UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Modelos de predicción de tiempos de viaje vehicular

Informe de Proyecto de Grado presentado por

Nicolás Escobar, Leonela Pereira

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Libertad Tansini
Pedro Piñeyro, Carlos Testuri

Montevideo, 4 de abril de 2025

Modelos de predicción de tiempos de viaje vehicular por Nicolás Escobar, Leonela Pereira tiene licencia CC Atribución 4.0.

Agradecimientos

Queremos agradecer a nuestros tutores, cuyo compromiso y acompañamiento constante fueron fundamentales en cada etapa del proyecto. Su orientación y experiencia nos brindaron herramientas y perspectivas clave para el desarrollo del trabajo.

A nuestras familias y amigos, por su apoyo incondicional a lo largo de toda nuestra formación. Su aliento y comprensión fueron esenciales en este camino.

Resumen

Este proyecto se enmarca en una investigación sobre la predicción de tiempos de viaje en el transporte vehicular, entendido como la predicción de tiempo viaje futuro en base a datos históricos y en tiempo real. Para comprender el estado actual de la disciplina, se realiza una revisión sistemática de la literatura, donde se identifican múltiples factores clave que influyen en la predicción del tiempo de viaje, como la recopilación y el procesamiento de datos, los tipos de modelos existentes, las diferencias según el tipo de vehículo para el cual se quiera predecir y el contexto del transporte según si el entorno es urbano o autopista. A partir de este análisis, se decide abordar el problema de predecir los tiempos de viaje para la línea 117 de ómnibus de CUTCSA. Más precisamente, dada la parada actual en la que se encuentre el bus, el objetivo es predecir el tiempo de viaje que le lleva llegar a destino. Para abordar esta tarea, se desarrollan y evalúan tres de los enfoques de modelado relevados en la literatura: un modelo estadístico base (ARIMA), un modelo de aprendizaje automático (XGBoost) y un modelo de aprendizaje profundo basado en redes neuronales recurrentes bidireccionales (Bi-GRU). Se implementan estos modelos en Python, utilizando un entorno virtual (venv) para gestionar las dependencias de manera modular y reproducible. Los datos se dividen en conjuntos de entrenamiento, validación y prueba particionados en 70 %, 15 % y 15 %, respectivamente. Los experimentos realizados confirman lo hallado en el estado del arte: los modelos estadísticos como ARIMA no son adecuados para capturar la naturaleza no lineal del tráfico. En cambio, XGBoost muestra un rendimiento sólido en la predicción de series temporales, mientras que Bi-GRU logra la mejor precisión a costa de mayor tiempo de entrenamiento (casi seis horas en el peor caso frente a un minuto en XGBoost). Los hallazgos de este estudio pueden ser valiosos para la gestión del tráfico en Montevideo. Una predicción precisa de los tiempos de viaje permitiría optimizar la planificación del transporte público, ajustar frecuencias y horarios de los ómnibus, y mejorar la experiencia de los pasajeros. Además, este trabajo abre la puerta a futuras investigaciones que podrían enfocarse en la predicción del tiempo de viaje entre paradas específicas, proporcionando información más útil para los usuarios y operadores del sistema de transporte.

Palabras clave: Predicción de Tiempos de Viaje, Modelos Estadísticos, Aprendizaje Automático, ARIMA, XGBoost, Bi-GRU.

Índice general

Capítulo 1

Introducción

De acuerdo con un reporte de las Naciones Unidas, en 2018 la población urbana constituía aproximadamente el 55 % de la población mundial, equivalente a unos 4.2 billones de personas, y se prevé que para 2050 esta cifra ascienda a dos tercios de la población global (Ou, 2022; Chughtai, Haq, y Muneeb, 2022; Lee, Choo, Choi, y Lee, 2022). La rápida urbanización trae consigo una serie de oportunidades, así como nuevos desafíos. Entre estos se encuentra la congestión, entendida como la condición en la que más del 50 % de los registros tienen una velocidad vehicular inferior a 20 kilómetros por hora para un segmento de carretera dado (Fang, Liu, Chen, y Hwang, 2022). Esta congestión se ha convertido en un problema crítico en muchas ciudades, repercutiendo en la vida de las personas, en la productividad laboral y generando un consumo excesivo de combustible. Muchos gobiernos han intentado lidiar con este problema agregando más infraestructura, lo que en algunos casos ha resultado en un aumento del tráfico (Tang, Kanamori, y Yamamoto, 2019). Resolver este problema resulta clave, ya que el transporte es vital para el desarrollo urbano, facilitando el acceso a empleos, educación, servicios y contribuyendo significativamente al crecimiento económico (Ou, 2022). En este contexto, los Sistemas de Transporte Inteligente (ITS, por su sigla en inglés), que integran sensores de datos IoT (Internet de las cosas), cámaras y computadoras, proveen una solución al problema del tráfico. Dentro de los ITS, uno de los problemas más desafiantes es la predicción de tiempo de viaje (TTP, por su sigla en inglés) (Islek y Oguducu, 2019), que se refiere a predecir el tiempo de viaje futuro en base a datos históricos y en tiempo real (Elmi y Tan, 2020). Esta predicción es esencial en los Sistemas Avanzados de Información para Viajeros (ATIS, por su sigla en inglés) y en aplicaciones de navegación, pues su precisión impacta en la estimación de la hora de llegada, la planificación y optimización de rutas, la gestión del tráfico y la reducción de la congestión (Wang y cols., 2023; Chughtai, ul Haq, ul Islam, y Gani, 2022; Chughtai, Haq, y Muneeb, 2022). En el caso particular de los ómnibus, un TTP preciso es un indicador importante para proporcionar un servicio confiable y atractivo para los pasajeros (Moosavi, Aghaabbasi, Yuen, y Armaghani, 2023).

El objetivo principal de este proyecto es realizar un estudio sobre TTP, comenzando con una revisión de la literatura que permita tener un conocimiento más profundo sobre el problema en general, seguido de una fase de experimentación en la cual se pueda verificar (o refutar) las conclusiones obtenidas de la revisión. Para ello, se plantean los siguientes objetivos específicos: elaborar un Estado del Arte (EA) para entender la importancia del estudio del problema de TTP en la actualidad, identificar los factores que lo afectan, los métodos utilizados, y las características principales a considerar según el tipo de vehículo y el contexto, ya sea urbano o en autopista; llevar a cabo experimentos que permitan verificar las conclusiones obtenidas en el EA, evaluando la calidad de los datos disponibles para la experimentación mediante un análisis exploratorio de datos (EDA, por su sigla en inglés) y aplicando técnicas de depuración para mejorar su utilidad; implementar y entrenar tres modelos seleccionados del Estado del Arte: XGBoost, Bi-GRU y ARIMA, este último como modelo de referencia (*baseline*); examinar cómo la cantidad de datos afecta la precisión de los modelos y determinar si algunos métodos requieren más datos que otros para alcanzar un buen desempeño; y comparar los resultados obtenidos con las conclusiones reportadas en la literatura, validando o contrastando las observaciones previas. Cabe aclarar que el problema del TTP es muy amplio y puede aplicarse a diversos contextos, pero el enfoque de este proyecto está centrado en el ámbito vehicular, es decir, predecir el tiempo de viaje en las vías de circulación, incluyendo carreteras, calles y autopistas, tanto para el transporte público como privado.

Antes de realizar los experimentos, y en base a las conclusiones obtenidas de la fase del EA, se establecen las siguientes hipótesis: XGBoost proporciona buenos resultados de manera consistente, debido a su capacidad para manejar grandes volúmenes de datos estructurados y su eficiencia computacional. Se espera que este modelo sea robusto frente a diferentes tipos de características y escenarios, permitiendo realizar predicciones precisas de tiempos de viaje en entornos complejos y dinámicos; Bi-GRU tiene un desempeño similar o superior a XGBoost, especialmente cuando se dispone de grandes volúmenes de datos históricos. Se anticipa que este modelo necesita una mayor cantidad de datos para lograr un buen desempeño, dado que las redes neuronales recurrentes dependen en gran medida de la calidad y cantidad de datos secuenciales. Además, se espera que Bi-GRU tenga un costo computacional más alto en comparación con XGBoost, lo que podría influir en su aplicabilidad en entornos con recursos limitados; ARIMA, al ser un modelo tradicional de series temporales, sirve como un modelo base adecuado para comparar los resultados de los otros métodos. Se espera que ARIMA tenga limitaciones al capturar patrones no lineales y al adaptarse a las fluctuaciones complejas en los tiempos de viaje, lo que podría reducir su desempeño en comparación con los modelos más avanzados como XGBoost y Bi-GRU. Además, ARIMA es sensible a la estacionariedad de los datos, lo que puede afectar su precisión si los datos no cumplen con este requisito; el Análisis Exploratorio de Datos (EDA) es fundamental para evaluar la calidad y estructura de los datos, permitiendo identificar patrones, tendencias y anomalías que pueden influir en el desempeño de los modelos. Al proporcionar

una comprensión profunda de los datos, el EDA facilita una mejor preparación, lo que puede traducirse en predicciones más precisas y robustas.

El resto del documento se organiza de la siguiente manera: en el Capítulo 2, se presenta un resumen del EA, el cual se adjunta en el Anexo A. Se abordan los principales aspectos que deben considerarse para el problema de TTP en el ámbito vehicular, así como los métodos más utilizados en la actualidad. En el Capítulo 3, se describe el análisis, la limpieza y la preparación del *dataset* utilizado en la fase de experimentación. Se detallan los procesos de tratamiento de los datos para garantizar su calidad y relevancia antes de ser empleados en los modelos. En el Capítulo 4, se explican los fundamentos teóricos de los métodos seleccionados para los experimentos, así como las razones por las cuales fueron escogidos. En el Capítulo 5, se describen los aspectos técnicos relacionados con la solución propuesta, abarcando la arquitectura propuesta para la implementación, el hardware y software utilizado para llevar a cabo los experimentos, detalles específicos de los modelos implementados y una breve descripción de la metodología de gestión utilizada en la fase de desarrollo. En el Capítulo 6, se presentan los conceptos teóricos de las métricas utilizadas para evaluar el desempeño de los modelos y comparar los resultados obtenidos. Además, se exponen los resultados de los experimentos realizados junto con un análisis comparativo. Finalmente, en el Capítulo 7, se exponen las conclusiones generales de este trabajo y se discuten las posibles líneas de investigación futura que podrían derivarse de los resultados obtenidos. Además de los capítulos, se incluyen una serie de anexos que complementan la información presentada en el documento.

Capítulo 2

Revisión de la Literatura

El presente capítulo contiene un breve resumen del Estado del Arte, adjunto en el Anexo A. Aquí se presentan los principales aspectos a tener en cuenta en el problema de TTP en el ámbito vehicular, así como los métodos más utilizados. Como se menciona en la introducción, debido a la creciente urbanización, se presenta el desafío de abordar la congestión, problema crítico en muchas ciudades del mundo. Resolver este problema es clave, debido a que el transporte es vital para el desarrollo urbano, ya que facilita el acceso a servicios, empleos, educación, y contribuye al crecimiento económico.

2.1. Revisión Sistemática

Para llevar a cabo la revisión de la literatura, se procede a realizar una Revisión Sistemática (RS), metodología que permite abordar una pregunta de investigación, de una manera confiable y rigurosa (Kitchenham, 2004). Para obtener y seleccionar los artículos a investigar, se lleva a cabo un proceso conformado por las siguientes etapas: recuperación y selección de artículos, selección de criterios de exclusión y por último selección de enfoque para analizar los mismos.

En la etapa de recuperación y selección de artículos, se refina el *string* de búsqueda hasta encontrar uno que retorne resultados satisfactorios. Luego de varias iteraciones, el string final es:

```
ABS (‘internet of vehicle’ OR ‘arrival time prediction’ OR ‘intelligent
  transportation system’ OR ‘deep learning’ OR ‘machine learning’ OR
    ‘artificial intelligence’) AND TITLE (‘travel time prediction’)
```

Las búsquedas se realizaron en las colecciones Springer Link, Scopus, IEEE Xplore y ScienceDirect al 17 de setiembre de 2023, obteniendo un total de 215 artículos.

Luego se procede a depurar la lista. Primero se eliminan los duplicados, luego se quitan los que están en un idioma que no sea inglés (no había artículos en

español). Se hace una revisión de los títulos y se eliminan artículos que contienen los términos *train* y *vessel* ya que no tienen relación con el ámbito vehicular. Por último se ordenan por año y se procede a seleccionar los últimos 4 años recuperados, 2019 a 2023. Esto arrojó un total de 76 artículos, de los cuales 20 no se pudieron recuperar por cuestiones de acceso. Esto deja un total de 56 artículos para analizar. La Tabla 2.1 y gráfica 2.1 detallan la cantidad de artículos recuperados por año y colección. De estos 56 artículos, luego de leer el *abstract* se determinó que tres de ellos no guardan relación con nuestro problema y uno de ellos solo es una corrección a otro de los artículos donde indica que el mismo pasa a ser *open access*. Esto reduce a 52 la lista de artículos finales.

Colección	2019	2020	2021	2022	2023	Total
Springer Link	1	4	1	3	4	13
Scopus	5	1	5	12	3	26
IEEE Xplore X	3	1	3	3	-	10
Science Direct	2	1	1	2	1	7
Total	11	7	10	20	8	56

Tabla 2.1: Desglose de los artículos recuperados para analizar.

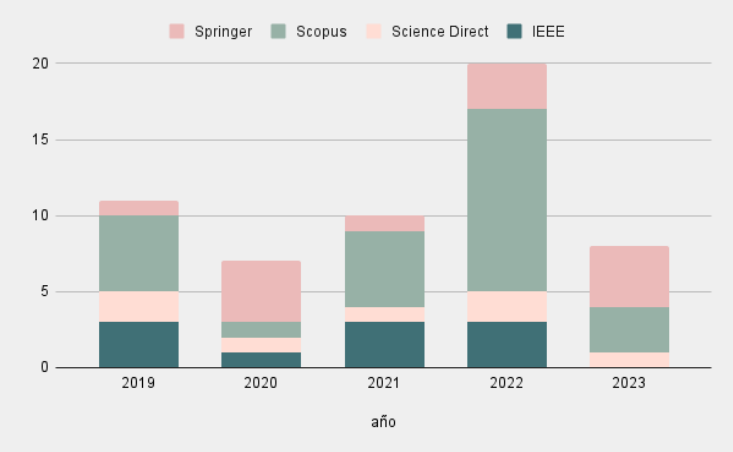

Figura 2.1: Artículos recuperados por año y colección.

Por último se determina el enfoque de análisis de los artículos. Debido a la complejidad del problema, se aborda el análisis desde dos perspectivas. Por un lado se analiza los elementos que participan en el proceso de predicción (factores, datos, métodos, ventana de predicción). Por otro lado se analizan los aspectos transversales al proceso de predicción, como lo son el contexto, el tipo de vehículo y el enfoque a abordar para resolver el problema. A continuación se

presentan cada uno de estos aspectos.

2.2. Factores que afectan el tiempo de viaje

Según Elmi y Tan (2020), uno de los elementos a considerar en el proceso de predicción de tiempos de viaje son los factores. Estos afectan a la predicción y pueden ser muy diversos. Entre ellos se encuentran la congestión, las condiciones climáticas, factores externos, como por ejemplo eventos especiales o manifestaciones que generan un flujo atípico de personas y pueden afectar la fluidez del tránsito.

Otro factor a considerar es la estructura de la red vehicular así como las condiciones del vehículo y hábitos del conductor (Shen, Jin, Hua, y Huang, 2022; K. Zhang, Lai, Jiang, y Yang, 2020).

Según Wang y cols. (2023), los factores se pueden clasificar en:

1. Dependencia Temporal: refiere a factores relacionados con la dimensión tiempo, como momento del día, hora o día de la semana. El tráfico varía significativamente entre días de semana y fin de semana, así como en horarios pico.

2. Dependencia Espacial: refiere a las características espaciales de la carretera como límites de velocidad, flujo de tráfico y reglas de tránsito. La predicción del tiempo de viaje para un segmento no solo está afectado por su propio estado, sino por los segmentos que conectan a él.

3. Dependencia Exógenas: refiere a factores externos, como el clima, conducta del conductor, accidentes de tráfico. Factores que son externos al sistema de transporte pero que tienen influencia directa en las condiciones de tráfico y el tiempo de viaje.

A lo largo de la literatura, se puede observar que la cantidad y tipo de factores utilizados es muy diversa. Algunos autores, como Islek y Oguducu (2019) solo utilizan velocidad promedio, fecha y hora en que se recibieron los datos del enlace, y datos para identificar el segmento sobre el que se está realizando la predicción.

En contraposición, K. Zhang y cols. (2020) resaltar la relevancia de factores externos como el clima, las condiciones del vehículo y los hábitos del conductor. Llevan a cabo un estudio enfocado en ómnibus, considerando una amplia variedad de características, entre ellas la condición climática, la identificación del conductor, vehículo y del intervalo de tiempo, así como la cantidad de personas que suben o bajan en cada parada del bus.

2.3. Recopilación y procesamiento de datos

Otro elemento fundamental en el proceso de predicción son los datos a utilizar. Por ese motivo, en esta sección se presenta todo el proceso por el que pasan los datos antes de estar listos para ser utilizados en el modelo de predicción.

Como se menciona en la Sección 2.2, las características utilizadas en la predicción pueden ser muy diversas y variadas. Por lo tanto, los dispositivos utilizados para la recolección de estos datos también varían.

En la RS el dispositivo utilizado que aparece con mayor frecuencia es el GPS. Este dispositivo aporta la información más elemental para la predicción: **id** para poder identificar la unidad, **latitud** y **longitud** de la ubicación por la que pasa el vehículo y **marca de tiempo** de cuando se obtiene el registro (Kumar, Jairam, Arkatkar, y Vanajakshi, 2019; Kumar, Mothukuri, y Vanajakshi, 2021; Pérez-González, Mora-Vargas, Piña-Barcenas, y Cedillo-Campos, 2023; Fan, Li, Lv, y Zhao, 2021; Sihag, Parida, y Kumar, 2022; Jakteerangkool y Muangsin, 2020; Ghosal, 2019).

Si bien los dispositivos GPS se han vuelto populares, existen áreas a las que no se puede acceder a esos datos por cuestiones de políticas y penetración de mercado, motivo por el cual dispositivos como los detectores también son bastante utilizados (Y. Li, Zhang, Ding, Zhou, y Xu, 2022).

Para el caso particular de los ómnibus, para conocer hora de salida y llegada, así como cantidad de pasajeros que suben y bajan en cada parada se puede utilizar información de sistemas como AVL (Sistema de Localización Automática de Vehículos), AFC (Sistema Automatizado de Recaudo) y APC (Sistema de conteo automático de pasajeros) (Moosavi y cols., 2023).

Otro factor que aparece mucho es el clima. Para obtener estos datos es necesario acceder a reportes climáticos. En el caso de Deb, Khan, Tanvir Hasan, Khan, y Alam (2019) los datos son obtenidos de *Wunderground*, servicio que proporciona información meteorológica actual e histórica (*WU Weather Underground*, 2025).

Luego de relevar los datos es necesario procesarlos. Este procesamiento puede incluir tareas de complementar la base de datos con más información, remover datos atípicos, incorrectos o irrelevantes, completar datos faltantes o llevar a cabo tareas de filtrado para aquellos casos en que la base de datos tenga datos que no son de interés para el estudio. No todos los experimentos llevan a cabo las mismas tareas, esto depende de la calidad y cantidad de datos disponibles, así como del tipo de datos que se maneje.

En la etapa de eliminación de valores atípicos, se eliminan valores extremos, lo cual depende mucho de las características del problema (Bharathi, Sopeña, Clarke, y Ghosh, 2023). Para eliminar estos datos se pueden utilizar diferentes mecanismos, por ejemplo el Rango Intercuartílico (IQR, por su sigla en inglés)[1] (Pérez-González y cols., 2023). Por más detalles sobre IQR ver Anexo E.

Luego de eliminar los valores atípicos, se puede analizar la cantidad de datos disponibles. En algunos casos, esta cantidad puede ser insuficiente debido a la naturaleza del problema, como en el caso de Ghosal (2019), donde los datos disponibles solo cubren el 3 % de los enlaces vehiculares; o puede ser insuficiente

[1] IQR: El rango intercuartílico es una técnica estadística utilizada para identificar valores atípicos en un conjunto de datos. Los valores que están por encima del límite superior o por debajo del límite inferior se consideran atípicos y pueden ser eliminados o tratados de manera especial en el análisis de datos.

debido a fallas en el mecanismo de recolección (Chang, Li, Fu, Liu, y Yang, 2019). En modelos orientados a datos, donde se necesita gran volumen de información, puede ser necesario recurrir a técnicas como la interpolación (proceso de estimar valores faltantes basándose en datos conocidos) para completar datos faltantes (Kumar y cols., 2019).

Por último, se debe decidir cómo representar los datos que se van a utilizar en el modelo. Los autores Wang y cols. (2023) señalan que los datos pueden ser categóricos[2] o discretos[3]. Además, mencionan que en los modelos de DL se utiliza ampliamente el método *One-Hot Encoding* para manejar variables categóricas, convirtiendo las características discretas en vectores binarios. Sin embargo, este método no contempla los valores de las características, lo que limita su capacidad de capturar la similitud entre ellas. Asimismo, la codificación *One-Hot* puede llegar a ser demasiado dispersa (con la mayoría de valores en cero) para que los modelos de DL puedan manejarla. Para superar esto, se propone usar el *embedding* de entidad, método que convierte las variables categóricas en vectores numéricos para que los modelos de aprendizaje automático puedan representar y procesar de manera más efectiva la información.

2.4. Horizontes de Predicción

El horizonte de predicción refiere al período de tiempo en el que se llevará a cabo la predicción y se divide principalmente en corto y largo plazo. No existe un consenso en la literatura sobre la duración exacta de estos conceptos.

El corto plazo se define como la predicción para dentro de unos pocos segundos/minutos, hasta 30 minutos o una hora (Chughtai, ul Haq, y cols., 2022; Qiu y Fan, 2021; Y. Huang, Dai, y Tseng, 2022; H. Huang, Pouls, Meyer, y Pauly, 2020) o incluso unas pocas horas en el futuro (Tang y cols., 2019).

Por otro lado, el largo plazo implica un horizonte de predicción mayor que va desde más de una hora (Y. Huang y cols., 2022), un día (Qiu y Fan, 2021; Chughtai, ul Haq, y cols., 2022), hasta varios días (H. Huang y cols., 2020), o al menos una semana (Islek y Oguducu, 2019).

Ambos horizontes se utilizan para la planificación de viajes, logística y gestión del tráfico, y ambos pueden verse afectados por las condiciones del tráfico y factores diversos (Qiu y Fan, 2021; Tang y cols., 2019; H. Huang y cols., 2020; Sihag y cols., 2022). La diferencia principal radica en el horizonte de tiempo y en la complejidad. Respecto a la complejidad, la predicción a largo plazo resulta más desafiante y complicada, ya que es necesario que los modelos cuenten con componentes que se encarguen de la extracción de patrones históricos periódicos (Y. Huang y cols., 2022).

[2]Categórico refiere a que se puede clasificar en categorías definidas, como el día de la semana o el tipo de vehículo (auto, moto, camión).

[3]Los datos discretos pueden asumir cualquier valor sin seguir un orden específico y no están limitados a una lista finita o categoría, como el número de pasajeros o el identificador del conductor.

Los autores Y. Huang y cols. (2022) mencionan que los modelos utilizados para la predicción en el corto plazo logran tener una performance ineficiente cuando enfrentan problemas de largo plazo. En contraposición a esto, Islek y Oguducu (2019) llevan a cabo un experimento donde utilizan el mismo método, LSTM, para ambos horizontes, obteniendo resultados razonables.

Es importante destacar que el tema horizonte de predicción no es abordado (al menos no de manera explícita) en todos los artículos analizados. Sin embargo, Y. Huang y cols. (2022) y Mendes-Moreira y Baratchi (2020) aseguran que la mayoría de los estudios de TTP se centran en la predicción a corto plazo, y Sihag y cols. (2022) añaden que la predicción a largo plazo es crucial para la planificación del transporte, la reducción de costos, entre otros.

2.5. Modelos de Predicción de Tiempos de Viaje

Luego de haber revisado los factores que afectan a la predicción, el proceso por el que pasan los datos antes de ser utilizados por el modelo a entrenar, las particularidades a tener en cuenta en la predicción según el tipo de vehículo, y los factores a tener en cuenta según si se trata de una predicción en entorno urbano o autopista, el siguiente paso es analizar cuáles son los métodos utilizados para la predicción.

Como se presenta en el EA, los métodos utilizados son muy diversos y existe una gran variedad de métodos explorados. No existe un criterio único para clasificar estos métodos. Para poder organizar la presentación, se toma como punto de partida la clasificación de Mori, Mendiburu, Álvarez, y Lozano (2015) presentado en la Figura 2.2. Esta clasificación agrupa los métodos en 3 grandes clases: **modelos ingenuos**, **modelos basados en la teoría del tráfico**, y **modelos basados en datos**.

2.5.1. Modelos ingenuos

Los modelos ingenuos son enfoques simples y rápidos para predecir el tiempo de viaje, que no requieren entrenamiento ni estimación de parámetros, pero presentan suposiciones restrictivas que limitan su confiabilidad (Moosavi y cols., 2023). Se clasifican en predictores **instantáneos**, **históricos** e **híbridos** (Mori y cols., 2015).

Los **predictores instantáneos** asumen condiciones de tráfico constantes, siendo efectivos solo en horizontes de predicción muy pequeños (Alkilane, Alfateh, y Yanming, 2023).

Los **predictores históricos** asumen que los tiempos de viaje son similares a los recopilados en un intervalo de tiempo similar en el pasado bajo condiciones de tráfico similares. Para predecir datos TT futuro usan el tiempo de viaje promedio histórico (Ou, 2022). Por ejemplo, para predecir el tiempo de viaje de un viaje que ocurrió un lunes a las 10 a.m se toman datos de lunes pasados en el entorno de ese horario y se realiza un promedio. La complejidad del método va

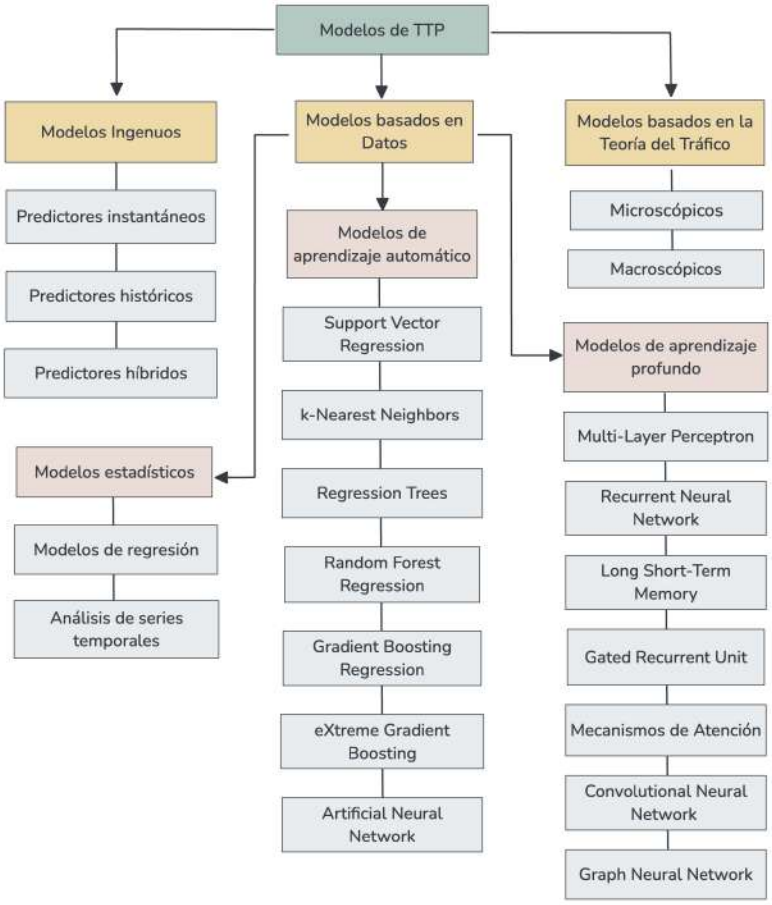

Figura 2.2: Taxonomía de los modelos de TTP. Fuente: elaboración propia.

a depender de qué patrones de tiempo se consideren para buscar las similitudes (C. H. Wu, Ho, y Lee, 2004).

Debido a su simplicidad, estos modelos se utilizan como línea base de comparación (Moosavi y cols., 2023).

2.5.2. Modelos basados en la teoría del tráfico

Estos modelos predicen basándose en modelos teóricos. Su objetivo es poder simular condiciones de tráfico futuras y en base a éstas poder derivar los tiempos de viaje (Mori y cols., 2015). Estos métodos aplican relaciones entre variables de tráfico obtenidas de la teoría del tráfico y se fundamentan en los principios de conservación y propagación del flujo (Schwinger, Frohnhofen, Wernz, Braun, y Jarke, 2021). Según Kumar, Vanajakshi, y Subramanian (2017) se clasifican en: modelos **microscópicos** y **macroscópicos**.

Los modelos **microscópicos** utilizan matrices de origen-destino (OD) y volúmenes de giro para predecir tiempos de viaje. Las matrices OD representan el flujo de tráfico desde cada origen posible hasta cada destino. Los volúmenes de giro indican el porcentaje del tráfico que gira en cada dirección en una intersección dada. A partir de estos, se simulan las trayectorias de los vehículos individuales para intervalos de tiempo futuros, teniendo en cuenta factores como las interacciones entre vehículos, el comportamiento del conductor, y el cambio de carril.

Los modelos **macroscópicos** usan ecuaciones que relacionan variables de tráfico agregadas como densidad, flujo y velocidad media. Según Mori y cols. (2015), no suelen proporcionar tiempos de viaje directamente y presentan limitaciones en entornos urbanos.

La principal ventaja de estos enfoques es su capacidad para describir el estado del tráfico en la red utilizando pocos datos (Kumar y cols., 2017). Sin embargo, su implementación requiere conocimientos avanzados en teoría del tráfico, matemáticas y programación. Además, la simulación debe replicar fielmente las condiciones reales para garantizar predicciones precisas (Moosavi y cols., 2023).

2.5.3. Modelos basados en datos

Con los recientes desarrollos tecnológicos en detección automatizada del tráfico, comunicación en tiempo real, e instalaciones de cómputo de alto rendimiento, gran cantidad de datos está disponible, y esto hace que las técnicas basadas en datos sean una buena opción para la predicción del tráfico.

Según Thakkar, Sharma, Advani, Arkatkar, y Bhaskar (2021) los enfoques basados en datos utilizan la base de datos histórica y relacionan el estado actual del tráfico con los patrones históricos más similares en cuanto a los parámetros de tráfico (volumen de tráfico, flujo, velocidad, otros).

De acuerdo a Nithishwer, Kumar, y Vanajakshi (2022) la principal ventaja de estos modelos es que no requieren ningún conocimiento de la teoría del tráfico y dinámica de la red.

Según las técnicas de implementación de estos modelos, se pueden agrupar en tres categorías: **modelos estadísticos**, **modelos de aprendizaje automático**, y **modelos de aprendizaje profundo** (Fang y cols., 2022).

Modelos Estadísticos

La idea principal de la predicción del tráfico en los modelos estadísticos se basa en el hecho de que los comportamientos del tráfico poseen propiedades parcialmente deterministas y parcialmente caóticas. Los resultados de la predicción se pueden obtener reconstruyendo el movimiento del tráfico determinista y prediciendo los comportamientos aleatorios causados por factores no anticipados (C. H. Wu y cols., 2004).

Los métodos **estadísticos** se pueden clasificar en: **modelos de regresión**, y **modelos de series temporales** (Bharathi y cols., 2023).

Los **modelos de regresión** asumen que el tiempo de viaje es el resultado de una función matemática con diferentes variables como ser condiciones climáticas, circunstancias del tráfico, etc. La función se utiliza para describir la relación entre la variable dependiente (el tiempo de viaje) y un conjunto de variables independientes entre sí (Ou, 2022). La regresión puede ser lineal y no lineal. La regresión lineal traza una línea de mejor ajuste en un diagrama de dispersión, definida por una ecuación lineal que relaciona la variable dependiente con una o más variables independientes mediante mínimos cuadrados (Deb y cols., 2019). Algunos de los modelos lineales son *Linear Regression* (**LR**), *Ridge Regression* (**RR**) y *Least Absolute Shrinkage and Selection Operator Regression* (**LASSO**). Los modelos de regresión no lineal se presentan dentro de la categoría de aprendizaje automático.

Ou (2022) afirma que la principal limitación de los métodos de regresión en el terreno de la predicción de tiempos de viaje es que las variables en los sistemas de transporte suelen estar inter-correlacionadas en lugar de ser completamente independientes.

El análisis de **series temporales** se utiliza para predecir los tiempos de viaje, dado que estos varían con el tiempo. Una serie temporal es una secuencia de datos ordenados temporalmente (Fang y cols., 2022). El análisis clásico se basa en la suposición de que los valores futuros dependen de los históricos y de ruidos aleatorios (Wang y cols., 2023). Los modelos más comunes para la predicción son *Autoregressive* (**AR**), *Moving Average* (**MA**), *Autoregressive Moving Average* (**ARMA**), *Autoregressive Integrated Moving Average* (**ARIMA**), *Seasonal Autoregressive Integrated Moving Average* (**SARIMA**) y el filtro de Kalman (Ran, Shan, Fang, y Lin, 2019). La precisión de estos modelos depende de la adecuación de las funciones matemáticas para modelar los datos históricos y la similitud entre los patrones históricos y en tiempo real Ou (2022).

Sin embargo, estos modelos no son adecuados para analizar las características no lineales de los datos en entornos de tráfico complejos (Fang y cols., 2022; Xu y Liu, 2021), lo que ha impulsado el interés en los métodos de aprendizaje automático.

Modelos de aprendizaje automático

En aplicaciones con datos no lineales, los modelos lineales presentan bajo rendimiento, por lo que los métodos no lineales son más efectivos en estas situaciones (Ashwini, Sumathi, y Sudhira, 2022). Dado que el tráfico presenta una relación no lineal espacio-temporal, los modelos de aprendizaje automático (ML, por su sigla en inglés) han demostrado ser más efectivos en su predicción.

El aprendizaje automático se basa en la idea de que los sistemas pueden aprender de los datos y mejorar sus decisiones a través de la experiencia. Según Ou (2022), la construcción de un modelo de aprendizaje automático incluye cuatro fases: preparar el conjunto de datos de entrenamiento (CDE), elegir un algoritmo, entrenar el modelo y mejorar su desempeño.

A continuación se presenta una breve introducción a algunos de los modelos de ML aplicados en la predicción de tiempos de viaje. Por más información, se

puede consultar el EA.

Support Vector Regression (SVR)

SVR tiene como objetivo encontrar el mejor hiperplano para predecir con precisión la distribución de los datos (M. Y. Chen, Chiang, y Yang, 2022). Según Awad y Khanna (2015), una de las ventajas clave de SVR es que su complejidad computacional no depende de la dimensionalidad del espacio de entrada, además de destacar por su excelente capacidad de generalización y alta precisión predictiva.

Árboles de Regresión

Los Árboles de Regresión son un tipo de modelo no lineal para predecir una variable continua mediante reglas simples extraídas de los datos (Ashwini y cols., 2022). Su estructura se basa en divisiones secuenciales, donde los nodos hoja son las predicciones finales. A diferencia de otros métodos de caja negra, éste es interpretable y puede capturar relaciones no lineales (Qiu y Fan, 2021). Si bien su desempeño en entrenamiento es bueno, un exceso de divisiones (altura del árbol) puede generar sobreajuste (N. Li y cols., 2023).

Random Forest (RF)

Random Forest consiste en generar múltiples árboles de regresión para mejorar la precisión de las predicciones al combinar los resultados de cada uno. Mientras que un solo árbol presenta inestabilidad en los resultados H. Huang y cols. (2020), RF reduce este problema al construir múltiples árboles a partir de muestras aleatorias de los datos de entrenamiento, lo que mejora la estabilidad y precisión Ashwini y cols. (2022). Entre sus ventajas se encuentra el buen desempeño con grandes volúmenes de datos (Deb y cols., 2019). Sin embargo, cuando los datos contienen errores o variaciones impredecibles, el proceso de entrenamiento puede volverse más lento (N. Li y cols., 2023).

Gradient Boosting Regression (GBR)

Este método se basa en la idea del impulso (*boosting*), donde un modelo débil (árbol de decisión poco profundo) puede mejorarse progresivamente para lograr una mejor precisión. En este enfoque, los árboles de regresión se entrenan en serie, de manera que cada árbol sucesivo aprende de los errores cometidos por los anteriores (Ashwini y cols., 2022). En GBR cada árbol se especializa en diferentes patrones. Esto permite mejorar la precisión del modelo sin necesidad de utilizar árboles muy profundos (Moosavi y cols., 2023). El proceso continúa hasta que el error se reduce por debajo de un umbral cercano a cero (M. Y. Chen y cols., 2022). Una vez alcanzado este criterio de parada, las predicciones de todos los árboles se combinan para obtener una estimación final más precisa.

eXtreme Gradient Boosting (XGBoost)

XGBoost es un modelo avanzado que utiliza un enfoque de impulso, similar a GBR, pero con mayor precisión. Este método pondera los aprendices débiles y ajusta las muestras con mayor tasa de error para mejorar los modelos (Qiu y Fan, 2021). Según N. Li y cols. (2023), XGBoost tiene tres ventajas sobre GBR: mejora la precisión, previene el sobreajuste y aumenta la velocidad de cálculo. XGBoost ha demostrado ser robusto frente a los valores atípicos y presenta un excelente rendimiento en series temporales, lo que lo hace adecuado para la predicción de tiempos de viaje. Sin embargo, su uso se limita a datos estructurados en tablas (Shen y cols., 2022).

En resumen, los métodos de aprendizaje automático mejoran la precisión en la predicción de tiempos de viaje en comparación con los métodos estadísticos, pero enfrentan dificultades en la modelización de eventos imprevistos y en la gestión de grandes volúmenes de datos (Wang y cols., 2023).

Modelos de aprendizaje profundo

El aprendizaje profundo (DL) ha permitido que las redes neuronales profundas procesen datos de alta dimensionalidad a gran escala (Nithishwer y cols., 2022). Estas arquitecturas aprenden de manera jerárquica, acumulando conocimiento mientras la información se propaga a través de sus capas. Diversos enfoques se han propuesto para la predicción del tiempo de viaje, mostrando ser ampliamente superiores a los métodos estadísticos y de aprendizaje automático (Nithishwer y cols., 2022). Sin embargo, estos modelos requieren un alto costo computacional (C. H. Chen, Hwang, y Kung, 2019).

A continuación se presentan brevemente algunos de estos métodos. Por una descripción más detallada, consultar el documento Estado del Arte.

Multi-Layer Perceptron (MLP)

El MLP es una red neuronal sencilla ampliamente utilizada para la predicción de tiempos de viaje. Se compone de varias capas de neuronas interconectadas, donde las capas ocultas y el número de neuronas dependen del tipo de problema. Aunque es útil para predicciones a corto plazo, tiene limitaciones en la captura de correlaciones a largo plazo entre los distintos enlaces de las rutas y la memoria del proceso de viaje (Wang y cols., 2023).

Recurrent Neural Network (RNN)

Las RNN, a diferencia del MLP, tienen conexiones retroalimentadas, lo que les permite retener y utilizar información de estados anteriores. Esta capacidad les permite capturar dependencias temporales en datos secuenciales, "memorizando" el historial y considerando las relaciones entre salidas pasadas y entradas actuales (Fang y cols., 2022). Si bien son eficaces para la predicción de series

temporales, al trabajar con secuencias largas, enfrentan desafíos como el desvanecimiento del gradiente (C. Y. Chen, Sun, Chang, y Lin, 2023).

Long Short-Term Memory (LSTM)

LSTM mejora las RNN al usar compuertas para controlar el flujo de información, permitiendo retener aquella relevante y filtrar la que no lo es. Además mitiga los problemas de desvanecimiento y explosión del gradiente (K. Zhang y cols., 2020). LSTM ha mostrado buenos resultados en la predicción de tiempo de viaje y condiciones de tráfico (Fang y cols., 2022).

Gated Recurrent Unit (GRU)

GRU es una versión simplificada del LSTM que reduce la complejidad mediante la fusión de puertas. Como resultado de esta simplificación, se mejora la velocidad de entrenamiento (Jakteerangkool y Muangsin, 2020). Este método ha demostrado ser eficaz en la predicción de tiempos de viaje, capturando correctamente las dependencias temporales.

Mecanismos de Atención

Los mecanismos de atención han sido introducidos recientemente al problema de TTP, debido a su éxito en otras tareas como la clasificación de imágenes y la traducción automática. Estos mecanismos se centran en las diferencias clave entre las características de entrada, aprendiendo solo el contexto relevante. Esto permite al modelo enfocarse en partes específicas, dándoles más peso para tomar decisiones más precisas. Han demostrado ser efectivos y flexibles en diversas aplicaciones de transporte (Chughtai, Haq, Shafiq, y Muneeb, 2022).

Convolutional Neural Network (CNN)

Las CNN han sido adaptadas para modelar redes de tráfico, transformando estas redes en imágenes para capturar características espaciales (Fang y cols., 2022). Diversos estudios han demostrado que los modelos CNN pueden mejorar la precisión predictiva en el análisis del tráfico. Sin embargo, la estructura no euclidiana de la red vehicular impide que la cuadrícula bidimensional capture adecuadamente las correlaciones topológicas complejas (Xu y Liu, 2021).

Graph Neural Network (GNN)

Por último, las redes neuronales de grafos han ganado popularidad en la predicción de tiempos de viaje debido a que la red de tráfico tiene una estructura natural de grafo. Un ejemplo de esto son las *Graph Convolutional Networks* (GCN), que se han aplicado con éxito en predicciones de tráfico. Aunque los modelos de GNN han mostrado un rendimiento destacado en tareas como la predicción de tiempos de viaje y el flujo de tráfico (Chughtai, Haq, Shafiq, y

Muneeb, 2022), su uso en modelos grandes puede requerir altos recursos computacionales, lo cual es un desafío (Fang y cols., 2022).

Concluyendo, existen diversos enfoques para predecir el tiempo de viaje, desde modelos simples hasta redes neuronales profundas. Los modelos ingenuos son rápidos pero poco precisos. Los modelos basados en la teoría del tráfico simulan condiciones futuras, ofreciendo una visión completa con pocos datos, aunque requieren conocimientos avanzados.

Los modelos basados en datos incluyen:

1. Estadísticos, que captan correlaciones lineales de forma sencilla.
2. Aprendizaje automático, que mejora la precisión e interpreta relaciones no lineales, pero puede verse afectado por grandes volúmenes de datos.
3. Aprendizaje profundo, que maneja relaciones complejas y espaciales mediante redes neuronales (RNN, CNN, GNN), aunque con altos costos computacionales y baja interpretabilidad.

Cada enfoque tiene ventajas y limitaciones, y la elección del modelo depende de la disponibilidad de datos, la complejidad del problema y los recursos computacionales.

2.6. Contextos de Predicción

Así como los factores utilizados para la predicción y la cantidad y calidad de datos son importantes para la predicción, también lo son el tipo de vehículo y el entorno en el que se intenta predecir. No es lo mismo una predicción para un bus, un auto particular o un taxi, como no es lo mismo la predicción en un entorno urbano que en autopista. A continuación se presentan algunas características particulares al problema según el tipo de vehículo o entorno.

2.6.1. Tipo de Vehículo

Para el caso de vehículo particular, la principal característica es que es un problema donde solo se conoce el par origen-destino, no sigue ninguna ruta en particular ni horarios predecibles, sino que están influenciados por las preferencias individuales de los conductores. Los autores Chughtai, Haq, Shafiq, y Muneeb (2022) proponen una red neuronal GRU para la predicción a corto plazo, permitiendo que la GRU aprenda el contexto relevante en las secuencias históricas de tiempo de viaje y logre de esta manera una mejor precisión en la predicción.

El problema del TTP para taxis, aunque similar al de vehículos particulares, difiere en la conducta del conductor, quien muestra patrones de conducta diferentes debido a ser el taxi un medio de empleo (J. Wu, 2014). Deb y cols. (2019) en su estudio con datos de taxis modelo Sedan, proponen predecir el nivel de congestión del tráfico mediante series temporales y aprendizaje automático. Por

su parte, Wang y cols. (2023) desarrollan un modelo de aprendizaje profundo que combina redes neuronales de grafos para capturar correlaciones espaciales, redes neuronales recurrentes para capturar información temporal, y un modelo específico para extraer datos exógenos. Finalmente, H. Huang y cols. (2020) aplican métodos basados en árboles para predecir tiempos de viaje, abordando pronósticos a corto y largo plazo, con mejores resultados en el primero.

Por último, tenemos el transporte público colectivo. Éste presenta particularidades que lo diferencian de otros vehículos, como el uso de rutas fijas y paradas, donde el tiempo de permanencia influye en el tiempo total de viaje. Esto hace que los métodos utilizados para predecir el tiempo de viaje de autos privados o taxis no tengan la misma precisión en el contexto del transporte público (K. Zhang y cols., 2020; Ashwini y cols., 2022). Según Moosavi y cols. (2023), no existe un modelo único de aprendizaje automático óptimo para predecir el tiempo de viaje en autobuses. En su estudio en Kuala Lumpur, Malasia, comparan modelos basados en árboles (GBR, RF y CHAID), concluyendo que GBR es preferible para rutas de alta frecuencia y CHAID para las de baja frecuencia. Por su parte, Mendes-Moreira y Baratchi (2020) proponen un modelo de reconciliación[4] en dos pasos: predicción de tiempos entre paradas consecutivas y reconciliación para el tiempo total de viaje. Similarmente, K. Zhang y cols. (2020) separan la predicción de tiempos de espera y en tránsito, aplicando modelos distintos y combinándolos para la predicción final.

2.6.2. Entornos

De la RS se desprenden dos entornos bien diferenciados: urbano y autopista. Estos entornos presentan características y condiciones bien diferentes, como velocidades permitidas, cantidad de intersecciones, señalizaciones, entre otros.

En entornos de **autopistas**, el tráfico tiende a ser fluido y predecible, ya que mientras no exista congestión, la velocidad de los vehículos va a tender al límite permitido. Esto hace que la predicción del tiempo de viaje en autopistas sea más estable y menos afectada por factores imprevistos en comparación con los entornos urbanos.

Por ejemplo, Xu y Liu (2021) proponen el modelo *Multi-Component Network* (MC-Net), que combina atención espacio-temporal, eventos de tráfico y fusión de múltiples fuentes, superando a otros métodos en precisión, especialmente en situaciones con eventos de tráfico no recurrentes. M. Y. Chen y cols. (2022) desarrollan un Sistema de Transporte Inteligente Colaborativo basado en 9 algoritmos, donde los vehículos reciben información en tiempo real para evitar de esta manera congestiones y accidentes. Qiu y Fan (2021) realizan una predicción a corto plazo (de 15 a 60 minutos) aplicando cuatro algoritmos diferentes: Árboles de Regresión, RF, XGBoost y LSTM, destacando RF en precisión y estabilidad. Bharathi y cols. (2023) emplean LSTM junto con un algoritmo de descomposición modal (para mejorar la calidad de los datos de entrada) y regresión cuantílica para predecir el tiempo de viaje en las autopistas de Dublín.

[4] La reconciliación combina predicciones en series temporales para obtener resultados óptimos mediante modelado conjunto.

Finalmente, C. Y. Chen y cols. (2023) desarrollan el método *Bi-Directional Isometric-Gated Recurrent Unit* (BDIGRU), que mejora la precisión predictiva, logrando predecir 30 minutos por adelantado.

Por otro lado, el entorno **urbano** presenta desafíos adicionales para la predicción de tiempos de viaje debido a factores como los ciclos de señalización en intersecciones conectadas, la congestión habitual y eventos como reparaciones de carreteras o mal tiempo, los cuales afectan significativamente el flujo de tráfico (Abdollahi, Khaleghi, y Yang, 2020; M. Y. Chen y cols., 2022). Según Wang y cols. (2023), la principal dificultad al momento de predecir radica en la dinámica de los sistemas de transporte urbano, destacando que la congestión alcanza su máximo al mediodía. Además, el tiempo de viaje de un segmento está influenciado no solo por su propio estado de tráfico, sino también por los segmentos conectados, por lo que es crucial considerar las interconexiones vehiculares.

En cuanto a los métodos empleados en este entorno, L. Zhang y cols. (2021) comparan modelos de aprendizaje automático y aprendizaje profundo, destacando que los primeros tienen dificultades para modelar el impacto de eventos imprevistos como accidentes o construcciones. Los métodos de aprendizaje profundo, por su parte, logran generar una "memoria" de patrones de tiempo de viaje, aunque no superan completamente a los métodos tradicionales en todos los indicadores de evaluación. Tang y cols. (2019) proponen un modelo basado en datos obtenidos en cada ciclo de señal de tráfico (como puede ser las señales de semáforos), mostrando su efectividad en comparación con modelos como k-NN. Por otro lado, Ashwini y cols. (2022) encuentran que los métodos no lineales, particularmente RF, ofrecen mejores resultados frente a los lineales.

2.7. Enfoques para Resolver la Predicción

Otro elemento transversal al proceso de predicción es el enfoque. Este refiere a la estrategia a utilizar para abordar el problema.

Según los autores Chughtai, Haq, Shafiq, y Muneeb (2022), los métodos existentes de TTP pueden clasificarse en dos categorías principales: **enfoques basados en la ruta** y **enfoques basados en datos**.

Los **enfoques basados en la ruta** consideran la predicción como la suma de los tiempos de los segmentos y pueden o no tener en cuenta los tiempos de transición entre estos. Esta categoría se subdivide a su vez en:

1. Enfoque basado en Segmento (*segment-based*): Realiza la predicción por cada uno de los segmentos y luego suma los tiempos para obtener el tiempo total del recorrido, ignorando la correlación entre los mismos. La desventaja de este enfoque es que acumula el error de predicción de cada segmento.

2. Enfoque basado en Subruta (*path-based*): Utiliza tanto el tiempo del segmento como los retrasos en las intersecciones.

Por otro lado, los **enfoques basados en datos** tienden a modelar el TTP de extremo a extremo, aprovechando las características espacio-temporales y aprendiendo correlaciones en los datos de tráfico. Esta categoría incluye:

1. Enfoque basado en la Trayectoria: Utiliza la red de carreteras y datos de trayectoria[5].

2. Enfoque basado en el Origen-Destino (OD): Considera solo los datos de ubicación de recogida y entrega y no incluye trayectorias intermedias.

Además, existe la posibilidad de emplear enfoques híbridos o combinados, donde se combinan diferentes modelos para mejorar la precisión de la predicción. Estos enfoques pueden ser:

1. Enfoque híbrido: combina modelos para crear uno nuevo, utilizado en toda la predicción.

2. Enfoque combinado: en la predicción de cada segmento se utilizan diferentes modelos y luego se realiza la suma.

La clasificación presentada no es un estándar y puede variar según los autores. Algunos autores presentan una clasificación un poco más simplificada y dividen los enfoques en basados en ruta y en OD (K. Zhang y cols., 2020), o basada en subruta, en segmento y en OD (Khaled, Elsir, y Shen, 2022).

[5]Los autores no aclaran cuáles serían estos datos, pero del contexto se infiere que es información del vehículo a lo largo de la ruta, como por ejemplo posición, velocidad, dirección, etc.

Capítulo 3

Conjunto de Datos

El presente capítulo describe el análisis, limpieza y preparación del conjunto de datos (al cual nos referiremos a partir de ahora como *dataset*) utilizado en la fase de experimentación. El mismo corresponde a la línea de ómnibus 117 de la compañía CUTCSA.

Esta línea es de carácter urbano y realiza viajes entre los barrios Ciudad Vieja y Punta Carretas en la ciudad de Montevideo, ida y vuelta. Los datos utilizados son datos abiertos proporcionados por la Gerencia de Tecnología en Ciudades Inteligentes de la Intendencia Municipal de Montevideo, y abarcan el período comprendido entre enero de 2021 y agosto de 2024.

3.1. Descripción del *Dataset*

El *dataset* contiene información detallada de todos los viajes realizados por los ómnibus de la línea 117 durante el período mencionado. La línea está conformada por diversas variantes, cada una de las cuales representa una ruta diferente, es decir, un par origen-destino distinto. A su vez, cada entrada del *dataset* corresponde al registro del paso del ómnibus por una parada específica, incluyendo detalles sobre el tiempo, ordinal de la parada, la variante, el identificador del recorrido y demás información complementaria. La Tabla 3.1 describe cada una de las variantes.

3.2. Análisis Exploratorio de Datos

En esta etapa, se realiza el análisis exploratorio de datos (EDA) con el fin de entender la información proporcionada por el *dataset*. Este análisis incluye la verificación de los tipos de datos de cada campo, rango de los mismos, la búsqueda de valores nulos y duplicados, y la comprobación de la integridad y formato de los datos. También se verifica que, dado un recorrido, la fecha y hora registrada para cada parada sea creciente a medida que el ómnibus avanza en su

Variante	Dirección Origen	Cantidad Paradas	Dirección Destino
635	Juncal y Sarandí	24	Bv Gral. Artigas y Parva Domus
2779	Ciudadela y Rincón	35	Bv Gral. Artigas y Parva Domus
2778	Bv. Gral. Artigas y Tabaré	21	Av. 18 de Julio y Convención
634	Bv. Gral. Artigas y Tabaré	22	Juncal y Sarandí

Tabla 3.1: Direcciones de origen y destino y cantidad de paradas por variante.

recorrido. Además, se grafican en un mapa las diferentes variantes para verificar si hay desvíos. Se pueden consultar los mapas en el Anexo D.

A partir de este análisis inicial, se identifica lo siguiente:

- Valores nulos: Se encuentran valores nulos en el campo que contiene la información sobre la fecha y hora a la cual pasa el bus por la parada, campo fecha_pasada_real.
- Duplicados: Solo se encuentra una instancia de duplicados en el *dataset*.
- Distintos formatos de fecha: Se identifica que los valores guardados en el campo fecha_pasada_real tienen más de un formato.
- Desplazamiento de paradas: Se detecta que, a partir de julio de 2021, la mayoría de las variantes tiene un traslado de una cuadra en todas sus paradas. Los gráficos de los mapas fueron claves en esta etapa, porque sin la ayuda visual, no hubiese sido posible identificar que existía este problema.
- Recorridos con fechas no crecientes: Se detecta que existen recorridos en los que los registros de tiempo de las paradas no siguen un orden creciente a medida que el ómnibus avanza en su trayecto.
- Dos destinos diferentes para variante 2778: Un destino en Av. 18 de Julio y Convención y el otro en Ciudadela y Rincón. En el mapa D.3 se puede observar el problema mencionado. El primer destino mencionado es la penúltima parada en el caso de los recorridos que terminan en Ciudadela y Rincón.

Además, se observa que el origen de los datos puede ser de tres tipos diferentes: “Aproximada por GPS”, “Detectada por GPS” y “Detectada Manualmente”. Al consultar a la Intendencia sobre el significado de estas categorías, explicaron que se refieren al nivel de precisión de los datos registrados.

Cada recorrido guarda puntos de GPS aproximadamente cada 15 segundos, y estos puntos se utilizan para calcular la fecha de pasada por cada parada. Si el

origen de los datos es "Aproximada por GPS", significa que la fecha de pasada se calcula con los puntos de GPS más cercanos a la parada, lo que ofrece la mayor precisión.

Cuando los datos son "Detectada por GPS", esto indica que el GPS no ha registrado correctamente las posiciones, por lo que se utiliza un método menos preciso para estimar la fecha de pasada, basándose en los puntos disponibles.

Finalmente, si ambos métodos anteriores fallan, se utiliza la "Detectada Manualmente", en la cual el conductor del ómnibus registra manualmente la llegada a la parada. Este método es el menos preciso, ya que el procedimiento es manual y puede no ser consistente. Además, a veces se registran varias paradas juntas al mismo tiempo.

3.3. Proceso de Limpieza de Datos

Con base en los problemas identificados durante el análisis exploratorio, se decide realizar un proceso de limpieza y depuración de los datos. Las acciones tomadas son las siguientes:

- Eliminar recorridos con valores nulos: Para simplificar el análisis y garantizar la calidad de los datos, se eliminan aquellos recorridos que contienen valores nulos en el campo fecha_pasada_real.
- Eliminar duplicados: Aunque solo se detecta un dato duplicado, se procede a su eliminación.
- Corregir formato de fechas: Se unifica el formato de fecha en el campo fecha_pasada_real.
- Eliminar datos previos a julio de 2021: Debido al desplazamiento de paradas mencionado arriba, se eliminan todos los datos anteriores a julio de 2021 para evitar trabajar con paradas que tienen más de una dirección asociada.
- Eliminar columnas irrelevantes: Se eliminan las columnas que no aportan valor al análisis, concentrándose solo en los datos esenciales.
- Eliminar trayectos con datos imprecisos: Se eliminan los trayectos cuyo origen de datos es distinto a "Detectada por GPS", dado que estos valores podrían no ser suficientemente precisos.
- Eliminar trayectos con fecha no creciente: Se eliminan los trayectos que no presentan un registro de tiempo creciente a medida que el ómnibus avanza.
- Eliminar *outliers*: Se eliminan valores atípicos, utilizando el Rango Intercuartílico.
- Eliminar datos que tienen la fecha cargada manualmente. Se toman los recorridos que solo tienen datos GPS.

- Para la variante 2778, se decide tomar como único destino a Av. 18 de Julio y Convención. De esta manera se garantiza tener la misma cantidad de datos para todas las paradas.

La Tabla 3.2 detalla las trayectorias finales disponibles para los experimentos por cada una de las variantes [1].

Variante	**Trayectorias Iniciales**	**Trayectorias Finales**
634	55648	25783
635	52678	29787
2778	8248	5118
2779	8298	4749

Tabla 3.2: Trayectorias finales.

3.4. Modificación y Creación de Características

Luego de la limpieza, se procede a realizar algunas modificaciones y adiciones en el *dataset* para mejorar su calidad y utilidad para el análisis posterior. Las acciones realizadas son:

- Renombrar campos: Se modifican los nombres de algunas columnas para hacerlos más representativos y comprensibles.

- Crear nuevas características: A partir del campo `fecha_pasada_real`, se calculan y agregan nuevas columnas que servirán como características adicionales para el análisis, tales como:

  - Día, mes, año: Desgloses de la fecha.
  - Día de la semana: Indicador numérico del día de la semana (ordinal entre 1 y 7).
  - Hora, minuto, segundo: Desgloses de la hora del día.
  - Tipo de día: Indicador de si el día corresponde a una semana laboral, un feriado o un fin de semana.
  - Campo *travel_time*: Campo que indica para cada parada cuánto tiempo falta para llegar a destino. Este campo se calcula como el valor de fecha_pasada_real de la última parada de ese recorrido, menos el valor de fecha_pasada_real de la parada en la que se está.

[1] El dataset original cuenta con seis variantes, dos de las cuales se decide no incluir en el estudio por tener muy pocos datos (menos de 500 trayectorias finales).

3.5. Metadata y Partición del *Dataset*

Una vez finalizado el proceso de limpieza y la creación de nuevas características, se procede a separar los datos de las variantes, creando un *dataset* por cada variante existente. Finalmente, cada *dataset* queda con las siguientes características:

- `id_recorrido`: Identificador único de cada instancia de recorrido, que corresponde a un viaje realizado por la línea.
- `coche`: Número de identificación del ómnibus que realiza el viaje. Campo de tipo numérico.
- `parada`: Ordinal de la parada en el recorrido de la variante. Numérico.
- `variante`: Identificador de la variante, que representa la ruta específica (origen-destino). Numérico.
- `fecha_pasada`: Hora en la que el ómnibus pasa por la parada. Campo de tipo fecha y hora.
- `travel_time`: Valor calculado que indica para cada parada cuánto tiempo falta para llegar a destino. Valor en minutos.
- `tipo_día`: Valor calculado que indica si el día es entre semana, fin de semana o feriado.
- `día`: Ordinal que indica cuál día de la semana es. 1 indica lunes, 7 domingo.
- `dia_numero`: Valor numérico que indica el día del mes. Valor entre 1 y 31.
- `mes`: Valor numérico que indica el mes del año. Valor entre 1 y 12.
- `año`: Valor numérico que indica el año. Valor entre 2021 y 2024.
- `hora`: Valor numérico que indica la hora del día. Valor entre 0 y 23.
- `minuto`: Valor numérico que indica el minuto de la hora. Valor entre 0 y 59.
- `segundo`: Valor numérico que indica el segundo del minuto. Valor entre 0 y 59.

Finalmente, se particiona cada *dataset* en 70 %, 15 % y 15 % para Entrenamiento, Validación y *Testing*, respectivamente. La división se realizó de manera entera para evitar fraccionar los recorridos, motivo por el cual se puede observar una pequeña diferencia en la cantidad de recorridos a favor del conjunto de datos para *Testing*. La Tabla 3.3 detalla la cantidad final de recorridos para cada conjunto.

Variante	**Total**	**Entrenamiento**	**Validación**	***Testing***
634	25783	18048	3867	3868
635	29787	20850	4468	4469
2778	5118	3582	767	769
2779	4749	3324	712	713

Tabla 3.3: Datos por Variante y por tipo de Conjunto.

El proceso de transformación de los datos según si son variables discretas o categóricas, se explica en el Capítulo 5, ya que fue distinto para cada método.

Capítulo 4

Métodos Seleccionados

En este capítulo se detallan los métodos seleccionados para los experimentos y sus bases teóricas.

De acuerdo a la revisión del Estado del Arte, se decidió tomar una línea base constituida por un modelo lineal de series temporales, y dos modelos más performantes: uno de aprendizaje superficial[1] y otro de aprendizaje profundo. Lo esperado es que el modelo base de series temporales tenga menor desempeño que el de aprendizaje superficial, y este último a su vez sea peor que el de aprendizaje profundo.

Los métodos seleccionados para los experimentos fueron: *Autoregressive Integrated Moving Average* (**ARIMA**), *eXtreme Gradient Boosting* (**XGBoost**) y *Bidirectional Gated Recurrent Unit* (**Bi-GRU**).

Las bases teóricas de cada método, así como los fundamentos de la selección, se describen en las siguientes secciones.

4.1. *Autoregressive Integrated Moving Average*

Dado que los tiempos de viaje de los segmentos de un recorrido cambian con el tiempo, la predicción de tiempos de viaje puede considerarse un problema de predicción de series temporales (Fang y cols., 2022). Para resolver este tipo de problemas, existe una clase general de modelos llamados **ARIMA**, que proporciona un buen ajuste a muchos tipos de series temporales (Chatfield, 2016). Esta clase incluye como casos especiales a los modelos *Autoregressive* (**AR**), *Moving Average* (**MA**), *Autoregressive Moving Average* (**ARMA**), todos ellos basados en principios estadísticos, como se vio en el Capítulo 2.5.3.

Para acercar el concepto detrás de estos modelos, imaginemos que queremos predecir la temperatura máxima para hoy en cierta ciudad. El modelo a construir podría observar las temperaturas de los días anteriores para detectar patrones de cómo varía esta magnitud con el tiempo. Si la temperatura máxima

[1] El aprendizaje superficial se refiere a algoritmos de aprendizaje automático que no utilizan arquitecturas profundas como las redes neuronales de múltiples capas.

de ayer fue por ejemplo de 25°C, el modelo podría predecir un valor similar para hoy, basándose en las temperaturas tanto de ayer, como de varios días anteriores. Este tipo de dependencia con valores pasados es característico de la parte autorregresiva (AR) de esta familia de modelos.

Pero además de depender de los valores pasados, el modelo podría también ajustar sus predicciones en función de los errores previos. Este componente es el de media móvil (MA), que permite al modelo corregir las desviaciones en las predicciones, incorporando información sobre cómo han cambiado los errores en el pasado. A continuación se formaliza el concepto del modelo ARMA.

4.1.1. ARMA

El modelo ARMA combina dos enfoques fundamentales para modelar series temporales: el componente **autorregresivo** (AR), que indica la relación entre el valor actual y sus valores previos, y el componente de **media móvil** (MA), que indica la relación entre el término de error actual y los términos de errores previos (J. Li, Yue, y Hou, 2024).

Matemáticamente, un modelo ARMA(p, q), donde p es el orden de rezago del componente AR, y q es el orden de rezago del componente MA, se expresa como:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t, \tag{4.1}$$

donde:

- y_t es el valor de la serie en el tiempo t,
- c es una constante,
- ϕ_i representa los coeficientes del componente AR,
- θ_j representa los coeficientes del componente MA,
- ϵ_t es un término de error, que representa la parte impredecible o aleatoria de la serie en el instante t[2].

En el análisis de series temporales, el objetivo es predecir una serie que, típicamente no es determinística, sino que contiene un componente aleatorio. De acuerdo a Brockwell y Davis (2002), si este componente aleatorio es estacionario (noción que se define en la siguiente sección), entonces es posible desarrollar técnicas más precisas para predecir los valores futuros. En particular, el modelo ARMA solo es válido cuando la serie cumple con esta propiedad, por lo que antes de aplicarlo es fundamental verificar que la serie sea estacionaria. A continuación, se define formalmente este concepto.

[2] A lo largo del tiempo, los valores de ϵ_t forman una secuencia que es llamada ruido blanco.

4.1.2. Estacionariedad

Una serie temporal se considera estacionaria si sus propiedades estadísticas no dependen del momento en que se observe la serie. De forma más precisa, si (y_t) es una serie temporal estacionaria, entonces para todo s, la distribución[3] de $(y_t, \ldots, y_{t+s})$ no depende de t (Hyndman y Athanasopoulos, 2018).

Continuando con el ejemplo anterior, si analizamos una serie de temperatura máxima diaria en Montevideo, ésta puede mostrar una tendencia creciente en verano y decreciente en invierno. Esto indica no estacionariedad, ya que si observamos la serie en diferentes momentos del año, su media (el promedio de los valores) cambiará, lo que contradice la idea de que las propiedades estadísticas de la serie no dependen del momento en que se observe. Existen pruebas para determinar si los datos de una serie temporal siguen un proceso estacionario o no estacionario, una de ellas es la llamada *Augmented Dickey Fuller* (**ADF**) (Brockwell y Davis, 2002). Si tras aplicar la prueba ADF se indica que la serie no es estacionaria, se puede recurrir a técnicas como la diferenciación para intentar convertirla en estacionaria, como se detalla a continuación.

4.1.3. Diferenciación

La diferenciación es una técnica utilizada en análisis de series temporales para convertir una serie no estacionaria en estacionaria, calculando las diferencias entre observaciones consecutivas (Hyndman y Athanasopoulos, 2018). La serie resultante de estas diferencias se denomina serie diferenciada.

Matemáticamente, la diferenciación de primer orden se define como:

$$y'_t = y_t - y_{t-1} \tag{4.2}$$

donde:

- y_t es el valor de la serie en el tiempo t,
- y_{t-1} es el valor de la serie en el tiempo $t-1$.

Por ejemplo, si tenemos la serie original:

$$[10, 12, 15, 20, 25],$$

vemos que hay una tendencia creciente. Para eliminar esta tendencia, calculamos la serie diferenciada de primer orden restando cada valor con el anterior:

$$12 - 10 = \mathbf{2}, \quad 15 - 12 = \mathbf{3}, \quad 20 - 15 = \mathbf{5}, \quad 25 - 20 = \mathbf{5}$$

La nueva serie diferenciada es:

$$[2, 3, 5, 5]$$

[3] La distribución de una serie temporal describe cómo varían sus valores a lo largo del tiempo. Si esta distribución no cambia, significa que los patrones generales de la serie se mantienen estables, lo que permite modelarla con mayor precisión.

Este proceso ayuda a estabilizar la media de la serie temporal, eliminando tendencias fuertes. Si la serie sigue sin ser estacionaria, el proceso se puede repetir otra vez, lo que se conoce como diferenciación de segundo orden. En la práctica, casi nunca es necesario ir más allá de la diferenciación de segundo orden (Hyndman y Athanasopoulos, 2018).

Por lo tanto, es posible tratar la no estacionariedad diferenciando la serie temporal hasta que se vuelva estacionaria, y luego ajustar un modelo ARMA a la serie diferenciada. Este enfoque es precisamente el que sigue ARIMA, que incorpora la diferenciación como un paso previo para trabajar con series estacionarias (Chatfield, 2016).

4.1.4. ARIMA

El modelo ARIMA es una extensión de los modelos ARMA, diseñado para abordar series temporales no estacionarias. Al incluir un nuevo componente llamado **I**, de integración[4], ARIMA permite manejar de manera explícita la no estacionariedad de la serie antes de aplicar los componentes autorregresivos y de media móvil.

Matemáticamente, un modelo ARIMA(p, d, q) se expresa como:

$$y'_t = c + \sum_{i=1}^{p} \phi_i y'_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t, \tag{4.3}$$

donde:

- y'_t representa la serie diferenciada de orden d,
- las demás aclaraciones son idénticas a la ecuación 4.1.

La determinación de los hiperparámetros[5] p, d y q en ARIMA no es arbitraria, sino que se basa en análisis estadísticos de la serie temporal. En la práctica, existen herramientas computacionales que emplean estrategias automatizadas de búsqueda de hiperparámetros. La selección de hiperparámetros realizada en este proyecto se detallará en el Capítulo 5.4.1.

4.1.5. Fundamentos de la selección

Antes de la fase de experimentación, se evaluó la estacionariedad de las series temporales mediante la prueba ADF. A nivel macro (considerando cada variante en su conjunto), los resultados indicaron estacionariedad. Sin embargo, al analizar la serie a nivel de cada parada, se identificaron múltiples casos de no

[4] En este contexto "integrado" o "integración" se refiere al proceso inverso de la diferenciación, que reconstruye la serie original a partir de su forma diferenciada.

[5] Los hiperparámetros son valores que se fijan antes del entrenamiento del modelo y no se estiman directamente a partir de los datos. En el caso de ARIMA, p, d y q deben determinarse previamente, mientras que los parámetros del modelo, como ser los coeficientes ϕ_i y θ_j, se estiman durante la fase de ajuste. Para eso existen funciones disponibles en lenguajes como Python y R que realizan esta estimación.

estacionariedad. Este resultado sugiere que el problema de predicción requiera un enfoque capaz de manejar la no estacionariedad de la serie. Dado que ARIMA incorpora la diferenciación como un paso explícito para estabilizar la serie antes de modelarla, su uso resultó adecuado dentro de este contexto.

4.1.6. Consideraciones sobre ARIMA

A pesar de su capacidad para modelar muchas series temporales, la precisión de ARIMA depende de la adecuación de las funciones matemáticas para representar los datos históricos, y de la similitud entre los patrones pasados y futuros (Ou, 2022). Además, al tratarse de un modelo estadístico lineal, presenta limitaciones cuando los datos exhiben relaciones no lineales o estructuras más complejas (Fang y cols., 2022; Xu y Liu, 2021). En estos casos, su rendimiento tiende a ser bajo, y enfoques no lineales pueden ofrecer mejores resultados (Ashwini y cols., 2022). Por esta razón, en contextos más complicados, puede ser necesario recurrir a modelos alternativos, con técnicas de aprendizaje automático o aprendizaje profundo, que permiten capturar patrones más complejos y no lineales de manera más efectiva.

4.2. *eXtreme Gradient Boosting*

XGBoost es un método ampliamente utilizado por científicos de datos debido a su capacidad para lograr resultados de vanguardia en tareas de aprendizaje automático (T. Chen y Guestrin, 2016). De acuerdo a sus creadores (Chen y Guestrin), su éxito se debe principalmente a su escalabilidad, siendo más de diez veces más rápido que otras soluciones populares, y capaz de procesar millones de ejemplos incluso en entornos con recursos limitados. Además, XGBoost ha presentado un rendimiento dominante en muchas competiciones de aprendizaje automático e inteligencia artificial aplicada (H. Huang y cols., 2020).

El método pertenece a la familia de los modelos basados en árboles de decisión (o regresión)[6]. Estos modelos dividen el espacio de datos en regiones más pequeñas siguiendo un conjunto de reglas de decisión. A continuación, se introduce el concepto de árboles de regresión, seguido de una explicación detallada sobre los fundamentos de XGBoost.

4.2.1. Árboles de Regresión

Un árbol de regresión es un modelo de aprendizaje automático no lineal aplicado para predecir una variable continua. En los modelos basados en árboles, no hay ecuaciones matemáticas para demostrar la relación entre múltiples variables, sino que se busca construir un modelo que pronostique la variable objetivo

[6]Los árboles de decisión pueden utilizarse tanto para problemas de clasificación como de regresión. En el caso de clasificación, se denominan simplemente árboles de decisión, mientras que para variables continuas se les conoce como árboles de regresión.

aprendiendo a través de reglas simples a partir de los datos de entrenamiento (Ashwini y cols., 2022).

El modelo divide iterativamente los datos en subconjuntos más pequeños según una serie de reglas basadas en las características de entrada. Cada nodo interno del árbol representa una condición o pregunta sobre un atributo, cada rama corresponde a un resultado posible, y las hojas del árbol contienen la predicción final.

La Figura 4.1 muestra un árbol de regresión para predecir el tiempo de viaje empleado por un vehículo dadas tres características de entrada: el día de la semana, la hora, y la condición climática. La característica "día de la semana" con el valor "fin de semana" se elige como primer punto de división. Los dos nodos siguientes dividen el espacio de características en las observaciones que pertenecen a los fines de semana (subárbol izquierdo) y todos los demás días (subárbol derecho). La variable "hora del día" con el valor de la característica "7 am" se elige para dividir los datos pertenecientes a los fines de semana produciendo dos hojas con los valores finales de predicción (12.5 minutos, y 7.4 minutos). Análogamente para la rama de la derecha, la característica "llueve" divide nuevamente en dos hojas de predicción (27.1 minutos, y 22.3 minutos). A modo de ejemplo, si se quiere predecir el tiempo de viaje en un día de semana con lluvia, el resultado del modelo será de 27.1 minutos.

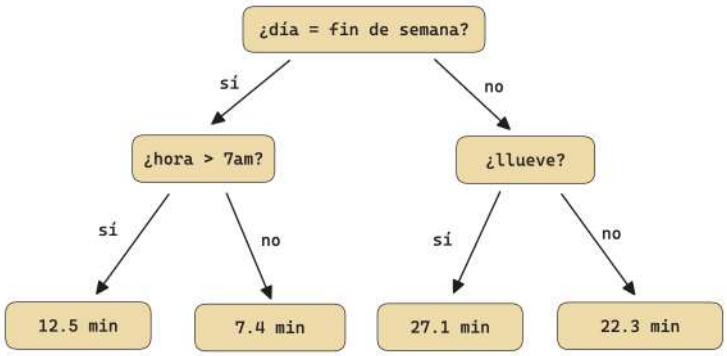

Figura 4.1: Árbol de regresión de profundidad 2. Fuente: elaboración propia.

Los árboles de regresión suelen tener un buen rendimiento predictivo en los conjuntos de entrenamiento, pero demasiadas divisiones pueden provocar un sobreajuste y un rendimiento deficiente en los conjuntos de prueba (N. Li y cols., 2023). Con el objetivo de mejorar el rendimiento de predicción y abordar las limitaciones de los árboles de regresión individuales, se han desarrollado enfoques avanzados que combinan múltiples árboles para aprovechar su poder predictivo colectivo (Ahmed y cols., 2022). Este tipo de métodos, conocidos como *ensemble learning* o aprendizaje por conjunto, permiten mejorar tanto la precisión como la robustez del modelo.

Dentro del *ensemble learning*, existen dos estrategias principales: ***bagging*** y ***boosting***. Mientras que *bagging* tiene un proceso paralelo –es decir, cada

árbol de regresión funciona de manera independiente y luego sus salidas se agregan al final–, *boosting* se comporta como un proceso gradual que mejora la predicción desarrollando múltiples modelos en secuencia, enfatizando en aquellos casos de entrenamiento que son difíciles de estimar (Qiu y Fan, 2021). Esta última estrategia es la que utiliza XGBoost, y sobre la que se profundiza a continuación.

4.2.2. *Boosting*

Boosting es una técnica que combina múltiples aprendices débiles[7] para construir un modelo fuerte. El concepto de impulso (o *boosting*) surge de la idea de que un aprendiz deficiente puede ser modificado para convertirse en un aprendiz mejor. Para esto se entrenan árboles sucesivos en serie con el objetivo de que el árbol *n-ésimo* corrija los errores de predicción de los árboles anteriores (Moosavi y cols., 2023). Finalmente, estos aprendices débiles se combinan en un aprendiz fuerte para mejorar el rendimiento de la predicción (J. Li y cols., 2024).

Cada vez que se genera un nuevo árbol de regresión, el modelo se actualiza y mejora basándose en el modelo previo y en la función de pérdida[8]. La forma de minimizar la pérdida al añadir nuevos modelos se realiza mediante un algoritmo de descenso por gradiente[9]. La idea principal es utilizar el residuo (la diferencia) generado por los valores predichos y reales durante el proceso de ajuste, para entrenar cada árbol nuevo, con el fin de acercarse constantemente al valor real (N. Li y cols., 2023). Cuando se alcanza un umbral de error cercano a cero, el ciclo finaliza.

4.2.3. Introducción a XGBoost

XGBoost es un algoritmo de *boosting* que destaca por su eficiencia y precisión tanto en tareas de clasificación como de regresión. Entre sus principales ventajas, resalta su capacidad para manejar grandes volúmenes de datos con múltiples variables y la posibilidad de trabajar con valores perdidos, lo que lo hace adecuado para una amplia variedad de conjuntos de datos (Zúñiga, 2020). Además, es conocido por su excelente velocidad de ejecución y la alta precisión de los resultados que produce (Ali, Abduljabbar, Tahir, Sallow, y Almufti, 2023). Su diseño incorpora varias estrategias de optimización que permiten acelerar el entrenamiento y mejorar el rendimiento del modelo.

En esta sección, se abordarán las optimizaciones clave de XGBoost, como ser la paralelización, la poda de árboles, la optimización del uso del hardware, y las técnicas de aleatorización.

[7] Un aprendiz débil es un modelo que por sí solo tiene un desempeño limitado y realiza predicciones poco precisas.

[8] La función de pérdida mide qué tan bien el modelo está realizando sus predicciones en comparación con los valores reales.

[9] El gradiente es un vector que contiene las derivadas parciales de la función de pérdida respecto a los parámetros del modelo.

Al final de la sección, se mostrará el flujo de predicción, que ilustra el proceso de ajuste del modelo, y un pseudocódigo del algoritmo simplificado de XGBoost, para proporcionar un acercamiento práctico a la comprensión del método.

Paralelización

Dado que la construcción de árboles puede generar una gran cantidad de nodos, XGBoost utiliza paralelización para procesar múltiples tareas en simultáneo, acelerando significativamente la generación de árboles. Al reorganizar los datos en bloques comprimidos y ordenados por característica, el algoritmo no solo mejora la eficiencia del proceso de entrenamiento, sino que también optimiza el uso de los recursos de CPU (T. Chen y Guestrin, 2016).

Poda de Árboles

La poda de árboles es una técnica fundamental utilizada para reducir el tamaño de los árboles de regresión mediante la eliminación de nodos que no contribuyen significativamente a la mejora de la predicción. Su propósito principal es evitar el sobreajuste, asegurando que el modelo generalice correctamente en nuevos datos sin ajustarse en exceso a los datos de entrenamiento (Ali y cols., 2023).

XGBoost genera nodos hasta la profundidad máxima establecida y luego comienza a podar en sentido inverso hasta que la pérdida sea menor que un umbral especificado. Esto permite obtener modelos más compactos y con mejor capacidad de generalización.

Optimización del Hardware

XGBoost optimiza el uso del hardware reduciendo la latencia de acceso a memoria y mejorando la eficiencia computacional. Almacena las estadísticas del gradiente (dirección y valor) para cada nodo de división en un búfer interno de cada hilo, acumulándolas en mini-lotes mediante una optimización que mejora el uso de la caché. Esto minimiza el tiempo de las operaciones de lectura/escritura y evita fallos de caché.

La eficiencia en el uso de la caché se logra seleccionando el tamaño de bloque óptimo, equilibrando paralelización y rendimiento, lo que mejora la velocidad del entrenamiento. Además, para manejar grandes volúmenes de datos, XGBoost usa técnicas de computación fuera de memoria, dividiendo los datos en bloques almacenados en disco y pre-cargándolos en un búfer en memoria, permitiendo la superposición de cómputo y lectura de disco (T. Chen y Guestrin, 2016).

Técnicas de Aleatorización

XGBoost implementa varias técnicas de aleatorización para reducir el sobreajuste y mejorar la velocidad de entrenamiento. Esto incluye el muestreo aleatorio de datos para construir cada árbol y la selección aleatoria de características, tanto a nivel de árbol como de nodo. Además, el algoritmo prioriza

las muestras de datos con mayor tasa de error, aumentando la probabilidad de que las observaciones más difíciles de predecir sean seleccionadas en iteraciones posteriores. Esto permite mejorar la precisión del modelo de manera progresiva (Qiu y Fan, 2021).

Funcionamiento de XGBoost

A partir de las técnicas de aleatorización, XGBoost logra construir un ensamble de árboles de manera eficiente, combinando múltiples aprendices débiles para formar un modelo fuerte, tal como se explicó en la sección de *Boosting*.

La Figura 4.2 ilustra el flujo del proceso de predicción en XGBoost durante el entrenamiento. Inicialmente, se parte de un conjunto de datos de muestra, del cual se extraen subconjuntos mediante selección aleatoria de columnas. Cada subconjunto de entrenamiento se construye con una selección diferente de características y se usa para entrenar un árbol de decisión.

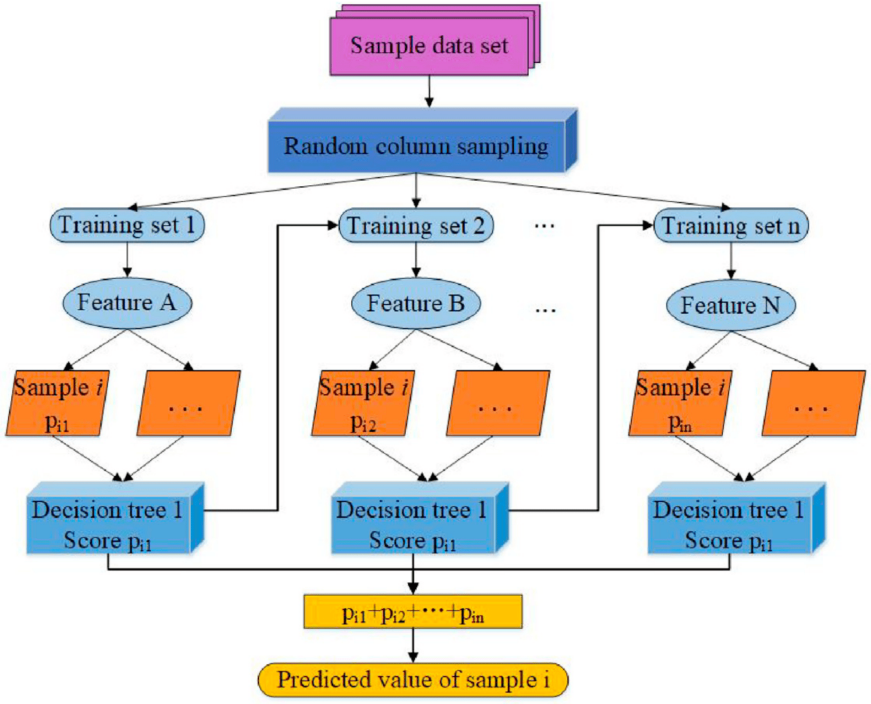

Figura 4.2: Proceso de predicción de XGBoost. Fuente: (N. Li y cols., 2023).

Para un ejemplo de entrada i, cada árbol genera una predicción parcial p_{ij}, donde j indica el árbol correspondiente. Estas predicciones no son independientes, sino que cada nuevo árbol se entrena considerando los errores acumulados por los árboles previos. La predicción final se obtiene sumando todas estas con-

tribuciones, lo que permite que el modelo se ajuste progresivamente a los datos y minimice el error de predicción.

Si bien este esquema ilustra de manera conceptual el proceso de predicción de XGBoost, a continuación, se presenta un pseudocódigo que resume de manera simplificada el algoritmo. En términos generales, el modelo comienza con una predicción base y luego, en cada iteración, se entrena un nuevo árbol h_m para ajustar los residuos de la predicción anterior y actualizar el modelo F_m.

Para evitar que el modelo aprenda demasiado rápido y reducir el riesgo de sobreajuste, se introduce la tasa de aprendizaje (η), un parámetro que controla cuánto contribuye cada nuevo árbol a la actualización del modelo.

El procedimiento continúa hasta alcanzar el número máximo de árboles (`n_estimators`) o hasta que el error sea suficientemente pequeño ($\|r_m\| < \epsilon$). Si bien aquí la norma del residuo se ha definido de manera genérica, en la práctica, el MSE es comúnmente usado para medir este error.

El Algoritmo 1 muestra este proceso en términos generales, omitiendo detalles técnicos avanzados, con el objetivo de mantener la claridad del flujo de entrenamiento.

Algoritmo 1 Algoritmo simplificado de XGBoost

Input: Conjunto de entrenamiento, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
Input: Tasa de aprendizaje, η
Input: Número máximo de árboles, $n\_estimators$
Input: Umbral de error, ϵ

Inicialización:

Obtener el primer modelo base F_0 (árbol inicial)

Calcular el residuo $r_0 \leftarrow y - F_0(x)$

for $m = 1$ ***hasta*** $n\_estimators$ **do**

Paso 1:

Obtener un nuevo árbol h_m que ajuste el residuo r_{m-1}

Paso 2:

Actualizar la predicción del modelo agregando h_m, ponderado por η

$F_m(x) \leftarrow F_{m-1}(x) + \eta \cdot h_m(x)$

Paso 3:

Calcular el nuevo residuo:

$r_m \leftarrow y - F_m(x)$

Paso 4:

if $\|r_m\| < \epsilon$ **then**

break

Output: Modelo final, F_m

4.2.4. Función de Pérdida y Regularización[10]

El núcleo de XGBoost se encuentra en la definición de una función objetivo, que integra dos componentes clave: el término de la **función de pérdida** y el término de **regularización**. Esta función no solo busca minimizar la discrepancia entre las predicciones y los valores reales, sino que también penaliza la complejidad del modelo para evitar el sobreajuste y mejorar su capacidad de generalización (J. Li y cols., 2024).

La función objetivo se puede definir entonces como:

$$Obj(\Theta) = \mathcal{L}(\Theta) + \Omega(\Theta) \tag{4.4}$$

donde:

- $\mathcal{L}(\Theta)$ es el término de la función de pérdida de entrenamiento.
- $\Omega(\Theta)$ es el término de regularización, que indica la complejidad del modelo.

Sea $\hat{y}_i^{(t)}$ la predicción de la instancia i-ésima en la iteración t-ésima, entonces la función objetivo se puede expresar como:

$$Obj(t) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{4.5}$$

donde:

- y_i es el valor real en la observación i.
- l es la función de pérdida utilizada.
- $f_t(x_i)$ es la contribución del nuevo árbol en la iteración t.

Para mejorar la eficiencia de resolución del método, se puede utilizar la expansión de Taylor de segundo orden (Qiu y Fan, 2021; N. Li y cols., 2023):

$$Obj(t) \simeq \sum_{i=1}^{n} \left(l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t) \tag{4.6}$$

donde:

- g_i es la primera derivada parcial de la función de pérdida.
- h_i es la segunda derivada parcial de la función de pérdida.

Además de estas optimizaciones, XGBoost también realiza ajustes adicionales en la estructura de los árboles y en la selección de divisiones óptimas, mejorando aún más el rendimiento del modelo. Estos procesos, detallados en T. Chen y Guestrin (2016), quedan fuera del alcance de este capítulo.

[10] La regularización es una técnica utilizada en el aprendizaje automático para penalizar la complejidad del modelo y evitar el sobreajuste, lo que mejora su capacidad de generalización.

4.2.5. Hiperparámetros

XGBoost es un método altamente configurable con una variedad de parámetros agrupados en diferentes categorías. A continuación, se presentan algunas de estas categorías, destacando los parámetros más relevantes de ellas. Para un listado exhaustivo de los parámetros acceder a la documentación oficial (*XGBoost Documentation: Parameters*, 2025).

- **Parámetros Generales**: Configuran aspectos básicos del algoritmo.
  - `nthread`: Número de hilos paralelos utilizados para ejecutar XGBoost.
  - `silent`: Controla si el algoritmo muestra mensajes durante la ejecución (`0`) o permanece en modo silencioso (`1`).
- **Parámetros del *Booster***: Especifican cómo se construyen los árboles.
  - `n_estimators`: Número de árboles a construir. Un valor mayor puede mejorar el rendimiento, pero también aumenta la complejidad del modelo (con su riesgo de sobreajuste) y el costo computacional.
  - `learning_rate`: Tasa de aprendizaje que controla cuánto contribuye cada árbol al modelo final. Valores pequeños (como `0.01` o `0.001`) hacen que el entrenamiento sea más lento, pero pueden mejorar la generalización. Valores más altos permiten que el entrenamiento sea más rápido, pero aumenta el riesgo de sobreajuste.
  - `max_depth`: Profundidad máxima de los árboles. Controla la complejidad del modelo. Profundidades mayores permiten aprender relaciones más complejas, pero pueden aumentar el riesgo de sobreajuste.
  - `subsample`: Proporción de muestras utilizadas para cada árbol. Define qué fracción del conjunto de datos de entrenamiento se usa para construir cada árbol. Un valor de `1.0` significa que se usan todas las muestras, mientras que un valor menor reduce la cantidad de datos y puede ayudar a evitar el sobreajuste.
  - `colsample_bytree`: Proporción de columnas (o características) seleccionadas aleatoriamente al construir cada árbol. Ayuda a reducir el riesgo de sobreajuste.
  - `gamma`: Reducción mínima en la función de pérdida que se necesita para que un nodo sea dividido. Si la ganancia obtenida al dividir un nodo no supera este umbral, la división no ocurre. Un valor mayor hace que el algoritmo sea más conservador.
- **Parámetros de la Tarea de Aprendizaje**: Especifican la tarea de aprendizaje y el objetivo de aprendizaje correspondiente.
  - `objective`: Define la función objetivo que el modelo optimizará durante el entrenamiento.

- `eval_metric`: Especifica las métricas de evaluación para los datos de validación.
- `seed`: Establece una semilla para la reproducibilidad.

Los parámetros de XGBoost son fundamentales para controlar el modelo, reducir el error de precisión y prevenir el sobreajuste, especialmente con un número elevado de árboles. Esta selección de hiperparámetros se realiza mediante una búsqueda automatizada. En el siguiente capítulo se amplía este tema, con la selección de los hiperparámetros para este proyecto.

4.2.6. Fundamentos de la selección

A lo largo de la Sección 4.2 se han destacado las múltiples características que hacen de XGBoost un método altamente eficiente. Su capacidad para manejar grandes volúmenes de datos, junto con su velocidad y precisión, lo convierten en una opción ideal para el problema de predicción de tiempos de viaje. La literatura ha validado consistentemente a XGBoost como un algoritmo de referencia en tareas de predicción, superando a otros métodos en estabilidad y rendimiento. Su capacidad para capturar relaciones no lineales en los datos y su excelente desempeño en series temporales refuerzan su selección para este proyecto.

Dada la necesidad de un modelo robusto y eficiente que pueda adaptarse a las fluctuaciones del tráfico, XGBoost es una opción adecuada para nuestro modelo de aprendizaje automático superficial.

4.2.7. Consideraciones sobre XGBoost

Si bien XGBoost es un método altamente eficiente, su aplicación requiere ciertas consideraciones. En primer lugar, puede demandar un uso significativo de recursos computacionales cuando se trabaja con bases de datos muy grandes. Para mitigar este impacto, es recomendable realizar un análisis previo de la importancia de las características y descartar aquellas que aporten poca información al modelo (Zúñiga, 2020). Esto permite mejorar el desempeño del modelo sin comprometer su capacidad predictiva. En el Capítulo 5.4.2 se retomará este concepto, presentando varios modelos de XGBoost, cada uno entrenado con un conjunto específico de características.

Por otro lado, XGBoost solo trabaja con vectores numéricos, por lo que se requiere convertir previamente los tipos de datos no numéricos a numéricos.

En cuanto a su capacidad para manejar información secuencial, XGBoost trabaja exclusivamente con datos estructurados en formato tabular. Esto puede representar una limitación en series temporales de longitud variable, ya que su adaptación a una longitud fija puede alterar la estructura original de los datos, eliminando información relevante o introduciendo valores artificiales (Shen y cols., 2022). En nuestro caso, esta restricción no afecta al modelo, dado que las paradas del ómnibus establecen una secuencia fija en cada variante.

Por último, la eficiencia de XGBoost puede degradarse a medida que el volumen de datos crece considerablemente (Chughtai, Haq, y Muneeb, 2022).

En estos casos, es conveniente explorar modelos más avanzados, como las redes neuronales profundas, que se abordan en la siguiente sección.

4.3. *Bidirectional Gated Recurrent Unit*

El aprendizaje profundo ha revolucionado múltiples áreas de la inteligencia artificial al permitir modelar relaciones complejas en los datos, mediante estructuras jerárquicas de procesamiento. Los resultados experimentales muestran que los enfoques de aprendizaje profundo superan ampliamente a los métodos basados en estadísticas y en aprendizaje automático (Nithishwer y cols., 2022). Mientras estos últimos dependen en gran medida de la ingeniería de características, las redes neuronales profundas pueden aprender representaciones abstractas directamente a partir de los datos. Sin embargo, estas técnicas también requieren un costo computacional relativamente alto (C. H. Chen y cols., 2019).

En el núcleo de estas arquitecturas se encuentra la neurona artificial, que en esencia es una función matemática que recibe un conjunto de entradas, las transforma mediante una operación no lineal, y devuelve una salida. Estas neuronas se organizan en capas, donde cada capa aplica transformaciones sucesivas a los datos de entrada, permitiendo que el modelo aprenda representaciones cada vez más abstractas.

A partir de este principio, han surgido diversas arquitecturas diseñadas para abordar distintos tipos de problemas. En tareas que involucran datos secuenciales, como series temporales, las redes neuronales recurrentes permiten identificar patrones y tendencias a lo largo del tiempo (Fang y cols., 2022). Entre ellas, Bi-GRU es un modelo bidireccional que procesa la información en dos direcciones, hacia adelante y hacia atrás en la secuencia de entrada, mejorando así su capacidad de aprendizaje (C. Y. Chen y cols., 2023).

A lo largo de esta sección se presentan los fundamentos de las redes neuronales, comenzando con las redes *feed-forward*, seguido de las redes neuronales recurrentes, donde se tratará la arquitectura GRU y su versión bidireccional Bi-GRU. Finalmente, se fundamenta la selección de esta arquitectura y se consideran diferentes aspectos sobre la aplicación de estos modelos.

4.3.1. Redes *feed-forward*

Las redes neuronales artificiales (ANN, por su sigla en inglés) están compuestas por múltiples capas de neuronas interconectadas. Cada capa se puede ver como una "columna" de neuronas que operan en paralelo.

Cada neurona recibe un conjunto de entradas ponderadas por pesos que reflejan su importancia en el procesamiento de la información. Luego, realiza una combinación lineal de estas entradas y aplica una función de activación no lineal para generar la salida.

La **función de activación** no lineal es clave, ya que sin ella la red solo aplicaría una transformación lineal, limitando su capacidad de modelar relaciones complejas. La noción de activación refleja cuánto responde una neurona a sus

entradas, modulando así el flujo de información en la red. Algunos ejemplos de funciones de activación son: sigmoide, tangente hiperbólica, ReLU, y ELU.

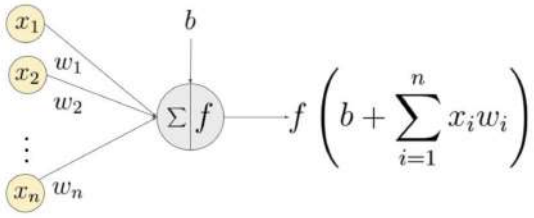

Figura 4.3: Neurona artificial. Fuente: (*Understanding Activation Functions in Deep Learning*, 2025)

La figura 4.3 muestra una neurona artificial, donde (x_i) representa las entradas, (w_i) los pesos asociados a dichas entradas y (f) la función de activación no lineal. Además, se incorpora el sesgo (b) que actúa como un desplazamiento en la entrada de la función de activación, similar al término independiente en una ecuación matemática. Su propósito es otorgar mayor flexibilidad a la neurona, permitiendo que pueda activarse incluso cuando las entradas sean bajas o nulas, lo que ayuda a mejorar la capacidad de la red para ajustarse a los datos.

Matemáticamente, la neurona puede expresarse como:

$$y = f\left(b + \sum_{i=1}^{n} x_i w_i\right) \tag{4.7}$$

donde (y) representa la salida de la neurona. Esta ecuación puede modelarse de manera más conveniente usando una representación vectorial:

$$y = f(b + \vec{x}.\vec{w}) \tag{4.8}$$

donde $(\vec{x})$ es el vector de entradas y $(\vec{w})$ es el vector de pesos, los cuales en su forma vectorial se representan como:

$$\vec{x} = [x_1, x_2, \ldots, x_n], \quad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Los pesos y el sesgo de cada neurona se ajustan usando el conjunto de datos de entrenamiento, generalmente mediante un algoritmo de retropropagación (***backpropagation***). La red optimiza estos parámetros de manera iterativa para reducir la función de pérdida, aprendiendo las representaciones necesarias para modelar la relación entre las entradas y las salidas.

Entre las arquitecturas más simples se encuentran las redes *feed-forward*, donde la información fluye en una única dirección: desde una **capa de entrada**

que recibe los datos, pasando por una o más **capas ocultas** encargadas del procesamiento intermedio, hasta una **capa de salida** que genera el resultado.

En este tipo de redes, la salida de cada neurona se conecta con todas las neuronas de la capa siguiente, formando lo que se conoce como una red densa. Cuando se utilizan múltiples capas ocultas, la red se denomina ***Multi-Layer Perceptron*** (MLP), un modelo capaz de aprender representaciones más complejas y abstractas de los datos. Su capacidad de aprendizaje depende de la cantidad de capas y neuronas: un diseño demasiado simple puede limitar la capacidad de generalización, mientras que uno excesivamente grande puede llevar al sobreajuste (Awad y Khanna, 2015).

La Figura 4.4 ilustra un MLP con n capas ocultas. Si bien en este caso el número de entradas y salidas coincide con n, en general no existe una relación fija entre estos elementos, ya que la arquitectura del modelo depende de la naturaleza del problema.

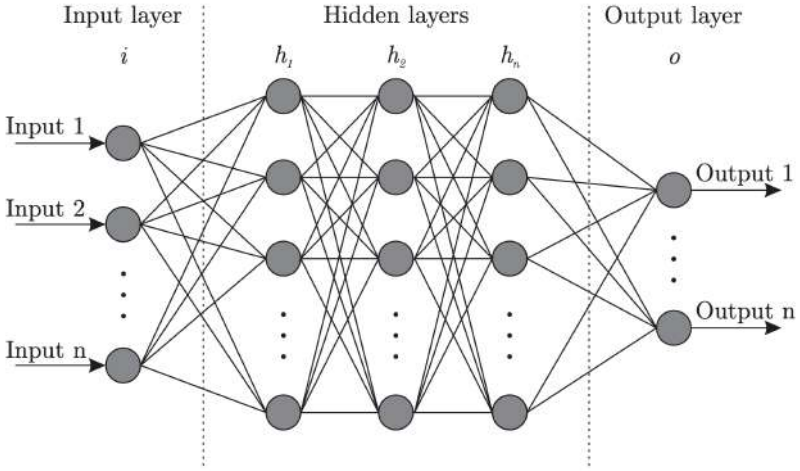

Figura 4.4: *Multilayer Perceptron*. Fuente: (Bre y cols., 2017).

A pesar de su amplio uso en diversas áreas, estos modelos, al aplicarse a series temporales, tienden a recordar fácilmente dependencias a largo plazo, lo que puede afectar el aprendizaje presente y futuro. Sin un mecanismo adecuado para regular la memoria, la red puede verse afectada por información desactualizada o irrelevante, lo que dificulta su capacidad para aprender y representar correctamente los patrones temporales (C. Y. Chen y cols., 2023).

Más allá de esta limitación, los MLP son un componente fundamental en arquitecturas profundas, donde se incorporan como capas ocultas para transformar las representaciones generadas por otras partes de la red.

No obstante, con la motivación de mejorar el modelado de datos secuenciales y capturar de manera más efectiva las dependencias temporales, surgieron arquitecturas avanzadas que integran mecanismos de memoria, como las ***Recu-***

rrent Neural Network (RNN), las cuales combinan información de distintos instantes de tiempo.

4.3.2. *Recurrent Neural Network*

A diferencia de los modelos de redes *feed-forward*, las RNN poseen conexiones retroalimentadas que le permiten mantener y utilizar la información de estados anteriores. A esta especie de memoria interna que almacena información sobre los estados anteriores de la secuencia procesada, se le denomina **estado oculto**, el cual se actualiza en cada paso temporal, permitiendo que la red integre de forma dinámica el historial al procesar nuevas entradas.

En la Figura 4.5 se muestra una red neuronal recurrente profunda con tres capas ocultas. La notación de la izquierda representa la estructura compacta de la red, donde cada capa oculta mantiene un estado recurrente. A la derecha, se despliega la red a lo largo de varios pasos de tiempo, mostrando cómo el estado se propaga en la secuencia.

Las RNN son ampliamente utilizadas para capturar las dependencias temporales en el aprendizaje secuencial, debido a su condición de "memorizar" el historial en la secuencia procesada. Además, también consideran las relaciones entre las salidas históricas y las entradas actuales (Fang y cols., 2022).

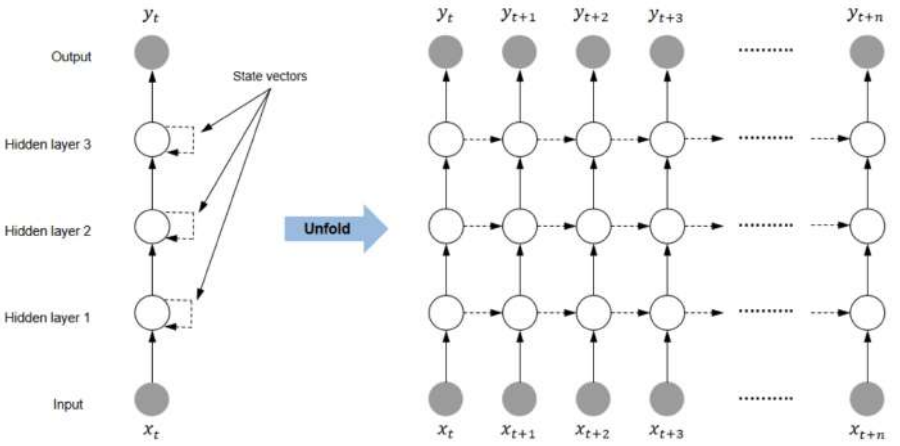

Figura 4.5: Red neuronal recurrente profunda y su despliegue en el tiempo. Fuente: (Moradi y Samwald, 2022).

Sin embargo, a medida que la longitud de la secuencia de entrada aumenta, las RNN estándar sufren de dos problemas conocidos en aprendizaje profundo bajo el nombre de **explosión del gradiente** y **desvanecimiento del gradiente**. Ambos problemas ocurren durante el proceso de *backpropagation*, en el cual, a partir del conjunto de entrenamiento, los pesos de la red se ajustan a medida que los gradientes de la función de pérdida se retropropagan desde la capa de salida hacia la capa de entrada.

La explosión del gradiente ocurre cuando los gradientes crecen rápidamente, lo que puede provocar que los parámetros de las capas cercanas a la entrada se actualicen de manera desmesurada, volviendo inestable el aprendizaje.

En cambio el desvanecimiento del gradiente ocurre cuando los gradientes de las capas más cercanas a la entrada se vuelven muy pequeños, lo que provoca que los parámetros de estas capas reciban actualizaciones mínimas, ralentizando el aprendizaje o incluso deteniéndolo por completo.

Para superar estas limitaciones se desarrollaron dos variantes específicas: ***Long Short-Term Memory*** (LSTM) y ***Gated Recurrent Unit*** (GRU). Estos modelos utilizan un mecanismo de compuertas para controlar el flujo de información a lo largo del tiempo, lo que no solo les permite aprender dependencias a largo plazo, sino que también mitigan los problemas de desvanecimiento y explosión del gradiente.

LSTM fue el primero en proponerse, y GRU surgió como una variante más simple, que utiliza un conjunto de compuertas reducido pero igualmente efectivo para gestionar el flujo de información a lo largo de la secuencia. El hecho de que el número de compuertas se reduzca, provoca que se disminuyan los parámetros de entrenamiento, y por consecuencia que se aumente tanto la velocidad de entrenamiento como la eficiencia general del modelo (Jakteerangkool y Muangsin, 2020).

En lo que sigue, se describe la arquitectura GRU y el funcionamiento de sus compuertas, las cuales permiten gestionar de manera eficiente la memoria a lo largo de una secuencia de entrada.

4.3.3. *Gated Recurrent Unit*

GRU introduce un mecanismo de procesamiento secuencial basado en unidades recurrentes (o celdas) que regulan el flujo de información mediante dos compuertas: **compuerta de reinicio** y **compuerta de actualización**. La Figura 4.6 muestra la arquitectura de la celda GRU, donde:

- x_t es la entrada en el instante t, que representa el dato actual de la secuencia
- y_t es la salida en el instante t, derivada del estado oculto h_t
- h_{t-1} es el estado oculto previo, que almacena la memoria de la secuencia hasta el instante $t-1$
- h'_t es el estado candidato, que representa una actualización intermedia de la memoria en el instante t
- h_t es el estado oculto final en el instante t, que se usará en el siguiente paso de la secuencia
- r_t es la compuerta de reinicio, que controla cuánta información de h_{t-1} se usará para calcular h'_t

- z_t es la compuerta de actualización, que controla cuánta información de x_t y h_{t-1} se integrará en el estado oculto final h_t y en la salida y_t

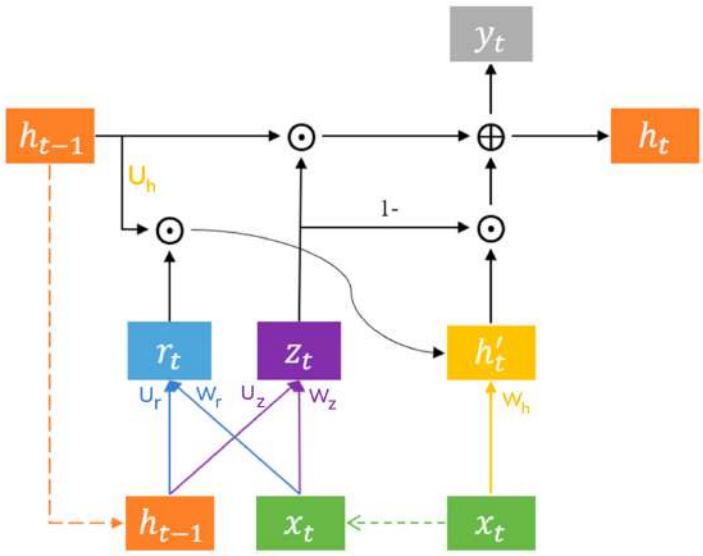

Figura 4.6: Arquitectura de GRU. Fuente: adaptación de (Fang y cols., 2022).

Si bien en el diseño original de GRU no se define explícitamente una salida y_t en cada paso temporal, en arquitecturas de redes recurrentes profundas, como la de la Figura 4.5, cada celda puede producir una salida intermedia que alimente a capas posteriores o que contribuya a la salida final, dependiendo de la estructura del modelo.

Las ecuaciones que reflejan el flujo de información dentro de la celda GRU en el instante t, se pueden expresar de la siguiente manera:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{4.9}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{4.10}$$

$$h'_t = \tanh(W_h x_t + r_t \odot U_h h_{t-1} + b_h) \tag{4.11}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{4.12}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.13}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.14}$$

donde:

- σ y tanh son las funciones de activación sigmoide y tangente hiperbólica
- $\odot$ representa el producto elemento a elemento
- W_r, W_z y W_h son matrices de pesos que transforman x_t
- U_r, U_z y U_h son matrices de pesos que transforman h_{t-1}
- b_r, b_z y b_h son sesgos aplicados como entrada a las funciones de activación

El mecanismo de compuertas de GRU almacena de manera flexible la información histórica durante el aprendizaje del patrón de una serie temporal. Si la compuerta de reinicio se desactiva totalmente ($r_t \to 0$), el modelo tiende a enfocarse en la dependencia a corto plazo, olvidando la información desactualizada. Esto se puede ver en la componente de la derecha de la ecuación 4.11. Por otro lado, cuando la compuerta de actualización se activa totalmente ($z_t \to 1$), el modelo considera más información del historial y obtiene la dependencia a largo plazo de la secuencia. Esto se puede ver en la ecuación 4.12.

Si bien el modelo GRU fue ideado originalmente para procesar secuencias en un único sentido (por ejemplo de izquierda a derecha, o de derecha a izquierda), es posible combinar dos capas de redes recurrentes para que el procesamiento se realice en los dos sentidos, mejorando así la comprensión contextual. Un modelo así es llamado ***Bidirectional* GRU**, que lo abreviamos **Bi-GRU**.

Finalizaremos la exposición de los modelos de redes neuronales con el desarrollo de Bi-GRU, ya que ha sido el método seleccionado para la construcción del modelo de aprendizaje profundo.

4.3.4. Bi-GRU

Una capa recurrente bidireccional consta de dos capas recurrentes, una de izquierda a derecha y la otra de derecha a izquierda, con sus salidas concatenadas en cada paso. Eventualmente las capas se podrían procesar en secuencia, utilizando la salida de una capa como entrada de la siguiente.

El uso de una arquitectura bidireccional permite que ambas direcciones procesen la misma secuencia de datos, lo que mejora la selectividad de la memoria. Esta estructura ayuda a reducir la duplicación innecesaria de información y a minimizar la omisión de datos relevantes. Además, al operar en ambas direcciones, el modelo reconoce y confirma la información importante dos veces, contribuyendo así a un mejor rendimiento (C. Y. Chen y cols., 2023).

La Figura 4.7 muestra la arquitectura de una red Bi-GRU donde, en el instante t se cuenta con dos celdas GRU, una que recibe x_t y h'_{t-1} para procesar la información hacia adelante, y otra que recibe x_t y h_{t+1} para procesar la información hacia atrás. Los estados ocultos resultantes de estas celdas (h_t y h'_t), se pasan a través de una función de activación para obtener la salida y_t. En este modelo las celdas GRU devuelven únicamente su estado oculto, no existe una salida independiente como en la Figura 4.6.

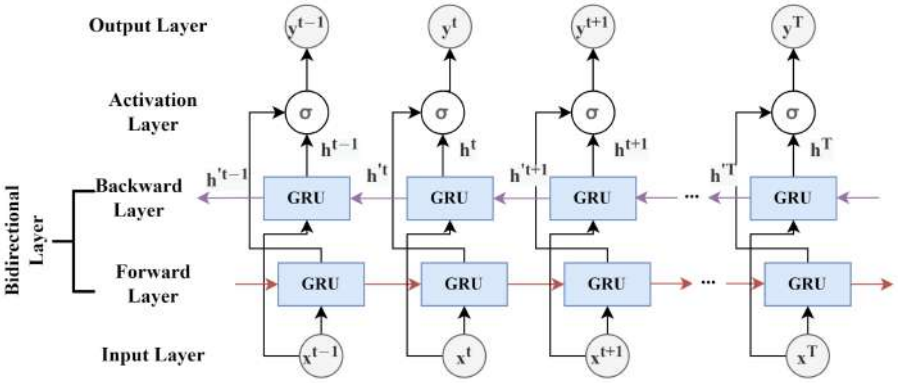

Figura 4.7: Arquitectura de Bi-GRU. Fuente: (Chughtai, ul Haq, y cols., 2022).

Utilizando una notación más gráfica, llamamos $\overrightarrow{h_t}$ y $\overleftarrow{h_t}$ a los estados ocultos resultantes de cada celda, entonces las ecuaciones para este modelo Bi-GRU se pueden expresar como:

$$\overrightarrow{h_t} = \overrightarrow{GRU}(x_t, \overrightarrow{h_{t-1}}) \tag{4.15}$$

$$\overleftarrow{h_t} = \overleftarrow{GRU}(x_t, \overleftarrow{h_{t+1}}) \tag{4.16}$$

$$y_t = \sigma(\overrightarrow{W_o}\overrightarrow{h_t} + \overleftarrow{W_o}\overleftarrow{h_t} + b_o) \tag{4.17}$$

donde:

- $\overrightarrow{GRU}$ y $\overleftarrow{GRU}$ son las capas recurrentes que procesan la secuencia en sentido directo e inverso
- $\overrightarrow{W_o}$ y $\overleftarrow{W_o}$ son matrices de pesos que transforman $\overrightarrow{h_t}$ y $\overleftarrow{h_t}$ respectivamente
- b_o es el sesgo aplicado como entrada a la función de activación

Por último, si bien la arquitectura presentada considera una salida temporal, existen diversas variantes que pueden adaptarse según las necesidades del problema a resolver. La capa recurrente oculta de Bi-GRU puede integrarse con capas densas completamente conectadas o combinarse con otros módulos, tanto de aprendizaje automático como profundo, permitiendo la creación de modelos híbridos más sofisticados. Gracias a esta flexibilidad, Bi-GRU se ha aplicado en diversos dominios además de la predicción de series temporales, incluyendo el procesamiento de lenguaje natural, el reconocimiento de gestos, el análisis de sentimientos, y la bioinformática, entre muchos otros, consolidándose como una herramienta versátil en el modelado de secuencias.

4.3.5. Fundamentos de la selección

Dado que el proyecto aborda la predicción de series temporales, la selección del método se centró en arquitecturas especializadas en datos secuenciales. Según la revisión sistemática, las RNN son el estándar en estas tareas, ya que preservan el contexto temporal, mejorando la capacidad predictiva.

Dentro de las variantes de RNN, se optó por GRU en lugar de LSTM. La elección de GRU se basa en su arquitectura más simple, lo que reduce los tiempos de entrenamiento sin afectar significativamente la capacidad del modelo para capturar patrones temporales complejos.

Finalmente, se optó por GRU bidireccional debido a sus ventajas específicas para la predicción de tiempos de viaje. Al procesar la información en ambas direcciones temporales, Bi-GRU mejora la captura de dependencias a lo largo del tiempo. En comparación con los modelos GRU simples, esta arquitectura no solo aprovecha los valores pasados, sino también los futuros, lo que le permite modelar las dependencias temporales de manera más efectiva, mejorando así el rendimiento general del modelo.

De esta manera, Bi-GRU se presentó como la mejor alternativa para escoger un método de aprendizaje profundo.

4.3.6. Consideraciones de Bi-GRU

Si bien las redes neuronales profundas (incluida Bi-GRU) ofrecen numerosas ventajas, también enfrentan desafíos como la maldición de la dimensionalidad, cuando el número de características o ejemplos de entrenamiento es elevado, lo que puede disminuir el rendimiento. Para mitigar este efecto, se pueden emplear técnicas de regularización y reducción de dimensionalidad, mejorando la eficiencia y la generalización del modelo (Awad y Khanna, 2015).

Las redes recurrentes son eficaces para modelar dependencias temporales, pero tienen limitaciones al capturar dependencias espaciales. Las correlaciones espaciales complejas no se pueden transformar en datos secuenciales de baja dimensionalidad (Wang y cols., 2023). Para superar esto, se han explorado enfoques que combinan RNN con *Convolutional Neural Networks* o *Graph Convolutional Networks*, modelos más adecuados para el manejo de datos espaciales.

Aunque las RNN, son eficaces para capturar dependencias temporales, no ajustan dinámicamente la importancia de los estados ocultos en la secuencia, lo que limita su capacidad para enfocarse en los pasos más relevantes. Los mecanismos de atención abordan esta limitación, permitiendo una ponderación flexible de los estados ocultos, lo que mejora la captura de dependencias temporales complejas y relaciones sutiles (Chughtai, Haq, y Muneeb, 2022).

Además, las RNN tienen una limitación inherente en cuanto a que procesan los datos de manera secuencial, lo que restringe los cálculos paralelos y puede reducir la eficiencia computacional (Alkilane y cols., 2023).

En el próximo capítulo se profundizará en la solución planteada, detallando aspectos como la construcción y entrenamiento de los modelos, las metodologías empleadas, las tecnologías utilizadas, así como el ambiente de ejecución.

Capítulo 5

Solución Propuesta

En el capítulo anterior se desarrollaron los métodos ARIMA, XGBoost y Bi-GRU, seleccionados como enfoques predictivos. En este capítulo, se describe la solución propuesta para implementar estos modelos, detallando el flujo de trabajo del sistema, la construcción y entrenamiento de los modelos, las tecnologías empleadas y el ambiente de ejecución, así como la gestión del proyecto.

Para cada variante de la línea de ómnibus 117, se desarrollan modelos independientes adaptados a las características específicas de cada trayecto.

5.1. Definición del Problema

En esta fase se pretende desarrollar modelos capaces de predecir el objetivo definido, tomando como referencia los datos históricos. Para eso se vale de una discretización del tiempo, de manera de contar con las marcas temporales de cuando el ómnibus pasa por cada una de las n paradas que componen el viaje de una variante.

En la Figura 5.1 se muestra esta discretización, donde se denota como t_k, t_{n-2} y t_{n-1} las marcas de tiempo de las parada k, $n-2$ y $n-1$ respectivamente, y como t_n la marca de tiempo de llegada a la última parada n (el destino).

Figura 5.1: Abordaje del Problema. Fuente: elaboración propia.

Esto da las bases para definir el problema de predicción. Dada la marca de tiempo de un ómnibus –de variante v– al arribar a una parada k, el objetivo es **predecir el tiempo restante hasta llegar a la última parada** (denotado

tt_k^v), en función de las características de entrada ($\vec{x_k}$). Matemáticamente, el problema se puede expresar mediante la ecuación 5.1, donde ϵ representa el margen de error o incertidumbre asociado con la predicción.

$$tt_k^v = f^v(\vec{x_k}) + \epsilon = (t_n - t_k), \quad \text{con } 1 \leq k \leq n-1, \quad n > 1 \tag{5.1}$$

Las características representadas en el vector de entrada $\vec{x_k}$ son todas las detalladas en el Capítulo 3, a excepción de `id_recorrido` –notar que el identificador de recorrido es único y no se repite en otros recorridos, por lo que no tiene sentido incorporarlo como entrada–, `variante` (ya está contemplada en la personalización de la función), y `travel_time` (ya que es la variable a predecir).

El vector $\vec{x}$ está compuesto entonces de la siguiente manera:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \end{bmatrix} = \begin{bmatrix} coche \\ parada \\ fecha\_pasada \\ tipo\_día \\ día \\ dia\_numero \\ mes \\ año \\ hora \\ minuto \\ segundo \end{bmatrix}$$

Si bien la función recibe todos los campos (características) del vector $\vec{x}$, esto no implica necesariamente que en cada modelo se utilicen todos ellos, como se explicará más adelante en este mismo capítulo. Debe entenderse la función anterior como una primera formulación del problema de predicción, expresada en forma genérica.

Una vez planteado el objetivo de predicción, en las siguientes secciones se detallarán los aspectos más relevantes que formaron parte de la solución propuesta.

5.2. Flujo de Trabajo del Sistema

El desarrollo de la solución se estructuró en tres grandes etapas:

1. **Preprocesamiento de datos**
2. **Modelado**
3. **Evaluación de desempeño**

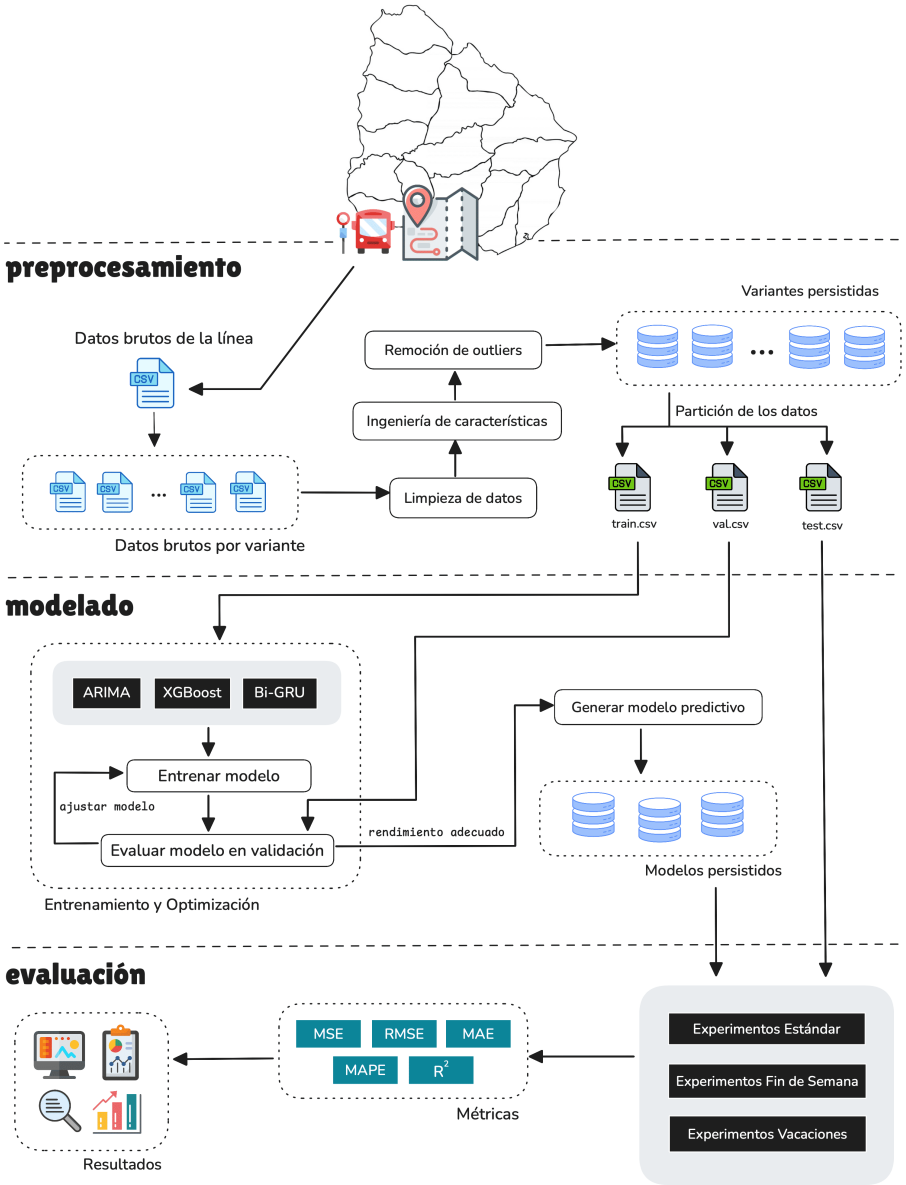

Figura 5.2: Flujo de Trabajo del Sistema. Fuente: elaboración propia.

La Figura 5.2 muestra una vista general de todo el proceso de trabajo, donde se representan las tres grandes etapas, sus componentes internos, y la conexión entre las diferentes partes.

La etapa del **preprocesamiento** ya fue detallada exhaustivamente en el Capítulo 3, sin embargo lo que resta comentar en este apartado es que el formato de datos utilizado para la persistencia a lo largo de todo el flujo, es *Comma Separated Values*, más conocido como CSV, por su sigla.

La etapa de **modelado** consistió en la construcción y validación de los modelos encargados de predecir el tiempo de viaje planteado.

A partir del conjunto de entrenamiento (`train.csv`), se entrenan los modelos utilizando técnicas específicas para cada enfoque. Luego de entrenados, éstos son validados utilizando el conjunto de validación (`val.csv`), con el fin de evaluar su capacidad de generalización. En esta etapa, se analizan sus métricas de rendimiento para determinar si la precisión alcanzada es satisfactoria; en caso de no serla, se realizan ajustes en la configuración del modelo y se vuelve a entrenar. Este proceso se repite hasta que los resultados sean considerados adecuados según las métricas establecidas. Tras la validación, los modelos generados fueron persistidos para su uso posterior en la experimentación y evaluación con diferentes escenarios.

La etapa de **evaluación de desempeño** es abordada en el Capítulo 6, donde se detallan los distintos escenarios utilizados para analizar el comportamiento de los modelos, así como sus métricas de evaluación.

A continuación se describen las tecnologías utilizadas para llevar a cabo la solución.

5.3. Tecnologías Utilizadas

La implementación de la solución se lleva a cabo utilizando un conjunto de herramientas y bibliotecas de código abierto ampliamente utilizadas en ciencia de datos y aprendizaje automático.

El lenguaje de programación principal utilizado es `Python` (`3.12.3`). Para asegurar un entorno controlado, se optó por usar un entorno virtual (`venv`), lo cual facilitó la configuración, garantizó la reproducibilidad de los resultados y evitó posibles problemas de compatibilidad entre versiones y proyectos. Para el procesamiento de datos se utilizaron las bibliotecas `Pandas`, versión `2.2.2`, y `NumPy`, versión `2.0.1`. Para la visualización de los resultados, se utilizaron `Matplotlib` (versión `3.9.1`) y `Seaborn` (versión `0.13.2`).

Para ARIMA, se seleccionó la librería de Python `Statsmodels`, versión `0.14.2`. Para XGBoost, se utilizó un *framework* específico que lo implementa, el cual está disponible como librería para Python. La versión utilizada de `XGBoost` es la `2.1.0`. Para Bi-GRU se optó por `Keras`, por disponer una menor curva de aprendizaje.

Para la etapa de análisis exploratorio y depuración de datos, se utilizó `Jupyter Notebook`, versión `7.2.1` por su versatilidad, ya que permite combinar código, texto explicativo, visualizaciones y resultados en un mismo entorno.

Por último, tanto para la transformación de datos en variables categóricas o discretas, como para las métricas de evaluación de performance, se utiliza la librería `Scikit-learn`, versión `1.5.2`.

5.4. Implementación de Modelos

En esta sección se detallan los tres tipos de enfoques utilizados para la predicción del tiempo de viaje, así como el abordaje concreto que fue llevado en cada uno de ellos, mostrando entre otras cosas los mejores hiperparámetros encontrados. El código fuente del proyecto está publicado en el repositorio GitLab de Facultad de Ingeniería, para acceder a él ver el Anexo B.

5.4.1. ARIMA

Como se mencionó en el capítulo anterior, antes de la fase de implementación se analizó la estacionariedad de la serie de datos de cada variante aplicando el test ADF, y se observó que los datos de algunas paradas presentaban un comportamiento estacionario, mientras que los de otras no.

Como resultado de esto se planteó utilizar ARIMA, y se tomó la decisión de modelar cada parada por separado, esto es, generar un modelo de predicción por cada parada de la variante. Ensayos previos con un único modelo resultaron muy poco precisos, como para sugerir captar los patrones de cada parada por separado.

El método ARIMA está específicamente diseñado para trabajar con series temporales, esto implica que para construir el modelo se necesitan únicamente los datos de la serie temporal (columna `travel_time`), y no el vector completo $\vec{x}$ de características. En otras palabras, el entrenamiento se basa en tomar una serie de marcas de tiempo (`fecha_pasada`) de la variable a predecir (`travel_time`), y a partir de esos datos construir un modelo que pueda generalizar a nuevas marcas de tiempo.

Búsqueda de Hiperparámetros

La búsqueda de hiperparámetros es un proceso crucial en el desarrollo de modelos de predicción, en el que se busca identificar los valores más adecuados para los hiperparámetros. Esta búsqueda se realiza mediante un abanico grande de técnicas que exploran diferentes combinaciones de valores, como ser búsqueda en cuadrícula, búsqueda aleatoria, búsqueda mediante hiperbanda, etc.

El rendimiento del modelo durante la búsqueda de hiperparámetros se evalúa utilizando una métrica de validación, como MAPE (*Mean Absolute Percentage Error*), MSE (*Mean Squared Error*), etc[1]. El objetivo es encontrar la combinación de hiperparámetros que optimice el desempeño del modelo en el conjunto de validación.

Para el modelo ARIMA, se utilizó una búsqueda en cuadrícula para optimizar los hiperparámetros `p`, `d` y `q`. La búsqueda en cuadrícula consiste en probar todas las combinaciones posibles de valores dentro de los rangos predefinidos para cada hiperparámetro. En este caso, los rangos fueron:

[1] La forma de cálculo de estas métricas, así como su significado, se explica en el capítulo siguiente.

- `p` $\in \{0, 1, 2\}$
- `d` $\in \{0, 1\}$
- `q` $\in \{0, 1, 2\}$

Para cada combinación de hiperparámetros, se ajustó un modelo ARIMA y su desempeño fue evaluado utilizando MAPE sobre el conjunto de validación `val.csv`.

El ajuste de los parámetros internos del modelo ARIMA (los coeficientes de la ecuación 4.3) se realizó utilizando el método MLE (*Maximum Likelihood Estimation*), que es el criterio por defecto en la implementación de *Statsmodels*.

Hiperparámetros Seleccionados

Al finalizar esta búsqueda para las $n - 1$ paradas (no se predice en la última parada, recordar la ecuación 5.1), se seleccionan automáticamente aquellas combinaciones de hiperparámetros que resultaron con menor MAPE respectivamente. Los resultados se muestran en la Tabla 5.1.

5.4.2. XGBoost

A diferencia de ARIMA, que modela la serie temporal como una secuencia de valores en el tiempo, XGBoost opera sobre datos en formato tabular, lo que permite aprovechar un conjunto de características de entrada para realizar la predicción. Este enfoque facilita la incorporación de información adicional más allá del historial de la serie, como ser información del coche, de la parada, u otros posibles factores que influyan en el tiempo de viaje.

Para este método, en lugar de generar un modelo independiente por parada, se construye uno que contempla todas las paradas. Este enfoque permite capturar posibles relaciones entre las diferentes paradas de una misma variante, aprovechando la información global del recorrido.

Además, con el fin de evaluar el impacto de distintas características en la predicción de este modelo, se desarrollaron tres versiones para cada variante. XGBoost fue elegido para este análisis debido a su rapidez en el entrenamiento, lo que permitió explorar diversas combinaciones de características de manera eficiente. Los modelos se detallan a continuación.

- **XGBoost_min**: Utiliza únicamente la característica `parada`.
- **XGBoost_med**: Utiliza un conjunto reducido de características: `parada`, `tipo_dia` y `hora`.
- **XGBoost_max**: Utiliza un conjunto amplio de características: `parada`, `tipo_dia`, `hora`, `mes`, `dia`, `año` y `coche`.

Las características utilizadas en los modelos **XGBoost_min** y **XGBoost_med** no fueron seleccionadas arbitrariamente. Tras entrenar **XGBoost_max**, se analizó la importancia de cada característica mediante su peso en el modelo. A partir

Var 634				Var 635				Var 2778				Var 2779			
Pda	**p**	**d**	**q**	**Pda**	**p**	**d**	**q**	**Pda**	**p**	**d**	**q**	**Pda**	**p**	**d**	**q**
1	2	0	2	1	0	0	2	1	2	1	2	1	2	1	2
2	1	1	2	2	2	1	2	2	2	1	2	2	2	1	2
3	2	0	2	3	1	1	2	3	2	1	2	3	2	1	2
4	2	0	0	4	2	1	2	4	1	0	1	4	2	1	2
5	2	0	0	5	1	1	2	5	1	0	1	5	2	1	2
6	2	0	0	6	2	0	2	6	1	0	1	6	2	1	2
7	2	0	0	7	2	0	2	7	1	0	1	7	2	1	2
8	2	0	0	8	1	1	2	8	1	0	1	8	2	1	2
9	2	0	0	9	2	1	2	9	1	0	1	9	2	1	2
10	0	0	2	10	2	1	2	10	1	0	2	10	2	1	2
11	0	0	2	11	2	1	2	11	0	0	1	11	2	1	2
12	2	1	2	12	1	1	2	12	1	0	2	12	2	1	2
13	2	1	2	13	2	1	2	13	1	0	2	13	0	1	0
14	2	1	2	14	1	1	2	14	1	0	1	14	2	1	0
15	2	1	2	15	1	1	2	15	1	0	1	15	2	1	2
16	2	0	0	16	2	1	1	16	1	0	1	16	0	1	1
17	1	1	1	17	1	1	2	17	2	1	2	17	1	1	0
18	2	1	1	18	1	1	2	18	2	0	2	18	0	1	0
19	0	0	0	19	1	1	2	19	0	0	0	19	1	1	0
20	2	1	0	20	2	1	2	20	1	0	2	20	1	1	0
21	0	1	1	21	2	1	0	-	-	-	-	21	1	1	0
-	-	-	-	22	1	0	0	-	-	-	-	22	1	1	0
-	-	-	-	23	1	1	2	-	-	-	-	23	1	1	0
-	-	-	-	-	-	-	-	-	-	-	-	24	1	1	0
-	-	-	-	-	-	-	-	-	-	-	-	25	2	1	2
-	-	-	-	-	-	-	-	-	-	-	-	26	0	1	1
-	-	-	-	-	-	-	-	-	-	-	-	27	1	1	0
-	-	-	-	-	-	-	-	-	-	-	-	28	2	1	1
-	-	-	-	-	-	-	-	-	-	-	-	29	0	1	1
-	-	-	-	-	-	-	-	-	-	-	-	30	2	1	2
-	-	-	-	-	-	-	-	-	-	-	-	31	0	1	1
-	-	-	-	-	-	-	-	-	-	-	-	32	0	1	0
-	-	-	-	-	-	-	-	-	-	-	-	33	0	1	0
-		-	-	-	-	-	-	-	-	-	-	34	0	1	0

Tabla 5.1: Mejores hiperparámetros de ARIMA. Var = Variante, Pda = Parada.

de este análisis, se identificaron las más influyentes, las cuales fueron utilizadas en las versiones reducidas del modelo.

Transformación de Características

Para garantizar el correcto funcionamiento del modelo, fue necesario transformar las características a formato numérico y ajustar su escala para evitar que algunas variables dominen el modelo. Además, cuando una variable categórica representa solo un identificador (como el id del coche), es necesario aplicar una codificación que refleje su relación con la variable objetivo, en lugar de interpretarla como un valor numérico con significado propio.

- ***One-Hot Encoding***: Método utilizado para transformar variables categóricas en una representación numérica. Cada categoría única se convierte en una nueva columna binaria (0 o 1). Se aplicó a las características `tipo_dia`, `dia` y `año`.
- **Codificación Cíclica**: Para variables que tienen un comportamiento cíclico (como los meses del año), se utilizó una transformación trigonométrica con *seno* y *coseno*, lo que permite representar mejor su naturaleza periódica. Esta técnica se aplicó a la característica `mes`.
- **Normalización Min-Max**: Se utiliza para escalar los valores de ciertas variables a un rango entre 0 y 1, evitando que aquellas con valores más grandes dominen el modelo. Se aplicó a las características `hora` y `parada`.
- ***Target Encoding***: Técnica utilizada para transformar variables categóricas en valores numéricos basados en la media de la variable objetivo. En este caso, se aplicó a la variable `coche`, asignando a cada valor de `coche` la media del tiempo de viaje (`travel_time`) correspondiente. Esta elección se debe a que *One-Hot Encoding* habría generado un número excesivo de columnas, aumentando la dimensionalidad sin aportar información adicional relevante.

Búsqueda de Hiperparámetros

Para ajustar los hiperparámetros de XGBoost, se utilizó una búsqueda aleatoria (`RandomizedSearchCV`), que permite explorar de manera eficiente un espacio de hiperparámetros sin necesidad de evaluar todas las combinaciones posibles, reduciendo así el tiempo computacional requerido. Se definieron los siguientes rangos:

- `n_estimators` $\in \{50, 1000\}$
- `learning_rate` $\in [0.01, 0.30]$
- `max_depth` $\in \{3, 7\}$
- `subsample` $\in [0.5, 1.0]$
- `colsample_bytree` $\in [0.3, 1.0]$

Para la búsqueda, se ejecutaron 50 iteraciones, `n_iter = 50`, donde cada iteración seleccionó una combinación aleatoria de hiperparámetros dentro de los rangos establecidos.

Para la optimización de hiperparámetros en `RandomizedSearchCV`, se utilizó la versión negativa del MSE (`neg_mean_squared_error`), ya que por convención, las herramientas de optimización en `Scikit-learn` maximizan el valor de la métrica, y como el MSE es una métrica de minimización, se toma su negativo para adaptarse a esta convención.

El proceso de validación del modelo se realizó con la técnica de división predefinida (*Predefined Split*), que permite mantener una separación fija entre los conjuntos de entrenamiento y validación, asegurando que el modelo se evalúe de manera adecuada sin mezclar los datos.

En cuanto a la capacidad de procesamiento, se configuró `n_jobs = -1` para utilizar todos los núcleos del procesador y acelerar el proceso.

Por último, se estableció `random_state = 42`, que corresponde a la semilla para la generación de números aleatorios, con el fin de garantizar la reproducibilidad de los resultados.

Hiperparámetros Seleccionados

Al finalizar la búsqueda de hiperparámetros, se seleccionaron automáticamente aquellos valores que ofrecieron el menor MSE en el conjunto de validación. Los hiperparámetros seleccionados se detallan en la Tabla 5.2.

		Hiperparámetros				
		n_estimators	**learning_rate**	**max_depth**	**subsample**	**colsample_bytree**
Variante 634	**min**	326	0.07370173320348283	6	0.8087407548138583	0.8827098485602951
	med	193	0.08738248831454667	6	0.9086111001006079	0.9141375473666866
	max	826	0.2996896099223678	4	0.5079831261101071	0.9642198760773333
Variante 635	**min**	659	0.12033493981577596	6	0.8167648553804474	0.8097514440283016
	med	293	0.06990213464750791	5	0.7962072844310213	0.8496231729751094
	max	610	0.012119891565915222	3	0.7623873301291946	0.7281572123417965
Variante 2778	**min**	456	0.038623034947123394	5	0.8344206263318037	0.6146154718967423
	med	201	0.26399834267149175	6	0.702254063561095	0.7902504809809399
	max	460	0.01208563915935721	6	0.7087055015743895	0.9025114082794403
Variante 2779	**min**	731	0.058366386176201324	4	0.9040601897822085	0.5028260170396376
	med	682	0.09163967481539059	5	0.9090073829612466	0.9347799090820277
	max	460	0.01208563915935721	6	0.7087055015743895	0.9025114082794403

Tabla 5.2: Mejores Hiperparámetros de XGBoost por variante y por modelo.

5.4.3. Bi-GRU

Las redes neuronales profundas tienen la ventaja de poder procesar una gran variedad de estructuras de datos, a diferencia de modelos como XGBoost, que trabajan exclusivamente con datos tabulares. En este caso, la entrada del

modelo Bi-GRU no consiste en una simple matriz de características, sino en secuencias de viajes organizadas de manera que el modelo pueda captar patrones temporales.

Antes de entrenar el modelo, fue necesario realizar un preprocesamiento de los datos, asegurando que la información fuera representada en un formato adecuado para ser consumida por la red neuronal. Este preprocesamiento incluyó: transformación de características, transformación de variable objetivo, y construcción de secuencias de entrada.

Transformación de características

Se utilizaron todas las características a excepción de `fecha_pasada`, por considerarse redundante, ya que se tomó el desglose de la misma en diferentes campos. Las características `hora`, `minuto` y `segundo` fueron transformadas usando una codificación cíclica. Se aplicó *One-Hot Encoding* a las variables categóricas, y las características numéricas fueron escaladas usando `StandardScaler`.

Transformación de variable objetivo

Dado que los tiempos de viaje varían mucho a lo largo del recorrido, con tiempos largos en las primeras paradas (por ejemplo, `30` minutos) y tiempos muy pequeños en las últimas (por ejemplo, `0.6` minutos), se aplicó una transformación logarítmica a la variable objetivo. Para evitar problemas con los valores muy pequeños (que podrían generar errores al aplicar el logaritmo), se sumó un pequeño valor $\epsilon = 1 \times 10^{-5}$ a los tiempos antes de aplicar la transformación. Esto hace que las diferencias entre los tiempos de viaje sean más comparables.

Construcción de Secuencias de Entrada

Las redes neuronales recurrentes requieren que los datos de entrada se organicen en secuencias temporales para capturar patrones en la evolución de los tiempos de viaje. En este caso, cada muestra de entrada representa un recorrido y contiene información de todas sus paradas, excepto la última.

A diferencia de los modelos unidireccionales, donde la información fluye únicamente desde el inicio del viaje hacia adelante, la arquitectura bidireccional utilizada permite que el modelo procese la secuencia en ambas direcciones. Esto significa que la red no solo aprende cómo los tiempos de viaje evolucionan a medida que avanza el recorrido, sino que también incorpora información desde las etapas finales hacia las iniciales. Este enfoque mejora la capacidad del modelo para identificar patrones en la variabilidad de los tiempos de viaje, considerando la influencia de factores tanto pasados como futuros dentro de la secuencia.

Para representar los recorridos de manera adecuada, las secuencias de entrada se estructuran en tres dimensiones:

(viajes, paradas, características)

donde:

- **Viajes**: Representa la cantidad de recorridos procesados en el conjunto de datos.
- **Paradas**: Cada secuencia incluye todas las paradas del recorrido (excepto la última).
- **Características**: Representa la cantidad de variables utilizadas en el modelo tras el preprocesamiento.

A nivel computacional, estas secuencias no se almacenan como simples listas de valores, sino como tensores. Un tensor es una estructura multidimensional que extiende el concepto de matrices y es fundamental para representar datos en redes neuronales profundas.

Para optimizar el uso de recursos computacionales y evitar el recálculo innecesario de estas estructuras, los tensores generados fueron persistidos en archivos con formato `.npy`, permitiendo su reutilización sin necesidad de repetir el preprocesamiento.

Arquitectura del Modelo

El modelo implementado utiliza una estructura basada en capas tal como se muestra en la Figura 5.3, que se puede describir de la siguiente manera:

- **Capa de Entrada**: Recibe las secuencias de entrada (viajes, paradas, características).
- **Capa Bidireccional GRU**: Contiene una cantidad de neuronas a configurar (denotada `units`), y devuelve una secuencia completa de estados ocultos para cada paso de tiempo. Dado que los datos de entrada –especialmente después de la transformación de tiempo– pueden incluir valores negativos, se escogió la función de activación ELU (*Exponential Linear Unit*), ya que ésta permite que el modelo maneje esos valores de manera eficiente.
- **Capas Densas**: Se incluyen dos capas densas (también llamadas totalmente conectadas o *Multilayer Perceptron*, recordar Figura 4.4), con 128 y 64 neuronas respectivamente, y emplean también ELU como activación.
- **Capa de Salida**: Una neurona densa sin activación para predecir el tiempo de viaje final.

Búsqueda de Hiperparámetros

La búsqueda de hiperparámetros se realizó utilizando `Ray Tune` con la estrategia de `HyperBandScheduler`, un método basado en hiperbanda que asigna

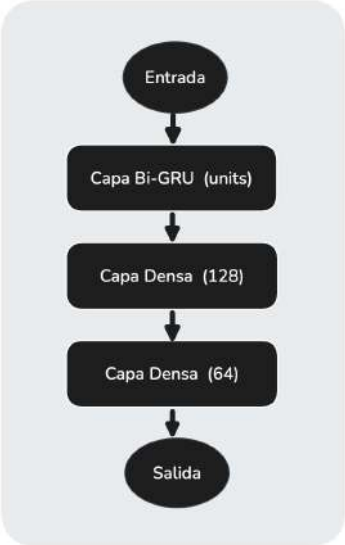

Figura 5.3: Arquitectura del modelo Bi-GRU. Fuente: elaboración propia.

dinámicamente recursos a los experimentos, enfocándose en las configuraciones de hiperparámetros más prometedoras y descartando aquellas con un rendimiento inferior. Las principales configuraciones fueron:

- `num_samples=30`: Cantidad de experimentos evaluados (se probaron 30 combinaciones diferentes de hiperparámetros).
- `max_t=50`: Cada experimento podía entrenarse hasta 50 épocas antes de ser descartado o promovido a la siguiente ronda.
- `reduction_factor=3`: En cada iteración, aproximadamente dos tercios de las configuraciones de peor desempeño se descartan, mientras que las mejores se conservan para la siguiente ronda de experimentos.
- `max_concurrent_trials=4`: Se ejecutaron hasta 4 experimentos simultáneamente, aprovechando múltiples núcleos de CPU.

En cuanto a los hiperparámetros para configurar en la búsqueda, se definieron tres clave: `units_layer` (unidades de capa Bi-GRU), `batch_size` (tamaño del lote) y `lr` (tasa de aprendizaje).

El número de unidades `units_layer` define cuántas neuronas habrá en la capa Bi-GRU, influyendo en la capacidad del modelo para aprender patrones complejos.

El tamaño del lote (`batch_size`) es un parámetro clave en el entrenamiento de modelos de aprendizaje profundo. En lugar de actualizar los pesos del modelo

después de cada observación, el entrenamiento se realiza en pequeños grupos de datos llamados lotes.

La tasa de aprendizaje (`lr`) es un hiperparámetro que controla la magnitud de los ajustes realizados a los pesos del modelo durante el entrenamiento.

Los rangos de los hiperparámetros fueron definidos según:

- `units_layer` $\in \{64, 128, 256\}$
- `batch_size` $\in \{16, 32, 64, 128\}$
- `learning_rate` $\in [1 \times 10^{-4}, 1 \times 10^{-2}]$

Optimización del Entrenamiento

Tras la búsqueda de hiperparámetros, el mejor modelo se seleccionó en función de la pérdida en el conjunto de validación (`val_loss`), utilizando la métrica MSE. El entrenamiento final se realizó con el optimizador `RMSprop`, el cual ajusta la tasa de aprendizaje de forma adaptativa, evitando oscilaciones y mejorando la convergencia, permitiendo hasta 200 épocas (`epochs=200`).

Para evitar el sobreajuste y mejorar la generalización, se implementaron las siguientes estrategias:

- **Early Stopping**: Detención temprana cuando la pérdida de validación no mejoraba después de 15 épocas (`patience=15`), restaurando los mejores pesos.
- **Reducción de la tasa de aprendizaje**: Disminución adaptativa de la tasa de aprendizaje si la mejora en la pérdida de validación se estancaba, permitiendo ajustes más finos en las últimas etapas del entrenamiento.

Estas estrategias ayudaron a mejorar la estabilidad del entrenamiento y a evitar el sobreajuste, logrando modelos que generalizan con mucha precisión. Sin embargo, el proceso de entrenamiento fue computacionalmente costoso como se mostrará en la siguiente sección.

Hiperparámetros Seleccionados

Tras haber construido los mejores modelos por cada variante, los mismos fueron persistidos utilizando el formato `.keras`, para su poterior carga. Los hiperparámetros finales para cada variante se muestran en la Tabla 5.3.

5.5. Ambiente y Tiempos de Ejecución

Las construcciones de todos los modelos detallados en la Sección 5.4, se llevaron a cabo en un computador Mac mini M4 Pro de 12 núcleos de CPU y 48 GB de RAM, dedicado exclusivamente a estas ejecuciones. Los tiempos de ejecución expresados en minutos se encuentran en la Tabla 5.4. Como se

Variante	units_layer	batch_size	lr
634	256	64	0.0034720593294174564
635	64	128	0.003309112137112071
2778	128	256	0.0009871660748163417
2779	128	32	0.0027472506889124285

Tabla 5.3: Mejores hiperparámetros de Bi-GRU.

Modelo	Variante 634	Variante 635	Variante 2778	Variante 2779
ARIMA	6.05	8.45	1.65	2.73
XGBoost_min	0.45	0.55	0.08	0.13
XGBoost_med	0.50	0.67	0.08	0.17
XGBoost_max	0.85	1.13	0.15	0.25
Bi-GRU	359.55	285.88	87.28	52.03

Tabla 5.4: Tiempos de ejecución en minutos para cada modelo y variante.

observa en los resultados, XGBoost destaca por sus tiempos de ejecución muy bajos, con un rango de entre 0.08 y 1.13 minutos, dependiendo de la variante. Este comportamiento es esperado dado que XGBoost es conocido por entrenar modelos rápidamente sin sacrificar precisión.

En comparación con XGBoost, ARIMA muestra tiempos de ejecución moderados, con un rango entre 1.65 minutos (para la variante 2778) y 8.45 minutos (para la variante 635). Estos tiempos son relativamente más altos que los de XGBoost, aunque no excesivos, considerando que ARIMA es un modelo tradicional de series temporales que, por su naturaleza, requiere un tiempo mayor para estimar los parámetros y modelar las dependencias temporales.

En cuanto a Bi-GRU, los tiempos de ejecución son extremadamente más altos, variando entre 52.03 y 359.55 minutos dependiendo de la variante. Esto se debe a la mayor complejidad de este modelo, que requiere un mayor número de iteraciones y procesos computacionales para entrenar adecuadamente. Cabe señalar que, aunque se utilizaron 30 experimentos para el entrenamiento y 50 épocas por cada uno de ellos, una configuración menor podría haber obtenido un desempeño similar sin que se excedieran tanto en los tiempos de ejecución. De todas formas, como se mencionó en el capítulo anterior, este tipo de redes neuronales recurrentes generalmente requiere más tiempo de entrenamiento debido a su complejidad estructural.

5.6. Gestión del Proyecto

Por último, para la organización del trabajo, se adoptaron elementos de metodologías ágiles, incluyendo reuniones periódicas, gestión de tareas con `Trello` y uso de un modelo de ramas en `GitHub` basado en `GitFlow`.

Como herramientas de comunicación se utilizaron `Slack` y `Zoom`, y la herramienta principal para el desarrollo del código fue `Visual Studio Code`.

Por más información sobre la gestión del proyecto, consultar el Anexo C.

En el siguiente capítulo se presentan los resultados obtenidos y un análisis comparativo del desempeño de los modelos propuestos, evaluando su precisión y capacidad de generalización.

Capítulo 6

Evaluación de Desempeño

Como se explica en el Capítulo 2.3, después de completar el proceso de limpieza y transformación, se divide el *dataset* en 70 %, 15 % y 15 % para entrenamiento, validación y *testing*, respectivamente.

Para evaluar el desempeño de los modelos se utiliza el conjunto de *testing* y se llevan a cabo diferentes experimentos a detallar más adelante. Estos experimentos se evalúan en base a determinadas métricas que se presentan a continuación.

6.1. Métricas Utilizadas

Del análisis de los artículos que conforman el Estado del Arte, se identifica que las métricas de desempeño más frecuentes son MAE, RMSE, MAPE y R^2. Estas métricas permiten analizar el error (diferencia entre el valor real y el predicho por el modelo) desde diferentes enfoques, ya que cada una evalúa el rendimiento del modelo desde una perspectiva distinta.

A continuación, se describen cada una de ellas (Chughtai, Haq, y Muneeb, 2022; Nicolás Arrioja, 2021).

El **Error Absoluto Medio (MAE)** es una métrica que mide la diferencia entre los valores predichos por el modelo y los valores reales u observados. Se calcula tomando el valor absoluto de las diferencias entre cada valor real y su correspondiente predicción, y luego promediando estos valores absolutos. Se considera el valor absoluto para que un error con valor positivo no cancele a un error con valor negativo.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6.1}$$

La unidad del resultado es la misma que la de los valores reales. Un valor pequeño de MAE indica mayor precisión del modelo.

Para entender el **RMSE** primero se tiene que presentar el **MSE**.

El **Error Medio Cuadrado (MSE)** indica qué tan cercana es la predicción al valor real. Se calcula de la siguiente manera:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.2}$$

Observar que en este caso se toma la diferencia al cuadrado entre los valores reales y predichos. Esto implica que cuanto más pequeña sea la diferencia (es decir, cuando los errores sean cercanos a cero), mejor será la predicción. Por el contrario, los errores grandes se penalizan significativamente, ya que al elevarlos al cuadrado, su impacto en el MSE aumenta considerablemente. En resumen, un MSE cercano a cero indica que las predicciones están muy cerca de los valores reales, lo que refleja un buen desempeño del modelo, mientras que un MSE alto indica que el modelo comete errores significativos y su rendimiento es pobre.

La **Raiz del Error Cuadrado Medio (RMSE)** es simplemente la raíz del MSE. Como el resultado del MSE está en unidades cuadradas, para poder interpretar mejor los resultados, se calcula la raíz cuadrada. De esta manera, el resultado está en la misma unidad que los valores reales.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.3}$$

El **Error Porcentual Absoluto Medio (MAPE)** es una métrica de precisión. Por cada observación se obtiene el porcentaje de error y luego se promedian todos los resultados.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{6.4}$$

La unidad es un porcentaje, aunque podría dar valores no definidos, infinitos o muy cercanos a cero.

En todos los casos (MAE, MAPE, MSE, RMSE) se toma el promedio porque interesa conocer el comportamiento del error de todas las observaciones.

Por último, la métrica **Coeficiente de Determinación (R^2)** indica qué tan bien ajusta el modelo a las observaciones reales. Esta métrica no tiene unidad ya que es una proporción. Un valor cercano a 1 indica que el modelo predice muy bien, mientras que valores cercanos a 0 indican que el modelo no es capaz de explicar la variación en los datos, es decir, el modelo tiene un ajuste muy pobre y las predicciones no se acercan a los valores reales.

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}} \tag{6.5}$$

$$\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

$$\mathrm{SS}_{\mathrm{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2,$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i.$$

Donde:

- y_i: Representa el valor real del i-ésimo dato. En el contexto de este estudio, corresponde al *tiempo de viaje real* asociado a la i-ésima entrada del *dataset* analizado. Cabe recordar que cada entrada del *dataset* incluye la información relativa a una parada de bus específica dentro de un recorrido particular.
- $\hat{y}_i$: Representa el valor predicho del i-ésimo dato. En nuestro caso, es el *tiempo de viaje predicho* para la i-ésima entrada según el modelo utilizado.
- n: Denota el número total de observaciones. En este caso, corresponde al número total de entradas en el *dataset*.
- $\mathrm{SS}_{\mathrm{res}}$: Corresponde a la *suma de los cuadrados residuales*. En nuestro caso, mide la *diferencia cuadrada* entre los tiempos de viaje reales y los predichos, y refleja qué tan bien el modelo ajusta los datos observados.
- $\mathrm{SS}_{\mathrm{tot}}$: Representa la *suma total de los cuadrados*. En nuestro caso, refleja la *variabilidad total* de los tiempos de viaje reales en relación con la media de todos los tiempos de viaje.
- $\bar{y}$: Indica la *media de los valores reales*. En nuestro caso, es el *tiempo de viaje promedio* de todas las entradas del *dataset*, utilizado como referencia para calcular la variabilidad de los datos.

La Tabla 6.1 presenta un resumen con las principales características de estas métricas.

6.2. Resultados y análisis comparativo de los modelos

Para evaluar la performance de los modelos se llevan a cabo tres experimentos sobre el conjunto de *testing*. El primer experimento consiste en evaluar el

Métrica	**Descripción**	**Características Clave**
MSE	Promedio de los errores al cuadrado	Penaliza grandes errores
RMSE	Raíz cuadrada del MSE	Mismas unidades que los valores reales
MAE	Promedio de los errores absolutos	Robusta frente a valores atípicos
MAPE	Promedio de los errores porcentuales	Expresada como un porcentaje
R^2	Proporción de la varianza explicada	Medida de ajuste del modelo

Tabla 6.1: Resumen de métricas de evaluación de modelos.

rendimiento de los modelos contra todo el conjunto completo. Los otros dos experimentos tienen como objetivo evaluar si los modelos, que fueron entrenados con *datasets* particionados de manera aleatoria, son igual de eficientes cuando se prueban sobre un conjunto de datos específico, como lo son el Fin de Semana y período de Vacaciones. Por lo tanto, tenemos:

- Experimento nº 1 - *Standard Test*: Evalúa rendimiento de los modelos sobre todo el conjunto de *testing*.
- Experimento nº 2 - Fin de Semana: Del conjunto de datos de *testing* se toman solo aquellos que corresponden al fin de semana. Se analiza si los modelos entrenados con datos genéricos presentan el mismo rendimiento cuando se usa datos con un patrón temporal específico.
- Experimento nº 3 - Vacaciones: Del conjunto de datos de *testing* se toman solo aquellos que corresponden a la primer quincena de enero, época del año que coincide con vacaciones académicas y mayoría de licencias laborales, generando una merma importante en el flujo del tránsito.

La cantidad de trayectos totales de cada variante fueron detallados y desglosados en la Tabla 3.3 del capítulo referente al conjunto de datos. La Tabla 6.2 muestra el largo del recorrido de cada variante en términos de cantidad de paradas. La Tabla 6.3 contiene el detalle de la cantidad de trayectos disponibles para cada experimento.

Variante	**Cantidad de Paradas**
634	22
635	24
2779	35
2778	21

Tabla 6.2: Largo de variantes, medido en cantidad de paradas.

A continuación, se presentan los resultados obtenidos en cada uno de los tres experimentos realizados. Para cada experimento, se incluye una tabla que muestra el desempeño de los cinco modelos entrenados en las cuatro variantes de estudio, evaluados con las métricas detalladas previamente: RMSE, MAE,

Variante	Experimento	Cantidad de Trayectos
2779	*Standard Test*	713
	Fin de Semana	92
	Vacaciones	30
635	*Standard Test*	4469
	Fin de Semana	923
	Vacaciones	176
634	*Standard Test*	3868
	Fin de Semana	686
	Vacaciones	134
2778	*Standard Test*	769
	Fin de Semana	160
	Vacaciones	28

Tabla 6.3: Cantidad de recorridos por variante y experimento.

MAPE y R^2 (métricas de desempeño mas utilizadas en el EA) y MSE, métrica que se decide incluir ya que fue utilizada como función de perdida para XGBoost y para Bi-GRU. La Tabla 6.4 presenta las métricas obtenidas para el primer Experimento, que evalúa los modelos sobre el conjunto de *testing* completo, el cual está conformado por el 15 % de las trayectorias seleccionadas de manera aleatoria. Las Tablas 6.5 y 6.6 muestran los resultados de los otros dos experimentos, en los que se evalúa el desempeño de los modelos considerando subconjuntos del conjunto de *testing* con características temporales específicas: Fin de Semana y Vacaciones, respectivamente.

Variante	Modelo	MSE(m^2)	RMSE(m)	MAE(m)	MAPE(%)	R^2
634	ARIMA	2.549	1.597	1.201	8.505	0.960
	Bi-GRU	**0.378**	**0.615**	**0.475**	**4.366**	**0.994**
	XGBoost_max	1.754	1.325	1.002	7.564	0.973
	XGBoost_medium	1.978	1.407	1.059	7.776	0.969
	XGBoost_min	2.544	1.595	1.200	8.562	0.960
635	ARIMA	1.946	1.395	1.000	10.578	0.973
	Bi-GRU	**0.081**	**0.284**	**0.216**	**3.786**	**0.999**
	XGBoost_max	1.376	1.173	0.845	10.146	0.981
	XGBoost_medium	1.575	1.255	0.897	9.734	0.978
	XGBoost_min	1.922	1.386	0.987	10.354	0.973
2778	ARIMA	2.570	1.603	1.203	9.501	0.943
	Bi-GRU	1.890	1.375	1.034	8.577	0.958
	XGBoost_max	**1.639**	**1.280**	**0.954**	**8.084**	**0.964**
	XGBoost_medium	1.797	1.340	0.999	8.264	0.960
	XGBoost_min	2.571	1.603	1.203	9.532	0.943
2779	ARIMA	4.234	2.058	1.485	10.542	0.969
	Bi-GRU	**0.440**	**0.663**	**0.493**	**4.611**	**0.997**
	XGBoost_max	2.578	1.606	1.147	9.922	0.981
	XGBoost_medium	2.831	1.682	1.190	8.812	0.979
	XGBoost_min	3.800	1.949	1.377	10.075	0.972

Tabla 6.4: Comparativa para las variantes del Experimento - *Standard_test*.

Variante	Modelo	MSE(m^2)	RMSE(m)	MAE(m)	MAPE(%)	R^2
634	ARIMA	3.353	1.831	1.410	10.285	0.938
	Bi-GRU	**0.385**	**0.620**	**0.480**	**4.537**	**0.993**
	XGBoost_max	1.757	1.326	1.010	7.975	0.968
	XGBoost_medium	1.897	1.377	1.044	8.059	0.965
	XGBoost_min	3.411	1.847	1.425	10.448	0.937
635	ARIMA	2.567	1.602	1.132	11.987	0.960
	Bi-GRU	**0.082**	**0.286**	**0.215**	**3.812**	**0.999**
	XGBoost_max	1.473	1.214	0.869	10.895	0.977
	XGBoost_medium	1.549	1.245	0.885	10.030	0.976
	XGBoost_min	2.487	1.577	1.111	11.796	0.961
2778	ARIMA	2.556	1.599	1.204	9.577	0.943
	Bi-GRU	1.875	1.369	1.043	8.626	0.958
	XGBoost_max	**1.687**	**1.299**	**0.963**	**8.020**	**0.962**
	XGBoost_medium	1.810	1.345	0.999	8.215	0.959
	XGBoost_min	2.557	1.599	1.204	9.597	0.943
2779	ARIMA	5.518	2.349	1.701	12.195	0.958
	Bi-GRU	**1.150**	**1.072**	**0.779**	**6.629**	**0.991**
	XGBoost_max	3.192	1.787	1.267	11.685	0.976
	XGBoost_medium	3.450	1.857	1.296	9.810	0.974
	XGBoost_min	4.936	2.222	1.586	11.757	0.963

Tabla 6.5: Comparativa para las variantes del Experimento - Fin de Semana.

Variante	Modelo	MSE(m^2)	RMSE(m)	MAE(m)	MAPE(%)	R^2
634	ARIMA	2.054	1.433	1.069	7.871	0.963
	Bi-GRU	**0.607**	**0.779**	**0.626**	**5.109**	**0.989**
	XGBoost_max	1.382	1.176	0.899	6.909	0.975
	XGBoost_medium	2.358	1.536	1.159	8.237	0.957
	XGBoost_min	2.144	1.464	1.089	7.953	0.961
635	ARIMA	2.264	1.505	1.071	11.958	0.966
	Bi-GRU	**0.109**	**0.330**	**0.250**	**4.024**	**0.998**
	XGBoost_max	1.302	1.141	0.827	10.344	0.980
	XGBoost_medium	2.232	1.494	1.085	12.243	0.967
	XGBoost_min	2.202	1.484	1.065	12.133	0.967
2778	ARIMA	2.752	1.659	1.272	10.602	0.937
	Bi-GRU	1.893	1.376	1.059	9.113	0.957
	XGBoost_max	**1.776**	**1.333**	**0.980**	**8.507**	**0.959**
	XGBoost_medium	1.860	1.364	1.009	8.882	0.957
	XGBoost_min	2.768	1.664	1.279	10.678	0.937
2779	ARIMA	6.392	2.528	1.871	12.522	0.951
	Bi-GRU	**0.897**	**0.947**	**0.693**	**5.601**	**0.993**
	XGBoost_max	3.497	1.870	1.378	11.091	0.973
	XGBoost_medium	4.292	2.072	1.530	10.076	0.967
	XGBoost_min	5.361	2.315	1.647	11.259	0.959

Tabla 6.6: Comparativa para las variantes del Experimento - Vacaciones.

Estas tablas permiten realizar diferentes tipos de análisis.

Primero, es posible comparar el rendimiento de los distintos modelos dentro de una misma variante, lo que facilita la evaluación de cuál modelo logra un mejor desempeño en cada caso.

Segundo, se pueden comparar los resultados obtenidos para un mismo modelo en distintas variantes, lo que permite evaluar cómo influye la cantidad de datos disponibles en el desempeño del modelo o el largo de la trayectoria.

Además, dado que los experimentos fueron diseñados para evaluar el rendimiento de los modelos en conjuntos de datos con diferentes características, estas tablas permiten analizar cómo varía su precisión según el tipo de datos utilizados. En particular, los experimentos 2 y 3 evalúan el desempeño del modelo cuando se lo aplica a datos que siguen ciertos patrones específicos, como los fines de semana en un caso y la primera quincena de enero en el otro. Esto permite examinar cómo influyen tanto la reducción en la cantidad de datos como las características propias de estos períodos en la capacidad del modelo para adaptarse a distintos escenarios.

Por último, en el caso de XGBoost, para el cual existen tres modelos diferentes, entrenados con distinta cantidad de características, es posible analizar el impacto que éstas tienen y evaluar si existe una mejora significativa en el rendimiento a medida se agregan más características.

6.2.1. Comparación de modelos dentro de una misma variante

Al comparar el desempeño de los modelos dentro de una misma variante, se observa que, en términos generales, el modelo Bi-GRU exhibe el mejor rendimiento según todas las métricas evaluadas. Presenta los valores más bajos de MSE, RMSE, MAE y MAPE, lo que indica menores errores de predicción. Además, su coeficiente de determinación (R^2) es el más alto en la mayoría de las variantes (superior a 0.99 en varios casos), lo que sugiere un ajuste casi perfecto a los datos.

Por el contrario, el modelo ARIMA tiende a mostrar el peor desempeño, con los valores más altos de MSE y RMSE, lo que indica que sus predicciones se alejan más de los valores reales. Esto sugiere que ARIMA tiene dificultades para capturar relaciones no lineales presentes en los datos.

Los modelos XGBoost (max, medium, min) presentan un desempeño intermedio entre Bi-GRU y ARIMA. En general, XGBoost_max obtiene mejores resultados que XGBoost_medium y XGBoost_min, aunque las diferencias entre estas configuraciones no son muy significativas. Esto sugiere que la cantidad de características utilizada para entrenar XGBoost influye en su rendimiento, pero no de manera determinante.

Sin embargo, los resultados muestran que el rendimiento de Bi-GRU no es consistentemente el mejor en todas las variantes. En particular, si comparamos los resultados de la variante 2778, con los de las variantes 634 y 635, donde la cantidad de datos disminuye, pero el largo del recorrido es similar (21, 22 y 24 paradas respectivamente) vemos que el desempeño de Bi-GRU se ve afectado

significativamente. En este caso, la brecha entre Bi-GRU y ARIMA se reduce, lo que indica que, bajo condiciones de largo de trayecto similar pero con poca cantidad de datos, Bi-GRU deja de ser el modelo con mejor ajuste, y XGBoost_max es el que da mejores resultados.

Por lo tanto, Bi-GRU no siempre es el modelo más preciso, XGBoost demuestra que puede llegar a dar muy buenos resultados, y ARIMA a pesar de ser el peor modelo, logra reducir la brecha en determinados escenarios. La cantidad y calidad de los datos disponibles influyen en la efectividad de cada modelo, lo que sugiere que la elección del modelo óptimo debe considerar estas condiciones específicas.

6.2.2. Comparación del desempeño de un mismo modelo en distintas variantes

Al evaluar el desempeño de un mismo modelo en diferentes variantes (dentro de un mismo experimento), se observa que la variante 2779 tiende a generar mayores errores en comparación con las variantes 634 y 635, particularmente en los modelos ARIMA y XGBoost. Esto se evidencia en los valores más elevados de MSE y RMSE, los cuales indican que en promedio, el error en la predicción está por encima de los 2 minutos en los 3 experimentos. Ya que las variantes 2779 y 2778 tienen una cantidad de trayectos similares, y la variantes 2778 dio muy buenos resultados, en un principio, no sería posible afirmar, que la causa del error elevado sea la poca cantidad de trayectorias que tienen estas variantes. Pero en lo que sí se diferencia 2779 de las demás es en el largo de su recorrido. Mientras 2779 tiene 35 paradas en su trayecto, las otras no superan las 24. Esto podría sugerir, que existe una relación entre el largo del camino y la capacidad del modelo.

En cuanto al modelo Bi-GRU, aunque mantiene un buen desempeño en todas las variantes, también muestra un incremento en los errores en la variante 2778, lo que refuerza la importancia de la cantidad de datos en los resultados del desempeño del modelo.

En conclusión, la cantidad de trayectos disponibles tiene un impacto directo en la capacidad predictiva de los modelos, afectando más a aquellos que dependen de un mayor volumen de datos para aprender patrones complejos, como Bi-GRU y XGBoost.

6.2.3. Análisis entre experimentos

En el experimento *Standard Test*, los modelos presentan un desempeño generalmente estable, aunque con algunas diferencias en función de la cantidad de datos disponibles. Bi-GRU destaca con los valores más altos de R^2 (cercanos a 0.99), lo que indica una capacidad de ajuste sobresaliente en todas las variantes, especialmente en aquellas con más trayectos disponibles, como 635 (con 4469 trayectos) y 634 (con 3868 trayectos). XGBoost y ARIMA, por su parte, muestran un desempeño más variable, especialmente en la variante 2779 (con 713 trayectos), donde los errores aumentan, lo que puede explicarse por la

menor cantidad de datos disponibles en esa variante y un largo de trayectoria significativamente mayor.

En términos absolutos, las diferencias entre los valores predichos y reales siguen siendo pequeñas, con desviaciones de minutos en la mayoría de los casos. Esto sugiere que, aunque los modelos varían en su precisión relativa, el impacto práctico de los errores es bajo, especialmente en variantes con un mayor número de trayectos. En general, el MAPE, que indica el porcentaje de error promedio, no supera el 10 %, siendo el peor caso, un valor 12.5 %, para la variante 2779 en el experimento de las Vacaciones, caso en el que coinciden que hay muy pocos datos y la trayectoria es muy larga.

En el experimento Fin de Semana, los errores aumentan en comparación con el *Standard Test*, y en general, los modelos experimentan una pérdida de precisión. Este deterioro es más notable en las variantes con menor cantidad de trayectos. Y se llega a las mismas conclusiones si se compara el experimento Vacaciones con los dos anteriores. Esto, permite concluir que los modelos, que fueron entrenados con datos aleatorios, no responden igual de bien, si se utilizan para probar datos con un patrón de tiempo en particular como lo son el fin de semana y el período de vacaciones.

En cuanto a XGBoost, para el caso del modelo que se entrena con una sola característica, XGBoost_min, en general los resultados son muy similares a los de ARIMA, y en todos los casos mejora cuando se consideran dos características o todas, aunque la diferencia para estos últimos en algunos casos es muy pequeña.

6.2.4. Conclusiones

Los resultados indican que, aunque los modelos presentan variaciones en su desempeño según el tipo de experimento, las diferencias absolutas en los tiempos predichos siguen siendo moderadas, con errores que generalmente no superan los pocos minutos.

Si bien Bi-GRU tiende a tener las mejores predicciones, todos los modelos demostraron una desmejora en la performance a medida que disminuye la cantidad de datos y todos demostraron una desmejora al enfrentarlos a conjuntos de datos con patrones de tiempo específicos, como lo demuestran los experimentos 2 y 3.

En conclusión, si bien ningún modelo es el mejor en todos los escenarios, Bi-GRU presenta el menor error de predicción en la mayoría de los casos, lo que indica que logra capturar mejor las relaciones temporales en los datos. Esto sugiere que los modelos basados en redes neuronales recurrentes pueden ofrecer un mejor desempeño cuando se trabaja con información secuencial.

La buena performance general de todos los modelos, con R^2 relativamente alto en la mayoría de los casos, podría ser consecuencia, entre otras cosas, del exhaustivo trabajo realizado en la calidad de los datos, asegurando que la información utilizada para el entrenamiento sea representativa y confiable. Sin embargo, la dificultad de predicción aumenta en ciertos contextos, como los datos de fin de semana y vacaciones, lo que resalta la importancia de adaptar los

modelos a escenarios específicos.

Estos experimentos permiten verificar las conclusiones iniciales obtenidas en la etapa del Estado del Arte. Los modelos basados en estadísticas, como ARIMA, no son adecuados para analizar las características no lineales de los datos en entornos de tráfico complejos. Los modelos de aprendizaje automático, en particular XGBoost, presenta un excelente rendimiento en series temporales, lo que lo hace adecuado para la predicción de tiempos de viaje. Por último, el experimento con Bi-GRU verifica ser superior a los métodos estadísticos y de aprendizaje automático, pero requiere un alto costo computacional. Como se muestra en la Tabla 5.4, mientras que los modelos de XGBoost tiene un tiempo de entrenamiento apenas superior a 1 minuto en el peor caso (variante 635), el modelo Bi-GRU tiene una demora de casi 6 horas (variante 634).

Capítulo 7

Conclusiones

La predicción del tiempo de viaje es un desafío clave en los sistemas de transporte inteligente y ha sido un tema ampliamente estudiado en la literatura. En este trabajo, a partir de una revisión sistemática, se verificaron los hallazgos teóricos evaluando el desempeño de distintos modelos en la predicción de tiempos de viaje en ómnibus.

El análisis exploratorio de datos fue una herramienta clave para comprender el *dataset*, compuesto por registros de ómnibus de la línea 117 de CUTCSA, entre 2021-2024. Aunque los datos no tenían problemas graves, el estudio permitió corregir inconsistencias a tiempo, lo que resultó en un conjunto de datos limpio para la partición y entrenamiento, mejorando así el desempeño de los modelos.

Con respecto a los enfoques utilizados, Bi-GRU destacó por su desempeño sobresaliente en la mayoría de las pruebas, alcanzando los valores más bajos de error en las métricas utilizadas. Esto refleja la capacidad de las redes neuronales recurrentes bidireccionales, para modelar de manera eficiente las complejas dinámicas de los tiempos de viaje. Sin embargo, este alto rendimiento viene acompañado de un costo computacional mucho mayor, con tiempos de entrenamiento desorbitantes en comparación con los otros modelos. Por otro lado, XGBoost mostró un equilibrio entre precisión y eficiencia, con tiempos de ejecución muy bajos y un desempeño muy bueno, especialmente en su configuración con mayor cantidad de características. Este modelo demostró ser el más rápido de entrenar y evaluar, lo que lo hace una opción atractiva para su aplicación en tiempo real o con recursos computacionales limitados. En cuanto al modelo base, ARIMA presentó los peores resultados en la mayoría de los casos, lo que refleja sus limitaciones como modelo estadístico. Su capacidad para abordar patrones complejos y no lineales es más restringida en comparación con modelos de aprendizaje automático superficial y profundo.

Por último, los distintos modelos de XGBoost mostraron que la cantidad de características influye en la precisión de la predicción final. Además, la cantidad de trayectos disponibles para el entrenamiento, también impacta significativamente, especialmente en modelos como Bi-GRU, que requieren más datos para mejorar su desempeño, como se indicó en el Estado del Arte.

En cuanto a limitaciones, en este estudio se optó por eliminar trayectorias con valores nulos, aunque el uso de técnicas de imputación, como la interpolación, podría permitir aprovechar más trayectos. Esto mejorararía la representatividad del conjunto de datos, lo que podría aumentar la capacidad predictiva del modelo.

Por otro lado, la búsqueda de hiperparámetros no fue unificada entre los modelos, lo que dificultó una comparación justa de los tiempos de entrenamiento. La eficiencia computacional varió según las técnicas de optimización empleadas, influyendo en los tiempos de ejecución junto con las propias características de cada modelo. A esto se suma que la GPU no fue utilizada en el entrenamiento de Bi-GRU debido a dificultades técnicas en su configuración y al tiempo limitado para resolverlas. Con una implementación adecuada, el uso de la GPU habría permitido reducir significativamente los tiempos de entrenamiento de Bi-GRU.

Otra limitación es que, actualmente el problema se plantea prediciendo en una sola instancia el tiempo total que resta hasta el destino final. Sin embargo, esta aproximación enfrenta dificultades, ya que cualquier error en la predicción inicial se acumula a lo largo del trayecto, especialmente en rutas largas y con alta variabilidad en las primeras paradas. Una alternativa sería dividir la predicción en tramos más pequeños, calculando el tiempo de viaje entre paradas consecutivas y luego sumando estos valores. Este enfoque permite distribuir mejor los errores y reducir su impacto en ciertos casos, aunque también podría introducir nuevas fuentes de desviación si los errores en cada tramo se acumulan.

En cuanto al impacto y la aplicación, los resultados obtenidos en este estudio pueden ser una herramienta valiosa para los reguladores del tráfico en Montevideo. Contar con predicciones más precisas de los tiempos de viaje permitiría tomar decisiones más informadas sobre la gestión del tráfico, como ajustar semáforos, dar prioridad a ciertos trayectos o implementar medidas de control para mejorar la circulación y reducir la congestión.

Por otro lado, la IMM podría ajustar los horarios de los ómnibus y la frecuencia de los viajes de manera más realista, basándose en una mayor precisión de los tiempos de viaje. Si se detectan tramos con retrasos frecuentes, se podrían modificar los horarios para reflejar los tiempos reales de recorrido, mejorando la fiabilidad del servicio.

Por último, aunque la predicción del tiempo restante hasta el destino es útil para los inspectores y choferes, puede que no sea el indicador más relevante para los pasajeros. Dado que muchos no viajan hasta la última parada, predecir el tiempo de viaje entre paradas consecutivas o estimar el tiempo de arribo a una parada específica ofrecería una información más útil y precisa, mejorando la experiencia del usuario. Este tipo de predicción no solo optimizaría la experiencia de viaje, sino que también contribuiría a una mayor confianza en el sistema de transporte público.

Trabajos futuros

Existen diversas líneas de trabajo que podrían extender este estudio.

Una mejora natural sería la incorporación de características exógenas, como condiciones climáticas, eventos especiales, o incidentes de tráfico. Esto permitiría enriquecer la capacidad predictiva del modelo, especialmente al evaluar el impacto de climas adversos, como lluvia o niebla, en la precisión de las predicciones. Otra mejora prometedora sería modelar separadamente los períodos en que el ómnibus está en marcha y los que está detenido en una parada, lo que permitiría obtener predicciones más precisas. Para ello, se necesitaría información adicional sobre cuándo el ómnibus está en movimiento o detenido. Esta información podría obtenerse mediante el aumento de la frecuencia de muestreo usando la locación del coche, o a partir del registro que efectúan las tarjetas del Sistema de Transporte Metropolitano al subir a la unidad.

Las líneas de ómnibus urbanos pueden estar sujetas a patrones temporales relacionados con factores como la hora del día, el día de la semana o la variabilidad a lo largo del año. Si se confirmara que los tiempos de viaje siguen estos ciclos, se esperaría que un modelo como *Seasonal Autoregressive Integrated Moving Average* mejorara la precisión de las predicciones en comparación con ARIMA, al capturar estos patrones y mejorar las predicciones a largo plazo.

En cuanto a nuevas arquitecturas, sería útil combinar redes neuronales recurrentes, que capturan las características temporales, con redes como *Convolutional Neural Networks* o *Graph Convolutional Networks*, que se enfocan en las relaciones espaciales, como la densidad de tráfico o la congestión en diferentes áreas. Esta combinación permitiría perfeccionar el modelo al aprovechar ambos tipos de dinámicas, mejorando así su precisión y robustez.

Otra dirección interesante podría ser integrar información en tiempo real para mejorar las predicciones de los tiempos de viaje. Usando *Kalman Filter*, se podrían ajustar las predicciones a medida que se actualizan los datos GPS del ómnibus, permitiendo que el modelo se adapte rápidamente a cambios imprevistos en el tráfico. Este enfoque mejoraría la precisión de las predicciones sin requerir información continua sobre el estado del tráfico o características espaciales, ya que *Kalman Filter* puede gestionar la incertidumbre en los datos de ubicación para ajustar las predicciones.

Por último, otra aplicación en tiempo real sería el ajuste dinámico de la frecuencia de los ómnibus en función de la demanda. A partir de los registros históricos y la cantidad de pasajeros que suben en cada parada, se podrían identificar patrones de uso y anticipar períodos de mayor o menor demanda. Esto permitiría adaptar la frecuencia de los servicios de manera más eficiente, optimizando la asignación de recursos y mejorando la experiencia de los usuarios.

Referencias

Abdollahi, M., Khaleghi, T., y Yang, K. (2020). An integrated feature learning approach using deep learning for travel time prediction. *Expert Systems with Applications*, *139*(112982).

Ahmed, I., Kumara, I., Reshadat, V., Kayes, A. S., van den Heuvel, W. J., y Tamburri, D. A. (2022). Travel time prediction and explanation with spatio-temporal features: A comparative study. *Electronics (Switzerland)*, *11*(106).

Ali, Z., Abduljabbar, Z., Tahir, H., Sallow, A., y Almufti, S. (2023). Exploring the power of extreme gradient boosting algorithm in machine learning: a review. *Academic Journal of Nawroz University*, *12*, 320-334.

Alkilane, K., Alfateh, M. T. E., y Yanming, S. (2023). Travel time prediction based on route links' similarity. *Neural Computing and Applications*, *35*, 3991-4007.

Ashwini, B. P., Sumathi, R., y Sudhira, H. S. (2022). Bus travel time prediction: A comparative study of linear and non-linear machine learning models. En (Vol. 2161). IOP Publishing Ltd.

Awad, M., y Khanna, R. (2015). *Efficient learning machines: Theories, concepts, and applications for engineers and system designers* (1st ed.). New York, NY: Apress Media, LLC.

Bharathi, D., Sopeña, J. M. G., Clarke, S., y Ghosh, B. (2023). Travel time prediction utilizing hybrid deep learning models. *Transportation Research Record*, *2678*, 56 - 65.

Bre, F., Gimenez, J., y Fachinotti, V. (2017). Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, *158*.

Brockwell, P. J., y Davis, R. A. (2002). *Introduction to time series and forecasting, second edition.* New York: Springer.

Chang, T. H., Li, Y. R., Fu, C. H., Liu, Y. B., y Yang, S. M. (2019). Travel time prediction based on missing data compensation. En (Vol. 128, p. 328-336). Springer Science and Business Media Deutschland GmbH.

Chatfield, C. (2016). *The analysis of time series: An introduction, sixth edition.* Boca Raton, FL, USA: Chapman and Hall/CRC.

Chen, C. H., Hwang, F. J., y Kung, H. Y. (2019). Travel time prediction system based on data clustering for waste collection vehicles. *IEICE Transactions on Information and Systems*, *E102D*, 1374-1383.

Chen, C. Y., Sun, E. W., Chang, M. F., y Lin, Y. B. (2023). Enhancing travel time prediction with deep learning on chronological and retrospective time order information of big traffic data. *Annals of Operations Research*.

Chen, M. Y., Chiang, H. S., y Yang, K. J. (2022). Constructing cooperative intelligent transport systems for travel time prediction with deep learning approaches. *IEEE Transactions on Intelligent Transportation Systems*, *23*, 16590-16599.

Chen, T., y Guestrin, C. (2016). XGBoost: A scalable tree boosting system. En *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794). New York, NY, USA: ACM.

Chughtai, J.-U.-R., Haq, I. U., y Muneeb, M. (2022). An attention-based recurrent learning model for short-term travel time prediction. *PloS ONE*, *17*(e0278064).

Chughtai, J.-U.-R., Haq, I. U., Shafiq, O., y Muneeb, M. (2022). Travel time prediction using hybridized deep feature space and machine learning based heterogeneous ensemble. *IEEE Access*, *10*, 98127-98139.

Chughtai, J.-U.-R., ul Haq, I., ul Islam, S., y Gani, A. (2022). A heterogeneous ensemble approach for travel time prediction using hybridized feature spaces and support vector regression. *Sensors*, *22*.

Deb, B., Khan, S. R., Tanvir Hasan, K., Khan, A. H., y Alam, M. A. (2019). Travel time prediction using machine learning and weather impact on traffic conditions. En (p. 1-8).

Elmi, S., y Tan, K. L. (2020). Travel time prediction in missing data areas: Feature-based transfer learning approach. En (p. 1088-1095). Institute of Electrical and Electronics Engineers Inc.

Fan, S., Li, J., Lv, Z., y Zhao, A. (2021). Multimodal traffic travel time prediction. En (p. 1-9). Institute of Electrical and Electronics Engineers Inc.

Fang, H., Liu, Y., Chen, C. H., y Hwang, F. J. (2022). Travel time prediction method based on spatial-feature-based hierarchical clustering and deep multi-input gated recurrent unit. *ACM Transactions on Sensor Networks*, *19*(26).

Ghosal, S. S. (2019). Travel time prediction of taxis using tensor factorization. En (p. 314-317). Association for Computing Machinery.

Huang, H., Pouls, M., Meyer, A., y Pauly, M. (2020). Travel time prediction using tree-based ensembles. En (Vol. 12433, p. 412-427). Springer International Publishing.

Huang, Y., Dai, H., y Tseng, V. S. (2022). Periodic attention-based stacked sequence to sequence framework for long-term travel time prediction. *Knowledge-Based Systems*, *258*(109976).

Hyndman, R. J., y Athanasopoulos, G. (2018). *Forecasting: Principles and practice*. Melbourne, Australia: OTexts.

Islek, I., y Oguducu, S. G. (2019). Use of lstm for short-term and long-term travel time prediction. En (Vol. 2482). CEUR-WS.

Jakteerangkool, C., y Muangsin, V. (2020). Short-term travel time prediction from gps trace data using recurrent neural networks. En (p. 62-66).

Institute of Electrical and Electronics Engineers Inc.

Khaled, A., Elsir, A. M., y Shen, Y. (2022). Gsta: gated spatial–temporal attention approach for travel time prediction. *Neural Computing and Applications*, *34*, 2307-2322.

Kitchenham, B. (2004). *Procedures for performing systematic reviews* (Inf. Téc.). Keele University.

Kumar, B. A., Jairam, R., Arkatkar, S. S., y Vanajakshi, L. (2019). Real time bus travel time prediction using k-nn classifier. *Transportation Letters*, *11*, 362-372.

Kumar, B. A., Mothukuri, S., y Vanajakshi, L. (2021). Numerical stability of conservation equation for bus travel time prediction using automatic vehicle location data. *International Journal of Intelligent Transportation Systems Research*, *19*, 141-154.

Kumar, B. A., Vanajakshi, L., y Subramanian, S. C. (2017). Bus travel time prediction using a time-space discretization approach. *Transportation Research Part C: Emerging Technologies*, *79*, 308-332.

Lee, G., Choo, S., Choi, S., y Lee, H. (2022). Does the inclusion of spatio-temporal features improve bus travel time predictions? a deep learning-based modelling approach. *Sustainability (Switzerland)*, *14*(7431).

Li, J., Yue, J., y Hou, J. (2024). Research on risk prediction models based on arima and xgboost. En *Proc. of the 3rd int. conf. on data analytics, computing and artificial intelligence* (pp. 588–593). New York, NY: IEEE.

Li, N., Wu, Y., Wang, Q., Ye, H., Wang, L., Jia, M., y Zhao, S. (2023). Underground mine truck travel time prediction based on stacking integrated learning. *Engineering Applications of Artificial Intelligence*, *120*(105873).

Li, Y., Zhang, M., Ding, Y., Zhou, Z., y Xu, L. (2022). Real-time travel time prediction based on evolving fuzzy participatory learning model. *Journal of Advanced Transportation*, *2022*.

Mendes-Moreira, J., y Baratchi, M. (2020). Reconciling predictions in the regression setting: An application to bus travel time prediction. En (Vol. 12080, p. 313-325). Springer.

Moosavi, S. M. H., Aghaabbasi, M., Yuen, C. W., y Armaghani, D. J. (2023). Evaluation of applicability and accuracy of bus travel time prediction in high and low frequency bus routes using tree-based ml techniques. *Journal of Soft Computing in Civil Engineering*, *7*, 74-97.

Moradi, M., y Samwald, M. (2022). Deep learning, natural language processing, and explainable artificial intelligence in the biomedical domain.

Mori, U., Mendiburu, A., Álvarez, M., y Lozano, J. A. (2015). A review of travel time estimation and forecasting for advanced traveller information systems. *Transportmetrica A: Transport Science*, *11*, 119-157.

Nicolás Arrioja. (2021). *Métricas en regresión.* https://medium.com/@nicolasarrioja/m%C3%A9tricas-en-regresi%C3%B3n-5e5d4259430b. (Accessed: 2024-12-28)

Nithishwer, M. A., Kumar, B. A., y Vanajakshi, L. (2022). Deep learning– just data or domain related knowledge adds value?: bus travel time prediction as a case study. *Transportation Letters*, *14*, 863-873.

Ou, Y. (2022). Ai for real-time bus travel time prediction in traffic congestion management. En (p. 63-84). Springer International Publishing.

Pérez-González, C. M., Mora-Vargas, J., Piña-Barcenas, J., y Cedillo-Campos, M. G. (2023). Method for travel time prediction in emerging markets based on anonymous truck gps data. *Annals of Operations Research*.

Qiu, B., y Fan, W. (2021). Machine learning based short-term travel time prediction: Numerical results and comparative analyses. *Sustainability (Switzerland)*, *13*(7454).

Ran, X., Shan, Z., Fang, Y., y Lin, C. (2019). A convolution component-based method with attention mechanism for travel-time prediction. *Sensors (Switzerland)*, *19*(2063).

Schwinger, F., Frohnhofen, C., Wernz, J., Braun, S., y Jarke, M. (2021). Combining profile similarity and kalman filter for real-world applicable short-term bus travel time prediction. En (Vol. 2021-September, p. 3738-3745). Institute of Electrical and Electronics Engineers Inc.

Shen, Y., Jin, C., Hua, J., y Huang, D. (2022). Ttpnet: A neural network for travel time prediction based on tensor decomposition and graph embedding. *IEEE Transactions on Knowledge and Data Engineering*, *34*, 4514-4526.

Sihag, G., Parida, M., y Kumar, P. (2022). Travel time prediction for traveler information system in heterogeneous disordered traffic conditions using gps trajectories. *Sustainability (Switzerland)*, *14*(10070).

Tang, R., Kanamori, R., y Yamamoto, T. (2019). Short-term urban link travel time prediction using dynamic time warping with disaggregate probe data. *IEEE Access*, *7*, 98959-98970.

Thakkar, S., Sharma, S., Advani, C., Arkatkar, S. S., y Bhaskar, A. (2021). Comparative analysis of travel time prediction algorithms for urban arterials using wi-fi sensor data. En (p. 697-702). Institute of Electrical and Electronics Engineers Inc.

Understanding Activation Functions in Deep Learning. (2025). https://learnopencv.com/understanding-activation-functions-in-deep-learning/. (Accessed: 2025-03-17)

Wang, D., Zhu, J., Yin, Y., Ignatius, J., Wei, X., y Kumar, A. (2023). Dynamic travel time prediction with spatiotemporal features: using a gnn-based deep learning method. *Annals of Operations Research*.

Wu, C. H., Ho, J. M., y Lee, D. T. (2004). Travel-time prediction with support vector regression. En (Vol. 5, p. 276-281).

Wu, J. (2014). *Analysis of taxi drivers' driving behavior based on a driving simulator experiment.*

WU Weather Underground. (2025). https://www.wunderground.com. (Accessed: 2025-03-06)

XGBoost Documentation: Parameters. (2025). https://xgboost.readthedocs.io/en/stable/parameter.html. (Accessed: 2025-03-06)

Xu, M., y Liu, H. (2021). A flexible deep learning-aware framework for travel time prediction considering traffic event. *Engineering Applications of Artificial Intelligence*, *106*(104491).

Zhang, K., Lai, Y., Jiang, L., y Yang, F. (2020). Bus travel-time prediction based

on deep spatio-temporal model. En (Vol. 12342, p. 369-383). Springer Science and Business Media Deutschland GmbH.

Zhang, L., Wang, Y., Yang, X., Zhang, C., Hao, Z., y Liu, Y. (2021). Travel time prediction for urban road based on machine learning: Review and prospect. En (p. 124-129). Institute of Electrical and Electronics Engineers Inc.

Zúñiga, J. (2020). Aplicación de algoritmos random forest y xgboost en una base de solicitudes de tarjetas de crédito. *Ingeniería Investigación y Tecnología*, *21*, 1-16.

Anexo A

Estado del Arte

Adjunto a este informe se incluye el documento **EP25_Estado_del_Arte.pdf**, el cual contiene la revisión sistemática completa del Estado del Arte relacionado con el problema de la predicción de tiempos de viaje.

Anexo B

Recursos

Repositorio

El repositorio con el código fuente de la fase de experimentación se encuentra disponible en el GitLab de la Facultad de Ingeniería (ttp_proyect) para que se pueda revisar o continuar los experimentos, ya sea explorando otras configuraciones de hiperparámetros, agregando nuevos modelos o incorporando más líneas de ómnibus.

El repositorio está ampliamente documentado e incluye un archivo *README.md* detallado, que proporciona toda la información necesaria para ejecutar el código, comprender la estructura del proyecto y acceder a datos relevantes para su uso.

El diseño del código está orientado a la escalabilidad, permitiendo que en el futuro se puedan añadir nuevas líneas de ómnibus o funcionalidades.

Anexo C

Gestión de Proyecto

El presente anexo contiene detalles sobre los elementos relacionados con la gestión del proyecto, herramientas utilizadas, planificación y tareas llevadas a cabo durante el desarrollo del mismo.

C.1. Etapas

El proyecto consistió de cuatro etapas principales:

1. Elaboración de Estado del Arte: Se realiza la depuración del *string* de búsqueda, la lectura de artículos y la elaboración del informe del Estado del Arte.
2. Búsqueda de *dataset* y evaluación de herramientas para el modelado: En paralelo mientras se lleva a cabo la búsqueda de *dataset*, se evalúan distintas herramientas/librerías para la implementación de los modelos.
3. Experimentación: Se realiza el proceso de análisis y limpieza de datos, así como la implementación de los modelos seleccionados.
4. Elaboración de Informe final.

La siguiente Figura C.1 muestra las diferentes fases a través del tiempo.

C.2. Metodología utilizada

Para organizar el trabajo se procedió a tomar algunos elementos de las metodologías ágiles.

- Reuniones **periódicas**: Se realizan reuniones cada dos o tres semanas con los tutores. En cada reunión se presentan los avances, se resuelven dudas y se establecen los objetivos para el siguiente "sprint".

Figura C.1: Línea de Tiempo.

- *Product Backlog*: Lista de tareas a realizar, ordenadas en un tablero Trello por prioridad, de arriba hacia abajo.
- Historias de Usuario: Tarjetas Trello con descripción de la tarea a realizar, pre-requisitos, persona asignada a la tarea y en los casos que corresponda, criterios de aceptación.
- Reuniones internas: Se organizan reuniones semanales para planificar el trabajo, revisar avances, discutir ideas y resolver problemas según las necesidades.

A su vez, durante la fase de Implementación, se utiliza:

- Tablero: diseñado específicamente para cubrir las distintas etapas del desarrollo. Dicho tablero contiene las siguientes columnas: *Backlog*, *ToDo*, *InProgress*, *Validation/Testing* y *Done*, de manera de organizar el trabajo pendiente, en progreso y finalizado.
- *GitFlow*: modelo de ramas para organizar y gestionar el desarrollo.
  - Rama principal (*main*): Contiene la versión estable del software (que ya no implica cambios salvo algún *hotfix*).
  - Rama de desarrollo (*develop*): Contiene el código en desarrollo. Es el punto de integración para las nuevas características.
  - Ramas de características (*feature/<branch-name>*): Se crean desde develop. Cada rama corresponde a una nueva funcionalidad, historia

de usuario, o tarea específica. Una vez completadas, se integran de nuevo en develop.

- Ramas de corrección rápida (*hotfix/<branch-name>*): para corregir problemas críticos.
- *Pull Request*: Creación de *Pull Request* para integrar una *branch* a *develop* y/o *main*. De esta manera se garantiza que el nuevo código sea revisado y testeado por el otro compañero.

Como herramienta principal de comunicación se utiliza Slack, con diferentes canales para organizar y compartir la información de manera ordenada e integrado con GitHub y Trello para recibir notificaciones de los cambios en dichas herramientas. Los principales canales utilizados son:

- Datos: Canal para centralizar los aspectos relacionados con el análisis y depuración del *dataset*.
- Modelos: Canal para compartir conceptos, material, discusión referente a los modelos a implementar.
- github-news: Genera un mensaje de notificación al momento de crear un nuevo *Pull Request* en *git*.
- trello-news: Genera un mensaje de notificación al momento de efectuar algún cambio en Trello, como creación de una nueva tarjeta o avance de la misma (moverla a una nueva columna).
- gestion-trello: Para todo lo referente a la gestión del Trello.

C.3. Herramientas Utilizadas

Herramientas	**Uso**
Mendeley	Herramienta utilizada para guardar la bibliografía del Estado del Arte.
Overleaf	Herramienta para la edición colaborativa en Latex de los informes (Estado del Arte e Informe Final).
Google Sheets	Se utiliza para: ▪ Respaldo de información (año, autor, buscador, etc.) de los artículos recuperados. ▪ Creación de gráficos. ▪ Análisis inicial de artículos: se crea *Template* para relevar información clave de una primera lectura: año, autor, contexto, modelos , tipo vehículo, etc.
Slack y Zoom	Comunicación y colaboración en equipo.
GitHub	Control de versiones y alojamiento de repositorio.
Visual Studio Code	Desarrollo y edición de código fuente.
Trello	Gestión de proyectos. Se utilizan tres tableros a lo largo del desarrollo del proyecto: ▪ Tablero para organizar las tareas relacionadas con la redacción del Estado del Arte. ▪ Tablero para centralizar la comparativa de las diferentes herramientas evaluadas para el desarrollo de cada modelo. ▪ Tablero para la fase de Implementación y desarrollo del Informe final.
Draw.io y Canva	Creación de diagramas y esquemas.
Google Docs y Notion	Herramientas colaborativas de creación de documentos.

Tabla C.1: Herramientas y su uso

Anexo D

Mapas de las Variantes

El presente anexo contiene mapas que muestran los recorridos para cada una de las variantes. Para construir estos mapas, se procede a marcar con un punto cada una de las direcciones asociadas a cada uno de los ordinales de la variante. Luego de marcar los ordinales, se traza una línea que conecta a cada uno de ellos. Por lo tanto, las líneas observadas no son una representación exacta del recorrido que hace el ómnibus, pero es una aproximación bastante acertada, que permite complementar el análisis con material visual, y que fue clave en el análisis para identificar el traslado de paradas. D.1, D.2, D.3 y D.4.

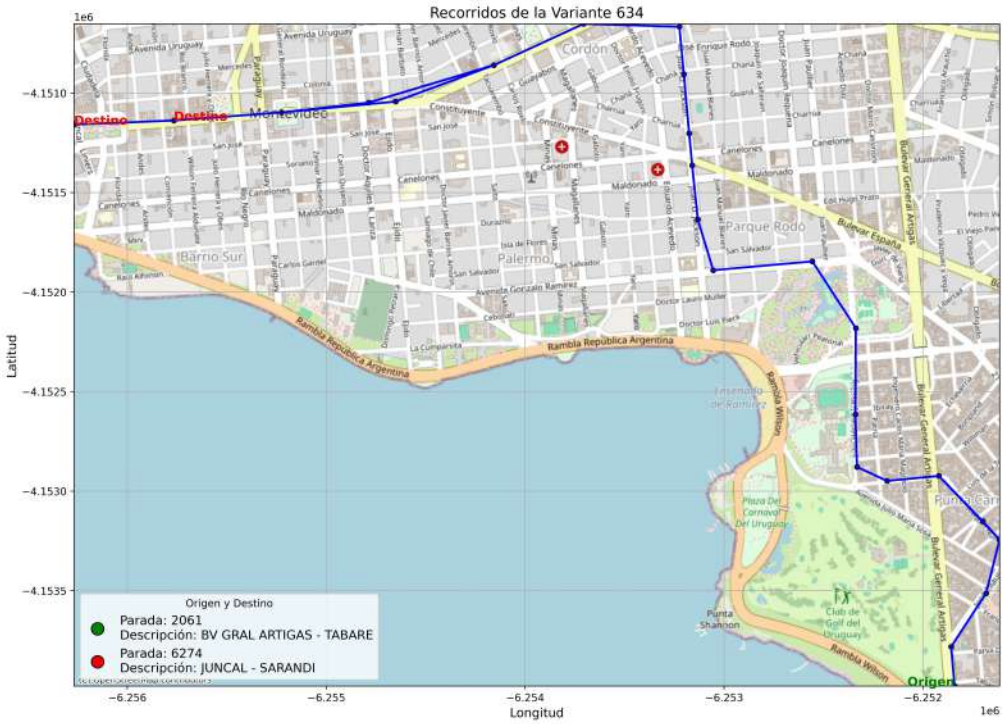

Figura D.1: Variante 634.

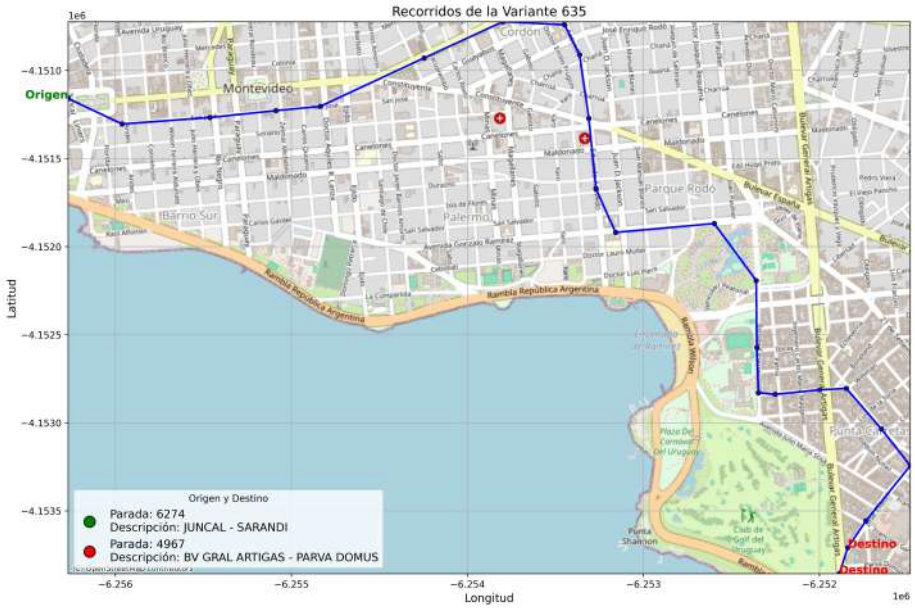

Figura D.2: Variante 635.

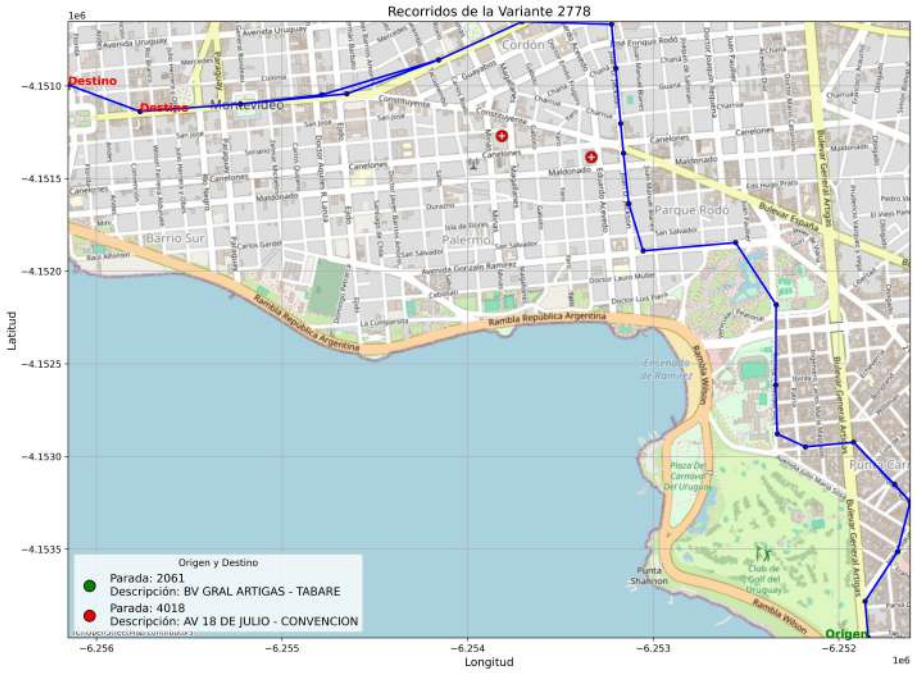

Figura D.3: Variante 2778.

Figura D.4: Variante 2779.

Anexo E

Definiciones Importantes

El presente anexo contiene un glosario con conceptos relevantes para este informe así como una tabla con todas las siglas relevadas en la revisión del estado del arte.

Glosario

- Detector: dispositivo instalado bajo la superficie de la carretera, que detecta cuando un vehículo pasa sobre él (Jakteerangkool y Muangsin, 2020; Thakkar y cols., 2021).
- GPS: dispositivo que aporta ID para poder identificar la unidad, latitud y longitud de la ubicación por la que pasa el vehículo y la marca de tiempo de cuando se obtiene el registro (Kumar y cols., 2019, 2021; Pérez-González y cols., 2023; Fan y cols., 2021; Sihag y cols., 2022; Jakteerangkool y Muangsin, 2020; Ghosal, 2019).
- Q1 (Primer Cuartil): Es el valor que deja el 25 % de los datos por debajo y el 75 % por encima. Es decir, es el punto donde termina el primer cuarto de los datos ordenados.
- Q3 (Tercer cuartil): Es el valor que deja el 75 % de los datos por debajo y el 25 % por encima. Es decir, marca el límite donde termina el tercer cuarto de los datos ordenados.
- Rango Intercuartílico (IQR): El rango intercuartílico (RI) es una medida de dispersión que indica la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1) de un conjunto de datos. Se calcula como:

  $$RI = Q_3 - Q_1$$

  Este rango mide la variabilidad central de los datos, excluyendo los valores extremos, y es útil para describir la dispersión sin ser afectado por valores atípicos.

- Variante: cada variante representa una ruta diferente, es decir, un par origen-destino distinto.

Siglas

Se presenta la sigla, su definición, y en los casos en los que hace sentido se presenta la sigla en español.

Sigla	**Definición en Inglés**	**Definición en Español**
AI	Artificial Intelligence	Inteligencia Artificial
ANN	Artificial Neural Network	Red Neuronal Artificial
AR	Auto-Regressive	Autoregresivo
ARIMA	Auto-Regressive Integrated Moving Average	Autorregresivo integrado de media móvil
ARMA	Auto-Regressive Moving Average	Autorregresivo de media móvil
ATIS	Advanced Traveler Information Systems	Sistemas Avanzados de Información para Viajeros
ATP	Arrival Time Prediction	Predicción de hora de llegada
AVL	Automatic Vehicle Location	Localización Automática de Vehículos
BDIGRU	Bidirectional Isometric-Gated Recurrent Unit	-
CHAID	Chi-square Automatic Interaction Detection	Detección automática de interacciones a partir de Chi-Cuadrado
CITS	Cooperative Intelligent Transport Systems	Sistemas de Transporte Inteligente Cooperativo
CNN	Convolutional Neural Network	Red Neuronal Convolucional
CORSIM	Corridor Simulation	Simulación de Corredores
DL	Deep Learning	Aprendizaje Profundo
DT	Decision Tree	Árbol de Decisión
FCD	Floating Car Data	Datos de Vehículos en Movimiento
GBDT	Gradient Boosting Decision Trees	-
GBR	Gradient Boosting Regression	-
GNN	Graph Neural Network	Red Neuronal de Grafos
GPS	Global Positioning System	Sistema de Posicionamiento Global
GRU	Gated Recurrent Unit	-

IoT	Internet of Things	Internet de las Cosas
IoV	Internet of Vehicle	-
ITS	Intelligent Transportation System	Sistema de Transporte Inteligente
JSTC	Joint Spatial-Temporal Correlation	-
k-NN	k-Nearest Neighbors	k-Vecinos Más Cercanos
KF	Kalman Filter	Filtro de Kalman
LASSO	Least Absolute Shrinkage and Selection Operator	-
LR	Linear Regression	Regresión Lineal
LSTM	Long Short-Term Memory	-
MA	Moving Average	Media Móvil
MC-GRU	Multimodal Convoluted Gated Recurrent Unit Network	-
ML	Machine Learning	Aprendizaje Automático
MLP	Multilayer Perceptron	Perceptrón Multicapa
OD	Origin-Destination	Origen-Destino
QR	Quantile Regression	Regresión Cuantil
R4R	Reconciliation for Regression	-
RF	Random Forest	Bosque Aleatorio
RFR	Random Forest Regression	-
RITIS	Regional Integrated Transportation Information System	-
RNN	Recurrent Neural Network	Red Neuronal Recurrente
RR	Ridge Regression	Regresión Ridge
RS	-	Revisión Sistemática
RT	Regression Trees	Árboles de regresión
SARIMA	Seasonal Auto-Regressive Integrated Moving Average	-
STHM	Spatio-Temporal Hidden Markov	-
SVR	Support Vector Regression	Regresión de Vectores de Soporte
TTP	Travel Time Prediction	Predicción de Tiempo de Viaje