Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Técnicas de Aprendizaje Automático aplicadas a la predicción de afluencia de pacientes a unidades de atención primaria

Autor: Jorge García González

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

Técnicas de Aprendizaje Automático aplicadas a la predicción de afluencia de pacientes a unidades de atención primaria

Autor: Jorge García González

Tutores: Valentín Cardeñoso Payo
Pablo Gutiérrez Díez (SERBATIC)

Dedicado a todos los que habéis acabado en esta página...

Ahora no puedo entrenar un modelo sin imaginarme un tren ...

Agradecimientos

Me gustaría agradecer en este espacio a todas las personas que han hecho posible que a día de hoy pueda haber terminando este trabajo y con ello mis estudios en InDat.

Gracias a los profesores que me han ido guiando desde mi etapa en el colegio Nuestra Señora del Carmen hasta el último curso del grado universitario aquí en la UVA, muy especialmente a mi tutor Valentín Cardeñoso, quien me ha aconsejado y facilitado el desarrollo de este documento enormemente.

Gracias, sobre todo, a mis amigos, mi tíos, mis abuelos, mis padres y mi hermano que han confiado siempre en mí, que me dan fuerzas cada día y por quienes he llegado hasta aquí.

No me olvido tampoco de mis compañeros de oficina en Serbatic que me hicieron más amenas muchas mañanas de trabajo ni de mi tutor de empresa Pablo Gutierrez Díez que me proporcionó todas las herramientas para hacer este trabajo y que ha desafiado para bien cualquier expectativa que pudiera tener sobre el ambiente en el mundo laboral.

Para todos ellos y para todos los que tengáis la curiosidad de leer este trabajo, os mando mi abrazo más sincero.

Resumen

La atención sanitaria en Castilla y León tiene como unidad mínima de atención primaria miles de consultorios repartidos por su geografía que reciben pacientes durante todos los días del año para tratar sus necesidades médicas. La organización y planificación de un conjunto tan grande de espacios sanitarios lleva consigo una gran dificultad asociada a la decisión de qué recursos asignar a cada unidad así como cuándo y dónde hacerlo.

El objetivo de este trabajo consiste en encontrar junto a la empresa Serbatic un modelo que se pueda emplear como guía en el proceso. Más concretamente la meta final es conseguir predecir con un grado de confianza aceptable la afluencia a cada uno de estos consultorios en el futuro próximo con todas las ventajas que esto supondría para la administración de dichos centros.

Hasta lograrlo en este documento se describirá todo el proceso paso a paso desde la planificación del proyecto hasta el testeo de los modelos pasando por las justificaciones teóricas de su elección y desarrollo así como por los detalles de programación que permitan que estos modelos matemáticos se trasladen a código útil.

Abstract

The health care system in Castilla y León is organized into thousands of primary care offices which treat patients every day of the year. Therefore,it is challenging to decide what resources to assign to every unit as well as when and where to do so in such a large set of healthcare facilities.

Ultimately, the objective of this work is to develop a model that can be used as a guide by those responsible for making these decisions, allowing them to predict the influx of patients to each clinic in the near future with a high degree of confidence. Without a doubt, all of this would be beneficial to the administration of these centers.

To achieve this goal, this document will provide a step-by-step description from project planning to model testing, covering both the theoretical justifications for their choice and development, as well as the programming details that enable these mathematical models to function as useful code.

Contents

List of Tables

List of Figures

Part I

Objeto, Concepto y Método

Chapter 1

Introducción

La gestión de recursos, tanto empleados como materiales, es una actividad de capital importancia en toda organización. Estos bienes son limitados y el objetivo de una administración eficaz es asignarlos de forma que se maximicen los beneficios pues las formas de distribuirlos pueden ser ser casi infinitas.

Esta tarea tan abstracta puede aumentar en complejidad enormemente cuando el número de factores que intervienen en ella es elevado y es por ello por lo que herramientas que permitan hacer estimaciones pueden suponer una gran ventaja a la hora de tomar estas decisiones de gestión. Actualmente una de las mejores opciones para hacer predicciones consiste en aplicar técnicas de *Machine Learning* al histórico de datos del que se disponga tratando de minimizar el error cometido en un conjunto de prueba. Sin embargo existen muchas variantes y múltiples modelos candidatos por lo que surge una importante tarea de exploración e investigación de modelos a realizar para poder obtener los resultados deseados.

En el caso concreto que se explora en este documento se contempla la gestión de los recursos sanitarios en la comunidad autónoma española de Castilla y León de la cual se hacen cargo tanto la Consejería de Sanidad como las distintas Gerencias de las Áreas de Salud. Una de las responsabilidades de estos organismos consiste en la elaboración de presupuestos y la asignación de médicos y material sanitario a las distintas consultorios Zonas Básicas de Salud de las que se divide cada Área, de tal forma que, como se ha mencionado antes, estas entidades se enfrentan decisiones de asignación dependientes de multitud de variables y lo hacen con un enfoque heurístico en el que se toman las decisiones basándose en qué funcionó otros años y en lo que ”suele” ocurrir.

Esta forma de actuar, aunque pueda resultar suficiente, posiblemente diste mucho de una metodología que permita maximizar la utilidad de los recursos de los que se disponen. Es muy probable que en ciertos consultorios haya más profesionales de los que son necesarios en cierta época del año o que por el contrario los recursos asignados a otro centro sean insuficientes y que la situación derive en la sobrecarga de trabajo para otros trabajadores. Seguramente la diferencia no sea suficiente como para provocar un deterioro en la asistencia médica tal que la salud de los pacientes se vea afectada en la mayoría de los casos, pero sin duda una mejor distribución de los recursos sería capaz de reducir la carga de trabajo de muchos trabajadores que desempeñan más tareas de las que deberían soportar.

Por lo tanto la empresa Serbatic quiere desarrollar una herramienta basada en *Machine Learning* que pueda integrarse en una web orientada al servicio de personal de las Gerencias de Atención Primaria y/o los propios coordinadores de áreas básicas de Salud.

Chapter 2

Objetivos y Alcance

2.1 Objetivos

Teniendo en cuenta el problema descrito es fácil deducir el principal objetivo que surge al enfrentarse a esta situación. El foco de este documento será determinar cuál es el modelo de predicción más útil y que permita mejorar la distribución de recursos entre los consultorios de Castilla y León.

Este objetivo sin embargo es algo abstracto ya que plantea ciertas dudas ¿Cómo se va a organizar el desarrollo?¿Qué datos previos deberían utilizarse y cómo deberían tratarse? ¿Qué se debería predecir exactamente? ¿Cómo se podría determinar que la predicción es útil? ...

Estas preguntas sugieren pues una serie de metas más concretas que perfilan el contenido final del proyecto y las tareas que van a componerlo.

Como objetivos finales que sería deseable alcanzar podemos enumerar los siguientes:

- El modelo debe llevar a cabo la predicción del número de pacientes de atención primaria que acuden a cierta zona básica de salud en un día concreto desde el día actual hasta un periodo razonable que se puede fijar inicialmente como 5 meses.
  Como cada zona básica de salud cuenta con su propio registro y comportamiento (aunque con similitudes entre ellos) parece bastante razonable la necesidad de entrenar un modelo para cada consultorio.
- Para que la predicción sea útil es imprescindible que el error de ésta sea aceptable. Sin haber inspeccionado los datos es aventurado establecer una cifra límite exacta pero lo que está claro es que se deberá probar distintos modelos y distintos hiperparámetros de estos para seleccionar las combinaciones que mejor funcionen en cada caso.

2.1.1 Tareas a realizar

Para lograr los objetivos mencionados hay muchas pequeñas metas que ir alcanzando progresivamente:

1. Establecer todos los objetivos concretos del proyecto.

2. Realizar una descripción de la metodología y una estimación de su coste en tiempo y recursos.

3. . Elegir un entorno de desarrollo, lenguaje y librerías con los que se va a codificar los modelos .

4. Hacer una exploración de los datos abiertos de la Junta de Castilla y León y seleccionar los más informativos en el contexto de la afluencia de pacientes de atención primaria.

5. Realizar un análisis exploratorio de los datos que de una idea de la distribución de los pacientes en los consultorios a lo largo del año y que revele la existencia de posibles *outliers*, anomalías o dependencias a tener en cuenta.

6. Investigar tanto el estado actual de los métodos de predicción de series temporales como los fundamentos teóricos de las distintas soluciones y justificar la elección de los modelos que se van a probar.

7. Desarrollar el código de los modelos elegidos y buscar una forma eficaz de compararlos para seleccionar el modelo y los hiperparámetros que conduzcan al menor error.

8. Contenedorizar los distintos scripts y dependencias para garantizar que el código pueda ejecutarse en cualquier máquina

9. Evaluar el grado de éxito del trabajo desarrollado y exponer tanto las conclusiones obtenidas en el proceso como posibles mejoras futuras que podrían añadirse al proyecto.

Chapter 3

Metodología

Todo trabajo requiere un método y una planificación de qué recursos van a invertirse y cuándo van a invertirse así como una especificación de las herramientas que van a emplearse en el desarrollo

3.1 Fases y costes

El desarrollo de la aplicación y de este documento se ha llevado a cabo a lo largo de distintas etapas estimadas de forma previa. Las fases y duración prevista de cada una (Semana inicial y final) son las que se detallan en la tabla 3.1.

Nombre de actividad	**Semanas**
Formación previa en la empresa	1 - 8
Inicio del documento, definición de objtivos y metodología	9-10
Búsqueda, exploración y depuración de datos	10-11
Investigación de modelos teóricos para predicción de series temporales	12-13
Desarrollo del código de los modelos	13-17
Testeo	18
Depurar la memoria del TFG	19

Table 3.1: Fases de desarrollo del proyecto previstas.

En términos planificación temporal, sin tener en cuenta las 8 semanas de formación previa en la empresa, el proyecto se extendería a lo largo de 11 semanas.

3.2 Tecnologías

Según el proyecto ha avanzado se han ido añadiendo a esta lista distintas tecnologías y herramientas que han cumplido una función específica. A continuación se detallan todas ellas:

3.2.1 Lenguajes de programación:

El lenguaje de programación empleado ha sido Python 3.12.2 y su elección está justificada por el hecho de que es el principal lenguaje empleado mundialmente en el tratamiento de datos y la implementación de técnicas de *Machine Learning* [4]

Pero sin duda alguna la mayor ventaja de elegir Python es la existencia de múltiples librerías increiblemente útiles y eficientes a la hora de realizar los cálculos necesarios en el proceso de limpieza de datos y entrenamiento de algoritmos

3.2.2 Librerías

- Numpy: Biblioteca esencial en Python para el cálculo numérico y científico, ofreciendo estructuras de datos eficientes como arrays multidimensionales y funciones matemáticas de alto rendimiento. Su eficiencia y funcionalidad son fundamentales para el desarrollo de proyectos de análisis de datos y modelado predictivo y tiene una gran integración con el resto de librerías empleadas.

- Pandas: Potente biblioteca de Python para manipulación y análisis de datos estructurados, ofreciendo estructuras de datos flexibles como DataFrames y Series. Esta librería proporciona herramientas para cargar, limpiar, transformar y analizar datos de manera eficiente. Con su capacidad para manejar fechas y horas, filtrar datos, y realizar operaciones de agregación y agrupamiento, Pandas es fundamental para explorar y preparar los datos antes de aplicar técnicas de modelado predictivo.
- Matplotlib, Pyplot y Seaborn: Bibliotecas de trazado en Python ampliamente utilizadas para crear visualizaciones de datos de alta calidad, incluyendo gráficos de líneas, barras, dispersión, histogramas y más. Son útiles para representar los datos de series temporales que van a intentar predecirse así como las propias predicciones o métricas de distinto tipo. Esto facilita enormemente la interpretación de los datos y la evaluación de modelos.

- XGBoost: Biblioteca de aprendizaje automático optimizada para árboles de decisión, conocida por su eficiencia, escalabilidad y precisión en una variedad de problemas de regresión y clasificación permitiendo la aplicación del conocido como Extreme Gradient Boosting. Es particularmente útil en proyectos de predicción de series temporales debido a su capacidad para manejar conjuntos de datos grandes, la flexibilidad para modelar relaciones no lineales y su capacidad para manejar características de series temporales, como las tendencias y las estacionalidades

- Pytorch: Biblioteca de aprendizaje profundo de código abierto que proporciona una plataforma flexible y dinámica para construir y entrenar modelos de redes neuronales. Es especialmente útil en proyectos de predicción de series temporales debido a su capacidad para trabajar con datos secuenciales y manejar modelos complejos, como redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN). Además, PyTorch ofrece una interfaz muy intuitiva y una amplia gama de herramientas para la experimentación rápida y la investigación en el campo del aprendizaje profundo.

- Prophet: Prophet es una biblioteca de Python desarrollada por Facebook (Meta) para realizar pronósticos de series temporales con patrones estacionales y efectos de días festivos. Utiliza un modelo aditivo con componentes de crecimiento lineal o logístico, estacionalidad anual, semanal y diaria, y efectos de días festivos. Es fácil de usar, robusto con datos faltantes, y permite una

amplia personalización, lo que lo hace adecuado para aplicaciones como pronósticos de ventas y planificación de recursos. Prophet trabaja bien con DataFrames de pandas y proporciona intervalos de confianza para sus predicciones, ofreciendo interpretabilidad y configurabilidad en sus modelos .

3.2.3 Entornos de desarrollo (IDE)

- Pycharm: PyCharm es un entorno de desarrollo integrado (IDE) diseñado específicamente para programadores de Python, ofreciendo características como depuración, completado automático de código, refactoring, soporte para pruebas unitarias y control de versiones integrado. Su entorno especializado y funcionalidades hacen que sea una opción eficiente para el desarrollo de proyectos de análisis de datos en Python. Este IDE será el que se emplee en el desarrollo del código de la aplicación final

- Jupyter Notebook: Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos interactivos que contienen código, visualizaciones y texto explicativo. Es especialmente útil ya que permite escribir y ejecutar código Python en bloques individuales, facilitando la experimentación y el análisis paso a paso. Además, ofrece la capacidad de integrar visualizaciones de datos directamente en el flujo de trabajo, lo que ayuda a comprender y comunicar los resultados de manera efectiva. Por ello este IDE se empleará en el proceso de análisis y exploración de datos.

3.2.4 Otras tecnologías

- Mlflow: Plataforma de código abierto para el ciclo de vida del aprendizaje automático que ayuda a gestionar proyectos, experimentos, modelos y flujos de trabajo de producción. Proporciona herramientas para realizar un seguimiento de los experimentos de aprendizaje automático, registrar y comparar métricas de rendimiento, almacenar y versionar modelos, y colaborar con otros miembros del equipo. En el contexto específico de este trabajo será de gran importancia a la hora de comparar, tanto con métricas como con gráficos, qué modelo y qué hiperparámetros son los adecuados para cada modelo desarrollado ya que es muy posible que no el mismo modelo sea el mejor para cada serie temporal a predecir.

- Docker: Plataforma de código abierto que permite empaquetar, distribuir y ejecutar aplicaciones en contenedores. Los contenedores de Docker encapsulan todo lo necesario para que una aplicación se ejecute de manera independiente, incluidas las bibliotecas, las dependencias y el código, garantizando la portabilidad y consistencia del entorno de ejecución. En este proyecto Docker será útil para crear un entorno de desarrollo y producción reproducible y aislado que permita el despliegue en diferentes plataformas sin preocuparse por las diferencias en los sistemas operativos o las configuraciones del entorno

- GitLab: GitLab es una plataforma de gestión del ciclo de vida del desarrollo de software que proporciona herramientas para la colaboración, el seguimiento de problemas, la integración continua y la implementación continua. Permite a los equipos de desarrollo trabajar de manera colaborativa en proyectos de software y gestionar el código fuente utilizando el sistema de control de versiones Git por lo que es una herramienta ideal para compartir el código con los tutores de este proyecto para facilitar el seguimiento durante el desarrollo.

Part II

Contexto

Chapter 4

Marco conceptual

Como ya se ha mencionado en la introducción, el contexto en el que se va a trabajar es el de la sanidad de la comunidad autónoma española de Castilla y León. Para poder entender la estructura de los organismos que gestionan la atención sanitaria en la comunidad es necesario describir los distintos niveles que la componen y que se presentan en la siguiente sección.

4.1 Niveles de estructura sanitaria

1. Consultorios Locales:
   - Los consultorios locales son las unidades básicas de atención sanitaria en las zonas rurales y urbanas. Están destinados a ofrecer servicios de salud básicos a los habitantes de pequeñas localidades y barrios.
   - Estos consultorios suelen estar atendidos por médicos de familia y enfermeros.
2. Centros de Salud:
   - Los centros de salud son unidades más grandes y complejas que los consultorios locales. Ofrecen una gama más amplia de servicios de atención primaria, incluyendo medicina general, pediatría, enfermería, atención a la mujer, salud bucodental, y trabajo social.
   - Además, en los centros de salud se pueden encontrar otros servicios como fisioterapia y atención a domicilio.
3. Zonas Básicas de Salud (ZBS):
   - Las Zonas Básicas de Salud agrupan varios centros de salud y consultorios locales para facilitar la gestión y coordinación de los servicios sanitarios en un área geográfica determinada.
   - Cada ZBS está dirigida por un equipo de atención primaria que coordina y supervisa las actividades de los profesionales sanitarios de la zona.
4. Gerencias de Atención Primaria:
   - Las Gerencias de Atención Primaria se encargan de la administración y gestión de las Zonas Básicas de Salud. Supervisan la calidad de los servicios, la asignación de recursos, y la implementación de políticas de salud.
   - En Castilla y León, hay varias gerencias distribuidas en función de las provincias.

5. Áreas de Salud:
    - Castilla y León se divide en varias Áreas de Salud, que son divisiones administrativas que engloban a las distintas Zonas Básicas de Salud y los servicios de atención especializada.
    - Cada Área de Salud tiene un hospital de referencia que proporciona servicios de atención especializada a los pacientes derivados desde la atención primaria.

Es entonces necesario recordar que el objetivo principal del proyecto es elaborar un modelo predictivo que puedan utilizar tanto las Gerencias de Atención Primaria como los coordinadores de las Áreas de Salud para poder estimar la cantidad de pacientes que van a acudir a las distintas Zonas Básicas de Salud repartidas por la geografía castellanoleonesa. Hasta el momento y tras consultar a varios profesionales de la zona, los administradores de estas Zonas de Salud asignan el número de profesionales ”a ojo”, es decir, sin una guía o metodología que les ayude y que conlleva en muchos casos una cierta improvisación. La elección de la Zona Básica de Salud como nivel al que realizar la predicción en lugar de hacerlo a nivel de consultorio se basa en dos razones de peso.

La primera es que existen muchos consultorios (3.628) lo cual tiene dos consecuencias muy indeseables. Una de ellas es el elevado número de modelos que deberían entrenarse y que supondrían un tiempo y esfuerzo de cómputo inasumibles. La otra es que existen consultorios tan poco concurridos que apenas cuentan con registros y que llevarían a predicciones ridículas y nada útiles.

La segunda razón para preferir el nivel de Zona Básica de Salud se trata de la flexibilidad con la que cuentan los profesionales para moverse entre consultorios en una misma zona. En muchas ocasiones los médicos asignados a cierto consultorio no pasan toda la semana en el mismo e incluso existen ”médicos de área” que acuden allí donde los necesitan. Por todo ello parece mucho más razonable enfocar el problema a este nivel aunque se pierda algo de concreción en favor de predicciones más útiles y viables.

Es entonces lógico decir que la información que es necesario obtener para entrenar los modelos que realicen la predicción debe estar relacionada con la afluencia a dichas Zonas Básicas de Salud a lo largo de los últimos años, es decir, la cantidad de pacientes de atención primaria para cada consultorio. Para concretar más los profesionales que trabajan en estos consultorios son tanto médicos de familia como pediatras y enfermeros de atención primaria.

4.2 Conjuntos de datos

Para buscar los datos mencionados se ha recurrido al Portal de Datos Abiertos de la Junta de Castilla y León [3] y se han identificado distintas fuentes que pueden ser útiles en la predicción.

La primera fuente se trata de un dataset con información sobre la *Actividad de medicina de familia a nivel de consultorio* [2]

Otra fuente que se ha considerado adecuado y que recoge otra parte de la actividad de atención primaria es el conjunto de datos es ’ Actividad de enfermería a nivel de consultorio’

Como puede observarse estos datasets cuentan con más de 1.200.000 (Medicina de familia) y 1.120.000 (Enfermería) registros obtenidos desde el 1 de enero de 2020 y múltiples campos que tienen idéntico significado en ambos conjuntos de datos:

- Fecha: Día, mes y año en el que se ha producido el registro
- Código Consultorio: Código único asignado al consultorio del cual se ha registrado afluencia

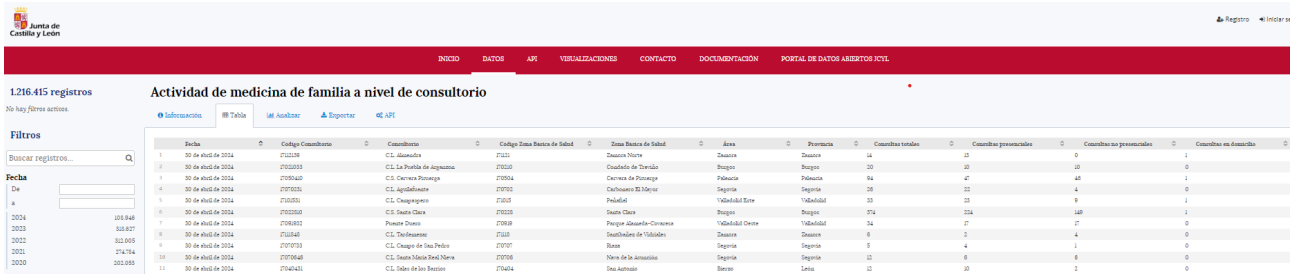


Figure 4.1: Dataset de actividad de medicina de familia a nivel de consultorio

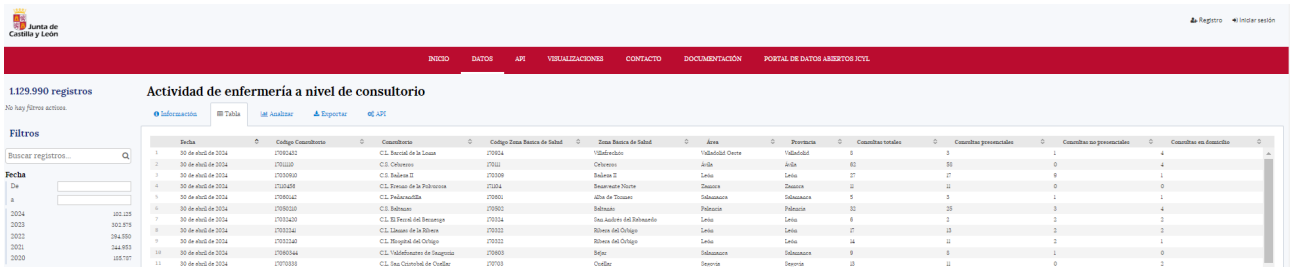


Figure 4.2: Dataset de actividad de enfermería a nivel de consultorio

- Consultorio: Nombre completo del consultorio del cual se ha registrado afluencia
- Código Zona Básica de Salud: Código único de la Zona Básica de Salud a la que pertenecen los pacientes que se han registrado
- Zona Básica de Salud: Nombre completo de la Zona Básica de Salud a la que pertenecen los pacientes que se han registrado
- Área: Nombre del Área de Salud al que pertenece el consultorio del cual se ha registrado afluencia
- Provincia: Nombre de la provincia a la que pertenece el consultorio del cual se ha registrado afluencia
- Consultas totales: Número de consultas totales de medicina de familia o enfermería (dependiendo el conjunto de datos) que se han producido en cierto consultorio para pacientes de cierta Zona Básica de Salud en cierto día
- Consultas presenciales: Número de consultas presenciales de medicina de familia o enfermería (dependiendo el conjunto de datos) que se han producido en cierto consultorio para pacientes de cierta Zona Básica de Salud en cierto día
- Consultas no presenciales: Número de consultas no presenciales de medicina de familia o enfermería (dependiendo el conjunto de datos) que se han producido en cierto consultorio para pacientes de cierta Zona Básica de Salud en cierto día
- Consultas en domicilio: Número de consultas a domicilio de medicina de familia o enfermería (dependiendo el conjunto de datos) que se han producido en cierto consultorio para pacientes de cierta Zona Básica de Salud en cierto día

Como peculiaridad destacable en estos datasets se podría señalar la presencia de múltiples instancias para un mismo día en ciertos consultorios. Esto se debe a que es posible que haya pacientes que pertenezcan a Zonas de Salud distintas de aquella a la que pertenece el consultorio de interés que quedan registrados en una entrada distinta en el conjunto de datos. Posteriormente se abordará el tratamiento de esta situación.

Chapter 5

Análisis y depuración previa

Una vez identificados y descritos los datos que se van a utilizar es importante tratarlos para darles una forma adecuada que refleje la información que nos interesa y permita al programa procesarla correctamente. Para determinar qué transformaciones son necesarias, parece conveniente crear un Jupyter Notebook ('Data_Preprocess_And_Analysis_TFG.ipnyb') con el que acceder a los datos y representarlos gráficamente.

5.1 Carga de datos

En primer lugar es necesario acceder a la API del Portal de Datos Abiertos de la Junta de Castilla y León para descargar el dataset. Esto se puede conseguir con una request sencilla en pocas líneas de código usando la librería **request** de Python, tal y como se ilustra en la figura 5.1. Los datos que se muestran analizados a lo largo de este documento se han accedido en mayo de 2024 por lo que los registros comienzan el 1 de enero de 2020 y se detienen el 30 de abril de 2024 para todos las Zonas Básicas de Salud

[1]

5.2 Visualización y limpieza

Una vez los datos se encuentran en un pandas dataset es necesario sumar el atributo de consultas totales de todos los consultorios de una misma zona cambiando así las instancias de nivel de consultorio a nivel de Zona Básica de Salud como se muestra en 5.1

Con este datset 5.1 es posible visualizar por ejemplo las 5 zonas con mayor 5.2 y menor 5.3 afluencia media:

[1] El código que se presente en este documento aparecerá en el formato de bloque de código gris con coloreado de texto que se ha adoptado del template de Overleaf [12]

Carga de datos de la API

```
# URL de la API
api_url = "https://analisis.datosabiertos.jcyl.es/api/explore/v2.1/catalog/dat"\
"asets/actividad-medicina-familia-consultorio/exports/json"
# Hacer la solicitud GET a la API
response = requests.get(api_url)
# Comprobar el estado de la respuesta
if response.status_code == 200:
    # Convertir la respuesta JSON a un diccionario de Python
    data = response.json()
    # Crear un DataFrame de pandas a partir de los registros

    df = pd.DataFrame(data)
    # Mostrar las primeras filas del DataFrame
    df.head()
    #Guardar en archivo csv
    df.to_csv("Consultas.csv")
```

Código 5.1: Fragmento de código para cargar datos abiertos de JCyL usando la API que proporcionan.

	fecha	zona_basica_de_salud	provincia	consultas_totales	fecha_dia	fecha_mes	fecha_weekday
0	2020-01-01	Alba de Tormes	Salamanca	20	01	01	2
1	2020-01-01	Arévalo	Ávila	7	01	01	2

Figure 5.1

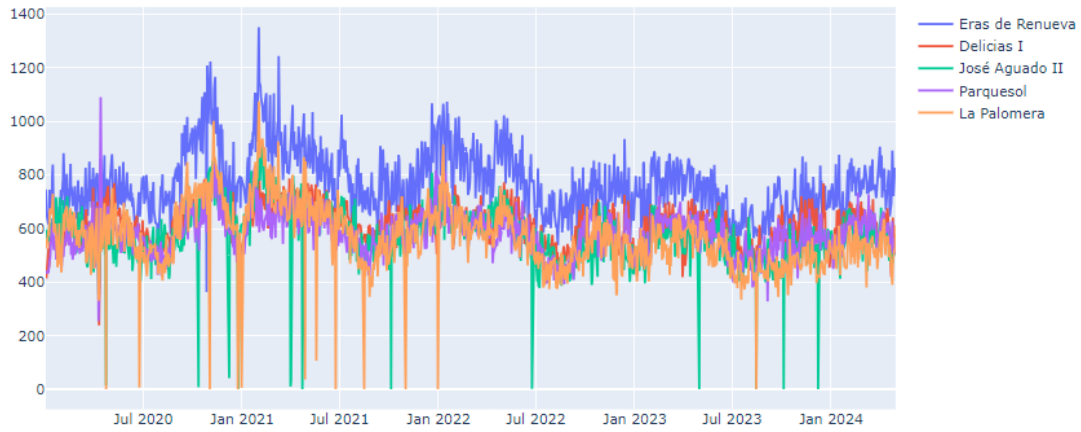


Figure 5.2: 5 zonas con mayor afluencia

Como puede observarse en estos gráficos parece existir un comportamiento diferente entre las zonas con mayor afluencia que presentan patrones curvados en comparación con las series que se muestran en las zonas menos visitadas que muestran un comportamiento más plano.

Sin embargo lo que parece más preocupante es la existencia de valores atípicos que pueden observarse

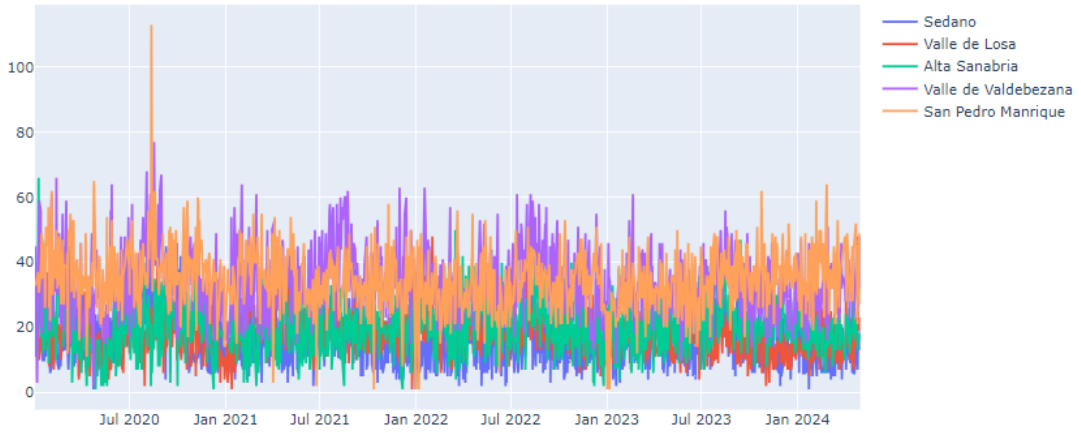


Figure 5.3: 5 zonas con menor afluencia

en 5.2. Esto parece indicar que hay días en los que la recogida de datos no se ha realizado por lo que es necesario investigar esta situación.

En primer lugar se ha extraído de cada fecha del conjunto de datos el día de la semana correspondiente lo que ha permitido confirmar que no hay datos para los fines de semana, cuando los consultorios no ofrecen su servicio regular. Por ello los fines de semana quedan excluidos del análisis y del modelo.

El conjunto de datos recogido tan solo presenta datos para las 247 zonas un 54% de los días (excluyendo como se ha mencionado los fines de semana), por lo que en el 46% restante será necesario imputar los datos perdidos de alguna forma ya que los modelos de predicción que se utilicen necesitarán entrenarse con una serie temporal ininterrumpida para cada zona. Además se ha comprobado que las distintas zonas presentan registros entr el 93% , como es el caso de Segovia II y Sotillo de Adrada y el 97% de Riaño y Periurbana por lo que parece razonable realizar la imputación ya que en ningún caso se sustituirá un porcentaje excesivo del total en cada zona.

El siguiente paso tomado para solucionar el problema de los datos faltantes ha sido la trnasformación del dataset para que cada zona cuente con una instancia en el conjunto de datos en cada fecha de forma que las fechas que no tuviera registradas se reflejen con un NaN en el campo ”consultas totales”.

Tras esto además se creó un nuevo conjunto calculando la media de consultas en cada fecha para cada zona ya que de esta forma será más fácil obtener una visión de la evolución anual de los datos.

Para tratar de entender qué días del año son *outliers* se ha tomado este nuevo dataset con el que es posible visualizar cuál ha sido el número medio de afluencia a las Zonas Básicas de Salud cada día del año como se aprecia en 5.4

En 5.4 se hace evidente que existen días en los que el número de registros es anormalmente bajo. Estos días son *outliers* en el conjunto de datos que pueden interferir con la predicción de días ordinarios y que deben ser tratados a parte.

Para empezar a lidiar con este problema se realizó un filtrado de *outliers* aplicando el rango intercuartílico. El proceso consiste en la detección de las fechas en las que el número medio de consultas se sale del intervalo (Q1 - 1.5 * IQR, Q3 + 1.5 * IQR) siendo Q1 y Q3 los cuartiles 1 y 3 y siendo IQR el rango intercuartílico IQR= Q3-Q1.

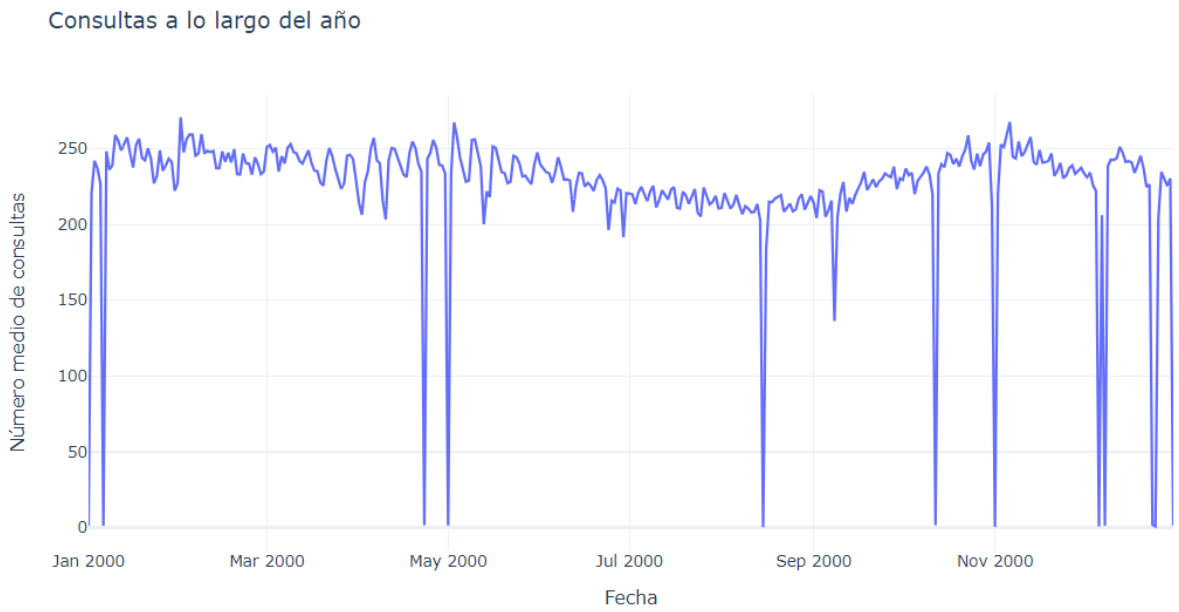


Figure 5.4: Número medio de consultas por zona y día del calendario (excluyendo fines de semana)

Este proceso dio como resultado una lista de fechas consideradas como "*outliers* autonómicos". Sin embargo teniendo en cuenta la naturaleza de los datos parece evidente que la mayoría de estos *outliers* tiene que ver con festividades. Por ello este mismo método se aplicó de forma separada por provincias obteniendo así distintas listas de "*outliers* provinciales".

Como paso final se comprobó que *outliers* autonómicos no eran compartidos por todas las provincias para eliminarlos en ese caso de la lista autonómica y del mismo modo se eliminó de las listas provinciales aquellos *outliers* compartidos por todas las provincias quedando de esta forma registrados únicamente en la lista de *outliers* autonómicos.

Las fechas identificadas en la lista autonómica coinciden en su totalidad con festividades y fueron:

- 01-01: Año nuevo
- 01-06: Día de Reyes
- 04-23: Día de Castilla y León
- 05-01: Fiesta del Trabajo
- 08-15: Asunción de la Virgen
- 10-12: Fiesta Nacional de España
- 11-01: Todos los Santos

- 12-06: Día de la Constitución Española
- 12-08: Inmaculada Concepción
- 12-24: Nochebuena
- 12-25: Navidad
- 12-31: Nochevieja

Por otro lado las listas de *outliers* provinciales identificados fueron las siguientes:

- Salamanca
  - 06-12: Fiestas patronales de Salamanca
  - 09-08: Natividad de la Virgen
- Soria
  - 01-10: Outlier no festivo
  - 04-10: Semana Santa
  - 06-25: San Juan en San Pedro Manrique
  - 10-02: Fiestas patronales de Soria
- Ávila
  - 04-02: Semana Santa
  - 05-02: San Segundo
  - 09-08: Natividad de la Virgen
  - 10-15: Fiestas patronales de Ávila
- León
  - 02-01: Santa Brígida
  - 24-06: San Juan
  - 08-16: San Roque
  - 09-08: Natividad de la Virgen
  - 10-05: San Froilán
- Burgos
  - 06-29: San Pedro y San Pablo ("Sampedros")
  - 08-16: San Roque
  - 09-12: Fiestas patronales de Miranda

- Segovia
  - 04-01: Semana Santa
  - 04-02: Semana Santa
  - 06-29: San Pedro
  - 08-16: San Roque
  - 10-25: San Frutos
- Zamora
  - 06-29: San Pedro
  - 08-16: Outlier no festivo
  - 09-08: Natividad de la Virgen
  - 12-07: Adelantamiento de la celebración de la Inmaculada
  - 12-26: Festividades navideñas
- Valladolid
  - 05-13 San Pedro Regalado
  - 09-08 Fietas de la Virgen de San Lorenzo
- Palencia
  - 02-02: Fiestas patronales de Palencia
  - 08-16: San Roque (Fiestas Villamuriel de Cerrato)
  - 09-02: San Antolín
  - 09-08: Fiestas de la Virgen del Valle (Saldaña)

Parece evidente que la gran mayoría de los *outliers* identificados tienen como justificación que los consultorios de muchas zonas no ofrecen sus servicios durante días festivos. Es por ello que se tomó la medida de crear un nuevo atributo que marque como outlier todas las entradas del dataset con día y mes coincidente con los *outliers* autonómicos así como las instancias de cada provincia que coincidieran con los *outliers* provinciales. De esta forma cuando se trabaje en la predicción podrá tratarse de forma separada los días festivos y los días regulares de forma que no haya interferencias entre los patrones de ambas situaciones.

A parte de todo esto se llevo a cabo una nueva detección de *outliers*, esta vez a nivel de zona básica de salud para encontrar días específicos en cada zona que presentan un comportamiento anómalo.

Sin embargo aunque se hayan separado será necesario crear un nuevo campo copia de "consultas totales" en el que sustituir los valores de estos *outliers* como si fueran datos faltantes, ya que para hacer la predicción de los días ordinarios es necesario contar con la serie para todos los días, y en el que imputar, es decir rellenar, los días sin recogida de datos que no se han identificado como *outliers*.

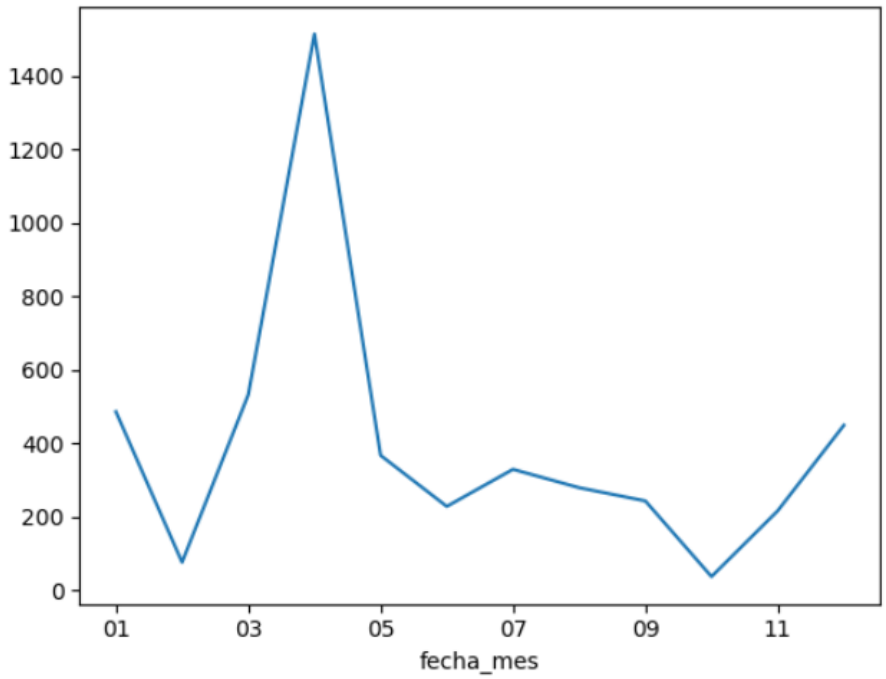


Figure 5.5: Distribución mensual de datos faltantes en días no identificados como *outliers*

Estas instancias sin registro y que no corresponden con *outliers* se dan mayoritariamente en abril como puede verse en 5.5, muy seguramente debido a la variabilidad entre los días de Semana santa Aún así no se ha considerado conveniente tratar estos días de forma especial y han sido tratados como el resto de días que faltan. Finalmente la decisión tomada para realizar la imputación fue la aplicación de uno de los múltiples métodos disponibles como lo es la interpolación lineal ya que parece que en el conjunto de datos no hay datos que faltan consecutivos que pudieran afectar al proceso. Se esta forma, siendo m el número de fechas entre la primera y última fecha del dataset, xj las distintas fechas, n el número de Zonas básicas de salud, y $f_i(x_j)$ el número de consultas para la fecha j en la zona i , la interpolación consiste en crear una nueva serie f' a partir de f de tal forma que

$$\forall i \in [1,n]_1 \forall j \in [1,m] : f_i'\left(x_j \mid x_{j-1}, x_{j+1}\right) = f\left(x_{j-1}\right) * \frac{f\left(x_{j+1}\right) - f\left(x_{j-1}\right)}{x_{j+1} - x_{j-1}} \left(x_j - x_{j-1}\right)$$

El código aplicado ha sido el del Listado 5.2:

De este modo los valores faltantes tanto de los *outliers* como de los no *outliers* serán sustituidos por los valores calculados de la interpolación creando así una nueva serie que a la que se hará referencia como ”depurada” permitiendo así mantener la continuidad en todas las fechas siguiendo la tendencia de los datos con los que si se cuenta de un modo similar a lo que se observa en la Figura 5.6.

El efecto de la interpolación salta a la vista al comparar la serie original con la depurada como puede apreciarse en 5.7 donde se comprueba que los días con una media de consultas anormalmente bajas han sido corregidos. Además este gráfico comprueba también que para todos los días del año hay 247 registros, es decir, que hay valores para todas las zonas en todas las fechas. Una vez tratados los datos faltantes se consideró adecuado estudiar la tendencia de la evolución de la afluencia a lo largo de un año, por lo que haciendo uso del conjunto de datos anual se puede apreciar el comportamiento de la serie depurada como se muestra en 5.8 donde se hace evidente un patrón claro un forma de U que

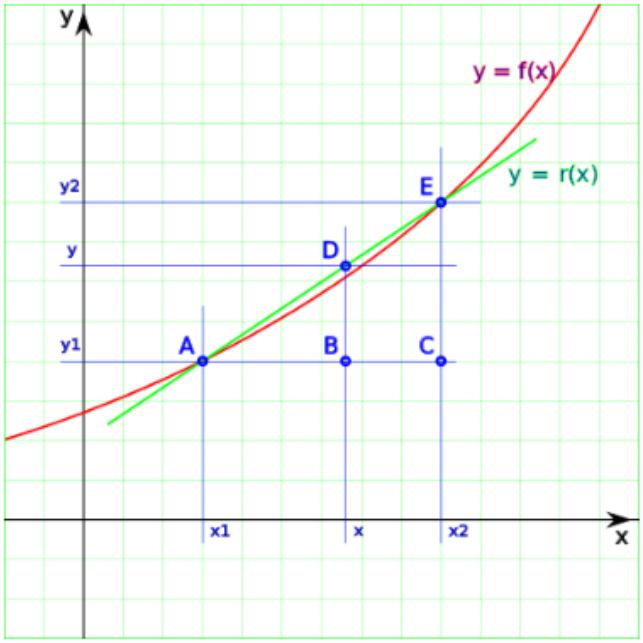


Figure 5.6: Descripción gráfica del proceso de interpolación

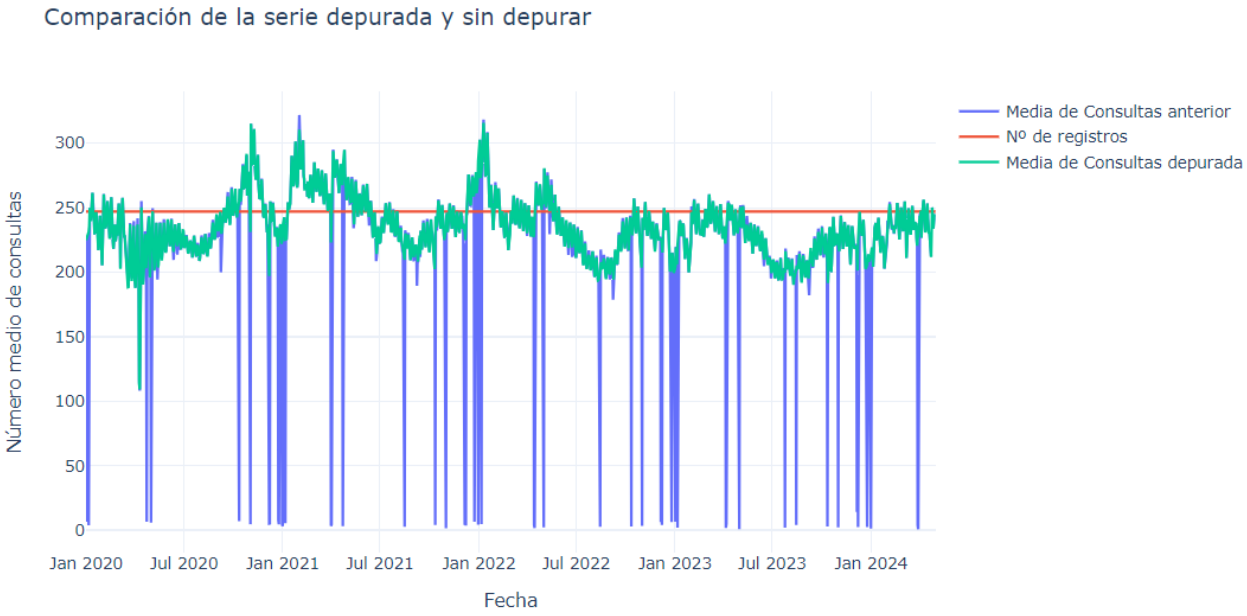


Figure 5.7: Comparación de la serie tras aplicar la interpolación

Imputación de datos faltantes por interpolación

```
df_final["consultas_totales_imp"]=df_final["consultas_totales"]
def interpolate(group):
    mask = (group['consultas_totales'].isna())
    group.loc[mask, 'consultas_totales_imp'] =
    ↪  group['consultas_totales'].interpolate()
    return group

df_final =
↪  df_final.groupby('zona_basica_de_salud').apply(interpolate_if_not_outlier)
df_final=df_final.reset_index(drop=True)
```

Código 5.2: Fragmento de código en el que se muestra la aplicación de la interpolación lineal para sustituir los datos faltantes en cada Zona Básica de Salud.

corresponde con una reducción del número de consultas durante el verano causada muy probablemente por los desplazamientos de habitantes de la comunidad a distintas zonas de veraneo durante el periodo estival.

5.3 Clustering

Como último paso del análisis previo se creyó conveniente tratar de agrupar distintas zonas con comportamientos similares puesto que en 5.2 y 5.3 se pudo notar una diferencia en el comportamiento de la afluencia a lo largo del año.

Para lograr este fin primero se realizó una transformación del dataset anual de tal forma que se creo un nuevo dataset con 247 instancias, una para cada zona, con 52 atributos correspondiente a la media semanal de afluencia media en la zona para cada semana del año.

A las 247 zonas se les aplicó normalización, pues lo que se busca es agrupar las series según la forma de su evolución en el tiempo sin que influya la cantidad media de consultas que registran. Tras la normalización y tomando como referencia el procedimiento seguido por [6] el conjunto de datos fue sometido a un análisis de componentes principales para reducir la dimensionalidad y permitir una visualización de la separación realizada por los clusters para, seguidamente, aplicar sobre dichas componentes clustering con el algoritmo de KMedias.

En el código utilizado se ha hecho uso de las librería sklearn.cluster y sklearn.decomposition de esta forma:

A la vista del plot 5.9 creado tras aplicar las técnicas mencionadas, se puede advertir que existen 4 grupos que quedan distinguidos en su mayoría por la primera componente principal y que parece razonable analizar.

En primer lugar, añadiendo la etiqueta de cluster correspondiente a cada zona al conjunto de datos anual que se creo con anterioridad se puede obtener un plot como el que se muestra en 5.10 que plasme la evolución de la afluencia media a las zonas de cada cluster a lo largo del año. Además de este gráfico se han identificado las zonas con más afluencia de cada cluster para comprender mejor

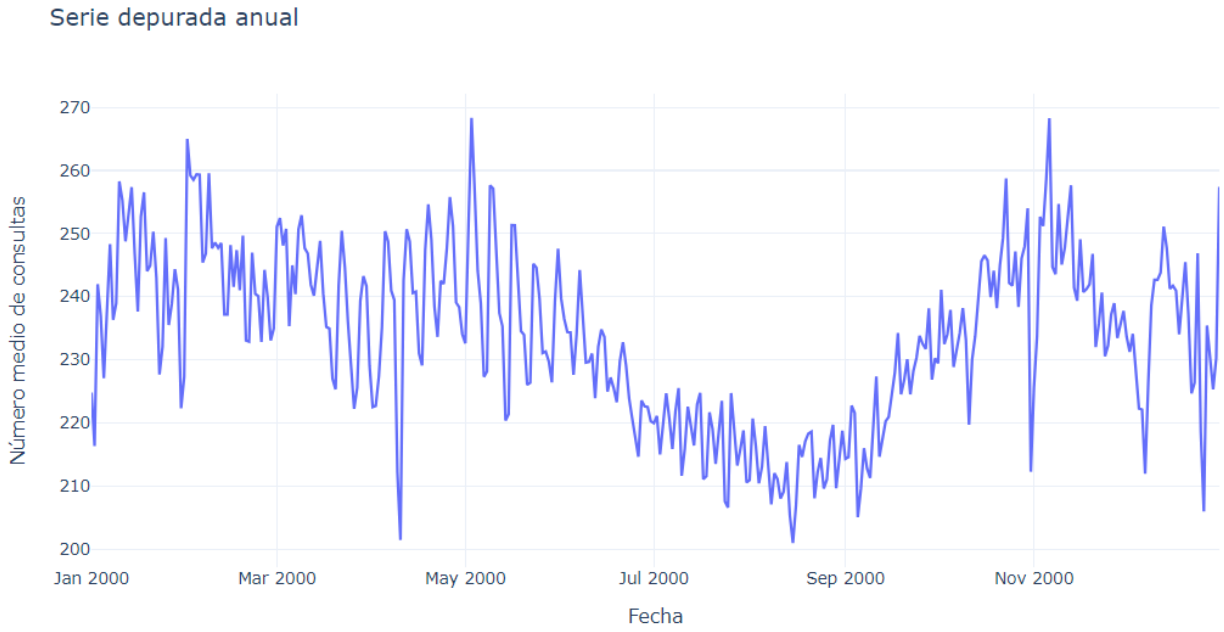


Figure 5.8: Serie anual depurada

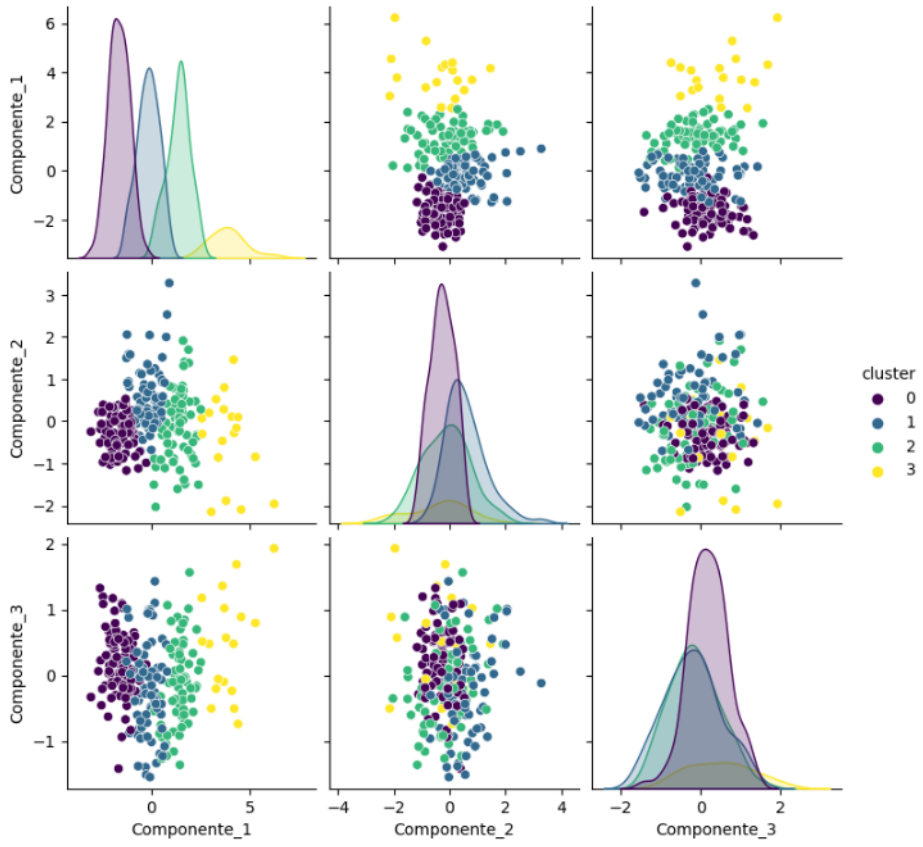


Figure 5.9: Clustering aplicado sobre las 3 primeras componentes principales del conjunto con datos semanales de cada zona

PCA y Kmedias para clustering de series temporales

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
#Extraer datos semanales
X = df_semanal_pivot.drop(['zona_basica_de_salud',"provincia"], axis=1)
# Aplicar la normalización por fila y devolver un nuevo DataFrame
X_scaled = X.apply(normalize_row, axis=1)

# Aplicar PCA para reducir la dimensionalidad a 3 componentes principales
n_com=3
pca = PCA(n_components=n_com)
X_pca = pca.fit_transform(X_scaled)

# Entrenar el modelo de clustering (k-means) en los datos reducidos
n_clus=4
kmeans = KMeans(n_clusters=n_clus, random_state=42)
clusters = kmeans.fit_predict(X_pca)
```

Código 5.3: Fragmento de código en el que se muestra la aplicación de la descomposición en componentes principales los 52 atributos de afluencia semanales de las distintas Zonas Básicas de Salud y el posterior agrupamiento de las zonas aplicando clustering con el algoritmo de las KMedias sobre las 3 componentes

cuáles conforman cada uno y de esta forma ha sido posible describir cada grupo.

- Cluster 0:
  - Zonas: Eras de Renueva, Delicias I, José Aguado II, Parquesol, La Palomera...
  - Comportamiento: Ligeros picos de afluencia en febrero , mayo y noviembre y marcado descenso tanto de la afluencia como de la variabilidad en verano
  - Interpretación: Zonas situadas en ciudades o grandes centros urbanos cuyos habitantes se marchan en verano de vacaciones o a zonas rurales lo que conlleva un descenso de la afluencia a los consultorios de atención primaria.
- Cluster 1:
  - Zonas: Medina del Campo Urbano, Aranda Norte, Miranda Oeste, Ribera del Órbigo, Benavente Sur, Ciudad Rodrigo...
  - Comportamiento: Plano a lo largo de todo el año con un ligero descenso en la afluencia y una notable reducción de la variabilidad en verano
  - Interpretación: Zonas de salud que se encuentran en pueblos de gran tamaño cuyo comportamiento se asemeja al de las ciudades por su tamaño pero que cuentan con una población que opta por permanecer en sus hogares en verano al encontrarse ya en un entorno rural.
- Cluster 2:
  - Zonas: Bañeza II, Aranda Rural, Cebreros, Guijuelo, Vitigudino, Briviesca, Cistierna...
  - Comportamiento: Plano a lo largo del año y con menos variabilidad que los anteriores
  - Interpretación: Similar al cluster 1 pero con pueblos más pequeños que sufren de menos variabilidad
- Cluster 3:

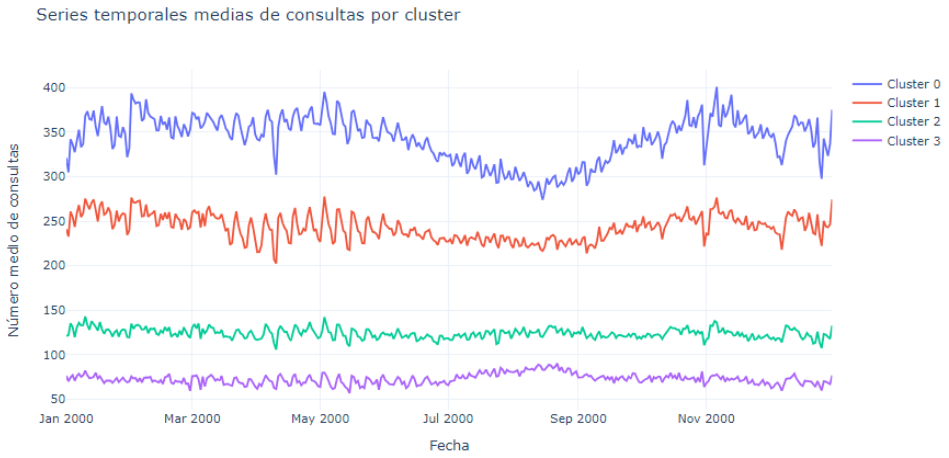


Figure 5.10: Evolución de la media de consultas por cluster a lo largo del año

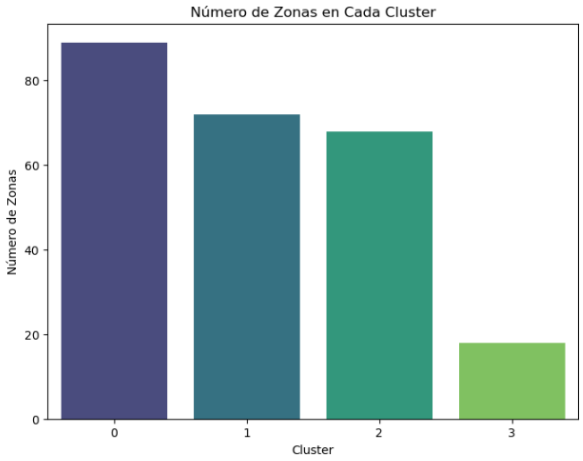


Figure 5.11: Proporción de clusters

- Zonas: Burgohondo, Barco de Ávila, Piedrahita, Quintanar de la Sierra, Muñana, Riaño...
- Comportamiento: Aumento de la afluencia en verano
- Interpretación: Pueblos muy pequeños que apenas cuentan con afluencia durante todo el año pero que en verano experimentan un aumento de la afluencia ya que una parte importante de su población pasa largos periodos de tiempo fuera de ellos hasta finalmente regresar en verano. Esto provoca finalmente un aumento de las consultas en las Zonas Básicas de Salud de estos pequeños núcleos rurales.

Como fin al análisis de los datos y una vez entendida la naturaleza los distintos grupos es interesante visualizar la distribución de estos. Como se puede comprobar en 5.11 y en 5.12 el grupo menos común es el cluster 3 (zonas rurales pequeñas), que aparece en mayor proporción en Ávila y Soria, mientras que el grueso de las zonas se reparten entre los grupos 2 y 3. Sin embargo el grupo más común es el cluster 0 (zonas urbanas) que destaca enormemente en Valladolid.

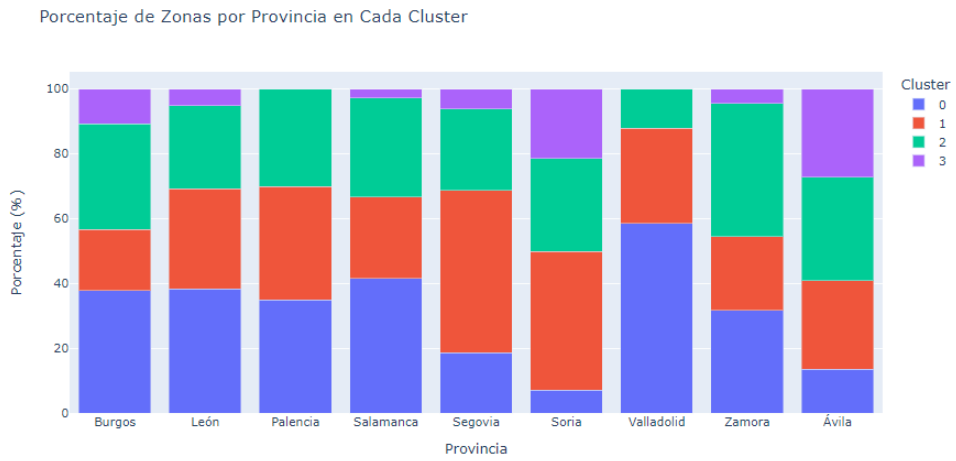


Figure 5.12: Porcentaje de clusters por provincia

Chapter 6

Soluciones Existentes

6.1 Series temporales

Las series temporales son aquellas colecciones de observaciones de cierto fenómeno en momentos sucesivos del tiempo por lo que este sería un término adecuado para referirse al tipo de datos que se está tratando en este proyecto en el que se cuenta, tras el preprocesamiento, con una medida de la afluencia para cada día en cada Zona Básica de Salud.

En este caso concreto se contará con 247 series temporales, cada una correspondiente a una zona distinta, de forma que siendo $t = 0, 1, 2, ..., n$ cada una de las n distintas fechas (equiespaciadas por 1 día) desde el 01-01-2020 hasta la última fecha registrada en el momento de acceso a los datos y siendo $i = 1, 2, ..., 247$ todas las distintas Zonas Básicas de Salud existen 247 funciones tales que:

$$\forall t, \forall i : \exists x_i(t)$$

Por lo tanto es de vital importancia para desarrollar un buen modelo identificar las distintas alternativas existentes en el campo de la predicción de series temporales y entender qué ventajas e inconvenientes ofrece cada una.

6.2 ARIMA

Los orígenes de la historia del desarrollo de modelos predictivos para series temporales se remontan a la introducción de las ecuaciones de regresión de Yule en 1927 que derivarían con el tiempo en el primer modelo que se va a tener en cuenta, ”auto-regressive integrated moving average model” más conocido como ARIMA que vino de la mano de Box y Jenkins en 1970.[10]

Los modelos ARIMA combinan dos enfoques, el enfoque autoregresivo (AR) y el de la media móvil (MA) (”Moving Average”).

Para la parte autoregresiva (AR) se seleccionan p observaciones anteriores de tal forma que se asume una correlación parcial entre cada observación de la serie $x(t)$ y las p observaciones anteriores $x(t-1)...x(t-p)$ de tal forma que:

$$x(t) = \beta_1 + \phi_1 * x(t-1) + ... + \phi_p * x(t-p)$$

Los p coeficientes dependen de la correlación entre las observaciones y la serie retrasada p unidades

Por otro lado, la parte de la media móvil se basa en analizar el error para las observaciones anteriores de forma que se obtenga una pejor predicción para la actual, de tal forma que siendo $\epsilon(t-1), ..., \epsilon(t-q)$ los errores cometidos en las observaciones anteriores:

$$x(t) = \beta_2 + \omega_1 * \epsilon(t-1) + ... + \epsilon_p * x(t-q)$$

Los q coeficientes son calculados a partir de las correlaciones.

Combinando ambas ecuacuiones se obtiene el modelo ARMA con p representando el desfase en AR y q representando el tamaño de la ventana de la media móvil en MA.

Un ultimo detalle a comentar es que estos modelos asumen que la serie es estacionaria, es decir, que tiene media y desviación típica constante y que no presenta estacionalidad. Para asegurar esto se lleva a cabo un proceso conocido como diferenciación en el que se crea una nueva serie de tal forma que:

$$y(t) = x(t+1) - x(t)$$

Este proceso se aplica d veces hasta lograr la estacionaridad, constituyendo así el parámetro correspondiente a la característica "Integrated" (I) de ARIMA. [7]

A pesar de haber sido el estándar durante un largo periodo de tiempo la alta complejidad y no linealidad de los datos ha hecho que nuevas opciones basadas en inteligencia artificial se impongan frenta a los modelos autoregresivos, que logran una precisión .[10]

6.3 Redes Neuronales - LSTM Y CNN

Una vez comprendido que el modelo ARIMA no es la opción con mejor desempeño en la actualidad parece natural buscar una alternativa en la inteligencia artificial, en concreto en las redes neuronales. Tal como sugieren distintos artículos [5], [10], [11] el principal tipo de red que forma la base de resultados consistentes en la predicción de series temporales son las conocidas como "Long-Short Term Memory" (LSTM).

Las LSTM son un tipo concreto de "Recurrent Neural Network" (RNN). Estas redes recurrentes se distinguen de una red neuronal como podría ser un perceptrón multicapa en el hecho de que permite conexiones de retroalimentación en su estructura lo que las permite realizar tareas que requieran memoria puesto que en el momento t la red utiliza se transmite su propio estado como información para $t+1$, que a su vez se transmitirá a $t+2$ manteniendo así siempre una cierta influencia de las observaciones pasadas.

La arquitectura LSTM es un tipo concreto de RNN que permite capturar dependencias a largo plazo gracias a que su estructura de "puertas" le permite solucionar el problema de desvanecimiento del gradiente que sufren otras RNNs.

La estructura de una celda LSTM se corresponde con lo que se puede observar en la Figura 6.1 . Por un lado encontramos una "puerta de olvido" que decide que información descartar, por otro lado existe

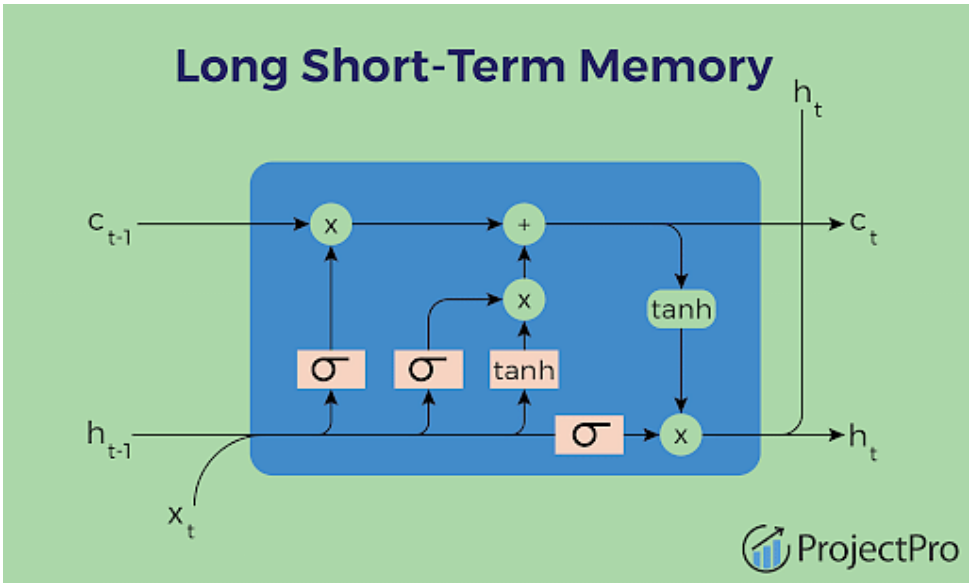


Figure 6.1: Celda LSTM - Imagen tomada de [15]

una ”puerta de entrada” que provee a la celda con nueva información y finalmente se cuenta con una ”puerta de salida” que determina el siguiente estado. En particular, la puerta de olvido desempeña un papel fundamental al determinar qué información del paso temporal anterior se debe desechar utilizando una función sigmoide. Esta puerta constituye un componente clave en la estructura de puertas de las LSTMs. En las redes LSTM la información pasada se transmite por c_t (Cell State) y h_t (Hidden State) que representan la memoria a largo y corto plazo respectivamente. Un pequeño recorrido deabajo a arriba y de izquierda a derecha por la celda puede permitir entender mejor el funcionamiento interno de este modelo.

En cada celda la nueva entrada se incorpora como x_t y es se suma con el h_t anterior. Este resultado pasa por la función sigmoide $\sigma(x) = \frac{1}{1+e^x}$ que devuelve un valor entre 0 y 1 que representa el porcentaje de la memoria a largo plazo que se recuerda. Este porcentaje se muliplica por la memoria a largo plazo anterior c_{t-1} constituyendo así la ”puerta de olvido antes mencionada” (a la izquierda en 6.1).

A continuación la suma de x_t y h_t se pasa por otra una función $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ que devuelve un número entre (-1, 1) como potencial adición a la memoria a largo plazo y otra función sigmoide que determina que porcentaje de esta adición conservar. Finalmente la adición calculada se añade a la memoria de largo plazo completando así lo que se conoce como puerta de entrada(en el centro en 6.1).

Finalmente se encuentra la puerta de salida en la que se aplica $tanh$ a la memoria a largo plazo c_t modificada en el paso anterior para obtener una potencial adición a la memoria a largo plazo y en la que de nuevo se aplica la función sigmoide a la suma de x_t y h_{t-1} para determinar el porcentaje de esta modificación que aplicar a h_t [21]

Estas celdas descritas producen cada instante t de la serie una predicción como resultado del ”Hidden State”

Como añadido al modelo LSTM se puede aplicar lo que se conoce como mecanismo de atención inspirado por la atención biológica de los animales para fijarse en sus objetivos. Este añadido consiste en modificar los ht resultado de la red de forma que siendo W el vector de pesos estimado, y b el vector de bias estimado, el nuevo h’ se calcula como:

$$e = tanh(W * h) + b$$

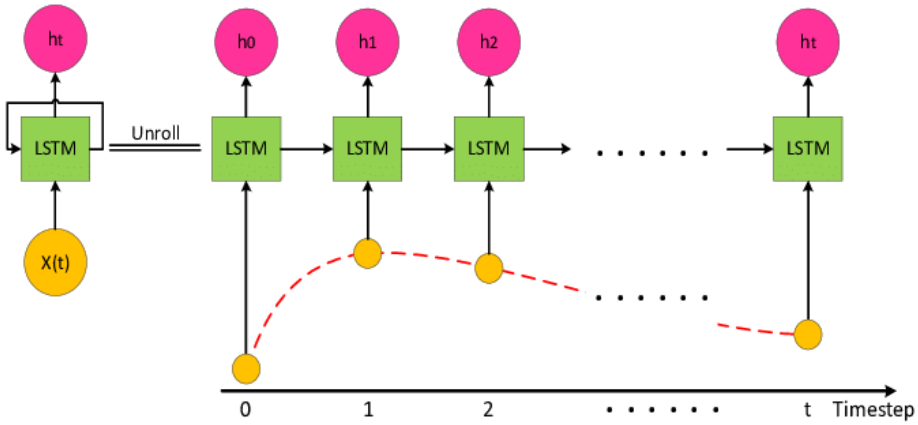


Figure 6.2: Red LSTM "desenrrollada" - Imagen tomada de [21]

$$\alpha = \frac{\exp e}{sum(\exp e)}$$

$$h'(t) = \alpha * h$$

Esta modificación ha demostrado tener resultado positivos.[21] y [10]

Sin embargo la mayoría de estas redes producen un resultado 'one-step', es decir que se predice solo 1 unidad de tiempo en el futuro. El problema es que en la mayoría de casos prácticos de predicción una predicción tan corta es inútil. Es por ello que existe un enfoque 'multistep' que genera predicciones para múltiples unidades de tiempo a futuro.

Existen dos clases de redes 'multi-step', las que generan las predicciones iterativamente o directamente. Los modelos iterativos gneran una predicción 'one-step' y a continuación generan la siguiente utilizando el valor predicho como nueva entrada. El problema de estos métodos consiste en la acumulación de errores y en una estimación optimista del error [16]

Las redes LSTM 'multistep' directas como la que se describe en 6.3 emplean múltiples valores de entrada (X como p valores anteriores de una variable exógena e Y como q valores anteriores de la serie a predecir en la imagen) y producen como resultado n predicciones a futuro de la serie estudiada conformando así un output múltiple. [22] Además de las redes LSTM, existe otro tipo de red neuronal que es usado con frecuencia en el ámbito de la predicción de series temporales y este es el caso de las redes convolucionales (CNN). Las redes convolucionales por si solas no tienen un gran rendimiento en este tipo de problemas, pero se ha constatado que pueden ofrecer muy buenos resultados al emplearse en combinación con las redes LSTM usándolas en cualquier orden o en paralelo puesto que no parece haber un consenso en qué orden ofrece mejores resultados.[11] [13]

La arquitectura de la red neuronal híbrida CNN-LSTM incluye capas de CNN que extraen características significativas de datos de entrada unidimensionales, combinadas con LSTM que las interpretan para apoyar la predicción de secuencias. La Figura 2 representa la arquitectura del modelo híbrido CNN-LSTM para predecir datos de series temporales univariadas unidimensionales .

El modelo híbrido CNN-LSTM puede predecir eficazmente datos de series temporales univariadas no lineales que implican una alta volatilidad e incertidumbre en los patrones de datos.[8]

Las redes CNN están principalmente pensadas para procesar datos en forma de cuadrícula y cuentan con distintos tipos de capas. Por un lado se encuentran las capas convolucionales que utilizan multitud de filtros sobre la cuadrícula para detectar distintos patrones locales y obtener nuevas características.

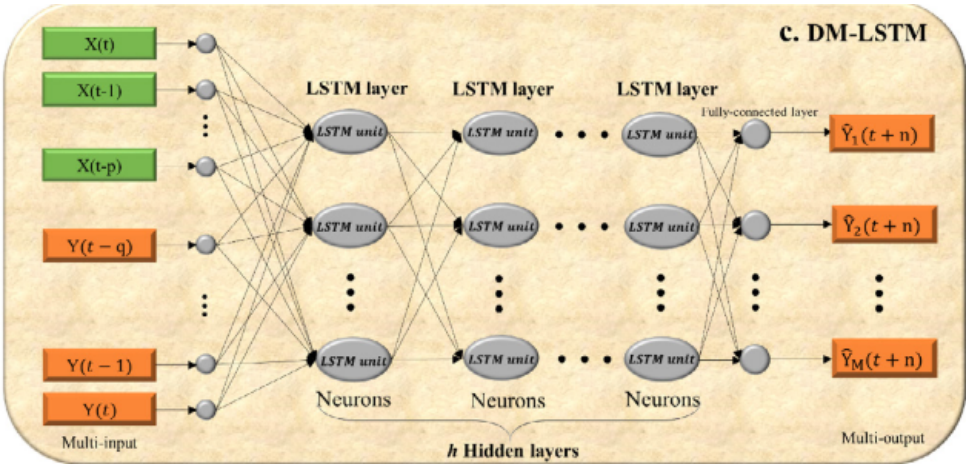


Figure 6.3: Red LSTM multistep
[22]

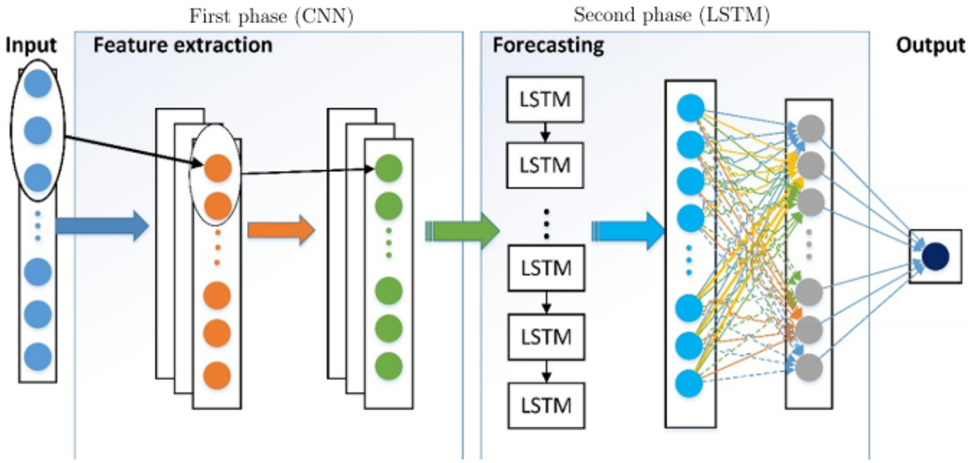


Figure 6.4: Arquitectura de modelo combinado CNN+LSTM - Imagen obtenida de [8]

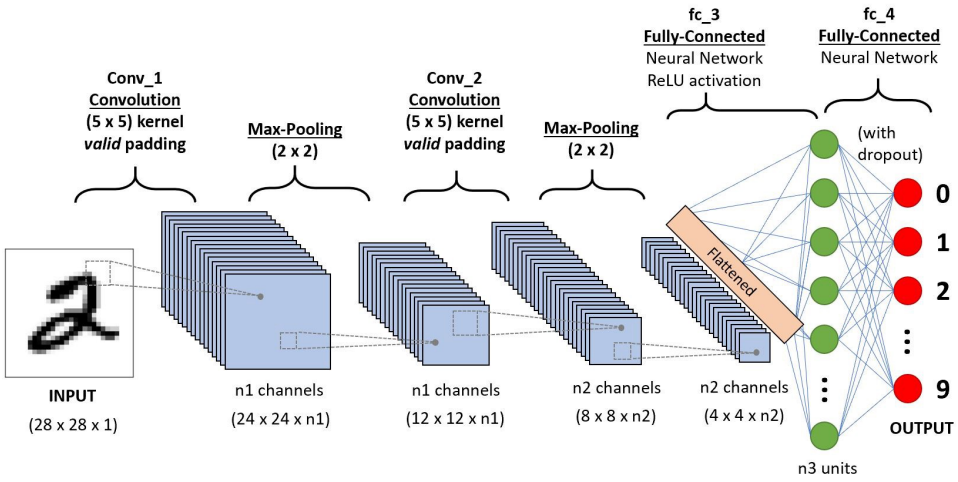


Figure 6.5: Arquitectura CNN - Imagen tomada de [13]

Estas características son sometidas a lo que se conoce como MaxPooling para reducir su dimensionalidad para terminar con un aplanado que da como resultado vectores que podrán emplearse como entrada en otra red como puede serlo una red LSTM. El proceso es similar a lo que puede observarse en 6.5

Hoy en día existen múltiples frameworks como Keras y Pytorch que permiten crear estas redes sin preocuparse de codificar todas las operaciones internas mencionadas por lo que esta opción es realmente viable.

6.4 Extreme Gradient Boosting (XGB)

Además de las redes neuronales, diversos artículos [17] [19] apuntan a que el uso de técnicas de Boosting sobre características extraídas de los datos puede producir predicciones que superen a los modelos ARIMA y que sean capaces de capturar relaciones no lineales entre los datos.

El denominado como Gradient Boosting es una técnica poderosa de *Machine Learning* basada en el ensamble. Como técnica de ensamblado, el Gradient Boosting combina los resultados de varios aprendices débiles, conocidos como aprendices base, para construir un modelo único. Normalmente, el Gradient Boosting utiliza árboles de decisión como aprendices base. Como el resto de métodos de boosting, el Gradient Boosting se basa en la idea de ir creando nuevos modelos con cada vez menos error de manera consecutiva pero no independiente. Los nuevos aprendices base se crean con el objetivo de minimizar una función de pérdida asociada a todo el ensamblado y las instancias que han sido peor predecidas en la iteración previa ganan pesos mayores a la hora de influir en el entrenamiento en un proceso similar a lo que puede observarse en 6.6.

XGBoost, que significa Extreme Gradient Boosting, es una implementación específica muy popular del Gradient Boosting.El Extreme Gradient Boosting cuenta con algunos detalles específicos como el uso de regularización avanzada que mejora la generalización del modelo y reduce el sobreajuste. Además, calcula gradientes de segundo orden de la función de pérdida, proporcionando más información sobre la dirección del gradiente, facilitando la minimización de la función de pérdida. Otras ventajas de la implementación de XGBoost incluyen su capacidad incorporada para manejar valores faltantes y permitir el procesamiento paralelo, lo que compensa la mayor demanda computacional. [20]

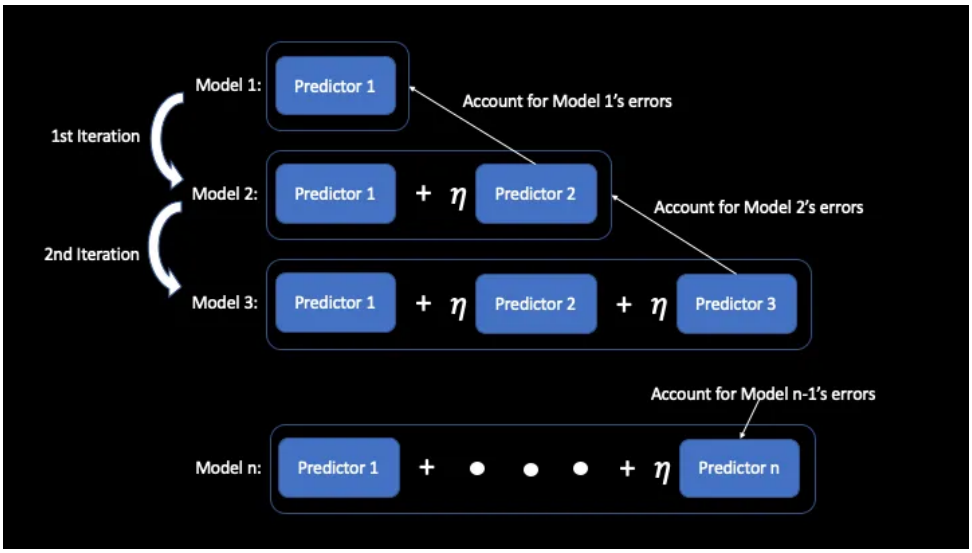


Figure 6.6: Proceso iterativo del XGB - Imagen tomada de [18]

Existen distintas librerías como puede serlo XGBoost en Python que ofrecen la posibilidad de aplicar este ensamble a los datos dejando de lado los detalles internos de codificación. No obstante

6.5 Prophet

Una última alternativa a tener en cuenta como modelo predictivo para series temporales es Prophet que también ha sido objeto de estudio en distintos artículos[19] [11] con resultados no muy alejados de los otros modelos mencionados. Prophet es una biblioteca de previsión de series temporales que fue desarrollado por el equipo de Core Data Science de Facebook en febrero de 2017 y que queda descrito en detalle en [14]

Este modelo se creó como un modelo de pronóstico de series temporales diseñado para manejar las características comunes de las series temporales empresariales. Prophet cuenta con parámetros muy intuitivos y implementación está disponible como software de código abierto en Python y R, llamada Prophet.

El modelo matemático subyacente puede dividirse en tres componentes principales: tendencia, estacionalidad y días festivos, combinados en la siguiente ecuación: $y(t) = g(t) + s(t) + h(t) + {}_t$

- $g(t)$: Función de tendencia que modela cambios no periódicos.
  - Siendo t el tiempo, C el cambio máximo en el valor de la tendencia, k la tasa de cambio (un k mayor significa cambios más bruscos en la tendencia, mientras que un k menor indica cambios más suaves y graduales) y m el punto de inflexión o punto medio en el tiempo.
  - $g(t) = \frac{C}{1+\exp k(tm)}$
- $s(t)$: Representa cambios periódicos (e.g., estacionalidad semanal y anual).
  - Siendo t el tiempo, P el periodo de estacionalidad y a_n y b_n coeficientes que se ajustan durante el entrenamiento se puede obtener $s(t)$ mediante una serie de Fourier:
  - $s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right)\right)$
- $h(t)$: Efectos de los días festivos.

- $_t$: Término de error que representa cambios impredecibles no acomodados por el modelo. Se asume que está distribuido normalmente.

Part III

Experimentación

Chapter 7

Metodología de selección de modelos

7.1 Hoja de ruta

Una vez repasados los distintos modelos es necesario determinar qué pruebas se van a realizar para determinar cual es el mejor modelo de predicción. De entre todas las alternativas se ha considerado conveniente poner a prueba 3 opciones: LSTM Multistep, XGBoosting y Prophet.

Teniendo en cuenta la existencia de 247 zonas a entrenar y la enorme variedad de hiperparámetros que pueden influir en el rendimiento de cada uno de estos modelos parece conveniente reducir el número de zonas a 12, tres representativas de cada cluster conformado en el apartado de análisis de datos, sobre las que testear el desempeño de los distintos modelos.

La forma de proceder consistirá en tratar de determinar si alguno de los modelos tiene mejor rendimiento en unos clusters o en otros o si por el contrario hay un modelo que es superior en todos los casos poniendo en comparación para cada zona los tres modelos con múltiples variantes que comprendan cierto rango de valores de sus hiperparámeteros.

Los hiperparámetros de un modelo son parámetros cuyo valor se establece antes de comenzar el proceso de aprendizaje y no se ajustan automáticamente durante el entrenamiento. A diferencia de los parámetros del modelo, que se aprenden y optimizan a partir de los datos durante el entrenamiento (como los pesos en una red neuronal), estps hiperparámetros deben ser definidos por el usuario o mediante un proceso de optimización previo al entrenamiento.

7.2 Criterio

Para realizar la selección de modelos el criterio a tener en cuenta se tratará del más común que se puede encontrar en la bibliografía de estudios de predicción. Este criterio se trata de la raíz del error cuadrático medio (RMSE), de tal forma que siendo $\hat{x}_i(t)$ la predicción de la serie i en el tiempo t y siendo $x_i(t)$ el valor real de esta:

$$RMSE(x, \hat{x}) = \sqrt{\frac{\sum_{t=0}^{n-1}(x_i(t) - \hat{x}_i(t))^2}{N}}$$

El RMSE es siempre no negativo y valores más bajos de este indican un mejor ajuste del modelo a los datos. Esta métrica es muy utilizado dado que tiene una interpretación muy útil de tal forma que un RMSE de x indica que, en promedio, las predicciones del modelo se encuentran a x unidades de diferencia del valor real a predecir.

7.3 Metodología de entrenamiento

En este trabajo se adoptará un enfoque de entrenamiento, validación y prueba para desarrollar y evaluar los modelos de predicción. Este enfoque es esencial para garantizar que el modelo no solo se ajusta bien a los datos de entrenamiento, sino que también generaliza adecuadamente a datos no vistos.

El proceso comienza con la división de los datos disponibles en tres conjuntos: entrenamiento, validación y prueba. El conjunto de entrenamiento se utiliza para ajustar los parámetros del modelo mediante la minimización de una función de pérdida específica, en este caso el RMSE. La selección del RMSE como criterio de pérdida es adecuada en este contexto porque mide la magnitud promedio del error de las predicciones del modelo, penalizando de manera significativa los errores grandes.

Después de ajustar el modelo con el conjunto de entrenamiento, se empleará un conjunto de validación para realizar una evaluación preliminar de su desempeño. El conjunto de validación se utiliza para ajustar los hiperparámetros del modelo, ya mencionados. Este paso ayuda a prevenir el sobreajuste, asegurando que el modelo no se ajuste demasiado a los datos de entrenamiento y pueda generalizar mejor a datos no vistos.

Finalmente, el conjunto de prueba se utiliza para evaluar la capacidad predictiva del modelo en datos completamente nuevos. Esta evaluación proporciona una estimación imparcial de su rendimiento real, ya que el modelo no ha sido expuesto a estos datos durante el proceso de entrenamiento o ajuste de hiperparámetros. La métrica RMSE en este conjunto de prueba permitirá cuantificar la precisión de nuestras predicciones y comparar el desempeño entre diferentes modelos [9].

Esta división en tres grupos se ha llevado a cabo tomando todos los datos desde 2020 hasta abril de 2023 como entrenamiento y utilizando el último año desde abril de 2023 a abril de 2024 como validación y test. Esta división ha planteado ciertos problemas ya que se trata de dos mitades de año distintas para validación y test, pero no ha sido posible realizar la división de otro modo ya que se comprobó en las primeras etapas de experimentación que reservar un año para validación y otro para test reducía enormemente la calidad del entrenamiento y producía unas predicciones mucho más pobres.

7.4 Tecnología

Para facilitar la selección de modelos y la experimentación se ha hecho uso de la ya descrita plataforma MLFlow y en concreto de su componente MLFlow Tracking. La experimentación en MLFlow se basa en dos componentes básicos, los experimentos y los runs.

Cada experimento está asociado a uno de los modelos que se van a comparar (LSTM, Prophet...) y contiene uno o más runs de tal forma run representa una ejecución de un modelo con una configuración específica. En cada run se registra tanto qué zona está siendo entrenada como los distintos hiperparámetros concretos de ese modelo específico junto con la medida de error en los conjuntos de entrenamiento, validación y test.

Además de todo esto es posible guardar para cada run lo que se conoce como artefactos, es decir, archivos adicionales que se quieran ligar a cierta ejecución. Esto es muy útil ya que permite guardar automáticamente en cada run el modelo entrenado que puede ser recuperado en cualquier momento así como plots que permitan visualizar el ajuste que se ha realizado.

Toda esta información se guarda en lo que se denomina el 'tracking server' que por defecto se encuentra en el almacenamiento local creándose una carpeta llamada `mlruns` en el directorio donde se ejecuta MLFlow, a menos que se especifique lo contrario. Viisualizar los datos de los runs de esta carpeta es increíblemente fácil ya que MLFlow cuenta con una iterfaz de usuario web a la que se puede acceder ejecutando el comando **$ mlflow ui**

Chapter 8

Diseño de la aplicación

La arquitectura de la aplicación con la que se va a llevar a cabo toda la experimentación, cuyo nombre es TFG_APP, está diseñada para facilitar el desarrollo, entrenamiento y seguimiento de modelos de machine learning en un entorno basado en contenedores Docker. A continuación se detalla la estructura y el funcionamiento de los componentes principales de esta aplicación que pueden verse representados gráficamente en la Figura 8.1.

8.1 Contenedor Aplicación

TFG_APP se implementa como un contenedor Docker que proporciona un entorno de ejecución uniforme y aislado para todos los componentes de la aplicación. Este enfoque asegura la portabilidad y la consistencia del entorno de desarrollo y producción.

8.1.1 Componentes

1. **Configuración Global (config.py)**:
   - Contiene constantes y configuraciones críticas que afectan al comportamiento global de la aplicación. Esta clase es esencial para la manipulación y gestión centralizada de parámetros clave que guían el flujo de trabajo y la configuración de modelos.
2. **Punto de Entrada (app.py)**:
   - `app.py` sirve como el punto de entrada principal de la aplicación. Se ejecuta automáticamente al iniciar el contenedor Docker mediante el comando `docker-compose up`. Esta clase inicia las operaciones centrales de la aplicación y coordina la secuencia de actividades.
3. **Descarga y Preprocesamiento de Datos (`data`)**:
   - download_data.py: Esta clase maneja la interacción con una API externa alojada en el portal de datos abiertos de la Junta de Castilla y León. Se encarga de descargar los conjuntos de datos necesarios para los experimentos.
   - preprocess_data.py: Luego de la descarga, preprocess_data.py realiza operaciones de preprocesamiento sobre los datos adquiridos. Este paso es crucial para garantizar que los datos estén limpios y estructurados adecuadamente para su análisis posterior.

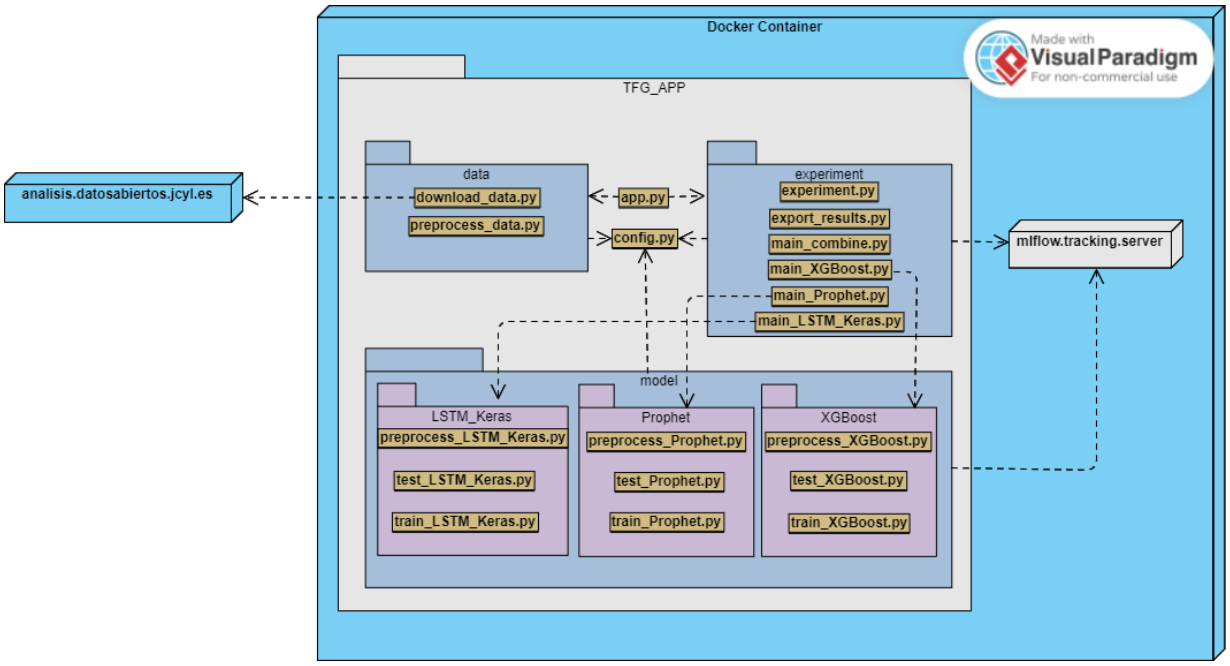


Figure 8.1: Diagrama UML de TFG_APP

4. **Experimentación (experiment)**:
   - experiment.py: Funciona como el coordinador principal de la experimentación. Dirige el ciclo de vida de los experimentos, incluida la variación de hiperparámetros y la gestión de múltiples runs. experiment.py llama a las siguientes clases del paquete experiment:
     - main_LSTM_Keras.py: Gestiona las llamadas de preprocesamiento, entrenamiento y test del modelo LSTM.
     - main_Prophet.py: Gestiona las llamadas de preprocesamiento, entrenamiento y test del modelo Prophet.
     - main_XGBoost.py: Gestiona las llamadas de preprocesamiento, entrenamiento y test del modelo XGBoost.
     - main_combine.py: Coordina la combinación y comparación de resultados de los diferentes modelos entrenados buscando entre los distintos runs del Tracking Server.
     - main_combine.py: Coordina la combinación y comparación de resultados de los diferentes modelos entrenados buscando entre los distintos runs del Tracking Server.
     - export_results.py: Accede al Tracking server de MLflow y exporta los resultados a un csv para poder realizar un posterior análisis.
5. **Modelos (model)**:
   - El paquete model encapsula las implementaciones específicas de los modelos utilizados en la aplicación:
     - LSTM_Keras, Prophet, XGBoost: Cada uno de estos paquetes representa un tipo distinto de modelo utilizado para tareas específicas de predicción.
     - Cada paquete de modelo contiene las siguientes clases:
       * preprocess<nombredelpaquete>: Funciones dedicadas al preprocesamiento específico necesario para cada tipo de modelo.
       * train<nombredelpaquete>: Funciones encargadas del entrenamiento del modelo utilizando los datos preprocesados.

  - test<nombredelpaquete>: Funciones para evaluar y probar la precisión y el rendimiento del modelo entrenado.

8.2 Contenedor Servidor MlFlow

- **MLflow**: Actúa como el servidor de seguimiento que centraliza el registro de experimentos, métricas y artefactos generados por la aplicación facilitando la comparación de modelos y la reproducción de resultados experimentales.

8.3 Justificación

Las ventajas de esta arquitectura son diversas. La utilización de contenedores de aplicación con Docker ofrece portabilidad, facilitando la implementación consistente de la aplicación en diferentes entornos y reduciendo problemas de dependencias. El uso de MLflow como servidor de seguimiento proporciona una gestión robusta de experimentos y modelos, permitiendo registrar métricas, artefactos y metadatos asociados a cada ejecución.

Todo esto facilita la colaboración, la reproducibilidad y el seguimiento del progreso del desarrollo de modelos, esencial en entornos de investigación y desarrollo de aprendizaje automático. Además, el enfoque modular de la aplicación permite una organización clara y mantenible del código, promoviendo la reutilización de componentes y la escalabilidad del sistema conforme se agregan nuevos modelos o experimentos.

Chapter 9

Implementación de modelos

Tras definir la metodología, el diseño de la aplicación y cómo esta puede ponerse a prueba es necesario detallar cómo se ha implementado el código para cada modelo pasando desde la selección de librerías hasta el preprocesamiento, entrenamiento y validación de las predicciones. Cada uno de estos modelos constituye un paquete de Python distinto dentro del paquete 'modelos' de la aplicación final.

9.1 LSTM Multistep

El primer modelo que se decidió codificar fue una red LSTM como la descrita en 6.3 basada en el trabajo de [22]. La estructura implementada es la que se muestra en6.3 con la peculiaridad de que no se cuenta con ninguna variable exógena si no que la entrada del modelo son q valores de la serie $(Y(t-q), ..., Y(t)$ en la imagen) y la salida es un vector con n predicciones a futuro ($\hat{Y}_1(t+n)$ en la imagen ya que solo hay una variable de salida a predecir).

9.1.1 Preprocesamiento

El primer paso de la implementación consistió por tanto en la preparación de los datos de la serie temporal para adaptarlos a la entrada de la red. Este proprocesamiento se recoge en distintas funciones dentro del archivo 'preprocess LSTM Keras.py'

Este módulo proporciona una serie de funciones para preparar datos de series temporales con el objetivo de introducirlos en una red LSTM multistep, es decir, con el objetivo de crear instancias de tamaño 'input length' con observaciones consecutivas previas que se utilizarán para entrenar y que se validarán con las 'output length' instancias posteriores. La primera función que se llama es 'prepare data', la cual se encarga de orquestar todo el proceso de preprocesamiento de los datos. A continuación se describe detalladamente cada función y su propósito dentro del flujo de preprocesamiento.

prepare data()

Esta es la función principal que coordina todo el preprocesamiento. Llama a las funciones posteriores para agregar características temporales, dividir los datos en conjuntos de entrenamiento, validación y prueba, crear conjuntos de datos supervisados y escalar los datos.

Parámetros

- data (pd.DataFrame): DataFrame con los datos originales.
- prop test (float): Proporción de datos para prueba.
- input length (int): Longitud de la ventana de entrada.
- output length (int): Longitud de la ventana de salida.

Retorno Conjuntos de datos de entrada y salida escalados para entrenamiento, validación y prueba, y el escalador para la columna de referencia.

add features()

Esta función agrega características temporales derivadas de la columna de fecha a un DataFrame. Estas características incluyen el día de la semana, el trimestre, el mes, el día del año, el día del mes y la semana del año. Además, la función crea variables dummy para cada una de estas características temporales y elimina la columna de fecha original.

Parámetros `df` (pd.DataFrame): DataFrame que contiene los datos, incluyendo una columna de fecha.

Retorno DataFrame con las nuevas características temporales agregadas y la columna de fecha eliminada.

train val test split()

Esta función divide un DataFrame en conjuntos de entrenamiento, validación y prueba basándose en los tamaños proporcionados.

Parámetros

- dataframe (pd.DataFrame): DataFrame a ser dividido.
- tr size (float): Proporción de datos de entrenamiento.
- vl size (float): Proporción de datos de validación.
- ts size (float): Proporción de datos de prueba.

Retorno Tres DataFrames correspondientes a los conjuntos de entrenamiento, validación y prueba.

create supervised dataset()

Genera conjuntos de datos supervisados a partir de una serie temporal. La función toma segmentos de datos de entrada y salida para entrenar el modelo LSTM.

Parámetros

- data (pd.DataFrame): DataFrame original con los datos.
- array (np.ndarray): Numpy array con los datos a ser procesados.
- prev (np.ndarray): Datos previos necesarios para concatenar.
- input length (int): Longitud de la ventana de entrada.
- output length (int): Longitud de la ventana de salida.
- ref col (str): Columna de referencia para la salida.

Retorno Arrays X (entrada) e Y (salida) para el modelo.

scale dataset()

Esta función escala los conjuntos de datos de entrenamiento, validación y prueba utilizando MinMaxScaler. El escalado se realiza de manera que las características de entrada y salida estén dentro del rango [-1, 1].

Parámetros

- data (pd.DataFrame): DataFrame original con los datos.
- data input (dict): Diccionario con los datos de entrada para entrenamiento, validación y prueba.
- col ref (str): Columna de referencia para la salida.
- x tr, y tr, x vl, y vl, x ts, y ts (np.ndarray): Arrays con los datos de entrada y salida para entrenamiento, validación y prueba.

Retorno Diccionario con los conjuntos de datos escalados y el escalador para la columna de referencia.

9.1.2 Entrenamiento

Una vez preparados los datos de tal forma que se cuenta con ventanas de entrada y salida para entrenamiento validación y prueba se procedió a la creación del modelo LSTM en sí mismo y al entrenamiento con la librería Keras. Este código se recoge en el archivo 'train LSTM Keras.py' y a continuación se detalla la función implementada, explicando su propósito y funcionamiento.

train model()

Entrena un modelo LSTM utilizando los datos de entrenamiento y validación escalados. La función configura el modelo, compila el optimizador y la función de pérdida, y entrena el modelo con EarlyStopping para evitar el sobreajuste.

Parámetros

- x_tr (np.ndarray): Datos de entrada escalados para el entrenamiento.
- y_tr (np.ndarray): Datos de salida escalados para el entrenamiento.
- x_vl (np.ndarray): Datos de entrada escalados para la validación.
- y_vl (np.ndarray): Datos de salida escalados para la validación.
- params (dict): Diccionario que contiene los parámetros del modelo, incluyendo:
- n_units (int): Número de unidades LSTM en cada capa.
- layers (int): Número de capas LSTM.
- output_length (int): Longitud de la salida del modelo.

Retorno El modelo LSTM entrenado.

Explicación Se emplea la API secuencial de Keras para crear el modelo con un código similar al que se muestra en el Listado 9.1, en el que se inicia la construcción del modelo secuencial añadiendo la primera capa LSTM con la forma de entrada especificada y una capa de *dropout*, seguida de un bucle con el que se añaden el resto de capas especificadas para añadir finalmente una última capa de activación lineal que produce la predicción lineal del modelo.

Creación de la red LSTM

```
model=Sequential()
model.add(
    LSTM(params['n_units'], input_shape=INPUT_SHAPE,
    ↪  return_sequences=(params["layers"]-1))
)
model.add(Dropout(DROPOUT_RATE))
for i in range(1, params['layers']):
    return_sequences=i + (params['layers']-1)
    model.add(LSTM(params['n_units'], return_sequences=return_sequences))
    model.add(Dropout(DROPOUT_RATE))
    model.add(Dense(params['output_length'], activation='linear'))
```

Código 9.1: Creación de red LSTM *multistep*, empleando la API secuencial de Keras.

Como detalles a destacar en la implementación del modelo es necesario aclarar que el optimizador empleado es RMSprop con una tasa de aprendizaje de 0.0005, se emplea *dropout*, es decir que cierto porcentaje de neuronas no se entrenan en cada iteración, para prevenir el sobre-ajuste, y se configura un *EarlyStopping* para monitorizar la pérdida de validación (val_loss) y detener el entrenamiento si

no hay mejora después de 20 épocas. El entrenamiento se realiza durante 150 *epochs* con un tamaño de *batch* de 32 Y en él se emplea como criterio de pérdida el ya mencionado RMSE.

Además de todo esto se guarda en MLflow una imagen con la evolución del error de entenamiento y validación durante los distintos epochs.

9.1.3 Test

Por último, tras entrenar la red se lleva a cabo la evaluación de esta con los subconjuntos preprocesados para obtener una estimación de los errores de entrenamiento validación y prueba. Este código se recopiló en el archivo 'test LSTM Keras.py' constando de dos funciones.

test_model()

Esta función evalúa el RMSE del modelo LSTM en los conjuntos de datos de entrenamiento, validación y prueba haciendo uso de la función predict() y genera gráficos de las predicciones en comparación con los valores reales que se guardan en MLflow

Parámetros

- model: El modelo LSTM previamente entrenado
- x tr, y tr, x vl, y vl, x ts, y ts: Conjuntos de datos de entrenamiento, validación y prueba escalados (entradas y salidas).
- output length: Longitud de la salida en cada predicción

Retorno Predicciones sin normalizar.

Explicación La evaluación de los modelos como se puede observar en el Listado 9.2 consiste simplemente en aplicar la función evaluate() que ofrece Keras para sus modelos

Evaluación de red LSTM

```
rmse_tr = model.evaluate(x=x_tr_s, y=y_tr_s, verbose=0)
rmse_vl = model.evaluate(x=x_vl_s, y=y_vl_s, verbose=0)
rmse_ts = model.evaluate(x=x_ts_s, y=y_ts_s, verbose=0)
```

Código 9.2: Evaluación de modelos LSTM.

predict()

Esta función calcula la predicción haciendo uso de la función y la desescala para devolver los valores en la escala original.

Parámetros

- x: Los datos de entrada escalados.
- model: El modelo LSTM previamente entrenado.
- scaler: El objeto *scaler* usado para escalar los datos durante el preprocesamiento
- output_length: Longitud de la salida en cada predicción.

Retorno Predicciones sin normalizar.

9.2 XGBoosting

El segundo modelo implementado es un modelo de regresión creado con la librería XGBoost que ofrece una implementación avanzada del algoritmo de boosting, específicamente del *Gradient Boosting Machine*. *XGBoost* es conocido por su velocidad flexibilidad y rendimiento superior en tareas de regresión y clasificación.

XGBoost se basa en la creación de un conjunto de árboles de decisión que se entrenan secuencialmente. Cada árbol intenta corregir los errores del árbol anterior, lo que resulta en un modelo que mejora iterativamente siguiendo este proceso [1]:

- El modelo comienza con una predicción inicial, que suele ser el valor promedio de la variable objetivo en el conjunto de entrenamiento.
- Se entrena el primer árbol de decisión utilizando los datos de entrenamiento (X_{train} e y_{train}). Este árbol intentará capturar las relaciones más fuertes entre las características temporales y la variable objetivo.
- Después de que el primer árbol haga sus predicciones, se calculan los residuos (diferencias entre las predicciones del árbol y los valores reales)
- Se entrena un nuevo árbol para predecir los residuos del árbol anterior. Este nuevo árbol se ajusta para corregir los errores del árbol anterior.
- El proceso se repite durante muchas iteraciones, con cada nuevo árbol tratando de corregir los errores de todos los árboles anteriores de forma que las predicciones finales se obtienen combinando las predicciones de todos los árboles, ajustadas por una tasa de aprendizaje que controla cuánto contribuye cada árbol al modelo final.

9.2.1 Preprocesamiento

El preprocesamiento que se lleva a cabo se recoge en el archivo 'preprocess XGBoost' transforma los datos de una serie temporal en un conjunto de características enriquecidas que pueden ser utilizadas para entrenar un modelo de XGBoost. La idea es agregar características temporales derivadas de la fecha, lo que permite que el modelo de XGBoost capture patrones temporales más complejos, y dividir los datos en conjuntos de entrenamiento, validación y prueba. Las funciones empleadas se detallan a continuación.

prepare data()

Esta función prepara los datos para el entrenamiento y evaluación de modelos de series temporales. Divide el DataFrame en conjuntos de entrenamiento, validación y prueba, y llama a 'add features()' para añadir características temporales a partir de la fecha.

Parámetros

- data (pd.DataFrame): DataFrame que contiene las series temporales, con una columna de fecha y una columna de respuesta.
- name response (str): Nombre de la columna que contiene la variable de respuesta.
- name date (str): Nombre de la columna que contiene las fechas.

Retorno Características X_train, X_val, X_test y valores de la serie y_train, y_val, y_test.

add_features()

Esta función se encarga de añadir características adicionales a un DataFrame basándose en la fecha que se encuentra en el índice del DataFrame. Las características añadidas incluyen el día de la semana, trimestre, mes, día del año, día del mes y semana del año.

Parámetros

- df (pd.DataFrame): DataFrame original que contiene las series temporales, con un índice basado en fechas.

Retorno DataFrame con las nuevas características añadidas.

9.2.2 Entrenamiento

Una vez obtenidas las características y preparados los datos se lleva a cabo la creación del modelo especificando los distintos parámetros. Este proceso se ha llevado a cabo con una única función.

train_model()

Esta función entrena un modelo de regresión XGBoost utilizando los datos de entrenamiento y prueba proporcionados.

Parámetros

- X_train (pd.DataFrame): Conjunto de datos de características de entrenamiento.
- X_test (pd.DataFrame): Conjunto de datos de características de prueba.
- y_train (pd.DataFrame): Variable de respuesta del conjunto de entrenamiento.

- *y_test* (pd.DataFrame): Variable de respuesta del conjunto de prueba.
- *max_depth* (int): Profundidad máxima de los árboles individuales en el modelo.
- *lamb* (float): Parámetro de regularización L2 (lambda) para prevenir el sobreajuste.
- *n_estimator* (int): Número de árboles (estimadores) en el modelo.

Retorno Modelo XGBoost entrenado.

Explicación El modelo se crea con la función XGBRegresor() y se entrena con la función fit() de la librería XGBoost como puede observarse en el Listado 9.3.

Creación del modelo XGBoost

```
reg = xgb.XGBRegressor(base_score=0.5,
        booster='gbtree',
        n_estimator=n_estimator,
        early_stopping_rounds=50,
        objective='reg:squarederror',
        max_depth=max_depth,
        learning_rate=0.1,
        reg_lambda=lamb,
        eval_metric='rmse')
reg.fit(X_train, y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=0)
```

Código 9.3: Creación de modelo XGBoost.

Es importante entender los distintos parámetros que se establecen en la función XGBRegressor:

- base_score: Valor inicial de predicción.
- _booster: Tipo de booster a usar, en este caso, árboles de decisión (gbtree).
- n_estimators: Número de árboles.
- early_stopping_rounds: Número de rondas sin mejora antes de detener el entrenamiento temprano.
- objective: Función de pérdida a optimizar, aquí se usa el error cuadrático medio (reg:squarederror).
- max_depth: Profundidad máxima de los árboles.
- learning rate: Tasa de aprendizaje.
- reg_lambda: Parámetro de regularización L2 utilizado para reducir el sobreajuste.
- eval_metric: Métrica de evaluación, el ya explicado 'rmse'.

Además cabe destacar que la importancia de las distintas variables se cuantifica y se representa gráficamente, de forma que queda registrada para cada modelo en MLflow.

9.2.3 Test

Una vez entrenado el modelo es necesario realizar igualmente una evaluación de su rendimiento. Este código pertenece al archivo test XGBoost.py y consta de una única función.

test model()

La función test model() se utiliza para evaluar el rendimiento del modelo entrenado en conjuntos de datos de entrenamiento, validación y prueba. Además, genera gráficos de las predicciones y calcula las métricas de error cuadrático medio (RMSE).

Parámetros

- model: El modelo XGBoost previamente entrenado
- x train, y train, x val, y val, x test, y test: Conjuntos de datos de entrenamiento, validación y prueba con las características y valores respuesta.
- data (pd.DataFrame): DataFrame original que contiene todas las observaciones y características, incluyendo las fechas.

Explicación Al calcular las predicciones y hacer el test se tiene en cuenta como peculiaridad que a los días detectados como *outliers* en el preprocesamiento inicial de este documento no se les asigna la predicción obtenida con el modelo, si no que se predicen como el valor medio de ese día en años anteriores. De este modo la predicción de la parte regular de la serie no se ve influida por los *outliers* pero estos no obtienen predicciones alejadas de sus valores típicos.

La predicción se realiza fácilmente con la función *evaluate()* de los modelos XGBoost como se observa en el Listado 9.4 de forma que las métricas RMSE se calculan de forma directa con éstas y los valores reales de la serie. En este caso se decidió guardar el RMSE de las predicciones, tanto para la serie filtrada como para la serie original con el ajuste para *outliers* comentado.

Validación del modelo XGBoost

```
rmse_tr = model.evaluate(x=x_tr , y=y_tr, verbose=0)
rmse_vl = model.evaluate(x=x_vl_s, y=y_vl, verbose=0)
rmse_ts = model.evaluate(x=x_ts_s, y=y_ts, verbose=0)
```

Código 9.4: Validación de modelo XGBoost

El valor de la serie en sus subconjuntos validación y test así como las predicciones de ambos se guardan en un gráfico para poder visualizar la calidad de las predicciones con MLflow.

9.3 Prophet

El último modelo implementado fue el modelo Prophet. Para ello al igual que en el resto de modelos se realizó un preprocesamiento, entrenamiento y validación.

9.3.1 Preprocesamiento

prepare data()

La función prepare data() se encarga de preparar un DataFrame para el modelo Prophet. Este preprocesamiento incluye la selección y renombrado de columnas Prophet,que requiere que las columnas de fecha y respuesta se llamen 'ds' e 'y' respectivamente, la conversión de fechas, y la división de los datos en conjuntos de entrenamiento, validación y prueba

Parámetros

- data(pd.DatFrame): Un DataFrame de pandas que contiene los datos originales, incluyendo las columnas de fecha y la variable de interés.
- prop test(float): Un flotante que indica la proporción de los datos que se reservarán para los conjuntos de validación y prueba.

Retorno Conjuntos de entrenamiento validación y prueba.

9.3.2 Entrenamiento

Para el entrenamiento del modelo Prophet se ha utilizado la librería Prophet que soporta la implementación del modelo.

train model()

La función train model() se utiliza para entrenar un modelo Prophet con los datos de entrenamiento proporcionados. En esta función, se configuran varios parámetros clave que influyen en la flexibilidad y precisión del modelo antes de ajustarlo a los datos.

Parámetros

- train_df (pd.DataFrame): Un DataFrame de pandas que contiene los datos de entrenamiento con dos columnas: 'ds' para las fechas y 'y' para los valores a predecir.
- params (dict): Un diccionario que contiene los parámetros de ajuste del modelo Prophet:
  - "cps": (*changepoint prior scale*), que controla la flexibilidad del modelo respecto a los cambios en la tendencia.
  - "sps": (*seasonality prior scale*), que controla la flexibilidad del modelo respecto a los patrones estacionales.

Retorno El modelo Prophet entrenado con los datos proporcionados.

Explicación El entrenamiento de los modelos como se puede observar en el Listado 9.5 consiste simplemente en utilizar la librería Prophet para crear un modelo Prophet con los parámetros deseados y utilizar la función fit para completar el entrenamiento con los datos adecuados.

Entrenamiento del modelo Prophet

Código 9.5: Entrenamiento del modelo Prophet.

```
model = Prophet(growth="linear", changepoint_prior_scale=params["cps"],
    seasonality_prior_scale=params["sps"], interval_width=0.9)
model.fit(train_df)
```

9.3.3 Validación

Como en el resto de ocasiones tras realizar el entrenamiento se lleva a cabo una validación de los resultados obtenidos para comprender el desempeño de estos modelos.

test model()

La función test_model se utiliza para evaluar el rendimiento del modelo entrenado en conjuntos de datos de entrenamiento, validación y prueba. Además, genera gráficos de las predicciones y calcula las métricas de error cuadrático medio (RMSE).

Parámetros

- model: El modelo Prophet previamente entrenado
- train df,val df,test df: Conjuntos de datos de entrenamiento, validación y prueba con las características y valores respuesta.
- data (pd.DataFrame): DataFrame originals.

Explicación Al igual que con el modelo de XGBoost al calcular las predicciones y hacer el test se tiene en cuenta como peculiaridad que a los días detectados como *outliers* en el pre-procesamiento inicial de este documento no se les asigna la predicción obtenida con el modelo, si no que se predicen como el valor medio de ese día en años anteriores. De este modo la predicción de la parte regular de la serie no se ve influida por los *outliers* pero éstos no obtienen predicciones alejadas de sus valores típicos.

La predicción se realiza fácilmente con la función predict() de los modelos XGBoost como se observa en el Listado 9.6 de forma que las métricas RMSE se calculan de forma directa con estas y los valores reales de la serie. De nuevo al igual que en XGBoost se decidió guardar el RMSE de las predicciones tanto para la serie filtrada como para la serie original con el ajuste para *outliers* comentado.

Validación del modelo XGBoost

```
forecast_train = model.predict(future_train)
forecast_val = model.predict(future_val)
forecast_test = model.predict(future_test)
```

Código 9.6: Validación del modelo XGBoost.

El valor de la serie en sus subconjuntos validación y test así como las predicciones de ambos se guardan en un gráfico para poder visualizar la calidad de las predicciones con MLflow.

9.4 XGBoosting + Prophet

Durante la experimentación, se observó que los modelos XGBoost y Prophet producían mejores resultados en comparación con los modelos LSTM. Por ello, se decidió combinar los resultados de XGBoost y Prophet con el objetivo de verificar si la combinación del valor medio de ambos resultados podría reducir aún más el error de predicción. Para ello, como puede observarse en el Listado 9.7 fue clave el uso de las funciones 'search runs()' y 'sort values()' de MLflow de forma que para cada zona se buscó los 'runs' de XGBoost y Prophet con menor RMSE y se combinó su valor para obtener las nuevas predicciones y registrarlas como el único run de un nuevo experimento.

Obtención de los mejores 'runs' para cada experimento

```
runs = mlflow.search_runs(experiment_ids=[experiment_id],
↪  filter_string=filter_string)
sorted_runs = runs.sort_values(by='metrics.rmse_val_reg', ascending=True)
best_run = sorted_runs.iloc[0]
run_id = best_run.run_id
pred_val = mlflow.tracking.MlflowClient().download_artifacts(run_id,
↪  "pred_val/pred_val.csv")
```

Código 9.7: Obtención de los mejores *runs* para cada experimento.

9.5 Hiperparámetros

Un aspecto de gran importancia en la implementación fue la selección de los distintos hiperparámetros. Para poder hacer una comparación eficaz de estos se crearon tres diccionarios (uno para cada modelo) con distintos valores de estos sobre cuyas entradas se iterava llamando con cada repetición a un nuevo proceso de entrenamiento y validación para testear dicha combinación. Sin embargo debido a la existencia de un enorme número de hiperparámetros entre los que elegir para los tres modelos se optó por seleccionar algunos para la exploración y fijar otros que se entendieron como menos influyentes o sensibles al cambio. Para cada modelo estos hiperparámetros fueron distintos.

9.5.1 LSTM

Para el modelo LSTM, se exploraron diversas configuraciones de estos parámetros:

- n_units: Número de unidades en cada capa LSTM. Con valores entre 1 y 15.
- layers: Número de capas LSTM. Con valores entre 1 y 10.
- lags: Número de desfases utilizados en la entrada. Con valores entre 5 y 10.
- output_length: Número de días a futuro a predecir. Con valores entre 10 y 40.

Se entiende que el número de capas y su tamaño pueden provocar sobre-ajuste si el modelo se hace demasiado complejo o infra-ajuste si se peca de demasiada simplicidad. Además es interesante variar el número de días pasados a tener en cuenta así como comprobar si al aumentar el número de días de predicción a futuro los resultados empeoran rápidamente.

En cambio los siguientes parámetros se fijaron para no disparar el número de combinaciones posibles:

- Learning Rate: 5e-4
- Dropout Rate: 0.2
- Epochs: 150
- Tamaño de batch: 32

9.5.2 XGBoost

En el caso de XGBoost el número de parámetros es muy elevado por lo que para poder hacer pruebas con todas las zonas se decidió iterar únicamente sobre estos 3:

- max_depth: Profundidad máxima de los árboles empleados en el proceso de Boosting. Con valores entre 3 y 15.
- lambda: parámetro de regularización L2. Con valores 0, 0.5 y 1.
- n_estimators: Número de árboles utilizados en el modelo.

Iterar sobre lambda permite encontrar un punto de regularización en el que se evite el sobre-ajuste del mismo modo que controlar el número y profundidad de los árboles hace posible encontrar un nivel de complejidad adecuado para el modelo.

En este caso los parámetros fijados fueron:

- base_score: Este parámetro establece el valor inicial de predicción de todos los ejemplos antes de que el modelo comience a entrenar. Fue fijado en 0.5.
- booster: Este parámetro especifica el tipo de modelo de boosting que se utilizará. Fue fijado en'gbtree' ya que se pretende utilizar árboles.
- early_stopping_rounds: Este parámetro permite detener el entrenamiento antes de tiempo si el modelo no muestra mejoras en un número determinado de rondas consecutivas. En este caso se fijó en 50.
- objective: Este parámetro define la función objetivo que el modelo trata de minimizar que se fijó como 'reg:squarederror' (MSE).
- learning_rate: Controla la contribución de cada árbol en el modelo. En este caso se fijó como 0.1.

9.5.3 Prophet

En Prophet el número de parámetros disponibles es notablemente menor. Para iterar se escogieron los dos parámetros más determinantes que ya fueron descritos con anterioridad:

- *Changepoint prior scale*: Un valor más alto hace que el modelo sea más flexible y detecte más puntos de cambio, mientras que un valor más bajo hace que el modelo sea más conservador. Con valores entre 0.001, 0.01, 0.1 y 1.
- *Seasonality prior scale*: Un valor más alto permite que el modelo se ajuste mejor a la estacionalidad observada en los datos, mientras que un valor más bajo lo hace menos sensible a estas variaciones estacionales. Con valores entre 0.001, 0.01, 0.1 y 1.

Por otro lado se fijaron dos parámetros:

- Interval_width: Define el ancho del intervalo de confianza alrededor de las predicciones del modelo. En este caso se fija en 0.9, lo que indica que se está utilizando un intervalo de confianza del 90%.
- Growth: Este parámetro especifica el tipo de crecimiento del modelo. En este caso, growth = ”linear” indica que se está utilizando un modelo de crecimiento lineal para las tendencias en la serie temporal.

Part IV

Resultados y Conclusiones

Chapter 10

Resultados

Una vez implementado todo el código y puesta en marcha la aplicación, se generaron todos los resultados, que fue necesario analizar para sacar las conclusiones de qué modelo ofrece mejores resultados y si estos son aceptables.

Los resultados generados por TFG_APP se guardan en el archivo 'Exp_results.csv' en el directorio `data` y éste se analiza con el Jupyter Notebook 'Process_Experiment_Results.ipnyb'. Este análisis comienza con la lectura y transformación del csv a un DataFrame de Pandas y seguidamente se estudia por separado los resultados de los distintos modelos. Además el estudio se complementa con los distintos gráficos que se van guardando en MLFlow a medida que se entrenan los modelos y que permiten comprobar gráficamente el ajuste de la predicción a los datos reales.

10.1 Descarte de LSTM

Pronto en las primeras etapas de experimentación se hizo evidente que el modelo LSTM era una opción que debía descartarse y que no ofrecía unas predicciones tan ajustadas como las que se observan al aplicar XGBoost y Prophet. Las razones para descartar este modelo fueron las siguientes:

- **Tiempo de cómputo:** Mientras que el tiempo de cómputo de los modelos XGBoost y Prophet rondan entre los 3 y los 6 segundos por 'run' en el caso de las redes LSTM esta duración asciende a los 40 o 50 segundos por iteración. Esto sumado al tiempo de preparación de datos y otros procesamientos conllevan un tiempo de espera mucho mayor para obtener los resultados.
- **Limitación en la ventana de predicción:**Como ya se ha descrito anteriormente las redes LSTM multistep tienen como uno de sus parámetros el tamaño de salida de la red. Sin embargo si este tamaño se hace demasiado grande la cantidad de agrupaciones para entrenamiento y validación se reduce drásticamente. Además la predicción queda limitada a cierto número de días en el futuro mientras que en XGBoost y Prophet la predicción puede realizarse simplemente aportando una fecha como entrada sin importar cómo de lejos en el futuro se encuentre. Este ha sido el principal factor para determinar el descarte de este enfoque ya que las predicciones a corto plazo en planificación no tienen el efecto práctico deseado.
- **Hiperparametrización:** Al problema de tiempo de cómputo se le añade sin duda el de la hiperparametrización. En total para poder hacer un buen 'tuning' de los distintos hiperparámetros

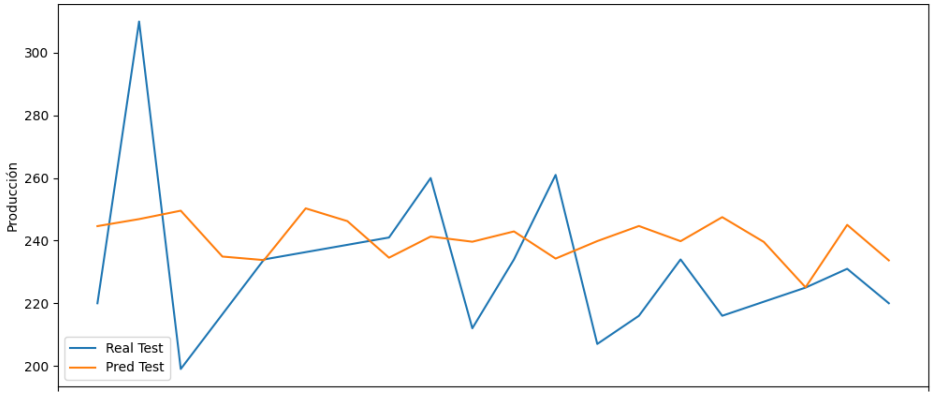


Figure 10.1: Valores reales y predicciones en Vitigudino

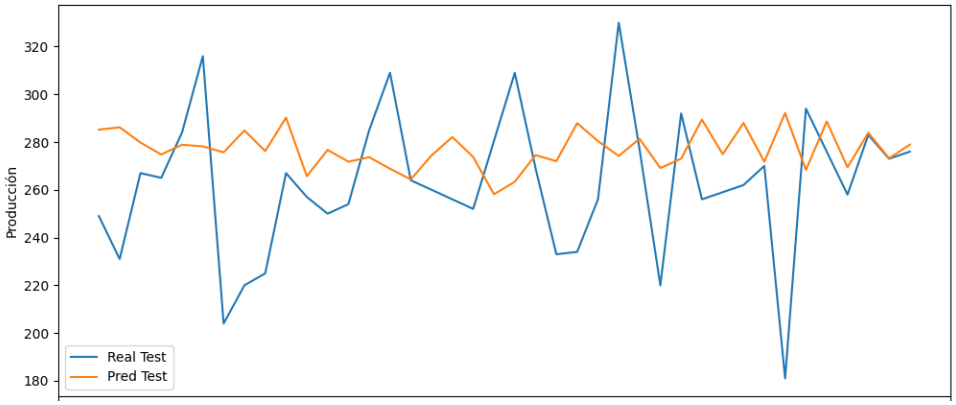


Figure 10.2: Valores reales y predicciones en Aranda Rural

ya comentados es necesario probar al menos combinaciones lo que eleva el tiempo de experimentación de forma extraordinaria, pudiendo tardar cada zona en entrenarse casi media hora.

Todo esto podría haber sido obviado si los resultados ofrecidos fueran excelentes. Sin embargo como se puede ver en ciertos ejemplos como el de Aranda Rural10.2 o Vitigudino 10.17 el ajuste no parecer ser extraordinario y no parece mejorar los resultados que se mostrarán a continuación y que ofrecen los otros modelos, mucho más rápidos.

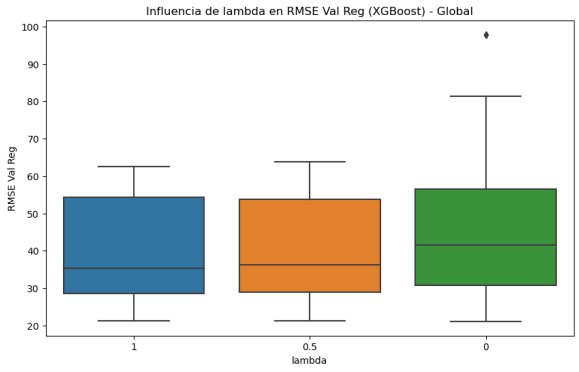
Por todas estas razones y para facilitar la selección de modelos de aquí en adelante se hará referencia únicamente a los resultados del estudio para XGBoost, Prophet y su combinación. Además en el propio archivo de `docker-compose.yaml` se establece como desactivada la experimentación con LSTM que debe ser activada de forma manual como se describe en B.

10.2 Hiperparámetros

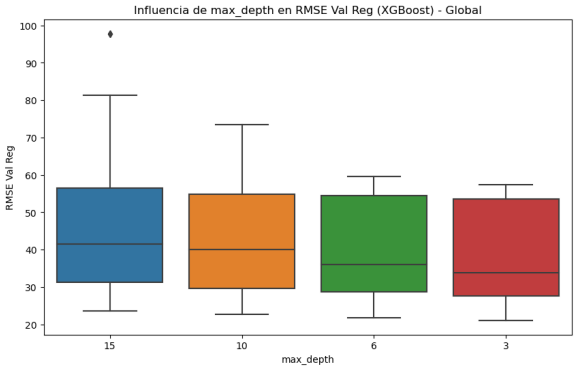
La primera parte del estudio consiste en explorar el efecto de los hiperparámetros seleccionados en el error cometido por los modelos evaluando este aspecto tanto de forma global como de forma local para cada clúster. Estas comparaciones se hacen en todo momento con el RMSE de validación para la serie filtrada ya que lo interesante es determinar la capacidad de los modelos de explicar la variabilidad regular de las series.

10.2.1 XGBoost

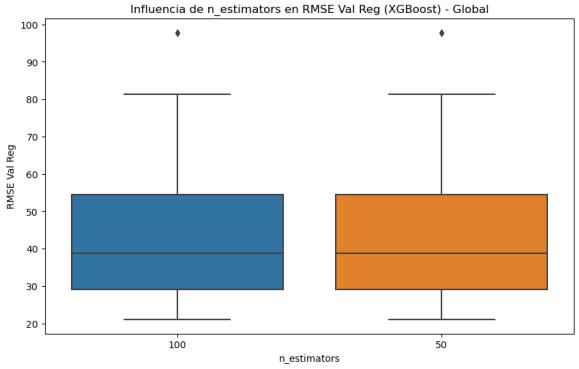
En el caso de los modelos XGBoost, poniendo atención únicamente en la visión global parece que los hiperparámetros tienen muy poco efecto sobre los resultados obtenidos para el conjunto de todas las zonas como se observa en el diagrama de cajas 10.3 donde parece claro que el error no varía con n_estimators entre 50 y 100, que lambda igual a 0 conduce a resultados algo peores y que max_depth mayores o iguales que 10 empeoran ligeramente la predicción.



(a) RMSE de la serie filtrada respecto a lambda



(b) RMSE de la serie filtrada respecto a max_depth

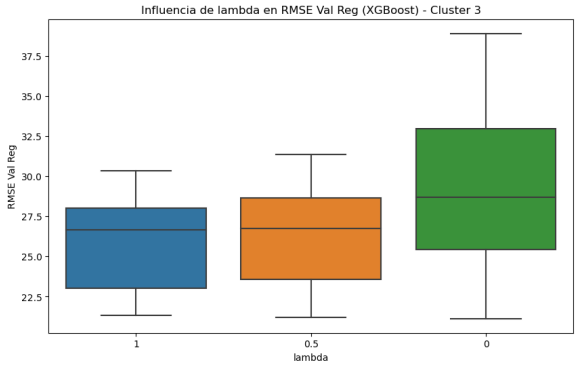


(c) RMSE de la serie filtrada respecto a n_estimators

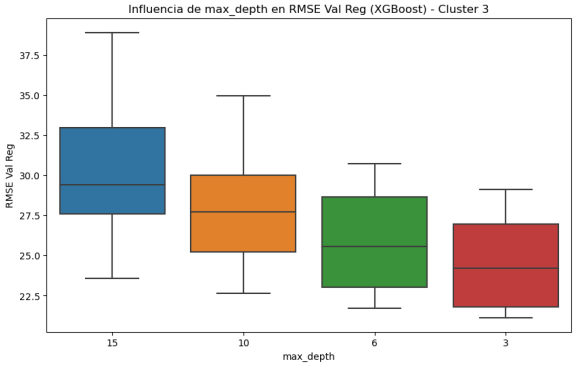
Figure 10.3: Efecto de los parámetros a nivel global en el error de la serie sin *outliers*

Sin embargo al disponer de la información de agrupamiento en clusters obtenida en el preprocesamiento 5 parece adecuado observar el efecto de los hiperparámetros en cada grupo.

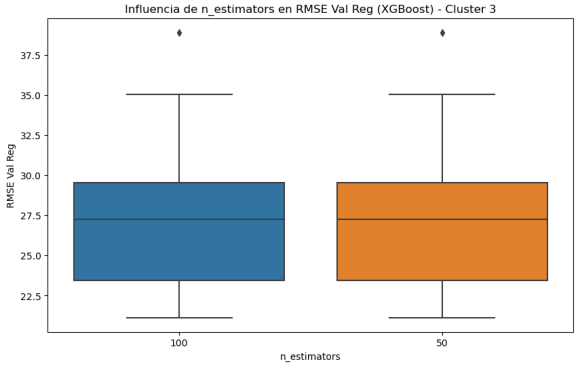
Atendiendo a 10.7, 10.6, 10.5, 10.4 se puede apreciar como los efectos globales mencionados anteriormente son moderados para los clusters 3 y 1, fuertes para el cluster 2 e increíblemente determinantes para el cluster 0.



(a) RMSE de la serie filtrada respecto a lambda

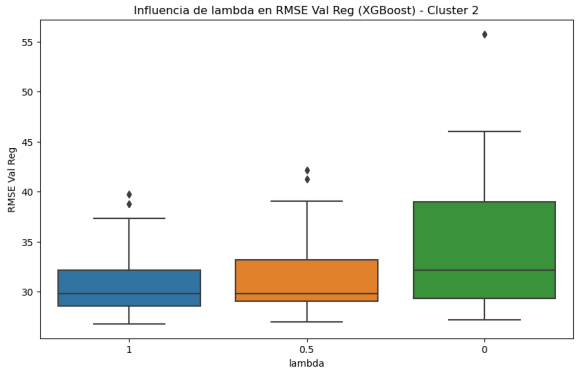


(b) RMSE de la serie filtrada respecto a max_depth

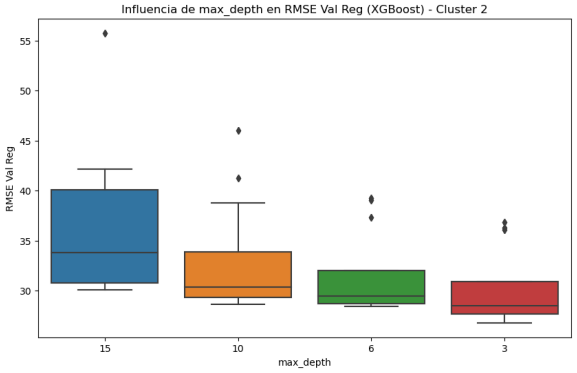


(c) RMSE de la serie filtrada respecto a n_estimators

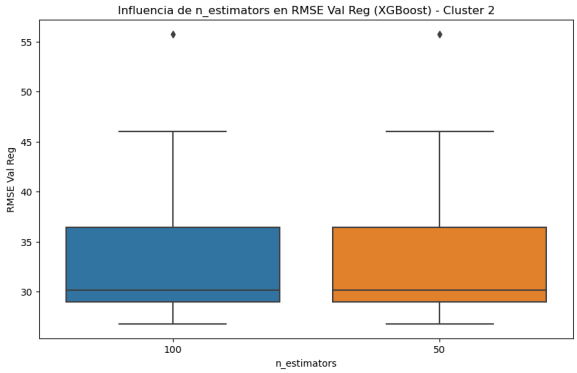
Figure 10.4: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 3



(a) RMSE de la serie filtrada respecto a lambda

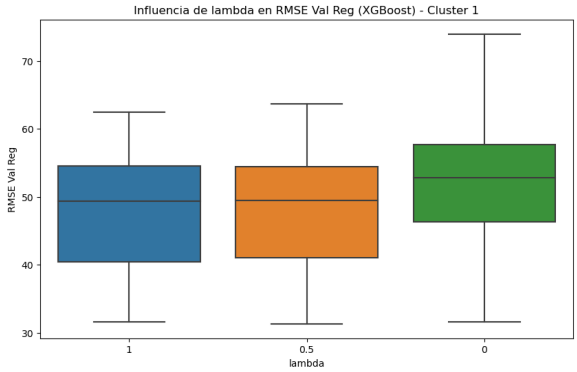


(b) RMSE de la serie filtrada respecto a max_depth

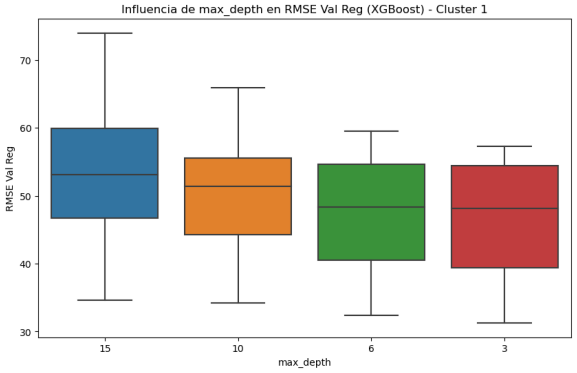


(c) RMSE de la serie filtrada respecto a n_estimators

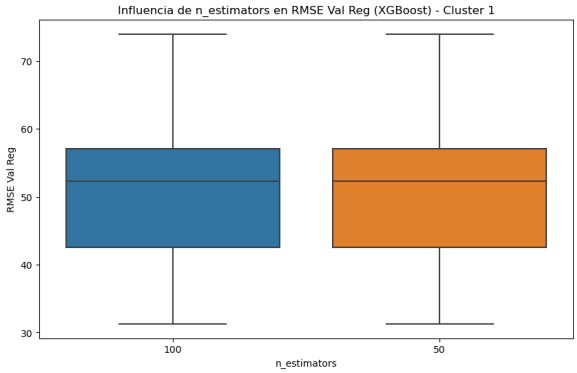
Figure 10.5: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 2



(a) RMSE de la serie filtrada respecto a lambda

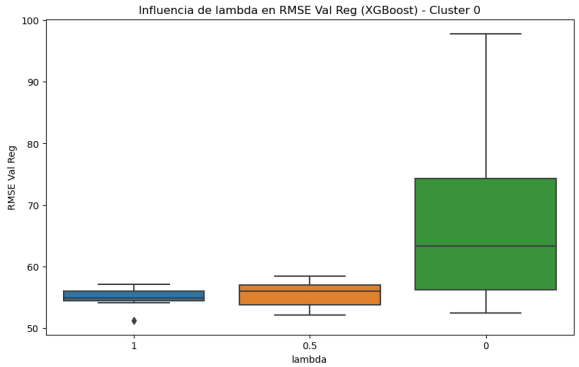


(b) RMSE de la serie filtrada respecto a max_depth

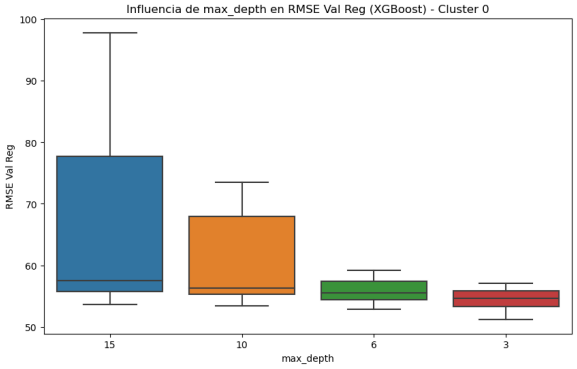


(c) RMSE de la serie filtrada respecto a n_estimators

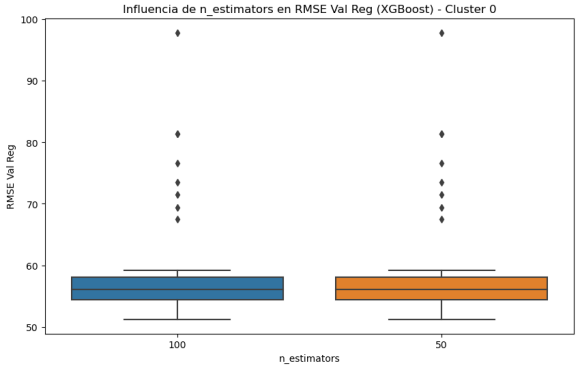
Figure 10.6: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 1



(a) RMSE de la serie filtrada respecto a lambda



(b) RMSE de la serie filtrada respecto a max_depth

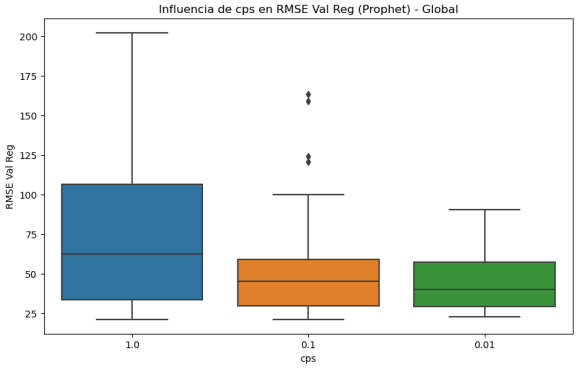


(c) RMSE de la serie filtrada respecto a n_estimators

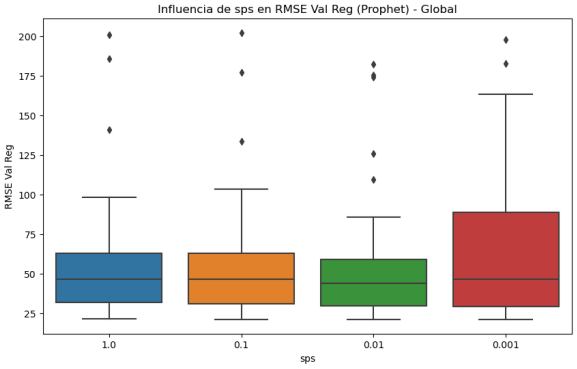
Figure 10.7: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 0

10.2.2 Prophet

Para el modelo Prophet se ha seguido un modelo de exploración muy similar al de XGBoost. La visión global que se muestra en el diagrama de cajas 10.8 en el que puede observarse que valores de cps cercanos a 1 y de sps cercanos a 0 producen mayor error en las predicciones.



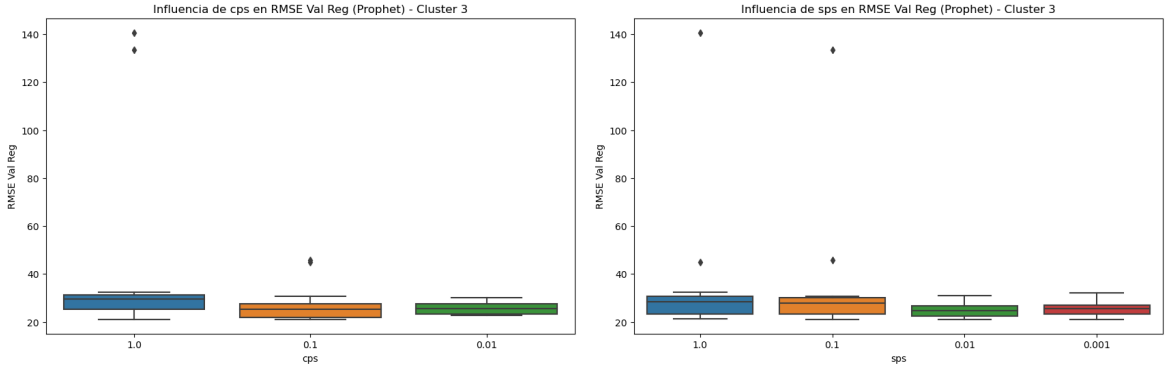
(a) RMSE de la serie filtrada respecto a max_depth



(b) RMSE de la serie filtrada respecto a n_estimators

Figure 10.8: Efecto global de los parámetros en el error de la serie sin *outliers*

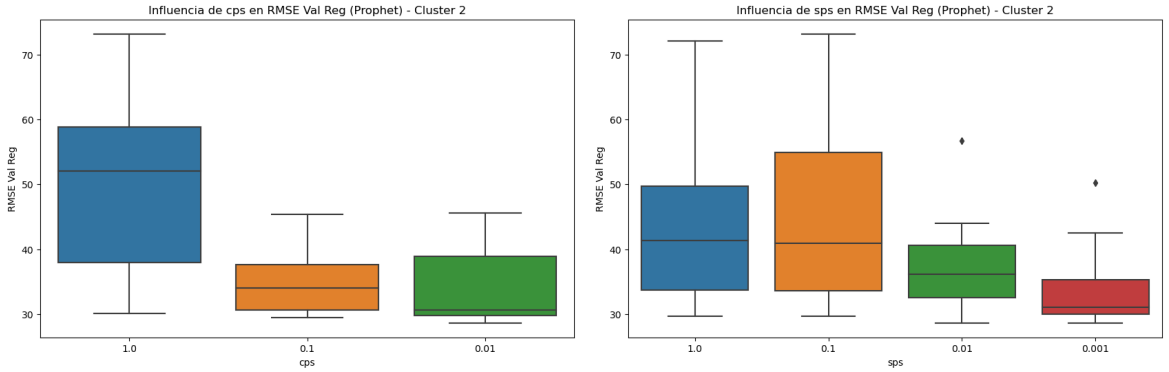
Si se separa de nuevo el análisis para explorar los distintos clusters de zonas se ven de nuevo diferencias interesantes que pueden observarse en 10.12, 10.11, 10.10 y 10.9. En estos gráficos se hace evidente que para el cluster 3 la proximidad de sps a 0 no empeora los resultados, que para el cluster 2 es la proximidad de sps a 1 la que empeora los resultados y que para el cluster 0 los efectos globales son, al igual que en XGBoost, mucho más evidentes.



(a) RMSE de la serie filtrada respecto a max_depth

(b) RMSE de la serie filtrada respecto a n_estimators

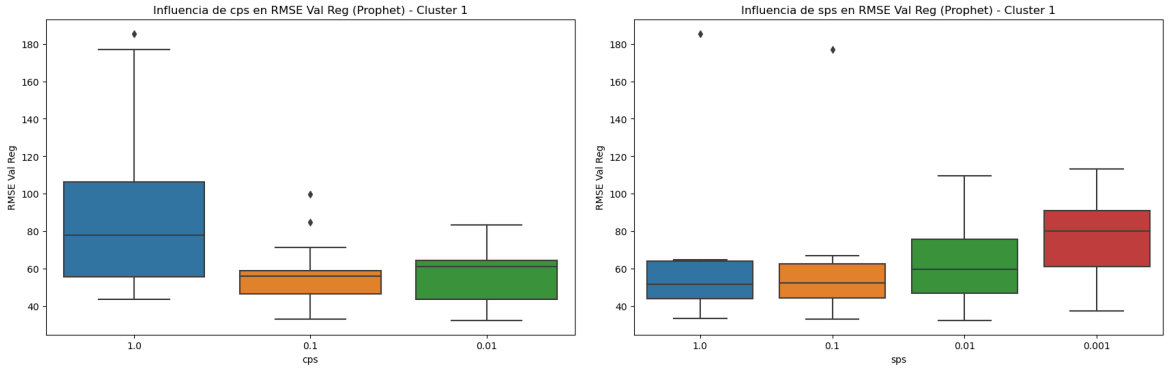
Figure 10.9: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 3



(a) RMSE de la serie filtrada respecto a cps

(b) RMSE de la serie filtrada respecto a sps

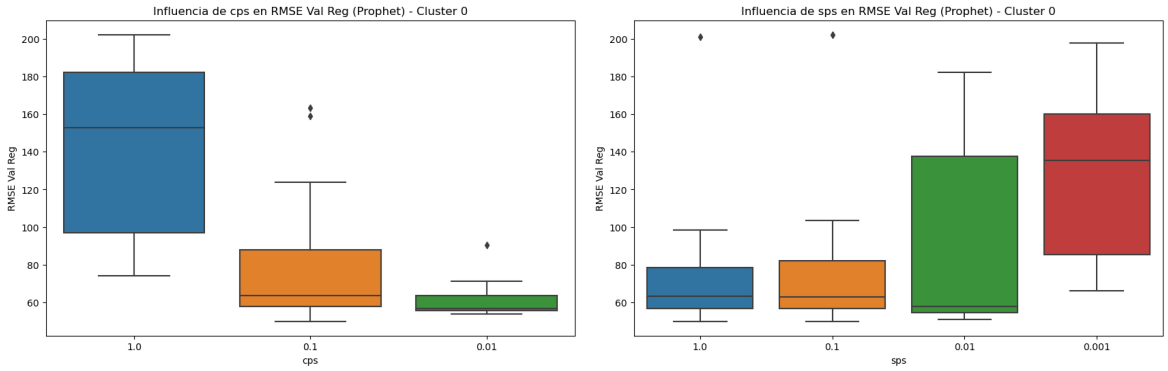
Figure 10.10: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 2



(a) RMSE de la serie filtrada respecto a cps

(b) RMSE de la serie filtrada respecto a sps

Figure 10.11: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 1



(a) RMSE de la serie filtrada respecto a cps (b) RMSE de la serie filtrada respecto a sps

Figure 10.12: Efecto de los parámetros en el error de la serie sin *outliers* para el cluster 0

10.3 Comparación entre modelos

Tras hacer una exploración de los hiperparámetros se consideró interesante tomar los mejores runs (combinaciones de hiperparámetros) tanto de XGBoost como de Prophet y la combinación entre ambas predicciones (valor medio del resultado de los mejores 'runs' de los dos modelos) para cada zona con el objetivo de tratar de identificar algún patrón que indicara un mejor desempeño de alguna de las opciones.

El resultado de esta comparación, que se puede observar en 10.13 y 10.14 muestra dos tendencias. El primer patrón notable es que de forma notablemente consistente entre los distintos conjuntos Prophet comete un mayor error de predicción (a excepción del caso de Ciudad Rodrigo), mientras que la opción de combinar ambos resultados parece ser la que produce un menor error en la mayoría de casos.

Otro aspecto importante a destacar es que no parece haber diferencias lo suficientemente grandes entre los RMSE de los distintos datasets que reflejen la diferencia real de afluencia que existe entre ellos, lo que indica que los modelos tienen un mayor rendimiento en las zonas con mayor afluencia que en las zonas menos concurridas.

10.4 Visualización y valoración de la predicción

Una vez comprendido el efecto de los hiperparámetros en el error de predicción y el desempeño de los distintos modelos es interesante observar gráficamente el resultado del ajuste en las distintas zonas. Para no saturar este documento se ha decidido tomar una zona de cada cluster y mostrar la predicción del modelo con menor RMSE de validación en la serie filtrada, tanto en la serie filtrada como en la serie original con outliers.

10.4.1 Parquesol - Cluster 0

Como sepuede observar en 10.15 el ajuste del mejor modelo, el modelo combinado, parece ser bastante acertado especialmente en el periodo de verano. Para cuantificar este ajuste se puede tomar el RMSE del conjunto de validación en la serie filtrada que es igual a 54.1

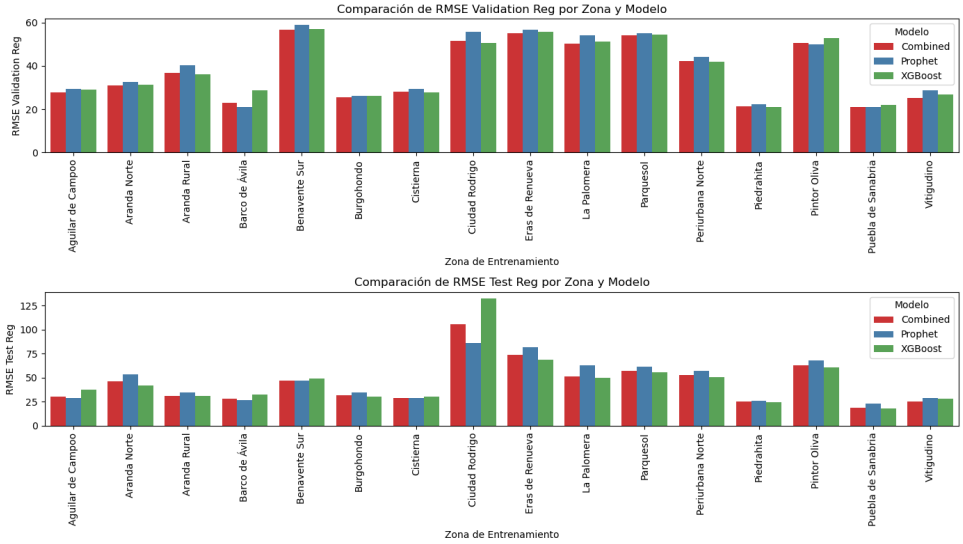


Figure 10.13: Comparación del RMSE obtenido por los mejores resultados de cada modelo en la predicción de la serie filtrada

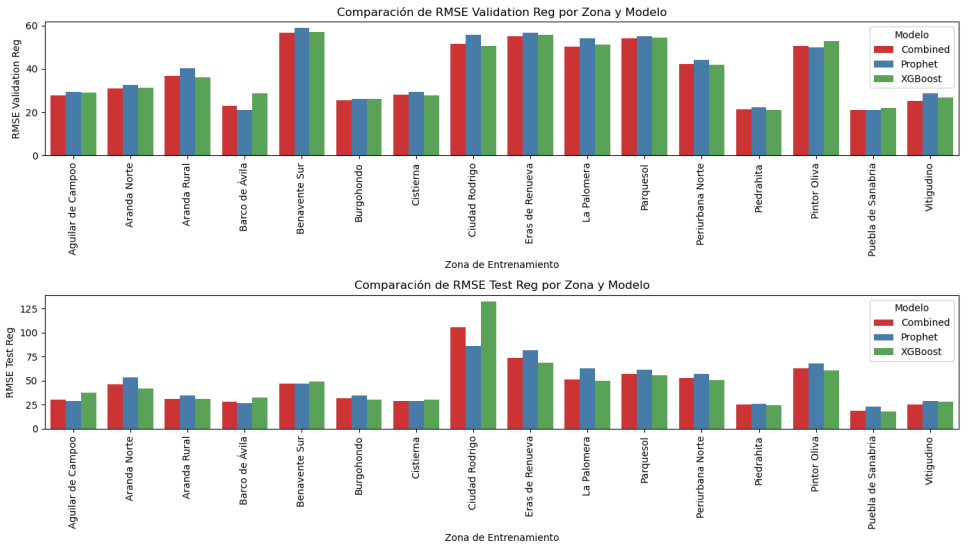
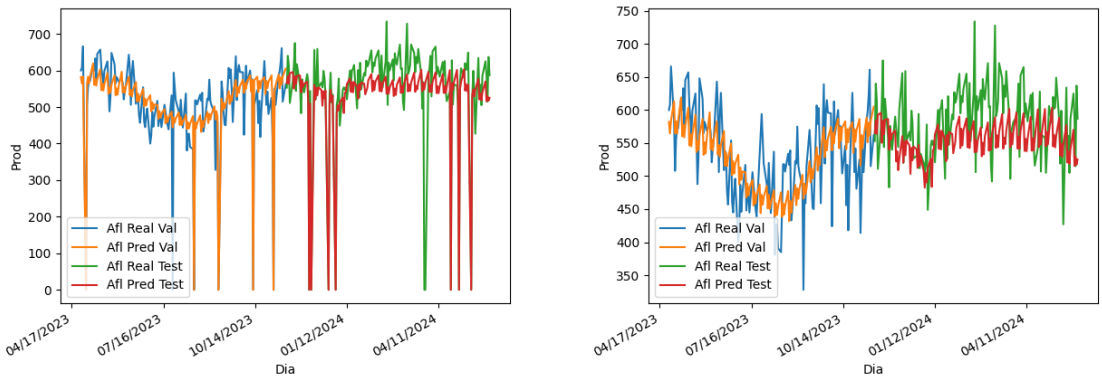


Figure 10.14: Comparación del RMSE obtenido por los mejores resultados de cada modelo en la predicción de la serie con *outliers*

Sin embargo el RMSE del conjunto test en la serie con outliers es 86.67 seguramente muy influenciado por el valor atípico que puede observarse. En todo caso el error esperado es razonable para una afluencia media superior a los 600 pacientes diarios.



(a) Predicción de afluencia en los conjuntos de validación y test para la serie original

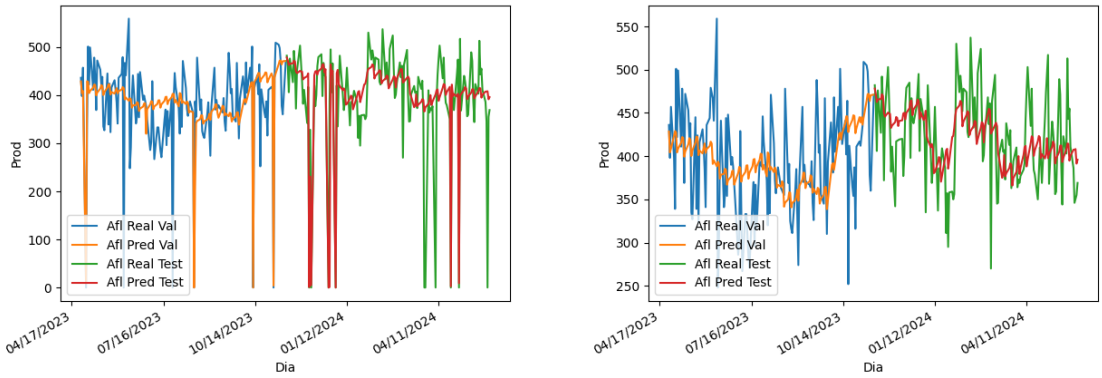
(b) Predicción de afluencia en los conjuntos de validación y test para la serie filtrada

Figure 10.15: Predicción para la serie de Parquesol (filtrada a la derecha y con outliers a la izquierda) con el modelo de combinación de XGBoost y Prophet (modelo de menor RMSE de validación en la serie filtrada)

10.4.2 Benavente Sur - Cluster 1

Atendiendo a la imagen 10.16 el ajuste del mejor modelo encontrado, de nuevo el modelo de combinación, parece ser bastante acertado en cuanto al valor medio de los datos en las distintas épocas del año aunque ciertamente no predice especialmente bien la variabilidad.

Para entender esta predicción en números basta con observar el RMSE del conjunto de validación de la serie filtrada que es igual a 56.78 y al RMSE del test de la serie original que es igual a 85.92. El error aumenta de nuevo al introducir los valores outliers que no han podido ser prevenidos con los datos anteriores de la serie. Los valores de error son muy similares a los de Parquesol pero el valor medio de afluencia baja hasta los 425 pacientes por día lo que implica un pero desempeño en esta zona.



(a) Predicción de afluencia en los conjuntos de validación y test para la serie original

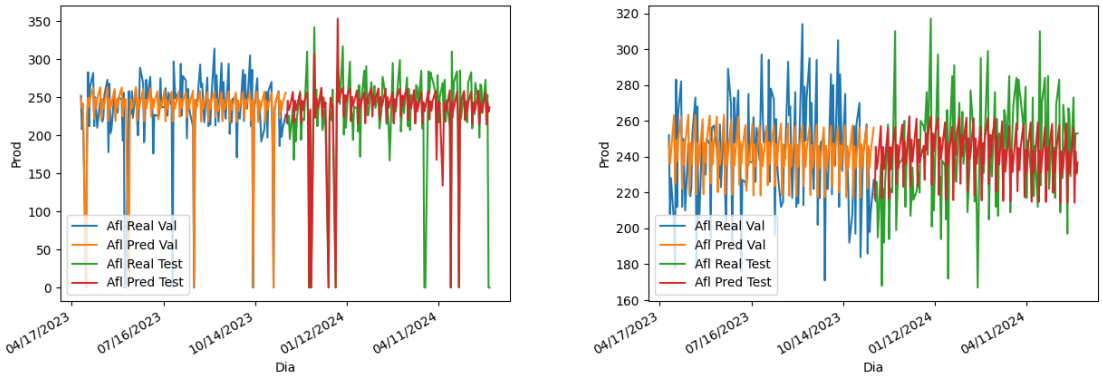
(b) Predicción de afluencia en los conjuntos de validación y test para la serie filtrada

Figure 10.16: Predicción para la serie de Benavente Sur (filtrada a la derecha y con outliers a la izquierda) con el modelo de combinación de XGBoost y Prophet (modelo de menor RMSE de validación en la serie filtrada)

10.4.3 Vitigudino - Cluster 2

La imagen 10.17 mustra el ajuste del mejor modelo encontrado, una vez más el modelo de combinación. En este caso los datos apenas parecen presentar un patrón por lo que el modelo ofrece una predicción sin tendencias.

Numéricamente es interesante de nuevo atender al RMSE del conjunto de validación de la serie filtrada que es igual a 25.12 y al RMSE del test de la serie original que es igual a 47.54. Los valores de error son mucho menores que los vistos con anterioridad pero lo són para una zona en la que la afluencia media ronda los 240 pacientes.



(a) Predicción de afluencia en los conjuntos de validación y test para la serie original

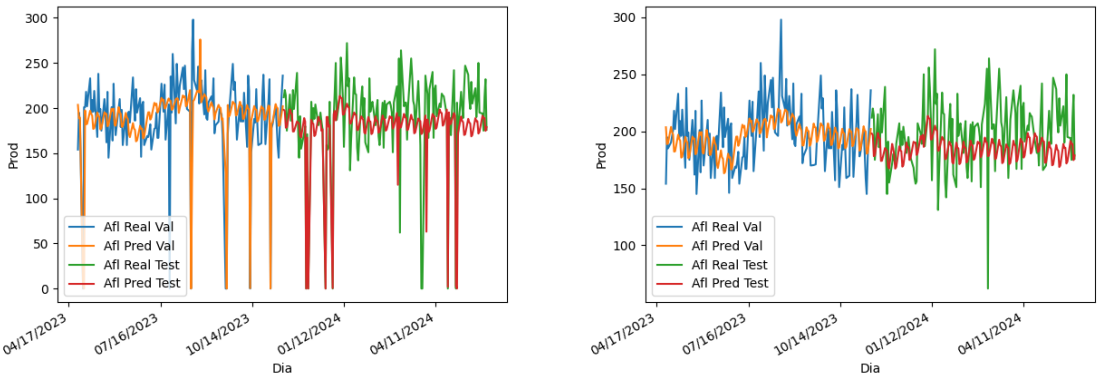
(b) Predicción de afluencia en los conjuntos de validación y test para la serie filtrada

Figure 10.17: Predicción para la serie de Vitigudino (filtrada a la derecha y con outliers a la izquierda) con el modelo de combinación de XGBoost y Prophet (modelo de menor RMSE de validación en la serie filtrada)

10.4.4 Burgohondo - Cluster 3

Tal y como se puede apreciar en 10.18 el ajuste del mejor modelo encontrado, en este caso una vez más se trata de la combinación XGBoost-Prophet, es capaz de anticipar ciertos altibajos en el valor medio de la serie aunque de nuevo se queda algo corto a la hora de recoger la variabilidad de los datos.

Para poner una perspectiva numérica de nuevo, el RMSE del conjunto de validación de la serie filtrada es igual a 25.47 y al RMSE del test de la serie original es igual a 43.07. Estos números son significativamente menores que en el resto de conjuntos pero hay que tener en cuenta que en este caso el valor promedio se encuentra en torno a los 200 pacientes diarios por lo que el desempeño del modelo parece similar al del resto de zonas.



(a) Predicción de afluencia en los conjuntos de validación y test para la serie original

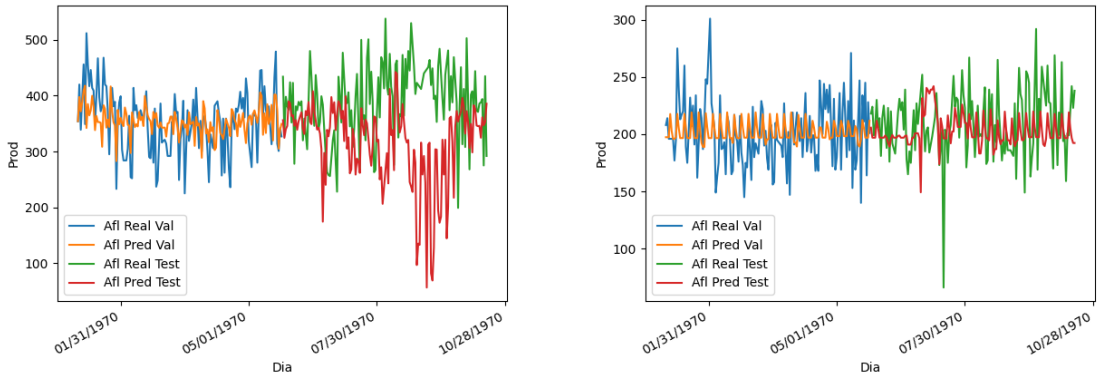
(b) Predicción de afluencia en los conjuntos de validación y test para la serie filtrada

Figure 10.18: Predicción para la serie de Burgohondo (filtrada a la derecha y con outliers a la izquierda) con el modelo XGBoost (lambda 0, max_depth 3 y n_estimators 100) (modelo de menor RMSE de validación en la serie filtrada)

10.4.5 Predicciones de error elevado

Por desgracia aunque los ejemplos mostrados sí que parecen recoger efectivamente la tendencia de los datos, se han detectado un par de zonas de las 16 testadas que no han tenido buenos resultados.

En concreto Ciudad Rodrigo con el modelo XGBoost (lambda 0, max_depth 10 y n_estimators 100)) y Cistierna con el modelo XGBoost (lambda 1, max_depth 3 y n_estimators 100, siendo estos sus mejores 'runs' ofrecen los resultados que se pueden observar en 10.19. Ciudad Rodrigo presenta un RMSE test para la serie original de 129.07 para una afluencia media de 400 pacientes mientras que en Cistierna es de 41.73 para una afluencia media de en torno a 200. Es por ello que no parece posible generalizar este proceso a absolutamente todas las zonas de salud de Castilla y León.



(a) Predicción en Ciudad Rodrigo

(b) Predicción en Cistierna

Figure 10.19: Predicciones para las series de Ciudad Rodrigo (con el modelo XGBoost (lambda 0, max_depth 10 y n_estimators 100)) a la izquierda y de Cistierna(con el modelo XGBoost (lambda 1, max_depth 3 y n_estimators 100)) a la derecha

Chapter 11

Conclusiones

A lo largo del desarrollo de este proyecto se han obtenido distintos resultados que han hecho posible obtener conclusiones, algunas de las cuáles ni siquiera se habían planteado al emepezar el trabajo, y cumplir varios de los objetivos que se plantearon en un inicio peermitiendo así determinar qué modelos, qué hiperparámetros y qué datos son los apropiados para llevar a cabo la predicción de afluencia a centros de atención primaria en Castilla y León.

11.1 Aportaciones

Este trabajo ha tenido como producto múltiples aportaciones en el entendimiento de los datos tratados y su predicción, siendo las conclusiones más importantes las siguientes:

- La aparición de outliers y valores atípicos en los datos de afluencia a centros de atención primaria están profundamente ligados a la fecha de festividades locales y autonómicas. Este hecho supone un problema en la predicción pues existe cierta variabilidad en las fechas de celebración de estas ocasiones que puede conducir a errores de gran magnitud en días concretos. Esto se traduce en un aumento significativo del error al tener en cuenta los valores atípicos como se ha observado en 10.4
- Siguiendo el procedimiento descrito en 5.3 es posible distinguir distintos grupos entre el total de zonas de salud estudiadas. En concreto se han encontrado 4 grupos con comportamientos distintos en cuanto a la evolución de la afluencia media por fechas a lo largo del año. Curiosamente estos grupos parecen corresponderse con zonas más y menos rurales de forma que parece evidente el efecto de las vacaciones en las zonas urbanas.
- El modelo LSTM multistep parece ser adecuado para predicciones a corto plazo, pero su necesidad de grandes ventanas de datos para la validación y su tiempo de cálculo superior hacen de esta una opción no viable a la hora de hacer predicciones a medio y largo plazo que serán las de mayor interés para las administraciones que pudieran emplear la información de estas predicciones
- Es posible aplicar modelos de XGBoosting tomando como características de entrada los distintos datos que esconden las fechas como pueden ser el día de la semana, cuatrimestre, día del mes... Aplicado a estos datos es recomendable como norma general utilizar valores de lambda cercanos

a 1 y max_depth pequeños.
- La aplicación del modelo Prophet ofrece consistentemente predicciones ligeramente peores que las proporcionadas por XGBoosting. En este caso los mejores valores de los hiperparámetro cps y sps varían con cada cluster como queda descrito en 10.4
- Las predicciones que conducen a un menor error RMSE corresponden a las formadas por el valor medio entre el mejor modelo XGBoosting y el mejor modelo Prophet obtenidos para cada zona. Esta combinación de valores se corresponde con el mejor modelo en un gran número de zonas

Por todo esto sería posible decir que los principales objetivos marcados en un inicio se han completado con éxito y han permitido hacer una selección de modelos informada. No obstante se ha comprobado que no todas las zonas ofrecen unos resultados aceptables.

11.2 Trabajo futuro

De igual forma durante la elaboración del proyecto se han detectado distintas áreas de mejora y posible futura investigación que podrían beneficiarse de un estudio a aparte. Los principales aspectos incompletos en este proyecto son los siguientes:

- Tal y como se comenta en 10.4.5 existen zonas en las que las predicciones se alejan de los valores esperados de forma inaceptable. Este comportamiento podría estar debido a cambios en la población de estas áreas, situaciones médicas inesperadas, falta de hiperparámetros suficientes para obtener el mejor modelo en estos casos concretos o simplemente una divergencia en el comportamiento de los datos con respecto a las zonas en las que se ha comprobado que los modelos funcionan tan grande que estos modelos no son aplicables en estos casos.
  Podría ser interesante explorar la idea de integrar otros datos de fuentes distintas o producto de la interacción entre las distintas zonas de salud para comprobar si es posible incorporar más información al modelo que permita obtener un modelo aplicable a la totalidad de zonas.
- Sería también interesante repetir este proyecto en un tiempo futuro en el que se cuente con más de los poco más de 4 años de datos con los que se contaba en la actualidad. Un mayor número de datos permitiría realizar una división entrenamiento-validación-test más homogénea, considerar el entrenamiento de redes LSTM multistep con una gran ventana de datos de salida y con gran seguridad obtener valores de RMSE más bajos para los modelos ya entrenados.
- Las predicciones obtenidas con los modelos estudiados podrían ser utilizadas por las administraciones en su toma de decisiones. Por ello sería interesante el desarrollo de una API que permita consultar el valor predicho por el mejor modelo de cada zona para una fecha determinada. Esta aplicación podría ser de gran valor para la gestión de recursos en la comunidad.

Appendix A

Repositorio GitLab

Todo el código desarrollado y al que se ha hecho mención en este documento se encuentra en el repositorio rep

Estructura del repositorio

El repositorio cuenta con tres carpetas:

- **Analisis_de_datos**: Contiene el fichero `Data_Preprocess_And_Analysis_TFG.ipynb` que recoge el código de preprocesamiento y análisis de datos previo junto con el fichero `Process_Experiment_Results.i` para el análisis de los datos exportados por el programa.
- **Prev**: Contiene ficheros con notebooks de trabajo previo en relación con el desarrollo de modelos de Machine Learning en Python.
- **TFG_APP**: Aplicación Dockerizada que produce los resultados de los distintos modelos y que se describe en el documento.

Appendix B

Manual de instalación

En esta sección se detallan los pasos necesarios para poner en marcha la aplicación de experimentación TFG_APP. El objetivo es proporcionar una guía clara y detallada para que cualquier usuario pueda ejecutar y evaluar los resultados que se discuten en este trabajo desde su propia máquina.

B.1 Requisitos

- **Entorno Docker**: Asegúrese de tener Docker instalado y configurado correctamente en su sistema operativo. Puede descargar Docker desde https://www.docker.com/get-started.
- **MLflow**: Es necesario tener MLflow instalado en su máquina local para acceder y visualizar los resultados de los experimentos que se ejecuten en el contenedor Docker. Puede instalar MLflow siguiendo las instrucciones en https://mlflow.org/docs/latest/quickstart.html#installation.

B.2 Puesta en Marcha

1. **Clonar o Descargar el Repositorio**: Descargue el directorio que contiene todo el código fuente de la aplicación.
   ```
   git clone <URL_del_repositorio>
   cd <nombre_del_directorio>
   ```
2. **Configurar el Entorno**: Asegúrese de tener Docker en ejecución y configurado para poder crear y ejecutar contenedores.
3. **Construir y Levantar el Contenedor**: Utilizando Docker Compose, construya y levante el contenedor que ejecutará la aplicación de experimentación. El archivo 'docker-compose.yml' está configurado para construir la imagen a partir del Dockerfile y exponer el servicio en el puerto 5000. Además, se monta un volumen para almacenar los resultados de MLflow en la máquina local del usuario.

```
docker-compose up --build
```

Este comando construirá la imagen del contenedor utilizando el Dockerfile proporcionado. El parámetro –build asegura que Docker Compose reconstruya la imagen si hay cambios en el Dockerfile o en los archivos relacionados.

4. **Acceder a los resultados**: Una vez que el contenedor esté en ejecución, los resultados de la experimentación se irán alojando en la carpeta mlruns. Para acceder a sus resultados simplemente será necesario ejecutar desde el mismo directorio :

   ```
   mlflow ui
   ```

   De esta forma se proporcionará un enlace para acceder a la interfaz gráfica de MLflow para poder visualizar los distintos resultados.

5. **Descargar Datos (Opcional)**: Si se desea descargar nuevos datos desde la API externa del portal de datos abiertos de la Junta de Castilla y León, se puede hacer cambiando el valor de DOWNLOAD_DATA en el archivo 'docker-compose.yml' bajo 'environment'.

   ```
   environment:
     - DOWNLOAD_DATA=true
   ```

6. **Activar LSTM (Opcional)**: Si se desea activar la experimentación con LSTM (modelo descartado en el documento), se puede hacer igualmente cambiando el valor de PERFORM_LSTM en el archivo 'docker-compose.yml' bajo 'environment'.

   ```
   environment:
     - PERFORM_LSTM=true
   ```

7. **Visualizar Resultados de MLflow**: Para acceder a los resultados de los experimentos realizados por la aplicación en MLflow, abra su navegador web y visite `localhost:5000/mlflow`. Esto redirigirá al usuario al servidor de MLflow dentro del contenedor Docker, donde se pueden ver los runs, métricas y artefactos asociados a cada experimento.

8. **Analizar datos exportados**: Para acceder a los resultados de los experimentos otra alternativa consiste en tomar el archivo csv que se generará en la carpeta data y analizarlo con el archivo Data_Preprocess_And_Analysis_TFG.ipynb que lo leerá con el nombre 'Exp_Results.csv'. De está forma se podrán obtener informes similares a los mostrados en este documento.

Bibliografía

[1] URL: https://xgboost.readthedocs.io/en/stable/tutorials/model.html.

[2] *Actividad de medicina de familia a nivel de consultorio — analisis.datosabiertos.jcyl.es*. https://analisis.datosabiertos.jcyl.es/explore/dataset/actividad-medicina-familia-consultorio/table/?disjunctive.zona_basica_de_salud&disjunctive.area&disjunctive.provincia&sort=fecha. [Accessed 21-05-2024].

[3] *Análisis de datos abiertos JCyL — analisis.datosabiertos.jcyl.es*. https://analisis.datosabiertos.jcyl.es/pages/home/. [Accessed 21-05-2024].

[4] *Cuándo utilizar cada lenguaje de programación en ciencia de datos?* https://datos.gob.es/es/blog/cuando-utilizar-cada-lenguaje-de-programacion-en-ciencia-de-datos. [Accessed 16-05-2024].

[5] Grzegorz Dudek. "Combining Forecasts of Time Series with Complex Seasonality Using LSTM-Based Meta-Learning". In: *Engineering Proceedings* 39.1 (2023). ISSN: 2673-4591. DOI: 10.3390/engproc2023039053. URL: https://www.mdpi.com/2673-4591/39/1/53.

[6] Davide Festa et al. "Unsupervised detection of InSAR time series patterns based on PCA and K-means clustering". In: *International Journal of Applied Earth Observation and Geoinformation* 118 (2023), p. 103276. ISSN: 1569-8432. DOI: https://doi.org/10.1016/j.jag.2023.103276. URL: https://www.sciencedirect.com/science/article/pii/S1569843223000985.

[7] Ayat Hamel and Baydaa Ismael. "Time series Forecasting Using ARIMA model". In: (Feb. 2022).

[8] Syed Nazir Hussain et al. "A Novel Framework Based on CNN-LSTM Neural Network for Prediction of Missing Values in Electricity Consumption Time-Series Datasets". In: *JIPS* 18.1 (2022), pp. 115–129. DOI: 10.3745/JIPS.04.0235. URL: https://doi.org/10.3745/JIPS.04.0235.

[9] V. Roshan Joseph and Akhil Vakayil. "SPlit: An Optimal Method for Data Splitting". In: *Technometrics* 64.2 (2022), pp. 166–176. DOI: 10.1080/00401706.2021.1921037. eprint: https://doi.org/10.1080/00401706.2021.1921037. URL: https://doi.org/10.1080/00401706.2021.1921037.

[10] Youru Li et al. "EA-LSTM: Evolutionary attention-based LSTM for time series prediction". In: *Knowledge-Based Systems* 181 (2019), p. 104785.

[11] Lorenzo Menculini et al. "Comparing prophet and deep learning to ARIMA in forecasting wholesale food prices". In: *Forecasting* 3.3 (2021), pp. 644–662.

[12] *OverLeaf: Online LaTeX Editor*. https://www.overleaf.com/latex/templates/typing-python-in-latex/vvdwybdwcnmk. [Accessed 21-05-2024].

[13] Mijanur Rahman. *Different ways to combine CNN and LSTM networks for time series classification tasks*. Jan. 2023. URL: https://medium.com/@mijanr/different-ways-to-combine-cnn-and-lstm-networks-for-time-series-classification-tasks-b03fc37e91b6.

[14] Sean J. Taylor and Ben Letham. "Forecasting at scale". In: *PeerJ Preprints* 5 (2017), e3190v2. DOI: 10.7287/peerj.preprints.3190v2. URL: https://doi.org/10.7287/peerj.preprints.3190v2.

[15] *The Ultimate Guide to Building Your Own LSTM Models — projectpro.io*. https://www.projectpro.io/article/lstm-model/832. [Accessed 06-06-2024].

[16] XiaoFeng Wang and Ying Zhang. "Multi-step-ahead time series prediction method with stacking LSTM neural network". In: *2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE. 2020, pp. 51–55.

[17] Yan Wang and Yuankai Guo. "Forecasting method of stock market volatility in time series data based on mixed model of ARIMA and XGBoost". In: *China Communications* 17.3 (2020), pp. 205–221.

[18] Bobby Tan Liang Wei. *The intuition behind Gradient Boosting & XGBoost*. May 2020. URL: https://towardsdatascience.com/the-intuition-behind-gradient-boosting-xgboost-6d5eac844920.

[19] Lingyu Zhang et al. "Time series forecast of sales volume based on XGBoost". In: *Journal of Physics: Conference Series*. Vol. 1873. 1. IOP Publishing. 2021, p. 012067.

[20] Huiting Zheng, Jiabin Yuan, and Long Chen. "Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation". In: *Energies* 10.8 (2017). ISSN: 1996-1073. DOI: 10.3390/en10081168. URL: https://www.mdpi.com/1996-1073/10/8/1168.

[21] Hangxia Zhou et al. "Short-Term Photovoltaic Power Forecasting Based on Long Short Term Memory Neural Network and Attention Mechanism". In: *IEEE Access* PP (June 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2923006.

[22] Yanlai Zhou et al. "Explore a deep learning multi-output neural network for regional multi-step-ahead air quality forecasts". In: *Journal of cleaner production* 209 (2019), pp. 134–145.