ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Desarrollo de una aplicación basada en aprendizaje de máquina para la transferencia automática de reconectadores en falla de la red eléctrica de Guayaquil

PROYECTO DE TITULACIÓN

Previo la obtención del Título de:

Magister en Automatización y Control

Presentado por:
César Daniel Rodríguez Flores

GUAYAQUIL - ECUADOR
Año: 2024

AGRADECIMIENTOS

Mi más sincero agradecimiento a mis padres, tutores y demás personas que ayudaron a la recopilación de material para la elaboración de este trabajo.

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; Yo, *César Daniel Rodríguez Flores* doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Firmado electrónicamente por:
CESAR DANIEL
RODRIGUEZ FLORES

César Daniel
Rodríguez Flores

COMITÉ EVALUADOR

Firmado electrónicamente por:
RICARDO ALFREDO
CAJO DIAZ

Ph.D Ricardo Cajo Díaz

PROFESOR TUTOR

EFREN VINICIO HERRERA MUENTES
Digitally signed by EFREN VINICIO HERRERA MUENTES
Date: 2024.09.11 15:49:58 -05'00'

Ph.D Efrén Herrera

PROFESOR EVALUADOR

RESUMEN

Debido a las regulaciones del ARCERNNR, toda empresa distribuidora de electricidad necesita cumplir estándares mínimos respecto a la frecuencia y tiempo de restablecimiento de fallas. Esto, añadido a la considerable importancia de mantener un continuo y confiable servicio eléctrico para la ciudad de Guayaquil, indica que es deseable la automatización de la red eléctrica para el aislamiento de fallas y restablecimiento del servicio. Para esto se desarrolló una aplicación para la transferencia automática de reconectadores en fallo, de la red eléctrica de Guayaquil. Esta aplicación consta de automatismos localizados en las RTU (Remote Terminal Unit) de las subestaciones a automatizar desde donde se programa y controla los distintos reconectadores de campo ubicados a lo largo de la ciudad. El programa corriendo en estas RTU se comunica con un servidor de soporte que corre un servicio de predicción habilitado por medio de aprendizaje de máquina. Mediante esta predicción el programa en las RTU decide si puede realizar la transferencia o no. Los distintos automatismos se pueden supervisar y controlar desde pantallas en el sistema SCADA de la empresa distribuidora. Se encontró que el error del modelo predictivo para la transferencia analizada fue del 5.25%. Se concluye que el modelo creado mediante aprendizaje de máquina representa una herramienta útil con resultados suficientemente adecuados para agilizar la transferencia automática de carga en redes de distribución.

Palabras Clave: Transferencia automática, LightGBM, Smart Grid, Predicción de carga

ABSTRACT

Due to ARCERNNR regulations, every electricity distribution company needs to meet minimum standards regarding the frequency of events and time of fault restoration. This, added to the considerable importance of maintaining a continuous and reliable electrical service for the city of Guayaquil, indicates that the automation of the electrical network is desirable for fault isolation and service restoration. For this purpose, a system was developed for the automatic transfer of faulty reclosers from the Guayaquil electrical network. This system consists of programs located in the RTUs (Remote Terminal Unit) of the substations to be automated, from where the different field reclosers located throughout the city are programmed and controlled. The program running in these RTUs communicates with a support server that runs a prediction service enabled by means of machine learning. Through this prediction, the program in the RTUs decides whether or not it can carry out the transfer. The different automations can be monitored and controlled from screens in the SCADA system of the distribution company. The error of the predictive model for the analyzed transfer was found to be 5.25%. It is concluded that the model created through machine learning represents a useful tool with sufficiently adequate results to speed up the automatic transfer of load in distribution networks.

Keywords: automatic transfer, LightGBM, Smart Grid, Load forecasting

ÍNDICE GENERAL

ABREVIATURAS

RTU	Remote terminal unit
NC	Normalmente cerrado
NA	Normalmente abierto
DB	Base de datos
SCADA	Supervisory control and data adquisition
ARCERNNR	Agencia de regulación y control de energía y recursos naturales
KVA	Kilo Voltio Amperio

ÍNDICE DE FIGURAS

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Descripción del problema

De acuerdo a las regulaciones del ARCERNNR (Agencia de regulación y control de energía y recursos naturales), las empresas de distribución eléctrica deben cumplir estándares mínimos en cuanto al tiempo y frecuencia en las fallas del servicio eléctrico. Adicionalmente, perturbaciones en la red eléctrica son capaces de afectar de forma negativa la infraestructura crítica en la ciudad tal como hospitales, redes de telecomunicación, sistemas de agua, o incluso expandirse a otros subsistemas de la red eléctrica debido a su alta interconectividad [1].

Es debido a esto que una empresa distribuidora de electricidad necesita mantener un alto estándar de confiabilidad para minimizar el tiempo de falla. Típicamente esto se logra mediante redundancias en la topología de la red eléctrica que redistribuyen la carga a distintos alimentadores una vez ha ocurrido un problema [2]. Para poder realizar esto decenas de reconectadores se encuentran instalados a través de toda la ciudad para redistribuir sectores específicos a un alimentador distinto en caso de falla.

Sin embargo, realizar una reconexión no es un proceso inmediato, operadores en los respectivos centros de control deben observar el fallo, analizarlo y realizar un estudio de carga en el alimentador a transferir. Luego de esto se realiza la transferencia manualmente desde el sistema SCADA del centro de control. Este proceso puede durar un tiempo importante, y dado el alto coste horario de los apagones en la matriz productiva, mientras más demore el tiempo de reconexión, mayor será el daño económico a la ciudad.

1.2 Justificación del problema

Una aplicación que se conecte a la red de control del sistema eléctrico de Guayaquil para automatizar la transferencia de carga en situaciones de fallo tiene el potencial de aumentar la confiabilidad del sistema y disminuir el costoso tiempo de falla en la red eléctrica.

Este trabajo se enfoca en la automatización de los sectores de cerro azul y cerro del Carmen, ambos con cargas críticas para el funcionamiento de la ciudad puesto que poseen la mayor parte de las antenas de comunicación de la ciudad

Para automatizar la transferencia de carga se propone como solución un sistema consistente en varios programas corriendo en distintos dispositivos de la red de control. En la red eléctrica, cada reconectador se encuentra conectado a un RTU (concentrador de datos) establecido en cada subestación de la ciudad. Los cuales se encuentran a su vez comunicados con el servidor central corriendo el software SCADA que controla la red eléctrica. Se desarrollarían programas locales corriendo en la RTU de cada reconectador, estos programas ejecutarían las secuencias de conexión y desconexión en los reconectadores respectivos basándose en la información de los medidores y equipos comunicados a ese RTU.

Para poder ejecutar la secuencia para la transferencia de carga los programas en el RTU deben asegurarse de que el alimentador y el transformador con la carga adicional transferida se encuentren dentro de los límites de funcionamiento por un período de varias horas en el futuro. Para lograr esto envían una petición a una aplicación corriendo en un servidor de soporte que se encargará de realizar la predicción de la demanda transferida y verificar que se encuentre dentro de los límites.

Esta aplicación corriendo en un servidor de soporte estaría conectada a todos los RTU de todas las subestaciones de Guayaquil. Este servidor tendría comunicación para protocolos MODBUS TCP y DNP3 [3], necesaria para comunicarse con los equipos de la red. La aplicación correría un modelo predictivo de aprendizaje de máquina para series de tiempo encargado de estimar la demanda futura en base a la demanda pasada e información meteorológica como temperatura precipitación y radiación solar [4].

Finalmente, con el ecosistema completo, las fallas en la red eléctrica se terminarían solucionando en cuestión de segundos de forma automática sin necesidad de intervención humana.

1.3 Objetivos

1.3.1 Objetivo General

- Desarrollar una aplicación para la transferencia automática de carga de reconectadores en fallo para la red eléctrica de Guayaquil.

1.3.2 Objetivos Específicos

- Crear un modelo predictivo reentrenable capaz de inferir la demanda futura en base a datos recolectados
- Implementar automatismos locales en los RTU respectivo de cada circuito reconectador con las secuencias de transferencia local de carga.
- Diseñar un programa servidor capaz de proveer el servicio de estudio de carga al automatismo respectivo que lo requiera.

1.4 Marco teórico

1.4.1 Automatización de redes de distribución

La **automatización de redes de distribución** (ARD) es un proceso que utiliza tecnologías avanzadas para controlar, monitorear y operar automáticamente las redes eléctricas de distribución. Su objetivo es mejorar la eficiencia, la fiabilidad y la resiliencia de las redes mediante la automatización de funciones como la detección de fallas, el reencaminamiento de la energía, la restauración del servicio y la gestión de la demanda [5]. Los sistemas ARD permiten reducir el tiempo de inactividad, mejorar la calidad del servicio y facilitar la integración de fuentes de energía renovables [6].

1. **Dispositivos Utilizados**

Entre los dispositivos clave utilizados en la automatización de redes de distribución se encuentran [6]:

- **Relés de protección**: Un relé de protección es un dispositivo electromecánico o electrónico que se utiliza en sistemas eléctricos para proteger equipos y circuitos contra condiciones anormales o fallos. Su función principal es detectar fallos como sobrecargas, cortocircuitos, o fallos a tierra y activar mecanismos de protección, como interruptores automáticos, para desconectar el equipo afectado y evitar daños mayores.

Los relés de protección monitorizan diversas variables eléctricas, como corriente, voltaje, frecuencia y resistencia, y comparan estas medidas con valores de referencia predefinidos. Cuando detectan que una variable está fuera del rango seguro, el relé emite una señal para activar el dispositivo de protección adecuado.

Figura 1.1 Relé de protección y controlador de reconectador Eaton Cooper

- **Controladores remotos (RTU, Remote Terminal Units)**: El RTU, o "Remote Terminal Unit" (Unidad Terminal Remota), es un componente clave en la automatización y control de subestaciones eléctricas. Su función principal es recolectar, procesar y transmitir datos desde el equipo de la subestación a sistemas de control centralizados, como los sistemas SCADA (Control de Supervisión y Adquisición de Datos). Dependiendo del modelo permiten además correr programas o aplicaciones.

Figura 1.2 RTU marca Novatech

- **Reconectadores automáticos**: Un reconectador en una red eléctrica es un dispositivo automático diseñado para proteger y restaurar la continuidad del suministro eléctrico en caso de fallos transitorios o momentáneos en la red. Su función principal es detectar interrupciones en el flujo de electricidad y, tras un

breve período, intentar restablecer el suministro automáticamente. Al intentar restaurar automáticamente el suministro, los reconectadores ayudan a reducir el tiempo de inactividad y mejorar la confiabilidad del sistema eléctrico.

Figura 1.3 Reconectador SIEMENS

- **Medidores inteligentes**: Los medidores inteligentes son medidores de energía que permiten la capacidad de telemtería y accionamientos a distancia. Es decir que pueden transmitir a un sistema SCADA el consumo de la carga medida, además que muchos modelos permiten cortar el servicio remotamente.

Figura 1.4 Medidor ION

2. Protocolos de Comunicación

Los protocolos de comunicación utilizados en este proyecto son [7]:

- **DNP3 (Distributed Network Protocol)**: DNP3, o "Distributed Network Protocol version 3," es un protocolo de comunicación utilizado en sistemas de automatización de redes eléctricas y otras aplicaciones de control industrial. Se diseñó para mejorar la comunicación y la interoperabilidad entre equipos y sistemas en una red de distribución eléctrica. Sus características básicas es su

funcionalidad para gestión de eventos lo que permite un consumo menor del ancho de banda al solo transmitirse información al ocurrir un evento, es decir se transmite cuando el valor de la señal cambia. En su forma más común es usado en cables de red lo que le permite usar como base el modelo TCP/IP y usar el modelo DNP3 directamente en la capa de aplicación. La estructura de las capas de los mensajes es la siguiente:

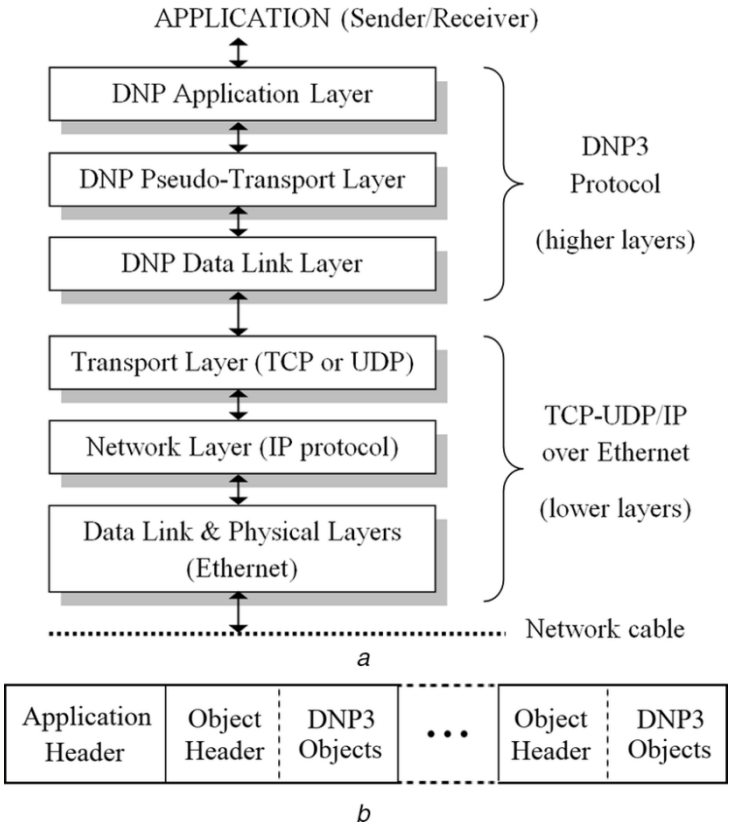

Figura 1.5 Capas del protocolo DNP3

- **Modbus**: Modbus es un protocolo de comunicación ampliamente utilizado en la automatización industrial y el control de sistemas. Fue desarrollado por Modicon (ahora parte de Schneider Electric) en 1979 para permitir la comunicación entre dispositivos en una red de automatización. Desde entonces, ha evolucionado y se ha convertido en un estándar abierto y ampliamente aceptado en la industria. El protocolo Modbus se basa en el modelo maestro esclavo, y el modelo cliente-servidor cuando usa como base el modelo TCP/IP. En una red Modbus, un maestro realiza polleos a una red de de esclavos mediante su identificador o su

IP, los esclavos responden informando sobre el contenido de registradoras específicas. El maestro necesita saber con antelación la dirección de la registradora en el esclavo que le será de interés. La estructura de sus mensajes es la siguiente:

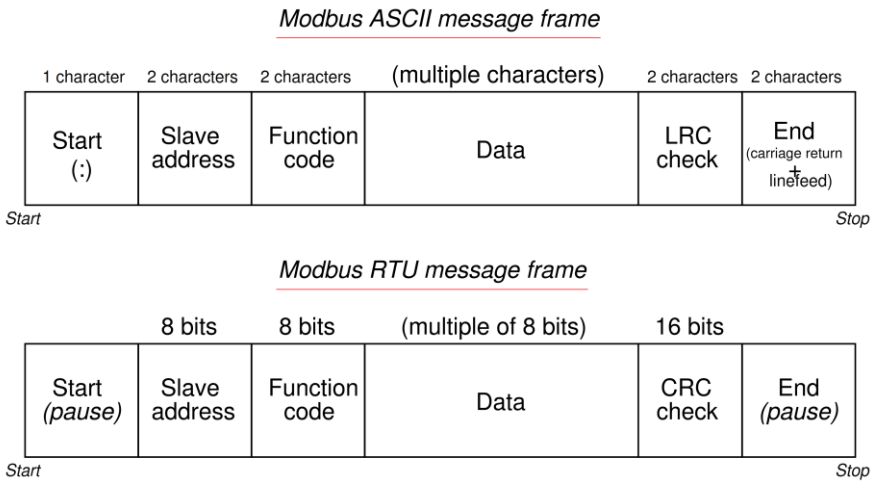

Figura 1.6 Trama protocolo Modbus

1.4.2 Time Series Forecasting

El *time series forecasting* (pronóstico de series temporales) es una técnica clave en la ciencia de datos que se utiliza para predecir valores futuros de una variable basada en datos históricos [4]. Este enfoque es especialmente útil cuando los datos son secuenciales, como las ventas diarias de un producto, la demanda eléctrica o las temperaturas a lo largo del tiempo.

Componentes de una Serie Temporal

Una serie temporal puede descomponerse en los siguientes componentes principales [4]:

1. **Tendencia**: Movimiento a largo plazo en la serie temporal, que puede ser creciente, decreciente o estable.
2. **Estacionalidad**: Patrones que se repiten en intervalos regulares de tiempo, por ejemplo, las fluctuaciones diarias o estacionales en los datos.
3. **Ciclos**: Fluctuaciones a largo plazo que no son necesariamente regulares y están frecuentemente ligadas a factores económicos o de otro tipo.
4. **Ruido**: Variabilidad aleatoria que no sigue ningún patrón identificable y puede distorsionar las observaciones de los otros componentes.

Modelado con aprendizaje automático

En los últimos años, técnicas como los árboles de decisión (LightGBM, XGBoost) y las redes neuronales recurrentes (LSTM, GRU) han ganado popularidad [4]. Estos modelos permiten capturar patrones más complejos en los datos, como relaciones no lineales y dependencias de largo plazo, lo que mejora la capacidad de predicción en escenarios complejos. En el caso de los modelos por árboles y gradient boosting, sus hiperparámetros de entrenamiento son: la razón de aprendizaje (que determina la velocidad de entrenamiento), la profundidad (que determina cuantos niveles tiene cada árbol), el número de estimadores (que determina cuantos árboles se usan), Alpha (un peso que pondera el primer término de regularización) y gamma (un peso que pondera el segundo termino de regularización)

1.4.3 LightGBM

LightGBM (Light Gradient Boosting Machine) es un algoritmo basado en técnicas de **gradient boosting** que construye modelos a partir de árboles de decisión. Su diseño está optimizado para ofrecer alta eficiencia y rendimiento, especialmente con grandes volúmenes de datos y alta dimensionalidad [8]. A continuación, se describe su funcionamiento y características clave [8].

1. **Funcionamiento de LightGBM**

1. **Gradient Boosting**:
    - LightGBM sigue el enfoque de **boosting por gradiente**, en el cual varios modelos débiles (generalmente árboles de decisión) se entrenan de manera secuencial. Cada árbol intenta corregir los errores residuales cometidos por el árbol anterior.
    - El objetivo es minimizar una función de pérdida (por ejemplo, error cuadrático medio o entropía cruzada), ajustando los predictores en función de los gradientes de esa función de pérdida.
2. **Construcción de Árboles Basada en Histogramas**:
    - LightGBM utiliza una técnica basada en **histogramas** para acelerar la construcción de los árboles. Los datos continuos se agrupan en bins discretos (histogramas), lo que reduce la cantidad de comparaciones necesarias en cada nodo del árbol y disminuye el costo computacional.
    - Esto permite que LightGBM sea mucho más rápido que otros algoritmos de boosting, como XGBoost, especialmente en grandes conjuntos de datos.
3. **Crecimiento por Hojas (Leaf-wise Growth)**:

- A diferencia de otros algoritmos que crecen los árboles de forma nivelada (level-wise), LightGBM emplea una estrategia de crecimiento basada en hojas (**leaf-wise**). Esto significa que, en cada iteración, LightGBM expande el nodo de la hoja con mayor error, lo que puede generar árboles más profundos y efectivos en la reducción del error.
- Aunque esta técnica mejora la precisión, también puede incrementar el riesgo de sobreajuste (overfitting). LightGBM introduce parámetros como la **profundidad máxima** y **número mínimo de observaciones por hoja** para controlar este comportamiento.

4. **Manejo Eficiente de Datos Categóricos**:
   - LightGBM incorpora mecanismos nativos para gestionar variables categóricas. En lugar de codificarlas manualmente, LightGBM puede tratar estos datos automáticamente durante el entrenamiento del modelo, mejorando tanto la eficiencia como la precisión.
5. **Reducción de la Complejidad Computacional**:
   - El algoritmo está optimizado para aprovechar al máximo los recursos de hardware. Puede paralelizar el entrenamiento y utilizar técnicas de optimización que reducen la memoria y el tiempo de cómputo, como la selección de características de forma incremental y la construcción de histogramas compartidos.

2. Características Clave:

- **Rapidez y Eficiencia**: LightGBM está diseñado para ser más rápido que otros algoritmos de boosting, especialmente en conjuntos de datos grandes y dispersos.
- **Mejor Rendimiento con Datos Grandes**: La técnica de histograma permite a LightGBM manejar eficientemente conjuntos de datos con millones de registros y características, lo que lo hace ideal para tareas de big data.
- **Soporte para Datos Categóricos**: La capacidad de manejar directamente variables categóricas sin necesidad de preprocesamiento mejora la simplicidad y rendimiento del modelo.

- **Control de Overfitting**: El crecimiento por hojas puede llevar a un sobreajuste, pero LightGBM introduce parámetros como la poda de hojas y regularización para evitar este problema.

3. **Proceso Básico de LightGBM:**
   - **Inicialización**: Comienza con un modelo inicial (usualmente una predicción promedio) y calcula el error residual.
   - **Cálculo de Gradientes**: En cada iteración, se calculan los gradientes de la función de pérdida con respecto a las predicciones actuales.
   - **Ajuste de Árbol**: Un nuevo árbol de decisión se ajusta para minimizar los gradientes residuales.
   - **Actualización de Predicciones**: Las predicciones se actualizan sumando el resultado del nuevo árbol al modelo existente.
   - **Repetición**: Este proceso se repite hasta que se alcanza un número predeterminado de iteraciones o se satisface un criterio de convergencia.

CAPÍTULO 2

2. SOLUCIÓN TECNOLÓGICA IMPLEMENTADA

2.1 Descripción de los sectores a automatizar

Debido a la considerable dimensión de la red eléctrica de Guayaquil, este trabajo se limitó a automatizar los sectores de cerro azul y cerro del Carmen, dos sectores que mostrarán el concepto detrás de la idea de usar aprendizaje de máquina para transferir carga, y que a su vez son de gran importancia debido a que energizan la mayor parte de las comunicaciones de la ciudad.

2.1.1 Sector Cerro del Carmen

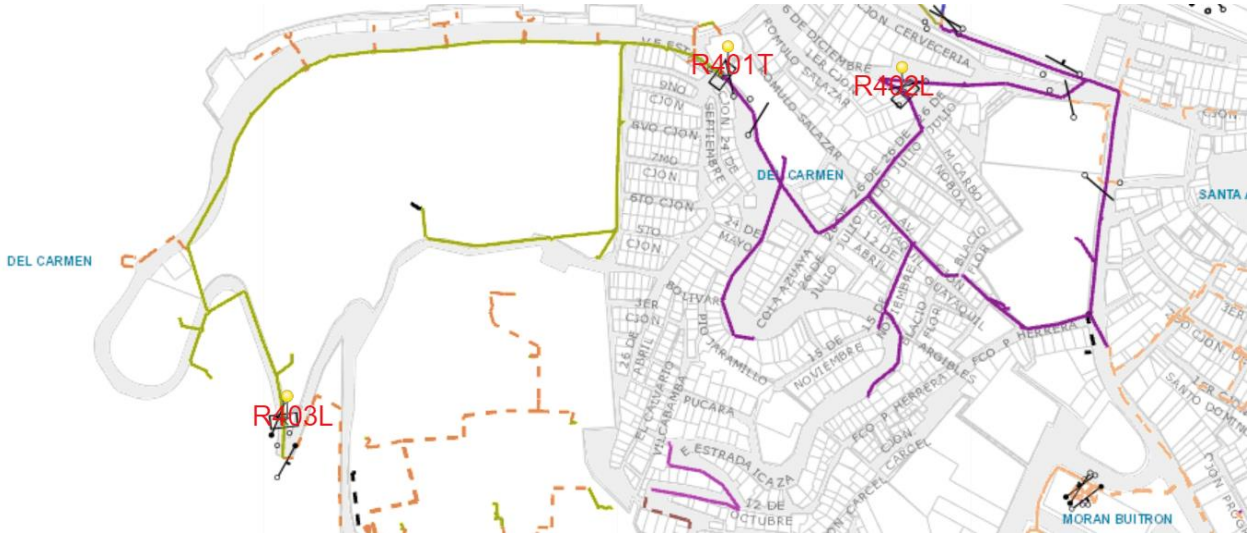

Figura 2.1 Circuito Cerro del Carmen

Como se observa en la figura superior, las cargas del cerro del Carmen se encuentran repartidas a lo largo de la alimentadora “Panamá” de la subestación “Boyacá” (amarillo) y la alimentadora “Puerto Santa Ana 2” de la subestación “Puerto Santa Ana” (morado). Tanto la alimentadora panamá como la alimentadora “Pto Sta Ana 2” tiene un reconectador NC (normalmente cerrado), y ambas alimentadoras tienen un punto en común que se puede unir a través del reconectador NA (normalmente abierto) R401T.

De esta forma en caso de ciertas fallas en el alimentador “Panamá” es posible transferir carga desde “Pto. Sta. Ana 2”, y viceversa.

2.1.2 Sector Cerro Azul

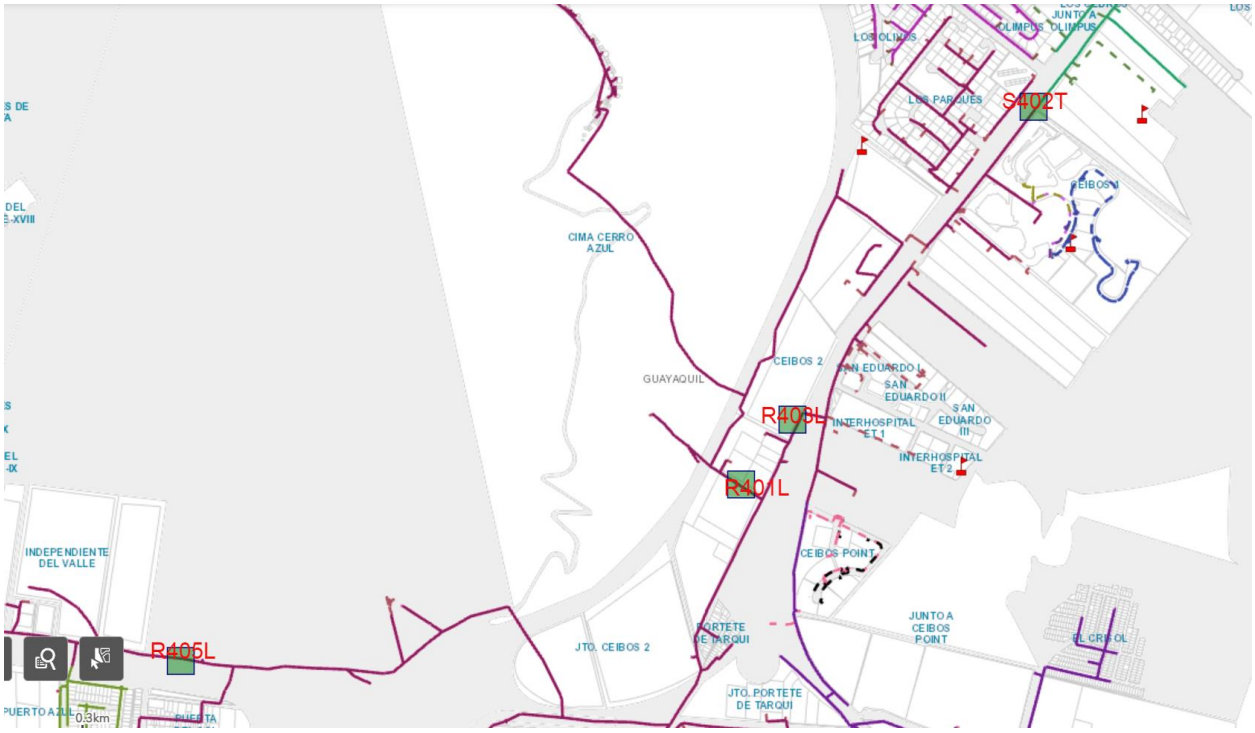

Figura 2.2 Circuito Cerro Azul

Como se observa en la figura superior, existen 2 reconectadores (R405L y R403L) en la troncal del alimentador “Bosques de la Costa” de la subestación “Belo Horizonte” (morado), existe además el reconectador R401L que sale en un ramal hacía cerro azul y es este el reconectador que controla la carga hacia las comunicaciones. Se encuentra también el seccionador normalmente abierto S402T en el punto de unión entre el alimentador “Bosques de la Costa” y “Los Ceibos” de la subestación “Ceibos” (verde).
Esta configuración de equipos permite que, si ocurrieran fallas en la alimentadora “Bosques de la Costa”, los reconectadores adecuados podrán abrirse y cerrase de forma que se aísle el sector con falla, y el seccionador S402T se abriría de ser necesario para transferir carga desde el alimentador Ceibos hasta “Bosques de la Costa”

2.2 Arquitectura del sistema

El sistema creado para realizar los automatismos de transferencia y correr el modelo predictivo de carga es el siguiente.

2.2.1 Conexión física

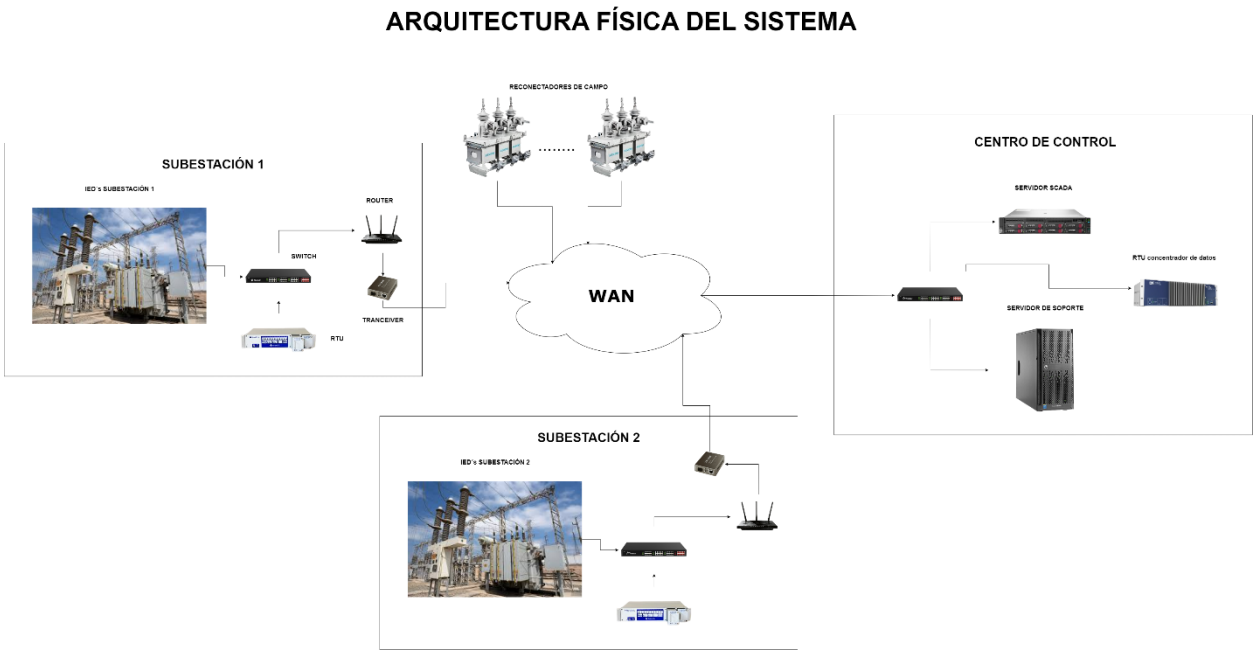

Figura 2.3 Conexión física del sistema

Como se puede observar en la figura superior, los reconectadores correspondientes a cada automatismo se encuentran conectados por medio de la red WAN de la empresa distribuidora con el RTU de la subestación más cercana. Así mismo, los RTU de cada subestación se encuentran conectados tanto a un concentrador de datos central como al servidor SCADA por medio de la misma red WAN.

La distribución lógica de los distintos programas con sus comunicaciones es la siguiente:

2.2.2 Conexión lógica

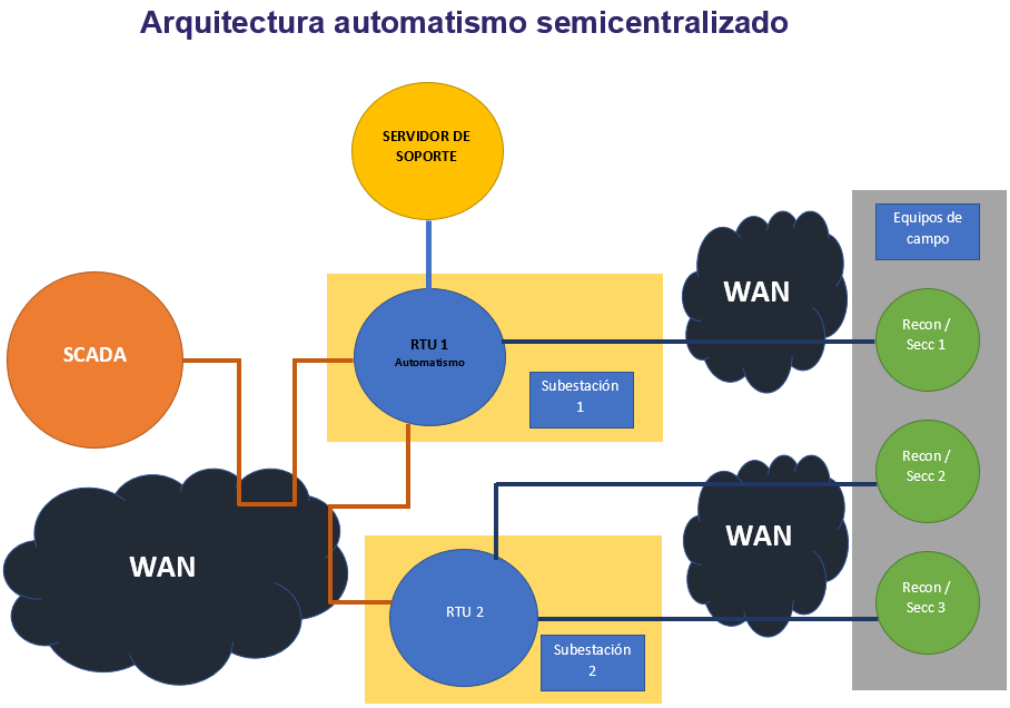

Figura 2.4 Conexión lógica

El automatismo de Cerro del Carmen se encuentra ubicado en la RTU de “Puerto Santa Ana”, este recibe vía DNP3 las señales de los reconectadores cercanos, y también de los reconectadores lejanos vía la RTU “Boyacá”. El automatismo de Cerro Azul se encuentra ubicado en la RTU de “Belo Horizonte”, este recibe vía DNP3 las señales de los reconectadores cercanos, y también de los reconectadores lejanos vía la RTU Ceibos

Ambos automatismos envían las señales de potencia, corriente y señales de estado de todos los dispositivos al concentrador de datos del centro de control vía DNP3. El concentrador de datos del centro del centro de control envía a su vez todas las señales juntas al servidor de soporte vía MODBUS TCP. El servidor de soporte se encarga de guardar todos estos datos en una base de datos PostgreSQL.

El servidor de soporte corre dentro de sí un pipeline de aprendizaje de máquina (usando la información de la base de datos PostgreSQL) que predice las siguientes 24h de

demanda o corriente. Este servicio es pedido directamente por el automatismo en los RTU de la subestación a la hora de tomar decisiones para el automatismo. El servicio de predicción lo piden los automatismos mediante una conexión MODBUS TCP.

Finalmente, los automatismos intercambian información con el servidor SCADA en el centro de control, para mostrar la información necesaria en las animaciones de las pantallas respectivas del sistema SCADA de la empresa distribuidora.

2.3 Automatismos locales

2.3.1 Automatismo local circuito cerro del Carmen

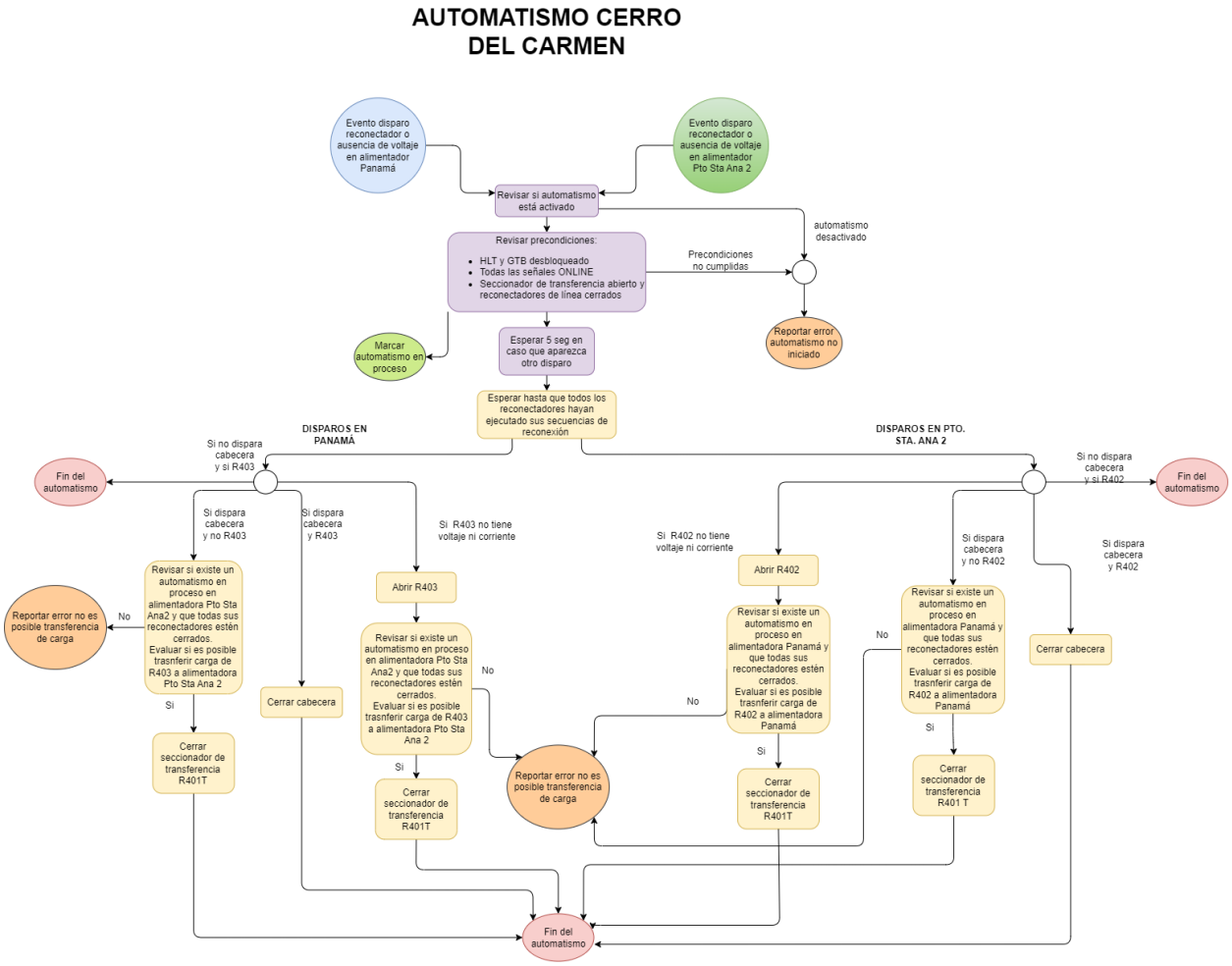

Figura 2.5 Flujograma automatismo cerro del Carmen

La figura superior describe el automatismo local corriendo en la RTU de cerro del Carmen. El automatismo trata con dos tipos de fallas, disparos por cortocircuito o sobrecorriente, y ausencia de voltaje provocada por una pérdida de la línea. El objetivo del automatismo es el de aislar el sector de la alimentadora con la falla por medio de los reconectadores y el de transferir carga para rehabilitar los sectores que sean posibles.

En este automatismo se presentan caso por caso las distintas combinaciones de reconectadores abiertos y cerrados con sus acciones correspondientes. Cuando se aisla una falla en el alimentador “Panamá” y es posible transferir carga, se transfiere carga del alimentador “Puerto Santa Ana 2”.
Lo mismo ocurre si la falla se encuentra en el alimentador “Puerto Santa Ana 2”, en este caso se intenta transferir carga desde el alimentador Panamá.

2.3.2 Automatismo local circuito cerro azul

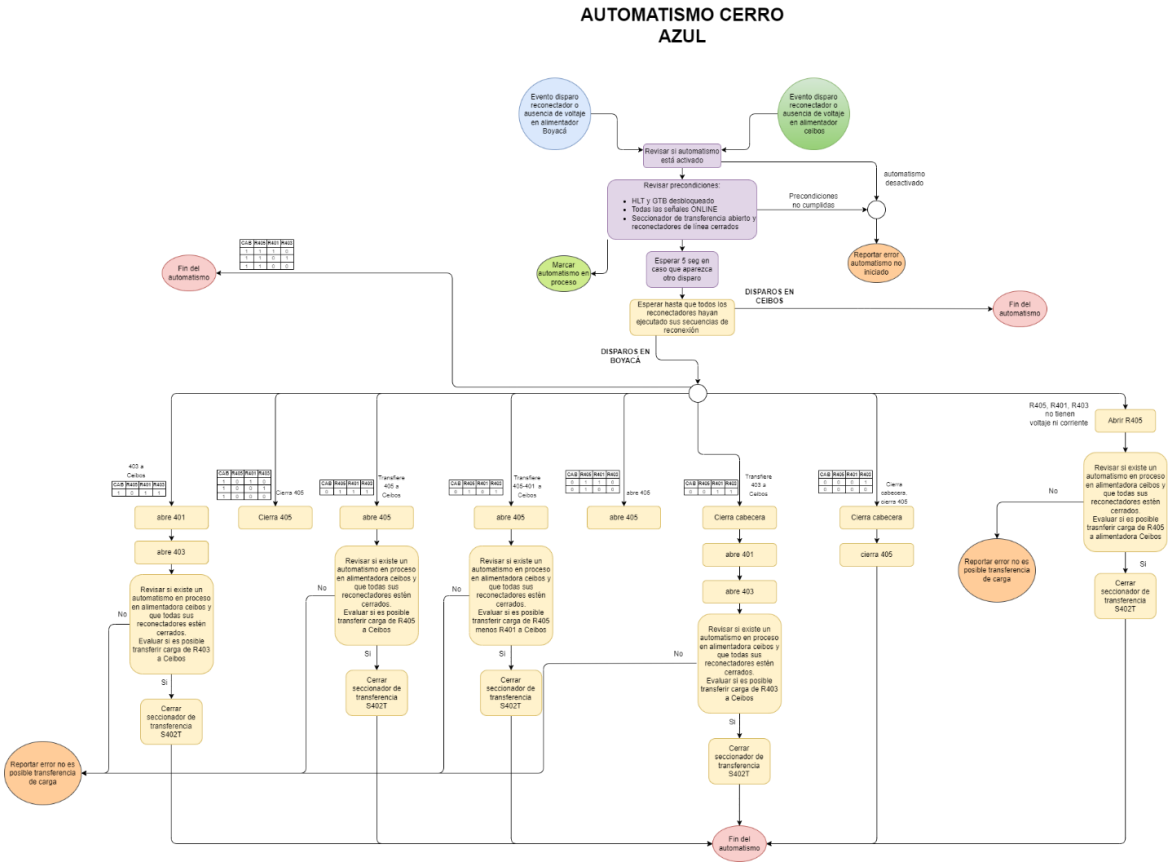

Figura 2.6 Flujograma automatismo cerro azul

La figura superior describe el automatismo de cerro azul. Este funciona de la misma forma que el anterior. El circuito de cerro azul tiene dos alimentadoras en contraposición separadas por un seccionador normalmente abierto usado para transferir carga.
Dado que hay 4 reconectadores hay 16 combinaciones, todas se describen en el flujograma arriba.
El objetivo del programa es así mismo el de detectar la falla, aislarla por medio de los reconectadores, y si es posible rehabilitar los sectores que se puedan por medio de la transferencia de la alimentadora opuesta.

Ambos automatismos requieren del servidor de soporte para predecir la carga de transferencia. Esta predicción se realiza en este servidor y se envía la predicción a las RTU por medio de una conexión MODBUS TCP.

2.4 Servidor de soporte usando Node-RED

El servidor de soporte consta de una computadora en el centro de control corriendo un servidor en Node-RED. Como se explicó en la sección de arquitectura, un concentrador de datos recopila todas las señales necesarias en un RTU en el centro del control. El servidor de soporte tiene una conexión MODBUS TCP con este RTU para leer en tiempo real todas las señales de interés, luego parsea estas señales para poder establecer las variables requeridas en los tipos de dato requeridos (float32 ó bool).

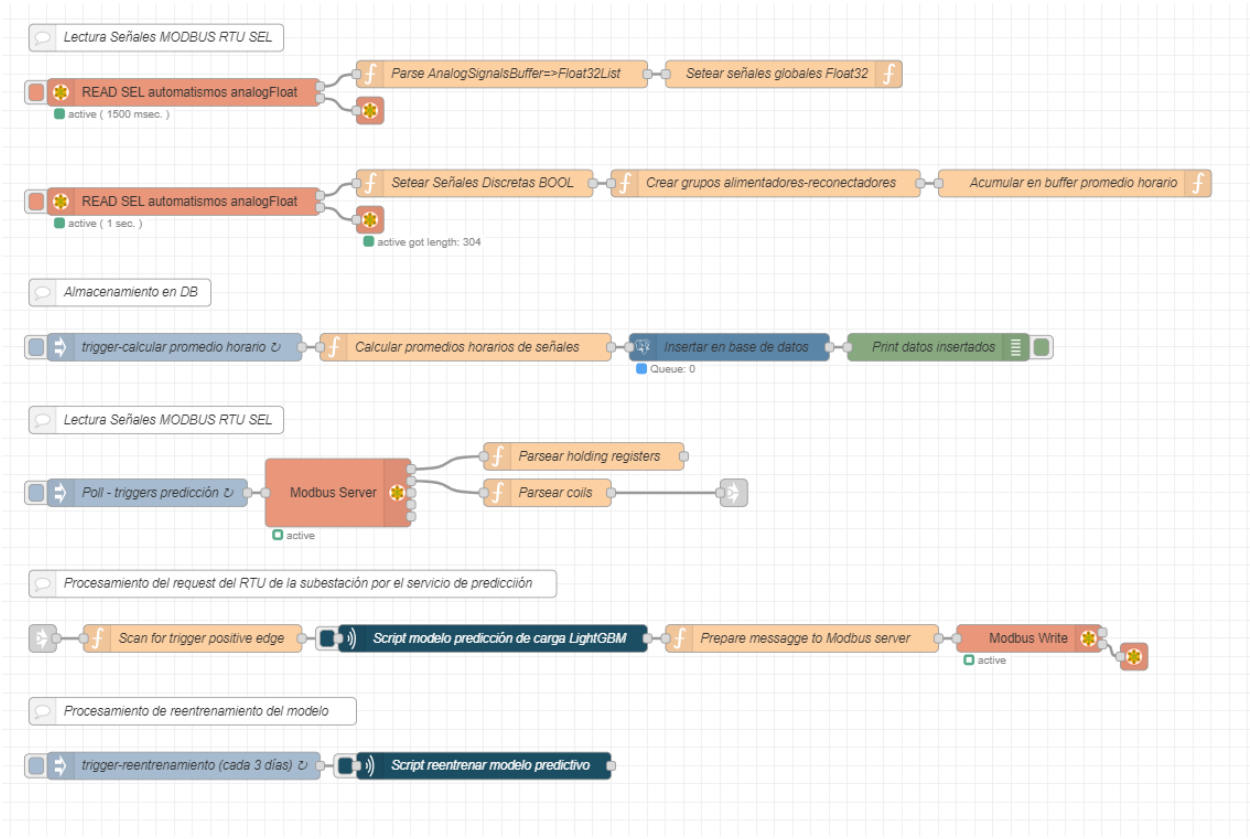

Figura 2.7 Flujos de procesos del servidor de soporte en Node-RED

La figura superior muestra los flujos de node-RED que conforman el servidor de soporte, este corre los siguientes procesos:

- Cliente MODBUS-TCP: Se usa para recopilar señales de interés de las subestaciones por medio del RTU concentrador en el centro de control.
- Procesado y promediado de variables: Las variables vía Modbus vienen repartidas en varias registradoras con varias particularidades por lo que deben ser parseadas al tipo de dato requerido para extraer la información. Una vez parseadas todas las variables capturadas en tiempo real se necesitan solamente con una frecuencia horaria, por lo que se crea un buffer en memoria que almacena todas las variables de forma temporal para promediarlas de forma horaria.
- Guardado en base de datos: Se graban regularmente las variables leidas en una base de datos PostgreSQL con una frecuencia horaria.

- Servidor MODBUS-TCP: Este servidor actúa como esclavo de todos los automatismos corriendo en las RTU de cada subestación. Los automatismos como clientes envían request del servicio de predición. El programa en Node-RED recibe el request por medio de la escritura en las registradoras del servidor Modbus. Una vez recibido el request, el servidor de soporte corre el modelo de aprendizaje de máquina y envía las 24 predicciones futuras al RTU que las solicitó por medio de las registradoras del servidor MODBUS-TCP.
- Script de predicción: Cuando se recibe el request de predicción por medio del servidor Modbus, se ejecuta el script de predicción por medio del nodo Python Shell. Este Script carga el modelo predictivo de la memoria, lee los datos de la base de datos PostgreSQL y los utiliza para predecir la carga futura por medio del algoritmo de gradient boosting con la librería skforecast y LightGBM.
- Proceso de reentrenamiento: Cada 3 días, el modelo se reentrena usando los 5 años de datos más recientes.

2.5 Base de datos

En la base de datos se guardan las corrientes de las fases Ia, Ib, Ic y la potencia aparente en KVA de cada uno de los reconectadores de línea y cabecera, además de la potencia en KVA del transformador de las subestaciones Ceibos, Puerto Santa Ana 2 y Panamá, todo con el timestamp de la fecha y hora específica de la variable.

2.6 Motor predictivo

El motor predictivo conforma el núcleo del sistema, es un script en Python que es llamado por el flujo del servidor de Node-RED. El script fue desarrollado por medio de las librerías skforecast, scikit-learn y LightGBM. Skforecast es una librería que permite hacer modelos de aprendizaje de máquina en series de tiempo usando como algoritmos de predicción otras librerías compatibles con scikit-Learn, y permite realizar entrenamiento de modelos, validación cruzada, y optimización de hiperparámetros. Scikit-learn es una librería general de aprendizaje de máquina con muchos modelos y herramientas que forma la base de skforecast. Finalmente, LightGBM es un algoritmo de aprendizaje de máquina que permite predecir cajas negras con múltiples inputs numéricos o categóricos, y múltiples outputs. LightGBM es una generalización del modelo de *gradient boosted trees*, el cual reúne un conjunto elevado de árboles de decisión débiles (con una certeza solo

ligeramente superior al 50%) y los junta para crear un modelo que aumenta dramáticamente la precisión y es resistente al sobreajuste.

El objetivo del motor es el de predecir las siguientes 24h de carga a transferir + la carga del alimentador o trasformador al cual se va a transferir la carga. Esto se logra extrayendo los datos históricos de carga guardados en la base de datos, y junto con datos meteorológicos obtenidos mediante un API se predicen las siguientes 24h de la carga conjunta. Esta información se usa para determinar si es posible hacer la transferencia por el automatismo dentro de las RTU en las subestaciones.

Se presenta a continuación extractos del script de predicción aplicados a la potencia del transformador Ceibos2 + R405L del automatismo de cerro azul para mostrar el flujo de trabajo para entrenar el motor predictivo, todo realizado de acuerdo a las recomendaciones del autor de la librería [9].

2.6.1 Extraer información de la base de datos

En el caso del automatismo de cerro azul se necesitará predecir entre otras cosas la suma de la potencia del reconectador R405L + el trasformador Ceibos2. Lo primero que necesitamos es extraer los últimos 5 años de datos de la DB PostgreSQL (43800 horas) para las variables de corriente y potencia del reconectador y corriente del alimentador de llegada (Ceibos) y la potencia del trasformador de llegada (Ceibos2).

```python
def ConsultarHistoricoVariable(str_pointname, crs):

    crs.execute(f"""SELECT a.datetime, p.pointname, a.value
    FROM pointnames p, archive a
    WHERE (a.pt_id = p.pt_id) AND  p.pointname = '{str_pointname}'
    ORDER BY a.datetime DESC LIMIT 50000;"""
    )

    columnas = [x.name for x in crs.description]
    rows=crs.fetchall()
    df_tablaVariable = pd.DataFrame(rows, columns = columnas)
    df_tablaVariable.sort_values(by = "datetime", inplace = True)
    df_tablaVariable.reset_index( drop = True, inplace=True)
    df_tablaVariable=df_tablaVariable.set_index("datetime")
    df_tablaVariable=df_tablaVariable.drop("pointname", axis=1)

    return df_tablaVariable

try:
    #Conectar a base de datos en RTU BeloH
    conn = psycopg2.connect(database="postgres",
                            host=DB_IPAdress_RTUAtomatismo,
                            user="rtu_belo_horizonte",
                            password="*******",
                            port="5432")
    conn.autocommit=True
    #Crear Cursor
    cursor = conn.cursor()
    #Consultar Datos Ia Reconectador Desde
    df_Ia_ReconectadorDesde = ConsultarHistoricoVariable(pointname_Ia_ReconectadorDesde, crs = cursor)/100
    df_Ib_ReconectadorDesde = ConsultarHistoricoVariable(pointname_Ib_ReconectadorDesde, crs = cursor)/100
    df_Ic_ReconectadorDesde = ConsultarHistoricoVariable(pointname_Ic_ReconectadorDesde, crs = cursor)/100
    df_S_ReconectadorDesde = ConsultarHistoricoVariable(pointname_S_ReconectadorDesde, crs = cursor)*10 # KVA
    df_Ia_AlimentadorHasta = ConsultarHistoricoVariable(pointname_Ia_AlimentadorHasta, crs = cursor)/100
    df_Ib_AlimentadorHasta = ConsultarHistoricoVariable(pointname_Ib_AlimentadorHasta, crs = cursor)/100
    df_Ic_AlimentadorHasta = ConsultarHistoricoVariable(pointname_Ic_AlimentadorHasta, crs = cursor)/100
    df_S_TrafoAlimentadorHasta = ConsultarHistoricoVariable(pointname_S_TrafoAlimentadorHasta, crs = cursor)/100

except (Exception, psycopg2.Error) as err:
    print("Problema en consulta a la base de datos:\n", err)

finally:

    if conn:
        cursor.close()
        conn.close()
```

Figura 2.8 Código extracción de información de base de datos (DB)

2.6.2 Separación train-validation-test sets

De los datos extraídos de la DB, asignamos 76% de los datos como el dataset de entrenamiento, asignamos 1 mes de datos como el dataset de testing, y asignamos el resto al dataset de validación. Estas proporciones las obtenemos de acuerdo a la literatura dado que se deja el mínimo tiempo significativo para pruebas que sería 1 mes, cerca de 80% para train, y el resto como validación. Usaremos dataset de entrenamiento para entrenar el modelo inicial, mientras que el dataset de validación será para entrenar

distintas versiones del modelo inicial con distintos hiperparámetros. Una vez obtenida la mejor combinación de hiperparámetros se entrenará una vez más con el dataset de entrenamiento + validación. Finalmente, en el dataset de prueba se determinará la exactitud del modelo.

```
# Separación datos train-val-test
# ======================================================================
datos = datos.loc['2019-01-01 00:00:00': '2024-04-01 00:00:00'].copy()
fin_train = '2022-12-31 23:00:00'
fin_validacion = '2023-12-31 23:00:00'
datos_train = datos.loc[: fin_train, :].copy()
datos_val   = datos.loc[fin_train:fin_validacion, :].copy()
datos_test  = datos.loc[fin_validacion:, :].copy()

print(f"Fechas train      : {datos_train.index.min()} --- {datos_train.index.max()}  (n={len(datos_train)}) -- {100*len(datos_train)/len(datos):0.2f}%")
print(f"Fechas validacion : {datos_val.index.min()} --- {datos_val.index.max()}  (n={len(datos_val)}) -- {100*len(datos_val)/len(datos) :0.2f}%")
print(f"Fechas test       : {datos_test.index.min()} --- {datos_test.index.max()}  (n={len(datos_test)}) -- {100*len(datos_test)/len(datos):0.2f}%")
```

Figura 2.9 Código separación de datos en train-validation-test

2.6.3 Exploración del dataset

En el script final que corre en el flujo de Node-RED no se grafican datos, puesto que todo se procesa en un solo pipeline con un output puramente numérico. Sin embargo, es importante tener una idea de los datos que el modelo usa para entrenarse.

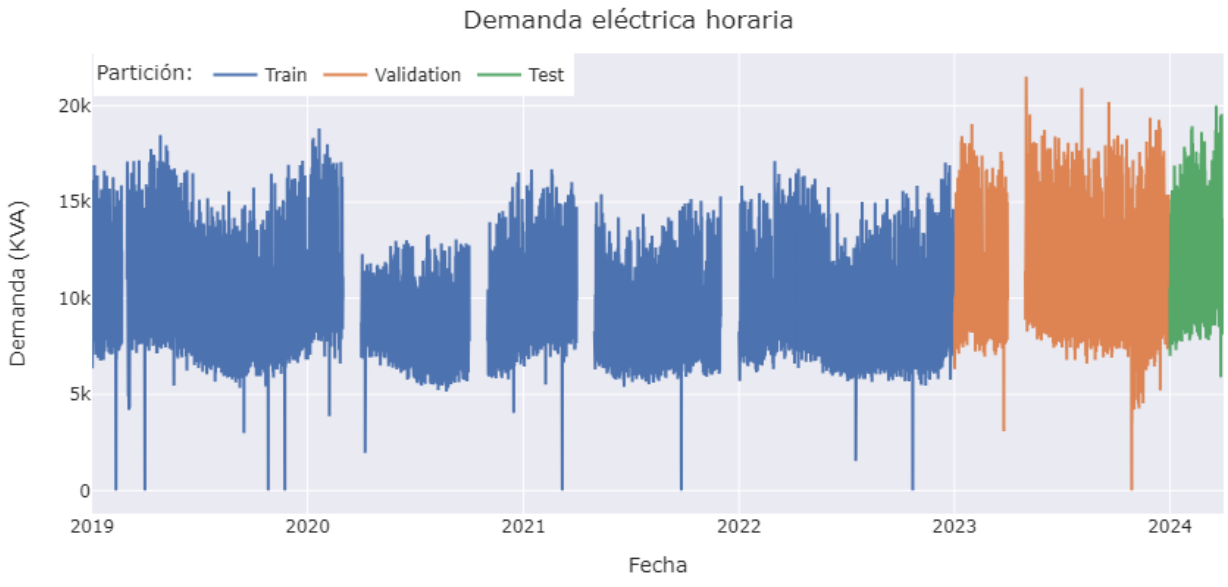

Figura 2.10 Potencia eléctrica del transformador Ceibos2 + el reconectador R405L

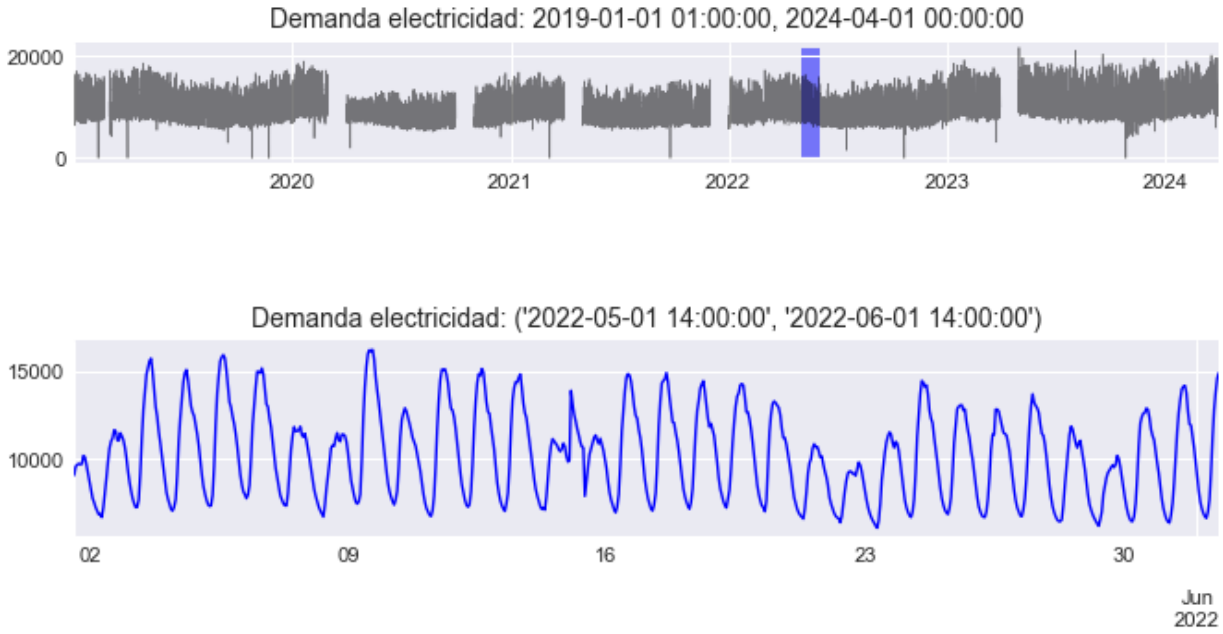

figura 2.11 Zoom de la figura anterior mostrando estacionalidad semanal y diaria

Como se puede observar en las figuras superiores los datos de demanda son cíclicos y por temporada, existen repeticiones cíclicas a nivel anual, semanal e incluso diario, estas son características que deberá explotar el modelo predictivo a la hora de escoger los features de entrada. Adicional, tenemos importantes agujeros en los datos históricos, estos pueden ir desde datos puntuales, hasta agujeros de meses sin datos. Esto es posible debido a fallas de comunicación entre los reconectadores y el SCADA, o debido a equipos dañados que demoran en reparase. A la hora de entrenar el modelo se tomarán medidas para lidiar con este problema, específicamente se asignarán pesos a distintos intervalos del dataset, y se asignará un peso de cero a todos los intervalos faltantes.

2.6.4 Análisis de lags

Siguiendo la tradición del análisis clásico de series de tiempo se observan los graficos de autocorrelación y gráfico de autocorrelación parcial

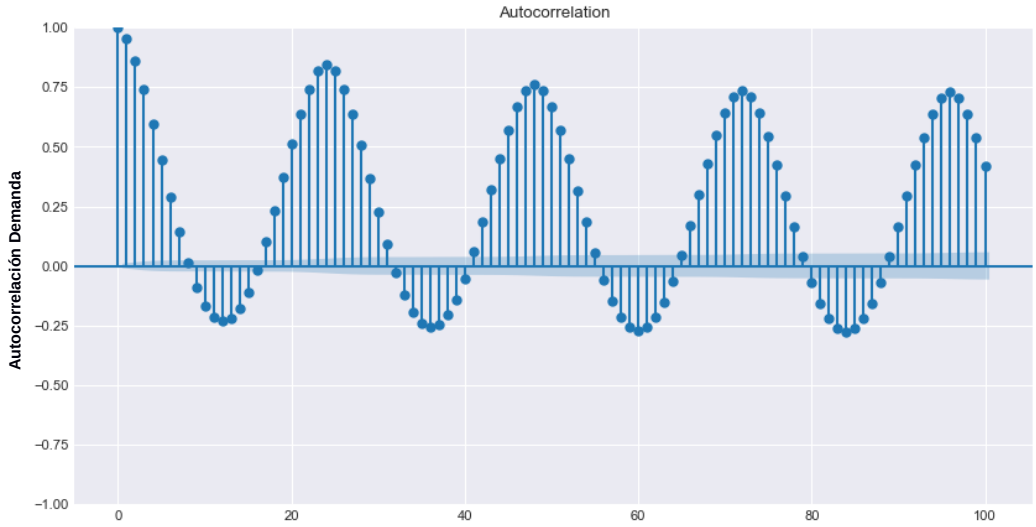

Figura 2.12 Autocorrelación de los datos de entrenamiento de la demanda transferida(KVA)

En el gráfico de autocorrelación observamos que la autocorrelación es periódica con un perído de 24 lags (1 lag por hora). El gráfico de autocorrelación sería suficiente para determinar cuantos lags son necesarios para predecir si el modelo predictivo fuera de Moving Average puro (MA), dado que esto es altamente improbable, el valor de 24 lags solo queda como hiperparámetro tentativo que deberá ser probado al optimizar los hiperparámetros.

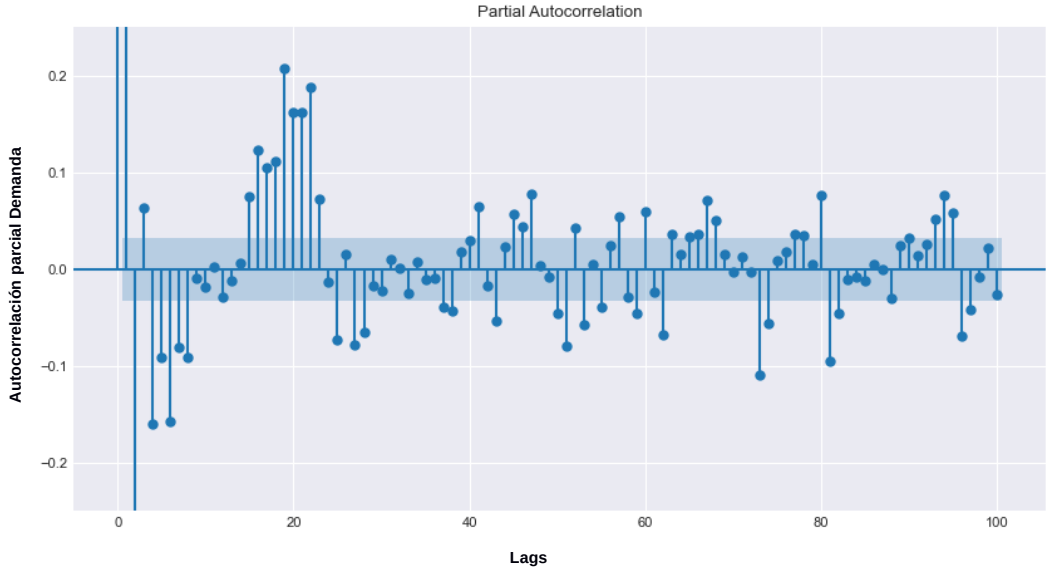

Figura 2.13 Autocorrelación parcial de los datos de entrenamiento de la demanda transferida (KVA)

En la figura superior se observa el gráfico de autocorrelación parcial, el cual mide la correlación del valor actual con cada uno de los valores pasados y compensa además por el efecto acumulativo entre ellos. En el gráfico superior podemos observar que los valores de:
[1,2,3,4,5,6,7,8,12,15,16,17,18,19,20,21,22,23,25,27,28,37,38,40,41,43,45,46,47,50,51,52,53,54,55,56,57,58,59,60] tienen valores superiores a la desviación estándar (sombra azul) lo cual sugiere que son significativos y deben ser probados a la hora de optimizar hiperparámetros.

2.6.5 Preprocesar valores faltantes

Antes de asignar pesos de cero a los valores faltantes, debemos rellenar los agujeros con datos, para esto usaremos interpolación lineal.

```
#Interpolar valores faltantes
datos['kVA sd del'] = datos['kVA sd del'].interpolate(method='linear')
datos['Ia'] = datos['Ia'].interpolate(method='linear')
datos['Ib'] = datos['Ib'].interpolate(method='linear')
datos['Ic'] = datos['Ic'].interpolate(method='linear')
datos_train = datos.loc[: fin_train, :].copy()
datos_val   = datos.loc[fin_train:fin_validacion, :].copy()
datos_test  = datos.loc[fin_validacion:, :].copy()
```

Figura 2.14 Código interpolación valores faltantes

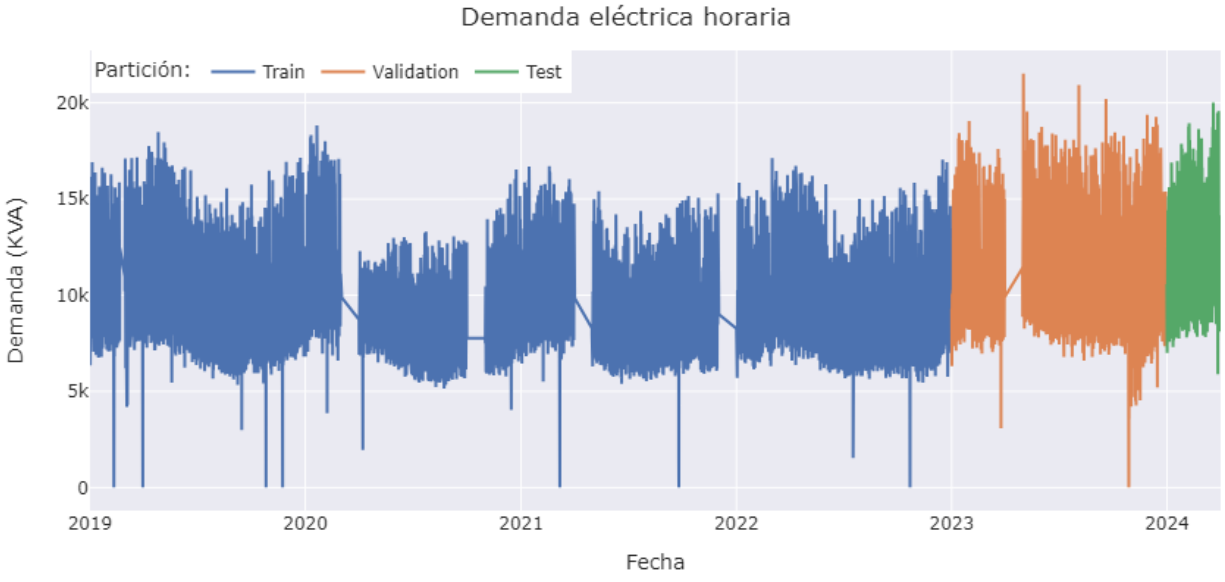

Figura 2.15 Datos demanda eléctrica con valores faltantes interpolados

Se interpoló linealmente los valores faltantes, sin embargo, como se puede apreciar en la figura superior, los intervalos faltantes son demasiado extensos para simplemente interpolar, por lo que desarrollaremos una función que asigna pesos de cero a los intervalos faltantes para que estos no sean tomados en cuenta por el algoritmo.

```
# Custom function to create weights
# ============================================================================
def custom_weights(index):
    """
    Return 0 if index is in any gap.
    """
    index2020 = pd.date_range(
                    start = pd.to_datetime("2020-03-01 00:00:00"),
                    end   = pd.to_datetime("2021-01-01 00:00:00"),
                    freq  = 'h'

    )
    combinedMissingIndex = index_na_missing_dates.union(index2020)

    weights = np.where(index.isin(combinedMissingIndex), 0, 1)

    return weights
```

Figura 2.16 Función asignadora de pesos a intervalos faltantes

En la figura superior podemos observar la función usada para asignar pesos a los intervalos de tiempo. Vemos que al intervalo del 1 de marzo de 2020 al 1 de enero de 2021 lo unimos a todos los intervalos donde hay fechas faltantes. A dicha unión de intervalos le asignamos un peso de cero.

2.6.6 Forecasting con variables exógenas

Hasta el momento hemos extraído de la base de datos los datos históricos de la variable a predecir (demanda energética en KVA). Los otros inputs del modelo son llamados variables exógenas, a continuación, comenzaremos a prepararlos:

2.6.6.1 Preparar variables solares y de calendario

```
# Variables basadas en el calendario
# ==============================================================================
variables_calendario = pd.DataFrame(index=datos.index)
variables_calendario['mes'] = variables_calendario.index.month
variables_calendario['semana_anyo'] = variables_calendario.index.isocalendar().week
variables_calendario['dia_semana'] = variables_calendario.index.day_of_week + 1
variables_calendario['hora_dia'] = variables_calendario.index.hour + 1

# Variables basadas en la luz solar
# ==============================================================================
lat_GYE = -2.1962
lon_GYE = -79.8862
location = LocationInfo(
    "Guayaquil",
    "Ecuador",
    latitude=lat_GYE,
    longitude=lon_GYE,
    timezone='America/Guayaquil'
)
hora_amanecer = [
    sun(location.observer, date=date, tzinfo=location.timezone)['sunrise'].hour
    for date in datos.index
]
hora_anochecer = [
    sun(location.observer, date=date, tzinfo=location.timezone)['sunset'].hour
    for date in datos.index
]
variables_solares = pd.DataFrame({
                        'hora_amanecer': hora_amanecer,
                        'hora_anochecer': hora_anochecer},
                        index = datos.index
                    )
variables_solares['horas_luz_solar'] = (
    variables_solares['hora_anochecer'] - variables_solares['hora_amanecer']
)
variables_solares['es_de_dia'] = np.where(
                                    (datos.index.hour >= variables_solares['hora_amanecer']) & \
                                    (datos.index.hour < variables_solares['hora_anochecer']),
                                    1,
                                    0
                                 )
```

Figura 2.17 Código variables basadas en luz solar

Como se puede observar en la figura superior, se preparan inputs adicionales para el modelo, se crean como variables adicionales, el día, mes, semana y hora del día. Además, se crean variables basadas en la luz solar como son la **hora del amanecer** del día, la **hora de anoche** del día, las **horas de luz solar** del día, y el **estado si es de día** o no.

2.6.6.2 Preparar variables tipo Holiday

Adicionalmente, se crea otro input denotando si el día es feriado o no.

```python
# Variables basadas en festivos
# ==============================================================================
datos['Holiday'] = pd.Series(data = datos.index, index = datos.index).map(isHolidayGuayaquileño)
variables_festivos = datos[['Holiday']].astype(int)
variables_festivos['holiday_dia_anterior'] = variables_festivos['Holiday'].shift(24)
variables_festivos['holiday_dia_siguiente'] = variables_festivos['Holiday'].shift(-24)
```

Figura 2.18 Código insertar estado feriado o no

2.6.6.3 Preparar variables de temperatura humedad y precipitación

```python
# Temperature features
# ==============================================================================

lat_GYE = -2.1962
lon_GYE = -79.8862
fecha_inicio_datos = datos.index.date.min().strftime("%Y-%m-%d")
fecha_final_datos = datos.index.date.max().strftime("%Y-%m-%d")
historical_url = f"""https://archive-api.open-meteo.com/v1/archive?latitude={lat_GYE}&longitude={lon_GYE}&start_date={fecha_inicio_datos}
&end_date={fecha_final_datos}&hourly=temperature_2m,relative_humidity_2m,apparent_temperature,rain,direct_radiation&timezone=America%2FGuayaquil"""
response_temp = requests.get(historical_url).json()

variables_temp = pd.DataFrame({
                    'temperature_2m': response_temp["hourly"]["temperature_2m"],
                    'relative_humidity_2m': response_temp["hourly"]["relative_humidity_2m"],
                    'apparent_temperature': response_temp["hourly"]["apparent_temperature"],
                    'rain': response_temp["hourly"]["rain"],
                    'direct_radiation': response_temp["hourly"]["direct_radiation"],
                    index = response_temp["hourly"]["time"]
                )

variables_temp.index = pd.to_datetime(variables_temp.index) #Convertir index de str a datetime
variables_temp = variables_temp.loc[datos.index.min():datos.index.max()] # Eliminar datos mas allá del rango de fechas de entrenamiento

variables_temp['temp_roll_mean_1_dia'] = variables_temp['temperature_2m'].rolling(24, closed='left').mean()
variables_temp['temp_roll_mean_7_dia'] = variables_temp['temperature_2m'].rolling(24*7, closed='left').mean()
variables_temp['temp_roll_max_1_dia'] = variables_temp['temperature_2m'].rolling(24, closed='left').max()
variables_temp['temp_roll_min_1_dia'] = variables_temp['temperature_2m'].rolling(24, closed='left').min()
variables_temp['temp_roll_max_7_dia'] = variables_temp['temperature_2m'].rolling(24*7, closed='left').max()
variables_temp['temp_roll_min_7_dia'] = variables_temp['temperature_2m'].rolling(24*7, closed='left').min()

variables_temp['hum_roll_mean_1_dia'] = variables_temp['relative_humidity_2m'].rolling(24, closed='left').mean()
variables_temp['hum_roll_mean_7_dia'] = variables_temp['relative_humidity_2m'].rolling(24*7, closed='left').mean()
variables_temp['hum_roll_max_1_dia'] = variables_temp['relative_humidity_2m'].rolling(24, closed='left').max()
variables_temp['hum_roll_min_1_dia'] = variables_temp['relative_humidity_2m'].rolling(24, closed='left').min()
variables_temp['hum_roll_max_7_dia'] = variables_temp['relative_humidity_2m'].rolling(24*7, closed='left').max()
variables_temp['hum_roll_min_7_dia'] = variables_temp['relative_humidity_2m'].rolling(24*7, closed='left').min()

variables_temp['aptemp_roll_mean_1_dia'] = variables_temp['apparent_temperature'].rolling(24, closed='left').mean()
variables_temp['aptemp_roll_mean_7_dia'] = variables_temp['apparent_temperature'].rolling(24*7, closed='left').mean()
variables_temp['aptemp_roll_max_1_dia'] = variables_temp['apparent_temperature'].rolling(24, closed='left').max()
variables_temp['aptemp_roll_min_1_dia'] = variables_temp['apparent_temperature'].rolling(24, closed='left').min()
variables_temp['aptemp_roll_max_7_dia'] = variables_temp['apparent_temperature'].rolling(24*7, closed='left').max()
variables_temp['aptemp_roll_min_7_dia'] = variables_temp['apparent_temperature'].rolling(24*7, closed='left').min()

variables_temp['rain_roll_mean_1_dia'] = variables_temp['rain'].rolling(24, closed='left').mean()
variables_temp['rain_roll_mean_7_dia'] = variables_temp['rain'].rolling(24*7, closed='left').mean()
variables_temp['rain_roll_max_1_dia'] = variables_temp['rain'].rolling(24, closed='left').max()
variables_temp['rain_roll_min_1_dia'] = variables_temp['rain'].rolling(24, closed='left').min()
variables_temp['rain_roll_max_7_dia'] = variables_temp['rain'].rolling(24*7, closed='left').max()
variables_temp['rain_roll_min_7_dia'] = variables_temp['rain'].rolling(24*7, closed='left').min()

variables_temp['radiation_roll_mean_1_dia'] = variables_temp['direct_radiation'].rolling(24, closed='left').mean()
variables_temp['radiation_roll_mean_7_dia'] = variables_temp['direct_radiation'].rolling(24*7, closed='left').mean()
variables_temp['radiation_roll_max_1_dia'] = variables_temp['direct_radiation'].rolling(24, closed='left').max()
variables_temp['radiation_roll_min_1_dia'] = variables_temp['direct_radiation'].rolling(24, closed='left').min()
variables_temp['radiation_roll_max_7_dia'] = variables_temp['direct_radiation'].rolling(24*7, closed='left').max()
variables_temp['radiation_roll_min_7_dia'] = variables_temp['direct_radiation'].rolling(24*7, closed='left').min()
```

Figura 2.19 Código insertar variables de temperatura, humedad y precipitación

Mediante la API de ***open-meteo*** extraemos los datos históricos de **temperatura**, **humedad relativa**, **temperatura aparente**, **precipitación de lluvia** y **radiación solar**. Estas variables las procesamos adicionalmente para crear los features de la variable **promedio móvil del último día**, **promedio móvil de los últimos 7 días**, el **valor máximo del último día**, el **valor mínimo del último día**.

```python
# Codificación cíclica de las variables de calendario y luz solar
# ==============================================================================
def codificacion_ciclica(datos: pd.Series, longitud_ciclo: int) -> pd.DataFrame:
    """
    Codifica una variable cíclica con dos nuevas variables: seno y coseno.
    Se asume que el valor mínimo de la variable es 0. El valor máximo de la
    variable se pasa como argumento.

    Parameters
    ----------
    datos : pd.Series
        Serie con la variable a codificar.
    longitud_ciclo : int
        La longitud del ciclo. Por ejemplo, 12 para meses, 24 para horas, etc.
        Este valor se utiliza para calcular el ángulo del seno y coseno.

    Returns
    -------
    resultado : pd.DataFrame
        Dataframe con las dos nuevas características: seno y coseno.

    """

    seno = np.sin(2 * np.pi * datos/longitud_ciclo)
    coseno = np.cos(2 * np.pi * datos/longitud_ciclo)
    resultado =  pd.DataFrame({
                    f"{datos.name}_seno": seno,
                    f"{datos.name}_coseno": coseno
                })

    return resultado

mes_encoded = codificacion_ciclica(variables_exogenas['mes'], longitud_ciclo=12)
semana_anyo_encoded = codificacion_ciclica(variables_exogenas['semana_anyo'], longitud_ciclo=52)
dia_semana_encoded = codificacion_ciclica(variables_exogenas['dia_semana'], longitud_ciclo=7)
hora_dia_encoded = codificacion_ciclica(variables_exogenas['hora_dia'], longitud_ciclo=24)
hora_amanecer_encoded = codificacion_ciclica(variables_exogenas['hora_amanecer'], longitud_ciclo=24)
hora_anochecer_encoded = codificacion_ciclica(variables_exogenas['hora_anochecer'], longitud_ciclo=24)

variables_ciclicas = pd.concat([
                        mes_encoded,
                        semana_anyo_encoded,
                        dia_semana_encoded,
                        hora_dia_encoded,
                        hora_amanecer_encoded,
                        hora_anochecer_encoded
                    ], axis=1)
variables_exogenas = pd.concat([variables_exogenas, variables_ciclicas], axis=1)
variables_exogenas.head(3)
```

Figura 2.20 Codificación cíclica de variables

Es buena práctica recomendada codificar variables que se repiten periódicamente como el mes, semana, día y hora. Para esto las codificamos como un componente seno y coseno con un período respectivo a su ciclo característico.

2.6.6.4 Interacción de polinomios de variables cíclicas

A menudo los modelos de aprendizaje de máquina se benefician de crear features mas complejas que son función de los inputs originales. Un ejemplo típico de esto es la creación de polinomios usando las variables cíclicas como base.

```
# Interacción entre variables exógenas
# ==============================================================================
transformer_poly = PolynomialFeatures(
                       degree           = 2,
                       interaction_only = True,
                       include_bias     = False
                   ).set_output(transform="pandas")
poly_cols = [
    'mes_seno',     'mes_coseno',
    'semana_anyo_seno',     'semana_anyo_coseno',
    'dia_semana_seno',     'dia_semana_coseno',
    'hora_dia_seno',     'hora_dia_coseno',
    'hora_amanecer_seno',     'hora_amanecer_coseno',
    'hora_anochecer_seno',     'hora_anochecer_coseno',
    'horas_luz_solar',     'es_de_dia',
    "temperature_2m",     "relative_humidity_2m",
    "apparent_temperature",     "rain",
    "direct_radiation",     "temp_roll_mean_1_dia",
    "temp_roll_mean_7_dia",     "temp_roll_max_1_dia",
    "temp_roll_min_1_dia",     "temp_roll_max_7_dia",
    "temp_roll_min_7_dia",     "hum_roll_mean_1_dia",
    "hum_roll_mean_7_dia",     "hum_roll_max_1_dia",
    "hum_roll_min_1_dia",     "hum_roll_max_7_dia",
    "hum_roll_min_7_dia",     "aptemp_roll_mean_1_dia",
    "aptemp_roll_mean_7_dia",     "aptemp_roll_max_1_dia",
    "aptemp_roll_min_1_dia",     "aptemp_roll_max_7_dia",
    "aptemp_roll_min_7_dia",     "rain_roll_mean_1_dia",
    "rain_roll_mean_7_dia",     "rain_roll_max_1_dia",
    "rain_roll_min_1_dia",     "rain_roll_max_7_dia",
    "rain_roll_min_7_dia",     "radiation_roll_mean_1_dia",
    "radiation_roll_mean_7_dia",     "radiation_roll_max_1_dia",
    "radiation_roll_min_1_dia",     "radiation_roll_max_7_dia",
    "radiation_roll_min_7_dia",     "Holiday",
    "holiday_dia_anterior",     "holiday_dia_siguiente"
]

variables_poly = transformer_poly.fit_transform(variables_exogenas[poly_cols].dropna())
variables_poly = variables_poly.drop(columns=poly_cols)
variables_poly.columns = [f"poly_{col}" for col in variables_poly.columns]
variables_poly.columns = variables_poly.columns.str.replace(" ", "__")
variables_exogenas = pd.concat([variables_exogenas, variables_poly], axis=1)
variables_exogenas.head(3)
```

Figura 2.21 Código interacción polinómica de variables

2.6.7 Creación del modelo con variables exógenas

Una vez reunidos los datos de entrenamiento de todas las variables exógenas, ya somos capaces de crear y entrenar el modelo usando LightGBM.

```
# Búsqueda bayesiana de hiperparámetros
# ==============================================================================
# params = {
#     'n_estimators': resultados_busqueda["params"].iloc[0]["n_estimators"],
#     'max_depth': resultados_busqueda["params"].iloc[0]["max_depth"],
#     'learning_rate': resultados_busqueda["params"].iloc[0]["learning_rate"],
#     'reg_alpha': resultados_busqueda["params"].iloc[0]["reg_alpha"],
#     'reg_lambda': resultados_busqueda["params"].iloc[0]["reg_lambda"],
#     'random_state': 15926,
#     'verbose': -1
# }
forecaster = ForecasterAutoreg(
                 regressor = LGBMRegressor(random_state=15926, verbose=-1),
                 lags      =24,
                 weight_func = custom_weights
             )

lags_grid = [25,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
                28,29,30,31,32,
                34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,
                52,54,57,59,60,61,62,63,64,65,66,67,68,69]]

# Espacio de búsqueda de hiperparámetros
def search_space(trial):
    search_space  = {
        'n_estimators'  : trial.suggest_int('n_estimators', 600, 1200, step=100),
        'max_depth'     : trial.suggest_int('max_depth', 3, 10, step=1),
        'learning_rate' : trial.suggest_float('learning_rate', 0.01, 0.5),
        'reg_alpha'     : trial.suggest_float('reg_alpha', 0, 1, step=0.1),
        'reg_lambda'    : trial.suggest_float('reg_lambda', 0, 1, step=0.1),
        'lags'          : trial.suggest_categorical('lags', lags_grid)
    }
    return search_space

resultados_busquedaExog, frozen_trial = bayesian_search_forecaster(
                                            forecaster         = forecaster,
                                            y                  = datos.loc[:fin_validacion, 'kVA sd del'],
                                            exog               = datos.loc[:fin_validacion, exog_features],
                                            steps              = 24,
                                            metric             = 'mean_absolute_error',
                                            search_space       = search_space,
                                            initial_train_size = len(datos[:fin_train]),
                                            refit              = False,
                                            n_trials           = 20, # Aumentar para una búsqueda más exhaustiva
                                            random_state       = 123,
                                            return_best        = True,
                                            n_jobs             = 'auto',
                                            verbose            = False,
                                            show_progress      = True
                                        )
```

Figura 2.22 Creación del modelo

Como se observa en la figura superior primero creamos el modelo usando los parámetros usados por default. Luego, procedemos a crear un espacio de hiperparámetros en el cual

el optimizador bayesiano probará diferentes combinaciones y encontrará el mejor conjunto de hiperparámetros.

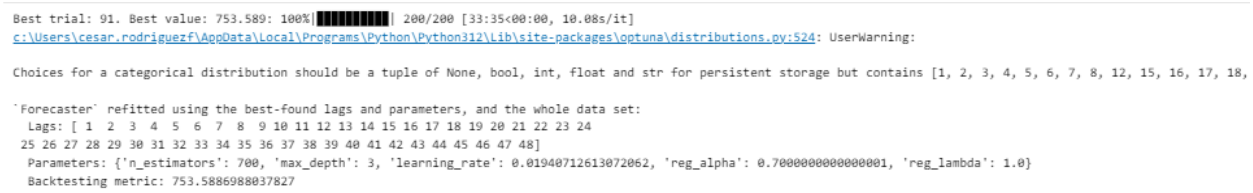

Figura 2.23 Resultado de optimización de hiperparámetros

La figura superior muestra el mejor conjunto de hiperparámetros para el modelo, junto con el error promedio de 753 KVA, el cual es el error hallado en el dataset de validación. Encontramos entonces, que el modelo óptimo para la predicción de la demanda de potencia del trasformador 2 de la subestación Ceibos + el reconectador R405L tiene los hiperparámetros:

- Número de estimadores: 1100 árboles
- Profundidad máxima de cada árbol: 4 niveles
- Razón de aprendizaje: 0.023980503484977468
- Regularización Alpha: 0.6000000000000001
- Regularización lambda: 0.1

Es importante establecer que esto hiperparámetros son los óptimos en las circunstancias en las que se entrenó este modelo, estos cambiarán conforme el modelo se continué reentrenando con nuevos datos.

En este capítulo hemos presentado la creación y entrenamiento del modelo para una condición (la transferencia del reconectador R405 al transformador Ceibos 2). De ocurrir otras condiciones, otras transferencias podrían necesitarse, lo que implicará la creación y entrenamiento de otro modelo.

Una vez entrenado el modelo, estos se guardarán en disco, desde donde se usarán para predecir rápidamente de llamarse el servicio por el RTU en la subestación. Los modelos se reentrenarán cada 72 horas para mantenerse aptos predecir adecuadamente.

2.7 Elaboración de pantallas SCADA

El automatismo programado en las RTU recibe los datos de predicción del servidor de soporte, pero se controla y supervisa desde las pantallas del sistema SCADA de la distribuidora.

Para esto se han desarrollado dos pantallas, una para cada automatismo

2.7.1 Pantalla SCADA automatismo cerro del Carmen

Figura 2.24 Pantalla SCADA automatismo cerro del carmen

Como se puede observar en la figura superior, la pantalla SCADA para supervisar el automatismo muestra el diagrama unifilar del circuito junto con el estado de los reconectadores y los valores instantáneos de corriente en los mismos.

Podemos encontrar también las condiciones lógicas para activar los automatismos y si se han cumplido o no. Finalmente encontramos un botón que se activa al dispararse el automatismo y permite al operador ejecutar la secuencia. En el caso del automatismo c. del Carmen, la transferencia puede ocurrir en ambas direcciones por lo que vemos dos conjuntos de condiciones y dos botones de accionamiento.

2.7.2 Pantalla SCADA automatismo cerro del Carmen

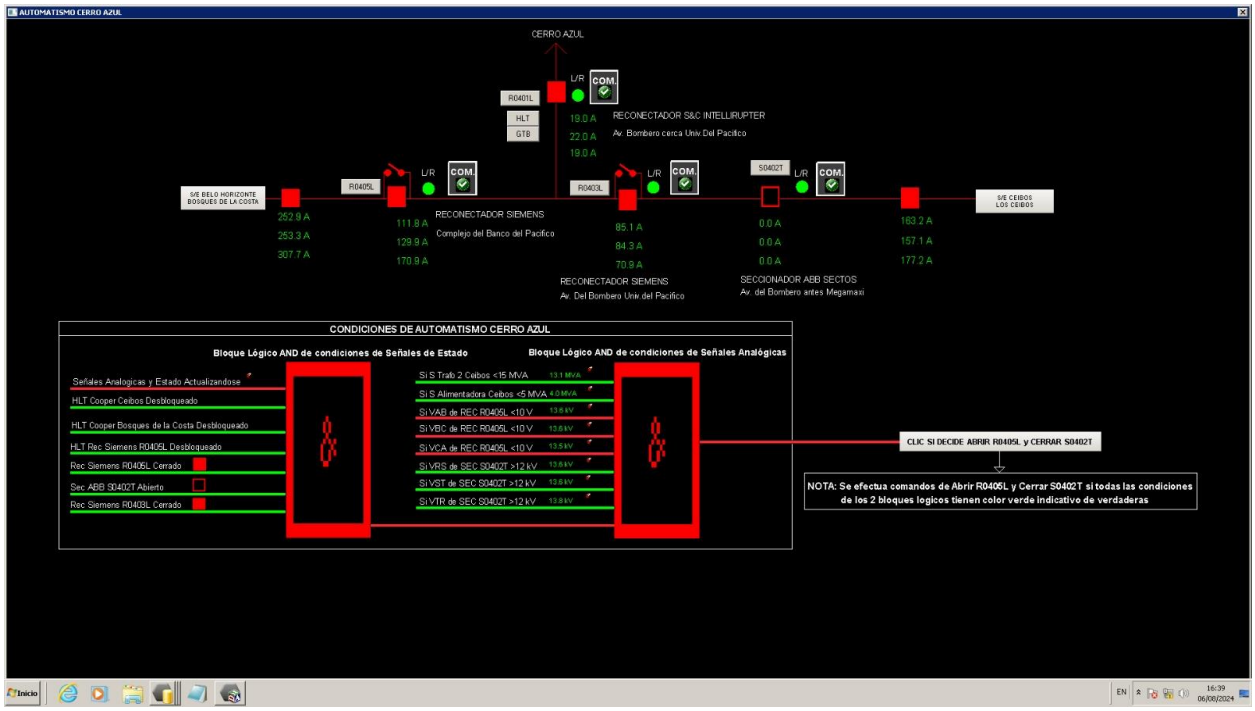

Figura 2.25 Pantalla SCADA automatismo cerro azul

Tal como la pantalla de c. del Carmen, esta pantalla muestra el estado de los reconectadores en el diagrama unifilar del circuito, las condiciones lógicas y un botón permitiendo el accionamiento del circuito de cumplirse las condiciones lógicas del mismo.

CAPÍTULO 3

3. RESULTADOS Y ANÁLISIS

3.1 Prueba final del modelo entrenado

Una vez entrenado el modelo se realizó un backtesting del modelo en el dataset de prueba, hasta ahora no usado y reservado para evitar contaminación. Para esto seguimos usando la librería skforecast que nos permite realizar este proceso de forma simple.

```
# Backtesting configuracion tuneada con modelo exógeno
# ==============================================================================
metrica, predicciones = backtesting_forecaster(
                            forecaster         = forecaster,
                            y                  = datos['kVA sd del'][:"2024-04-01 00:00:00"],
                            exog               = datos[exog_features][:"2024-04-01 00:00:00"],
                            steps              = 24,
                            metric             = 'mean_absolute_error',
                            initial_train_size = len(datos[:fin_validacion]),
                            refit              = False,
                            n_jobs             = 'auto',
                            verbose            = False,
                            show_progress      = True
                        )

print(f"Error de backtest (MAE): {metrica:.2f}")
predicciones.head()
```

Figura 3.1 Código de backtesting del modelo

```
100%|██████████| 91/91 [00:01<00:00, 57.64it/s]
Error de backtest (MAE): 651.38
```

Figura 3.2 Resultados del backtesting

Como se puede observar en la figura superior, el el modelo final en el set de prueba tiene un error promedio de **651.38 KVA.**

Respecto al error porcentual del modelo, corremos el siguiente código para encontrarlo:

```python
error_promedio_modelo = (np.abs( 1.0*predicciones['pred'] - datos_test['kVA sd del'])/datos_test['kVA sd del']).mean()*100
error_std_modelo = (np.abs( 1.0*predicciones['pred'] - datos_test['kVA sd del'])/datos_test['kVA sd del']).std()*100
max_error_modelo = (np.abs( 1.0*predicciones['pred'] - datos_test['kVA sd del'])/datos_test['kVA sd del']).max()*100
min_error_modelo = (np.abs( 1.0*predicciones['pred'] - datos_test['kVA sd del'])/datos_test['kVA sd del']).min()*100

print(f"Promedio error porcentual del modelo es: {error_promedio_modelo:.2f}%")
print(f"std del error porcentual del modelo es: {error_std_modelo:.2f}%")
```

Figura 3.3 Código error porcentual del modelo

```
Promedio error porcentual del modelo es: 5.25%
std del error porcentual del modelo es: 5.78%
```

Figura 3.4 Error porcentual del modelo

Como se puede observar en la imagen superior el error porcentual promedio del modelo al correr el backtesting en el set de prueba es de **5.25%** con una desviación estándar de **5.78%**, estos son valores bastante aceptables que sugieren que el modelo funcionará adecuadamente para aplicación deseada.

3.2 Comparación datos predichos con datos reales

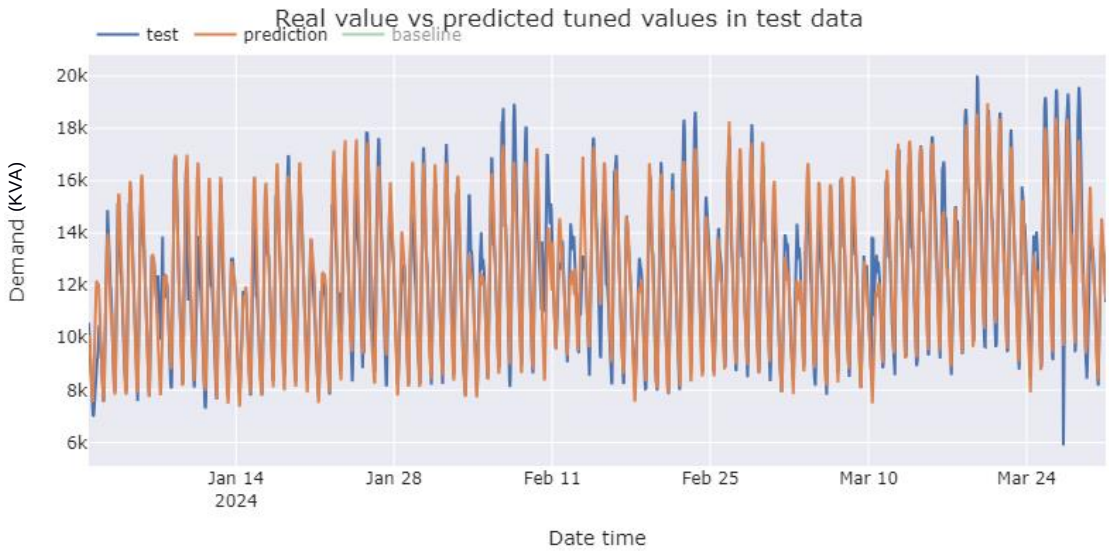

figura 3.5 Comparación datos predichos (naranja) con datos reales (azul)

En la figura superior, observamos la comparación de datos reales (azul) con los datos predichos (naranja), como se puede observar en la f igura inferior, la predicción es

bastante acertada y coincide con el error descrito de 5.25% de la sección anterior. Es importante mencionar que a pesar de que el error es relativamente bajo, los días donde las discrepancias parecen aumentar corresponden a días atípicos (curva distinta). Esto sugiere que la discrepancia ocurre debido a variables de entrada no modeladas. Dado que solo se han utilizado como entradas predictoras los datos históricos de la serie de tiempo y variables meteorológicas, es posible que existan mas variables de interés que no han sido tomadas en cuenta que expliquen el comportamiento de la demanda en estos momentos atípicos.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Se ha elaborado un sistema para trasferir automáticamente la carga de alimentadoras en la red eléctrica. Esto permite una rápida respuesta de una compañía distribuidora de electricidad a emergencias ocurridas y el mantenimiento de la confiabilidad del sistema eléctrico.

El sistema se creó mediante automatismos programados en RTU´s locales en cada subestación, conectadas a reconectadores de campo, un sistema SCADA y un servidor de soporte que corre un modelo predictivo basado en árboles de decisión con *gradient boosting*. Este modelo se reentrenará cada 72 horas con datos que se capturan de forma continua del sistema SCADA a una base de datos PostgreSQL.

El error encontrado para la predicción de la transferencia mostrada en este proyecto fue de 5.25% lo cual es mas que suficiente para poder predecir adecuadamente las transferencias a realizar.

4.2 Limitaciones y recomendaciones

Es importante mencionar que los automatismos descritos en este documento son aplicables para circuitos con una sola transferencia posible al mismo tiempo. En redes mas complejas es posible transferir a varias alimentadoras al mismo tiempo. En estos casos flujos de potencia deben ser elaborados para conocer que proporción del flujo inicial será transferido a cada alimentador. Esto se logra típicamente con estudias de carga previamente elaborados de los cuales se ha determinado los peores casos posibles.

Adicional a esto, el modelo predictivo elaborado en este documento utiliza para predecir los datos históricos de la demanda, variables de fecha y variables meteorológicas. Existen otros parámetros de interés que también tienen poder predictivo en estudias de demanda, tales como indicadores económicos locales y

de cantidad de población. Dado que esta información no es posible obtenerla en tiempo real de los instrumentos de la red eléctrica, no se ha tomado en cuenta a la hora de realizar el modelo. Sin embargo, para trabajos futuros de necesitarse una mayor precisión, esta es una ruta que se podría tomar para mejorar el modelo.

BIBLIOGRAFÍA

[1] A. Devanand, «OntoPowSys: A power System ontology for cross domain interactions in an eco industrial park,» *Energy and AI vol 1,* p. 100008, Agosto 2020.

[2] A. Pansini, Electrical Distribution Engineering, Boca Raton, 2006.

[3] N. A. Reyes, «"Modbus TCP Bridging for Interconnecting Non-Compatible Devices in the Energy Sector Using Node-RED and Edge Computing,» *2023 IEEE 41st Central America and Panama Convention (CONCAPAN XLI),* pp. 1-4, 2023.

[4] B. Auffarth, Machine Learning for Time-Series with Python: Forecast, predict, and detect anomalies with state-of-the-art machine learning methods., Packt, 2021.

[5] J. Smith, «Automation of Distribution Networks: Technologies and Protocols,» *IEEE Transactions on Power Delivery,* vol. 35, nº 4, pp. 2300-2309, 2020.

[6] L. Zhang, «An Overview of Distribution Automation Systems,» *IEEE Power & Energy Magazine,* vol. 18, nº 1, pp. 56-64, 2022.

[7] M. Johnson, «Communication Protocols for Smart Grids: A Comparative Analysis,» *IEEE Internet of Things Journal,,* vol. 7, nº 12, pp. 12010-12020, 2021.

[8] Microsoft, «LightGBM Documentation,» 4 07 2024. [En línea]. Available: https://lightgbm.readthedocs.io/en/stable/.

[9] J. A. Rodrigo, «Predicción (forecasting) de la demanda energética con machine learning,» 2023. [En línea]. Available: https://www.cienciadedatos.net/documentos/py29-forecasting-demanda-energia-electrica-python.html.

[10] M. Bruch, M. Munch y M. Kuhn, «Power blackout risks,» de *CRO Forum*, 2011.