UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE CIENCIAS DE LA COMPUTACIÓN

Desarrollo de modelos de aprendizaje automático: Regresión GBT, Regresión Lineal y RNN para la predicción del transporte público, con el propósito de presentar una propuesta de implementación en la ciudad de Lima

TRABAJO DE INVESTIGACIÓN

Para optar el grado de bachiller en Ciencias de la Computación

AUTOR

Mateo Noel Rabines

ASESOR

Jose Antonio Fiestas Iquira

Lima - Perú

2022

TABLA DE CONTENIDO

Pág.

RESUMEN

Este documento presenta el diseño de un sistema de predicción para el transporte público en Lima, Perú. El sistema se basa en la aplicación de técnicas de aprendizaje automático (ML) a grandes conjuntos de datos, con el objetivo de predecir las características de los autobuses, como los tiempos de llegada. Se exploran varios modelos de predicción, incluyendo *GBTRegression*, *Linear Regression*, RNN (LSMT), ANN, y se discute la optimización de la función para adaptarla mejor al caso de los autobuses. El documento también detalla el proceso de preprocesamiento de datos, que incluye la eliminación de ciertos datos, la agrupación de datos por 'vehicle id', y diversas funciones. Los algoritmos desarrollados para este sistema están disponibles en un repositorio de GitHub. El documento concluye con una discusión de los resultados obtenidos, obteniendo un modelo RNN(LSTM) como mejor solución para el caso de subrutas y algunos casos de rutas. Además de proponer un diseño de predicción para la realidad actual de Lima.

GitHub

https://github.com/mateonoel2/tesisUtecMateoNoel

Palabras clave:

GBT; ANN; Linar Regression; RNN; ML; Dask; Spark

ABSTRACT

Design of a Public Transportation Prediction System in Lima

This document presents the design of a prediction system for public transportation in Lima, Peru. The system is based on the application of machine learning (ML) techniques to large datasets, with the aim of predicting bus characteristics such as arrival times. Several prediction models are explored, including GBTRegression, Linear Regression, RNN (LSMT), and ANN, and the optimization of the function to better adapt it to the bus case is discussed. The document also details the data preprocessing process, which includes data removal, data grouping by vehicle ID, and various functions. The algorithms developed for this system are available in a GitHub repository. The paper concludes with a discussion of the results obtained, obtaining a RNN(LSTM) model as the best solution for the case of subroutes and some cases of routes. In addition to proposing a predictive design for the current reality of Lima.

Keywords:
GBT; ANN; Linear Regression; RNN; ML; Dask; Spark

INTRODUCCIÓN

El tráfico urbano es un problema al cual los habitantes de Lima se enfrentan a diario. Lima, Perú se encuentra dentro de las 20 ciudades con mayor congestión de tráfico en el mundo *(tomtom, Lima traffic report [1])*. En este contexto, las personas buscan formas de movilizarse por la ciudad de manera más óptima. Por ende, lo que se va a lograr es realizar un sistema que nos ayude en este rubro.

El tráfico en Lima es un problema real. La ATU (Autoridad de Transporte Urbano) permite ver la ruta abiertamente de los medios formales de transporte que en este caso es el Metropolitano y los corredores. Sin embargo, debido a la informalidad, tenemos una gran cantidad de microbuses que tienen una ruta, pero si no vives por la zona no vas a tener conocimiento de cómo se trasladan. Esto genera mayor congestión e incertidumbre. Necesitamos que el transporte público sea ordenado y una planificación urbana de transporte bien gestionada.

Ya que el tráfico es desordenado e impredecible con cálculos manuales, la aplicación de técnicas de ML (Machine Learning) puede ser una herramienta valiosa para predecir el comportamiento del tráfico y mejorar la eficiencia del transporte público. Un estudio reciente realizado por Pałys et al, Polonia, utilizó datos de geolocalización de autobuses en tiempo real para predecir retrasos en el transporte público *(Pałys et al., 2022 [2])*. Este tipo de enfoques podría ser de utilidad en una ciudad como Lima, donde la congestión de tráfico es alta y los patrones de tráfico son impredecibles. En otra investigación, G. Bravo 2018 realizó un estudio donde se procesa la información de buses de Nueva York y la utiliza para predecir la posición de los buses, su recorrido, velocidad y otras características *(Bravo, 2018 [3])*. Entonces buscamos mejorar el algoritmo y aplicarlo a la ciudad de Lima.

Lo que se busca realizar en este trabajo es dejar una base que sirva para un futuro análisis de las líneas del corredor y poder predecir cómo se van a comportar en el futuro; en este caso en la ciudad de Lima. Se utlizarán herramientas de Big Data

y ML para el pre-procesamiento y predicción del comportamiento de traslado de autobuses.
Se utilizará Dask y pySpark para distribuir los datos en el preprocesamiento y realizarlo en paralelo, optimizando tiempos de procesamiento. También se utilizará pySpark.ml y Keras para entrenar los modelos de ML. Además, existen distintos métodos de predicción de buses, como se explora en *(Yin et al., 2017 [4])*, *(Bachu et al. 2016 [5])* y *(Pałys et al., 2022 [2])*. Donde se utilizan técnicas de KNN, SVM (Support vector machine) y ANN (Artificial neural network) para realizar modelos de ML. Algunos algoritmos que no han sido aplicados en esta área son los algorítmos de regresión GBT(Gradient Boosted Tree) y Random Forest, y la red neuronal RNN(Recurrent Neural Network) los cuales serán implementados en este proyecto.

Formulación del problema

El problema es que uno como usuario de las líneas de buses no sabe a qué hora llega un bus a una determinada parada ni cuánto tiempo tiene que esperar. En esta ciudad el tráfico vehicular es altamente congestionado e impredecible. Con este trabajo lograremos brindar información sobre rutas actuales y tiempos de traslado de los autobuses. Podríamos replantear rutas y observar bajo el algoritmo si es que son más eficientes. Cada persona necesita obtener más tiempo en su día para realizar más actividades, una manera de conseguir más horas en el día es pasar menos tiempo esperando en una parada el autobús.

Desde el enfoque de Ciencias de la Computación, el problema se puede abordar utilizando técnicas de Machine Learning (ML) y Big Data para analizar datos de geolocalización en tiempo real de los autobuses. Estas técnicas pueden ayudar a predecir el comportamiento del tráfico y mejorar la planificación del transporte público. Al aplicar algoritmos de ML, como regresión GBT (Gradient Boosted Tree), Random

Forest y redes neuronales recurrentes (RNN), se espera obtener modelos precisos para estimar los tiempos de llegada de los autobuses y predecir patrones de tráfico futuros.

Objetivos de investigación

Objetivo General:

El objetivo principal es mejorar el nivel de predicción de tiempos de llegada a estaciones de buses. Esto se realizará al testear distintos algoritmos ML que predigan el tiempo en el que un microbús llega a una parada. Se espera obtener uno o más modelos los cuales brinden mejores resultados que el ANN desarrollado en *(Dahl et al. 2014 [6])*. En el texto de *(Pałys et al., 2022 [2])* se menciona que el ANN fue utilizado para predecir tiempos de llegada en paradas sucesivas utilizando datos GPS. El perceptron de 3-capas alcanzó el mejor MAPE (*Mean Absolute Percentage Error*), al obtener un resultado de 18.3 %.

El documento también discute el uso de una ANN con datos históricos GPS y un automatic toll collection system para los datos. Se desarrolló un modelo híbrido ANN (HANN), con subredes separadas entrenadas para períodos de tiempo específicos, tales como días laborables días, fines de semana, y horas punta.

En resumen, el documento sugiere que modelos ANN, particularmente el modelo HANN, son relevantes y eficaces para tareas de predicción de autobuses. Sin embargo, el mejor modelo puede variar dependiendo en las circunstancias especificas y datos disponibles.

Entonces vamos a producir el ANN desarrollado en dicho estudio y lo compararemos con distintos modelos de deep learing y regresión. Con el objetivo de obtener un modelo el cual tenga mejores resultados en una o más circunstancias.

Objetivos Específicos:

Otros objetivos que se cumplirán varían entre

1. Obtener el dataset de forma óptima utilizando herramientas como Dask y Spark para poder realizar el entrenamiento de máquina y obtener data relevante.

    - Se reconoce la importancia de contar con un dataset de alta calidad y representativo para entrenar los modelos de Machine Learning.
    - Se busca utilizar herramientas como Dask y Spark, que permiten distribuir y procesar grandes volúmenes de datos de manera eficiente y exitosa, optimizando los tiempos de procesamiento.
    - El objetivo es obtener datos relevantes relacionados con la geolocalización de los autobuses, los horarios de salida y llegada, y otras variables que puedan influir en los tiempos de llegada.

2. Búsqueda de hiperparámetros de ML: Encontrar los hiperparámetros más relevantes para cada modelo ML utilizando distintos métodos, por ejemplo de bayesian optimization.

    - Se plantea la necesidad de buscar los mejores hiperparámetros para cada modelo de Machine Learning utilizado.
    - Se propone utilizar el método de optimización bayesiana para buscar los hiperparámetros más óptimos, maximizando el rendimiento y la precisión de los modelos.
    - El objetivo es encontrar la configuración adecuada de los modelos de Machine Learning que permita lograr la mayor precisión en la predicción de los tiempos de llegada.

3. Aplicación de sistemas ML en Lima: Propuesta de un modelo de predicción el cual se pueda aplicar a la realidad de lima.

    - Se busca proponer un modelo de predicción que sea aplicable y relevante a la realidad de Lima.
    - Se reconoce la importancia de considerar las particularidades del sistema de transporte público de Lima, como el tráfico congestionado y los patrones de tráfico impredecibles.
    - El objetivo es diseñar una propuesta que tenga en cuenta estos factores específicos de Lima y que pueda brindar estimaciones precisas y actualizadas de los tiempos de llegada de los autobuses en la ciudad.

Justificación

Lima se sitúa en la posición 19 entre los países con mayor congestión vehicular a nivel mundial *(tomtom, Lima traffic report, [1])*. En este estudio, nos centraremos en el análisis del Corredor Rojo, un servicio de autobús que experimenta una demora promedio de 2 horas y 15 minutos al recorrer la Avenida Javier Prado desde la Avenida La Marina hasta la Avenida Ceres, abarcando aproximadamente 22 km *(El Comercio, corredores complementarios [7])*. Esto implica que la velocidad promedio de desplazamiento es de tan solo 10 km/h, lo cual indica una clara oportunidad de mejora.

La situación se agrava cuando consideramos las esperas en las paradas de autobús, donde los usuarios no disponen de información precisa sobre los horarios de llegada. Al acceder a los sitios web del Metropolitano o el Corredor Rojo, encontramos la carencia de información sobre las horas de arribo de los autobuses a las estaciones. Actualmente, existe la aplicación Moovit *(Moovit [8])*, que calcula la frecuencia con la que un autobús pasa por una parada y ofrece un promedio de intervalo de tiempo

entre cada llegada. No obstante, este enfoque presenta limitaciones, ya que puede suceder que dos autobuses pasen en un intervalo de 2 minutos, seguidos de una espera de media hora, y la aplicación informe que el tiempo promedio de llegada es de 16 minutos, cuando en realidad los horarios de llegada varían significativamente.

En este estudio, proponemos cuantificar y proporcionar a los usuarios información precisa sobre los tiempos de llegada de los autobuses, con el objetivo de que puedan planificar su tiempo de manera más efectiva. Además, al analizar los tiempos de recorrido en diferentes rutas de la ciudad, los usuarios podrán encontrar rutas más óptimas para sus desplazamientos.

Asimismo, este trabajo podría servir como punto de partida para futuras investigaciones y desarrollos en el ámbito de la predicción del tráfico y la optimización del transporte público en Lima y en otras ciudades que enfrenten problemas similares de congestión vehicular.

En resumen, este estudio tiene el potencial de contribuir de manera significativa a la mejora de la eficiencia del transporte público en Lima, a la vez que abre nuevas vías de investigación y desarrollo en el campo de la predicción del tráfico mediante técnicas de ML.

CAPÍTULO I

REVISIÓN CRÍTICA DE LA LITERATURA

En esta sección, primero profundizaremos en dos trabajos de Bravo, *(Bravo, 2018 [3])* y *(Bravo y Torres, 2018 [9])*. La literatura será revisada en base al aprendizaje automatizado del transporte urbano, explorando el estado actual de la investigación y las oportunidades de mejora. Ambos trabajos están enfocados en ML Pymach y Sparkmach. En el paper 'Pymach y Sparkmach: Sistemas semiautomáticos de procesamiento de datos con dimensión variable usando Aprendizaje Automático y técnicas escalables', estas técnicas se utilizan para crear modelos personalizados que puedan predecir eventos futuros, como ubicaciones y frecuencias de accidentes de tráfico, localización, tiempos de espera de autobuses, consumo de combustible, entre otros *(Bravo, 2018 [3])*. Se trabajó con datos simulados y reales. Por otro lado, el *paper (Bravo y Torres, 2018 [9])* se centra principalmente en el modelado de ML, proporcionando un paquete de modelado y análisis semi-automatizado. Este paquete es Sparkmach, una herramienta diseñada para reducir los pasos involucrados en un roadmap de data science tradicional. Sparkmach se encarga de construir modelos base para problemas de clasificación y regresión. Sparkmach es una librería basada en Pymach, que aborda los mismos problemas que Sparkmach pero para conjuntos de datos medianos, de hasta 10 millones de filas. Sparkmach escala Pymach para manejar conjuntos de datos grandes utilizando computación distribuida a través de Apache Spark, un motor distribuido para el procesamiento de datos a gran escala que se encarga de resolver varios problemas de data science en un entorno de clúster. En el paper *(Bravo, 2018 [3])*, se utiliza la herramienta desarrollada en *(Bravo y Torres, 2018 [9])* para resolver problemas de predicción. Se realiza un experimento con los datos de los autobuses de Nueva York con el objetivo de predecir la posición

de los autobuses, su recorrido, velocidad y otras características, ya que estos contaban con sensores que monitorean estos valores. Se aplicó el algoritmo de Gradiente Descendiente a todos estos datos.

Ahora, se analizan los algoritmos de ANN y SVM. Se explorarán 3 estúdios, *(Bachu et al. 2016 [5])*, *(Yin et al., 2017 [4])* y *(Pałys et al., 2022 [2])*.Comenzaremos con el paper de Bachu 2016 *(Bachu et al. 2016 [5])*. Este proyecto tiene como objetivo proporcionar información precisa sobre los tiempos de llegada de los autobuses a los pasajeros. En este artículo, se compara el rendimiento de una metodología utilizando ANN y otra utilizando SVM. El estudio se realizó en Chennai, India. Se concluye que SVM logró tener mejores resultados, obteniendo un MAPE (Mean Absolute Percentage Error) menor que ANN y el promedio de los datos históricos. En el paper *(Yin et al., 2017 [4])*, se difiere con el paper de *(Bachu et al. 2016 [5])*, indicando inicialmente que el método de ANN es superior al de SVM. Por lo tanto, se realiza otro estudio comparando nuevamente estos métodos y se implementa una mejora en ellos. La mejora se basa en presentar tres factores que afectan el tiempo de llegada de los autobuses. El *paper* concluye que ambos métodos son funcionales, mostrando entre un 8 % y un 11 % de MAPE. En el paper de Pałys 2022, se observan distintas metodologías y se resalta el MAPE y la STD (desviación estándar) *(Pałys et al., 2022 [2])*. Se destacan ciertas características del método híbrido de ANN y SVM. Por ejemplo, se concluye que en ciudades con mayor población o en aquellas con infraestructura menos desarrollada, existen más demoras debido al tráfico, lo cual afecta la precisión de las predicciones. Además en esta lectura se rescata que el mejor modelo evaluado hasta la fecha es un ANN realizado investigado en *(Dahl et al. 2014 [6])*.

Siguiendo con el análisis actual, importante mencionar librerías ampliamente utilizadas en el ámbito del aprendizaje automático y preprocesamiento: Dask, pySpark,

Sklearn y Keras. Dask es una biblioteca de Python diseñada para trabajar con grandes conjuntos de datos de manera eficiente y escalable. Permite procesar datos que no caben en la memoria RAM de un solo ordenador al dividirlos en bloques más pequeños y distribuir el procesamiento en múltiples núcleos o incluso en múltiples máquinas *(Dask [10])*. PySpark es una biblioteca de Python que proporciona una interfaz para programar aplicaciones de procesamiento distribuido utilizando el framework Apache Spark. Spark es un motor de procesamiento de datos a gran escala que permite procesar grandes conjuntos de datos de manera eficiente y escalable en clústeres de computadoras. PySpark permite realizar tareas de procesamiento de datos distribuidos utilizando una API en Python, lo que facilita el procesamiento de datos a gran escala y la realización de operaciones complejas en paralelo. PySpark es ampliamente utilizado en aplicaciones de big data y machine learning, ya que proporciona capacidades para el procesamiento distribuido de algoritmos de aprendizaje automático en conjuntos de datos masivos *(PySpark [11])*. Keras es una biblioteca de Python de alto nivel para construir y entrenar redes neuronales. Proporciona una interfaz simple y fácil de usar que permite a los desarrolladores crear modelos de redes neuronales con pocas líneas de código. Keras está diseñado para ser modular y extensible, lo que facilita la construcción de modelos complejos con capas personalizadas y funciones de activación. Además, Keras se integra bien con otras librerías de aprendizaje automático, como TensorFlow, lo que permite aprovechar las capacidades de procesamiento distribuido y aceleración de hardware. Es una opción popular para tareas como clasificación, regresión, procesamiento de imágenes y procesamiento del lenguaje natural *(Keras [12])* . Sklearn es una librería en Python para utilizar modelos de ML ya implementados, permitiendo al desarrollador adaptarlos a sus necesidades modificando los hiperparámetros que mejor se ajusten al conjunto de datos *(Scikit-learn [13])*.

Gracias al estado del arte, aprendemos una estructura de análisis de datos para poder realizar operaciones de ML sobre grandes estructuras de datos. Nos brindan

técnicas de ML y nos enseñan cómo se aplican en ejemplos específicos. En el caso de *(Bravo, 2018 [3])*, no se muestran los resultados de la predicción ni para qué puede ser utilizada. Por lo tanto, en el siguiente trabajo, realizaremos un modelo de predicción de características de los autobuses utilizando distintos modelos como GBTRegression, LR (Regresión Lineal) y RNN (lsmt) sobre una base de datos amplia, y analizaremos los resultados. Además, realizaremos una optimización en la función para adaptarla mejor al caso de los autobuses.

Se llegó a la elección de utilizar un RNN (LSTM), regresión GBT (Gradient Boosted Trees) y repetir el mejor modelo hasta la fecha (ANN) para predecir los tiempos de llegada de autobuses a partir de las siguientes razones:

1. Capacidad de modelar dependencias temporales: Los RNN, especialmente las capas LSTM, son conocidos por su capacidad para capturar dependencias temporales en los datos. Al predecir los tiempos de llegada de los autobuses, es crucial tener en cuenta las secuencias de tiempo anteriores, ya que los patrones de tráfico y el flujo de pasajeros pueden cambiar con el tiempo. Los RNN con LSTM son capaces de aprender y recordar patrones a largo plazo, lo que los hace adecuados para modelar estos tipos de dependencias temporales.

2. Flexibilidad y adaptabilidad de GBT: La regresión GBT es un algoritmo de aprendizaje automático que combina múltiples árboles de decisión débiles en un modelo más fuerte. Los GBT pueden manejar una variedad de tipos de características y pueden capturar relaciones no lineales y complejas en los datos. Esto puede ser beneficioso al trabajar con datos de llegada de autobuses, ya que pueden haber múltiples factores que influyan en los tiempos de llegada, como el clima, la hora del día, los eventos especiales, entre otros.

3. Complementariedad y ensemble: Utilizar múltiples algoritmos para predecir los tiempos de llegada de los autobuses, como un RNN (LSTM), regresión GBT

y Random Forest, permite aprovechar las fortalezas de cada algoritmo. Cada algoritmo puede capturar diferentes aspectos de los datos y tener diferentes enfoques para modelar las relaciones subyacentes. Al combinar las predicciones de varios modelos en un ensemble, es posible obtener una predicción más precisa y robusta.

Es importante tener en cuenta que la elección de los algoritmos dependerá del conjunto de datos específico y de las características del problema. Luego de aplicar estos algoritmos, se realizará un análisis exploratorio de los datos y se experimentará con diferentes modelos para evaluar su rendimiento y seleccionar los más adecuados en función de las métricas de evaluación y las necesidades del proyecto.

CAPÍTULO II

MARCO TEÓRICO

En este trabajo buscamos aportar a la solución del problema de el tráfico en Lima. Se busca desarrollar un programa basado en ML que a partir de información de los vehículos se pueda predecir su estado futuro. Hemos observado en la revisión de literatura que ya existen métodos para solucionar este problema, utilizando una herramienta para un modelo ML que nos ayude a filtrar base de datos grandes (más de 10 millones de filas) y utilizar técnicas para la solución de este problema. En los trabajos anteriores que manejan la misma base de datos se plantea cómo es que se realizan estos pero no muestran los resultados de las predicciones ni se analiza esta información. Por lo que nos da a entender que esa solución no está confirmada, por ende existe una oportunidad de mejora al algoritmo empleado para dicha solución. Entonces se puede plantear un nuevo modelo a partir de la implementación en *(Bravo, 2018 [3])* y analizar los resultados. Luego de obtener dicha información se planteará un modelo de solución aplicado a la realidad de Lima. Ahora pasaremos a conceptos relevantes que serán implementados en el estudio.

2.1 Rutas y Subrutas

En este trabajo se están realizando dos tipos de modelos, unos que predicen subrutas y otros que predicen rutas. Una ruta está definida como una distancia de una parada inicial p_1 al resto de paradas del recorrido p_i. Una subruta es el recorrido de una parada p_i a la parada $p_i + 1$.

2.2 Medidas de rendimiento

RMSE (*Root Mean Square Error*): RMSE es una metrica utilizada para evaluar la precisión de un modelo de regresión. Mide la diferencia promedio entre los valores predichos y los valores reales en el dataset. La formula para RMSE es la siguiente:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(t_i - \hat{t}_i)^2}$$

Donde:

- n es el número de puntos de datos en el *dataset*,
- t_i es el valor real del tiempo para el punto de datos i, y
- $\hat{t}_i$ es el valor predicho del tiempo para el punto de datos i.

RMSE proporciona una medida del error promedio de la predicción del modelo. Un RMSE más pequeño indica mejor precisión, ya que ellos representan menores diferencias entre los datos predecidos y los valores reales.

R^2 (*R-squared*): R^2 Es una metrica estatística que representa la proporcion de la varianza de una variable dependiente que se predice a partir de variables independiantes dentro de un modelo de regresión. El rango va de 0 a 1, donde 0 indica que el modelo no explica nada a cerca de la variabilidad de las variables dependientes mientras 1 indica que el modelo predice perfectamente la variable dependiente. La formula para R^2 es:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(t_i - \hat{t}_i)^2}{\sum_{i=1}^{n}(t_i - \bar{t})^2}$$

Donde:

- n numero de puntos de datos en el *dataset*,

- t_i es el valor real de la variable dependiente en el punto de datos i,
- $\hat{t}_i$ es el valor predecido de la variable dependiente para el punto de datos i, y
- $\bar{t}$ es el la media de la variable dependiente.

R^2 provee un indicador de que tan bueno es el *fit* en un modelo de regresión. Un R^2 alto indica que gran proporción de la variabilidad de la variable dependiente se explica por las variables independientes, lo cual indica un mejor *fit*. Sin embargo, es importante tener en cuenta que R^2 por su cuenta no determina la validez o que tan apropiado es el modelo; otros factores tal como el contexto y supuestos de el modelo tienen que ser considerados.

Accuracy: Hace referencia a la precisión relativa del modelo en diferentes rangos de diferencia de tiempo. En lugar de utilizar una única medida de error promedio como el RMSE, se consideran varios umbrales de diferencia de tiempo. Estos umbrales representan intervalos en los que se considera que una predicción está "dentro del rango"si la diferencia entre el valor predicho y el valor real es menor o igual al umbral. Se itera sobre cada uno de los umbrales de tiempo definidos en la lista rangos. Para cada umbral, calcula la diferencia absoluta entre las predicciones $\hat{t}$ y los valores reales t. Luego, verifica qué elementos de esta diferencia están dentro del rango actual utilizando la expresión $diff <= r$, donde $diff$ es la diferencia absoluta y r es el umbral actual. Después, se calcula el porcentaje de datos que caen dentro del rango actual dividiendo el número de elementos dentro del rango por el tamaño total de las predicciones $\hat{t}$. Finalmente, se obtiene el porcentaje de datos dentro del rango y el valor del rango en segundos. Esta evaluación proporciona información adicional sobre la precisión del modelo en diferentes rangos de diferencia de tiempo. Cuanto mayor sea el porcentaje de datos dentro de un rango dado, mejor será la capacidad del modelo para predecir valores que caen dentro de ese rango específico.

2.3 Método optimisación bayesiana

La optimización bayesiana es una técnica utilizada para encontrar el máximo o mínimo global de una función objetivo costosa o computacionalmente costosa de evaluar. Se basa en la teoría de la probabilidad y utiliza el proceso de inferencia bayesiana para tomar decisiones secuenciales y explorar el espacio de búsqueda de manera eficiente. En esté caso se utilizará para encontrar los mejores hiperparámetros en la RNN. El objetivo de la optimización bayesiana es encontrar el conjunto óptimo de parámetros de entrada x (unidades LSTM, épocas y tamaño de *batch*) que minimicen o maximicen una función objetivo $f(x)$ (Error cuadrático medio (RMSE) del modelo LSTM.). En lugar de evaluar $f(x)$ exhaustivamente en todo el espacio de búsqueda, la optimización bayesiana utiliza un enfoque secuencial y construye un modelo probabilístico de $f(x)$ basado en las evaluaciones previas. El modelo probabilístico utilizado en la optimización bayesiana es conocido como proceso gaussiano (también llamado regresión gaussiana). El proceso gaussiano modela la función objetivo $f(x)$ como una distribución gaussiana sobre todas las posibles funciones que son consistentes con las evaluaciones observadas. El proceso gaussiano está completamente definido por su media y su matriz de covarianza. La optimización bayesiana busca encontrar el conjunto óptimo de parámetros de entrada x mediante la maximización de una función de adquisición. La función de adquisición es una medida que equilibra la exploración (buscar en áreas no evaluadas) y la explotación (buscar en áreas prometedoras) del espacio de búsqueda. La función de adquisición más comúnmente utilizada es la esperanza de mejora (Expected Improvement, EI). La fórmula de la función de adquisición de EI se define como:

$$EI(x) = \int max(f(x) - f(x^+), 0) \cdot p(f(x)|D)\, dx$$

donde x^+ es el punto más óptimo conocido hasta el momento, $p(f(x)|D)$ es la distribución posterior de la función objetivo basada en las evaluaciones previas D. La esperanza de mejora mide la expectativa de mejora con respecto al mejor valor encontrado hasta el momento. El proceso de optimización bayesiana se realiza de manera iterativa. En cada iteración, se actualiza el modelo probabilístico basado en las evaluaciones observadas y se calcula la función de adquisición para encontrar el próximo punto de evaluación. Este punto se evalúa en la función objetivo y se agrega a las observaciones para continuar el proceso iterativo.

2.4 Dask

Dask es una biblioteca de Python diseñada para trabajar con grandes conjuntos de datos de manera eficiente y escalable. Permite procesar datos que no caben en la memoria RAM de un solo ordenador al dividirlos en bloques más pequeños y distribuir el procesamiento en múltiples núcleos o incluso en múltiples máquinas. Un Dask Bag es una estructura de datos que representa una colección de elementos no estructurados que se pueden procesar en paralelo *(Dask [10])*.

2.5 Spark

Spark es un *framework* de computación distribuida. Está diseñado para procesar grandes cantidades de data distribuidas en los *clusters* de una computadora. De esta forma se logra un proceso rápido y eficiente de procesamiento de datos *(Apache Spark [11])*. Al utilizar apache Spark para realizar el procesamiento de forma veloz, existe un límite de qué algoritmos de entrenamiento podemos utilizar. Spark cuenta con librerías que realizan predicciones con el uso de funciones de regresión.

En este caso cuenta con: AFTSurvivalRegression, DecisionTreeRegressor, GBTRegressor, GeneralizedLinearRegression, IsotonicRegression, LinearRegression y RandomForestRegressor. Se ha decidido utilizar 2 modelos diferentes para la regresión: Linear Regression y Gradient Boosted Tree Regressor. Estos modelos son populares en machine learning y se utilizan comúnmente para problemas de regresión. A continuación, se presentan algunas razones por las cuales estos algoritmos podrían ser buenas opciones para este caso en particular: Gradient Boosted Tree Regressor: Gradient Boosted Trees también es un modelo de ensamblado que combina múltiples árboles de decisión para hacer predicciones. En contraste con Random Forest, cada nuevo árbol se entrena para corregir los errores del árbol anterior. Este enfoque permite la creación de modelos complejos que pueden capturar relaciones no lineales en los datos. En este caso estamos buscando un modelo que pueda capturar relaciones complejas no lineales entre las características y la variable objetivo, Gradient Boosted Tree Regressor puede ser una buena opción. Linear Regression: La regresión lineal es un algoritmo simple e interpretable que a menudo se utiliza como modelo de referencia para problemas de regresión. Es efectivo cuando hay una relación lineal entre las variables de entrada y la variable objetivo. En este caso, no sabemos si existe una relación lineal entre las características que tienes y la variable objetivo, la regresión lineal puede ser un buen punto de partida.

2.6 Keras

Keras es una biblioteca de aprendizaje profundo de código abierto escrita en Python. Proporciona una interfaz fácil de usar y de alto nivel para construir y entrenar modelos de aprendizaje profundo. Keras permite la creación de diversos tipos de redes neuronales, incluidas las redes neuronales recurrentes (RNN), utilizando diferentes tipos de capas, como las capas LSTM (Long Short-Term Memory). Una red neuronal recurrente (RNN) es un tipo de red neuronal que tiene conexiones recurrentes entre

las neuronas, lo que le permite procesar datos secuenciales. Una variante común de las RNN es la LSTM, que es una arquitectura de red neuronal recurrente mejorada que puede aprender y recordar dependencias a largo plazo en los datos de entrada.

2.7 Khipu

Khipu es un cluster proporsionado por la universidad de ingeniería y tecnología (UTEC). Este nos brinda un total de 80 cores disponibles. Este utiliza un Simple *Linux Utility of Resource Mangement* (SLURM), es el encargado de coordinar los recursos de todos los nodos del cluster y asignarlos de acuerdo la prioridad de los jobs, cantidad de recursos solicitados y cantidad de recursos disponibles. Slurm tiene un reparto de prioridad justo, donde cada job tiene una prioridad que depende de: a) los recursos usados por el usuario o grupo, b) la contribución del grupo al clúster y c) el tiempo en fila *(Khipu SLURM [14])*. Las dependencias del cluster se observan en la tabla 2.1.

Nodos CPU	**Especificaciones**
Nombre	nCPU
Procesador	Intel(R) Xeon(R) Gold 6130 CPU @2.10 GHz 16 cores por socket, 32 por nodo
Memoria	DRAM DDR4-1333 MHz, 128 GB por nodo
Almacenamiento	960 GB SSD
Red	Infiniband FDR MT4119

Tabla 2.1: Dependencias por nodo CPU en khipu

Software del sistema

- Sistema Operativo: CentOS Linux 7
- Message Passing Library: MPICH
- Compiladores: Intel, GCC

- Job Scheduler: SLURM
- Manejo de software: Módulos de ambiente

2.8 Comparando Nueva York y Lima

En esta sección nos enfocamos en encontrar similaridades entre Nueva York y Lima. Aprendemos el porqué de realizar un estudio con data de buses de Nueva York es relevante para realizar un modelo en Lima. En esta sección exploramos similitudes en densidad de gente, cantidad usuarios de buses, estructura de la ciudad, etc. Ambos Lima y Nueva York cuentan con un total de 8 millones de habitantes. Según el paper *(Pałys et al., 2022 [2])* mientras una ciudad tiene más densidad de población se encuentran las mismas inaccuracies que en una ciudad con baja infraestructura.por ende al tener una ciudad como Nueva York con 3 veces más la densidad poblacional de lima pero mejor infraestructura podemos esperar resultados similares en la predicción de tiempos de llegada de buses.

2.9 Módulos de trabajo

En la Figura 2.1. Se pueden apreciar los distintos módulos que se van a realizar en este proyecto y cómo se enlazan.

1. Filtrado de datos: En está sección se implementa Dask y pySpark sobre la data recuperada de los buses MTA, y se filtra en forma óptima para ser utilizada por el modelo.
2. Entrenamiento de ML: En esta sección se utilizan distintos algoritmos de predicción como GBT, RandomForest y RNN para entrenar nuestro modelo y predecir los datos. Se hacen ajustes en el algoritmo para mejorar la precisión

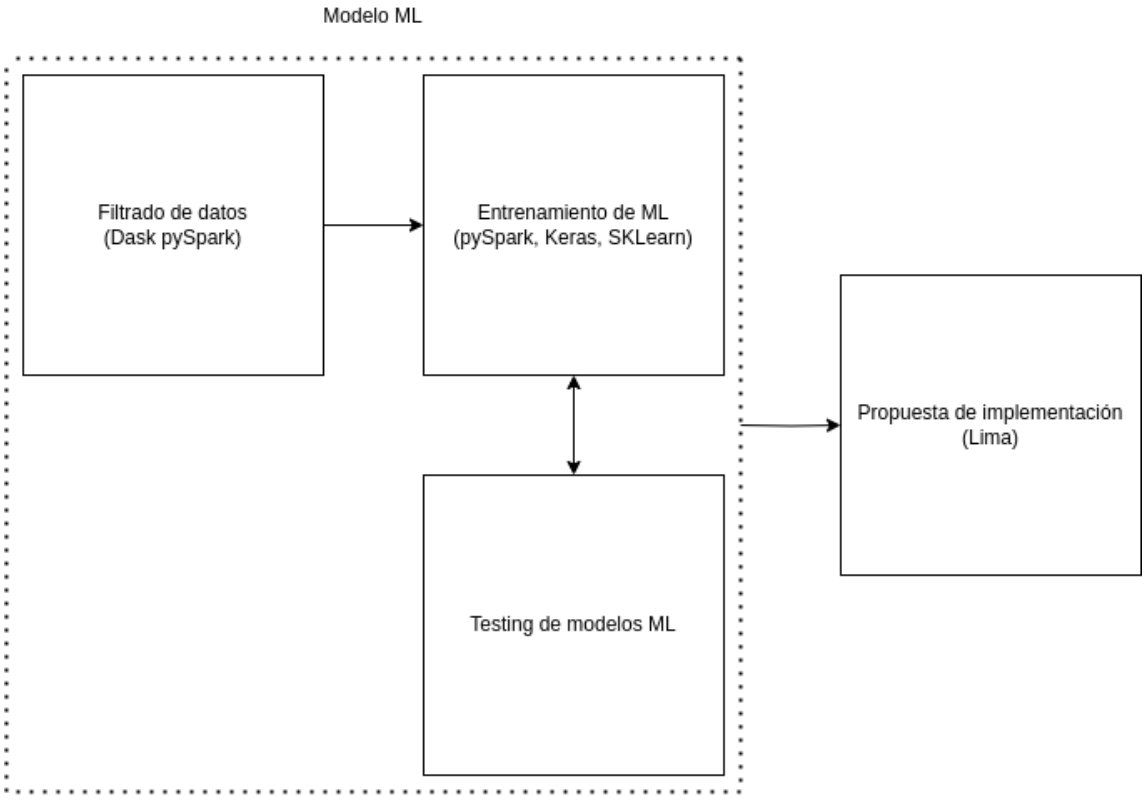

FIGURA 2.1: Diagrama de Módulos de trabajo *(Autoría propia)*

y se realiza una búsqueda de los mejores hiper-parámetros para obtener los mejores resultados.

3. Testing de modelos ML: Realizamos distintas pruebas con los modelos entrenados para obtener resultados de desempeño.

4. Propuesta de implementación (Lima): A partir de los Modelos ML evaluados, se generará una propuesta donde se detallan que herramientas he información se necesitan para poder implementar este método de predicción sobre la realidad actual de Lima.

CAPÍTULO III

MARCO METODOLÓGICO

3.1 Preproceso de datos con Dask

Utilizamos Dask para realizar un filtro inicial sobre el dataset de autobuses para poder ingresarla a los modelos de ML. Es importante tener data ordenada y numeral para el modelo, ya que estos funcionan con este tipo de información.

Para empezar se descargan los archivos de texto del *(MTA, Bus Historical Data [15])* que contienen toda la información que se va a utilizar para entrenar y hacer *testing* del modelo.

Estos son un total de 89 archivos, que van del día 2014-08-01 hasta el día 2014-10-31. Con archivos no registrados en los días 2014-09-21, 2014-10-06. Estos archivos estan guardados en formato .txt y ocupan un total de 64419 MB de memoria. Además los registros cuentan con el siguiente *schema*, el cual se puede observar en la **Tabla 3.1**.

Se empieza descargando cada uno de los archivos de texto en un cluster, en este caso se almacenarán en Khipu.

Para el preprocesamiento el objetivo es tener data ordenada la cual pueda ser luego procesada por pySpark y entrenada en los modelos.

La estructura del procesamiento es la siguiente:

Procesamos cada archivo con dask, almacenandolo primero en un dask dataframe. Se procesarán 4 archivos a la vez, observar **Figura 3.1**, este procedimiento se repite hasta procesar los 89 archivos.

Tabla 3.1: *Schema* de data a inicial de autobuses

Column Name	**Description**	**Data Type**
latitude	Latitude received from on-board GPS Unit (WGS84)	float
longitude	Longitude received from on-board GPS Unit (WGS84)	float
time_received	Time (in UTC) of message receipt by server	string
vehicle_id	3 or 4-digit bus number	string
distance_along_trip	Distance along trip (in meters)	float
inferred_direction_id	Direction ID from GTFS trips.txt	int
inferred_phase	The phase of the bus in its duty cycle; either IN_PROGRESS or LAYOVER_DURING	string
inferred_route_id	Route ID the bus was inferred to be serving	string
inferred_trip_id	A GTFS trip_id representing the inferred stopping pattern for the bus at the given time	string
next_scheduled_stop_distance	The distance (in meters) of the bus from the next scheduled stop	string
next_scheduled_stop_id	The GTFS stop_id of the next stop the bus will serve	string

De las columnas definidas en **Tabla 3.1** se obtienen las más relevantes, las cuales en este caso son:

'time_received', 'vehicle_id', 'distance_along_trip', 'inferred_phase', 'next_scheduled_stop_distance', 'next_scheduled_stop_id'

3.1.1 Preprocesamiento por archivo

Para empezar, se cargan las columnas relevantes y se realiza el preprocesamiento. En la **Figura 3.2**, se observa un diagrama de flujo del preprocesamiento en cada archivo.

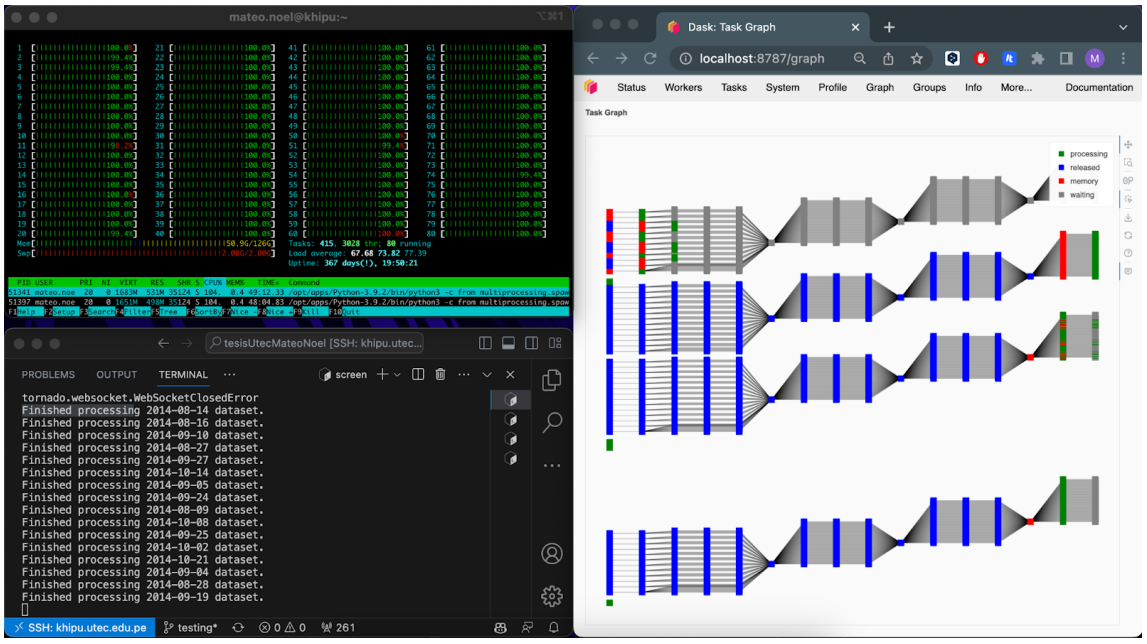

FIGURA 3.1: Preprocesamiento con dask

1. Primero se divide en 80 particiones, ya que trabajamos con 80 cores gracias a khipu, cada core va a trabajar sobre una partición.

2. Luego se realiza el primer filtro, se filtran todos los buses que no están operando en el momento y se eliminan todas las filas que tienen un valor null.

3. Luego de eso nos deshacemos de la columna 'inferred_phase'.

4. Después, agrupamos por 'vehicle_id' el dask dataframe y operamos sobre cada vehículo. Ahí es donde se realiza el primer shuffle. Cuando un worker termina con su partición en vez de esperar este va a empezar a trabajar sobre una partición de otro archivo txt que está siendo procesado.

5. Ahora, por cada grupo de 'vehicle_id' se realiza la siguiente función, sort & calc. sort & calc obtiene la hora en la cual un vehículo pasa por una parada.

6. Después se estandarizan las distancias entre paradas, se agrupa por la columna 'trip' y se realiza el algoritmo dist_fix.

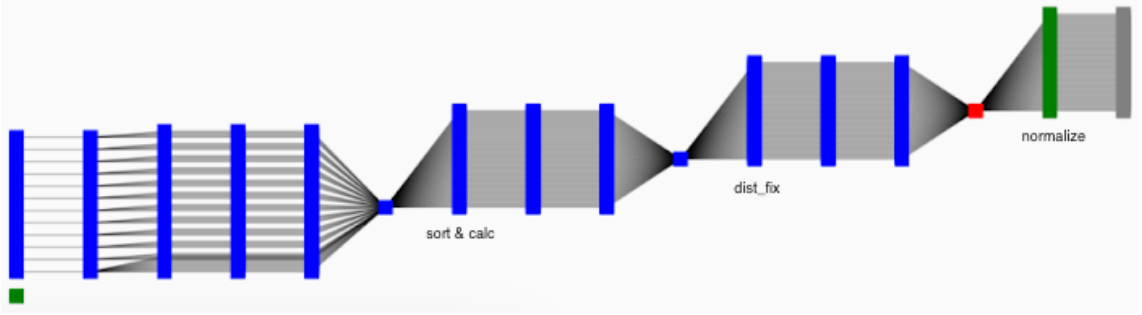

FIGURA 3.2: Preprocesamiento con dask por archivo

7. Se agrupa por la columna 'vehicle_id' y se realiza el algoritmo normalize.

8. Finalmente, Luego de tener todos los dataframes por vehicle_id estos se guardan en un parquet con el nombre de la fecha procesada.

3.1.2 sort & calc

Algoritmo 1: sort and calc

Resultado: DataFrame *data*

Input : DataFrame *partition*

Output: DataFrame *data*

$partition \leftarrow sort(partition)$; $partition \leftarrow removeDuplicates(partition)$;
$data \leftarrow createDataFrame()$; $first \leftarrow true$; $skip\_count \leftarrow 0$;
para $i \in range(length(partition) - 1)$ **hacer**
    $current \leftarrow partition[i]$; $next \leftarrow partition[i + 1]$;
    si $next\_distance < current\_distance$ *or* $dist\_two\_times \neq dist\_next\_stop$ *or* $skip\_count > 0$ *or* $dist\_next\_stop\_from\_current > dist\_two\_times$ *or* $speed < 0.1$ *or* $speed > 20$ **entonces**
        $first \leftarrow true$; $skip\_count \leftarrow \max(skip\_count - 1, 0)$;
    en otro caso
        si $first$ **entonces**
            $prev\_time\_to\_stop, prev\_distance, prev\_along\_distance, next\_stop\_from\_prev \leftarrow calculateArrivalTimeAndSpeed()$; $first \leftarrow false$;
        en otro caso
            $time\_to\_stop, arrived\_time, dist\_two\_stops \leftarrow calculateOtherArrivalTimeAndSpeed()$;
            si $dist\_two\_stops \geq 100$ *and* $dist\_two\_stops \leq 2000$ **entonces**
                $new\_row \leftarrow [next\_stopfromprev + current, dist, dateofprevtime, timeofprevtime, arrivedtime]$;
                $data \leftarrow append(data, new\_row)$;
                $prev\_time\_to\_stop, prev\_distance, prev\_along\_distance, next\_stop\_from\_prev \leftarrow updatePrevValues()$;
            fin
        fin
    fin
fin
$data \leftarrow insert(data, 'vehicle\_id', partition['vehicle\_id'][0])$;
devolver *data*;

El **Algoritmo 1** ‘sort_and_calc‘, realiza una serie de operaciones en un grupo de ’vehicle_id’ para procesar y analizar datos de viaje.

En la **Figura 3.3** se pueden observar un gráfico que muestra que valores representa cada variable.

1. Primero, la función ordena y limpia los datos de la partición. Elimina los duplicados y reasigna los índices del DataFrame para garantizar la consistencia en el conjunto de datos.

2. Después, se inicializa un nuevo DataFrame ‘data‘, junto con una variable booleana ‘first‘ y un contador ‘skip_count‘. Estas serán utilizadas posteriormente durante el procesamiento de los datos.

3. La función luego comienza a iterar a través de cada fila del DataFrame ‘partition‘. Para cada par de entradas sucesivas en el DataFrame, realiza varias comparaciones y cálculos.

4. Si la distancia del segundo elemento (‘next‘) es menor que la del elemento actual (‘current‘), el ciclo se reinicia y se omite el par actual. También se verifica si la diferencia de distancia entre los dos tiempos es igual a la diferencia de distancia hasta la siguiente parada. Si no es así, el ciclo se reinicia.

5. Se calcula la velocidad entre las dos entradas y si esta velocidad está fuera del rango aceptable (menor que 0.1 o mayor que 20), el ciclo se reinicia.

6. Si estamos en la primera parada de la ruta, la función calcula la hora de llegada y la velocidad a la próxima parada y luego pasa a la siguiente iteración.

7. Si no estamos en la primera parada, la función calcula la hora de llegada a la próxima parada y la distancia entre las dos paradas. Si la distancia está fuera del rango aceptable (menor que 100 o mayor que 2000), el ciclo se reinicia. De lo contrario, la información se agrega al DataFrame ‘data‘.

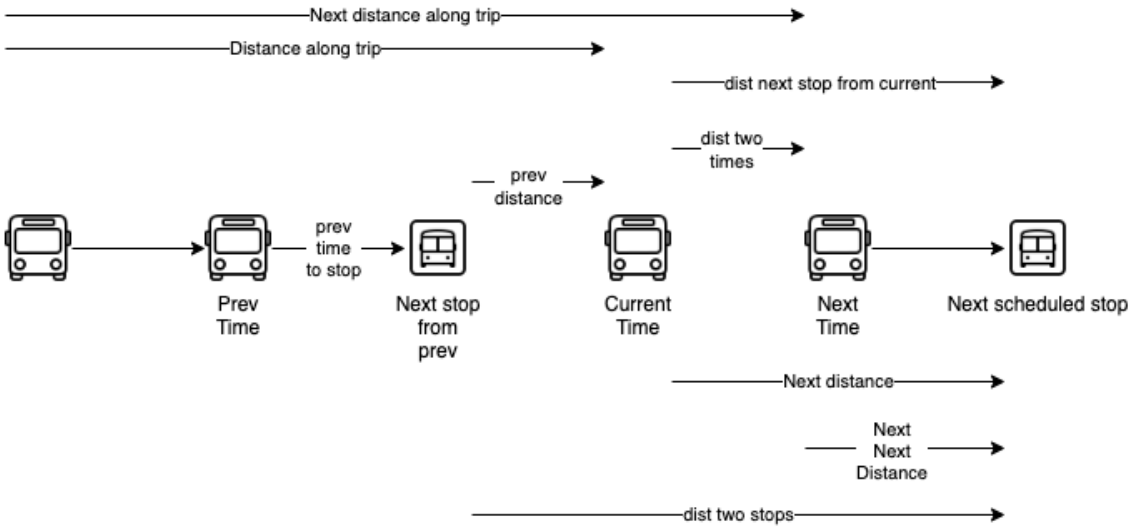

FIGURA 3.3: Diagrama de variables del algoritmo sort & calc

8. Finalmente, la función inserta el 'vehicle_id' en el DataFrame 'data' y lo retorna. Este DataFrame contiene datos de viaje limpios y procesados, listos para el análisis posterior.

3.1.3 dist_fix

Después de obtener los resultados se estandarizan las distancias entre paradas se agrupa por la columna 'trip' y se realiza el **Algoritmo 2**.

Algoritmo 2: dist_fix

Resultado: DataFrame *df*

Input : DataFrame *df*

Output: DataFrame *df*

$df['distance'] \leftarrow round(df['distance'], -1)$;

$mode \leftarrow mode(df['distance'])$;

$df \leftarrow df[(df['distance'] > mode - 10)$ and $(df['distance'] < mode + 10)]$;

$df['distance'] \leftarrow mode$;

devolver *df*;

1. La función dist_fix realiza una serie de operaciones de manipulación de datos en un DataFrame de pandas, df, específicamente en la columna 'distance'. Primero, redondea los valores en esta columna al múltiplo de 10 más cercano. Luego, calcula la moda (el valor que ocurre con más frecuencia) de los valores en esta columna.

2. A continuación, filtra el DataFrame para que solo contenga las filas donde el valor en la columna 'distance' está dentro de un rango de +/- 10 unidades de la moda. Es decir, elimina las filas donde 'distance' es menor a mode - 10 o mayor a mode + 10.

3. Por último, actualiza todos los valores en la columna 'distance' al valor de la moda. La función devuelve el DataFrame modificado.

4. Al terminar con la función dist fix se realiza el último algoritmo, agrupamos nuevamente por vehicle id y realizamos la siguiente operación.

3.1.4 normalize

Algoritmo 3: normalize parte 1

Resultado: DataFrame *data*

Input : DataFrame *data*

$data['date'] = \text{to\_datetime}(data['date'], format = '\ \%Y - \%m - \%d');$

$data = \text{insert}(0,'month', data['date'].dt.month);$

$data = \text{insert}(0,'day\_of\_month', data['date'].dt.day);$

$data = \text{insert}(0,'day\_of\_week', data['date'].dt.dayofweek);$

$data = \text{drop}('date', axis = 1);$

$data['exit\_time'] = \text{time\_to\_seconds}(data['exit\_time']);$

$data['arrive\_time'] = \text{time\_to\_seconds}(data['arrive\_time']);$

$data = \text{insert}(0,'target\_stop','');$

$data = \text{insert}(0,'exit\_stop','');$

$data['exit\_stop'] = \text{Series}(\text{split}('MTA\_', data['trip'], 1));$

$data['target\_stop'] = \text{Series}(\text{split}('MTA\_', data['trip'], 2));$

$data['exit\_stop'] = \text{astype}(data['exit\_stop'], int);$

$data['target\_stop'] = \text{astype}(data['target\_stop'], int);$

$data = \text{drop}('trip', axis = 1);$

$mask = \sim data['exit\_stop'].isin(data['target\_stop']);$

$routes = \{elem : \{elem\} \text{ for } elem \text{ in } data['exit\_stop'][mask]\};$

$data = \text{insert}(0,'first\_stop', None);$

$data = \text{insert}(0,'total\_distance', 0.0);$

Algoritmo 4: normalize parte 2

```
loop_count = 0;
mientras data['first_stop'].isna().any() and loop_count < 100 hacer
    para route_name, stops in routes.items() hacer
        mask = data['exit_stop'].isin(stops) and data['first_stop'].isna();
        mask2 = data['exit_stop'].isin(stops);
        total_distance = data.loc[mask2,'total_distance'].max();
        new_stops = data.loc[mask,'target_stop'].to_list();
        data.loc[mask,'total_distance'] = data.loc[mask,'distance'] + total_distance;
        data.loc[mask,'first_stop'] = route_name;
        routes[route_name] = routes[route_name]|set(new_stops);
    fin
    loop_count = loop_count + 1;
fin
si loop_count == 100 entonces
    data = dropna(subset = ['first_stop']);
fin
devolver data;
```

El **Algoritmo 3 y 4** toma un DataFrame data y realiza varias operaciones de manipulación de datos.

1. Primero, convierte la columna 'date' al formato datetime, luego extrae las características adicionales de la fecha como el mes, día del mes y día de la semana y las añade al DataFrame.

2. Después, elimina la columna 'date' y convierte las columnas de tiempo (exit time y arrive time) a segundos desde la medianoche.

3. Luego, añade las columnas 'target_stop' y 'exit_stop', separa los elementos de la columna 'trip' y los coloca en las nuevas columnas. Después, convierte 'exit_stop' y 'target_stop' a tipo de dato numérico.

4. Elimina la columna 'trip' y luego crea una máscara para identificar las paradas iniciales de cada ruta. Luego crea un diccionario donde las claves son las paradas iniciales y los valores son conjuntos que contienen la parada inicial.

5. Posteriormente, añade las columnas 'first_stop' y 'total_distance' al DataFrame.

6. A continuación, inicia un bucle para llenar estas columnas. Dentro de este bucle, para cada conjunto de paradas en el diccionario, identifica las filas del DataFrame donde 'exit_stop' está en ese conjunto de paradas y 'first_stop' aún no tiene un valor.

7. Para estas filas, añade la parada de destino a la lista de paradas y actualiza 'first_stop' y 'total_distance'. Si el bucle no ha llenado todos los valores de 'first_stop' después de 100 iteraciones, elimina las filas con valores faltantes.

8. Finalmente, la función devuelve el DataFrame modificado.

9. Luego de tener todos los dataframes por vehicle_id estos se guardan en un parquet con el nombre de la fecha procesada.

3.2 Preproceso de datos con pySpark

Algoritmo 5: Preprocesamiento con PySpark parte 1

Resultado: parquet *data*

Input : parquet *data*

```
import pyspark.sql.functions as F
from pyspark.sql.window import Window
compressed_data = compress_files_to_parquet();
data = spark.read.parquet('compressed_data');
all_stops = set(data.select('exit_stop').distinct().rdd.flatMap(lambda x: x).collect());
all_stops.union(set(data.select('target_stop').distinct().rdd.flatMap(lambda x: x).collect()));
vehicle_ids = set(data.select('vehicle_id').distinct().rdd.flatMap(lambda x: x).collect());
mapping_s = stop: new_value for new_value, stop in enumerate(all_stops);
exit_stop_mapping = F.udf(lambda stop: mapping_s[stop]);
target_stop_mapping = F.udf(lambda stop: mapping_s[stop]);
first_stop_mapping = F.udf(lambda stop: mapping_s[stop]);
mapping_v = vehicle: new_value for new_value, vehicle in enumerate(vehicle_ids);
vehicle_mapping = F.udf(lambda vehicle: mapping_v[vehicle]);
data = data.withColumn('exit_stop', exit_stop_mapping(data['exit_stop']).cast('int'));
.withColumn('target_stop', target_stop_mapping(data['target_stop']).cast('int'));
.withColumn('first_stop', first_stop_mapping(data['first_stop']).cast('int'));
.withColumn('vehicle_id', vehicle_mapping(data['vehicle_id']).cast('int'));
.withColumnRenamed('arrive_time', 'label');
```

Luego de realizar el preproceso con Dask, se realiza el preproceso con pyspark. Se efectúa el **Algoritmo 5 y 6** sobre el dataset, es descrito en las siguientes lineas. Primero se comprimen todos los archivos en un solo parquet y se ejecuta el siguiente código. Se utiliza la biblioteca PySpark para realizar el preprocesamiento de datos en un entorno distribuido utilizando Spark.

Algoritmo 6: Preprocesamiento con PySpark parte 2

```
window_spec = Window.partitionBy('vehicle_id').orderBy('month', 'day_of_month',
  'exit_time');
data = data.withColumn('first_time', F.when(data['first_stop'] !=
  F.lag(data['first_stop']).over(window_spec), data['exit_time']).otherwise(None));
data = data.withColumn('window_values', F.when(data['first_stop'] !=
  F.lag(data['first_stop']).over(window_spec), data['exit_stop']).otherwise(None));
data = data.withColumn('first_time', F.last(data['first_time'],
  ignorenulls=True).over(window_spec));
.withColumn('window_values', F.last(data['window_values'],
  ignorenulls=True).over(window_spec));
data = data.filter(data['first_stop'] == data['window_values']);
data = data.drop('window_values');
data = data.dropna(subset=['first_time']);
data = data.select('first_time', 'vehicle_id', 'month', 'day_of_month', 'day_of_week', 'first_stop',
  'exit_stop', 'target_stop', 'total_distance', 'distance', 'label');
data = data.filter((data['label'] ¿= 25200) & (data['label'] ¡= 39600));
data = data.withColumn('travel_time', (data['label'] - data['first_time']));
.withColumn('speed', (data['distance'] / data['travel_time']));
.withColumn('total_speed', (data['total_distance'] / data['travel_time']));
data = data.withColumn('speed', F.round(data['speed'], 2));
.withColumn('total_speed', F.round(data['total_speed'], 2));
.drop('travel_time');
data = data.filter(data['speed'] ¿ 1);
data.write.parquet('unNormalized_dataset/dataset');
```

1. Obtiene los valores únicos de las columnas 'exit_stop' y 'target_stop' utilizando las funciones 'distinct()' y 'flatMap()'. Luego, combina estos valores únicos en una lista utilizando conjuntos ('set') y asigna el resultado a la variable 'all_stops'.

2. Obtén los valores únicos de la columna 'vehicle_id' de manera similar y asígnalos a la variable 'vehicle_ids'.

3. Crea un mapeo de los valores antiguos a los nuevos valores para las paradas utilizando un diccionario ('mapping_s'). Los nuevos valores se generan utilizando la función 'range()'.

4. Crea funciones definidas por el usuario (UDFs) utilizando ‘udf()‘ para aplicar el mapeo a cada columna correspondiente. Se definen UDFs para las columnas ’exit_stop’, ’target_stop’ y ’first_stop’.

5. Crea otro mapeo de los valores antiguos a los nuevos valores para los vehículos utilizando un diccionario (‘mapping_v‘).

6. Crea UDFs para las columnas "vehicle_idütilizando ‘udf()‘ y el mapeo de vehículos.

7. Aplica los cambios al DataFrame ‘data‘ utilizando el método ‘withColumn()‘ de Spark. Se aplican las transformaciones correspondientes para cada columna:

   - Se aplica la UDF ‘exit_stop_mapping‘ a la columna ’exit_stop’ y se convierte a tipo entero.
   - Se aplica la UDF ‘target_stop_mapping‘ a la columna ’target_stop’ y se convierte a tipo entero.
   - Se aplica la UDF ‘first_stop_mapping‘ a la columna ’first_stop’ y se convierte a tipo entero.
   - Se aplica la UDF ‘vehicle_mapping‘ a la columna ’vehicle_id’ y se convierte a tipo entero.
   - Se cambia el nombre de la columna ’arrive_time’ a ’label’.

El código carga datos desde un archivo Parquet, realiza transformaciones en el DataFrame utilizando UDFs y funciones de Spark, ordena el DataFrame.

8. Crea una especificación de ventana (‘window_spec‘) utilizando la clase ‘Window‘ de PySpark. Define la partición por la columna "vehicle_idz el orden por las columnas "month", "day_of_monthz .exit_time.en orden ascendente.

9. Añade la columna "first_time.ªl DataFrame (‘df‘) utilizando el método ‘withColumn()‘ de Spark. La columna se calcula utilizando la función ‘when()‘ de PySpark. Compara el valor de la columna "first_stopçon el valor anterior de la columna "first_stopütilizando la función ‘lag()‘. Si son diferentes, asigna el valor de la columna .exit_time.ª la columna "first_time". De lo contrario, asigna ‘None‘ a la columna "first_time". Esta operación se realiza dentro de la ventana especificada por ‘window_spec‘.

10. Añade la columna "window_values.ªl DataFrame de manera similar a la columna "first_time". Calcula el valor de la columna .exit_stopçuando hay un cambio en el valor de la columna "first_stop", y asigna ‘None‘ de lo contrario.

11. Rellena los valores nulos en la columna "first_timeçon el último valor no nulo utilizando la función ‘last()‘ de PySpark y la opción ‘ignorenulls=True‘. Esta operación se realiza dentro de la ventana especificada por ‘window_spec‘.

12. Rellena los valores nulos en la columna "window_values"de manera similar a la columna "first_time".

13. Filtra el DataFrame (‘df‘) manteniendo solo las filas donde el valor de la columna "first_stop.es igual al valor de la columna "window_values". Esto se hace para eliminar las filas que no coinciden en las paradas iniciales.

14. Elimina la columna "window_values"del DataFrame.

15. Elimina las filas con valores nulos en la columna "first_timeütilizando el método ‘dropna()‘ de PySpark y especificando la columna de interés.

16. Selecciona las columnas en un orden específico, colocando la columna 'first time' como la primera columna del DataFrame. Esto se realiza utilizando el método ‘select()‘ de PySpark y el acceso a las columnas mediante la indexación.

Este código realiza operaciones de ventana en el DataFrame cargado desde un archivo Parquet. Se calculan las columnas "first_timez "window_valuesütilizando condiciones y funciones de ventana, se filtran y rellenan los valores nulos según ciertas condiciones.

17. Se filtran los datos utilizando la función filter() y la condición ($col("label") >= 25200)\&(col("label") <= 39600$). Esto selecciona solo los datos que cumplen con el rango de valores especificado en la columna "label".

Finalmente se realiza el último filtro y la información estará lista para ser administrada por los modelos ML

18. Creación de nuevas columnas: Crea nuevas columnas "travel_time", "speed", y "total_speed"basadas en los cálculos realizados con los valores de otras columnas.

19. Ajuste de precisión y eliminación de columna: Redondea los valores en las columnas "speedz "total_speed.[a] dos decimales y luego elimina la columna "travel_time".

20. Filtrado: Filtra las filas donde el valor en la columna "speed.[es] mayor que 1.

21. Almacenamiento: Escribe el DataFrame resultante en el directorio ünNormalized_dataset/dataset.[en] formato Parquet

Gracias a este código podemos visualizar la data, revisar que todo está en orden y estamos listos para entrenar. La base de datos final tendrá la siguiente forma, la cual se observa en la tabla 3.2, está tiene las siguientes cualidades:

```
Number of vehicles: 5647
```

```
Number of stops: 14834
Number of first_stops: 14063
Minimum distance: 100 m
Maximum distance: 2000 m
Maximum total distance: 39920 m
Minimum exit time: 24614 (6:50:14 am)
Maximum label (arrive time): 39571 (10:59:31 am)
Number of rows: 4512672
```

3.3 Entrenando los modelos ML

En esta sección se explican los algoritmos de entrenamiento de los distintos modelos ML que serán comparados. Después del preprocesamiento, el conjunto de datos utilizado contiene 5 variables de entrada para subrutas y rutas. Estas variables son:

Para subrutas:

- "day_of_week": día de la semana.
- "$exit\_time$": hora de salida de exit_stop.
- "$distance$": distancia entre exit_stop y target_stop. "$exit\_stop$": parada de salida. "$target\_stop$": parada objetivo.

Para rutas:

- "day_of_week": día de la semana.
- "$first\_time$": hora de salida de first_stop.
- "$total\_distance$": distancia entre fist_stop y target_stop.

Tabla 3.2: *Schema* de data final de autobuses

Column Name	**Description**	**Data Type**
vehicle_id	entero de 1 a v donde v es la cantidad de autobuses únicos	int
month	entero de 1 a 12 el cual representa 1 mes del año	int
day_of_month	entero de 1 a 31 el cual representa 1 día del mes	int
day_of_week	entero de 1 a 7 el cual representa 1 día de la semana	int
first_stop	entero el cual identifica una parada, esta no es un punto de llegada para ni un viaje	int
exit_stop	entero el cual identifica la última parada por la cual un autobús se detuvo	int
target_stop	entero el cual identifica la parada donde el autobús se dirige	int
total_distance	distancia recorrida total de un autobús desde la primera parada en metros	int
distance	distancia entre exit_stop y target stop en metros	int
first_time	hora de salida del autobús del first_stop en segundos a partir de la media noche	int
exit_time	hora de salida del autobús del exit_stop en segundos a partir de la media noche	int
arrive_time	hora de llegada del autobús al target_stop en segundos a partir de la media noche	int

- "$first\_stop$": primera parada.
- "$target\_stop$": parada objetivo.

La variable de salida en ambos casos es "*label*" que también es conocido como $arrive\_time$", el tiempo el cual el bus llega a la parada objetivo.

3.4 Modelo de regresión GBT

Primero se empezará con el desarrollo del modelo de regresión GBT. Para esto se desarrolla un script en Python el cual utiliza la herramienta spark.ml para el desarrollo del modelo. El modelo se crea con 'GBTRegressor' del módulo 'pyspark.ml.regression'. Se utiliza un *scaler* en este caso de 'StandardScaler' del módulo 'pyspark.ml.feature', para aplicar el escalado a los valores de entrada. En este caso se utilizará el 80 % de los datos para el entrenamiento del modelo. Se instancia un objeto 'GBTRegressor' con las siguientes cualidades iniciales (maxDepth: int = 5, maxBins: int = 32, minInstancesPerNode: int = 1, lossType: str = "squared", maxIter: int = 20, stepSize: float = 0.1, impurity: str = "variance") y se entrena el modelo.

Luego de entrenar el modelo con el 80 % del dataset se muestran los resultados del 20 % faltante:

```
GBT subroutes:
98.03% of data is within a range of 600 seconds
91.90% of data is within a range of 300 seconds
43.18% of data is within a range of 60 seconds
22.39% of data is within a range of 30 seconds
R-squared = 0.9974
RMSE = 175.7534

GBT routes:
86.74% of data is within a range of 600 seconds
64.52% of data is within a range of 300 seconds
16.57% of data is within a range of 60 seconds
8.35% of data is within a range of 30 seconds
```

```
R-squared = 0.9355
RMSE = 175.7534
```

3.5 Modelo de regresión lineal

Al igual que en el caso del modelo de regresión GBT se utiliza spark.ml para el desarrollo del modelo, en este caso 'LinearRegression' de el módulo 'pyspark.ml.regression para armar el modelo y la data se escala de la misma forma. En este caso también se utilizará el 80 % de los datos para el entrenamiento del modelo. Se instancia el objeto 'LinearRegression' con las siguientes cualidades iniciales (maxIter: int = 100, tol: float = 1e-6, loss: str = "squaredError", epsilon: float = 1.35)

Los resultados del entrenamiento son los siguientes:

```
LR subroutes:
100.00% of data is within a range of 600 seconds
99.98% of data is within a range of 300 seconds
95.47% of data is within a range of 60 seconds
76.24% of data is within a range of 30 seconds
R-squared = 0.9999
RMSE = 30.8276

LR routes:
35.38% of data is within a range of 600 seconds
17.20% of data is within a range of 300 seconds
3.38% of data is within a range of 60 seconds
1.69% of data is within a range of 30 seconds
R-squared = 0.5982
RMSE = 2174.1705
```

Para subrutas, el modelo de regresión lineal logra un ajuste casi perfecto, ya que el R2 es 0.9999, lo que indica que el modelo explica el 99 % de la variabilidad en los datos. Además, el RMSE es bajo, lo que indica un buen ajuste entre las etiquetas y las predicciones.

Sin embargo, para las rutas, el rendimiento del modelo de regresión lineal es inferior. El R2 es de 0.5684, lo que indica que el modelo explica aproximadamente el 56.84 % de la variabilidad en los datos. El RMSE también es más alto, lo que indica que hay una discrepancia más significativa entre las etiquetas y las predicciones.

En general, estos resultados sugieren que el modelo de regresión lineal funciona mejor para subrutas en comparación con rutas. Para rutas, puede ser necesario considerar otros modelos de regresión más sofisticados o ajustar los datos de entrada de alguna manera para mejorar el rendimiento del modelo.

3.6 Modelo RNN

Antes de realizar la experimentación se realizó el *tunning* de hiperparámteros a partir de optimisación bayesiana sobre los primeras 1000 filas del *dataset*, los resultados mostraron que en un rango de [10-100] épocas, [16-256] tamaño de batch y [32-1024] se optenían mejores resultados en 100 épocas, un batch de 256 y 1024 unidades de LSTM. A partir de esto se desidió realizar el modelo inicial del RNN con los hiperparámetros mencionados.

El código utiliza la librería Keras para entrenar un modelo de red neuronal recurrente (RNN) con una capa LSTM y una capa densa. A continuación, se explica el algortimo implementado y se muestran los resultados del modelo inicial.

1. Se importa el modelo secuancial y las capas LSTM y densa a partir de la librería Keras.

2. Se utiliza StandardScaler de sklearn para estandarizar las características.

3. Se divide el conjunto de datos en conjuntos de entrenamiento, prueba y validación utilizando la función 'train_test_split' de scikit-learn.

4. Se define el modelo de la red neuronal secuencial utilizando Keras. El modelo consta de una capa LSTM con 1024 unidades y una capa densa.

5. Se establece el número total de épocas para entrenar el modelo y se inicializa una variable 'min_loss' para realizar un seguimiento del valor de pérdida mínimo.

6. Se itera sobre el número total de épocas y se entrena el modelo en cada iteración utilizando el conjunto de entrenamiento.

7. Se evalúa el modelo en el conjunto de prueba y se calcula la pérdida.

8. Si la pérdida es menor que el mínimo anterior, se actualiza el mínimo y se imprimen y calculan varias métricas de evaluación, como el coeficiente de determinación (R2) y el error cuadrático medio (RMSE). También se guarda el modelo con el mejor rendimiento además de mostrar los resultados en el *set* de validación.

```
RNN subroutes:
100.00% of data is within a range of 600 seconds
99.98% of data is within a range of 300 seconds
95.72% of data is within a range of 60 seconds
76.27% of data is within a range of 30 seconds
R-squared: 0.9999
RMSE: 30.3562

RNN routes:
```

```
92.63% of data is within a range of 600 seconds
79.88% of data is within a range of 300 seconds
29.02% of data is within a range of 60 seconds
15.24% of data is within a range of 30 seconds
R-squared: 0.9363
RMSE: 865.7873
```

En general, el modelo de red neuronal recurrente (RNN) entrenado en este código muestra un rendimiento bastante bueno, con un alto porcentaje de datos dentro de los rangos de tiempo establecidos y métricas de evaluación favorables. Sin embargo, es importante tener en cuenta que estos resultados son específicos para los datos y el problema abordado en este código en particular, y podrían variar para diferentes conjuntos de datos y problemas.

3.7 Modelo ANN

En esta sección se detallarán los resultados del algoritmo ANN implementado en *(Dahl et al. 2014 [6])*, el cual obtuvo los mejores resultados segun *(Pałys et al., 2022 [2])* en lo que constituye predicción de tiempos de llegada de autobuses. Al igual que en el caso del RNN resulta apropiado establecer los valores iniciales del tamaño de *batch* en un nivel significativo de 256, tal como ha sido comprobado mediante la aplicación de la técnica de búsqueda de hiperparámetros bayesianos sobre el subconjunto de datos. Se utilizan 3 capas de 49 neuronas, ya como lo menciona *(Dahl et al. 2014 [6])*, y la función de activación relu para administrar los valores de entrada y una capa densa para los valores de salida. Al igual que el caso de la RNN se itera sobre cáda época y si el valor de perdida es menor a cualquier resultado previo que haya sido obtenido se guarda el modelo. Entonces los resultados de el mejor modelo ANN obtenido son los siguientes:

```
ANN routes:
95.80% of data is within a range of 600 seconds
82.99% of data is within a range of 300 seconds
26.30% of data is within a range of 60 seconds
13.08% of data is within a range of 30 seconds
R-squared: 0.9787
RMSE: 500.3673

ANN subroutes:
100.00% of data is within a range of 600 seconds
99.98% of data is within a range of 300 seconds
95.81% of data is within a range of 60 seconds
74.57% of data is within a range of 30 seconds
R-squared: 0.9999
RMSE: 31.0038
```

El modelo ANN en subrutas tiene un rendimiento similar al de LR y RNN, con porcentajes de datos dentro de los rangos de tiempo y métricas de rendimiento comparables. Los porcentajes de datos dentro de los rangos de tiempo son altos y similares a los de LR y RNN. El RMSE es ligeramente más alto que en LR y RNN, con un valor de 31.0038. El R2 es de 0.9999, lo cual indica una muy buena capacidad de ajuste. En rutas muestra un rendimiento similar al de RNN en rutas. Los porcentajes de datos dentro de los rangos de tiempo son más altos que en RNN en rutas, pero aún están por debajo de los modelos en subrutas. El RMSE es más bajo que en RNN en rutas, con un valor de 500.3673. El R2 es de 0.9787, lo cual indica una capacidad razonable de explicación de la variabilidad, pero inferior a la de los modelos en subrutas.

En resumen, los modelos en subrutas (LR, GBT, RNN, ANN) muestran un rendimiento más preciso en términos de porcentajes de datos dentro de los rangos de tiempo, así como valores de RMSE más bajos y R2 más altos en comparación con los modelos en rutas. Los modelos en subrutas parecen tener una mejor capacidad para predecir los tiempos de llegada en rutas específicas. El rendimiento menos preciso de los modelos en rutas en comparación con los modelos en subrutas puede deberse a la falta de consideración del patrón de paradas de los buses en las rutas. En las rutas de transporte público, diferentes buses pueden tener diferentes patrones de paradas a lo largo de su recorrido. Si un bus tiene más paradas en su ruta, naturalmente tomará más tiempo en llegar a su parada objetivo. Los modelos en subrutas, al centrarse en tramos más cortos del recorrido, pueden capturar mejor estos patrones de paradas específicos. Esto les permite realizar predicciones más precisas en términos de tiempos de llegada, ya que consideran las características individuales de cada subruta y su correspondiente patrón de paradas. En cambio, los modelos en rutas generalizan el tiempo de llegada para todo el recorrido, sin tener en cuenta las variaciones en los patrones de paradas de los buses. Como resultado, su rendimiento es inferior, con porcentajes de datos dentro de los rangos de tiempo más bajos y mayores valores de RMSE. Para mejorar el rendimiento de los modelos en rutas, sería beneficioso incorporar información sobre los patrones de paradas de los buses en el análisis. Esto permitiría ajustar las predicciones de acuerdo con la cantidad de paradas que un bus debe realizar antes de llegar a su destino final. Al considerar estas variaciones, los modelos podrían proporcionar estimaciones más precisas de los tiempos de llegada en rutas completas.

CAPÍTULO IV

RESULTADOS

En esta sección se presentan los resultados de los modelos realizados. Comenzando por las subrutas, se implementaron los siguientes 4 modelos de aprendizaje automático (ML): Regresión Lineal (LR), GBT (Gradient Boosting Trees), RNN (Redes Neuronales Recurrentes) y ANN (Redes Neuronales Artificiales).

4.1 Subrutas

Los modelos fueron entrenados con el 80 % del *dataset* y se usó el siguiente 20 % para evaluar los resultados.

Metrica	**LR**	**GBT**	**RNN**	**ANN**
RMSE	30.8276	175.7534	30.3562	31.0038
R^2	0.9999	0.9974	0.9999	0.9999

TABLA 4.1: Métricas de desempeño de los modelos en el cálculo de tiempos en Subrutas

A partir de la tabla 4.1 y la figura 4.1, se pueden extraer las siguientes conclusiones numéricas para la predicción de tiempos en subrutas:

Todos los modelos muestran un alto grado de precisión, con R^2 cercanos a 1 y RMSE relativamente bajos. Sin embargo, la Regresión Lineal, las Redes Neuronales Recurrentes y las Redes Neuronales Artificiales parecen tener un rendimiento ligeramente mejor que los Gradient Boosting Trees, especialmente en rangos de tiempo más cortos (60 y 30 segundos). Por ende un modelo LR, RNN o ANN serían útiles en la realidad. Un usario al saber que el tiempo predicho está dentro de un rango de error

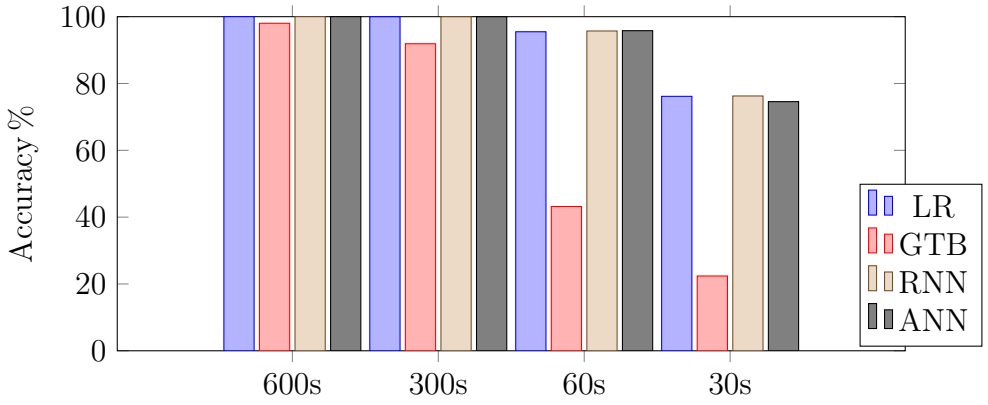

FIGURA 4.1: Porcentaje de precisión en distintos rangos de tiempo en la predicción en Subrutas

de aproximadamente 30s va a saber con certeza la hora que un autobús llegará a la hora esperada a una parada. Observamos en la figura 4.10 que a partir de los 1100 metros el error en el RNN comienza a incrementar. Como se observa en la figura 4.2, un viaje de 1.1km dura aproximadamente 200 segundos, entonces se puede concluir que el modelo es confiable y útil para proveer información sobre la hora de llegada del autobús.

Al observar la figura 4.5, figura 4.14 y figura 4.11 obtenemos resultados parecidos, a medida que la distancia se incrementa, el error de la predicción también lo hace. Pero en el caso de la figura 4.8 se observan más puntos donde el error es grande, esto indica que existen más instancias donde el modelo predice de forma errónea los tiempos de llegada.

En general, a medida que aumenta la distancia entre las paradas, también aumenta el error entre el valor predicho y el valor real. En términos de desempeño de los modelos, se puede concluir que el modelo RNN tiene el mejor rendimiento, seguido por LR, ANN y, por último, el modelo GBT, lo cual cumpliría con el objetivo principal del trabajo. Esto indica que el modelo RNN es el más adecuado para capturar los patrones y relaciones en los datos de las subrutas, ya que muestra el menor

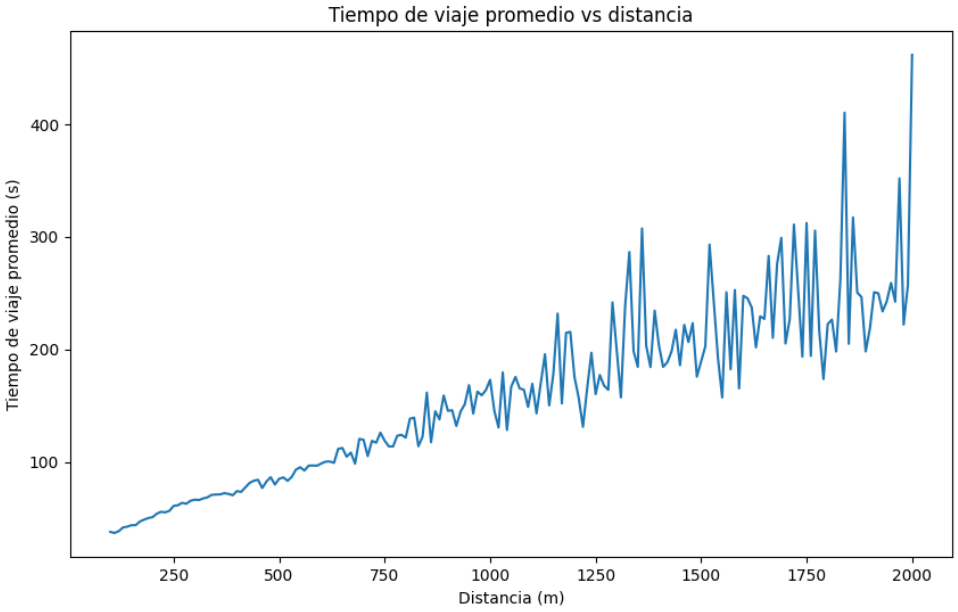

FIGURA 4.2: Tiempo de viaje vs distancia en subrutas

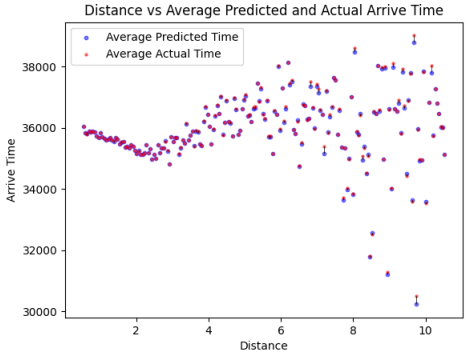

FIGURA 4.3: Distance vs Average Predicted and Actual Arrive Time (LR)

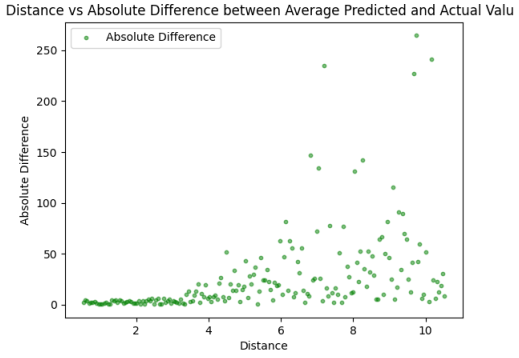

FIGURA 4.4: Distance vs Difference between Average Predicted and Actual Value (LR)

FIGURA 4.5: Lineal Regression subroutes

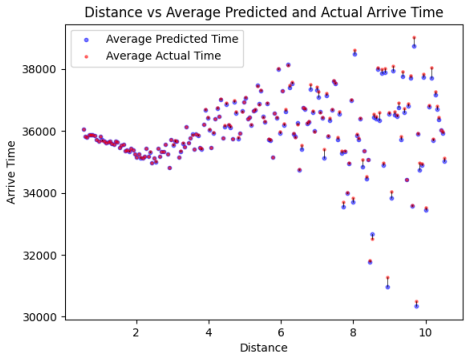

FIGURA 4.6: Distance vs Average Predicted and Actual Arrive Time (GBT)

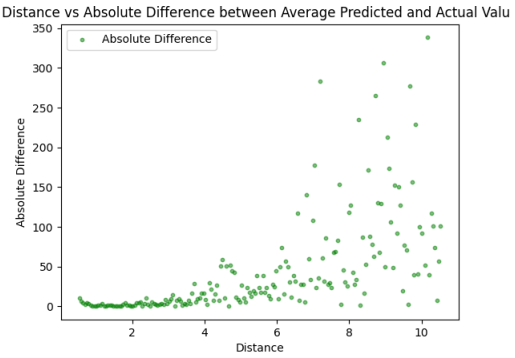

FIGURA 4.7: Distance vs Difference between Average Predicted and Actual Value (GBT

FIGURA 4.8: GBT Regression subroutes

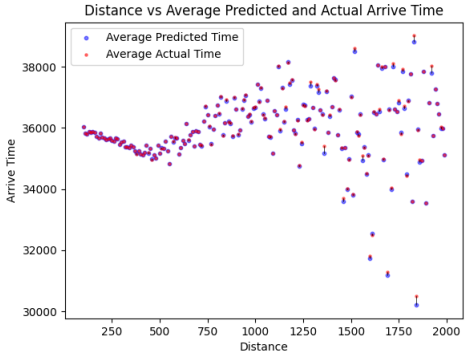

FIGURA 4.9: Distance vs Average Predicted and Actual Arrive Time (RNN)

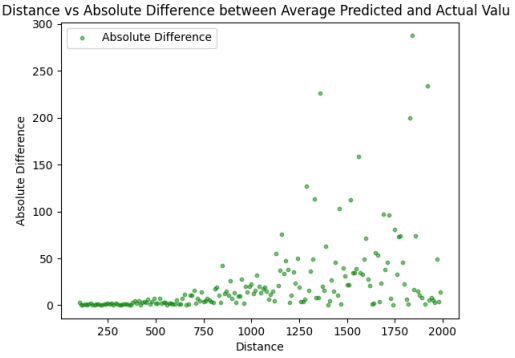

FIGURA 4.10: Distance vs Difference between Average Predicted and Actual Value (RNN)

FIGURA 4.11: RNN subroutes

error en los valores predichos en comparación con los valores reales, incluso para distancias más largas entre paradas. El modelo LR también tiene un rendimiento aceptable, aunque con un error ligeramente mayor en comparación con el RNN. Los modelos ANN y GBT presentan un mayor error, lo que indica que tienen dificultades para capturar los patrones y relaciones específicos de las subrutas, especialmente en distancias más largas. En resumen, el modelo RNN es el más recomendado para predecir los tiempos de llegada en subrutas, ya que presenta el menor error en

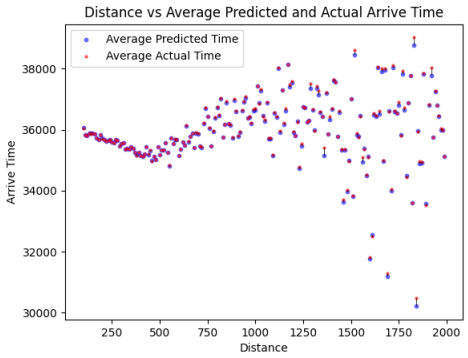

FIGURA 4.12: Distance vs Average Predicted and Actual Arrive Time (ANN)

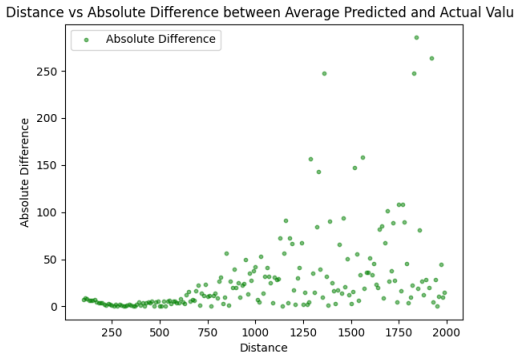

FIGURA 4.13: Distance vs Difference between Average Predicted and Actual Value (ANN)

FIGURA 4.14: ANN subroutes

comparación con los modelos LR, ANN y GBT. Sin embargo, es importante tener en cuenta que todos los modelos muestran un aumento del error a medida que aumenta la distancia entre paradas, lo cual puede indicar un mayor desafío en la predicción precisa de tiempos de llegada para distancias más largas.

4.2 Rutas

Ahora pasamos a los modelos de predicción de tiempos en rutas.

Metrica	**LR**	**GBT**	**RNN**	**ANN**
RMSE	2174.1705	870.9205	865.7873	500.3673
R^2	0.5982	0.9355	0.9363	0.9787

TABLA 4.2: Métricas de desempeño de los modelos en el cálculo de tiempos en rutas

A partir de la tabla 4.2 y la figura 4.15, se observan las siguientes métricas:

En términos de precisión y capacidad para predecir rutas completas, los modelos de Regresión Lineal (LR) tienen un rendimiento bastante bajo, con porcentajes muy bajos de datos dentro de los rangos de tiempo definidos y un RMSE alto. Los modelos

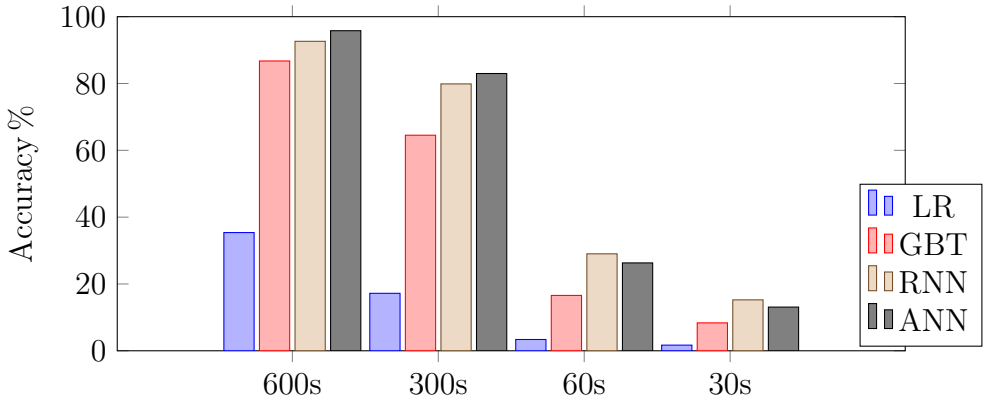

FIGURA 4.15: Porcentaje de precisión en distintos rangos de tiempo en la predicción en rutas

de Gradient Boosting Trees (GBT), Redes Neuronales Recurrentes (RNN) y Redes Neuronales Artificiales (ANN) tienen un rendimiento mucho mejor, con porcentajes significativos de datos dentro de los rangos de tiempo y valores de RMSE más bajos. Entre estos modelos, el modelo de Redes Neuronales Artificiales (ANN) muestra el mejor rendimiento, con los porcentajes más altos de datos dentro de los rangos de tiempo, un R2 más alto y el menor RMSE. Pero aún así el RMSE es más preciso obteniendo información correctamente en intervalos de tiempo menores, por ejemplo en el caso de 60s y 30s. Entonces se cumple que el modelo RNN es superior cuando se cumplen ciertas circunstancias.

A partir de la figura 4.19, figura 4.22, figura 4.25 y figura 4.28, se puede observar la relación entre la distancia y los tiempos de llegada promedio tanto en las predicciones como en los valores reales para cada modelo. En general, a medida que aumenta la distancia, los tiempos de llegada tanto predichos como reales tienden a incrementarse. En esta comparación, los modelos RNN y ANN muestran una mejor correspondencia entre los tiempos de llegada predichos y los valores reales a medida que aumenta la distancia. Esto indica que estos modelos son más consistentes en la predicción de los tiempos de llegada en distancias más largas. Por otro lado, los modelos LR y GBT muestran una mayor discrepancia entre las predicciones y los

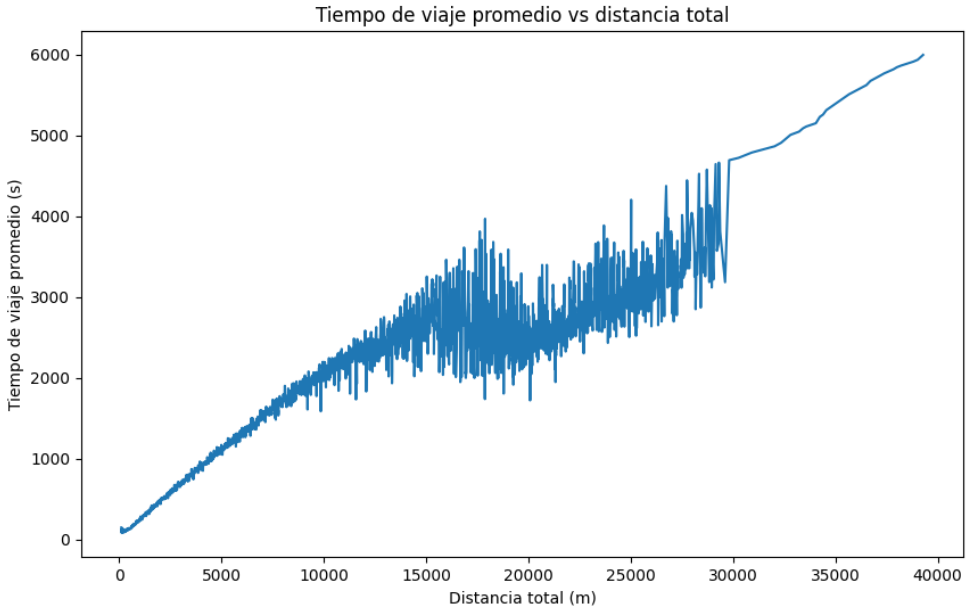

FIGURA 4.16: Tiempo de viaje vs distancia en rutas

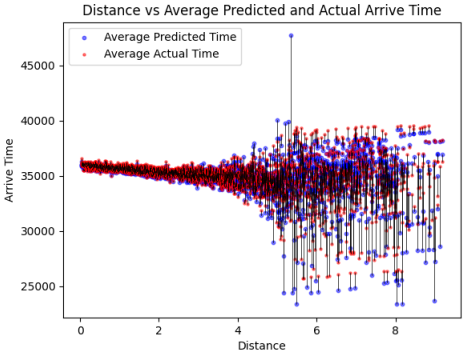

FIGURA 4.17: Distance vs Average Predicted and Actual Arrive Time (LR)

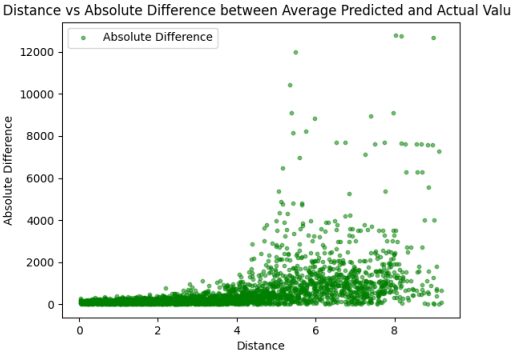

FIGURA 4.18: Distance vs Difference between Average Predicted and Actual Value (LR)

FIGURA 4.19: Lineal Regression routes

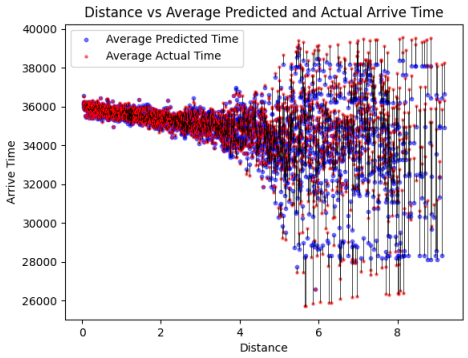

FIGURA 4.20: Distance vs Average Predicted and Actual Arrive Time (GBT)

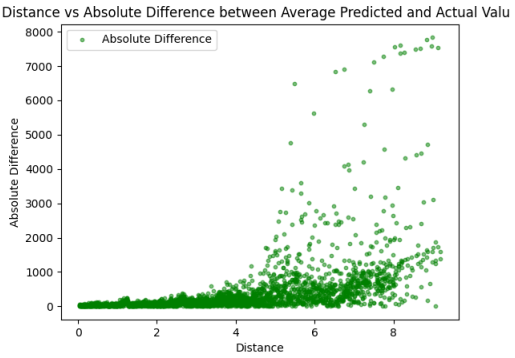

FIGURA 4.21: Distance vs Difference between Average Predicted and Actual Value

FIGURA 4.22: GBT Regression routes

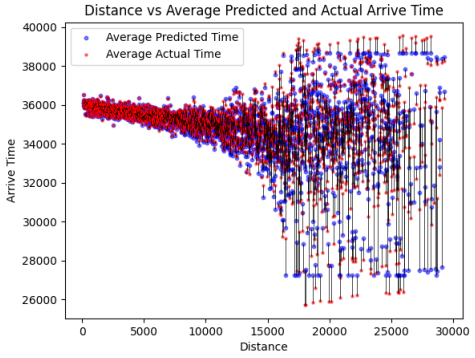

FIGURA 4.23: Distance vs Average Predicted and Actual Arrive Time (RNN)

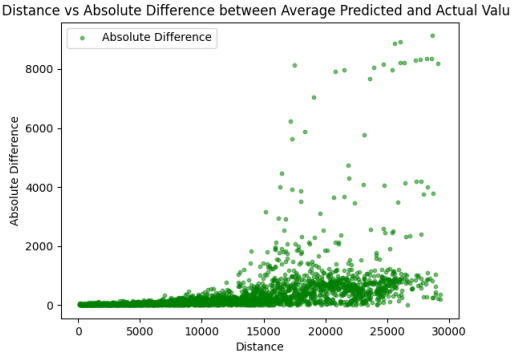

FIGURA 4.24: Distance vs Difference between Average Predicted and Actual Value (RNN)

FIGURA 4.25: RNN routes

valores reales, especialmente en distancias más largas. En las gráficas de la diferencia entre el tiempo real y predicho se puede observar que los modelos RNN y ANN tienen una menor diferencia entre las predicciones y los valores reales en comparación con los modelos LR y GBT. Esto indica que los modelos RNN y ANN son más precisos en la predicción de los tiempos de llegada en comparación con los modelos LR y GBT, especialmente en distancias más largas. En resumen, las gráficas respaldan las conclusiones previas de que los modelos RNN y ANN presentan un

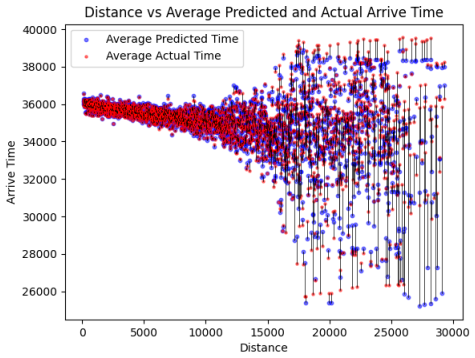

FIGURA 4.26: Distance vs Average Predicted and Actual Arrive Time (ANN)

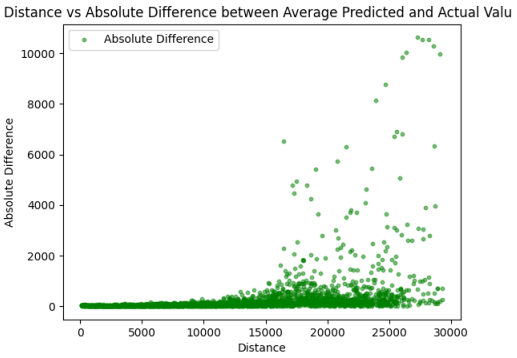

FIGURA 4.27: Distance vs Difference between Average Predicted and Actual Value (ANN)

FIGURA 4.28: ANN routes

mejor desempeño en la predicción de los tiempos de llegada en rutas. Estos modelos muestran una mejor correspondencia y una menor diferencia entre las predicciones y los valores reales, especialmente en distancias más largas. Por otro lado, los modelos LR y GBT presentan una mayor discrepancia y una mayor diferencia entre las predicciones y los valores reales, lo que indica un rendimiento inferior en la predicción de los tiempos de llegada en rutas.

Una conclusión adicional es que los modelos en subrutas son más adecuados cuando se necesita una precisión más fina en la predicción de tiempos de llegada para segmentos específicos del recorrido. Estos modelos pueden ser útiles, por ejemplo, para proporcionar estimaciones de tiempo de llegada en paradas intermedias o para planificar la duración de trayectos cortos. Por otro lado, los modelos en rutas son más apropiados para tener una visión general del tiempo de viaje completo, sin profundizar en los detalles de cada parada intermedia. Aunque su rendimiento es inferior en términos de precisión, aún pueden ser valiosos para proporcionar estimaciones aproximadas del tiempo total de viaje y ayudar en la planificación de rutas más largas.

4.3 Propuesta de solución para Lima

Después de la comparación se detalla los sensores e información que se necesita para emplear esta predicción en los buses Corredor Rojo en Lima. Se necesitan cubrir los siguientes campos: vehicle_id, month, day_of_month, day_of_week, first_stop, exit_stop, target_stop, total_distance, distance, first_time, exit_time, arrive_time. Para esto se debe trabajar con un censor de cercanía para capturar cuando a llegado a una parada. Además debemos contar con documento que determine las paradas y la distancia entre las paradas. Se puede aplicar el sensor de ultrasonido Hc-sr04 con un arduino, donde si la distancia es menor a cierto umbral, entonces se considera que el vehículo llegó a dicha parada. Asimismo, este sensor debería estar conectado a la red por medio de un protocolo MQTT a un broker de Mosquitto para enviar sus datos cada cierto tiempo. Con este nuevo sistema ya no sería necesario el preprocesamiento de datos con Dask, solo se especificarían los intervalos de tiempos con pySpark, ya que la data va a estar ordenada para realizar el entrenamiento de los modelos.

Si se desea predecir los tiempos de llegada de un bus a una parada en un día futuro, como parte de la planificación del transporte público, utilizar un modelo basado en Redes Neuronales Recurrentes (RNN) o Redes Neuronales Artificiales (ANN) puede ser más adecuado. Se puede destacar la utilidad de la tecnología de Waze o Google Maps para predecir el tiempo de llegada de un autobús durante su recorrido. Estos sistemas aprovechan información en tiempo real, como la velocidad de los vehículos y las condiciones del tráfico obtenidas a través de la visualización satelital, para estimar los tiempos de llegada en función de las condiciones actuales de la vía. A diferencia de estas tecnologías los modelos ML desarrollados son capaces de capturar patrones y relaciones complejas en los datos históricos, lo cual incluiría las características de los patrones de paradas a lo largo del recorrido. Al entrenar el modelo con datos pasados que incluyan información de diferentes días de la semana, horarios y condiciones de tráfico, se puede obtener un modelo capaz de hacer predicciones futuras basadas en

esos patrones aprendidos. En contraste, los sistemas como Waze o Google Maps, que utilizan información en tiempo real para predecir el tráfico del día actual, no son tan útiles para predecir los tiempos de llegada de un bus en un día futuro. Estos sistemas se basan en datos actuales, como la velocidad de los vehículos y las condiciones del tráfico en tiempo real (a partir de visualización satelital), y no tienen en cuenta patrones históricos específicos de los recorridos de los buses. Por lo tanto, si el objetivo es obtener estimaciones de tiempo de llegada precisas para un día futuro, los modelos basados en RNN o ANN, entrenados con datos históricos, son más apropiados. Estos modelos considerarán las particularidades de las subrutas y los patrones de paradas para hacer predicciones más precisas en el contexto de días y horarios específicos. En resumen, si se desea predecir los tiempos de llegada de un bus a una parada en un día futuro, se recomienda utilizar modelos como RNN o ANN entrenados con datos históricos. Esto permitirá capturar los patrones de paradas y las relaciones complejas, brindando estimaciones más precisas en comparación con los sistemas basados en información de tráfico en tiempo real como Waze o Google Maps.

CONCLUSIONES

4.4 Mejorar el nivel de predicción de tiempos de llegada a estaciones de buses

La aplicación de RNN con LSTM ha demostrado ser efectiva para mejorar el nivel de predicción de tiempos de llegada a estaciones de buses, incluso superando una ANN, que anteriormente eran reconocidas como una de las mejores en el área. El documento explica que los modelos basados en RNN o ANN, cuando se entrenan con datos históricos, son más apropiados para considerar las particularidades de las subrutas y los patrones de paradas, haciendo predicciones más precisas en el contexto de días y horarios específicos. Sin embargo, el modelo RNN, en particular, demostró tener el mejor rendimiento, seguido por LR, ANN y, por último, el modelo GBT. Esto indica que el modelo RNN es el más adecuado para capturar los patrones y relaciones en los datos de las subrutas, ya que muestra el menor error en los valores predichos en comparación con los valores reales, incluso para distancias más largas (rutas). Además, el modelo RNN presentó el menor error en comparación con los modelos ANN y GBT, lo que indica que tiene menos dificultades para capturar los patrones y relaciones específicos de las subrutas, especialmente en distancias más largas. En resumen, la aplicación de RNN con LSTM ha demostrado ser una estrategia efectiva para mejorar la precisión de las predicciones de los tiempos de llegada de los autobuses, superando a las técnicas previamente reconocidas como las mejores en el campo, como las ANN en el caso de subrutas, y, en el caso de rutas el RNN obtuvo mejores resultados al obtener más información precisa dentro de un rango de 30s y de 60s.

4.5 Obtención del dataset para el entrenamiento ML

La obtención del dataset se realizó de manera óptima utilizando las herramientas Dask y Spark, que permitieron realizar el entrenamiento de máquina y obtener datos relevantes de manera eficiente. El primer paso fue descargar los archivos de texto del MTA (Metropolitan Transportation Authority) *(MTA, Bus Historical Data [15])*, que contenían toda la información que se utilizaría para entrenar y hacer pruebas con el modelo. Estos archivos, un total de 89 que abarcaban desde el 1 de agosto de 2014 hasta el 31 de octubre de 2014, se almacenaron en formato .txt y ocupaban un total de 64419 MB de memoria. Se utilizó Dask para realizar un filtro inicial sobre el dataset de autobuses para poder ingresarla a los modelos de Machine Learning. Dask es una biblioteca de Python diseñada para trabajar con grandes conjuntos de datos de manera eficiente y escalable. Permite procesar datos que no caben en la memoria RAM de un solo ordenador al dividirlos en bloques más pequeños y distribuir el procesamiento en múltiples núcleos o incluso en múltiples máquinas. El preprocesamiento con Dask se realizó por archivo. Por cada grupo de 'vehicle_id', se realizó una función, sort & calc, que obtenía la hora en la que un vehículo pasa por una parada. Después, se estandarizaron las distancias entre paradas, se agrupó por la columna 'trip' y se realizó el algoritmo dist fix. Finalmente, luego de tener todos los dataframes por vehicle id, estos se guardaron en un parquet con el nombre de la fecha procesada . Una vez realizado el preprocesamiento con Dask, se realizó el preprocesamiento con PySpark. PySpark es un framework de computación distribuida diseñado para procesar grandes cantidades de datos distribuidas en los clusters de una computadora, logrando un proceso rápido y eficiente de procesamiento de datos. Se comprimieron todos los archivos en un solo parquet y se ejecutó el código correspondiente. En resumen, la combinación de Dask y Spark permitió obtener y preprocesar el dataset de manera eficiente, facilitando el entrenamiento de los modelos de Machine Learning y la obtención de datos relevantes para la predicción de

tiempos de llegada a estaciones de buses.

4.6 Búsqueda de hiperparámetros de ML

La búsqueda de hiperparámetros se realizó de manera exitosa utilizando el método de optimización bayesiana, que es particularmente útil en el contexto del aprendizaje automático. Los hiperparámetros son parámetros que no se aprenden directamente dentro de los estimadores. En scikit-learn, se pasan como argumentos a la constructora de las clases de estimadores. Ejemplos típicos incluyen C, kernel y gamma para Support Vector Classifier, alpha para Lasso, etc. La optimización bayesiana aborda este problema construyendo un modelo probabilístico de la función de pérdida y utilizando este modelo para seleccionar los valores de los hiperparámetros que probablemente resulten en una pérdida menor. En lugar de buscar en todo el espacio de hiperparámetros, la optimización bayesiana se centra en las regiones del espacio que son más prometedoras. En el caso del modelo RNN, antes de realizar la experimentación se realizó el tunning de hiperparámetros a partir de optimización bayesiana sobre las primeras 1000 filas del dataset. Los resultados mostraron que en un rango de [10-100] épocas, [16-256] tamaño de batch y [32-1024] se obtenían mejores resultados en 100 épocas, un batch de 256 y 1024 unidades de LSTM. A partir de esto se decidió realizar el modelo inicial del RNN con los hiperparámetros mencionados. En resumen, la optimización bayesiana es una herramienta poderosa para la optimización de hiperparámetros en el aprendizaje automático, permitiendo seleccionar de manera eficiente los hiperparámetros óptimos para un algoritmo de aprendizaje.

4.7 Aplicación de sistemas ML en Lima

La implementación de un modelo ML para la predicción de tiempos de llegada de buses en Lima implicaría varios pasos. Primero, se necesitaría recoger un conjunto de datos relevante, que podría incluir datos de geolocalización de autobuses en tiempo real, horarios de autobuses, información sobre las rutas de los autobuses, datos de tráfico, condiciones climáticas, entre otros factores que puedan influir en los tiempos de llegada de los autobuses. Estos datos recogidos necesitarían ser preprocesados para ser utilizados en un modelo de ML. Una vez preprocesados los datos, se seleccionarían y entrenarían los modelos de ML adecuados para el problema. En el documento, se exploraron varios modelos de predicción, incluyendo GBTRegression, Linear Regression, RNN (LSMT), y ANN. Además, se realizó una búsqueda de los mejores hiperparámetros para cada modelo ML utilizando el método de optimización bayesiana. Después de entrenar los modelos, se realizarían pruebas para obtener resultados de rendimiento y entender la eficacia de los diferentes modelos de ML en la predicción de los tiempos de llegada de los autobuses. A partir de los modelos ML evaluados, se podría generar una propuesta que detalla las herramientas e información necesarias para implementar este método de predicción en la realidad actual de Lima. Finalmente, una vez implementado, el modelo debería ser monitoreado para asegurarse de que sigue proporcionando predicciones precisas a medida que cambian las condiciones. El modelo podría necesitar ser reentrenado o ajustado con el tiempo para mantener su precisión. Los algoritmos desarrollados para este sistema están disponibles en un repositorio de GitHub, lo que facilita su implementación y adaptación a la realidad de Lima.

Se puede resumir que el proyecto logró con éxito su objetivo principal de mejorar el nivel de predicción de tiempos de llegada a estaciones de buses en Lima. Esto se logró mediante la implementación de un modelo de RNN con LSTM, que demostró tener un rendimiento superior en algunos casos en comparación con otros modelos,

como las ANN, que anteriormente eran reconocidas como las mejores en el área. En cuanto a la obtención del dataset para el entrenamiento de ML, se utilizó una combinación eficiente de las herramientas Dask y Spark para recoger y preprocesar los datos. Estas herramientas permitieron manejar grandes cantidades de datos y prepararlos para su uso en los modelos de ML, facilitando así el entrenamiento de la máquina y la obtención de datos relevantes para la predicción de tiempos de llegada a estaciones de buses. La búsqueda de hiperparámetros de ML se realizó con éxito mediante el método de optimización bayesiana. Este enfoque permitió seleccionar de manera eficiente los hiperparámetros óptimos para cada modelo, mejorando así la precisión de las predicciones. Finalmente, se propuso un modelo de predicción que se puede aplicar a la realidad de Lima. Este modelo toma en cuenta las particularidades de las rutas de los autobuses y los patrones de paradas en Lima, lo que permite hacer predicciones más precisas. Una vez implementado, el modelo debería ser monitoreado y ajustado con el tiempo para mantener su precisión. Los algoritmos desarrollados para este sistema están disponibles en un repositorio de GitHub, lo que facilita su implementación y adaptación a la realidad de Lima. En resumen, el proyecto logró con éxito sus objetivos de mejorar la predicción de tiempos de llegada a estaciones de buses en Lima, obtener de manera óptima el dataset para el entrenamiento de ML, realizar una búsqueda eficiente de hiperparámetros de ML, y proponer un modelo de predicción aplicable a la realidad de Lima.

Este proyecto ha demostrado la capacidad de la ciencia de datos y el aprendizaje automático para transformar y mejorar la eficiencia de los sistemas de transporte público. A través de la aplicación de técnicas avanzadas de procesamiento de datos y la implementación de modelos de aprendizaje automático, hemos logrado desarrollar un sistema de predicción de tiempos de llegada a estaciones de buses que es altamente preciso y aplicable a la realidad de Lima. Este logro no solo representa un avance técnico, sino que también tiene el potencial de tener un impacto significativo en la vida cotidiana de las personas en Lima. Al proporcionar predicciones precisas de los

tiempos de llegada de los autobuses, podemos ayudar a los pasajeros a planificar mejor sus viajes, reducir el tiempo de espera y mejorar la eficiencia general del sistema de transporte público. Además, este proyecto ha demostrado el valor de la colaboración y la innovación. A través de la combinación de diferentes herramientas y técnicas, hemos logrado superar desafíos y alcanzar nuestros objetivos. Este espíritu de innovación y colaboración es fundamental para el avance de la ciencia de datos y el aprendizaje automático. Por último, aunque este proyecto ha logrado mucho, también nos ha mostrado las posibilidades para el futuro. Con la continua evolución de la tecnología y el aprendizaje automático, hay un potencial ilimitado para mejorar aún más la eficiencia y la precisión de los sistemas de predicción de transporte. Estamos emocionados por las oportunidades que el futuro nos depara y esperamos continuar contribuyendo a este campo en constante evolución.

Gracias por acompañarme en este viaje de descubrimiento e innovación. Espero que este proyecto sirva como un ejemplo de lo que es posible cuando combinamos la ciencia de datos, el aprendizaje automático y una pasión por mejorar el mundo que nos rodea.

TRABAJOS FUTUROS

Se pueden identificar varias direcciones prometedoras para futuros trabajos de investigación y desarrollo. A continuación, se presentan algunas de estas posibilidades, cada una de las cuales representa una oportunidad para expandir y mejorar el trabajo realizado hasta ahora.

4.8 Mejora y expansión de los modelos de aprendizaje automático

Aunque el modelo RNN con LSTM demostró ser el más eficaz para predecir los tiempos de llegada de los autobuses, siempre hay espacio para la mejora y la innovación. Los futuros trabajos podrían explorar la aplicación de otros tipos de modelos de aprendizaje automático, como las Redes Neuronales Convolucionales (CNN) o los Transformers, que han demostrado ser eficaces en una variedad de tareas de aprendizaje automático. Además, se podrían realizar más investigaciones para ajustar aún más los hiperparámetros de los modelos existentes, con el objetivo de mejorar la precisión de las predicciones.

4.9 Incorporación de datos adicionales

El modelo actual se basa en una serie de datos, incluyendo la geolocalización de los autobuses en tiempo real, los horarios de los autobuses, y la información sobre las rutas de los autobuses. Sin embargo, hay muchos otros tipos de datos que podrían ser útiles para predecir los tiempos de llegada de los autobuses. Por ejemplo, los futuros trabajos podrían explorar la incorporación de datos en tiempo real sobre las condiciones del tráfico, el clima, o incluso eventos especiales que podrían afectar al tráfico, como los partidos de fútbol o los conciertos. La incorporación de estos tipos de datos podría mejorar aún más la precisión de las predicciones. Además de que

en este caso no se consideró un valor existente crucial como el índice de *trip*, el cual identificaba el patrón de paradas de cada bus.

4.10 Expansión a otras ciudades y regiones

Aunque este proyecto se centró en la realidad de Lima y se realizó en base a datos de Nueva York, los métodos y técnicas utilizados podrían ser aplicables a otras ciudades o regiones. Los futuros trabajos podrían explorar la implementación de este sistema de predicción en otras ciudades, adaptándolo a las particularidades de cada una. Esto podría implicar la recopilación y el análisis de nuevos conjuntos de datos, así como la adaptación de los modelos de aprendizaje automático para tener en cuenta las diferencias en las rutas de los autobuses, los patrones de tráfico, y otros factores locales.

4.11 Desarrollo de una interfaz de usuario e integración con sistemas existentes

Actualmente, los algoritmos desarrollados para este sistema están disponibles en un repositorio de GitHub. Sin embargo, para que este sistema de predicción sea verdaderamente útil para los pasajeros, sería beneficioso desarrollar una interfaz de usuario amigable y fácil de usar. Esta interfaz podría presentar las predicciones de los tiempos de llegada de los autobuses de una manera clara y comprensible, y podría ser accesible a través de una aplicación móvil o un sitio web. Además, este sistema de predicción podría ser integrado con sistemas de transporte público existentes o aplicaciones de navegación para proporcionar predicciones de tiempos de llegada en tiempo real a los pasajeros.

Estas son solo algunas de las posibles direcciones para futuros trabajos basados en la tesis realizada. Cada una de estas direcciones representa una oportunidad para expandir y mejorar el trabajo laborado hasta el momento, y para continuar avanzando en el objetivo de mejorar la eficiencia y la eficacia de los sistemas de transporte público. A medida que la tecnología y las técnicas de aprendizaje automático continúan avanzando, es probable que surjan aún más oportunidades para la innovación y la mejora en este campo. En última instancia, el objetivo de estos futuros trabajos será continuar mejorando la precisión de las predicciones de los tiempos de llegada de los autobuses, con el fin de mejorar la experiencia de los pasajeros y la eficiencia del sistema de transporte público.

REFERENCIAS BIBLIOGRÁFICAS

[1] "Lima traffic report," https://www.tomtom.com/traffic-index/lima-traffic/, [Online; accessed 2022-09-18].

[2] Ł. Pałys, M. Ganzha, and M. Paprzycki, "Machine learning for bus travel prediction," in *Computational Science – ICCS 2022*, D. Groen, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Eds. Cham: Springer International Publishing, 2022, pp. 703–710.

[3] G. Bravo Rocca, "Pymach y sparkmach: Sistemas semiautomáticos de procesamiento de datos con dimensión variable usando aprendizaje automático y técnicas escalables," https://renati.sunedu.gob.pe/handle/sunedu/3265416, 2018.

[4] T. Yin, G. Zhong, J. Zhang, S. He, and B. Ran, "A prediction model of bus arrival time at stops with multi-routes," *Transportation Research Procedia*, vol. 25, pp. 4623–4636, 2017, [Online; accessed 2022-11-14].

[5] A. Bachu, R. Behera, V. Kumar, K. Jetty, L. Vanajakshi, and S. Subramanian, "Bus travel time prediction using machine learning approaches," 12 2016.

[6] E. Dahl, A. A. Sjåfjell, and S. Skogen, "On implementations of bus travel time prediction utilizing methods in artificial intelligence," Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2014.

[7] J. P. León Almenara, "La fórmula para viajar en la mitad de tiempo en los corredores complementarios," https://elcomercio.pe/lima/transporte/la-formula-para-viajar-en-la-mitad-de-tiempo-en-los-corredores-complementarios-notepases-noticia/, nov 28 2019, [Online; accessed 2022-09-18].

[8] "Moovit, corredor rojo: Lima," https://moovitapp.com/index/es-419/transporte_pblico-lines-Lima-1102-858864, [Online; accessed 2023-04-20].

[9] G. Bravo Rocca and P. Torres Robatty, "Sparkmach: a Distributed Data Processing System Based on Automated Machine Learning For Big Data."

[10] M. Rocklin, "Dask: Parallel computing with task scheduling," https://dask.org/, 2015, accessed: April 4, 2023.

[11] "Apache spark." https://spark.apache.org/, accessed: April 21, 2023.

[12] M. Kula, "Keras: Simple. flexible. powerful." https://keras.io/, 2015, accessed: July 3, 2023.

[13] F. Pedregosa, G. Varoquaux, and A. e. a. Gramfort, "Scikit-learn: Machine learning in python," https://scikit-learn.org/stable/, 2011, accessed: April 4, 2023.

[14] "Khipu," https://docs.khipu.utec.edu.pe/, accessed: May 5, 2023.

[15] "Mta Bus Time® Historical Data," http://web.mta.info/developers/MTA-Bus-Time-historical-data.html, [Online; accessed 2022-11-13].