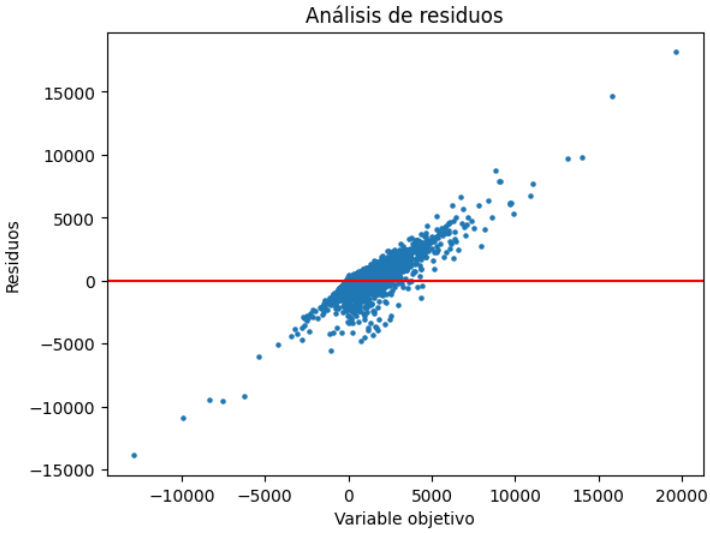
$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \tag{32}$$

Es relación entre la suma de cuadrados debida a la regresión (SSR) y la suma de cuadrados totales (SST). A su vez, la suma de cuadrados debida a la regresión es igual a la diferencia entre la suma de cuadrados totales y la suma de residuos al cuadrado (SSE). Es probable que, en modelos lineales múltiples, el R^2 vaya incrementando a medida que se agregan más variables predictoras, por lo que es recomendable penalizar de alguna manera la adición de estas variables. Es por este motivo que, además de calcular el coeficiente de determinación R^2, se va a calcular el llamado R^2 ajustado, definido como:

$$R^2 ajustado = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1} \tag{33}$$

Siendo R^2 el coeficiente de determinación, n la cantidad de registros y p la cantidad de variables explicativas.

El análisis del modelo va a hacerse sobre el conjunto aislado de testeo. Para ello, se graficaron los residuos en función de los valores reales de la variable objetivo. Si el modelo es bueno, debería poder notarse una nube de puntos alrededor del valor $y = 0$.



MSE	175939.09
R2	0.4797
R2 adj	0.4779

Figura No. (15) [Modelo 1] Residuos en validación del modelo crudo de regresión múltiple.

Puede observase que el ajuste no es bueno, ya que se visualiza una línea recta. Esto indica que el modelo no está siendo capaz de explicar la variable de respuesta de manera correcta. Puede deberse a que faltan variables explicativas en el modelo: variables nuevas o variables transformadas.

Se decidió agregar variables socioeconómicas para ver si tenían la capacidad de explicar la entrada de reservas. Para ello, se recopilaron los siguientes indicadores mensuales, desde 2018 hasta 2023:

- USD: valor de cambio del dólar paralelo. Variable continua que puede adoptar valores de 0 hasta infinito.
- DESOCUP: índice de desocupación. Este indicador sale oficialmente por trimestre, pero se interpoló para tenerlo por mes. Variable continua que puede adoptar valores de 0 hasta 100.
- TASA_EMPLEO: índice de empleo. Variable continua que puede adoptar valores de 0 hasta 100.
- INF: índice de precios al consumidor a nivel general. Variable continua que puede adoptar valores de 0 hasta infinito.
- INF_ACUM: el índice de precios acumulado a partir de 2018. Variable continua que puede adoptar valores de 0 hasta infinito.
- SALARIO_MIN: el salario mínimo real, expresado en dólares. Variable continua que puede adoptar valores de 0 hasta infinito.

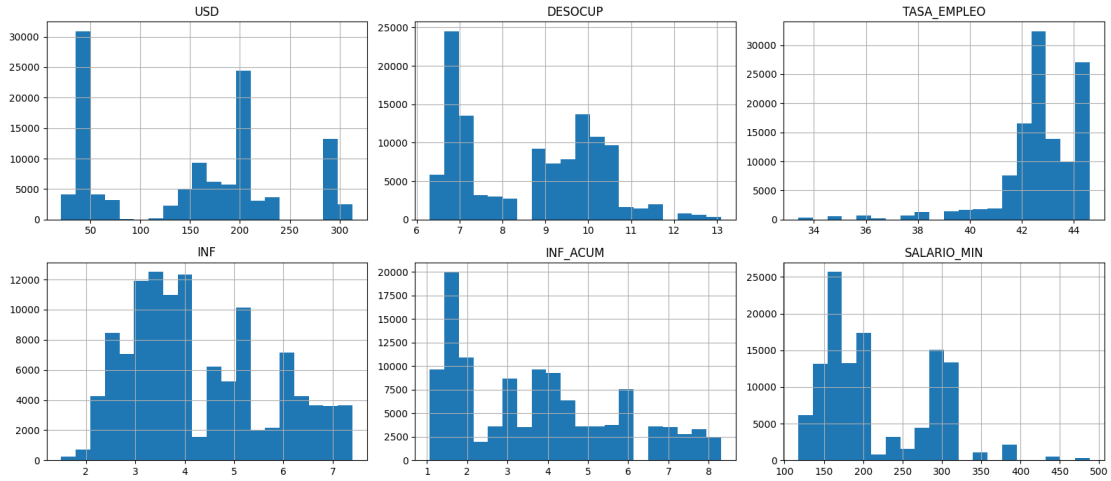
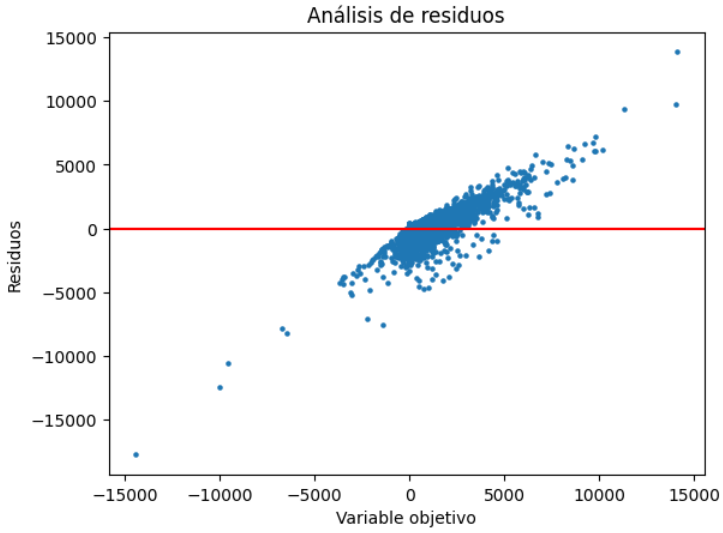


Figura No. (16) Histograma de frecuencias de variables socioeconómicas.

La base de datos tiene registros a partir de enero de 2019. Estos son los vuelos que salían en enero de ese año, pero la venta de boletos estuvo abierta casi un año antes, es por eso que se necesitó de índices de 2018, porque había que hacer coincidir los datos no con el período en que volaban, sino con el período en que se realizaron las ventas.

Una vez agregadas estas seis variables a la base de datos, se volvió a aplicar *one-hot-encoding* y luego se ajustó un modelo de regresión lineal múltiple sobre el mismo conjunto de entrenamiento (mismas observaciones, sólo que ahora tiene más *features*). A continuación, se muestran los residuos.



MSE	171402.82
R2	0.4810
R2 adj	0.4792

Figura No. (17) [Modelo 2] Residuos en validación del modelo de regresión múltiple con variables socioeconómicas.

La relación lineal en los residuos sigue presente, pero los tres indicadores dieron mejores valores: el error cuadrático medio obtuvo un valor más bajo, y el coeficiente de determinación ajustado dio un valor ligeramente más alto. Esto permite suponer que las variables agregadas resultaron ser útiles para explicar la variable objetivo.

Para intentar mejorar el desempeño del modelo, se agregaron más *features*. Como la base de datos tiene una estructura de serie de tiempo, los valores de la semana i están correlacionados con los valores de la semana $i-1, i-2, \ldots$ y así sucesivamente. Entonces adicionaron los valores de 4 semanas anteriores, correspondientes a todas las columnas, separadas por METRO_RT, CAB, ANIO y MES. Esta acción aumentó la cantidad de variables de 20 a 70, y redujo ligeramente las observaciones debido a que, al agregar columnas con observaciones previas, aparecieron valores NaN.

Además, sobre ese nuevo *dataset*, se agregaron más variables transformadas, y variables de interacción. Las transformaciones que se utilizaron fueron: logaritmo natural, raíz cuadrada, cuadrado y cubo. Las primeras tres transformaciones no pueden aplicarse a todas las variables sin hacer algunas modificaciones previas. El argumento del logaritmo no puede ser menor o igual a cero, y la imagen es siempre positiva. Como quería respetarse el signo, se aplicó la operación sobre el valor absoluto y se la multiplicó por la función $signo$. Como la función no está definida para el

valor cero, en los casos en que debía forzarse esta operación, se reemplazó al cero por 0.01. Para los casos de raíz cuadrada y cuadrado se hicieron modificaciones similares para conservar el signo.

En cuanto a las variables de interacción, se tomaron todas las combinaciones posibles de pares de variables y se multiplicaron entre sí (esto sobre el *dataset* de 18 columnas, no incluye las transformaciones).

Las variables CAP y PAX pueden adoptar valores muy grandes, algunas por encima de 100 mil, por lo que para éstas se omitieron las operaciones de interacción y las transformaciones de cuadrado y cubo, para evitar valores excesivamente grandes.

Este algoritmo produjo un *dataset* de 1563 columnas que, luego de aplicar *one-hot-encoding*, resultó en otro de 1662. Si se ajusta un modelo de regresión lineal sobre esta base de datos, es muy probable que el ajuste sea muy pobre y las predicciones sean peor que al principio debido a que el modelo sería muy complejo. Por este motivo es que se aplicó una regularización ridge al modelo, para obtener coeficientes más controlados y evitar el sobreajuste. Si las variables tienen diferentes escalas, este método puede penalizar en mayor medida a las variables con valores más grandes. Es por ello que, antes de correr la regresión ridge, se escalaron las variables. Es importante remarcar que se escalaron únicamente las numéricas continuas y se dejaron inalteradas las variables dicotómicas que produjo la codificación.

Al hiperparámetro λ se le asignaron 100 valores en escala exponencial, desde 10^{-15} hasta 10^{-7}. Con cada uno de esos valores, se ajustó la regresión con todos los datos a excepción de 5 observaciones, que se dejaron intencionalmente afuera para utilizarlas como testeo y medir el error cuadrático medio. Este proceso se repitió tantas veces hasta agotar el *dataset*, cambiando en cada iteración los datos que se apartaban. Finalmente, se promediaron todas las métricas para constituir la *performance* final de ese valor de λ. A este método se lo conoce como *k-fold cross validation.* A continuación, se expone un gráfico que muestra la variación de cada uno de los coeficientes en función del valor del hiperparámetro, mostrando con una línea azul punteada el mejor valor de λ.

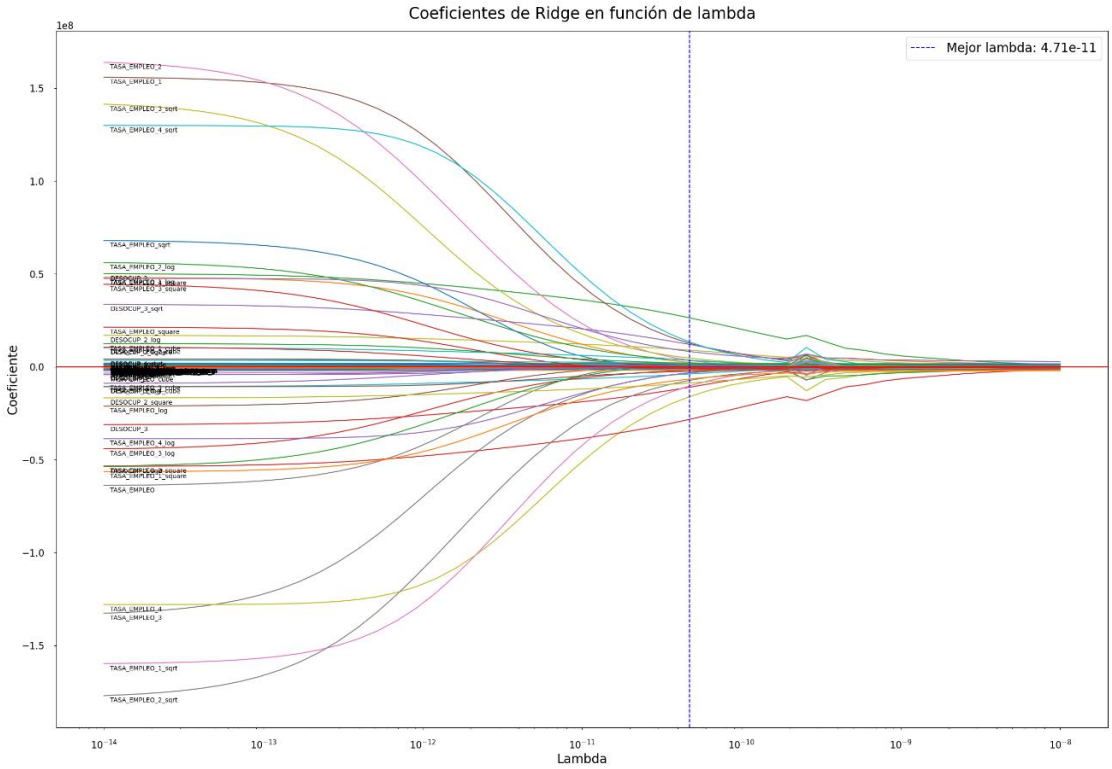


Figura No. (18) [Modelo 3] Variación de las estimaciones de los coeficientes Ridge.

Como las variables fueron escaladas, los coeficientes con mayor valor absoluto son los que el modelo consideró como más importantes. En este caso fueron la tasa de empleo al cuadrado y la desocupación de dos semanas previas. Con este primer modelo regularizado, se obtuvo un error cuadrático medio de 117.176 y coeficiente de determinación ajustado de 0.6568.

Se puede notar una leve mejora en las métricas. Sin embargo, el análisis de residuos continúa afirmando el mal desempeño (ver gráfico en el apéndice). Como ridge no puede llevar a cero los coeficientes, la cantidad de variables no decreció y, por lo tanto, la complejidad del modelo se mantuvo igual.

Posteriormente, se intentó ajustar una regresión regularizada con lasso pero, debido al tamaño del *dataset* y la alta presencia de *outliers*, el algoritmo no logró converger. El algoritmo de ridge suele ser más rápido porque tiene una solución cerrada, es decir que se puede obtener una solución analítica directamente a partir de una fórmula matricial que tiene la misma complejidad que el método de mínimos cuadrados ordinario. Lasso, por otro lado, utiliza una optimización basada en el descenso por coordenadas que es computacionalmente más costosa de calcular que ridge. Existe

otro algoritmo, desarrollado por Efron et al.[4], denominado LARS, que tiene el mismo orden de complejidad que mínimos cuadrados ordinario, y que es especialmente eficiente para conjuntos de datos más grandes. A continuación, se muestra la variación de los coeficientes en función del hiperparámetro lambda, ajustados por el algoritmo LassoLARS.

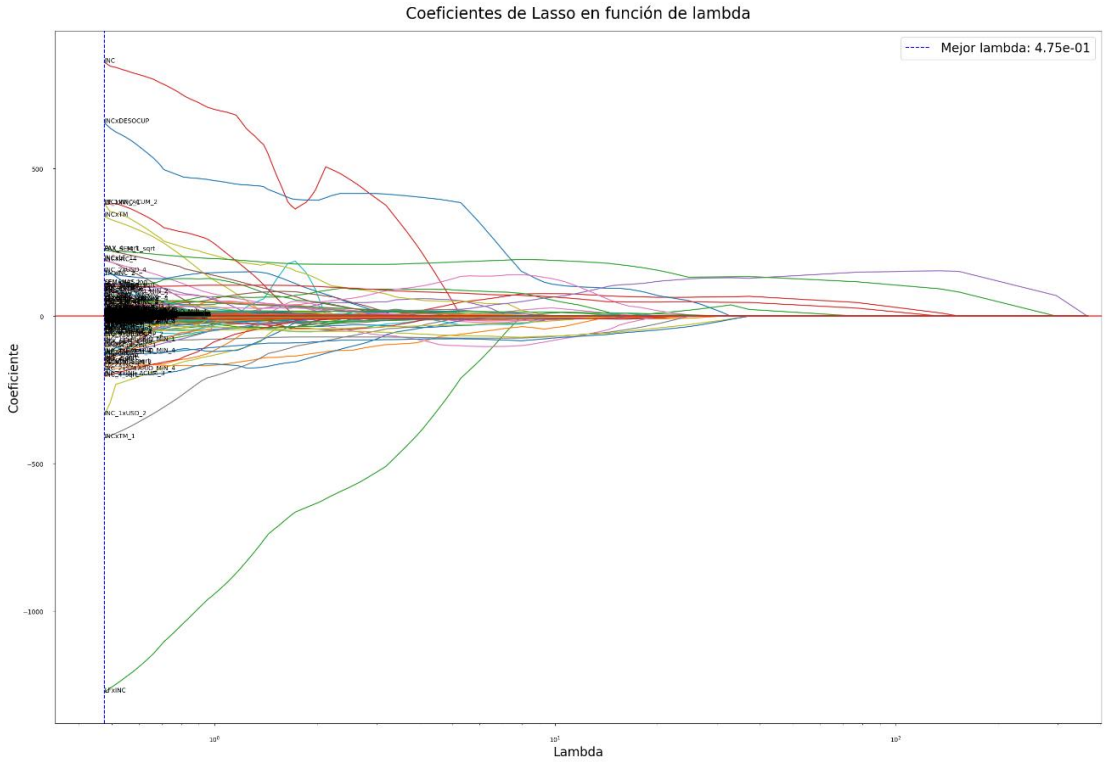


Figura No. (19) [Modelo 4] Variación de las estimaciones de los coeficientes LassoLARS.

El mejor valor del hiperparámetro que controla el encogimiento es de 0.475. Al estar este valor al extremo izquierdo del gráfico, esto permite suponer que es probable que haya otro valor mejor, más pequeño, que minimice aún más el error cuadrático medio. Desafortunadamente, el algoritmo LARS utilizado permite un rápido cálculo de las estimaciones de los coeficientes, pero no tiene flexibilidad para probar distintos valores de lambda, sino que los calcula automáticamente según el *dataset*. Esto es una limitación únicamente para visualizar este gráfico, pero no para encontrar el mejor valor del hiperparámetro, que se halló con otro algoritmo con validación cruzada. El número obtenido fue de 5.21e-5, una estimación más pequeña que la mostrada en el gráfico.

[4] Efron et al. (2004). Least Angle Regression. Annals of Statistics, Vol 32. No 2, pp. 407-499. Institute of Mathematical Statistics.

Lasso, a pesar de tener la cualidad de eliminar variables irrelevantes, sólo retiró 2 de ellas, dejando un modelo apenas menos complejo y consiguiendo un error cuadrático medio de 171.745. En este caso, el modelo no lo hizo mejor que en una regresión múltiple sin regularizar. Además, el ajuste del modelo vino acompañado de un aviso en donde el algoritmo no pudo alcanzar el valor de lamba porque no puede controlarlo adecuadamente, lo que significa que puede estar sobreajustando los datos, por lo que no se puede confiar en este modelo. El punto débil de LARS es que es especialmente sensible a los valores atípicos debido a que se basa en un reajuste iterativo de los residuos.

Lo que está ocurriendo, tanto en ridge como en lasso, es la fuerte presencia de *outliers*. Esto hace que el error cuadrático medio no sea una buena métrica para escoger el hiperparámetro, porque es muy sensible a los valores atípicos. Por lo tanto, es necesario hacer una identificación y tratamiento de *outliers* antes de volver a correr la regresión regularizada.

5.1.1 Tratamiento de outliers

La variable de respuesta PAX_SEM corresponde a las reservas semanales. Es esperable que este número siempre sea positivo porque los vuelos van ocupándose a medida que se aproxima el mes de salida. Es posible que dentro de la semana haya cancelaciones que se computan como valores negativos, pero son pasajeros muy puntuales y, dentro de todas las reservas semanales, éstos son atípicos y no deberían ser mayores que la entrada de reservas como para obtener un resultado negativo. Con este concepto claro, se ordenó la variable de respuesta en orden ascendente para visualizar el *dataset* desde las entradas semanales más chicas hasta las más grandes y se observó que hay varios registros negativos. Se tomó la primera observación para estudiarla, que correspondía a una determinada ruta del mes de mayo del año 2021. A continuación, se muestran los gráficos de entrada de reservas, capacidad y *load factor* en función de las semanas para esta selección.

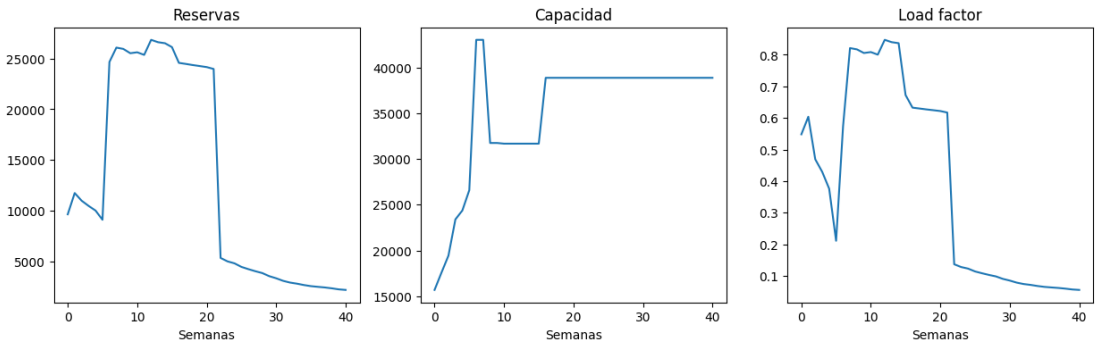


Figura No. (20) Entrada de reservas (izquierda), capacidad (centro) y *load factor* (derecha) en función de las semanas que faltan para finalizar el mes de vuelo.

Se puede ver en el primer gráfico cómo hay un salto en la semana 22, y luego vuelve a caer en la semana 6. Este comportamiento en la ruta sugiere que, por algún motivo, una determinada cantidad de pasajeros se asignó a esta ruta en el sistema, pero antes de volar se retiraron. Lo que está ocurriendo en las semanas 22 y 6 son *outliers* y deben ser tratados para mejorar el desempeño de los modelos. Estos *outliers* afectan la variable *load factor* también, porque ésta última depende de las reservas, puesto que se define como el cociente entre la capacidad y las reservas. Por este motivo, cuando se traten las reservas, también se va a tratar el *load factor*.

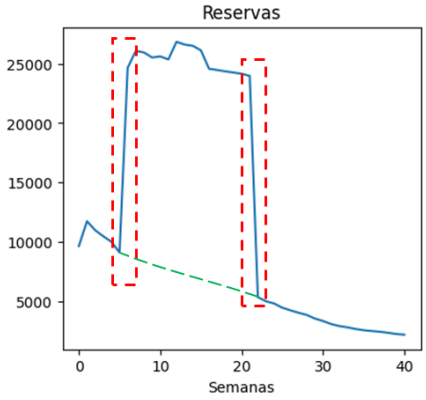


Figura No. (21) Ejemplo de saltos en la entrada de reservas (rojo) y corrección de la curva (verde).

Siguiendo con el ejemplo anterior, se identifican dos saltos, uno con entrada semanal positiva (rectángulo punteado rojo de la derecha) y el otro con entrada semanal negativa (rectángulo punteado rojo de la izquierda). No basta con modificar solo los saltos, porque los valores comprendidos dentro de ese intervalo se ven afectados también, porque es una secuencia, en donde cada valor depende del anterior. El objetivo es recrear la línea punteada verde para que continúe

con la tendencia. Se intentó crear un algoritmo que corrija estos saltos en la evolución de pasajeros, pero no fue muy confiable y se optó por hacer las correcciones manualmente, para asegurar el buen trabajo.

Se identificaron aproximadamente 1300 registros de rutas con este comportamiento. Como la cantidad de rutas era demasiada, se decidió atender a los 200 *outliers* más representativos. No es lo mismo un pequeño salto en la curva de reservas en una ruta con poca capacidad y baja influencia, que un gran salto en una ruta importante con mucha influencia. Para poder determinar qué *outliers* son los más influyentes, se calculó el cociente entre la entrada de reservas semanal atípica (la que era negativa) y la capacidad de la ruta. Esto daba como resultado un valor ajustado que independizaba a las rutas de su volumen, permitiendo que se puedan realizar comparaciones entre ellas. Finalmente, se ordenaron de manera descendente y se corrigieron las 200 primeras. Este número representaba el 90% de la influencia de los *outliers*.

Luego, para pulir un poco más el *dataset*, se eliminaron las observaciones que presentaban entradas de reservas negativas y las que excedían un valor de 5000. Finalmente, sobre el conjunto de datos resultante, se corrió un modelo de *Isolation Forest* para identificar datos anómalos contemplando todas las variables (las entradas de reservas negativas eran *outliers* únicamente del vector PAX_SEM).

Isolation Forest tiene como objetivo identificar patrones en los datos que sean distintos a la mayoría, utilizando el modelo de *Random Forest*. El algoritmo funciona dividiendo repetidamente los datos de manera aleatoria en dos partes a través de la selección de un atributo aleatorio y un corte de división aleatorio en ese atributo. A medida que se divide el conjunto de datos, los elementos anómalos se separarán más rápido que los elementos normales y requerirán menos divisiones para ser aislados. Luego de muchas iteraciones, el algoritmo asigna una puntuación de anormalidad a cada observación en función del número de divisiones necesarias para el análisis. Finalmente, el 5% de las observaciones con puntuaciones más bajas se retiró del *dataset*. Como resultado, el conjunto de datos se redujo un 10%: de 124.699 observaciones a 112.532.

5.1.2 Mejoras

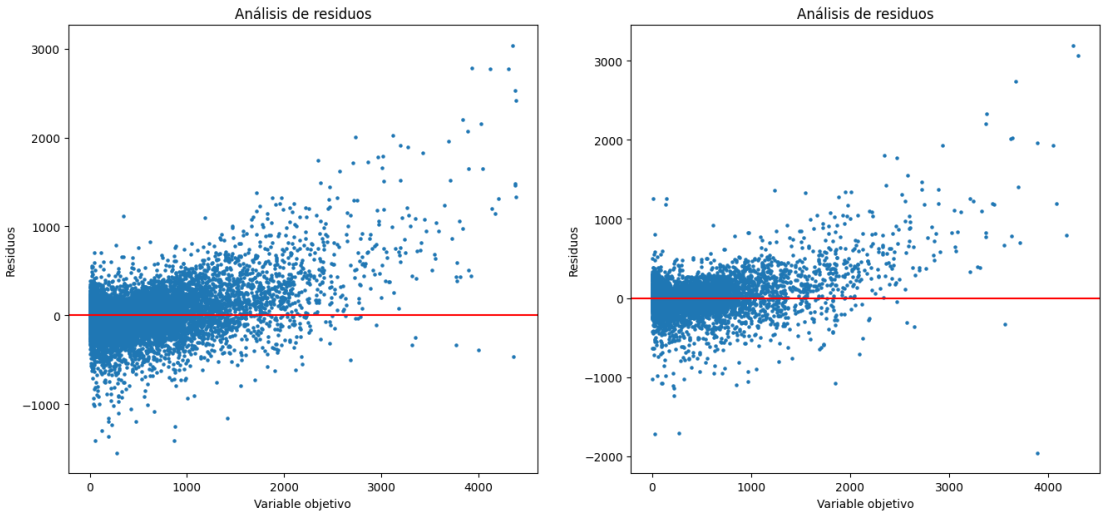
Además de la remoción de datos y del tratamiento de valores atípicos, se propusieron 4 acciones más para mejorar las predicciones:

- Utilización de *target encoder* agrupado para variable CAB.
- Adición de *features* correspondientes a semanas -5 y -6.
- Adición de *features* transformados: se utilizó la tangente hiperbólica sobre todas las variables escaladas.
- Separación de conjunto de entrenamiento y validación balanceados.

La variable CAB se había codificado como *dummy*, con el valor de 1 para la cabina Y, y el valor de 0 para la cabina W. Dentro de un modelo de regresión lineal, el coeficiente asociado a esa variable *dummy* actúa como una constante que se suma al intercepto cuando la variable se "enciende". Existe otro tipo de codificación, denominado *target encoding,* que consiste en promediar la variable objetivo para cada valor de la variable CAB. En ese caso, serían dos promedios: uno para Y y otro para W. Este enfoque es muy simplista, porque asigna el mismo valor de CAB, independientemente a la ruta y al mes, únicamente separa por cabina. Para mejorar la codificación, se calculó un *target encoding* separado por ruta, año, mes y cabina, en lugar de separar solo por cabina, como el método original sugiere. De esta manera, la variable pasó de ser categórica a continua.

También, para lograr capturar una tendencia de más largo plazo, se agregaron los datos en columnas de 2 semanas en atraso más: antes, por observación, se tenía la información de las semanas actual, la -1, -2, -3 y -4, y ahora se agregaron las semanas previas -5 y -6. Asimismo, se agregó una transformación no lineal más al *dataset*: tangente hiperbólica. Esta, además de aplicarse a todas las columnas, también se multiplicó con todas las columnas restantes, a modo de variable de interacción, lo que provocó que se tengan 3110 covariables en total. Finalmente, para lograr una menor diferencia en los resultados entre el conjunto de entrenamiento y el de validación, se separaron estos dos conjuntos, pero balanceados en cantidad de observaciones según ruta, año, mes y cabina: 70% entrenamiento y 30% validación.

Con el nuevo *dataset* transformado se volvieron a correr las regresiones regularizadas. A continuación, se muestra el gráfico de residuos en entrenamiento (a la izquierda) y en validación (a la derecha), correspondiente al modelo regularizado con ridge. El hiperparámetro λ se ajustó con validación cruzada, luego de probar 100 valores diferentes.



MSE	14863.85
R2	0.8636
R2 adj	0.8557

MSE	17694.14
R2	0.8406
R2 adj	0.8172

Figura No. (22) [Modelo 5] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo de regresión múltiple con variables socioeconómicas y tratamiento de *outliers*, regularizado con ridge.

Se puede notar una mejora en las tres métricas, donde disminuyó el error cuadrático medio al mismo tiempo que aumentaron los coeficientes de determinación. Además, el modelo no sobreajustó, puesto que las métricas son muy similares en ambos conjuntos.

También se volvió a ajustar la regresión regularizada con lasso, con el mismo criterio de validación cruzada para hallar el mejor hiperparámetro.

En este caso se obtuvo en validación MSE = 21.333 y, al juzgar por las métricas obtenidas, el desempeño del modelo fue ligeramente inferior al anterior, De las 3110 covariables, el modelo determinó que fueron necesarias únicamente 238 para explicar la entrada de reservas semanal. Al estar escaladas las variables, los coeficientes de mayor valor absoluto corresponden a las variables más explicativas. Por lo tanto, luego de ordenarlas por importancia, se seleccionaron las 10 primeras, ordenadas de mayor a menor relevancia.

INC
LFxINC
INC_sqrt
PAX_2_sqrt
INC_2_sqrt
PAX_sqrt
PAX_SEM_1_sqrt
INCxTM
INCxDESOCUP
INC_1_sqrt

Tabla No. (03) Las diez variables más relevantes.

Al parecer, la variable más determinante a la hora de explicar la entrada de reservas es INC, o sea los ingresos totales, que ocupan, además, 7 de estos 10 predictores.

Finalmente, se intentó con un modelo de redes elásticas, que contempla ambas regularizaciones en simultáneo. En este caso, los hiperparámetros son dos: el λ, que controla el encogimiento, y el α, que controla la proporción de regularización L1. Para hallar la mejor combinación de pares de valores, se hizo una búsqueda bayesiana que minimizaba la función de costo de la red elástica. Este método de búsqueda de parámetros es menos costoso computacionalmente que *grid search*, puesto que no es un método exhaustivo que prueba todas las combinaciones posibles de hiperparámetros, sino que se basa en un enfoque probabilístico que modela la distribución posterior de éstos y selecciona la siguiente combinación que optimiza la función objetivo. En este caso en particular, *grid search* hubiese iterado entre 2000 combinaciones de hiperparámetros (100 lambdas y 20 *alphas*), mientras que *bayesian search* sólo requirió 100 iteraciones.

Luego de la búsqueda, se concluyó que los mejores valores eran $\lambda = 7.56e - 29$ y $\alpha = 0.97$, los que consiguieron un error en entrenamiento de 14.454 y en validación de 19.120.

De las 3 regularizaciones, ésta fue la que menor error en entrenamiento tuvo. Sin embargo, respecto de ridge lo hizo peor en validación, lo que sugiere que hubo un mayor sobreajuste.

5.1.3 Regresión ponderada

Las alternativas propuestas hasta el momento mejoraron el desempeño de los modelos, pero no lo suficiente como para fiarse del mismo. La regresión lineal asume que los errores son independientes y se distribuyen idénticamente bajo una distribución normal. Este fenómeno es conocido como homocedasticidad y puede ser un problema para ajustar el modelo. La ausencia de homocedasticidad se conoce heterocedasticidad, y tiene consecuencias no deseables. Por ejemplo, el modelo no tendrá el menor error cuadrático posible; además, la estimación de coeficientes, junto con los errores estándar, serán poco precisos, dificultando la inferencia.

Un método para ajustar un modelo en presencia de heterocedasticidad es recurrir a una generalización de la regresión lineal, en donde se introduce la matriz de covarianza de los errores (que antes se suponía constante) a modo de ponderación para generar varianza constante en los residuos. Idealmente, la ponderación correcta es el recíproco de la varianza del error. Como dicha distribución es desconocida, se puede estimar a través de los residuos del mejor modelo que se tiene hasta ahora: el modelo de ridge.

Luego de ajustada la regresión ponderada, se notó que los cambios no fueron grandes respecto del modelo base de ridge: apenas se puede notar una leve mejora en el conjunto de validación, que mejoró de 17.694 a 16.861.

5.1.4 Regresión de componentes principales

La regresión de componentes principales consiste en ajustar una regresión lineal sobre una nueva matriz de características que surge de una transformación lineal de las variables originales a una nueva base, cuyos vectores base no estén correlacionados y que mayor varianza posible expliquen. El caso ideal en un modelo de regresión es que las variables independientes sean, justamente, independientes. Se sabe que la gran mayoría de los predictores están correlacionados, puesto que surgieron de una operación entre pares de ellos (variables de interacción), o de transformaciones no lineales.

Para entender de manera más analítica cuán correlacionadas estás las variables, se va a calcular el VIF (*variance inflation factor*) con la siguiente fórmula.

$$VIF_i = \frac{1}{1 - R_i^2} \tag{34}$$

El VIF es una medida estadística que se utiliza para evaluar la muticolinealidad entre las variables en un modelo de regresión múltiple. Se van a considerar variables altamente correlacionas las que muestren valores mayores a 10. A continuación, se muestran las métricas para las primeras 15 variables, a modo de ejemplo.

FEATURE	VIF
CAB	2.54
SEMANAS	9809.41
LF	19.81
TM_SEM	17.81
INC	26.50
TM	13.80
USD	712.25
DESOCUP	151.43
TASA_EMPLEO	125.61
INF	14.31
INF_ACUM	487.62
SALARIO_MIN	117.62
LF_1	35.30
LF_2	28.80
LF_3	28.04

Tabla No. (04) VIF para las primeras 15 variables.

Como se puede notar, en la gran mayoría de las variables existe muticolinealidad. Este método de regresión es una alternativa para ajustar una regresión cuando este fenómeno está presente, pero no afecta a las predicciones, sino a la interpretación de los coeficientes a la hora de hacer inferencia. De todas maneras, a pesar de no haber podido superar el primer supuesto de linealidad, se va a intentar ajustar una regresión con esta técnica y observar los resultados.

Del *dataset*, que contiene 3110 covariables, se van a considerar las primeras 200 componentes que acumulan el 99.45% de la varianza explicada. A continuación, se muestra el gráfico de varianza acumulada en función de la cantidad de componentes principales.

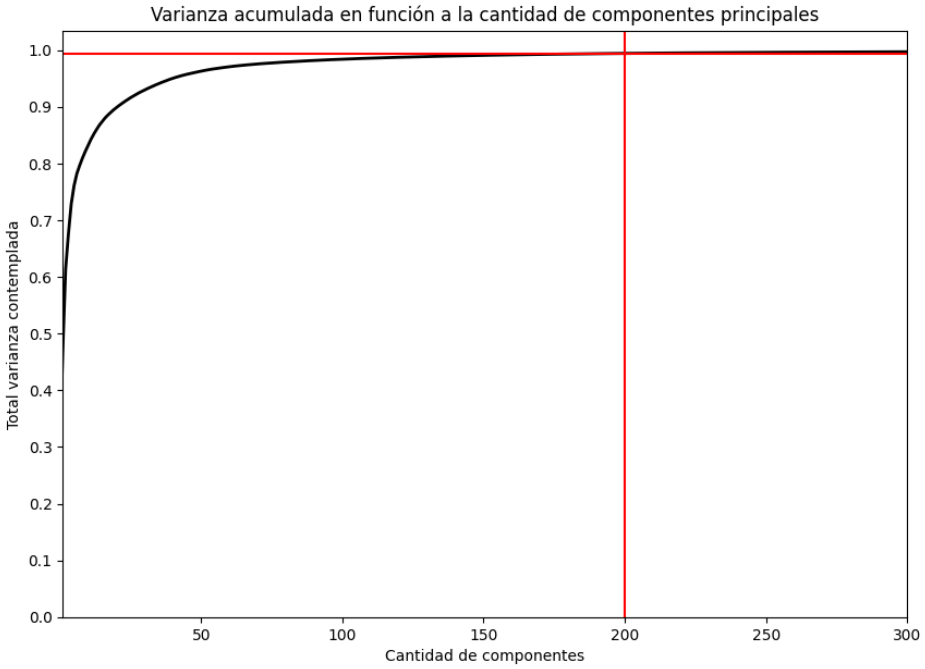


Figura No. (23) [Modelo 9] Varianza acumulada en función de la cantidad de componentes principales contempladas.

El gráfico muestra hasta 300 componentes en el eje de las abscisas porque la varianza alcanza prácticamente el valor unitario para valores mayores. Se puede ver que, incluso tomando 100 componentes, la pérdida de varianza respecto de la capturada con 200 es apenas perceptible.
Se calcularon, entonces, las primeras 200 componentes principales y se ajustó una regresión lineal regularizada con ridge. Además, se ponderaron las observaciones según los residuos. Con esta combinación de técnicas, se obtuvo un error en validación de 28.166, que no alcanzó para mejorar las regresiones regularizadas con las covariables originales.

5.1.5 Regresión de mínimos cuadrados parciales

Este método es similar al anterior en la medida de que ambos consisten en una reducción de dimensionalidad de la matriz de regresión, con la diferencia de que el caso de regresión de componentes principales no considera la variable de respuesta a la hora de calcular los nuevos

vectores base. De esta manera, la regresión de mínimos cuadrados parciales suele modelar el comportamiento de la variable objetivo con menos componentes.

Para poder comprar este método con el de regresión de componentes principales, se van a considerar 200 componentes también, regularizadas con ridge y ponderadas las observaciones de la misma manera.

De todos los métodos, éste fue el que mejor resultados arrojó, en donde se alcanzó un error en validación de 13.892. Se esperaba un desempeño superior al método anterior porque PLS puede capturar información relevante para la relación entre las variables predictoras y la variable de respuesta, aunque no necesariamente captura toda la varianza en los datos. La información es la relación o estructura en los datos que es relevante para la variable de respuesta. PLS busca encontrar las componentes de las variables predictoras que estén altamente correlacionadas con la variable de respuesta, lo que implica que la información relevante para predecir la variable objetivo se está capturando.

Es importante remarcar que la varianza no siempre captura toda la información relevante para el problema en cuestión, es por eso que, en este caso, PLS tuvo mejores resultados que PCR.

5.1.6 Comparación de modelos

A continuación, se muestra una tabla que resume los resultados de los modelos lineales entrenados.

N	TIPO	MODELO	MSE	R2	R2 adj
1	Lineal	Múltiple	175939,09	0,4797	0,4779
2	Lineal	Múltiple + vars. socioeconómicas	171402,82	0,4810	0,4792
3	Lineal	Múltiple + vars. socioeconómicas + lasso	117176,44	0,6754	0,6568
4	Lineal	Múltiple + vars. socioeconómicas + ridge	171745,52	0,4800	0,4781
5	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + ridge	17694,14	0,8406	0,8172
6	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + lasso	21333,55	0,8078	0,7796
7	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + enets	19120,25	0,8153	0,7894
8	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + ponderada + ridge	16861,97	0,8466	0,8251
9	Lineal	PCR + vars. socioeconómicas + trat. outliers + ponderada + ridge	28166,49	0,7279	0,7257
10	Lineal	PLS + vars. socioeconómicas + trat. outliers + ponderada + ridge	13892,98	0,8588	0,8577

Tabla No. (05) Resumen de resultados de cada modelo entrenado (Lineal).

5.1.7 Conclusiones de modelos estadísticos lineales

Ninguna de estas alternativas es un buen modelo, puesto que todas fallan el primer supuesto de la regresión lineal, que asume linealidad en los parámetros. Esto es evidente al observar el gráfico de

residuos, en donde se percibe una tendencia lineal: aumentan los residuos para entradas de reservas mayores. Esto lo que sugiere es que faltan predictores que logren explicar ese crecimiento en los residuos; o que existen valores atípicos que influencian demasiado el ajuste de la regresión. Como ya se analizaron y trataron los *outliers*, y se agregaron múltiples variables al *dataset*, se concluye que un modelo de regresión lineal no tiene la capacidad de explicar el comportamiento de la variable de respuesta, por lo que es necesario cambiar el enfoque y probar otras alternativas. Dicho esto, al no poder superar el supuesto más importante, se descarta con ello la posibilidad de realizar inferencia, como test de hipótesis para la interpretabilidad de coeficientes y la creación de intervalos de confianza para las predicciones.

5.2 Modelos basados en árboles

Los modelos lineales no lograron capturar adecuadamente la complejidad de la relación entre las variables predictoras y la variable de respuesta. En esta sección, se van a entrenar modelos basados en árboles que pueden ser más adecuados para capturar relaciones no lineales o interacciones entre las variables predictoras.

Estos modelos son capaces de captar patrones no lineales en los datos mediante la división del espacio de características en regiones más pequeñas y manejables. Además, estos suelen ser menos sensibles a valores atípicos y errores en los datos, lo que puede mejorar la precisión de las predicciones. Sin embargo, los modelos basados en árboles pueden ser propensos al sobreajuste, especialmente si se construyen árboles profundos, con muchos nodos y hojas.

5.2.1 Random forest

Se comenzó utilizando un *Random Forest*. *Random Forest* es un modelo de aprendizaje automático que utiliza múltiples árboles de decisión para realizar predicciones. En lugar de confiar en un solo árbol, el modelo combina la salida de muchos árboles para mejorar la precisión y reducir el riesgo de sobreajuste.

El modelo cuenta con varios hiperparámetros configurables, dentro de los cuales se encuentran:

- N_ESTIMATORS: número de árboles en el bosque. Un valor más alto puedo mejorar la precisión, pero también aumenta el costo computacional.

- MAX_DEPTH: profundidad máxima del árbol. Árboles más profundos pueden mejorar las precisiones, pero también puede haber riesgo de sobreajuste.
- MIN_SAMPLE_SPLIT: número mínimo de muestras necesarias para dividir un nodo interno. Controla la cantidad de divisiones y, por lo tanto, el nivel de complejidad del modelo.
- MIN_SAMPLE_LEAF: controla el número mínimo de muestras que deben estar en una hoja del árbol de decisión.
- MAX_FEATURES: número máximo de características que se consideran para cada división. Controla la variabilidad del modelo.

Para este primer modelo, se eligió:

- n_estimators = 100
- max_depth = 6
- min_sample_split = 2
- min_sample_leaf = 2
- max_features = raíz cuadrada de la cantidad de predictores

Como resultado, se obtuvo un error cuadrático medio en el conjunto de entrenamiento de 40.320. El desempeño del modelo fue pobre, y el efecto de la dependencia de los errores con la variable objetivo fue más notorio.

Para mejorar el desempeño, se hizo una búsqueda de parámetros bayesiana, en donde se ajustaron 50 bosques con distintos valores de hiperparámetros y se escogió el modelo que minimizaba el error en validación. Los mejores hiperparámetros fueron:

- n_estimators = 100
- max_depth = 15
- min_sample_split = 3
- min_sample_leaf = 2
- max_features = raíz cuadrada

lo que arrojó un MSE = 15.786.

5.2.2 XGBoost

Posteriormente, se entrenó un modelo *XGBoost*, que es un algoritmo de aprendizaje automático basado en árboles de decisión con un enfoque *boosting*.

El modelo cuenta con varios hiperparámetros configurables, dentro de los cuales se encuentran:

- N_ESTIMATORS: número de árboles. Un valor más alto puedo mejorar la precisión, pero también aumenta el costo computacional.
- MAX_DEPTH: profundidad máxima del árbol. Árboles más profundos pueden mejorar las precisiones, pero también puede haber riesgo de sobreajuste.
- ALPHA: parámetro de regularización L1 (lasso).
- LAMBDA: parámetro de regularización L2 (ridge).
- LEARNING_RATE: controla la tasa de aprendizaje del modelo.
- COLSAMPLE_BY_TREE: controla la fracción de columnas que se seleccionan aleatoriamente para cada árbol. Un valor más bajo significa que se seleccionan menos columnas.

Se comenzó entrenando un modelo con los hiperparámetros:

- n_estimators = 100
- max_depth = 6
- alpha = 10
- lambda = 1
- learning_rate = 0.10
- colsample_bytree = 0.80

En este primer intento se consiguió un error cuadrático medio de 19.465 en el conjunto de validación. El modelo se puede mejorar si se escogen los valores de hiperparámetros adecuados. Para ello, se ejecutó una búsqueda bayesiana que entrenaba modelos con diferentes valores. Se logró alcanzar un error en validación de 11.570 correspondiente a un modelo cuyos parámetros fueron:

- n_estimators = 961
- max_depth = 3
- alpha = 7.13
- lambda = 3.77
- learning_rate = 0.17
- colsample_bytree = 0.80

Entrenar un solo modelo con un *dataset* tan grande requirió de mucho tiempo. Para la búsqueda de parámetros fue necesario entrenar 30 modelos, lo que implicó un tiempo total de 12 horas aproximadamente. Como lo que se desea obtener es un modelo denominado *eager*, los tiempos de entrenamiento no suponen un problema porque el modelo se entrena una vez al principio, y las predicciones se realizan rápidamente. Si, en cambio, se estaría pensando en un modelo *lazy*, como estos aplazan el procesamiento y la toma de decisiones hasta el momento de la predicción, las 12 horas de entrenamiento serían un inconveniente.

5.2.3 Comparación de modelos

A continuación, se muestra una tabla que resume los resultados de cada modelo entrenado hasta el momento.

N	TIPO	MODELO	MSE	R2	R2 adj
1	Lineal	Múltiple	175939,09	0,4797	0,4779
2	Lineal	Múltiple + vars. socioeconómicas	171402,82	0,4810	0,4792
3	Lineal	Múltiple + vars. socioeconómicas + lasso	117176,44	0,6754	0,6568
4	Lineal	Múltiple + vars. socioeconómicas + ridge	171745,52	0,4800	0,4781
5	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + ridge	17694,14	0,8406	0,8172
6	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + lasso	21333,55	0,8078	0,7796
7	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + enets	19120,25	0,8153	0,7894
8	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + ponderada + ridge	16861,97	0,8466	0,8251
9	Lineal	PCR + vars. socioeconómicas + trat. outliers + ponderada + ridge	28166,49	0,7279	0,7257
10	Lineal	PLS + vars. socioeconómicas + trat. outliers + ponderada + ridge	13892,98	0,8588	0,8577
11	Árboles	Random forest	40320,26	0,7422	0,7403
12	Árboles	Random forest + *bayesian search*	15786,47	0,8464	0,8250
13	Árboles	XGBoost	19465,61	0,8756	0,8746
14	Árboles	XGBoost + *bayesian search*	11739,99	0,8858	0,8698

Tabla No. (06) Resumen de resultados de cada modelo entrenado (Lineal + Árboles).

5.2.4 Conclusiones acerca de modelos basados en árboles

De todo los probado hasta el momento, el modelo que menor error consiguió fue el de *XGBoost*, tanto en entrenamiento como en validación (gráficos en apéndice). Sin embargo, al analizar los residuos, estos siguen altos, y continúan presentando dependencia con la variable objetivo. Si bien éste no es un supuesto a validar para poder ajustar un modelo de regresión basado en árboles, es importante tenerlo en cuenta para poder determinar la fiabilidad de las predicciones.

5.3 Modelos de redes neuronales

Las predicciones no fueron buenas para ninguna de las técnicas mostradas anteriormente. Los modelos lineales no lograron explicar la entrada semanal como una combinación lineal de los atributos del *dataset*; y los modelos basados en árboles, que proponen relaciones no lineales, tampoco arrojaron buenas predicciones en el conjunto de validación. Los modelos que se van a explicar en esta sección son no lineales, y tienen la capacidad de refinar la información progresivamente a medida que los datos pasan por las diferentes capas de la red. Además, este tipo de modelos trabajan muy bien con los *embeddings*; que no es más que otra forma de codificación de datos. Más específicamente, los *embeddings* son una representación numérica de una categoría en un espacio n-dimensional, siendo n un hiperparámetro a configurar. Los *embeddings* son especialmente eficaces para codificar variables categóricas con muchos niveles, es por eso que van a utilizarse para codificar las rutas.

Las redes neuronales pueden ser muy costosas en términos de cálculo computacional, especialmente cuando se ejecutan en la CPU. Esto se debe a que la CPU está diseñada para realizar una amplia variedad de tareas y no está optimizada específicamente para el procesamiento paralelo intensivo computacional, que es necesario para el entrenamiento y la ejecución de redes neuronales. La GPU, por otro lado, al contar con múltiples núcleos de procesamiento, muchos más que la CPU, puede realizar muchas más operaciones en paralelo y reducir significativamente los tiempos de cómputo. En este trabajo se utilizó una tarjeta gráfica NVIDIA GeForce RTX 2070 Max-Q para agilizar los cálculos, lo que disminuyó los tiempos hasta 10 veces menos.

5.3.1 Red neuronal *feed-forward*

Se diseñó una red neuronal profunda de 2 capas ocultas, con 800 y 400 neuronas respectivamente. La capa de entrada recibe vectores de 2932 dimensiones, que resultan de concatenar el vector de la matriz de datos de 2924 dimensiones y el *embedding* de 8 dimensiones; y la capa de salida produce un único valor, que es la predicción. A continuación, se muestra un esquema de la arquitectura propuesta.

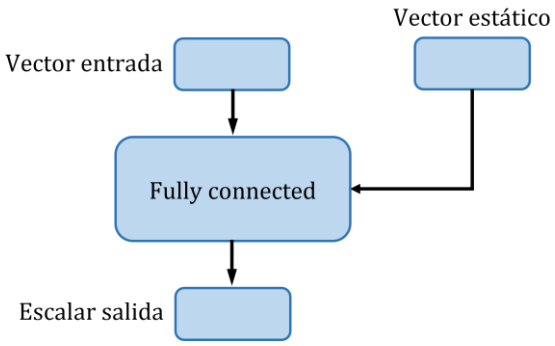


Figura No. (24) Esquema de red neuronal *fully-connected*.

Al final de cada capa oculta, se utilizó una función de activación ReLU y una capa de regularización *dropout*, con probabilidad 20%.

A continuación, se muestran dos tablas con detalles de la arquitectura de la red y la cantidad de parámetros entrenables.

Módulo	Parámetros
embedding	num_emb = 85, emb_dim = 8
lineal_1	in = 2924 + 8, out = 800, bias = True
relu_1	
dropout_1	prob = 0.20
lineal_2	in = 800, out = 400, bias = True
relu_2	
dropout_2	prob = 0.20
lineal_3	in = 400, out = 1, bias = True

Tabla No. (07) [Modelo 15] Arquitectura y parámetros de red neuronal *feed-forward*.

Módulo	Parámetros
embedding.weight	680
linear_1.weight	2.345.600
linear_1.bias	800
linear_2.weight	320.000
linear_2.bias	400
linear_3.weight	400
linear_3.bias	1
Total parám. entrenables	**2.667.881**

Tabla No. (08) [Modelo 15] Cantidad de parámetros entrenables de red neuronal *feed-forward*.

La cantidad de parámetros en cada módulo se calcula como la cantidad de valores que ingresan multiplicado por las neuronas de dicha capa, más la cantidad de neuronas (*bias*).

Para crear los *embeddings*, a cada ruta se le asignó un número natural a modo de índice. Estos números son mapeados al conjunto de parámetros n-dimensional (en este caso de configuró en 8 dimensiones), los cuales se inician con números aleatorios. Durante el entrenamiento, los valores de los vectores *embedding* se ajustan para minimizar la función de pérdida del error cuadrático medio.

Para entrenar el modelo, la ingesta de datos se hizo a través de un *dataloader*. Ésta es una técnica para automatizar la carga de datos en lotes (*batches*) que tiene varias ventajas en comparación con cargar todo el conjunto de entrenamiento de un vez. Por ejemplo, al dividir en lotes los registros, esto ahorra espacio en la memoria RAM, permitiendo que se puedan trabajar con cantidades muy grandes. Además, el rendimiento del entrenamiento mejora, puesto que los gradientes se calculan para el lote completo, en lugar de para la muestra individual. Por otro lado, el *dataloader* permite mezclar los datos aleatoriamente en cada época, evitando el sesgo.

Para este primer modelo, se decidió utilizar un tamaño de lote de 64; el algoritmo de optimización AdAM (*Adaptative Moment Estimaton*), que es el encargado de calcular los gradientes y hallar los mejores parámetros; una tasa de aprendizaje (*learning rate*) de 0.001; y 200 épocas. A lo largo de todas las iteraciones (por lotes y épocas), se calculan los gradientes, que es la dirección en donde el error aumenta. Por lo tanto, los próximos parámetros avanzarán en la dirección opuesta al gradiente, para disminuir el error. La velocidad con la que se modifican estos parámetros es la tasa de aprendizaje. Elegir un número adecuado para este hiperparámetro es muy importante ya que, si se escoge un número muy grande, el algoritmo de optimización puede oscilar alrededor de la función de pérdida y no converger nunca y, por el contrario, si se elige un valor muy pequeño, el algoritmo puede tardar mucho tiempo en converger, o incluso nunca hacerlo. Es recomendable trabajar con una tasa de aprendizaje variable, comenzando con un valor grande y, a medida que la red aprende, ir disminuyendo la velocidad de avance. Dicho esto, además de establecer un valor de tasa de aprendizaje, se va a establecer otro hiperparámetro, denominado gamma, que controle la velocidad. El valor que se utilizó es de 0.20 cada 20 épocas.

Para evitar el sobreajuste, se utilizó la técnica *dropout*, como se mencionó al principio. Ésta tiene la función de reducir la dependencia de las unidades de la red neuronal, evitando así que las neuronas de especialicen demasiado en el conjunto de entrenamiento. La técnica *dropout* implica la eliminación aleatoria de algunas unidades de la red neuronal durante el entrenamiento. Esto se

logra mediante la multiplicación de la salida de cada unidad por una máscara binaria aleatoria que es igual a cero en algunos elementos, y unos en otros. Pero existe otra técnica de regularización denominada *weight decay*. Ésta se utiliza para penalizar los pesos grandes de la red neuronal y evitar que la red se adapte demasiado al conjunto de entrenamiento. Esta técnica es exactamente igual a ridge, puesto que agrega un término adicional a la función de costo basado en la norma L2. *Dropout* solicita un hiperparámetro *p* que controla la probabilidad de 'apagar' neuronas, y *weight decay* se configura con un valor que controle el encogimiento de los hiperparámetros (lambda en ridge). En este trabajo se utilizaron ambas regularizaciones en simultáneo, estableciendo p = 0.20 para *dropout* y 0.0001 para *weight decay*.

A continuación, se muestra la evolución de los errores en los conjuntos de entrenamiento y validación a lo largo de las 200 épocas.

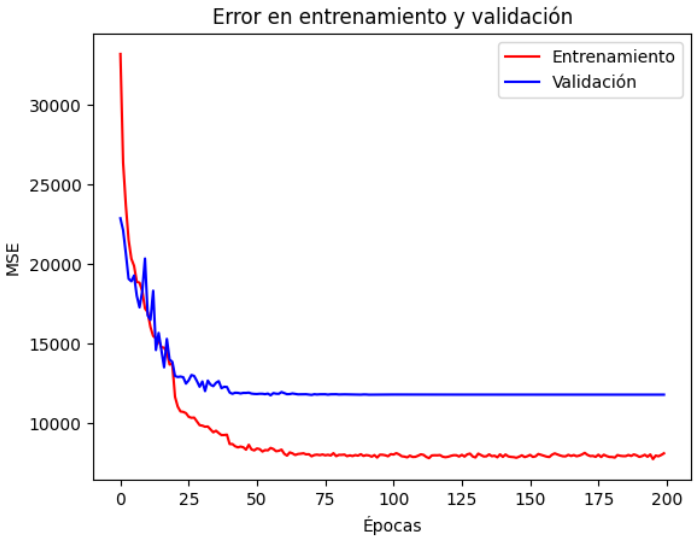
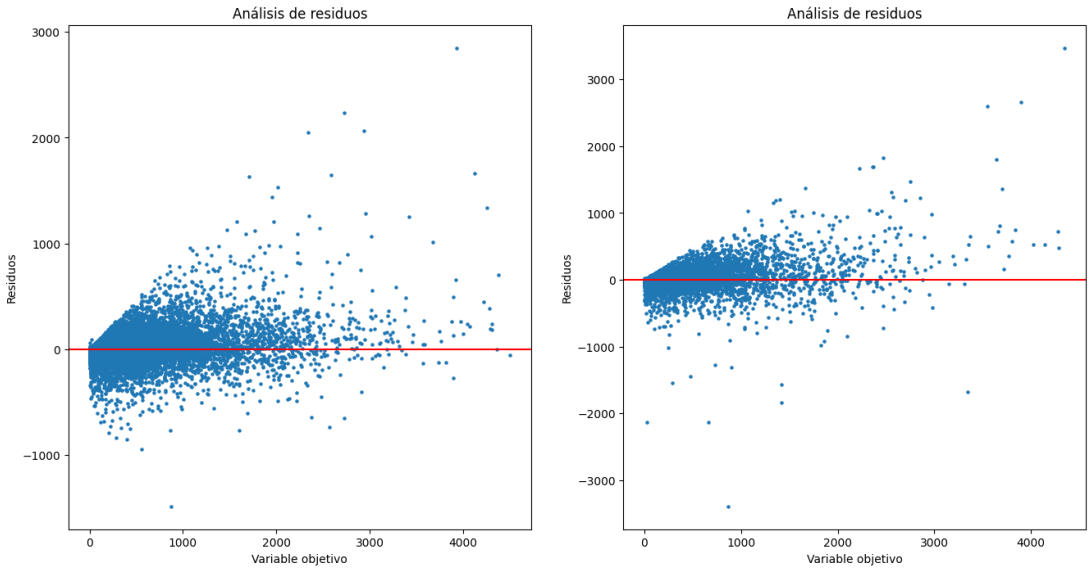


Figura No. (25) [Modelo 15] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales *feed-forward*.



MSE	6577.02
R2	0.9404
R2 adj	0.9370

MSE	11755.62
R2	0.8878
R2 adj	0.8734

Figura No. (26) [Modelo 15] Residuos en entrenamiento (izquierda) y validación (derecha) del primer modelo de redes neuronales *feed-forward*.

Se puede ver que a partir de la época 50 aproximadamente, el modelo deja de mejorar en ambos conjuntos, alcanzando un MSE = 11.755 en el conjunto de validación.

Se entrenaron dos modelos más, pero más profundos y complejos, es decir, con más capas ocultas y con más cantidad de neuronas. Los detalles de la arquitectura y los resultados del entrenamiento se pueden encontrar en el apéndice. Ninguno de estos dos modelos lo hizo mejor que el anterior, que reviste menor complejidad. Los MSE en el conjunto de validación dieron 12.617 en el primero, y 13.070 en el segundo.

5.3.2 Red neuronal recurrente

La entrada de reservas, que es lo que se intenta modelar, posee un comportamiento que varía a lo largo del tiempo (semanalmente en este caso). Esto implica que la entrada de reservas acumulada hasta la semana i guarda correlación con las reservas acumuladas hasta la semana $i-1$, y ésta, a su vez, con la semana anterior, y así sucesivamente. Esta naturaleza secuencial se había tenido en cuenta en todos los modelos anteriores, cuando se agregaron variables con datos de 6 semanas atrás. Sin embargo, el *dataset* seguía conservando una estructura estática. Para poder analizar

secuencias de datos es necesario utilizar otro tipo de arquitectura de redes neuronales, denominadas red neuronal recurrentes. Estas tienen la capacidad de modelar relaciones de dependencia temporal entre los elementos de una secuencia, y esto las hace especialmente útiles en problemas donde el orden y la temporalidad de los datos son importantes.

Lo primero que hay que hacer es cambiar la estructura de los datos. El modelo de red neuronal *feed-forward* recibía como *input* un tensor de dimensiones:

$$(tamaño\ de\ lote, cantidad\ de\ variables)$$

Las redes neuronales recurrentes reciben, en cambio, un tensor de tamaño:

$$(tamaño\ de\ lote, longitud\ de\ secuencia, cantidad\ de\ variables)$$

La longitud de la secuencia es el número de elementos en una secuencia de entrada que se procesa a la vez. Dicho de otra manera, en lugar de alimentar una entrada completa en la red de una sola vez, se divide en una serie de elementos o "pasos de tiempo" y se procesa uno por uno. La longitud de secuencia es un hiperparámetro importante a considerar al entrenar un modelo de red neuronal recurrente, ya que puede afectar el rendimiento y la capacidad del modelo para capturar patrones a largo plazo en la secuencia de entrada. Una longitud de secuencia demasiado corta puede no capturar suficiente información temporal, mientras que una longitud de secuencia demasiado larga puede aumentar la complejidad del modelo y hacer que sea más difícil de entrenar.

La secuencia de datos de cada elemento de lote debe ser homogénea. Esto quiere decir que no pueden mezclarse registros de diferentes rutas, diferentes cabinas, diferentes meses ni diferentes años. Las variables que se decidieron introducir en la secuencia son, precisamente, las variables que variaban semanalmente. Estas son: la entrada de reservas acumulada, la capacidad, el *load factor,* los ingresos acumulados, la tarifa media acumulada y las variables socioeconómicas de tipo de cambio, desocupación, tasa de empleo, inflación acumulada y salario mínimo.

Hay más datos relevantes a considerar, pero que no varían en el tiempo. Estos son: la ruta, el año, el mes y la cabina. Estos datos están contenidos en un vector, fuera de la secuencia, y que se van a concatenar con la salida de la red recurrente para ingresar a una red *feed-forward* con 1 neurona como *output*.

En resumen, van a ingresar a la red recurrente solamente los datos secuenciales, y va a salir un vector de tantas dimensiones como cantidad de neuronas se haya elegido. Luego, este vector se va a concatenar con el vector que contenga a los datos estáticos. Hay que recordar que este último contiene la información de la ruta, en forma de *embedding*, el año, el mes y la cabina. La siguiente imagen ejemplifica los valores de secuencia de entrada, el valor objetivo y el vector estático.

PAX	CAP	LF	INC	TM	...	SALARIO
7.000	15.000	0,46	546.000	78	...	120
7.800	15.000	0,52	616.200	79	...	120
8.900	16.000	0,55	720.900	81	...	120
10.000	16.000	0,62	850.000	85	...	110

SECUENCIA DE ENTRADA

VALOR OBJETIVO

METRO_RT	ANIO	MES	CAB
RUTA A	2022	4	Y

VECTOR ESTÁTICO

Tabla No. (09) Ejemplo de secuencia de entrada y vector estático.

Finalmente, a través de una capa lineal, se vinculará este vector con una única salida, que será la predicción, es decir la entrada de reservas acumulada de la próxima semana. A continuación, se muestra el esquema de la arquitectura planteada:

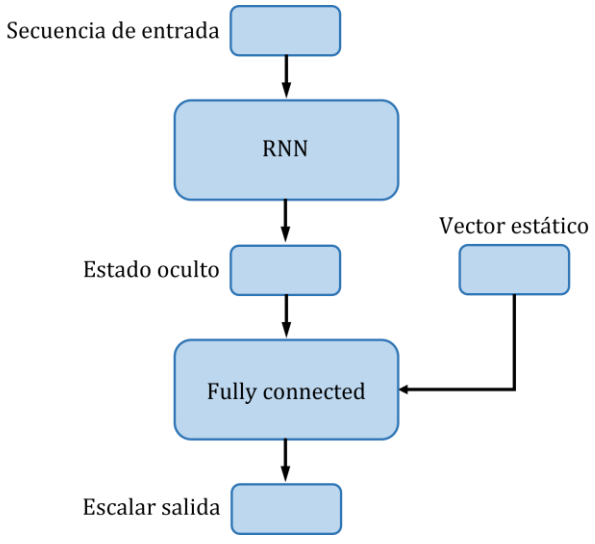


Figura No. (27) Esquema de red neuronal recurrente ordinaria.

El *dataset* base, antes de transformarlo en un tensor de secuencias, se modificó ligeramente respecto de los utilizados para las redes neuronales *feed-forward*. En los modelos anteriores, como

la entrada era un vector estático, no importaba la relación de una entrada con la correspondiente a la semana anterior. Por lo tanto, se eliminaron los registros con valores de entrada semanales negativos, que se consideraron como *outliers*. Si se utilizaba ese mismo *dataset* en esta arquitectura, se corría el riesgo de tener saltos muy grandes en las entradas de reservas, lo que iba a dificultar el aprendizaje. La imagen siguiente ejemplifica esta situación.

SEMANA	PAX	PAX_SEM
5	2000	1000
4	-500	-500
3	2100	2600

SEMANA	PAX	PAX_SEM
5	2000	1000
3	2100	2600

SEMANA	PAX	PAX_SEM
5	2000	1000
4	2000	0
3	2100	100

Tabla No. (10) Tratamiento de entrada de reservas negativas.

En la semana 4 se tiene una entrada de reservas negativa. Si se elimina, la secuencia experimenta un salto en la variable SEMANAS. Lo que se hizo fue reemplazar todas las entradas negativas por cero, y ajustar el valor siguiente de PAX_SEM como la diferencia entre PAX de la semana actual y PAX de la semana anterior. Los valores negativos representaban el 0.8%, por lo que no se alteró demasiado el *dataset*, y se espera que este cambio ayuda a la red a capturar mejor el comportamiento de las reservas. Además, para evitar eliminar más registros, no se utilizó el algoritmo de *IsolationForest* para depurar el conjunto de datos. Estas acciones hicieron que el *dataset* sea ligeramente más grande que en los modelos anteriores.

Se comenzó entrenando un modelo con una longitud de secuencia de 5, una capa oculta con 256 neuronas en el módulo recurrente y una capa en el módulo lineal. Luego de iterar 100 épocas, se consiguió un error en validación de 0.1335.

Este número no es comparable con los errores de los modelos anteriores porque en este caso, la variable de respuesta es PAX, no PAX_SEM y, además, está escalada, por lo que es esperable que sea mucho menor.

Para intentar mejorar las predicciones, se probaron a continuación dos modelos más complejos, con más neuronas y más capas ocultas tanto en los módulos recurrente como en el lineal (se puede

consultar las arquitecturas en el apéndice). Esta arquitectura más profunda trajo consigo un aumento importante de la cantidad de parámetros entrenables, pasando de 69.374 en el primer modelo a dos modelos de más de 20 millones de parámetros (un aumento de casi 300 veces más). Este incremento en los parámetros impidió graficar todos los datos del conjunto de validación porque el espacio en la memoria de la tarjeta gráfica era insuficiente, por lo que se seleccionó aleatoriamente un subconjunto de 10 mil registros. Luego de entrenar ambas redes, se obtuvieron errores en validación de 4.3514 y de 0.2935 respectivamente. La mayor diferencia entre los dos últimos modelos fueron los valores de entrada que se escogieron. Para el primer modelo complejo se agregaron 4 variables más: además de las 9 consideradas al principio, se agregaron capacidad, *load factor*, tarifa media acumulada y tarifa media semanal. Para el segundo modelo complejo, se agregó una capa oculta adicional a la red recurrente (pasando de 4 a 5) y se mantuvieron las 9 variables originales. Este último tuvo mejores resultados (0.2935 vs 4.3514), lo que permite deducir que las variables de entrada consideradas al principio son mucho más explicativas. Aun así, ninguna de estas dos alternativas lo hizo mejor que el primer modelo recurrente entrenado.

Existe otro tipo de red neuronal recurrente llamada LSTM (*Long Short-Term Memory*). Estas redes utilizan celdas de memoria especiales que permiten retener información a largo plazo y olvidar información irrelevante. El problema que solucionan las redes LSTM es el del gradiente que desaparece (*vanishing gradient*), que se produce en redes neuronales recurrentes convencionales. Éste aparece cuando la red retropropaga el error a través de múltiples capas en una secuencia temporal larga, y los gradientes se vuelven cada vez más pequeños a medida que se retropropagan hacia capas anteriores. Como resultado, las capas reciben gradientes muy pequeños, lo que dificulta el aprendizaje de patrones a largo plazo en los datos.

Las redes LSTM son especialmente efectivas para secuencias largas. Es por eso que en este caso van a armarse tensores con una longitud de secuencia de 16. Para que la red pueda capturar este comportamiento a largo plazo, se va a utilizar una arquitectura compleja, de más de 70 millones de parámetros entrenables. El modelo, luego de 100 épocas, obtuvo un error en validación de 6.7214 y hasta el momento fue el que peor desempeño mostró.

Los errores más bajos se consiguieron con arquitecturas más parsimoniosas. Por lo tanto, lo que resta por intentar es una red poco compleja, similar a la recurrente ordinaria, pero LSTM. Se consideró una longitud de secuencia de 4 semanas (1 mes), 1 capa oculta en el módulo recurrente, con 256 neuronas, y una sola capa lineal que conecte la salida de la recurrente con un único *output*.

Como resultado, se obtuvo un error en validación de 0.1202, lo que convierte a este modelo en el mejor de toda la sección de redes recurrentes.

5.3.3 Red neuronal *encoder-decoder*

Las redes neuronales recurrentes son efectivas para datos autocorrelacionados debido a su capacidad inherente para capturar y modelar dependencias secuenciales. Esta cualidad se puede utilizar como base para diseñar otra estructura denominada *encoder-decoder* que pueda abstraer información más compleja y generar una secuencia de salida variable.

Estas arquitecturas cuentan con un codificador (*encoder*) que se encarga de procesar la secuencia de entrada y comprimir la información más relevante en un tensor (estado oculto). Luego, el decodificador (*decoder*), a partir de este tensor, intentará replicar lo mejor posible la secuencia de entrada. Una vez entrenado el modelo, este será capaz de generar secuencias de datos variables que obedezcan al comportamiento particular de cada secuencia de entrada. Esta habilidad hace especialmente útiles a estas arquitecturas para la predicción de series de tiempo.

Como los mejores resultados en las redes neuronales recurrentes se dieron en las secuencias más cortas, se comenzó probando con un codificador recurrente que tomaba una secuencia de 2 entradas de tiempo. Este, a la salida, produce un estado oculto que es procesado por un decodificador recurrente de las mismas características que el anterior. La salida, que es concatenada con el vector de características estáticas y con el *embedding*, es pasada por una capa *fully-connected* que tiene la misma cantidad de neuronas de salida que la secuencia de entrada (12 variables). A continuación, se muestra un esquema de la arquitectura.

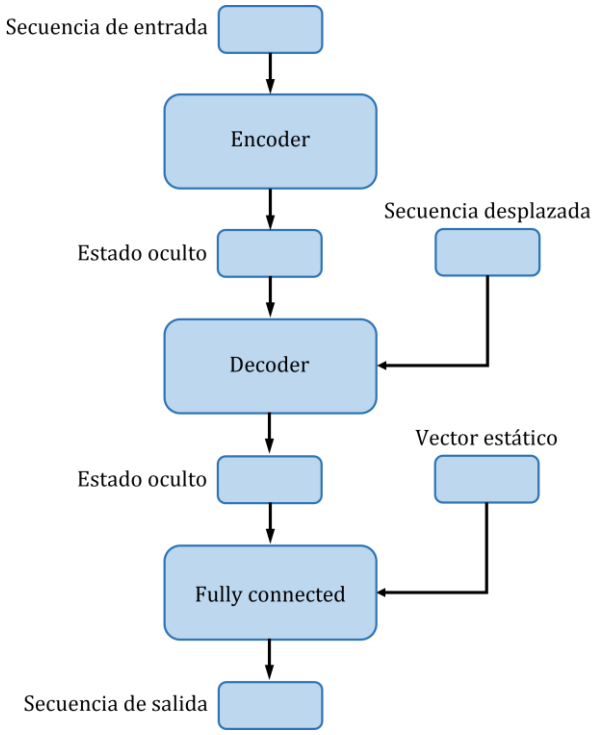


Figura No. (28) Esquema de red neuronal recurrente *encoder-decoder*.

Para calcular la pérdida, se tomó la primera columna de la matriz de salida, correspondiente a la entrada de reservas. El modelo, que se entrenó durante 100 épocas, obtuvo un error en validación de 0.1341.

Posteriormente, se intentó con otra arquitectura más compleja, que tome una secuencia de entrada de 4, y se aumentaron las capas ocultas del codificador y decodificador, que pasaron de 2 a 4 (porque se la hizo bidireccional); y las neuronas en cada capa, que aumentaron de 256 a 1024. Además, se agregó un vector al que se lo llamó "vector dinámico" con la información de la semana, capacidad y tarifa media. Este vector solo se agregó porque facilitaba la mecánica de las predicciones, manipulando los valores que se conocen de antemano. Adicionalmente, para intentar mejorar el desempeño del modelo, se agregó un mecanismo de atención que de lo que se encarga es ponderar algunas entradas de reservas previas para arrojar predicciones más precisas. A continuación, se muestra un esquema de la arquitectura.

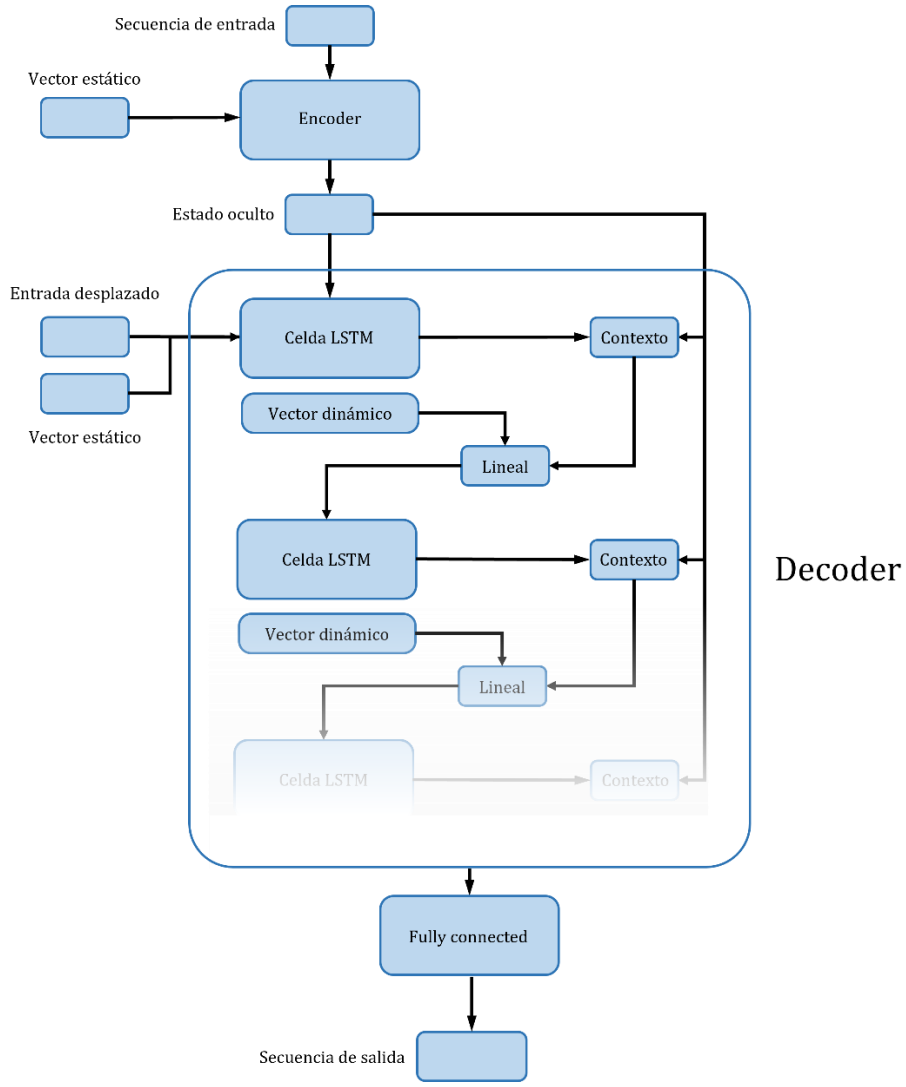


Figura No. (29) Esquema de red neuronal recurrente *encoder-decoder* con mecanismo de atención.

Ingresan una secuencia de entrada y el vector estático al codificador y este produce un estado oculto y una salida. Luego, una celda LSTM (en el decodificador) recibe el estado oculto, una entrada desplazada y el vector estático para producir un *output*. La salida del codificador contiene las salidas de toda la secuencia de entrada luego del procesamiento; estas se multiplican por la salida de la primera celda LSTM para generar un vector de contexto (primero se multiplican, luego se aplica una función *softmax* para transformar el resultado en pesos (*weights*), y luego se multiplican estos pesos por la salida de la celda LSTM). Posteriormente, el vector de contexto se concatena con el vector dinámica y es pasado por una capa lineal que traslada el vector resultante a un espacio de características apto para que pueda ingresar a la próxima celda LSTM. Finalmente, cada salida de cada celda pasa por una capa lineal para constituir las predicciones finales. Luego de un entrenamiento de 100 épocas, el modelo consiguió un error en validación de 0.0813.

5.3.4 Conclusiones acerca de modelos de redes neuronales

El modelo de red neuronal de tipo *feed-forward* demostró ser superior a los modelos estadísticos lineales y los modelos de árboles. Este resultado indica que la capacidad de las redes neuronales para capturar relaciones no lineales y aprender representaciones más complejas fue fundamental para su mejor rendimiento.

Además, al comparar un modelo de *encoder-decoder* con atención frente a otros modelos calculados con recurrentes, se observó que el primero obtuvo resultados superiores. Esto sugiere que la inclusión de mecanismos de atención en el proceso de codificación y decodificación permitió una mejor captura de la información relevante y una generación de resultados más precisos.

5.4 Selección del mejor modelo

A continuación, se muestra una tabla que resume los resultados de todos los modelos entrenados.

N	TIPO	MODELO	MSE	R2	R2 adj
1	Lineal	Múltiple	175939,09	0,4797	0,4779
2	Lineal	Múltiple + vars. socioeconómicas	171402,82	0,4810	0,4792
3	Lineal	Múltiple + vars. socioeconómicas + lasso	117176,44	0,6754	0,6568
4	Lineal	Múltiple + vars. socioeconómicas + ridge	171745,52	0,4800	0,4781
5	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + ridge	17694,14	0,8406	0,8172
6	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + lasso	21333,55	0,8078	0,7796
7	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + enets	19120,25	0,8153	0,7894
8	Lineal	Múltiple + vars. socioeconómicas + trat. outliers + ponderada + ridge	16861,97	0,8466	0,8251
9	Lineal	PCR + vars. socioeconómicas + trat. outliers + ponderada + ridge	28166,49	0,7279	0,7257
10	Lineal	PLS + vars. socioeconómicas + trat. outliers + ponderada + ridge	13892,98	0,8588	0,8577
11	Árboles	Random forest	40320,26	0,7422	0,7403
12	Árboles	Random forest + *bayesian search*	15786,47	0,8464	0,8250
13	Árboles	XGBoost	19465,61	0,8756	0,8746
14	Árboles	XGBoost + *bayesian search*	11739,99	0,8858	0,8698
15	Redes neuronales	*Feed forward*	11755,62	0,8878	0,8734
16	Redes neuronales	*Feed forward* profunda 1	12617,29	0,8796	0,8641
17	Redes neuronales	*Feed forward* profunda 2	13070,35	0,8753	0,8593
18	Redes neuronales	RNN seq_len = 5	0,1335	-	-
19	Redes neuronales	RNN de complejidad alta	4,3514	-	-
20	Redes neuronales	RNN de comlejidad moderada	0,2935	-	-
21	Redes neuronales	LSTM seq_len = 16	6,7214	-	-
22	Redes neuronales	LSTM seq_len = 4	0,1202	-	-
23	Redes neuronales	*Encoder-Decoder* seq_len = 2	0,1341	-	-
24	Redes neuronales	*Encoder-Decoder* seq_len = 4 + atención	0,0813	-	-

Tabla No. (11) Resumen de resultados de cada modelo entrenado (Lineal + Árboles + Redes neuronales).

Se entrenaron múltiples modelos hasta el momento que no pueden seleccionarse en función al error en validación porque las variables de respuesta fueron diferentes.

Cuando lo que se predijo era la variable PAX (escalada), el que menor error tuvo fue el modelo de redes neuronales *encoder-decoder* con atención. Sin embargo, cuando lo que se predijo era la

variable PAX_SEM (no escalada) hubo 3 modelos que arrojaron errores similares: red neuronal *feed-forward*, XGBoost y PLS. De estos últimos 3 modelos se van a pre-seleccionar los que presenten diferencias significativas. Para ello, se hicieron 1000 muestreos *Bootstrap* con reemplazo sobre el conjunto de validación y se midieron los errores cuadráticos medios. A continuación se muestran las distribuciones de los errores para cada modelo.

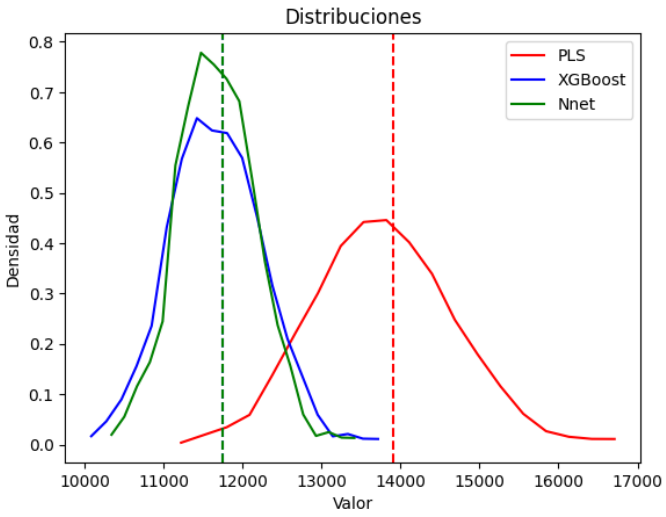


Figura No. (30) Distribuciones de los MSE para cada modelo.

Se puede notar que las distribuciones de los modelos de *XGBoost* y red neuronal son muy parecidas y ninguna es significativamente superior a la otra. Por otro lado, el modelo de PLS, que tiene una media mayor, también muestra una desviación mayor. Para determinar si la diferencia es significativa al 5%, se van a plantear las siguientes hipótesis:

$$H_0: \mu_1 - \mu_2 = 0 \quad vs \quad H_1: \mu_1 - \mu_2 \neq 0 \tag{35}$$

El test de hipótesis con nivel de significación α será:

$$\varphi(X) = \begin{cases} 1 & si \quad |Z| > k_\alpha \\ 0 & en\ otro\ caso \end{cases} \tag{36}$$

Con

$$Z = \frac{\overline{x_1} - \overline{x_2}}{\sqrt{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}}} \quad (37)$$

que, bajo H_0, tiene distribución t de *student* con n grados de libertad. Para el caso en que se consideren las distribuciones de los errores de PLS y de NNET, el estadístico de prueba da 64.96, que es superior al valor crítico a una cola de 1.64, por lo tanto, se puede concluir que las diferencias son estadísticamente significativas, y se procede a descartar el modelo de PLS. Como los modelos de NNET y XGBOOST arrojar errores muy similares, se van a considerar ambos para la elección del modelo definitivo. Finalmente, los candidatos son: NNET, XGBOOST y ENCODER-DECODER.

Para poder determinar el mejor modelo se van a seleccionar 10 rutas que representan el flujo de tráfico más significativo dentro de la red doméstica. Sobre estas rutas se van a realizar las predicciones para los períodos de febrero, marzo y abril de 2022, con 8 semanas en avance. Como se cuenta con la información a la semana 0 (la cantidad de pasajeros que efectivamente volaron la ruta en cada período), se va a poder evaluar el error de cada uno de ellos, y se seleccionará el que menor error demuestre. La tabla siguiente muestra las predicciones de cada modelo para cada una de las 10 rutas en los tres meses considerados.

feb	1	2	3	4	5	6	7	8	9	10
REAL	27.670	17.825	48.355	47.377	14.300	36.437	35.263	30.278	45.630	17.129
NNET	21.070	20.337	53.147	50.112	15.410	40.424	37.215	29.047	43.491	16.573
XGBOOST	21.002	18.319	50.482	51.114	16.490	41.637	36.843	28.024	39.142	15.471
ENC-DEC	22.496	17.126	54.172	46.539	13.025	32.578	31.743	26.940	48.656	14.125
mar	1	2	3	4	5	6	7	8	9	10
REAL	33.776	22.608	59.181	52.746	14.797	42.088	47.060	41.086	47.133	20.239
NNET	37.969	27.493	68.294	56.218	17.074	46.219	50.480	45.667	51.003	23.362
XGBOOST	41.000	28.868	64.879	61.840	18.781	47.606	49.975	48.864	54.063	21.960
ENC-DEC	29.640	18.233	63.450	58.253	12.867	39.417	43.250	38.517	51.437	16.741
abr	1	2	3	4	5	6	7	8	9	10
REAL	37.763	27.143	42.627	36.946	13.811	53.456	45.283	51.612	34.364	23.756
NNET	40.965	29.199	48.471	41.362	19.891	57.014	47.557	54.155	36.478	28.378
XGBOOST	38.590	30.075	46.091	41.787	21.887	59.914	47.814	57.937	39.020	28.094
ENC-DEC	35.973	22.157	43.028	36.406	12.227	49.331	43.747	54.871	35.563	17.940

Tabla No. (12) Predicciones para escoger el mejor modelo.

Para poder seleccionar el modelo más preciso, se sumaron los cuadrados de los residuos, y se tomó la raíz cuadrada. A continuación, se muestran ambas métricas.

MODELO	MSE	RMSE
NNET	15.969.628	3.996
XGBOOST	25.294.708	5.029
ENC-DEC	12.128.287	3.483

Tabla No. (13) Errores para escoger el mejor modelo.

Se puede ver que, en promedio, el modelo de *encoder-decoder* es el que menos se equivoca por lo que se selecciona este modelo para continuar el trabajo.

5.5 Interpretación de *embeddings*

Cada una de las rutas aéreas fue codificada un vector de 8 dimensiones. El valor de cada elemento del vector fue determinado por la red neuronal durante el entrenamiento. Si bien la explicación de cada número y de cada dimensión no es interpretable, se puede hacer un esfuerzo en entender la elección de todos los números en conjunto. Esto se puede lograr si se grafican los *embeddings* en el plano. Como los vectores tienen 8 dimensiones, es inevitable la pérdida de información.

Un método para lograr la reducción en la dimensionalidad es *Principal Component Analysis* (PCA), que lo que busca capturar la mayor varianza en los datos. Sin embargo, existe otro método que es no lineal, y que se enfoca en la preservación de las relaciones de vecindad entre los puntos, llamado *t-Distributed Stochastic Neighbor Embedding* (t-SNE).

El algoritmo de t-SNE preserva simultáneamente las relaciones y la estructura de proximidad entre las muestras originales, y mapea las muestras de datos de alta dimensionalidad en un espacio menor, de manera tal que las muestras similares se representen cerca una de la otra en el nuevo espacio. Utiliza una distribución de probabilidad *t-student* para medir la similitud entre las muestras en el espacio original y el espacio reducido. A continuación, se muestran los *embeddings* representados en el plano.

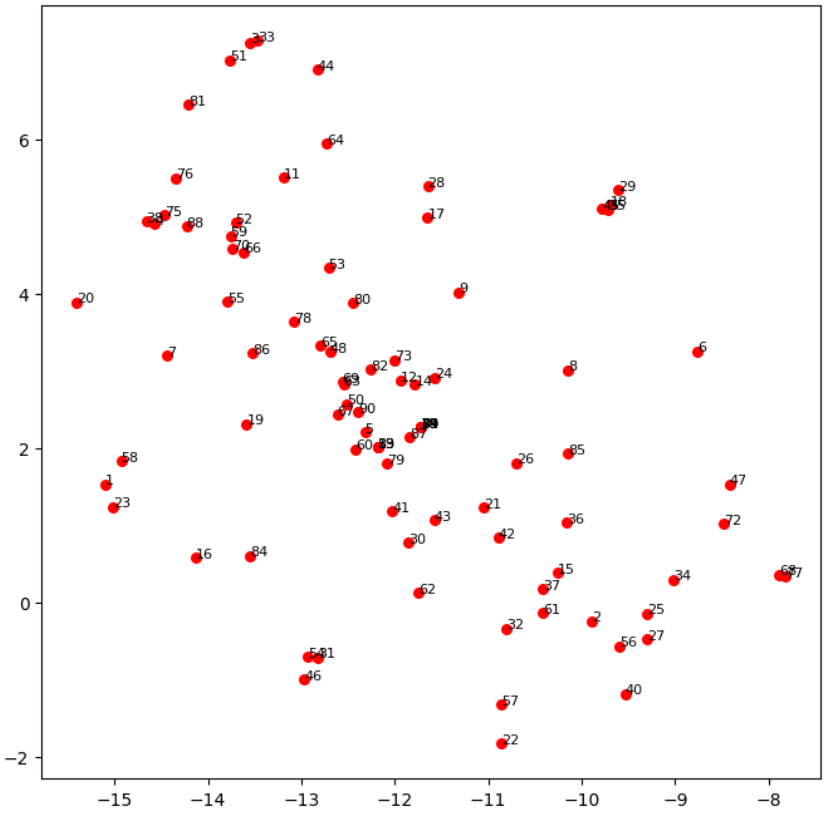


Figura No. (31) Representación de los *embeddings* en el plano con el método de t-SNE.

La rutas que están más próximas entre sí son más "parecidas", es decir que tienen comportamientos similares. Son similares en las variables con las que se entrenaron. Lo que esto sugiere es que, si una ruta tiene un buen desempeño durante determinada temporada, es esperable que la ruta parecida también lo tenga. Con los datos presentados de esta manera, se podría aplicar un algoritmo de agrupamiento para separar rutas.

El algoritmo de t-SNE es efectivo para visualizar agrupamientos, pero al ser un método estocástico, los resultados pueden cambiar en cada intento. Además, sería riesgoso aplicar un *K-means* para agrupar, dado que no se conservan las distancias. Por eso, para la generación de *clusters*, se va a aplicar el algoritmo luego de representar a los *embeddings* sobre las dos componentes principales con PCA.

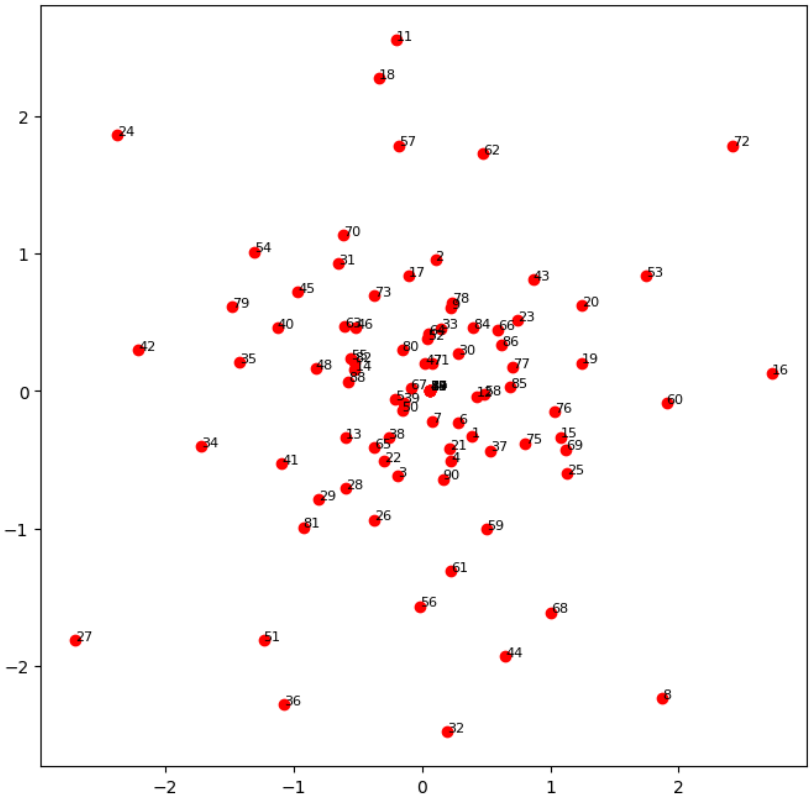


Figura No. (32) Representación de los *embeddings* en el plano con el método de PCA.

K-means es un algoritmo de agrupamiento no supervisado utilizado para dividir un conjunto de datos en k grupos (*clusters*) basándose en un distancia entre ellos. El algoritmo solicita como hiperparámetro la cantidad de grupos que se desea generar. Para poder determinar la cantidad de grupos más efectiva, se entrenaron 9 modelos de *K-means* diferentes, cada uno con una cantidad de grupos distinta, y se graficaron la suma de las distancias al cuadrado entre cada punto y su centroide.

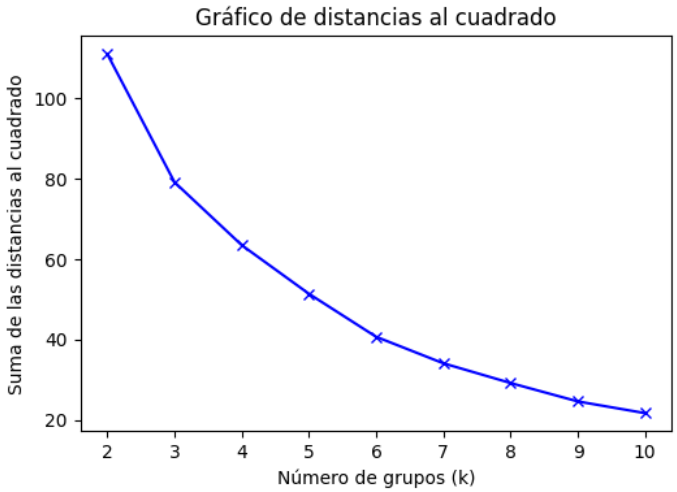


Figura No. (33) Sumatoria del cuadrado de las distancias en función del número de *clusters* (representación PCA).

Cuanto menor sea la distancia, mejor será el agrupamiento. Sin embargo, es lógico que si se escogen más grupos, la distancia será menor, por lo que la elección termina siendo un *tradeoff* entre la distancia y la cantidad de grupos. En este gráfico la disminución de la distancia con la cantidad de grupos es bastante uniforme, por lo que no está siendo muy útil para elegir el número óptimo. Se esperaría ver un "codo" en la curva que signifique una notable disminución en la distorsión (suma de distancias al cuadrado).

Existe otro gráfico en donde lo que se intenta buscar es el "codo", solo que en el eje de las ordenadas, en lugar de la suma de distancias al cuadrado, se mide un cociente entre varianzas: varianza dentro de los *clusters* sobre varianza entre los *clusters*. Como se busca el mejor agrupamiento, la varianza dentro de los *clusters* (numerador) debería ser baja, mientras que la varianza entre los *clusters* (denominador) debería ser alta.

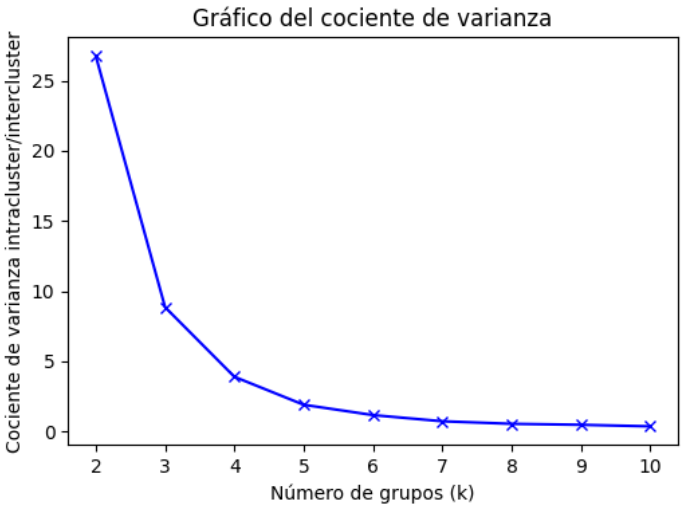


Figura No. (34) Cociente entre la varianza *intra-cluster* y varianza *inter-clusters* s en función del número de *clusters* (representación PCA).

El "codo" del gráfico parecería estar en el 3. Se puede recurrir a otro método, *Sillhouette*, para poder tomar una decisión final con todos los gráficos en conjunto. El coeficiente de silueta, que se utiliza para evaluar la calidad de los resultados de agrupamiento generados por el algoritmo, proporciona una medida de cuán bien están separados los diferentes grupos y qué tan similar son los puntos dentro de cada grupo. El coeficiente se calcula de la siguiente manera:

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}} \tag{38}$$

Donde:

- a(x) es la distancia promedio entre un punto de datos x y todos los demás puntos en el mismo grupo (distancia *intra-cluster*). Cuanto más pequeña sea esta distancia, mejor, puesto que indica que el punto x está más cerca de los puntos dentro de su propio grupo.
- b(x) es la distancia promedio entre un punto de datos x y todos los puntos en el grupo más cercano diferente al que pertenece (distancia *entre-clusters*). Cuanto mayor sea esta distancia, mejor, puesto que indica una mayor separación entre el punto x y los puntos en otros grupos.

El valor del coeficiente de silueta varía entre -1 y 1. Un valor cercano a 1 indica que el punto está bien clasificado y está separando bien lo datos; mientras que un valor cercano a -1 indica todo lo contrario.

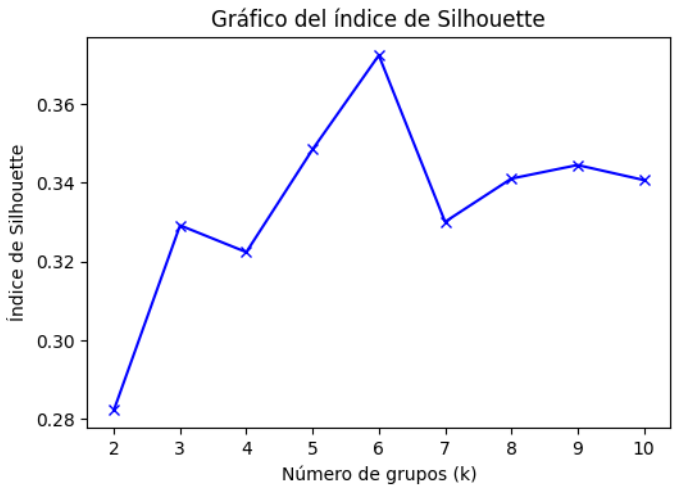


Figura No. (35) Índice de *Silhouette* en función del número de *clusters* (representación PCA).

Según el coeficiente de silueta, el mejor número es 6. Como los demás gráficos no proporcionaron información que contrarreste esta decisión, se optó por escoger este número como el definitivo. A continuación, se muestra cómo quedaron agrupadas las rutas en los 6 *clusters*:

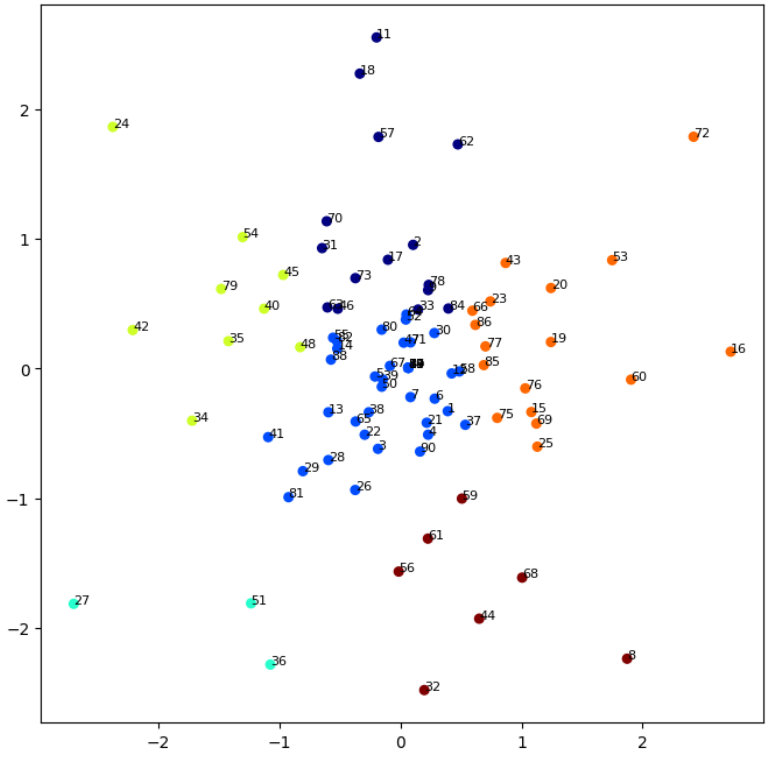


Figura No. (36) Agrupación final (representación PCA).

Para poder notar la diferencia de agrupamientos entre los métodos PCA y t-SNE, en el apéndice se pueden encontrar los mismos gráficos para determinar los *clusters* para los *embeddings* graficados con t-SNE.

5.6 Asignación de capacidad

Se van a considerar 10 rutas aéreas en los períodos de febrero, marzo y abril de 2023 para analizar una posible reasignación de capacidad. Las rutas cuentan con la siguiente oferta de capacidad.

CAP	1	2	3	4	5	6	7	8	9	10
feb	34.344	29.484	82.620	50.220	17.172	58.320	46.656	46.980	57.672	31.428
mar	42.120	40.824	101.412	54.756	19.116	76.788	66.420	50.868	61.236	41.796
abr	46.332	44.712	66.420	40.500	20.088	89.748	67.392	57.348	53.460	51.840

Capacidad total	1.548.072

Tabla No. (14) Capacidad de cada ruta para cada mes.

La capacidad total son 1.548.072 asientos (9556 vuelos de 162 de capacidad cada uno). Con el modelo de proyección de demanda entrenado se van a estimar las entradas de reservas para esas rutas en esos meses, utilizando como tarifa media el último valor registrado. Las tablas siguientes muestran las estimaciones de reservas (tabla 1) y los ingresos esperados (tabla2).

RES sin OPT	1	2	3	4	5	6	7	8	9	10
feb	29.876	23.339	70.639	49.596	13.140	41.855	36.529	41.413	58.242	21.932
mar	32.712	27.283	72.094	58.951	14.139	42.873	49.631	46.219	55.601	25.088
abr	34.972	29.490	63.473	52.047	10.449	52.402	51.666	63.500	48.315	21.806

Reservas totales	1.239.272

INC sin OPT	1	2	3	4	5	6	7	8	9	10
feb	3.238.958	2.154.126	7.930.270	7.873.066	1.259.889	4.396.808	4.084.283	4.838.462	10.250.996	1.652.935
mar	2.751.100	1.849.659	5.317.104	7.357.818	1.139.173	2.742.261	4.402.006	5.871.350	8.828.421	1.929.777
abr	3.320.536	1.780.584	5.487.060	6.409.872	698.169	3.624.175	5.512.748	6.258.721	5.964.365	1.657.457

Ingresos totales	130.582.151

Tabla No. (15) Estimaciones de reservas (superior) e ingresos esperados (inferior).

Los valores resaltados son lo que excedieron la capacidad actual y es donde debería aumentarse para no perder pasajeros. Estos primeros valores estimados utilizaron un valor de tarifa media *semi-arbitrario* (porque no corresponden a los valores óptimos de tarifa).

La asignación de capacidad surgirá como solución a un problema de optimización en donde se maximicen los ingresos (tarifa media * entrada de reservas) sujeto la restricción de capacidad total. Para el planteamiento del problema, se pensó trabajar con un tensor de 3 dimensiones, en donde cada elemento del tensor es una predicción de reservas. La primera dimensión i corresponde a cada una de las 10 rutas, la segunda dimensión j es corresponde a la tarifa con la que se vendió cada reserva, y la tercera dimensión k corresponde a los meses en estudio. Para completar el tensor, se corrió un algoritmo que iteraba sobre todas las dimensiones. Como el modelo entrenado tiene una variable de tarifa media, los resultados se verán afectados por el valor de tarifa que se escoja. Es por eso que se reservó una dimensión del tensor (dimensión 2) para esta variable, y se consideraron 30 valores diferentes, desde 40 hasta 190 dólares.

A continuación, se muestran las 3 matrices correspondientes a los 3 meses (febrero, marzo y abril) que conforman el tensor para resolver el problema de optimización. En el primer eje vertical se enumeran las 10 rutas y en el eje horizontal se itera sobre 30 valores de tarifa distintos. Cada elemento de las matrices es la demanda estimada.

MES 1	40	50	60	70	80	90	100	...	190
RUTA 1	dem ruta1 con tm=40	...	...	...	...	...	...		...
RUTA 2	...	...	...	...	...	...	...		...
RUTA 3	...	...	...	...	...	...	...		...
RUTA 4	...	...	...	...	...	...	...		...
RUTA 5	...	...	...	...	...	...	...		...
RUTA 6	...	...	...	...	...	...	...		...
RUTA 7	...	...	...	dem ruta7 con tm=70	...	...	...		...
RUTA 8	...	...	...	...	...	...	...		...
RUTA 9	...	...	...	...	...	...	...		...
RUTA 10	...	...	...	...	...	...	...		...

MES 2	40	50	60	70	80	90	100	...	190
RUTA 1	dem ruta1 con tm=40	...	...	...	...	...	...		...
...	...	...	...	...	...	...	...		...
RUTA 10	...	...	...	...	...	...	...		...

MES 3	40	50	60	70	80	90	100	...	190
RUTA 1	dem ruta1 con tm=40	...	...	...	...	...	...		...
...	...	...	...	...	...	...	...		...
RUTA 10	...	...	...	...	...	...	...		...

Tabla No. (16) Matrices que conforman el tensor para la optimización.

Lo recién explicado puede expresarse matemáticamente de la siguiente manera.

Maximizar:

$$\sum_{i=1}^{10}\sum_{j=1}^{30}\sum_{k=1}^{3} x[i,j,k] * tensor\_reservas[i,j,k] * vector\_tm\,[j] \quad (39)$$

Sujeto a:

$$\sum_{i=1}^{10}\sum_{j=1}^{30}\sum_{k=1}^{3} x[i,j,k] * tensor\_reservas[i,j,k] \leq 1.548.072 \quad (40)$$

$$\sum_{i=j}^{30} x[i,j,k] = 1\ para\ todo\ \ i \in I = [1,2,\dots,10],\ \ k \in K = [1,2,3] \quad (41)$$

Con:

$$x[i,j,k]\ binario\ f.a.\ \ i \in I = [1,2,\dots,10], j \in J = [1,2,\dots,30], k \in K = [1,2,3] \qquad (42)$$

Con la solución arrojada por el modelo de programación lineal, se armaron las siguientes tablas, en donde se muestran las tarifas que deben aplicarse, las reservas y los ingresos esperados para cada ruta y mes.

TM con OPT	1	2	3	4	5	6	7	8	9	10
feb	187,55	187,55	187,55	187,55	187,55	187,55	187,55	187,55	187,55	187,55
mar	187,55	187,55	187,55	187,55	187,55	187,55	187,55	145,91	187,55	187,55
abr	88,66	187,55	187,55	177,14	119,88	187,55	187,55	130,29	187,55	187,55

RES con OPT	1	2	3	4	5	6	7	8	9	10
feb	25.592	22.664	66.593	46.789	10.728	38.392	31.609	34.907	56.658	21.220
mar	21.093	36.899	78.122	50.566	11.879	54.844	40.138	38.424	51.290	21.380
abr	30.092	46.696	67.242	40.381	5.992	63.119	41.125	52.111	51.047	13.700

Reservas totales	1.171.292

INC con OPT	1	2	3	4	5	6	7	8	9	10
feb	4.799.702	4.250.564	12.489.314	8.775.134	2.012.004	7.200.303	5.928.172	6.546.702	10.626.035	3.979.746
mar	3.955.928	6.920.295	14.651.543	9.483.499	2.227.870	10.285.825	7.527.760	5.606.392	9.619.283	4.009.754
abr	2.667.833	8.757.693	12.611.032	7.152.984	718.349	11.837.776	7.712.869	6.789.762	9.573.709	2.569.393

Ingresos totales	211.287.227

Tabla No. (17) Tarifa media óptima (superior), estimación de reservas (centro) e ingresos esperados (inferior)

Si se comparan estos resultados obtenidos a partir de la optimización lineal con los anteriores, se puede notar que los ingresos totales aumentaron en un 62%. En cuanto a la capacidad, ésta fue excedida en dos rutas para el mes de abril. Como criterio para reasignar la capacidad, se dividieron las reservas por 0.77. Este quiere decir que se espera que el *load factor* sea 77% a fin de mes, porque:

$$LF = \frac{RPK}{ASK} = \frac{PAX * DIST}{CAP * DIST} = \frac{PAX}{CAP} \rightarrow CAP = \frac{PAX}{LF(0.77)} \qquad (43)$$

Como se están analizando los LF para cada ruta de manera individual, el multiplicando DIST (distancia) es el mismo, por lo que puede cancelarse, y el LF puede pasar a expresarse como el cociente entre los pasajeros finales y la capacidad.

Al hacer la división por 0.77, se obtiene la capacidad. A este valor, se lo divide por 162 (que es la cantidad de asientos de un equipo) y luego se lo divide por 2 (por idas y vueltas). El resultado debería ser un numero entero, que corresponde a la cantidad de vuelos asignados. Como el número es racional, se lo redondea al entero mayor y luego se lo multiplica por 162 y por 2 para obtener la capacidad real. Por el momento, así es como quedó la oferta de asientos.

CAP	1	2	3	4	5	6	7	8	9	10
feb	33.372	29.484	86.508	60.912	14.256	49.896	41.148	45.360	73.872	27.864
mar	27.540	47.952	101.736	65.772	15.552	71.280	52.164	50.220	66.744	27.864
abr	39.204	60.912	87.480	52.488	8.100	82.296	53.460	67.716	66.420	17.820

Capacidad total	1.548.072
Capacidad asignada	1.525.392
Capacidad sobrante	22.680

Tabla No. (18) Oferta de asientos.

Para completar la oferta asignada al principio, todavía restan repartir 22.680 asientos (70 vuelos ida y vuelta). Se consideró sumarles 10 idas y vueltas adicionales a las 7 rutas con más ingresos. Las tablas a continuación muestran la asignación de capacidad original, la reasignación de capacidad en función a la entrada de reservas que maximizan los ingresos, y la diferencia.

CAP_original	1	2	3	4	5	6	7	8	9	10
feb	34.344	29.484	82.620	50.220	17.172	58.320	46.656	46.980	57.672	31.428
mar	42.120	40.824	101.412	54.756	19.116	76.788	66.420	50.868	61.236	41.796
abr	46.332	44.712	66.420	40.500	20.088	89.748	67.392	57.348	53.460	51.840

CAP_reasignada	1	2	3	4	5	6	7	8	9	10
feb	33.372	29.484	89.748	60.912	14.256	49.896	41.148	45.360	77.112	27.864
mar	27.540	47.952	104.976	65.772	15.552	74.520	52.164	50.220	69.984	27.864
abr	39.204	60.912	90.720	52.488	8.100	85.536	53.460	67.716	66.420	17.820

CAP_diferencia	1	2	3	4	5	6	7	8	9	10
feb	-972	0	7.128	10.692	-2.916	-8.424	-5.508	-1.620	19.440	-3.564
mar	-14.580	7.128	3.564	11.016	-3.564	-2.268	-14.256	-648	8.748	-13.932
abr	-7.128	16.200	24.300	11.988	-11.988	-4.212	-13.932	10.368	12.960	-34.020

Capacidad total	1.548.072

Tabla No. (19) Capacidad original (superior), capacidad reasignada luego de optimización lineal (centro) y diferencia de capacidad (inferior).

6 CONCLUSIÓN Y REFLEXIONES

6.1 Conclusión

El modelo de predicción constituye una herramienta para ayudar al analista a tomar decisiones. Contar con información anticipada sobre la demanda de tráfico aéreo ofrece diversas ventajas estratégicas. En primer lugar, permite realizar una planificación más precisa de la capacidad y los recursos necesarios. Al tener una idea clara de cuántos pasajeros se esperan en determinados períodos, se puede ajustar la disponibilidad de asientos y optimizar la asignación de aeronaves en función de la demanda proyectada. Esto ayuda a evitar tanto la falta de capacidad como el exceso de esta, maximizando así los ingresos y minimizando los costos operativos.

Además, al anticipar la demanda de tráfico, se pueden aplicar estrategias de *pricing* más efectivas. estableciendo tarifas competitivas y ajustarlas en consecuencia para maximizar la ocupación de los vuelos y aumentar los ingresos. De esta manera, se pueden aprovechar al máximo los períodos de alta demanda y, al mismo tiempo, implementar estrategias para estimular la demanda en períodos más tranquilos, ofreciendo tarifas promocionales o descuentos selectivos.

6.2 Reflexiones

Este trabajo logró mostrar una mejora del 62% en los ingresos esperados para las 10 rutas y los 3 meses en consideración. Sin embargo, hay que mirar a este número con cuidado porque el modelo entrenado parte de varios supuestos muy fuertes que no hay que subestimar.

En primer lugar, el modelo aprende relaciones en base a los datos que recibe. La industria aeronáutica es una industria muy competitiva que, en muchas oportunidades, las empresas involucradas compiten por precio. Dicho esto, en ningún momento el modelo consideró la presencia de empresas competidoras que podrían amenazar la entrada de reservas.

En segundo lugar, en Argentina suele haber promociones en la venta de pasajes que motivan mucho al consumidor. En estos períodos, las entradas de reservas son excepcionalmente mayores y el modelo no incluyó una variable que contemple este comportamiento.

En tercer lugar, el modelo, al generar una secuencia de entrada de reservas, para realizar una predicción futura, se basa en una predicción anterior, por lo que va acumulando errores que aumentan cuanto más en avance se quiera predecir. Es decir, el modelo va a tener más error si se utiliza para proyectar cómo van a ser las reservas a 20 semanas a que si se lo usara para proyectar la semana próxima. Por este motivo, cuando se realizaron las predicciones para la reasignación de capacidad, es esperable que para el mes de abril el error sea mayor que para el mes de febrero, puesto que se predijo con 14 semanas en avance, mientras que para febrero se predijo con 6.

En cuarto, lugar, cuando se realizaron las primeras predicciones sin la optimización lineal, en donde se tomó la última tarifa media, se supuso que la tarifa no cambiaría nunca. Esto nunca es así, puesto que la tarifa suele modificarse en función a la reacción del consumidor, y se la utiliza como "válvula" para regular las entradas.

Con todo lo mencionado anteriormente, se puede concluir que el modelo es una herramienta útil para tomar decisiones, pero hay que reconocer sus limitaciones. La industria es volátil en sus variables, y el modelo sólo consideró algunas de ellas. Es recomendable correr las predicciones semanalmente para que el modelo pueda corregir en cada semana sus predicciones y converger al valor real conforme se vaya acercando la fecha de finalización del mes.

6.3 Trabajo futuro

El hecho de haber entrenado un solo modelo para todas las rutas permitió aprovechar la información compartida entre ellas y entrenar *embeddings*. Esto resultó beneficioso, ya que los *embeddings* capturaron relaciones y similitudes entre las diferentes rutas y pudieron mejorar la capacidad de generalización del modelo. Además, al tener un solo modelo, también se redujo la complejidad y el costo computacional del entrenamiento y la inferencia.

Sin embargo, sería interesante intentar entrenar un modelo diferente para cada ruta. Al tener un modelo separado para cada ruta, se aprovecharía la especificidad y las características únicas de cada una. Esto podría permitir una adaptación más precisa a las particularidades de cada ruta y, potencialmente, mejorar las predicciones.

Al dividir el modelo en múltiples modelos individuales, también permitiría una flexibilidad para ajustar y optimizar cada modelo de manera independiente. De esta manera, se podrían ajustar los

hiperparámetros, la arquitectura y los datos de entrenamiento específicos para cada ruta, lo que puede conducir a un rendimiento mejorado y más ajustado a las necesidades de cada una en particular.

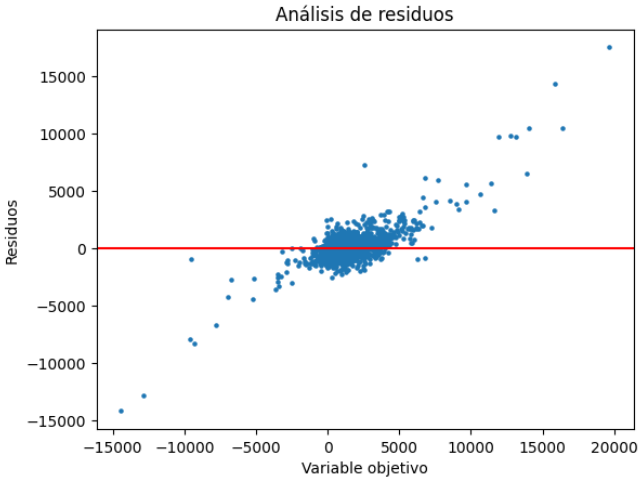
Para futuras investigaciones, se sugiere explorar la posibilidad de utilizar el modelo de seleccionado para predecir no solo un valor puntual, sino un rango de valores con un intervalo de confianza. Este enfoque puede lograrse mediante la realización de múltiples simulaciones que generen resultados diferentes en cada ejecución debido a la componente aleatoria de las capas *dropout*. Posteriormente, será posible seleccionar los cuantiles relevantes y construir un intervalo de confianza que brinde una estimación más completa y precisa de la incertidumbre asociada a las predicciones del modelo.

7 BIBLIOGRAFÍA

1. Adeniran A. O., Kanyio O. A. & Owoeye A. S. (2018). Forecasting Methods for Domestic Air Passenger Demand in Nigeria. Journal of Applied Research on Industrial Engineering.
2. Ahmad T. Al-Sultan, Amani Al-Rubkhi, Ahmad Alsaber & Jiazhu Pan. (2021). Forecasting air Passenger traffic volume: evaluating time series models in long-term forecasting of Kuwait air passenger data. Kuwait University & University of Strathclyde.
3. Cardenas, R. S. (2022). Apuntes de cátedra Procesamiento de Lenguaje Natural de la carrera Especialización en Inteligencia Artificial. Universidad de Buenos Aires.
4. Charu, C. A. (2018). Neural Networks and Deep Learning. Springer.
5. Devore, J. L. (2018). Fundamentos de probabilidad y estadística. Cenegage.
6. Doğan, İ; Kılıç, I; Saracli, S & Gazeloğlu, C. (2015). Principal Components Regression and an Application.
7. Efron et al. (2004). Least Angle Regression. Annals of Statistics, Vol 32. No 2, pp. 407-499. Institute of Mathematical Statistics.
8. García M. J. (2022). Apuntes de cátedra Aprendijaze Estadístico de la carrera Maestría en Ingeniería Matemática. Universidad de Buenos Aires.
9. Gaset, C. (2022). Redes neuronales recurrentes aplicadas a series de tiempo: predicción de pasajeros de rutas regionales en Argentina. Universidad Torcuato Di Tella. Tesis de maestría.
10. Goodfellow, I; Bengio, Y. & Cournville, A. (2016). Deep Learning. The MIT Press.
11. Hastie, T; Tibshirani, R. & Friedman, J. (2008). The elements of statistical learning. Springer.
12. Maillot M. U. (2022). Apuntes de cátedra Aprendijzaje Profundo de la carrera Especialización en Inteligencia Artificial. Universidad de Buenos Aires.
13. Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830.
14. Phillips, R. L. (2021). Princing and Revenue Optimization. Standford Business Books.
15. Seber, George. A. F. & Lee, Alan J. (2003). Linear Regression Analysis. Wiley.
16. Strang, G. (2019). Linear Algebra and Learning from Data. Massachusetts Institute of Tecnology.
17. Weatherford L. R., Gentry W. T. & Wilamowski B. (2002). Neural network forecasting for airlines: A comparative analysis. Journal of Revenue and Princing Management, Vol 1 No. 4.

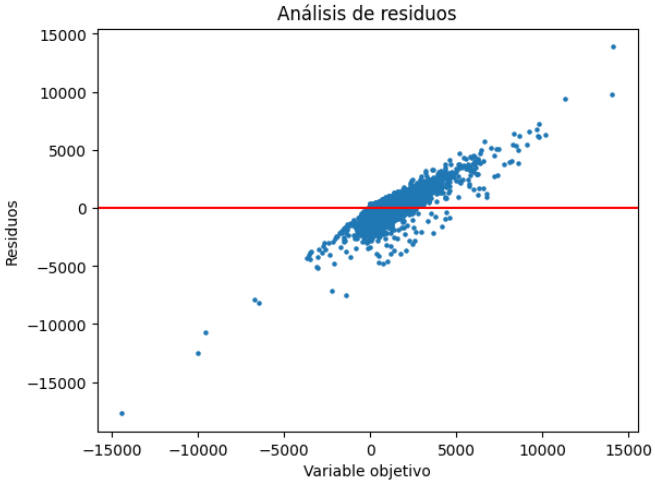
8 APÉNDICE

8.1 Modelos lineales



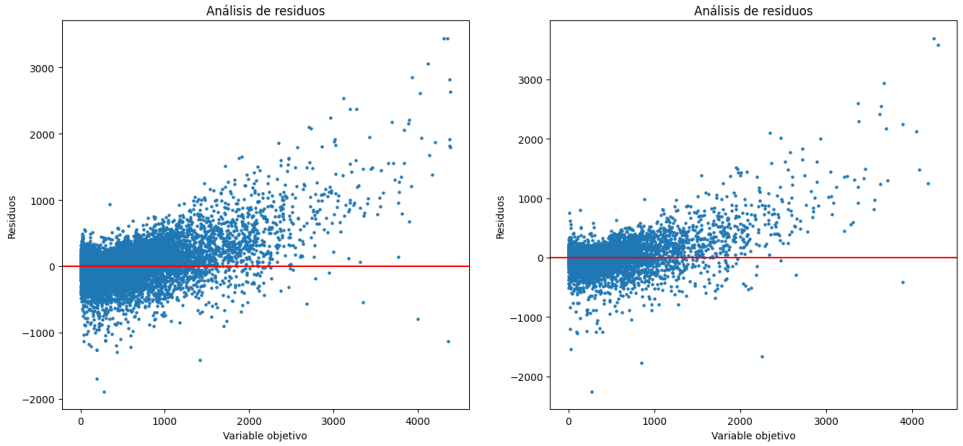
MSE	117176.44
R2	0.6754
R2 adj	0.6568

Figura No. (37) [Modelo 3] Residuos en validación del modelo de regresión múltiple.



MSE	171745.52
R2	0.4800
R2 adj	0.4781

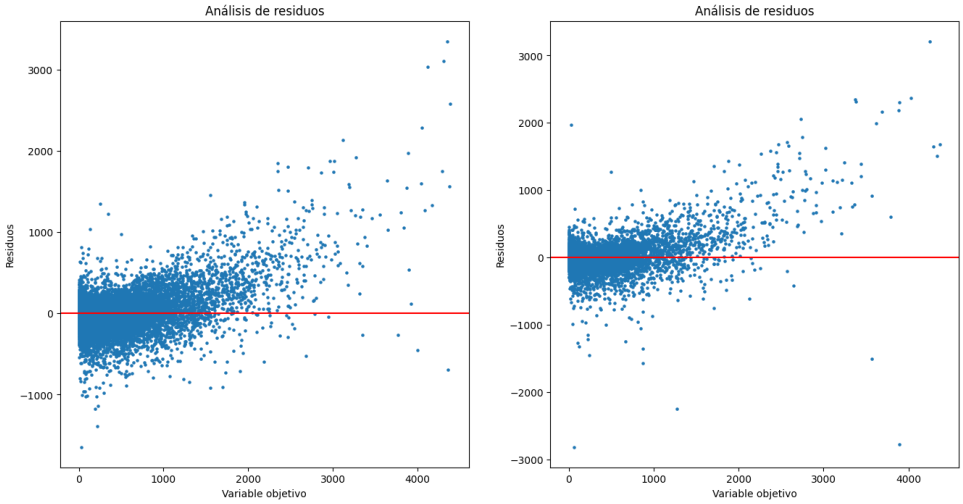
Figura No. (38) [Modelo 4] Residuos en validación del modelo de regresión múltiple.



MSE	19219.55
R2	0.8236
R2 adj	0.8134

MSE	21333.55
R2	0.8078
R2 adj	0.7796

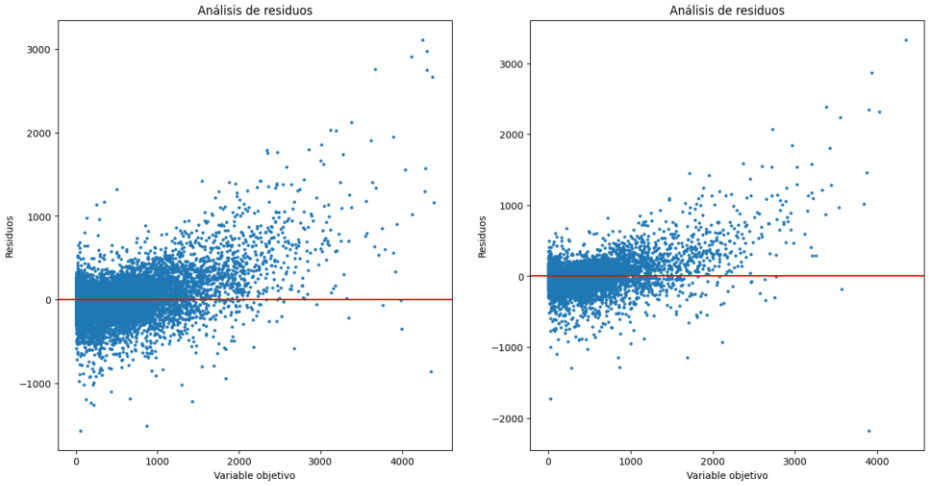
Figura No. (39) [Modelo 6] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo de regresión múltiple.



MSE	14454.30
R2	0.8490
R2 adj	0.8405

MSE	19120.25
R2	0.8153
R2 adj	0.7894

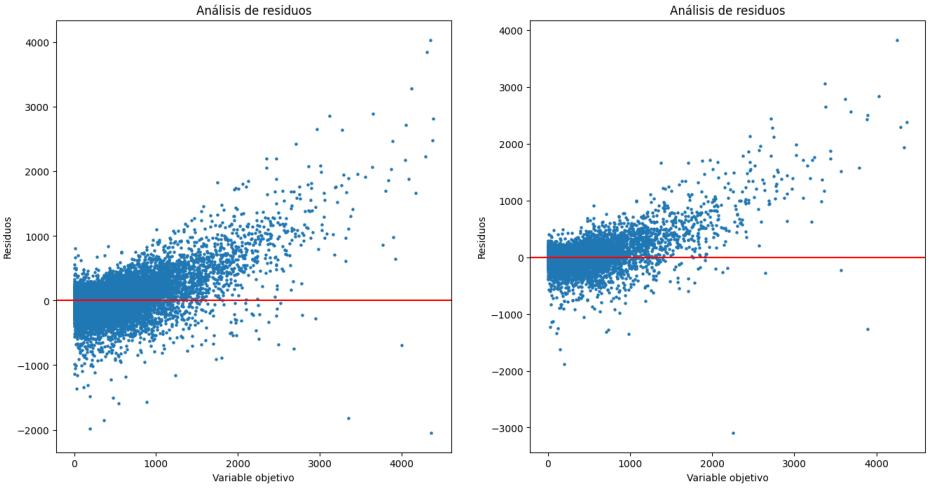
Figura No. (40) [Modelo 7] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo de regresión múltiple.



MSE	14675.42
R2	0.8596
R2 adj	0.8518

MSE	16861.97
R2	0.8466
R2 adj	0.8251

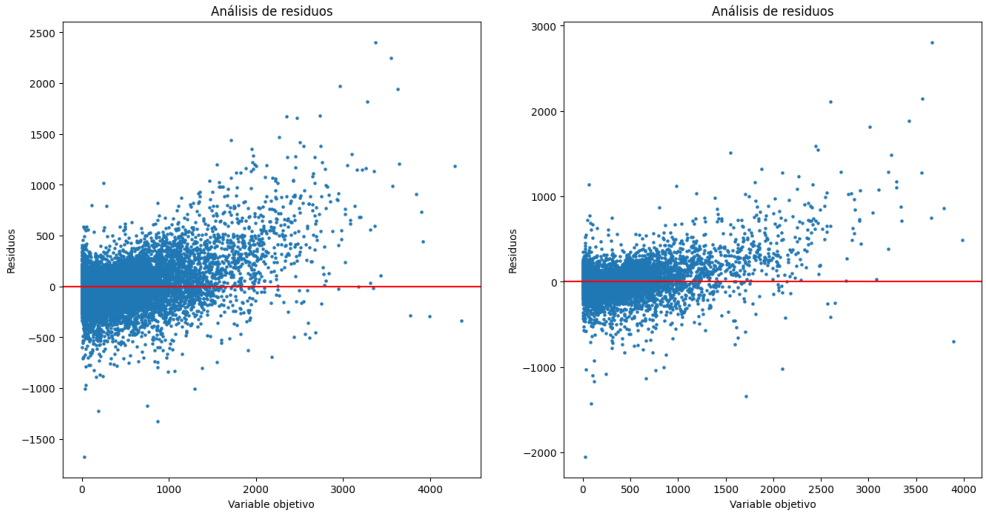
Figura No. (41) [Modelo 8] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo de regresión múltiple ponderada.



MSE	24437.80
R2	0.7446
R2 adj	0.7437

MSE	28166.49
R2	0.7279
R2 adj	0.7257

Figura No. (42) [Modelo 9] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo de regresión de componentes principales.

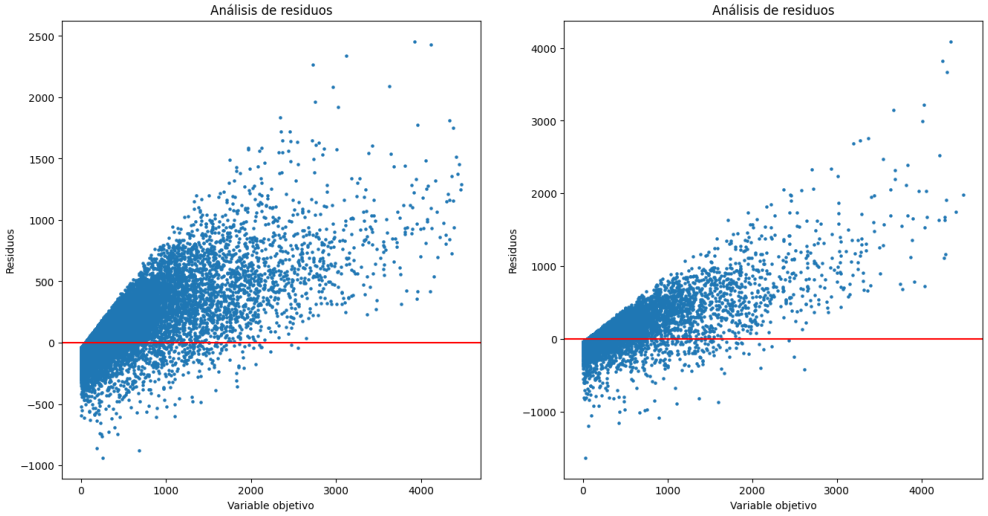


MSE	11610.60
R2	0.8765
R2 adj	0.8761

MSE	13892.98
R2	0.8588
R2 adj	0.8577

Figura No. (43) [Modelo 10] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo de regresión de mínimos cuadrados parciales.

8.2 Modelos basados en árboles



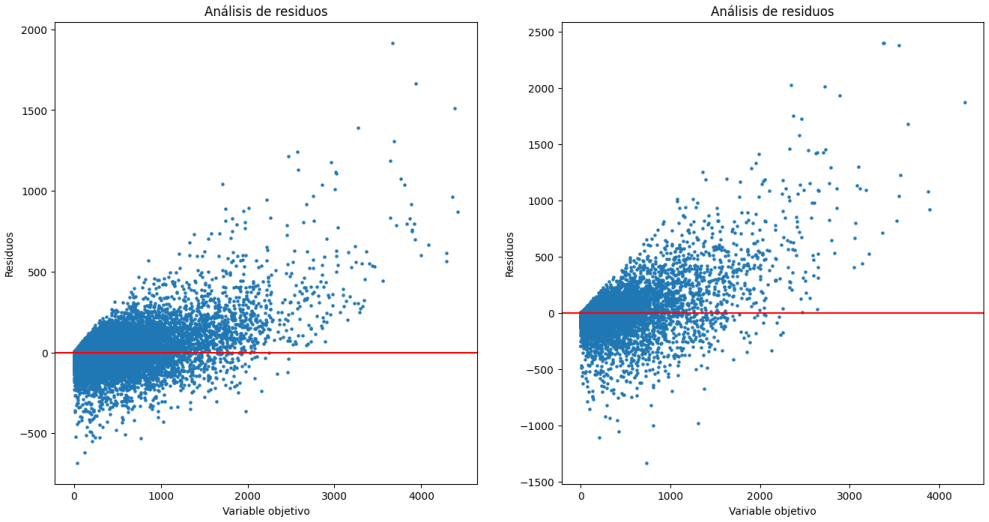
MSE	25254.92
R2	0.8301
R2 adj	0.8296

MSE	40320.26
R2	0.7422
R2 adj	0.7403

Figura No. (44) [Modelo 11] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo Random Forest.

Trial	max_depth	min_samples_split	min_samples_leaf	max_features	MSE
0	12	2	2	sqrt	16.323,73
1	2	11	5	sqrt	41.509,37
2	6	7	5	log2	28.938,18
3	12	12	5	sqrt	16.979,60
4	4	8	2	sqrt	29.533,82
5	12	12	4	log2	19.455,06
6	1	8	1	log2	64.874,26
7	14	6	5	log2	18.609,57
8	7	11	1	sqrt	21.490,04
9	5	6	4	sqrt	26.168,74
10	10	2	3	sqrt	17.638,99
11	12	9	2	sqrt	16.680,86
12	9	9	2	sqrt	18.936,89
13	12	2	2	sqrt	16.646,11
14	3	2	2	sqrt	34.160,21
15	15	3	2	sqrt	15.777,40
16	15	3	3	sqrt	15.976,64
17	15	3	3	sqrt	15.963,63
18	15	3	3	log2	17.854,64
19	8	10	3	sqrt	19.862,49
20	11	5	3	sqrt	17.182,27
21	15	3	3	sqrt	15.940,06
22	15	3	3	sqrt	15.817,47
23	15	3	3	sqrt	15.889,26
24	13	4	3	sqrt	16.267,50
25	15	3	1	sqrt	15.913,94
26	15	3	4	log2	18.214,68
27	15	3	3	sqrt	15.798,98
28	14	4	3	sqrt	16.060,73
29	11	10	2	sqrt	17.259,78
30	2	7	2	sqrt	41.626,92
31	15	3	3	sqrt	15.778,51
32	15	3	3	sqrt	15.982,58
33	6	3	3	sqrt	23.720,65
34	15	5	3	sqrt	15.895,20
35	4	3	3	sqrt	29.656,98
36	13	11	5	log2	18.995,47
37	10	12	4	sqrt	17.940,91
38	8	7	5	log2	24.173,53
39	1	8	1	sqrt	53.964,71
40	7	3	2	sqrt	21.224,52
41	15	3	3	sqrt	15.843,69
42	15	3	3	sqrt	15.808,54
43	5	6	3	sqrt	26.157,17
44	9	3	3	sqrt	18.710,55
45	3	3	5	sqrt	34.523,27
46	15	9	4	log2	18.283,04
47	2	12	1	sqrt	41.945,41
48	6	11	2	sqrt	23.424,57
49	15	8	3	sqrt	15.862,60
15	**15**	**3**	**2**	**sqrt**	**15.777,40**

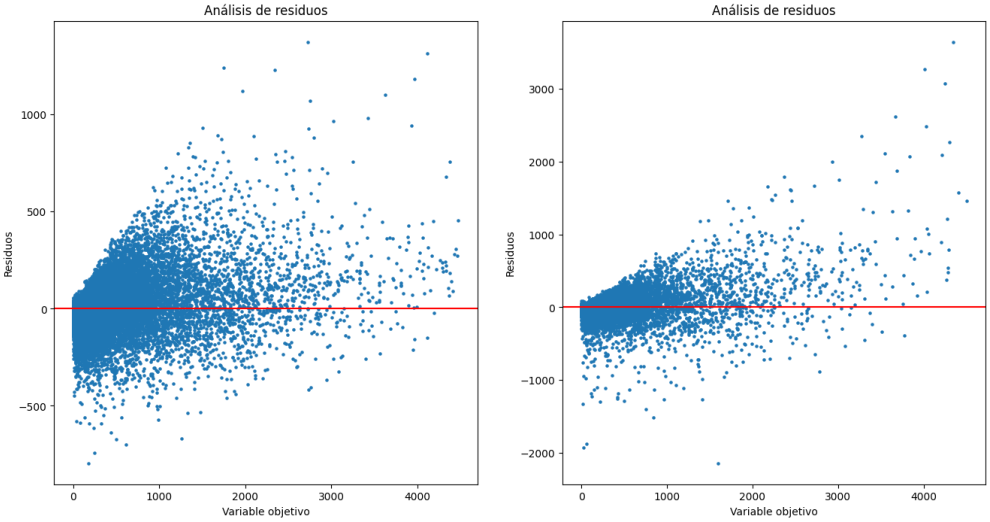
Tabla No. (20) [Modelo 12] Resultados de búsqueda de hiperparámetros bayesiana para modelo de árboles Random Forest.



MSE	4326.84
R2	0.9584
R2 adj	0.9561

MSE	15786.47
R2	0.8464
R2 adj	0.8250

Figura No. (45) [Modelo 12] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo Random Forest con *beyesian search*.



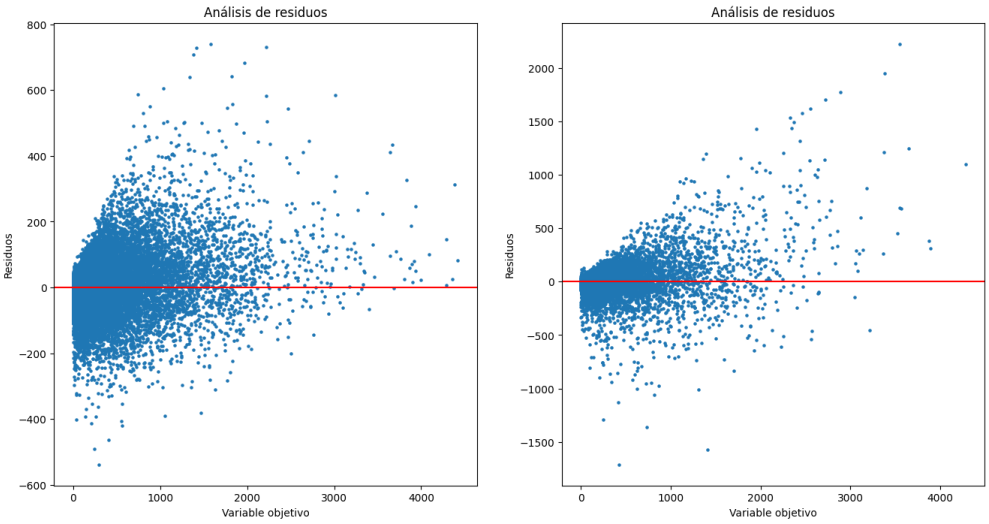
MSE	6388.29
R2	0.9570
R2 adj	0.9569

MSE	19465.61
R2	0.8756
R2 adj	0.8746

Figura No. (46) [Modelo 13] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo XGBoost.

Trial	max_depth	alpha	lambda	n_estimators	learning_rate	colsample_bytree	MSE
0	16	2,18	2,15	309	0,71	0,30	21.126,86
1	7	1,86	2,45	501	0,64	0,20	17.659,03
2	19	0,02	2,44	846	0,70	0,90	24.231,69
3	14	2,77	3,18	606	0,24	0,90	15.262,10
4	15	8,18	3,29	501	0,17	0,70	13.965,66
5	10	2,54	1,34	376	0,49	0,60	15.680,95
6	15	9,40	0,64	683	0,22	0,10	14.630,82
7	14	3,27	4,83	856	0,97	1,00	24.961,15
8	13	3,49	4,89	635	0,67	0,30	19.673,69
9	7	4,91	2,10	644	0,50	1,00	13.899,22
10	7	2,35	3,62	424	0,46	1,00	13.762,95
11	7	4,11	4,07	203	0,40	1,00	12.680,70
12	7	6,83	2,79	433	0,89	1,00	19.011,88
13	20	3,59	1,53	491	0,61	0,50	20.092,32
14	1	7,90	0,21	424	0,53	0,40	21.818,41
15	11	5,44	4,24	203	0,01	1,00	20.291,33
16	4	8,22	4,64	424	0,70	0,80	14.566,57
17	12	9,20	1,27	203	0,44	1,00	15.514,80
18	3	7,71	1,35	836	0,45	0,80	12.191,14
19	3	2,17	3,84	836	0,40	0,80	11.950,61
20	3	7,13	3,77	961	0,17	0,80	11.570,67
21	3	4,59	0,94	961	0,96	0,80	17.559,12
22	3	8,68	2,46	836	0,28	0,80	11.703,04
23	3	6,70	2,49	836	0,90	0,80	15.199,38
24	9	8,29	3,90	529	0,25	0,80	12.182,49
25	3	8,83	2,20	971	0,12	0,80	12.106,33
26	18	3,36	3,73	117	0,79	0,80	22.954,52
27	5	0,39	3,76	961	0,41	0,50	12.548,41
28	6	9,04	3,65	184	0,97	0,20	21.536,07
29	8	5,32	4,95	241	0,28	0,40	12.133,23
20	3	7,13	3,77	961	0,17	0,80	11.570,67

Tabla No. (21) [Modelo 14] Resultados de búsqueda de hiperparámetros bayesiana para modelo de árboles XGBoost.



MSE	2869.32
R2	0.9724
R2 adj	0.9709

MSE	11739.99
R2	0.8858
R2 adj	0.8698

Figura No. (47) [Modelo 14] Residuos en entrenamiento (izquierda) y validación (derecha) del modelo XGBoost con *bayesian search*.

8.3 Modelos de redes neuronales

Módulo	Parámetros
embedding	num_emb = 85, emb_dim = 8
lineal_1	in = 2924 + 8, out = 1600, bias = True
relu_1	
dropout_1	prob = 0.20
lineal_2	in = 1600, out = 800, bias = True
relu_2	
dropout_2	prob = 0.20
lineal_3	in = 800, out = 200, bias = True
relu_3	
dropout_3	prob = 0.20
lineal_4	in = 200, out = 1, bias = True

Tabla No. (22) [Modelo 16] Arquitectura y parámetros de red neuronal *feed-forward*.

Módulo	Parámetros
embedding.weight	680
linear_1.weight	4.691.200
linear_1.bias	1.600
linear_2.weight	1.280.000
linear_2.bias	800
linear_3.weight	160.000
linear_3.bias	200
linear_4.weight	200
linear_4.bias	1
Total parám. entrenables	**6.134.681**

Tabla No. (23) [Modelo 16] Cantidad de parámetros entrenables de red neuronal *feed-forward*.

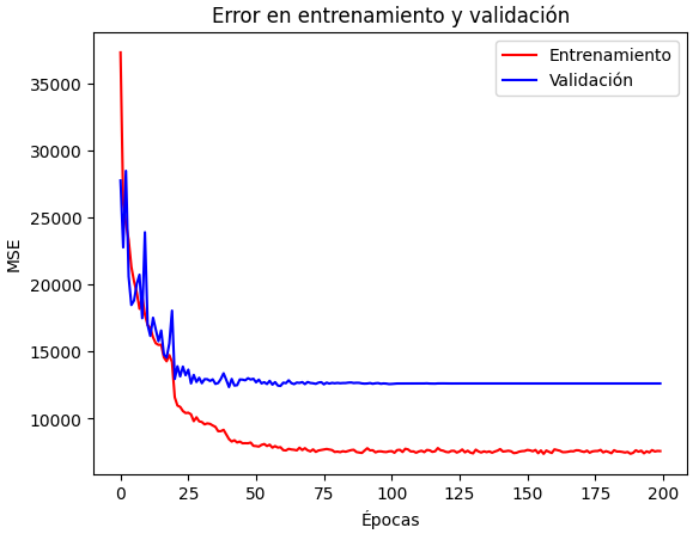
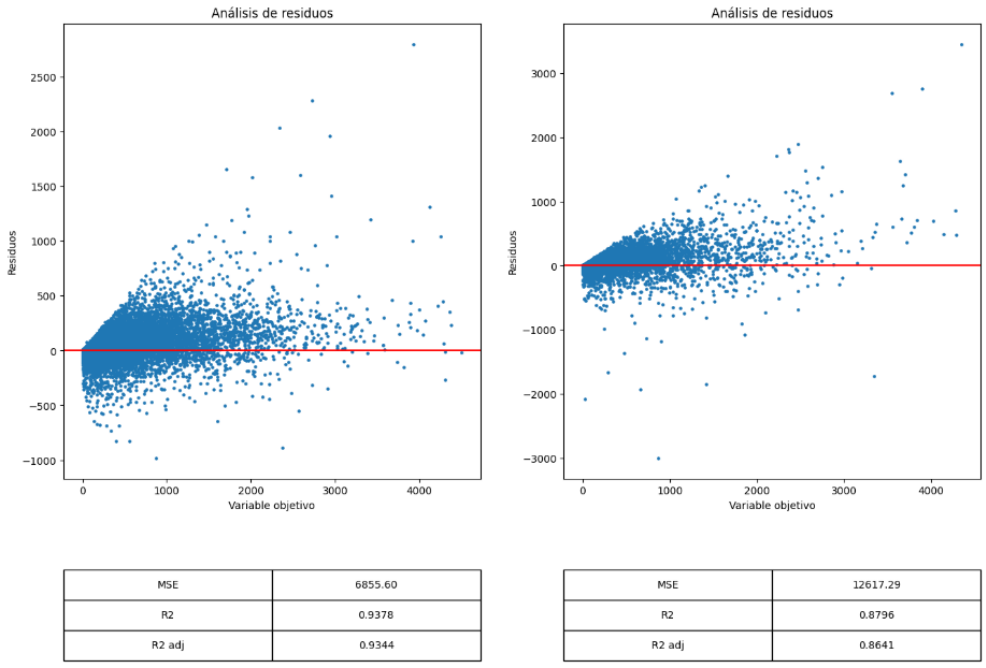


Figura No. (48) [Modelo 16] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales *feed-forward*.



MSE	6855.60
R2	0.9378
R2 adj	0.9344

MSE	12617.29
R2	0.8796
R2 adj	0.8641

Figura No. (49) [Modelo 16] Residuos en entrenamiento (izquierda) y validación (derecha) de modelo de redes neuronales *feed-forward*.

Módulo	Parámetros
embedding	num_emb = 85, emb_dim = 8
lineal_1	in = 2924 + 8, out = 2500, bias = True
relu_1	
dropout_1	prob = 0.20
lineal_2	in = 2500, out = 1000, bias = True
relu_2	
dropout_2	prob = 0.20
lineal_3	in = 1000, out = 200, bias = True
relu_3	
dropout_3	prob = 0.20
lineal_4	in = 200, out = 1, bias = True

Tabla No. (24) [Modelo 17] Arquitectura y parámetros de red neuronal *feed-forward*.

Módulo	Parámetros
embedding.weight	680
linear_1.weight	7.330.000
linear_1.bias	2.500
linear_2.weight	2.500.000
linear_2.bias	1.000
linear_3.weight	200.000
linear_3.bias	200
linear_4.weight	200
linear_4.bias	1
Total parám. entrenables	**10.034.581**

Tabla No. (25) [Modelo 17] Cantidad de parámetros entrenables de red neuronal *feed-forward*.

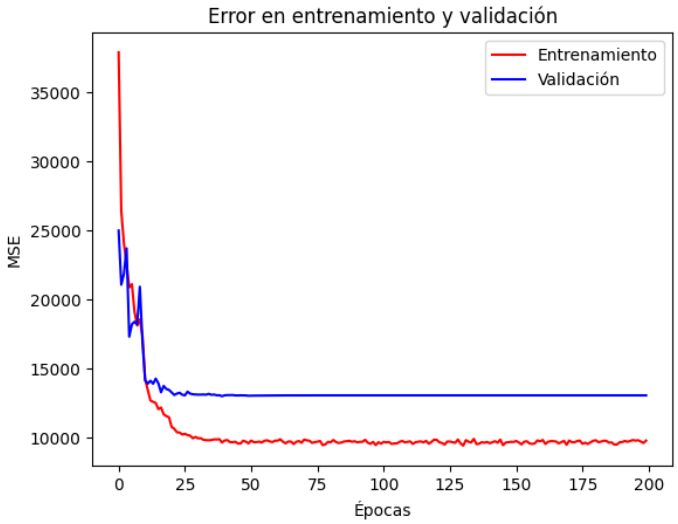
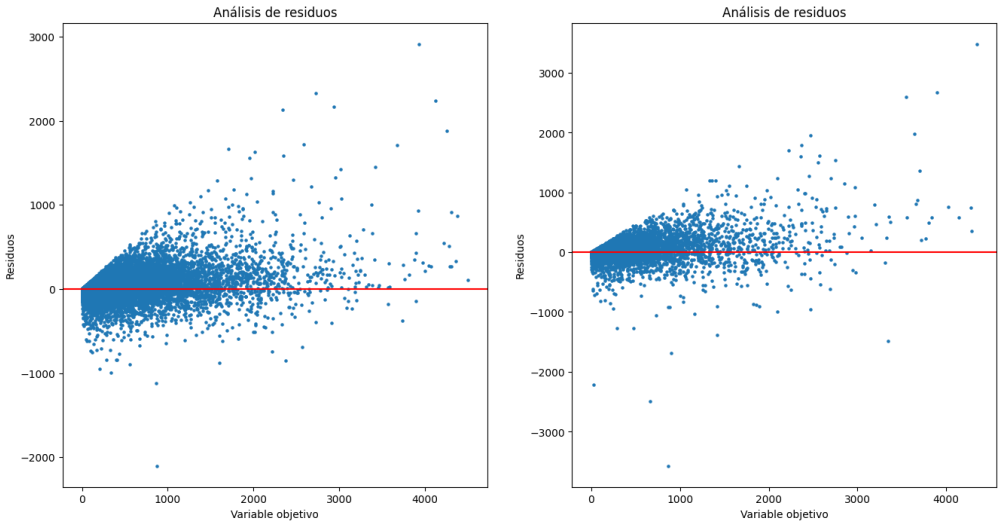


Figura No. (50) [Modelo 17] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales *feed-forward*.



MSE	8402.36
R2	0.9238
R2 adj	0.9196

MSE	13070.35
R2	0.8753
R2 adj	0.8593

Figura No. (51) [Modelo 17] Residuos en entrenamiento (izquierda) y validación (derecha) de modelo de redes neuronales *feed-forward*.

Módulo	Parámetros
rnn_1	in = 9, hid_size = 256, num_layers = 1
embedding	num_emb = 92, emb_dim = 8
lineal_1	in = 256 + 8 + 22, out = 1, bias = True

Tabla No. (26) [Modelo 18] Arquitectura y parámetros de red neuronal recurrente.

Módulo	Parámetros
rnn.weight_ih_l0	2.304
rnn.weight_hh_l0	65.536
rnn.bias_ih_l0	256
rnn.bias_hh_l0	256
embedding.weight	736
linear_1.weight	285
linear_1.bias	1
Total parám. entrenables	**69.374**

Tabla No. (27) [Modelo 18] Cantidad de parámetros entrenables de red neuronal recurrente.

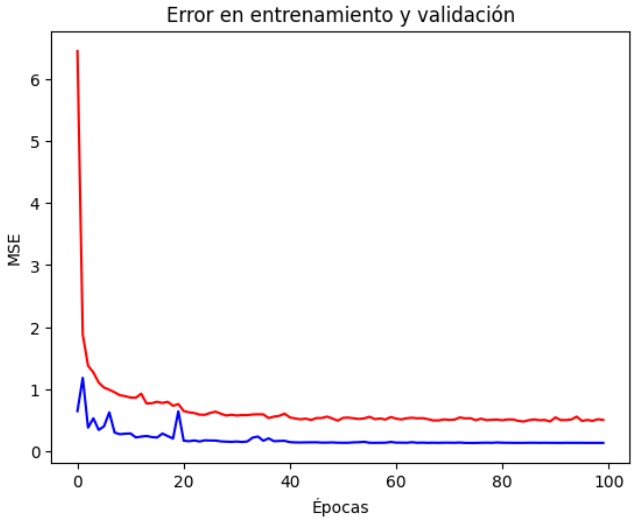


Figura No. (52) [Modelo 18] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales recurrente.

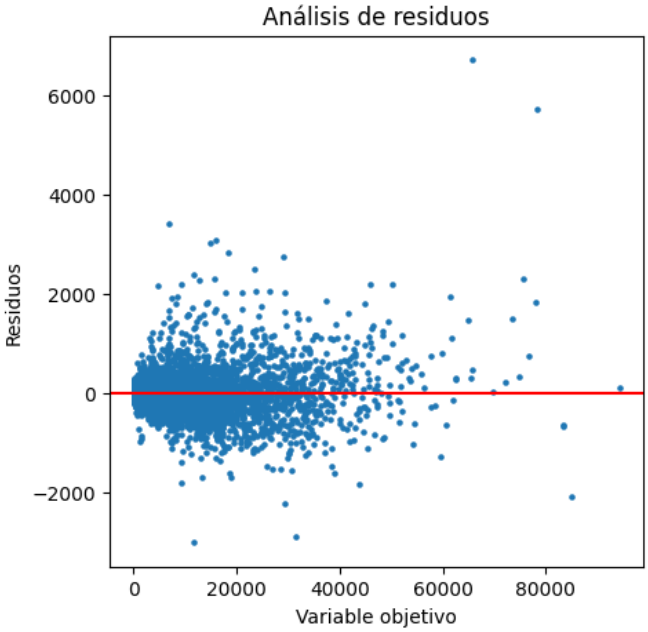


Figura No. (53) [Modelo 18] Residuos en validación de modelo de redes neuronales recurrente.

Módulo	Parámetros
lstm_1	in = 13, hid_size = 1280, num_layers = 4
embedding	num_emb = 92, emb_dim = 8
lineal_1	in = 1280 + 8 + 22, out = 4000, bias = True
relu_1	
dropout_1	prob = 0.20
lineal_2	in = 4000, out = 2000, bias = True
relu_2	
dropout_2	prob = 0.20
lineal_3	in = 2000, out = 500, bias = True
relu_3	
dropout_3	prob = 0.20
lineal_4	in = 500, out = 100, bias = True
dropout_4	prob = 0.20
lineal_5	in = 100, out = 1, bias = True

Tabla No. (28) [Modelo 19] Arquitectura y parámetros de red neuronal recurrente.

Módulo	Parámetros
rnn.weight_ih_l0	13.312
rnn.weight_hh_l0	1.048.576
rnn.bias_ih_l0	1.024
rnn.bias_hh_l0	1.024
rnn.weight_ih_l1	1.048.576
rnn.weight_hh_l1	1.048.576
rnn.bias_ih_l1	1.024
rnn.bias_hh_l1	1.024
rnn.weight_ih_l2	1.048.576
rnn.weight_hh_l2	1.048.576
rnn.bias_ih_l2	1.024
rnn.bias_hh_l2	1.024
rnn.weight_ih_l3	1.048.576
rnn.weight_hh_l3	1.048.576
rnn.bias_ih_l3	1.024
rnn.bias_hh_l3	1.024
embedding.weight	736
linear_1.weight	4.212.000
linear_1.bias	4.000
linear_2.weight	8.000.000
linear_2.bias	2.000
linear_3.weight	1.000.000
linear_3.bias	500
linear_4.weight	50.000
linear_4.bias	100
linear_5.weight	100
linear_5.bias	1
Total parám. entrenables	**20.630.973**

Tabla No. (29) [Modelo 19] Cantidad de parámetros entrenables de red neuronal recurrente.

Figura No. (54) [Modelo 19] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales recurrente.

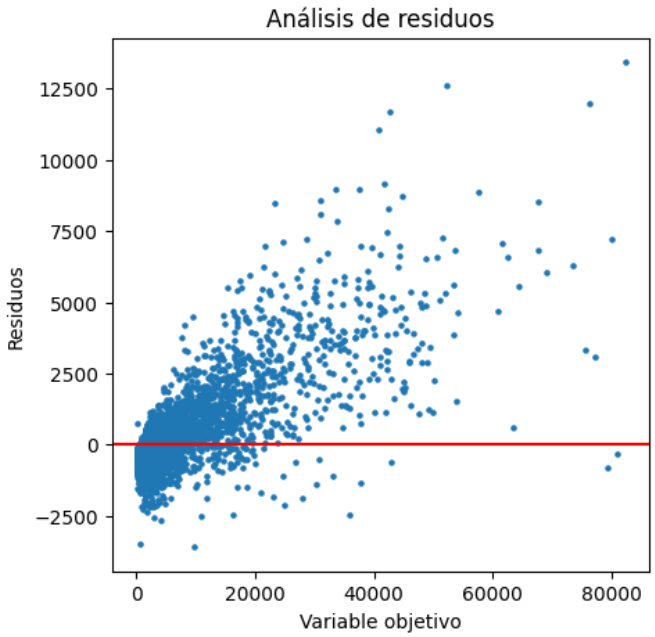


Figura No. (55) [Modelo 19] Residuos en validación de modelo de redes neuronales recurrente.

Módulo	Parámetros
lstm_1	in = 9, hid_size = 1280, num_layers = 5
embedding	num_emb = 92, emb_dim = 8
lineal_1	in = 1280 + 8 + 22, out = 4000, bias = True
relu_1	
dropout_1	prob = 0.20
lineal_2	in = 4000, out = 2000, bias = True
relu_2	
dropout_2	prob = 0.20
lineal_3	in = 2000, out = 500, bias = True
relu_3	
dropout_3	prob = 0.20
lineal_4	in = 500, out = 100, bias = True
dropout_4	prob = 0.20
lineal_5	in = 100, out = 1, bias = True

Tabla No. (30) [Modelo 20] Arquitectura y parámetros de red neuronal recurrente.

Módulo	Parámetros
rnn.weight_ih_l0	9.216
rnn.weight_hh_l0	1.048.576
rnn.bias_ih_l0	1.024
rnn.bias_hh_l0	1.024
rnn.weight_ih_l1	1.048.576
rnn.weight_hh_l1	1.048.576
rnn.bias_ih_l1	1.024
rnn.bias_hh_l1	1.024
rnn.weight_ih_l2	1.048.576
rnn.weight_hh_l2	1.048.576
rnn.bias_ih_l2	1.024
rnn.bias_hh_l2	1.024
rnn.weight_ih_l3	1.048.576
rnn.weight_hh_l3	1.048.576
rnn.bias_ih_l3	1.024
rnn.bias_hh_l3	1.024
rnn.weight_ih_l4	1.048.576
rnn.weight_hh_l4	1.048.576
rnn.bias_ih_l4	1.024
rnn.bias_hh_l4	1.024
embedding.weight	736
linear_1.weight	4.212.000
linear_1.bias	4.000
linear_2.weight	8.000.000
linear_2.bias	2.000
linear_3.weight	1.000.000
linear_3.bias	500
linear_4.weight	50.000
linear_4.bias	100
linear_5.weight	100
linear_5.bias	1
Total parám. entrenables	**22.726.077**

Tabla No. (31) [Modelo 20] Cantidad de parámetros entrenables de red neuronal recurrente.

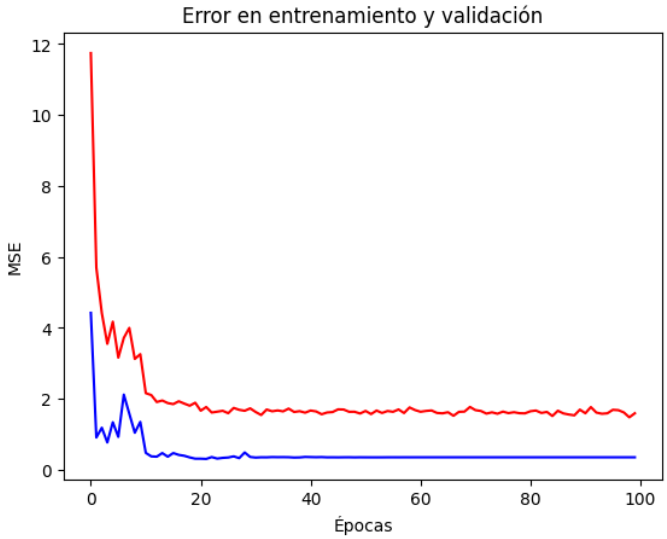


Figura No. (56) [Modelo 20] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales recurrente.

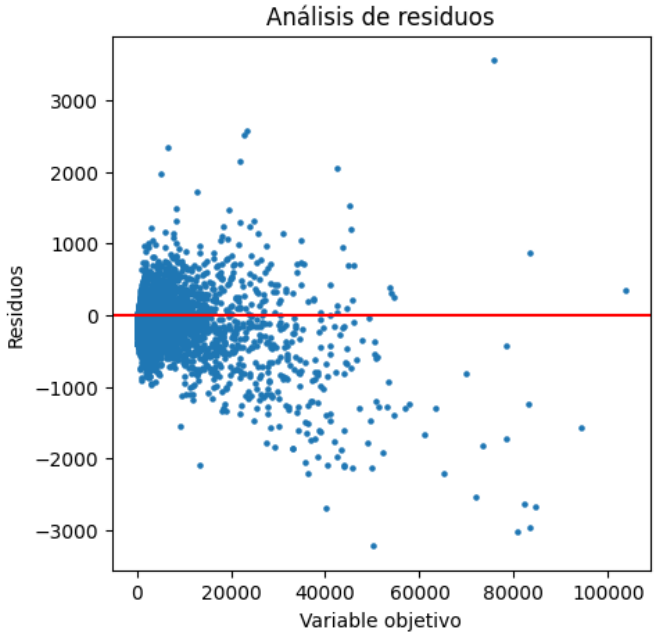


Figura No. (57) [Modelo 20] Residuos en validación de modelo de redes neuronales recurrente.

Módulo	Parámetros
lstm_1	in = 9, hid_size = 1280, num_layers = 5
embedding	num_emb = 92, emb_dim = 8
lineal_1	in = 1280 + 8 + 22, out = 4000, bias = True
relu_1	
dropout_1	prob = 0.20
lineal_2	in = 4000, out = 2000, bias = True
relu_2	
dropout_2	prob = 0.20
lineal_3	in = 2000, out = 500, bias = True
relu_3	
dropout_3	prob = 0.20
lineal_4	in = 500, out = 100, bias = True
dropout_4	prob = 0.20
lineal_5	in = 100, out = 1, bias = True

Tabla No. (32) [Modelo 21] Arquitectura y parámetros de red neuronal recurrente

Módulo	Parámetros
lstm.weight_ih_l0	46.080
lstm.weight_hh_l0	6.553.600
lstm.bias_ih_l0	5.120
lstm.bias_hh_l0	5.120
lstm.weight_ih_l1	6.553.600
lstm.weight_hh_l1	6.553.600
lstm.bias_ih_l1	5.120
lstm.bias_hh_l1	5.120
lstm.weight_ih_l2	6.553.600
lstm.weight_hh_l2	6.553.600
lstm.bias_ih_l2	5.120
lstm.bias_hh_l2	5.120
lstm.weight_ih_l3	6.553.600
lstm.weight_hh_l3	6.553.600
lstm.bias_ih_l3	5.120
lstm.bias_hh_l3	5.120
lstm.weight_ih_l4	6.553.600
lstm.weight_hh_l4	6.553.600
lstm.bias_ih_l4	5.120
lstm.bias_hh_l4	5.120
embedding.weight	736
linear_1.weight	5.240.000
linear_1.bias	4.000
linear_2.weight	8.000.000
linear_2.bias	2.000
linear_3.weight	1.000.000
linear_3.bias	500
linear_4.weight	50.000
linear_4.bias	100
linear_5.weight	100
linear_5.bias	1
Total parám. entrenables	**73.377.117**

Tabla No. (33) [Modelo 21] Cantidad de parámetros entrenables de red neuronal recurrente.

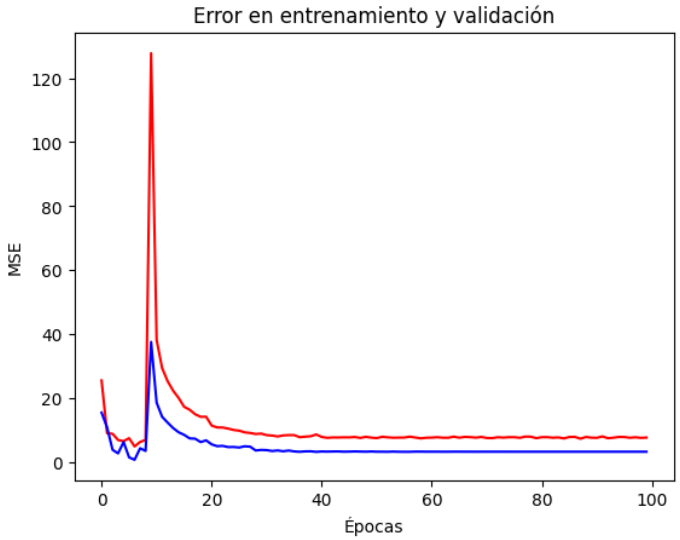


Figura No. (58) [Modelo 21] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales recurrente.

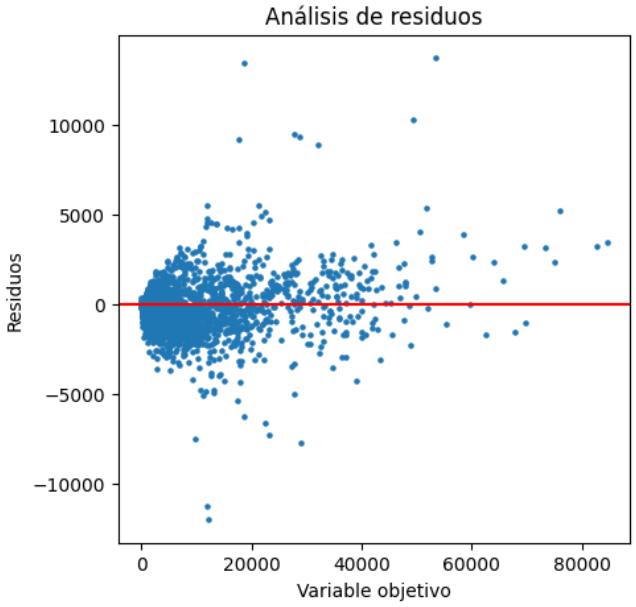


Figura No. (59) [Modelo 21] Residuos en validación de modelo de redes neuronales recurrente.

Módulo	Parámetros
lstm_1	in = 9, hid_size = 256, num_layers = 1
embedding	num_emb = 92, emb_dim = 8
lineal_1	in = 128 + 8 + 22, out = 1, bias = True

Tabla No. (34) [Modelo 22] Arquitectura y parámetros de red neuronal recurrente

Módulo	Parámetros
lstm.weight_ih_l0	9.216
lstm.weight_hh_l0	262.144
lstm.bias_ih_l0	1.024
lstm.bias_hh_l0	1.024
embedding.weight	736
linear_1.weight	287
linear_1.bias	1
Total parám. entrenables	**274.432**

Tabla No. (35) [Modelo 22] Cantidad de parámetros entrenables de red neuronal recurrente.

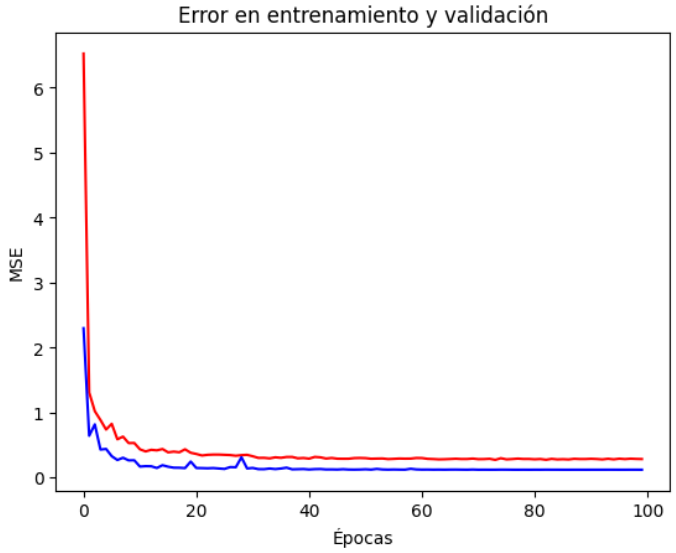


Figura No. (60) [Modelo 22] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales recurrente.

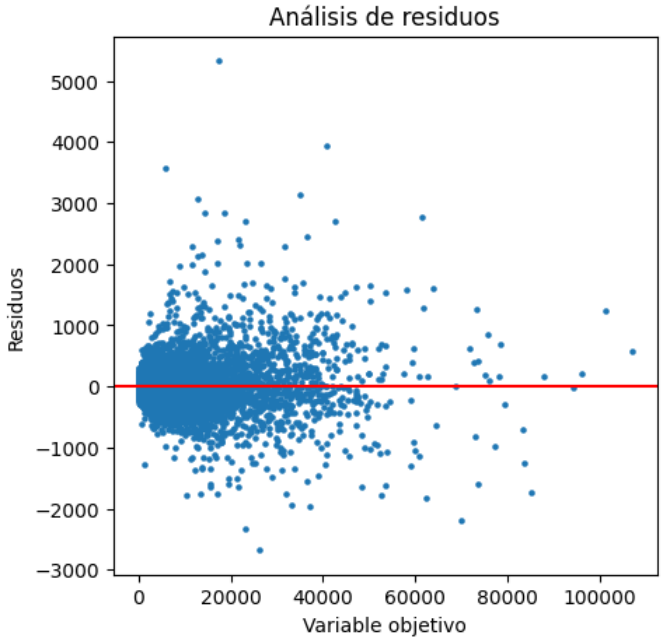


Figura No. (61) [Modelo 22] Residuos en validación de modelo de redes neuronales recurrente.

Módulo	Parámetros
embedding	num_emb = 92, emb_dim = 8
lstm_encoder	in = 12, hid_size = 1024, num_layers = 2, drp = 0.20
lstm_dencoder	in = 12, hid_size = 1024, num_layers = 2, drp = 0.20
lineal_1	in = 1024 + 8 + 22, out = 12, bias = True

Tabla No. (36) [Modelo 23] Arquitectura y parámetros de red neuronal *encoder-decoder*.

Módulo	Parámetros
embedding.weight	736
encoder_lstm.weight_ih_l0	24.576
encoder_lstm.weight_hh_l0	1.048.576
encoder_lstm.bias_ih_l0	2.048
encoder_lstm.bias_hh_l0	2.048
encoder_lstm.weight_ih_l1	1.048.576
encoder_lstm.weight_hh_l1	1.048.576
encoder_lstm.bias_ih_l1	2.048
encoder_lstm.bias_hh_l1	2.048
decoder_lstm.weight_ih_l0	24.576
decoder_lstm.weight_hh_l0	1.048.576
decoder_lstm.bias_ih_l0	2.048
decoder_lstm.bias_hh_l0	2.048
decoder_lstm.weight_ih_l1	1.048.576
decoder_lstm.weight_hh_l1	1.048.576
decoder_lstm.bias_ih_l1	2.048
decoder_lstm.bias_hh_l1	2.048
enc_fc.weight	276.480
enc_fc.bias	512
dec_fc.weight	6.480
dec_fc.bias	12
Total parám. entrenables	**6.641.212**

Tabla No. (37) [Modelo 23] Cantidad de parámetros entrenables de red neuronal *encoder-decoder*.

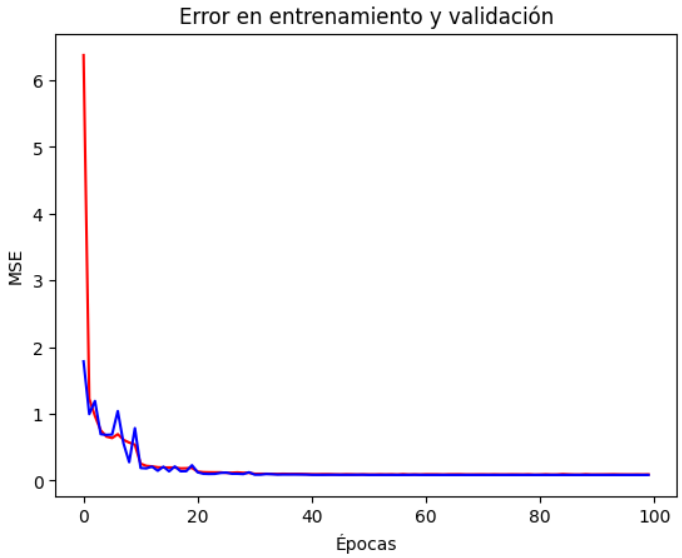


Figura No. (62) [Modelo 23] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales *encoder-decoder*.

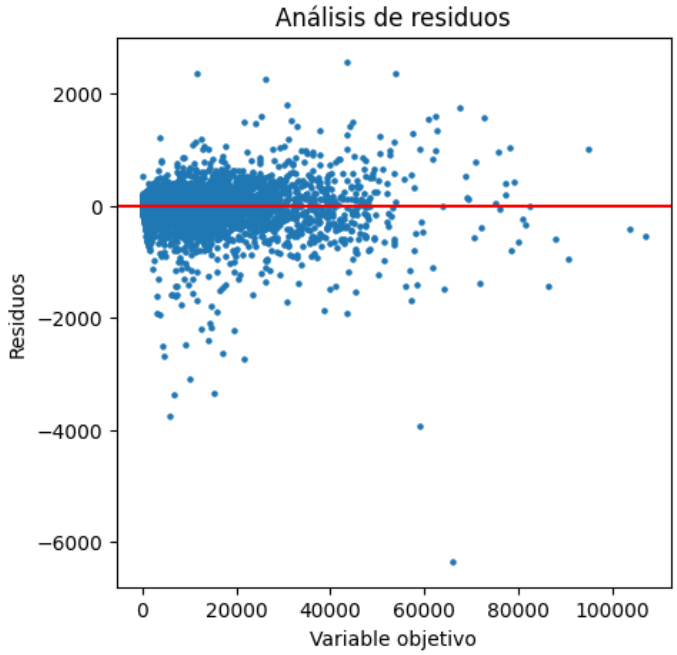


Figura No. (63) [Modelo 23] Residuos en validación de modelo de redes neuronales *encoder-decoder*.

Módulo	Parámetros
embedding	num_emb = 90, emb_dim = 8
lstm_encoder	in = 12, hid_size = 1024, num_layers = 2, drp = 0.20, dirs = 2
lstm_dencoder	in = 12, hid_size = 1024, num_layers = 2, drp = 0.20, dirs = 2
lineal_1	in = 1024 * 2 + 3, out = 12, bias = True
lineal_2	in = 12, out = 1, bias = True

Tabla No. (38) [Modelo 24] Arquitectura y parámetros de red neuronal *encoder-decoder*.

Módulo	Parámetros
embedding.weight720	720
encoder_lstm.weight_ih_l0	155.648
encoder_lstm.weight_hh_l0	4.194.304
encoder_lstm.bias_ih_l0	4.096
encoder_lstm.bias_hh_l0	4.096
encoder_lstm.weight_ih_l0_reverse	155.648
encoder_lstm.weight_hh_l0_reverse	4.194.304
encoder_lstm.bias_ih_l0_reverse	4.096
encoder_lstm.bias_hh_l0_reverse	4.096
encoder_lstm.weight_ih_l1	8.388.608
encoder_lstm.weight_hh_l1	4.194.304
encoder_lstm.bias_ih_l1	4.096
encoder_lstm.bias_hh_l1	4.096
encoder_lstm.weight_ih_l1_reverse	8.388.608
encoder_lstm.weight_hh_l1_reverse	4.194.304
encoder_lstm.bias_ih_l1_reverse	4.096
encoder_lstm.bias_hh_l1_reverse	4.096
decoder_lstm.weight_ih_l0	155.648
decoder_lstm.weight_hh_l0	4.194.304
decoder_lstm.bias_ih_l0	4.096
decoder_lstm.bias_hh_l0	4.096
decoder_lstm.weight_ih_l0_reverse	155.648
decoder_lstm.weight_hh_l0_reverse	4.194.304
decoder_lstm.bias_ih_l0_reverse	4.096
decoder_lstm.bias_hh_l0_reverse	4.096
decoder_lstm.weight_ih_l1	8.388.608
decoder_lstm.weight_hh_l1	4.194.304
decoder_lstm.bias_ih_l1	4.096
decoder_lstm.bias_hh_l1	4.096
decoder_lstm.weight_ih_l1_reverse	8.388.608
decoder_lstm.weight_hh_l1_reverse	4.194.304
decoder_lstm.bias_ih_l1_reverse	4.096
decoder_lstm.bias_hh_l1_reverse	4.096
dec_fc1.weight	24.612
dec_fc1.bias	12
dec_fc2.weight	12
dec_fc2.bias	1
Total parám. entrenables	**67.822.349**

Tabla No. (39) [Modelo 24] Cantidad de parámetros entrenables de red neuronal *encoder-decoder*.

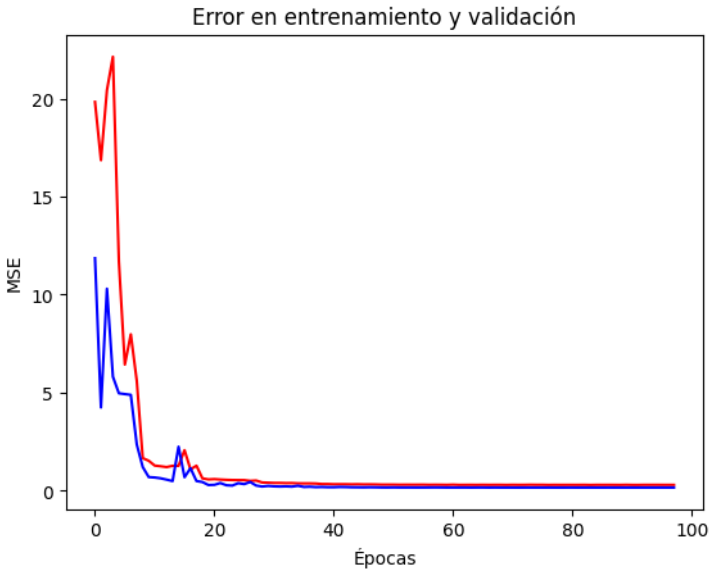


Figura No. (64) [Modelo 24] Evolución de los errores en entrenamiento (azul) y validación (rojo) para modelo de redes neuronales *encoder-decoder*.

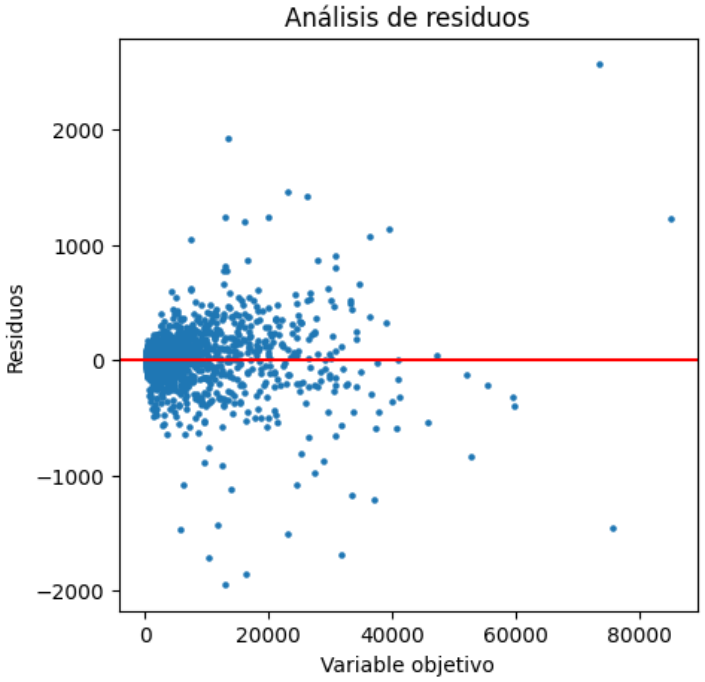


Figura No. (65) [Modelo 24] Residuos en validación de modelo de redes neuronales *encoder-decoder*.

8.4 Embeddings

Figura No. (66) Sumatoria del cuadrado de las distancias en función del número de *clusters* (representación t-SNE).

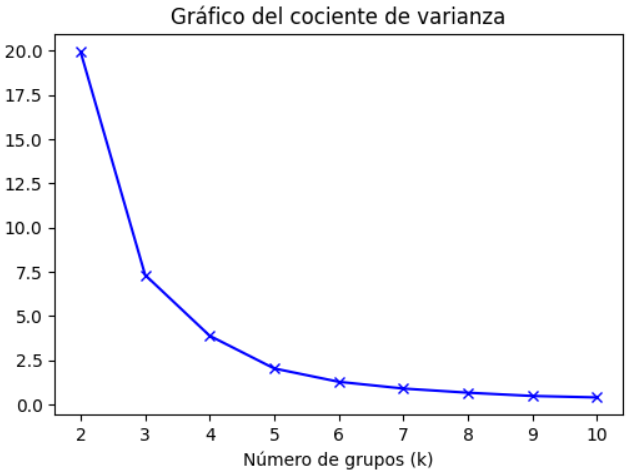


Figura No. (67) Cociente entre la varianza *intra-cluster* y varianza *inter-clusters* s en función del número de *clusters* (representación t-SNE).

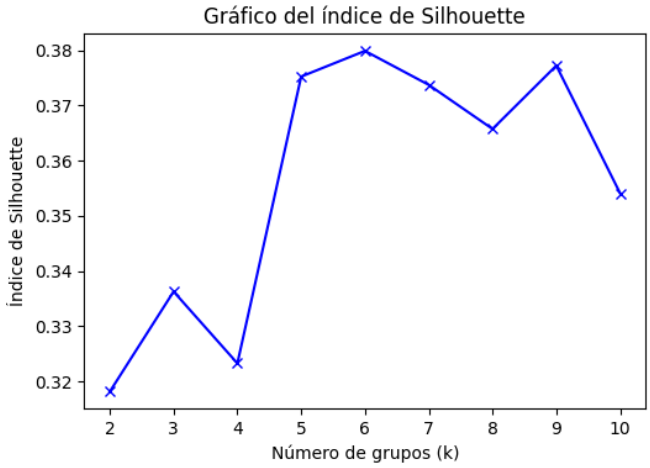


Figura No. (68) Índice de *Silhouette* en función del número de *clusters* (representación t-SNE).

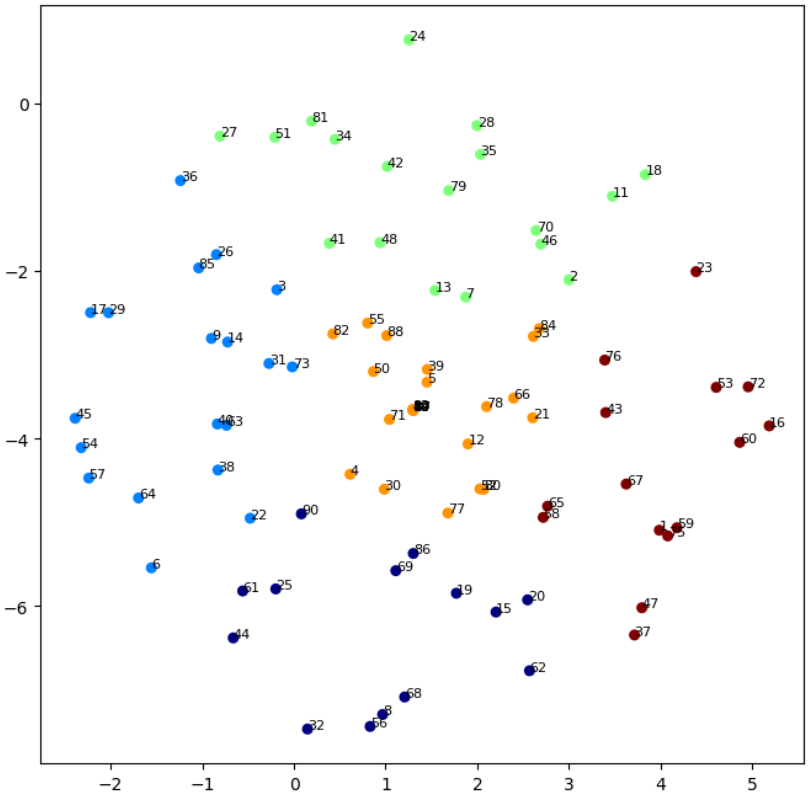


Figura No. (69) Agrupación final (representación t-SNE).