UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Elaboración de un algoritmo predictivo para la reposición de hipoclorito en los depósitos mediante técnicas de Machine Learning

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Chilet Vera, Álvaro

Tutor/a: López Patiño, José Enrique

CURSO ACADÉMICO: 2022/2023

Resumen

El objetivo de este TFG es desarrollar un algoritmo capaz de predecir la necesidad de reposición de hipoclorito en los procesos de potabilización del agua a partir de la medida del consumo y el nivel de esta sustancia en los depósitos habilitados para dicha potabilización.
Se emplearán técnicas de *Deep Learning*, redes neuronales recurrentes y técnicas de *forecasting*.
La predicción busca analizar cuánto hipoclorito se consume para potabilizar el agua y cada cuanto habría que hacer una compra de esta sustancia.
Los datos necesarios para alimentar el algoritmo se han tomado desde unos sensores ubicados en los depósitos, que han estado recogiendo datos de manera fiable durante más de un año. La periodicidad de las muestras ha sido diaria, aunque en algunos casos ha sido incluso horaria.
La elección de emplear redes neuronales recurrentes y *forecasting* viene dada por el hecho de que ambas técnicas son las más eficientes trabajando con predicciones sobre una línea temporal.
En los resultados se expondrán diferentes métricas que ayudarán a la hora de evaluar la precisión de este modelo y se explicarán las líneas futuras de procedimiento a la hora de la implantación real de este algoritmo en la fase de producción.

Palabras clave: *ciencia de datos, inteligencia artificial, aprendizaje automático, aprendizaje profundo, red neuronal recurrente, forecasting, backtesting, error cuadrático medio, hipoclorito, algoritmo, predicción, entrenamiento, test.*

Resum

L'objectiu d'aquest TFG és desenvolupar un algoritme capaç de predir la necessitat de reposició d'hipoclorit en els processos de potabilització de l'aigua a partir de la mesura del consum i el nivell d'aquesta substància als dipòsits habilitats per a aquesta potabilització.
Es faran servir tècniques de *Deep Learning*, xarxes neuronals recurrents i tècniques de *forecasting*.
La predicció cerca analitzar quant hipoclorit es consumeix per potabilitzar l'aigua i cada quan caldria fer una compra d'aquesta substància.
Les dades necessàries per alimentar l'algoritme s'han pres des d'uns sensors ubicats als dipòsits, que han estat recollint dades de manera fiable durant més d'un any. La periodicitat de les mostres ha estat diària, tot i que en alguns casos ha estat fins i tot horària.
L'elecció de fer servir xarxes neuronals recurrents i *forecasting* ve donada pel fet que les dues tècniques són les més eficients treballant amb prediccions sobre una línia temporal.
Als resultats s'exposaran diferents mètriques que ajudaran a l'hora d'avaluar la precisió d'aquest model i s'explicaran les futures línies de procediment a l'hora de la implantació real d'aquest algoritme a la fase de producció.

Paraules clau: *ciència de dades, intel·ligència artificial, aprenentatge automàtic, aprenentatge profund, xarxa neuronal recurrent, forecasting, backtesting, error cuadràtic mitjà, hipoclorit, algoritme, predicció, entrenament, test.*

Abstract

The aim of this TFG is to develop an algorithm capable of predicting the need for hypochlorite replenishment in water purification processes based on the measurement of consumption and the level of this substance in the tanks set up for this purification.
Deep Learning techniques, recurrent neural networks and forecasting techniques will be used.
The prediction aims to analyse how much hypochlorite is consumed to make water drinkable and how often this substance should be purchased.
The data needed to feed the algorithm were taken from sensors located in the tanks, which have been reliably collecting data for more than a year. The periodicity of the samples was daily, although in some cases it was even hourly.
The choice of using recurrent neural networks and forecasting is due to the fact that both techniques are the most efficient when working with predictions on a timeline.
In the results, different metrics will be presented that will help to evaluate the accuracy of this model and the future procedure for the real implementation of this algorithm in the production phase will be explained.

Keywords: *data science, artificial intelligence, machine learning, deep learning, recurrent neural network, forecasting, backtesting, mean squared error, hypochlorite, algorithm, prediction, training, test.*

Tabla de contenidos

Índice de figuras

Índice de gráficas

Índice de tablas

1. Introducción

Actualmente, el ámbito de la Inteligencia Artificial (IA) está en auge. En su libro "Sobre la inteligencia", publicado en 2005, Sandra Blakeslee y Jeff Hawkins definían la inteligencia como la capacidad de predecir el futuro en base a los patrones almacenados en la memoria. Ese mismo principio está detrás del *Machine Learning* (ML), también conocido como aprendizaje automático [1].

El presente trabajo consta de la creación de un algoritmo predictivo, en base a datos y variables, de la reposición del hipoclorito de sodio en los depósitos.

La cloración es uno de los métodos más utilizados para potabilizar el agua. Consiste en añadir hipoclorito de sodio al agua eliminando así virus, bacterias y sustancias orgánicas. De esa manera podemos suministrar agua en condiciones óptimas y de calidad a la población.

Este trabajo de investigación se centra en la Estación de Tratamiento de Agua Potable (ETAP) de L'Eliana y los depósitos de agua de Paterna. Ambas estaciones cuentan con un depósito donde se almacena y suministra hipoclorito al agua con el fin de tratarla para que sea apta para el consumo humano.

1.1 Motivación

El agua es fuente de vida y salud. Su calidad está íntimamente relacionada con el nivel de vida y con el nivel sanitario de un país. El agua se somete a controles y tratamientos que permiten que llegue en buenas condiciones a nuestros hogares y sea consumida con total seguridad. La mágica acción de abrir el grifo y que salga agua lista para el consumo presenta una gran complejidad donde intervienen varios agentes como los municipios, empresas abastecedoras, laboratorios de control y administraciones sanitarias que velan por que el agua sea de buena calidad, sin riesgos para la salud, fácilmente accesible y en la cantidad requerida [2] [3].

Este proyecto consta de una parte fundamental en la potabilización del agua como es el almacenamiento y suministro de hipoclorito.

Está impulsado por "Hidraqua, Gestión de Aguas de Levante S.A.", más en concreto por Dinapsis.

Dinapsis es un centro referente en transformación digital. Esta digitalización mejora la gestión del ciclo integral del agua, así como optimiza las operaciones a realizar en el plan urbanístico de la ciudad [4].

Esta investigación nace del deseo de Dinapsis de automatizar un proceso clave en la potabilización como es el almacenamiento y el suministro de hipoclorito al agua.

Este trabajo me ha permitido adentrarme en el mundo de la Inteligencia Artificial, que es algo que llevaba tiempo queriendo, ya que el análisis de los datos se lleva a cabo mediante la técnica del *forecasting* y entrenando una red neuronal.

La mayor motivación para la realización de este algoritmo predictivo ha sido en todo momento la certeza de que es algo extrapolable a la vida real, que aporta beneficios no solo económicos sino también medioambientales y que puede asentar las bases para una futura implementación a gran escala.

1.2 Objetivos

La Estación de Tratamiento de Agua Potable (ETAP) de L'Eliana y los depósitos de agua de Paterna cuentan con un sensor que mide el nivel de hipoclorito en el depósito, la temperatura interna y la energía consumida en este proceso. El sensor tiene lecturas de los últimos años y con una periodicidad horaria.

A estos datos se le suman la cantidad de agua suministrada diaria y los pedidos de hipoclorito realizados en cada uno de los municipios.

El objetivo de este proyecto es, analizando los datos anteriormente mencionados, empleando técnicas de *forecasting* y utilizando redes neuronales mediante el uso de *Machine Learning* (ML), la creación de un algoritmo que pueda predecir el nivel de hipoclorito y cuando se debe realizar un pedido de este para reponer el depósito. Esta predicción será contrastada y verificada con los datos del sensor actual.

Con la realización de este trabajo se pretende optimizar el consumo de hipoclorito y ahorrarse los costes de la adquisición de los sensores físicos, así como los costes de mantenimiento de este.

2. Marco teórico

Para la mejor comprensión de las técnicas y métodos empleados para este proyecto, se cree necesaria la creación de un marco conceptual.

Se desarrollará de manera más general a más específica para explicar en detalle la metodología seguida, así le será más sencillo al lector situarse y entender las decisiones tomadas a lo largo del trabajo.

2.1 *Big Data*

El término *Big Data* no tiene una definición clara ni reconocida, sino que hace referencia más bien a un paradigma de programación para resolver problemas que no son abordables con las técnicas de computación tradicionales.

Estos problemas de gran envergadura se caracterizan por cuatro propiedades, más bien conocidas como las 7V's del *Big Data*, que hacen referencia a volumen, velocidad, variedad, veracidad, valor, visualización y viralidad de los datos.

Para abordar problemas de estas características se debe enfocar la solución de manera diferente a como se ha estado realizando hasta ahora. Entre otros aspectos, se debe tener en cuenta otra forma de configurar la arquitectura, tales como motores de bases de datos o *frameworks* de procesamiento, la estructuración de los datos o utilizar *clústers* de máquinas usando el potencial de procesamiento paralelo y distribuido. Este nuevo enfoque implica que gran parte de los modelos y sistemas que se han estado utilizando hasta ahora queden limitados, forzando a la comunidad investigadora a abrir nuevos frentes para abordar dichas limitaciones [5].

Figura 1: 7 V 's del *Big Data [5]*.

2.2 Inteligencia Artificial

El término Inteligencia Artificial fue acuñado en la Conferencia de Dartmouth en 1956 por el informático estadounidense John McCarthy. Él decía que cualquier aspecto del aprendizaje o rasgo de la inteligencia podían, en principio, ser descritos de una forma tan precisa que se pudiera crear una máquina que los simulara [6].

A día de hoy, ese es su principal objetivo: la creación de sistemas que puedan funcionar de manera independiente e inteligente y que intenten emular la inteligencia humana. Que sean capaces de aprender, mejorar y adaptarse a distintos entornos.

Se emplea en infinidad de materias y con diversas aplicaciones, lo cual hace prácticamente imposible enumerarlas todas, pero estas son algunas de las más destacables:

- Sector financiero: se emplea la Inteligencia Artificial para multitud de funciones, como pueden ser la gestión y adquisición de acciones, la organización de propiedades, la administración de operaciones, etc.

- Sector sanitario: la IA es usada en este sector para automatizar diversas actividades. El uso más destacable es para la prevención y detección de enfermedades. También es conocido el robot *Da Vinci*, empleado para asistir a cirujanos a la hora de facilitar una operación.

- Atención al cliente: los asistentes virtuales, más conocidos como *chatbots,* que son capaces mediante la Inteligencia Artificial de atender a un usuario de manera automática.

- Sector de la música: empleándose en la edición y procesamiento automatizado de canciones. Además, se está avanzando en los últimos meses con tecnologías que son capaces de crear canciones desde cero e incluso usar la voz de un cantante en canciones que no son suyas.

- Sector de la industria: lo más empleado en tareas industriales son los robots automatizados mediante IA. Estos robots pueden realizar tareas repetitivas, que sean peligrosas para el ser humano o que requieran una fuerza sobrehumana.

No podemos hablar de la Inteligencia Artificial sin comentar al que es considerado el padre de la ciencia de computación. Alan Mathison Turing fue el creador, en 1950, de la prueba de Turing, un criterio mediante el cual puede juzgarse la inteligencia de una máquina si sus respuestas en la prueba son indistinguibles de las de un ser humano.

En resumen, consiste en hacer que un usuario converse con distintas máquinas. Algunas de las máquinas están controladas por humanos y otras por la Inteligencia Artificial. El usuario debe descubrir cuál de las máquinas no está siendo controlada por un humano. En caso de no detectar cual está siendo controlada por Inteligencia Artificial, se podría decir que esa máquina ha superado el test de Turing [7].

Figura 2: Test de Turing [7].

2.3 *Machine Learning*

El *Machine Learning* es una disciplina del campo de la Inteligencia Artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones. Este aprendizaje permite a los ordenadores realizar tareas específicas de forma autónoma sin necesidad de ser programados, que es exactamente lo que se busca en este proyecto.

Arthur Samuel, programador y diseñador de la gigante americana IBM, fue el pionero en el *Machine Learning* cuando en los años 50 contribuyó a que una máquina mejorara al juego de las damas. Esto fue posible ya que el ordenador estudiaba los movimientos que componían estrategias ganadoras [8].

Hoy en día, su uso es muy amplio: detección y filtrado de *SPAM*, personalización y depuración de contenido en redes sociales, video vigilancia, asistentes personales como *Siri* o *Alexa*, predicción del tráfico y asistencia en los desplazamientos, atención al cliente online, detección de fraudes online y personalización de publicidad…

Se trata de un campo que en el mundo actual presenta infinidad de aplicaciones. Las empresas están invirtiendo cada vez más en este aspecto ya que puede aportarles una gran eficiencia en sus tareas y puede reducir sus costes económicos, como es el caso de este trabajo de investigación.

2.3.1 Aprendizaje supervisado

En este paradigma se divide el conjunto de datos en dos subconjuntos, el set de entrenamiento y el set de testeo. Primero se entrena el modelo utilizando unos datos etiquetados. Que estén etiquetados significa que para cada característica de entrada X, conocemos el valor de su atributo objetivo Y. Una vez que termina el proceso de entrenamiento, se usan los datos objetivo para predecir el atributo objetivo y contrastar con los valores reales y así hallar la precisión de nuestro algoritmo. El Aprendizaje

Supervisado se usa en problemas de predicción de valores continuos, llamados problemas de regresión y en problemas de predicción de valores discretos, llamados problemas de clasificación. En este TFG se utilizará aprendizaje supervisado dado que se trata de un problema de regresión.

2.3.2 Aprendizaje no supervisado

Cuando no tenemos una característica objetivo y solo conocemos los datos de entrada, usamos el Aprendizaje No Supervisado. En este tipo, describimos la estructura de los datos para tratar de obtener un ordenamiento de estos. Se usa en problemas de segmentación y *clustering*.

2.3.3 Aprendizaje por refuerzo

Se usa un sistema de retroalimentación, donde la máquina aprende por ensayo y error. Se intenta alcanzar un comportamiento óptimo para la resolución del problema, aprendiendo de iteraciones pasadas y maximizando la recompensa.

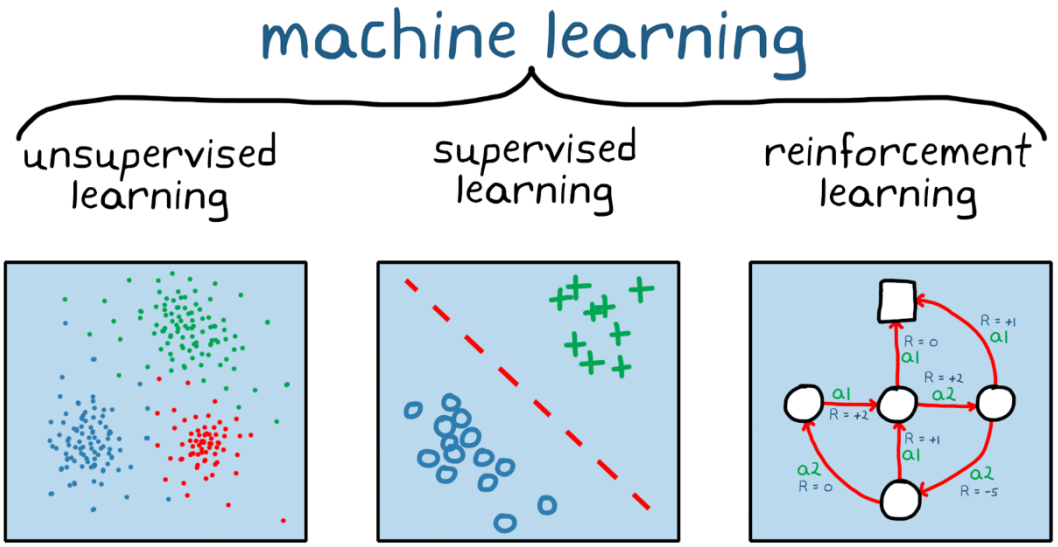


Figura 3: Tipos de aprendizaje automático [9].

2.4 *Deep Learning*

Para situarnos de cuando se empezó a hablar del *Deep Learning* debemos remontarnos a los años 40. El hecho de que este término parezca algo reciente es porque a lo largo del tiempo ha sido rebautizado con muchos nombres hasta llegar en los últimos años a ser conocido como lo denominamos hoy en día.

Las etapas del *Deep Learning* son las siguientes:

- Etapa cibernética: esta primera corriente arrancó con los estudios sobre el aprendizaje biológico, impulsado por McCulloch&Pitts (1943) y Hebbs (1949). Estos dieron pie a las implementaciones de primeros modelos, como por ejemplo el *'Perceptrón Rossenblat'* (1958), que permitía el entrenamiento de una única neurona.
- Etapa de Conexionismo: fue en esta época en la que surgió el concepto de *'backpropagation'* con la llegada de Rummerhart et al (1986). Este término se emplea de manera masiva en el entrenamiento de redes neuronales para calcular los pesos de las neuronas correspondientes a las distintas capas de las mismas.
- Etapa *Deep Learning*: aunque la forma de aprendizaje está ligada al cerebro biológico, ya no se encuentra tan relacionado con la perspectiva neurocientífica. Hay que pensar que esta disciplina busca dar una solución al hecho de que hay máquinas que sobrepasan a nuestras mentes en tareas formales o abstractas.

En la siguiente imagen (figura 4) podemos observar como se engloba de una manera visual todo lo que hemos expuesto hasta ahora.

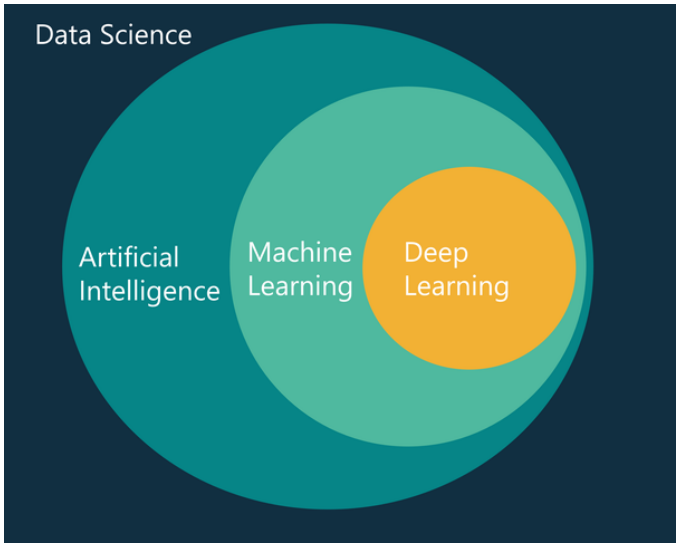


Figura 4: Explicación visual de *Big Data*, Inteligencia Artificial, *Machine Learning* y *Deep Learning [10]*.

2.5 Redes neuronales artificiales

Las Redes Neuronales Artificiales o RNAs tienen el objetivo de replicar el funcionamiento de una red neuronal real mediante *software*.

Los nodos de una RNA se llaman neuronas artificiales y son las encargadas de ejecutar las operaciones. Las conexiones entre estas neuronas contienen el resultado de estas operaciones y la importancia (peso) que hay que darle a ese resultado. [11]

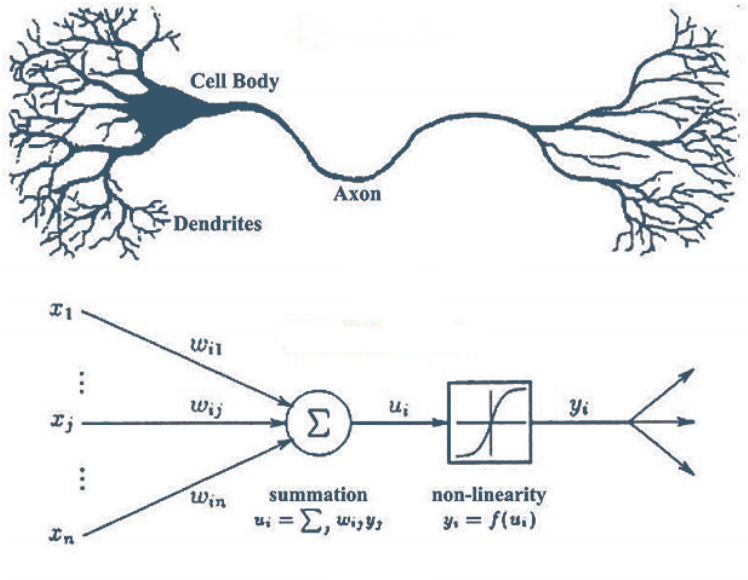


Figura 5: Comparación de neurona humana con neurona artificial [11].

Las redes se estructuran en capas, siempre habiendo una de entrada y una de salida. En la capa de entrada se recogen todos los datos que van a ser procesados y en la de salida se muestran los resultados de las operaciones. También suelen tener una o más capas ocultas donde se aplican las operaciones para dar la salida que se haya solicitado.

Dentro de esta red también actúan funciones que limitan los valores que permiten pasar a la siguiente capa. Estas funciones son llamadas funciones de activación.

Para poder utilizar la red es necesario entrenarla. Con los resultados de entrenamiento se crea un modelo. El entrenamiento se lleva a cabo utilizando un conjunto de datos de los que ya se conoce la respuesta esperada. Estos datos se separan en entrenamiento y validación. Los de entrenamiento los utiliza la red para aprender y los de validación son empleados para comprobar su funcionamiento.

Para comprobar el éxito de la red, debemos mirar la función de pérdida, que comprueba cuánto se asemejan los resultados a la realidad. Nuestro objetivo será minimizar el resultado de esta función.

Una vez consideramos que la red se ha ajustado lo suficiente a los datos de forma que devuelve resultados fiables para ser utilizados, se exporta el modelo que contiene los pesos que ha asignado la red a cada una de las variables de los datos de entrada. Este modelo es el que se utiliza con datos reales para calcular los resultados deseados.

2.5.1 Red neuronal monocapa

Es la red más simple que podemos encontrar. Está compuesta por tan solo una capa de entrada que envía los datos directamente a la capa de salida, donde se realizan operaciones simples.

Como podemos observar en la figura 6, esto sería un ejemplo gráfico del perceptrón simple [12].

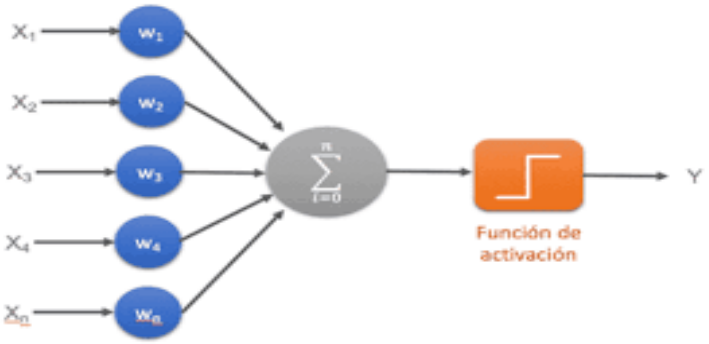


Figura 6: Ejemplo de Red Neuronal Monocapa [12].

2.5.2 Red neuronal multicapa

Se trata de una extensión de la red anterior. Su principal diferencia es la existencia de capas ocultas entre la capa de entrada y la de salida [13].

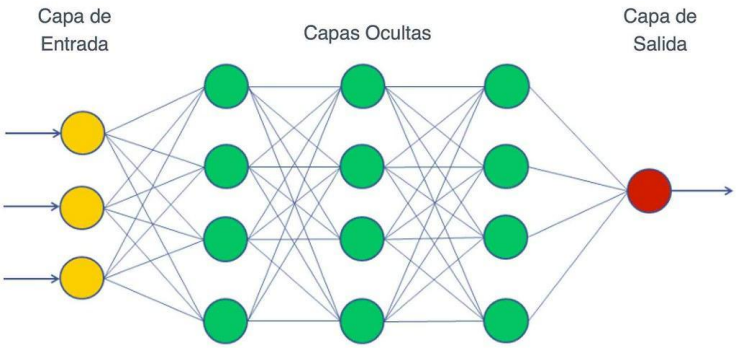


Figura 7: Ejemplo de Red Neuronal Multicapa [13].

2.5.3 Red neuronal convolucional

Es una red de neuronas donde no todas ellas están conectadas entre sí, sino que se especializan en un pequeño grupo. De esta forma conseguimos disminuir la complejidad computacional y la cantidad de neuronas que necesitamos para ejecutar la red.

Las neuronas de las capas convolucionales se encargan de crear un mapa de características de los datos de entrada. También se compone de capas reductoras que son las encargadas de extraer las características más habituales. Es por esto por lo que estas redes son ampliamente utilizadas en la detección de imágenes [14].

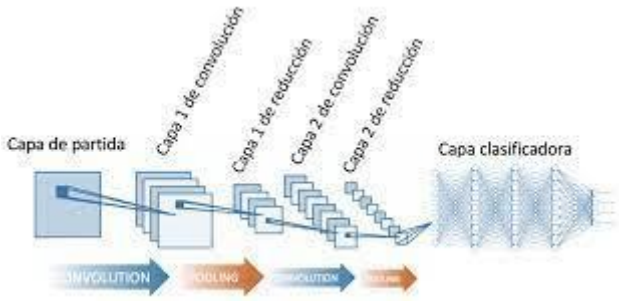


Figura 8: Ejemplo de Red Neuronal Convolucional [14].

2.5.4 Red neuronal recurrente

Se trata de una red muy similar a la multicapa. La principal diferencia se encuentra en que las neuronas se interconectan no solo con la siguiente capa, sino también con ellas mismas. De esta forma se crean ciclos en los que su salida sirve de entrada adicional para que los datos se ajusten mejor a la red.

Este tipo de red junto con la técnica del *forecasting* son los métodos que se van a utilizar en este proyecto ya que son excelentes para la realización de predicciones sobre una serie temporal [15].

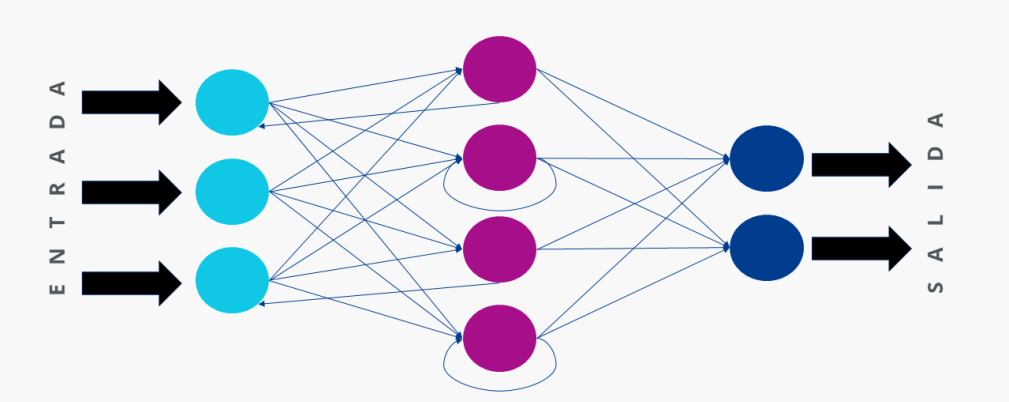


Figura 9: Ejemplo de Red Neuronal Recurrente [15].

3. Metodología

El proceso seguido para la construcción del modelo y la aplicación del algoritmo predictivo ha consistido en las siguientes fases:

- Visita a las instalaciones de Paterna y L'Eliana para comprender el proceso de cloración del agua.
- Elección del entorno y lenguaje de programación para la aplicación de técnicas de *Machine Learning*.
- Obtención, limpieza y carga de los datos al modelo.
- Evaluación de variables predictivas determinantes.
- Construcción y entrenamiento de los modelos.
- Conclusiones y desarrollo futuro.

3.1 Contexto y explicación de las instalaciones

A continuación, para el mejor entendimiento de la materia, se ha decidido ofrecer un breve resumen que de manera visual haga comprender al lector el proceso de cloración del agua y la importancia de la implantación de un algoritmo predictivo.

3.1.1 ETAP de L'Eliana y depósito de Paterna

Para la mejor precisión a la hora de la obtención de datos y la comprobación de resultados, se ha decidido hacer el estudio en los municipios valencianos de Paterna y L'Eliana. Esta decisión se ha llevado a cabo debido a que son municipios relativamente grandes, con altas cantidades de agua a potabilizar, por ello, la implantación de este algoritmo para la reposición de hipoclorito puede suponer un impacto económico y medioambiental mayor que en otras zonas.

Figura 10: ETAP de L'Eliana.

3.1.1.1 Extracción de agua de pozos

El primer paso es la extracción de agua de los pozos. Esta agua es extraída y canalizada hacia los depósitos de agua bruta. En este momento, el agua aún no es apta para el consumo, ya que contiene una cantidad de nitratos superior al 50%, que es el límite marcado por la OMS que determina si el agua es potable o no. Por ejemplo, en L'Eliana el agua se extrae de 4 pozos diferentes que llegan al depósito de agua bruta de la ETAP. En las siguientes figuras podemos observar uno de los cuatro pozos de L'Eliana y el depósito de agua bruta donde es almacenada el agua extraída.

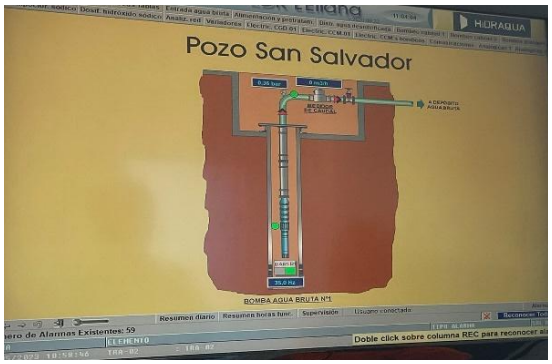


Figura 11: Pozo de extracción de agua.

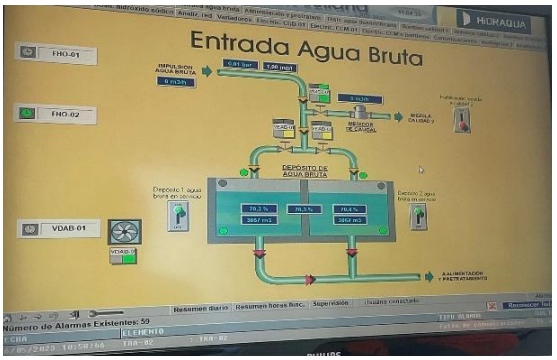


Figura 12: Depósito de agua bruta.

3.1.1.2 Carga de hipoclorito al depósito

El agua se está extrayendo de los pozos y clorando continuamente y al mismo tiempo. En este punto todavía tenemos un agua que no es apta para el consumo, esperando a ser clorada. El hipoclorito llega a las instalaciones mediante un camión cisterna en todas las condiciones de seguridad pertinentes, ya que es un vertido irritable y corrosivo que puede causar daño en las personas. El hipoclorito del camión es traspasado al depósito de esta misma sustancia mediante una tubería.

Figura 13: Caseta de carga de hipoclorito, Paterna

Figura 14: Tubería para la carga de hipoclorito, Paterna.

Figura 15: Depósito de hipoclorito, Paterna.

Figura 16: Depósito de hipoclorito, L'Eliana.

3.1.1.3 Bombas dosificadoras de hipoclorito al agua

Una vez tenemos el hipoclorito por un lado y el agua por otro, solo queda dosificarlo para conseguir que esta sea apta para el consumo. El suministro de esta sustancia al agua se hace mediante bombas, donde se traspasa de manera automática a través de una pequeña tubería. El suministro óptimo de cloro para que el agua sea potable está en torno al 0.8-0.9 miligramos de cloro por cada litro de agua.

En la siguiente imagen podemos ver un esquema donde se aplica el hipoclorito al agua bruta (círculos rojos) y se almacena en un depósito de agua potable (círculo verde).

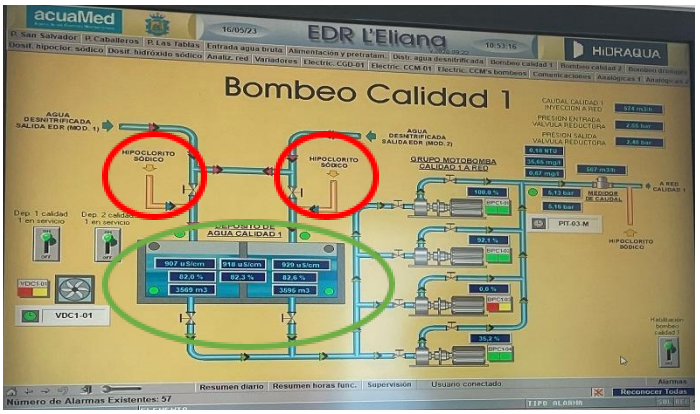


Figura 17: Esquema de cloración y almacenaje.

A continuación, se muestran las bombas que suministran el cloro almacenado en los depósitos mostrados anteriormente.

Figura 18: Bombas del depósito de Paterna.

Figura 19: Bombas de la ETAP de L'Eliana.

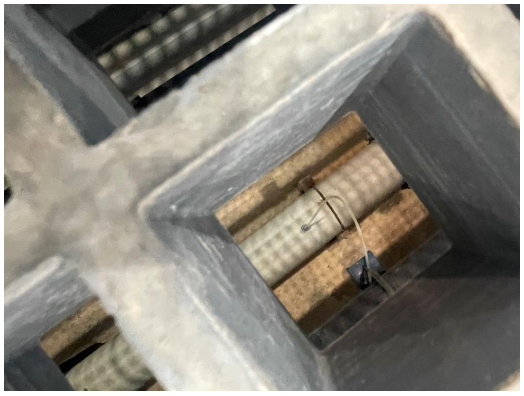


Figura 20: Tubería ancha(agua bruta) y tubería estrecha(hipoclorito).

3.1.1.4 Sensor actual

Los depósitos actuales cuentan con un sensor que mide, a través de ultrasonidos, el nivel de hipoclorito que hay. El problema de estos sensores es que son muy costosos, presentan un alto coste de mantenimiento y son a veces imprecisos. A la hora de la creación del algoritmo predictivo mediante *Machine Learning* nos han servido para la recolección y limpieza de datos, la alimentación para el aprendizaje de la red neuronal y para la comprobación de resultados.

Figura 21: Sensor *'Ilevel'* actual en Paterna.

3.2 Entorno y lenguaje para el algoritmo predictivo

Trabajar con algoritmos de *Machine Learning* y manejar redes neuronales requiere de un coste computacional muy elevado. Esto hace que para trabajar en este campo se necesite un equipo con grandes prestaciones y por tanto costoso.

Por ello, para este proyecto se va a emplear uno de los conceptos más novedosos e interesantes de los últimos tiempos, la programación colaborativa gracias a 'Google Colab'.

Google Colab es un entorno de codificación creado por 'Jupyter Notebook', utilizado para el aprendizaje automático y el análisis de datos.

La gran ventaja de este entorno es que incorpora una gran variedad de bibliotecas sobre las que poder realizar diferentes funciones y ofrece acceso a CPU de servidor, GPU de última generación y TPU de forma gratuita.

Google Colab funciona sobre Python. Python es un lenguaje de programación que permite trabajar rápidamente e integrar sistemas de manera efectiva. Es muy útil en nuestro caso ya que tiene funcionalidades especializadas para el análisis numérico y el aprendizaje automático.

Python es un lenguaje muy maduro, versátil, y se puede utilizar como una plataforma específica para el *Data Science*, gracias a su gran ecosistema de librerías científicas y a su comunidad, que es muy activa.

3.2.1 Bibliotecas

Como se dice anteriormente, Python es una de las comunidades de programación más grande y activa que existe, lo que hace que incluya muchas librerías. En este proyecto se emplean algunas de ellas que nos facilitaran el proceso a la hora de realizar diferentes funciones.

3.2.1.1 *NumPy*

Numerical Python es la biblioteca principal para el cálculo científico ya que proporciona potentes estructuras de datos, implementando matrices N – dimensionales. Estas garantizan cálculos eficientes ya que son compactas y tienen acceso a lectura y escritura más rápida.

NumPy contiene un potente conjunto de herramientas y técnicas que pueden ser usadas para resolver modelos matemáticos en ciencia e ingeniería.

La estructura de datos central de *NumPy* es el array *NumPy (ndarray)*. Con esta tecnología hemos podido ejecutar operaciones con vectores muy eficientes, ya que permite procesar arrays completos, en vez de iterar y operar elemento a elemento [16].

```
#ejemplos que la red usará para aprender
#entradas con los valores de agua suministrada diaria de 2019 a 2023

                                #Año 2019
agua_suministrada = np.array([8778, 8778, 8778, 8778, 8778, 8778, 8778,
                              8787, 8787, 8787, 8787, 8787, 8787, 8787,
                              7839, 7839, 7839, 7839, 7839, 7839, 7839,
                              7996, 7996, 7996, 7996, 7996, 7996, 7996,
                              8969, 8969, 8969, 8969, 8969, 8969, 8969,
                              9692, 9692, 9692, 9692, 9692, 9692, 9692,
                              9863, 9863, 9863, 9863, 9863, 9863, 9863,
                              9417, 9417, 9417, 9417, 9417, 9417, 9417,
                              9449, 9449, 9449, 9449, 9449, 9449, 9449,
                              9858, 9858, 9858, 9858, 9858, 9858, 9858,
                              9859, 9859, 9859, 9859, 9859, 9859, 9859,
                              8255, 8255, 8255, 8255, 8255, 8255, 8255,

                              #Año 2020
                              9420, 9420, 9420, 9420, 9420, 9420, 9420,
                              9375, 9375, 9375, 9375, 9375, 9375, 9375,
                              8400, 8325, 8250, 8175, 8100, 8025, 8050,
                              7414, 7612, 7329, 7519, 7472, 7388, 7848,
                              8644, 7803, 8055, 8597, 7869, 8253, 8313,
                              8317, 8646, 9006, 9154, 9205, 7231, 8933,
                              9429, 8466, 9677, 8062, 8283, 9814, 11676,
                              8732, 8585, 9183, 9129, 9318, 8959, 9475,
                              9742, 9685, 9401, 9733, 8463, 8347, 9934,
                              9586, 10080, 8511, 9003, 9554, 9923, 9747,
                              8825, 8906, 8736, 8513, 8268, 8384, 8348,
                              8951, 8566, 9166, 7313, 7771, 7918, 7996,
```

Tabla 1: Ejemplo de la biblioteca *NumPy.* Elaboración propia.

3.2.1.2 *Pandas*

Panel Data proporciona estructuras de datos de alto rendimiento y herramientas de análisis de datos, que es lo que más nos interesa para poder limpiar y analizar los datos.

La característica clave de *Pandas* es el objeto *Data Frame*. *Data Frame* puede verse como una hoja de cálculo dentro del propio entorno y ofrece formas muy flexibles de trabajar con él. Es capaz de transformar un conjunto de datos de muchas maneras, por ejemplo, añadiendo o eliminando filas y columnas, insertando datos o modificando los ya existentes. Estas funciones nos han ayudado mucho a la hora de realizar el trabajo.

Como observamos en las siguientes figuras, se pueden extraer estadísticas de nuestro conjunto de datos como la media o la desviación estándar. También se pueden ver y eliminar los conjuntos vacíos, entre otras funciones [17].

```
#Mostrar las estadísticas de los datos(media, percentiles, máximo, mínimo...)

dataframe.describe()
```

	Agua_Suministrada	Agua_Suministrada_Diaria	Temperatura	Precipitación
count	12.000000	12.000000	12.000000	12.000000
mean	257140.583333	8452.250000	17.666667	23.750000
std	17293.804716	480.413908	5.804909	10.480501
min	227827.500000	7614.500000	11.000000	5.000000
25%	248095.500000	8123.375000	12.500000	18.500000
50%	262717.000000	8642.000000	17.500000	23.500000
75%	269007.375000	8744.125000	23.000000	29.250000
max	282756.000000	9121.000000	26.000000	43.000000

```
#Ver si hay alguna variable vacía (True en caso de estar vacía)

dataframe.isna()
```

	Mes	Agua_Suministrada	Agua_Suministrada_Diaria	Temperatura	Precipitación
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	False	False
10	False	False	False	False	False
11	False	False	False	False	False

```
#Eliminar todas las variables vacías en caso de haber alguna

dataframe = dataframe.dropna(axis = 0)
```

```
#Vuelvo a ejecutar el dataframe con las variables vacías ya eliminadas

dataframe.isna()
```

	Mes	Agua_Suministrada	Agua_Suministrada_Diaria	Temperatura	Precipitación
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False

Tablas 2 y 3: Ejemplos empleados de la biblioteca *Pandas*. Elaboración propia.

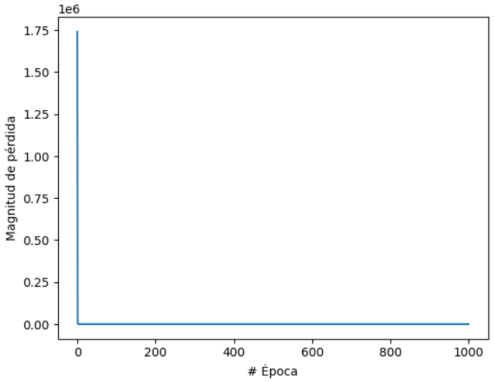
3.2.1.3 *Matplotlib*

No se puede hablar del análisis de datos con *Pandas* sin mencionar *Matplotlib*. Esta librería de *software* gráfico permite la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o *arrays*. Junto con *Pandas* han sido las dos librerías de Python que más se han empleado en el proyecto para el tratamiento de los datos [18].

```
#resultado de la función de pérdida, para ver que tan mal están los resultados de la red en cada vuelta(epoch)

import matplotlib.pyplot as plt
plt.xlabel("# Época")
plt.ylabel("Magnitud de pérdida")
plt.plot(historial.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7f6c31b59de0>]
```



Gráfica 1: Ejemplo de la biblioteca *Matplotlib*. Elaboración propia.

3.2.1.4 *TensorFlow*

Es una librería orientada a la creación de diversos modelos de aprendizaje automático, permitiendo su ejecución de manera eficiente en equipos con un rango de prestaciones *hardware* muy variable.

Esta librería fue liberada bajo la licencia Apache en 2015. La API más estable y desarrollada es la de Python.

La razón por la que se ha decidido trabajar con esta biblioteca ha sido la gran comunidad que colabora con *TensorFlow*, su eficiencia y la facilidad de integración para el *Machine Learning* [19].

3.2.1.5 *Keras*

Fue creada en 2015 por el ingeniero de Google François Chollet. Es una librería para Python diseñada para construir redes neuronales de forma rápida y modular en *TensorFlow*.

Su vocación es la de ser una interfaz para diversas bibliotecas de *Deep Learning*, actualmente también posee soporte en Theano y Microsoft.

Para este proyecto, se ha empleado en la creación de las redes neuronales corriendo sobre *TensorFlow* [20].

```
#identificar las capas de entrada y salida con keras
#capa de tipo densa para usar conexiones desde cada neurona hacia todas las neuronas de la siguiente capa
#units = neuronas de la capa
#input_shape = numero de neuronas en la capa de entrada

###capa = tf.keras.layers.Dense(units = 1, input_shape = [1])
###modelo para introducir las capas y trabajar con el
###modelo = tf.keras.Sequential([capa])

oculta1 = tf.keras.layers.Dense(units = 3, input_shape = [1])
oculta2 = tf.keras.layers.Dense(units = 3)
oculta3 = tf.keras.layers.Dense(units = 3)
oculta4 = tf.keras.layers.Dense(units = 3)
oculta5 = tf.keras.layers.Dense(units = 3)
salida = tf.keras.layers.Dense(units = 1)

modelo = tf.keras.Sequential([oculta1, oculta2, oculta3, oculta4, oculta5, salida])
```

```
#propiedades de como quiero que procese para poder aprender mejor
#optimizador y función perdida (ajustar pesos y sesgos de manera eficiente)
#0.1 = tasa de aprendizaje

modelo.compile(
    optimizer = tf.keras.optimizers.Adam(0.01),
    loss = 'mean_squared_error'
)
```

Figura 22: Ejemplo de la biblioteca *Keras*. Elaboración propia.

3.2.1.6 *Scikit-Learn*

Librería de *software* libre para Python enfocada en el campo del aprendizaje automático. Contiene una gran variedad de algoritmos de clasificación, regresión y reducción de dimensionalidad. Se ha empleado en este proyecto en la tarea operacional con los vectores, ya que está diseñada para interoperar con la biblioteca *NumPy*.

En el apartado de la elaboración del algoritmo predictivo se explica la técnica *skforecast* de la librería *Scikit-Learn* [21].

3.3 *Dataset*

"La información es el aceite del siglo XXI, y la analítica es el motor de combustión" es una frase de Peter Sondergaard, el que fue vicepresidente ejecutivo del grupo Gartner y ahora creador de Sondergaard Group, dos empresas líderes en la consultoría y la investigación de las tecnologías de la información [22].

En este trabajo se han recopilado datos de distintas fuentes, pero lo más importante, se han analizado de manera que han podido ser usados de una manera real en la creación del algoritmo predictivo.

Más adelante se explica la obtención de los datos, el análisis previo y la evaluación de las variables, la carga, limpieza y preparación de estos y por último, la separación de los datos en entrenamiento, validación y test.

3.3.1 Obtención de los datos

Los datos se han obtenido del depósito de Paterna y la ETAP de L'Eliana. De ambas partes me han facilitado los datos de agua suministrada al municipio, consumo diario de hipoclorito y pedidos de hipoclorito. Todos estos datos con una periodicidad diaria desde diciembre de 2019 hasta diciembre de 2022.

También de la web de AEMET, con una API key, se han podido extraer datos de temperatura y precipitación históricos, variables que nos podrían servir de utilidad al menos en el análisis previo

A parte de estos datos, el sensor implantado en los depósitos que he mostrado previamente, nos ha facilitado datos del nivel de hipoclorito continuo que hay en el depósito y la temperatura interna que hay en el depósito. Estos datos son los más relevantes en cuanto a la elaboración del algoritmo y en cuanto al testeo de las predicciones porque así se pueden contrastar con la realidad.

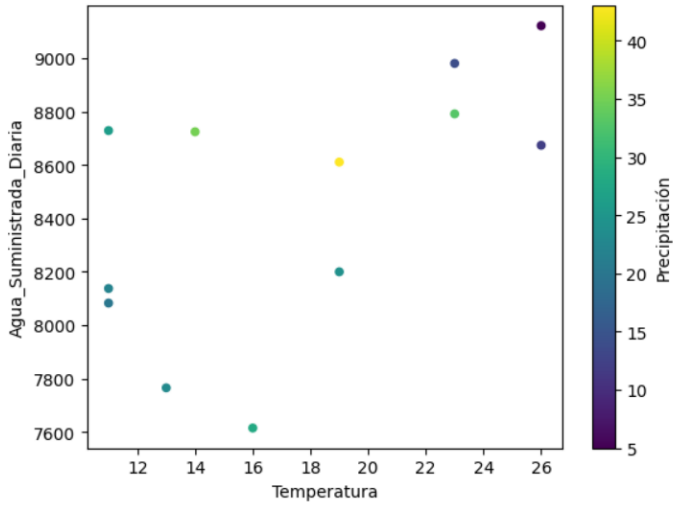
3.3.2 Análisis previo y evaluación de variables

Una vez tenemos los datos que sobre el papel son los más relevantes para la creación del algoritmo predictivo, lo más importante es su análisis y ver la importancia que tienen cada uno de ellos en la práctica.

Comenzamos analizando la importancia que tiene la temperatura en el consumo de agua suministrada, ya que esta variable es clave para nuestro objetivo a predecir que es el hipoclorito.

```
[ ] dataframe.plot.scatter(x = 'Temperatura', y = 'Agua_Suministrada_Diaria', c = 'Precipitación', colormap = 'viridis')
```

```
<Axes: xlabel='Temperatura', ylabel='Agua_Suministrada_Diaria'>
```

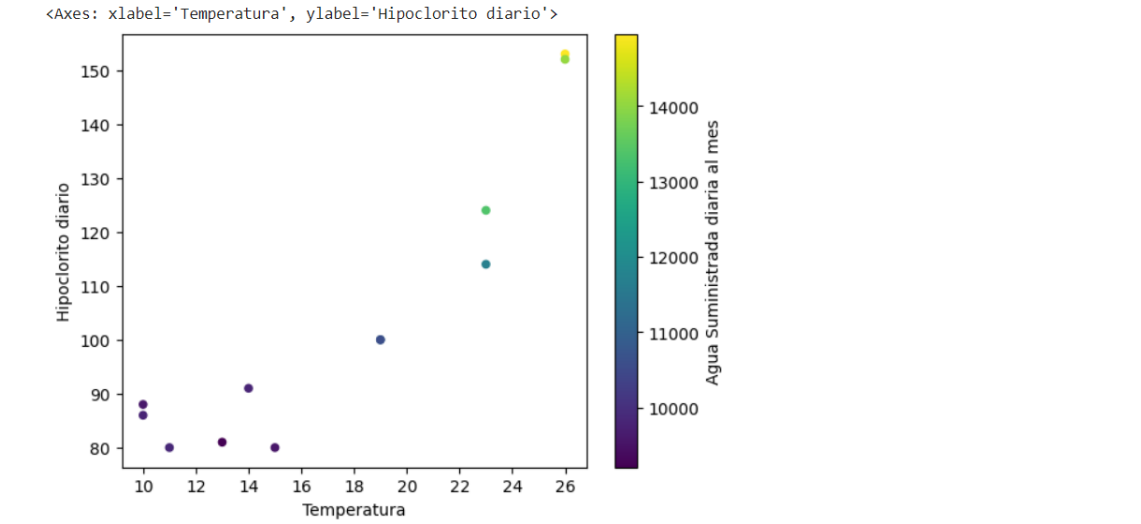


Gráfica 2: Temperatura vs agua suministrada vs precipitación en Paterna. Elaboración propia.

Como podemos observar, hay una clara relación entre la temperatura y el agua suministrada. Es una relación directamente proporcional, ya que a más temperatura, más agua es suministrada. En cuanto a la precipitación, de momento no vemos una relación clara.

Al realizar esta comparación en L'Eliana vemos que ocurre lo mismo en la relación de la temperatura y el agua. Por tanto, para ir más allá en este análisis previo de los datos, vamos a analizar el impacto de la temperatura y el agua en el consumo de hipoclorito, que es nuestra variable a predecir.

```
[ ] df.plot.scatter(x = 'Temperatura', y = 'Hipoclorito diario', c = 'Agua Suministrada diaria al mes', colormap = 'viridis')
```



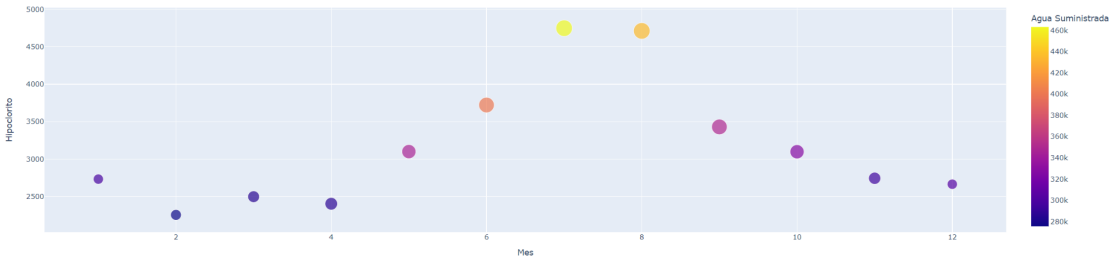
Gráfica 3: Temperatura vs hipoclorito consumido vs agua suministrada en L'Eliana. Elaboración propia.

Como era de esperar, a más agua suministrada, más hipoclorito es consumido y a más temperatura, más agua es suministrada y más hipoclorito es consumido.

De momento los datos están guardando una coherencia entre sí y se puede continuar analizando.

El último paso, es comprobar si hay una estacionalidad, es decir, si el tiempo supone una variable para el consumo de hipoclorito, la temperatura y el agua suministrada. En teoría, si la temperatura depende de las otras dos variables, en los meses de más calor, el agua suministrada y el hipoclorito consumido presentarán valores altos. Para ello, hemos dividido el dataset mensualmente y lo hemos graficado para comprobarlo.

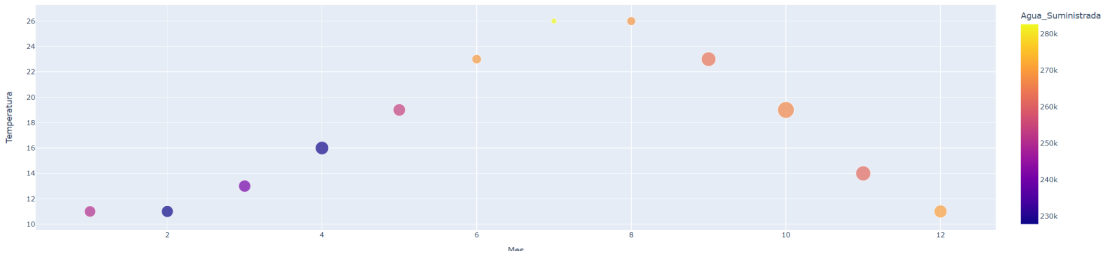
```
[ ] #Graficar de manera más precisa que la anterior, añadiendo una nueva variable.
    fig = px.scatter(df, x = 'Mes', y = 'Hipoclorito', size = 'Temperatura', color = 'Agua Suministrada')
    fig.show()
```



Gráfica 4: Hipoclorito vs agua suministrada por meses en L'Eliana. Elaboración propia.

```
[ ] #Añadimos una nueva variable a la gráfica

    fig = px.scatter(dataframe, x = 'Mes', y = 'Temperatura', size = 'Precipitación', color = 'Agua_Suministrada')
    fig.show()
```



Gráfica 5: Temperatura vs agua suministrada por meses en Paterna. Elaboración propia.

3.3.3 Carga, limpieza y preparación de los datos

Google Colab nos permite trabajar con infinidad de librerías para el tratamiento de los datos. En este apartado se expone cómo se han cargado los datos a nuestro modelo y cómo se han adaptado para poder ser analizadas en profundidad y de una manera óptima y práctica.

Una vez obtenidos los datos, se crea un .csv a través de una hoja de cálculo y se carga dentro de nuestro modelo predictivo, como se observa en la siguiente figura.

```
#Carga de datos
#=============================================================================
from google.colab import files
uploaded = files.upload()

import io
datos = pd.read_csv(io.BytesIO(uploaded['Forecasting_consumo_Eliana.csv']))
print(datos)
```

```
Elegir archivos  Forecastin..._Eliana.csv
• Forecasting_consumo_Eliana.csv(text/csv) - 24146 bytes, last modified: 31/5/2023 - 100% done
Saving Forecasting_consumo_Eliana.csv to Forecasting_consumo_Eliana.csv
          Fecha  Agua  Hipoclorito  Temperatura
0     01-12-2019  5693           98         11.3
1     02-12-2019  5693           49         10.9
2     03-12-2019  6269           74         12.1
3     04-12-2019  6743          123         11.3
4     05-12-2019  4505          116          9.8
...          ...   ...          ...          ...
1122  27-12-2022  7814           98          7.3
1123  28-12-2022  7618           74          7.5
1124  29-12-2022  7382           98          7.3
1125  30-12-2022  7570           74          7.2
1126  31-12-2022  7524           98          6.7

[1127 rows x 4 columns]
```

Tabla 4: Carga de datos al modelo. Elaboración propia.

La ventaja de cargar nuestro .csv es que puede analizarse fácilmente y de manera muy intuitiva, además de tener un amplio espectro para mostrar gráficos y diagramas a través de las librerías que proporciona Google Colab.

El siguiente paso a realizar en nuestro set de datos es la preparación y la limpieza. Lo primero de todo es cambiar la columna "Fecha" a un dato del tipo *'DateTimeIndex'*, ya que por defecto se marca como un *string* y no es la manera adecuada de analizar este dato. También vamos a crear en nuestro dataset un modelo de codificación *'One Hot Encoding'* que nos diga el día de la semana en el que nos encontramos marcando con un 1 el día que es y con un 0 los demás días. Así quizás podamos observar si hay un consumo mayor o menor en un determinado día o por ejemplo el fin de semana.

```
[4]  #Conversión del formato fecha
     #==============================================================================
     datos['Time'] = pd.to_datetime(datos['Fecha'], format = '%d-%m-%Y')
     datos = datos.set_index('Time')
```

```
[5]  #Elimino la columna fecha para que no de error al agregar
     #==============================================================================
     datos = datos.drop(columns = 'Fecha')
     datos = datos.resample(rule = 'H', closed = 'left', label = 'right').mean()
```

```
     #Creación de una variable que indique si es fin de semana
     #==============================================================================
     datos['dia_semana'] = datos.index.dayofweek
     datos = pd.get_dummies(datos, columns = ['dia_semana'])
```

Figura 23: Conversión a *'DateTimeIndex'* y creación del *'OneHotEncoding'*. Elaboración propia.

La generación del formato *'DateTimeIndex'* añade las 24 horas del día a todos los días del dataset. Esto supone un problema debido a que el sensor no tiene registro del consumo de hipoclorito de todas las horas del día y por tanto todas estas filas que se han añadido contienen conjuntos vacíos o *'NaN'*. A este problema se le suman lecturas erróneas del sensor, como valores de nivel 0 los cuales son imposibles. Como observamos en la siguiente figura, de esta manera eliminamos los datos que no son válidos para la interpretación y que contaminan nuestro análisis.

```
[7]  datos.head(5)
```

Time	Agua	Hipoclorito	Temperatura	dia_semana_0	dia_semana_1	dia_semana_2	dia_semana_3	dia_semana_4	dia_semana_5	dia_semana_6
2019-12-01 01:00:00	5693.0	98.0	11.3	0	0	0	0	0	0	1
2019-12-01 02:00:00	NaN	NaN	NaN	0	0	0	0	0	0	1
2019-12-01 03:00:00	NaN	NaN	NaN	0	0	0	0	0	0	1
2019-12-01 04:00:00	NaN	NaN	NaN	0	0	0	0	0	0	1
2019-12-01 05:00:00	NaN	NaN	NaN	0	0	0	0	0	0	1

```
[ ]  #Eliminar los conjuntos vacios
     #==============================================================================
     datos = datos.dropna()
```

Tabla 5: *Dataset* con *'NaN'* y código para eliminarlo. Elaboración propia.

Ahora tenemos un conjunto de datos listo para ser analizado en profundidad y empezar nuestra predicción. Se ha cargado el .csv con nuestras variables principales de interés y se ha introducido el día de la semana mediante codificación *'One Hot Encoding'*, hemos

cambiado el formato de la fecha ya que siendo un *string* no se puede analizar de manera correcta y hemos solucionado el problema de la creación de conjuntos vacíos y lecturas erróneas del sensor.

De esta manera, así queda nuestro *dataset* a la espera de ser analizado.

```
datos
```

Time	Agua	Hipoclorito	Temperatura	dia_semana_0	dia_semana_1	dia_semana_2	dia_semana_3	dia_semana_4	dia_semana_5	dia_semana_6
2019-12-01 01:00:00	5693.0	98.0	11.3	0	0	0	0	0	0	1
2019-12-02 01:00:00	5693.0	49.0	10.9	1	0	0	0	0	0	0
2019-12-03 01:00:00	6269.0	74.0	12.1	0	1	0	0	0	0	0
2019-12-04 01:00:00	6743.0	123.0	11.3	0	0	1	0	0	0	0
2019-12-05 01:00:00	4505.0	116.0	9.8	0	0	0	1	0	0	0
2022-12-27 01:00:00	7814.0	98.0	7.3	0	1	0	0	0	0	0
2022-12-28 01:00:00	7618.0	74.0	7.5	0	0	1	0	0	0	0
2022-12-29 01:00:00	7382.0	98.0	7.3	0	0	0	1	0	0	0
2022-12-30 01:00:00	7570.0	74.0	7.2	0	0	0	0	1	0	0
2022-12-31 01:00:00	7524.0	98.0	6.7	0	0	0	0	0	1	0

Tabla 6: *Dataset* final con todas las modificaciones pertinentes. Elaboración propia.

3.3.4 Separación de los datos en entrenamiento, validación y test

Para poder calibrar si un modelo funciona, necesitamos probarlo con un conjunto de datos diferente. Por ello, en todo proceso de *Machine Learning*, los datos de trabajo se dividen en dos partes: datos de entrenamiento y datos de prueba o test.

En este modelo en concreto se ha querido añadir además un tercer conjunto de datos entre el entrenamiento y el test. El conjunto de datos de validación, que será el encargado de seleccionar el mejor de los modelos entrenados. Se puede observar en la siguiente figura como se ha procedido a la división de los datos, empleando aproximadamente un 70% para entrenamiento, un 20% para validación y el 10% restante para el test.

```
[ ]  #Separación datos entrenamiento-validación-test
     #================================================================================
     datos = datos.loc['2019-12-01 01:00:00' : '2022-12-31 01:00:00']
     fin_train = '2021-12-31 00:59:00'
     fin_validacion = '2022-09-30 00:59:00'
     datos_train = datos.loc[: fin_train, :] # 2 años y un mes
     datos_val = datos.loc[fin_train:fin_validacion, :] # 9 meses
     datos_test = datos.loc[fin_validacion:, :] # 3 meses

     print(f"Fechas train      : {datos_train.index.min()} --- {datos_train.index.max()} (n={len(datos_train)})")
     print(f"Fechas validacion : {datos_val.index.min()}   --- {datos_val.index.max()}   (n={len(datos_val)})")
     print(f"Fechas test       : {datos_test.index.min()}  --- {datos_test.index.max()}  (n={len(datos_test)})")

     Fechas train      : 2019-12-01 01:00:00 --- 2021-12-30 01:00:00 (n=761)
     Fechas validacion : 2021-12-31 01:00:00   --- 2022-09-29 01:00:00   (n=273)
     Fechas test       : 2022-09-30 01:00:00  --- 2022-12-31 01:00:00  (n=93)
```

Figura 24: Separación de los datos en entrenamiento, validación y test. Elaboración propia.

Los datos de entrenamiento son los que usamos para entrenar el modelo. La calidad de nuestro modelo de aprendizaje automático va a ser directamente proporcional a la calidad de los datos, por ello la limpieza de los datos realizada anteriormente es primordial para un buen análisis.

Los datos de prueba son los datos que nos guardamos para comprobar si el modelo que hemos generado a partir de los datos de entrenamiento funciona. Es decir, si las respuestas predichas por el modelo para un caso totalmente nuevo son acertadas o no. Es importante que el conjunto de datos de prueba tenga un volumen suficiente como para generar resultados estadísticamente significativos, y a la vez, que sea representativo del conjunto de datos global.

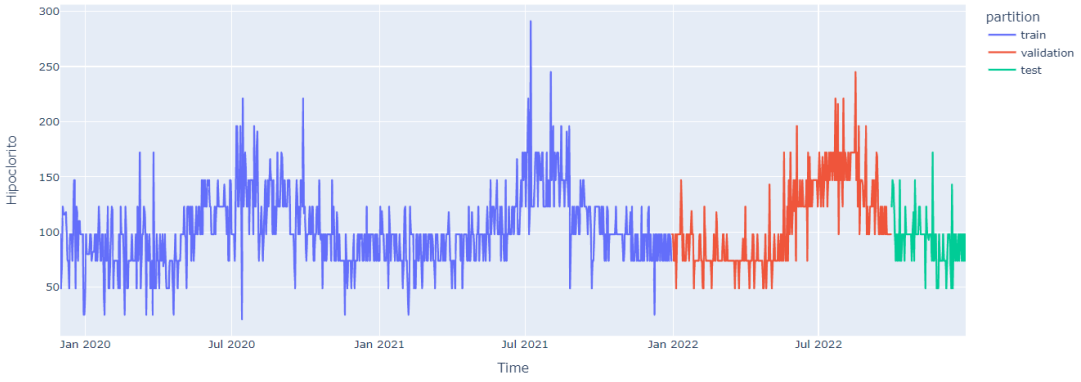
A la hora de la división de los datos surge un problema muy común llamado sobreajuste u *"overfitting"*. Esto ocurre cuando un modelo está sobreentrenado y no puede ser exportado a otros modelos, restándole así una gran utilidad. También podemos encontrarnos justo con lo contrario, subajuste o *"underfitting"*, que ocurre cuando los datos de entrenamiento son insuficientes y en consecuencia tienen muy poco valor predictivo.

A continuación, se muestra el gráfico con la división del conjunto de los datos, donde se observa una gran diferencia en el consumo en los meses de verano con respecto al resto.

```
[ ] #Gráfico interactivo de la serie temporal
    #================================================================================
    datos.loc[:fin_train, 'partition'] = 'train'
    datos.loc[fin_train:fin_validacion, 'partition'] = 'validation'
    datos.loc[fin_validacion:, 'partition'] = 'test'

    fig = px.line(
        data_frame = datos.reset_index(),
        x          = 'Time',
        y          = 'Hipoclorito',
        color      = 'partition',
        width      = 1200,
        height     = 500
    )

    fig.show()
    datos = datos.drop(columns = 'partition')
```



Gráfica 6: Separación del entrenamiento(azul), validación(rojo) y test(verde). Elaboración propia.

4. Elaboración del algoritmo predictivo

Una serie temporal es una sucesión de datos ordenados cronológicamente. Para este tipo de conjuntos de datos, lo mejor es emplear un proceso de *forecasting* o una red neuronal recurrente.

En este apartado, vamos a analizar y predecir valores con ambos modelos y vamos a determinar cuál se ajusta más a la realidad a través de métricas de evaluación.

4.1 *Forecasting*

El *forecasting* consiste en predecir el valor futuro de una serie temporal, modelando la serie en función de su comportamiento pasado.

Para la realización de este algoritmo se emplea la librería de *scikit-learn*, más en concreto de *skforecast*, la cual contiene las clases y funciones necesarias para adaptar cualquier modelo de regresión de *scikit-learn* a problemas de *forecasting*.

La principal adaptación que se necesita hacer para aplicar modelos de *Machine Learning* a problemas de *forecasting* es transformar la serie temporal en una matriz en la que cada valor está asociado a la ventana temporal (*lags*) que le precede. Una vez que los datos se encuentran ordenados de esta forma, se puede entrenar cualquier modelo de regresión para que aprenda a predecir el siguiente valor de la serie.

Sin embargo, en nuestro caso no se quiere predecir solo el siguiente elemento de la serie, sino todo intervalo futuro. Por ello, se emplea un proceso recursivo donde cada nueva predicción hace uso de la predicción anterior. A este proceso se le conoce como *'ForecasterAutoreg'* y es el que vamos a emplear para nuestro algoritmo predictivo [23].

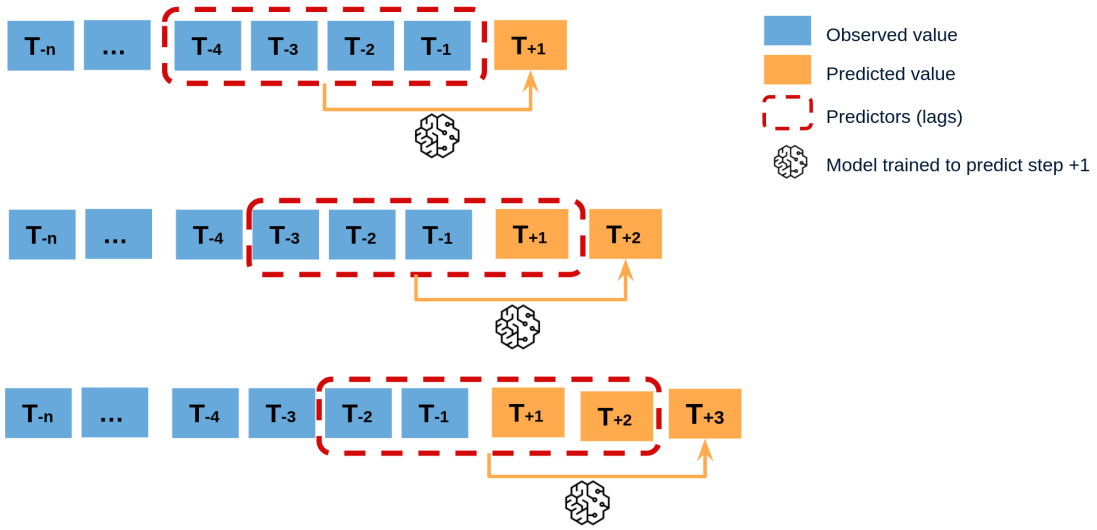


Figura 25: Diagrama teórico donde se predicen 3 valores futuros utilizando 4 *lags* anteriores como predictores [23].

4.1.1 *ForecasterAutoreg*

Para crear nuestras predicciones del consumo de hipoclorito, se crea y entrena un modelo *ForecasterAutoreg* a partir de un regresor *LGBMRegressor* y una ventana temporal de 24 *lags*. Esto último significa que, el modelo, utiliza como predictores los 24 días anteriores.

```
#Entrenamiento del forecaster
#Crear y entrenar forecaster
#=======================================================================
forecaster = ForecasterAutoreg(
    regressor = LGBMRegressor(max_depth = 3, learning_rate = 0.07, n_estimators = 20),
    lags = 24
)
forecaster.fit(y = datos.loc[:fin_validacion, 'Hipoclorito']) #Entrenamiento con conjuntos de train + validación
forecaster
```

```
================
ForecasterAutoreg
================
Regressor: LGBMRegressor(learning_rate=0.07, max_depth=3, n_estimators=20)
Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
Transformer for y: None
Transformer for exog: None
Window size: 24
Weight function included: False
Exogenous included: False
Type of exogenous variable: None
Exogenous variables names: None
Training range: [Timestamp('2019-12-01 01:00:00'), Timestamp('2022-09-29 01:00:00')]
Training index type: DatetimeIndex
Training index frequency: 24H
Regressor parameters: {'boosting_type': 'gbdt', 'class_weight': None, 'colsample_bytree': 1.0, 'importance_type':
'num_leaves': 31, 'objective': None, 'random_state': None, 'reg_alpha': 0.0, 'reg_lambda': 0.0, 'silent': 'warn',
fit_kwargs: {}
Creation date: 2023-05-31 11:42:30
Last fit date: 2023-05-31 11:42:30
Skforecast version: 0.8.1
Python version: 3.10.11
Forecaster id: None
```

Figura 26: Creación, entrenamiento y ajuste del regresor del *forecaster*. Elaboración propia.

4.1.2 Predicciones

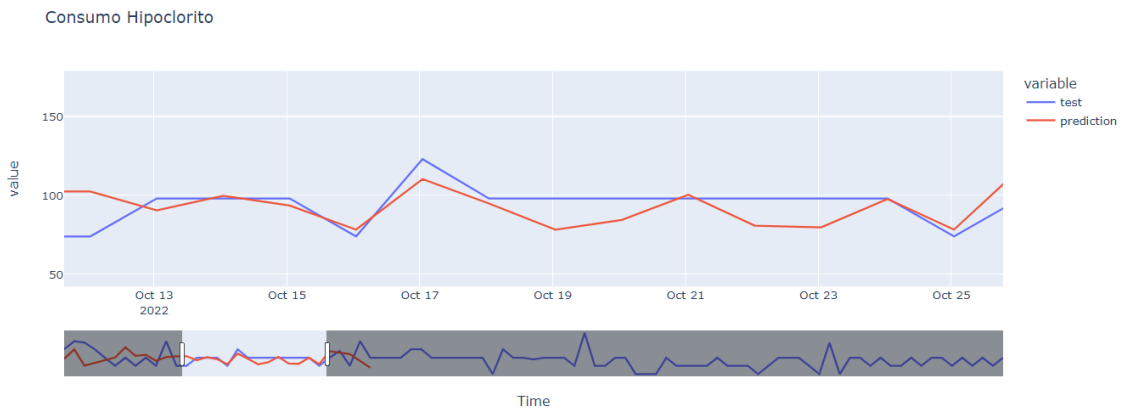
Una vez entrenado el modelo, se predicen los datos de test con un *step* = 31, lo que significa que vamos a predecir 31 días a futuro.

```
#Predicciones
#=======================================================================
predicciones = forecaster.predict(steps = 31)
predicciones.head()
```

```
2022-09-30 01:00:00     94.487520
2022-10-01 01:00:00    122.541032
2022-10-02 01:00:00     74.235974
2022-10-03 01:00:00     81.926779
2022-10-04 01:00:00     89.112659
Freq: 24H, Name: pred, dtype: float64
```

Figura 27: Código para realizar la predicción del mes siguiente. Elaboración propia.

```
#Gráfico interactivo de predicciones
#==============================================================================
datos_plot = pd.DataFrame({
    'test': datos_test['Hipoclorito'],
    'prediction': predicciones,
})
datos_plot.index.name = 'Time'
fig = px.line(
    data_frame = datos_plot.reset_index(),
    x          = 'Time',
    y          = datos_plot.columns,
    title      = 'Consumo Hipoclorito',
    width      = 1200,
    height     = 500
)
fig.update_xaxes(rangeslider_visible = True)
fig.show()
```



Gráfica 7: Código de creación del gráfico y gráfico de predicción. Elaboración propia.

De momento la predicción parece bastante acertada, vamos a proceder a implantar un modelo de *backtesting* y un ajuste de hiperparámetros para poder comparar los resultados y observar si mejora.

4.1.3 *Backtesting*

El proceso de *backtesting* consiste en evaluar el comportamiento de un modelo predictivo al aplicarlo de forma retrospectiva sobre datos históricos. Por lo tanto, es una estrategia de validación que permite cuantificar la capacidad predictiva de un modelo.

Es importante no incluir los datos de test en el proceso de búsqueda puesto que se pueden caer en el problema del *'overfitting'* mencionado anteriormente.

Hay varios tipos de *backtesting*, para este trabajo se sigue una estrategia de *backtesting* con reentrenamiento. Con esto conseguimos que toda la información que tenemos hasta el momento se incorpore dentro del modelo y en lugar de hacer un reparto aleatorio de las observaciones, el conjunto de entrenamiento se incrementa de manera secuencial manteniendo el orden de los datos.

Emplear este tipo de técnica sobre los datos introducidos dentro de una serie temporal aporta mucha rapidez y eficacia a la hora de realizar predicciones del modelo [24].

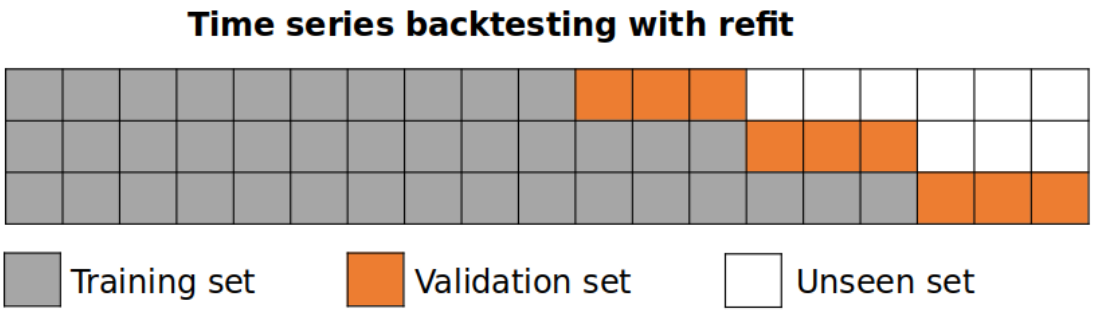


Figura 28: Diagrama de *backtesting* con entrenamiento de 10 observaciones, 3 *steps* de predicción y reentrenamiento en cada iteración [24].

```
#Backtesting
#=======================================================================
metrica, predicciones = backtesting_forecaster(
    forecaster         = forecaster,
    y                  = datos['Hipoclorito'],
    initial_train_size = len(datos.loc[:fin_validacion]),
    steps              = 24,
    metric             = 'mean_absolute_error',
    refit              = False,
    verbose            = True
)
```

```
Information of backtesting process
----------------------------------
Number of observations used for initial training: 1034
Number of observations used for backtesting: 93
    Number of folds: 4
    Number of steps per fold: 24
    Number of steps to exclude from the end of each train set before test (gap): 0
    Last fold only includes 21 observations.

Fold: 0
    Training:   2019-12-01 01:00:00 -- 2022-09-29 01:00:00  (n=1034)
    Validation: 2022-09-30 01:00:00 -- 2022-10-23 01:00:00  (n=24)
Fold: 1
    Training:   2019-12-01 01:00:00 -- 2022-09-29 01:00:00  (n=1034)
    Validation: 2022-10-24 01:00:00 -- 2022-11-16 01:00:00  (n=24)
Fold: 2
    Training:   2019-12-01 01:00:00 -- 2022-09-29 01:00:00  (n=1034)
    Validation: 2022-11-17 01:00:00 -- 2022-12-10 01:00:00  (n=24)
Fold: 3
    Training:   2019-12-01 01:00:00 -- 2022-09-29 01:00:00  (n=1034)
    Validation: 2022-12-11 01:00:00 -- 2022-12-31 01:00:00  (n=21)

100% 4/4 [00:02<00:00, 1.55it/s]
```

```
print(f"Backtest error: {metrica}")
```

```
Backtest error: 23.130511101652196
```

Figura 29: Código y ejecución del *backtesting* con la métrica de error. Elaboración propia.

Como observamos, nuestra predicción tiene un error de *backtesting* de aproximadamente 23 unidades. Esto significa que la predicción difiere de la realidad en 23 kilos de hipoclorito de media.

Ahora se va a realizar un ajuste de hiperparámetros en el modelo para intentar reducir este error y conseguir una predicción más acertada.

4.1.4 Ajuste de hiperparámetros

Para el *'ForecasterAutoreg'* entrenado en nuestro modelo, se ha empleado una ventana temporal de 24 *lags* y un modelo *LGBMRegressor* con los hiperparámetros por defecto. No hay ninguna razón por las que estos valores sean los más adecuados.

Para identificar la mejor combinación de lags e hiperparámetros, la librería *skforecast* dispone de la función *'grid_search_forecaster'* con la que comparar los resultados obtenidos con cada configuración del modelo.

```
#Grid search de hiperparámetros
#==============================================================================
#Hiperparámetros del regresor
param_grid = {
    'max_iter'      : [100, 500],
    'max_depth'     : [3, 10],
    'learning_rate' : [0.01, 0.1]
}

#lags utilizados como predictores
lags_grid = [24, 48, [1, 2, 24]]

resultados_grid = grid_search_forecaster(
    forecaster         = forecaster,
    y                  = datos.loc[:fin_validacion, 'Hipoclorito'], #Conjunto de train + validación
    param_grid         = param_grid,
    lags_grid          = lags_grid,
    steps              = 24,
    refit              = False,
    metric             = 'mean_absolute_error',
    initial_train_size = int(len(datos_train)), #El modelo se entrena con los datos de entrenamiento
    return_best        = True,
    verbose            = False
)
```

```
`Forecaster` refitted using the best-found lags and parameters, and the whole data set:
  Lags: [ 1  2 24]
  Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'max_iter': 500}
  Backtesting metric: 20.88756853532052
```

Figura 30: Código y ejecución de la función *'grid_search_forecaster'*. Elaboración propia.

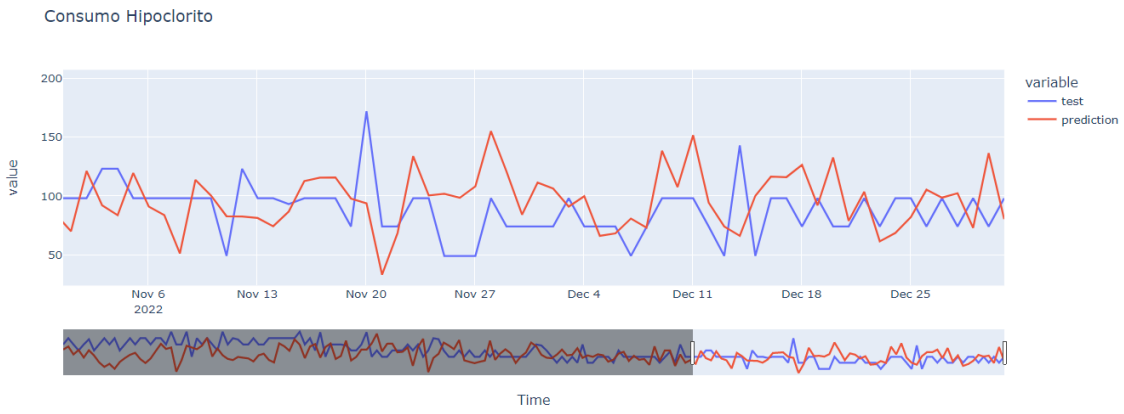
```
#Resultados Grid Search
#==============================================================================
resultados_grid.head(10)
```

	lags	params	mean_absolute_error	learning_rate	max_depth	max_iter
17	[1, 2, 24]	{'learning_rate': 0.01, 'max_depth': 3, 'max_i...	20.887569	0.01	3.0	500.0
9	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	{'learning_rate': 0.01, 'max_depth': 3, 'max_i...	21.009109	0.01	3.0	500.0
20	[1, 2, 24]	{'learning_rate': 0.1, 'max_depth': 3, 'max_it...	21.232976	0.10	3.0	100.0
21	[1, 2, 24]	{'learning_rate': 0.1, 'max_depth': 3, 'max_it...	21.296647	0.10	3.0	500.0
1	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	{'learning_rate': 0.01, 'max_depth': 3, 'max_i...	21.319009	0.01	3.0	500.0
12	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	{'learning_rate': 0.1, 'max_depth': 3, 'max_it...	22.144896	0.10	3.0	100.0
14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	{'learning_rate': 0.1, 'max_depth': 10, 'max_i...	22.617435	0.10	10.0	100.0
11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	{'learning_rate': 0.01, 'max_depth': 10, 'max_...	22.750917	0.01	10.0	500.0
3	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	{'learning_rate': 0.01, 'max_depth': 10, 'max_...	23.120145	0.01	10.0	500.0
23	[1, 2, 24]	{'learning_rate': 0.1, 'max_depth': 10, 'max_i...	23.829279	0.10	10.0	500.0

Tabla 7: Resultados de la función *'grid_search_forecaster'*. Elaboración propia.

Como podemos observar en las dos figuras anteriores, los mejores resultados para nuestro modelo se van a obtener utilizando una ventana temporal de 24 *lags* y una configuración de *LGBMRegressor{'learning_rate' : 0.01, 'max_depth' : 3, 'max_iter' : 500}*.

En esta gráfica se observa la predicción y la realidad de los meses de noviembre y diciembre.



Gráfica 8: Realidad(azul) vs predicción(rojo) de los meses de noviembre y diciembre. Elaboración propia.

Por último, en estas dos tablas se puede observar el valor de predicción, realidad y diferencia de cada día de noviembre y diciembre y sus totales mensuales.

De esta manera se puede comprobar que nuestro modelo se ajusta a la realidad y el error de *backtesting* medido es el correcto.

	Noviembre predicción	Noviembre real	Diferencia		Diciembre predicción	Diciembre real	Diferencia
Día 1	100,14	98	2,14	Día 1	83,73	74	9,73
Día 2	100	98	2	Día 2	78,79	74	4,79
Día 3	109,51	123	-13,49	Día 3	67,39	98	-30,61
Día 4	87,55	123	-35,45	Día 4	94	74	20
Día 5	122,49	98	24,49	Día 5	91,61	74	17,61
Día 6	102,82	98	4,82	Día 6	60,17	74	-13,83
Día 7	95,76	98	-2,24	Día 7	107,26	49	58,26
Día 8	44,25	98	-53,75	Día 8	109,54	74	35,54
Día 9	103	98	5	Día 9	97,8	98	-0,2
Día 10	100,8	98	2,8	Día 10	98,47	98	0,47
Día 11	61,96	49	12,96	Día 11	57,1	98	-40,9
Día 12	67,39	123	-55,61	Día 12	36,59	74	-37,41
Día 13	73	98	-25	Día 13	81,43	49	32,43
Día 14	48,45	98	-49,55	Día 14	50,13	143	-92,87
Día 15	96,06	93	3,06	Día 15	76,41	49	27,41
Día 16	86,48	98	-11,52	Día 16	91,89	98	-6,11
Día 17	121,35	98	23,35	Día 17	92,39	98	-5,61
Día 18	82,93	98	-15,07	Día 18	117	74	43
Día 19	72	74	-2	Día 19	119,1	98	21,1
Día 20	109,31	172	-62,69	Día 20	115,77	74	41,77
Día 21	127,94	74	53,94	Día 21	65,43	74	-8,57
Día 22	76,07	74	2,07	Día 22	109,49	98	11,49
Día 23	71,29	98	-26,71	Día 23	48,07	74	-25,93
Día 24	82,62	98	-15,38	Día 24	41,8	98	-56,2
Día 25	84,51	49	35,51	Día 25	67,5	98	-30,5
Día 26	89	49	40	Día 26	91,55	74	17,55
Día 27	111,75	49	62,75	Día 27	65,37	98	-32,63
Día 28	126	98	28	Día 28	73,68	74	-0,32
Día 29	117,67	74	43,67	Día 29	94,43	98	-3,57
Día 30	66	74	-8	Día 30	96,22	74	22,22
				Día 31	99	98	1
	Total predicción = 2738,12	Total real = 2768	Diferencia = 29,88		Total predicción = 2579,11	Total real = 2600	Diferencia = 20,89

Tabla 8: Valores predichos y reales diarios y totales mensuales de consumo de hipoclorito en noviembre y diciembre. Elaboración propia.

En conclusión, con este modelo ajustado de la manera que se ha explicado anteriormente y analizando su error de *backtesting*, la predicción realizada difiere de la realidad en aproximadamente 21 unidades en promedio. La predicción no se ajusta del todo a la realidad aunque su error tampoco es extremadamente alto y es un modelo fiable.

4.2 Red neuronal recurrente

Con el fin de comprobar si el modelo se podría ajustar mejor a la realidad, se ha decidido emplear otra técnica de *Machine Learning* distinta al *forecasting* y una métrica de evaluación diferente al *backtesting*.

Vamos a implementar una Red neuronal recurrente, donde a diferencia del *forecasting*, nos vamos a centrar en el nivel del depósito más que en el consumo de hipoclorito.

La obtención de los datos, el análisis previo, la limpieza, la carga y la separación del conjunto en datos de entrenamiento, validación y test es exactamente la misma que la empleada en el modelo de *forecasting*.

Dicho esto y partiendo de la misma base utilizada anteriormente, nos disponemos a la creación y ejecución de este modelo.

4.2.1 Escalado de los datos

Para que funcionen mejor, en muchos algoritmos de *Machine Learning* se emplea el escalado de los datos para normalizar las variables de entrada. Normalizar significa comprimir o extender los valores de las variables predictivas para que estén en un rango definido. Sin embargo, una mala normalización o un uso inadecuado del método de normalización puede hacer que los datos ya no sean correctos para el modelo predictivo.

El método de escalado de los datos para realizar la Red neuronal recurrente ha sido el *'MinMaxScaler'* de la librería *'scikit-learn'*. La razón por la que se emplea este método es porque los datos de origen se reajustan a un mínimo de 0 y a un máximo de 1, lo cual a la hora del análisis y el procesado hace que el modelo sea más sencillo y más preciso para predecir. No obstante, si nuestro *set* de datos contiene ruido o valores erróneos, al ser comprimido, estos serían ampliados dando resultados imprecisos en nuestro algoritmo predictivo.

En la siguiente figura podemos observar como se ha creado la función con todos los *lags* temporales anteriores y como mediante el escalado *'MinMaxScaler'* se han comprimido los valores entre 0 y 1.

```
PASOS = 7

def series_to_supervised(data, n_in = 1, n_out = 1, dropnan = True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data)
  cols, names = list(), list()

  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]

  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:
      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]

    agg = pd.concat(cols, axis = 1)
    agg.columns = names

    if dropnan:
      agg.dropna(inplace = True)
    return agg

values = df.values
values = values.astype('float32')
scaler = MinMaxScaler (feature_range = (-1, 1))
values = values.reshape(-1,1)
scaled = scaler.fit_transform(values)
reframed = series_to_supervised(scaled, PASOS, 1)
reframed.head()
```

	var1(t-7)	var1(t-6)	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
7	0.984068	0.909304	0.812789	0.731792	0.631529	0.553639	0.478931	0.416634
8	0.909304	0.812789	0.731792	0.631529	0.553639	0.478931	0.416634	0.320101
9	0.812789	0.731792	0.631529	0.553639	0.478931	0.416634	0.320101	0.226636
10	0.731792	0.631529	0.553639	0.478931	0.416634	0.320101	0.226636	0.154358
11	0.631529	0.553639	0.478931	0.416634	0.320101	0.226636	0.154358	0.057787

Tabla 9: Función y escalado de los datos. Elaboración propia.

4.2.2 Creación del modelo

La Red neuronal recurrente se crea utilizando la librería *'Keras'* explicada en nuestro marco conceptual. A la hora de la realización de la predicción con esta librería se tienen que tener en cuenta el modelo de *'Keras'*, las capas y la función de activación a emplear.

En cuanto al modelo, vamos a utilizar el modelo *'Sequential'*, el cual crea una pila lineal de capas de entrada las cuales se describen de manera muy sencilla, ya que cada definición de la capa nada más requiere una línea de código.

En cuanto a las capas, se emplea la capa de modo *'Dense'*, ya que tiene la capacidad de interconectar los datos de entrada con cada valor de la capa oculta. Así todos los datos de todas las capas ocultas se iteran entre sí de forma que conseguimos una mayor precisión en nuestro algoritmo.

El siguiente paso es elegir la función de activación. Las más comunes para este tipo de red son la activación sigmoidal, la tangente hiperbólica y la *ReLU* (Rectified Lineal Unit) [25].

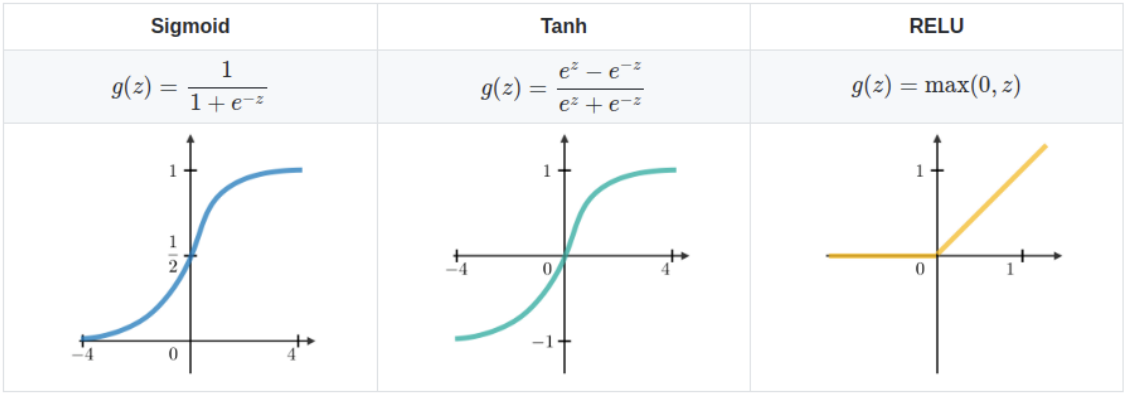


Figura 31: Comparación de las funciones de activación sigmoidal, tangente hiperbólica y *ReLU [25]*.

Como se observa la función sigmoidal y la tangente hiperbólica son muy parecidas en forma, pero la sigmoide corta el eje de coordenadas en 0.5 y no presenta valores negativos, al contrario de la tangente hiperbólica que corta el eje en 0 y presenta valores negativos.

Sin embargo, la más común y la que hemos utilizado nosotros en este modelo es la *ReLU*. Esta función coloca los *inputs* negativos a 0 mientras que devuelve los *inputs* positivos de manera lineal. Esto acelera en gran medida el proceso para modelos complicados y con gran cantidad de información sin comprometer su eficacia. Es por ello que esta función se ha instaurado en el mundo del *Machine Learning* y es la más implementada a la hora de la creación de Redes neuronales.

Por último, se ha querido añadir en este modelo la instrucción *'Flatten'*, con este recurso que también nos proporciona la librería de *'Keras'*, podemos convertir nuestros

datos de entrada de una matriz a un *array* plano. De esta manera conseguimos eficiencia y sencillez a la hora de tratar y analizar los datos.

En la siguiente imagen del código realizado para la creación de la Red neuronal recurrente, se pueden observar todos los parámetros mencionados anteriormente y su posterior ejecución.

```
def crear_modelo():
  model = Sequential()
  model.add(Dense(PASOS, input_shape = (1,PASOS), activation = 'tanh'))
  model.add(Flatten())
  model.add(Dense(1, activation = 'tanh'))
  model.compile(loss = 'mean_absolute_error', optimizer = 'Adam', metrics = ["mse"])
  model.summary()
  return model
```

Figura 32: Creación de la Red neuronal recurrente con todos los parámetros mencionados. Elaboración propia.

Una vez creado el modelo y decididos los parámetros que se van a emplear, se ajustan las *'Epochs'*. Las *'Epochs'* son las "vueltas" que nuestro modelo de *Machine Learning* le va a dar a los datos para entrenar el algoritmo y realizar una predicción.

```
EPOCHS = 100

model = crear_modelo()

history = model.fit(x_train, y_train, epochs = EPOCHS, validation_data = (x_val, y_val), batch_size = PASOS)
```

Figura 33: Determinación de las *'epochs'* y ejecución del modelo. Elaboración propia.

Finalmente, podemos ver el resultado de la ejecución época a época y podemos analizar cómo se va reduciendo el error. Esta medida del error la explicaremos más adelante.

```
Epoch 1/100
25/25 [==============================] - 1s 10ms/step - loss: 0.5947 - mse: 0.5186 - val_loss: 0.6841 - val_mse: 0.6232
Epoch 2/100
25/25 [==============================] - 0s 3ms/step - loss: 0.5355 - mse: 0.4227 - val_loss: 0.6120 - val_mse: 0.4992
Epoch 3/100
25/25 [==============================] - 0s 3ms/step - loss: 0.4758 - mse: 0.3363 - val_loss: 0.5267 - val_mse: 0.3770
Epoch 4/100
25/25 [==============================] - 0s 3ms/step - loss: 0.4175 - mse: 0.2660 - val_loss: 0.4316 - val_mse: 0.2727
Epoch 5/100
25/25 [==============================] - 0s 3ms/step - loss: 0.3587 - mse: 0.2104 - val_loss: 0.3508 - val_mse: 0.2086
Epoch 6/100
25/25 [==============================] - 0s 3ms/step - loss: 0.3142 - mse: 0.1760 - val_loss: 0.2986 - val_mse: 0.1794
Epoch 7/100
25/25 [==============================] - 0s 3ms/step - loss: 0.2859 - mse: 0.1635 - val_loss: 0.2521 - val_mse: 0.1648
Epoch 8/100
25/25 [==============================] - 0s 4ms/step - loss: 0.2592 - mse: 0.1559 - val_loss: 0.2156 - val_mse: 0.1587
Epoch 9/100
25/25 [==============================] - 0s 4ms/step - loss: 0.2368 - mse: 0.1552 - val_loss: 0.1941 - val_mse: 0.1596
Epoch 10/100
25/25 [==============================] - 0s 3ms/step - loss: 0.2221 - mse: 0.1552 - val_loss: 0.1780 - val_mse: 0.1583
```

Figura 34: Épocas de la 1 a la 10 con su pérdida y su error. Elaboración propia.

Ahora vemos que en las últimas épocas la pérdida y el error se han reducido considerablemente.

```
Epoch 90/100
25/25 [==============================] - 0s 3ms/step - loss: 0.1304 - mse: 0.1294 - val_loss: 0.1448 - val_mse: 0.1543
Epoch 91/100
25/25 [==============================] - 0s 3ms/step - loss: 0.1301 - mse: 0.1306 - val_loss: 0.1461 - val_mse: 0.1555
Epoch 92/100
25/25 [==============================] - 0s 4ms/step - loss: 0.1300 - mse: 0.1301 - val_loss: 0.1448 - val_mse: 0.1545
Epoch 93/100
25/25 [==============================] - 0s 5ms/step - loss: 0.1302 - mse: 0.1301 - val_loss: 0.1453 - val_mse: 0.1550
Epoch 94/100
25/25 [==============================] - 0s 4ms/step - loss: 0.1311 - mse: 0.1303 - val_loss: 0.1455 - val_mse: 0.1552
Epoch 95/100
25/25 [==============================] - 0s 5ms/step - loss: 0.1300 - mse: 0.1302 - val_loss: 0.1466 - val_mse: 0.1548
Epoch 96/100
25/25 [==============================] - 0s 4ms/step - loss: 0.1301 - mse: 0.1300 - val_loss: 0.1470 - val_mse: 0.1552
Epoch 97/100
25/25 [==============================] - 0s 4ms/step - loss: 0.1294 - mse: 0.1304 - val_loss: 0.1440 - val_mse: 0.1545
Epoch 98/100
25/25 [==============================] - 0s 4ms/step - loss: 0.1294 - mse: 0.1294 - val_loss: 0.1453 - val_mse: 0.1553
Epoch 99/100
25/25 [==============================] - 0s 5ms/step - loss: 0.1296 - mse: 0.1300 - val_loss: 0.1437 - val_mse: 0.1543
Epoch 100/100
25/25 [==============================] - 0s 5ms/step - loss: 0.1292 - mse: 0.1298 - val_loss: 0.1450 - val_mse: 0.1549
```

Figura 35: Épocas de la 90 a la 100 con su pérdida y su error. Elaboración propia.

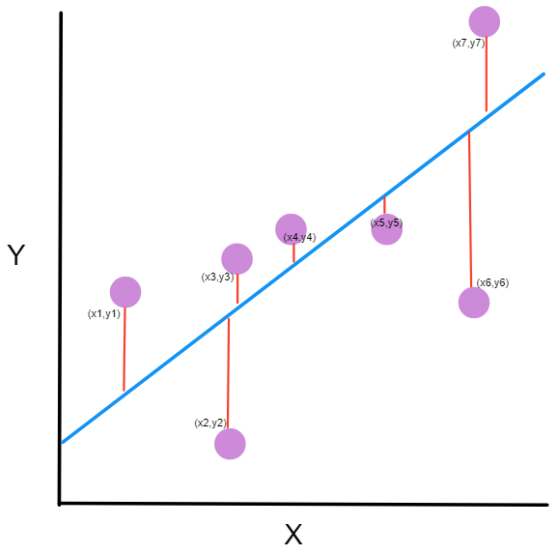
4.2.3 Pérdidas y precisión del modelo

Para todo modelo de *Machine Learning* y mucho más importante, para un algoritmo predictivo, se necesita saber si las estimaciones se ajustan a la realidad.

En el modelo anterior de *forecasting* empleamos la métrica *backtesting*, para este modelo vamos a utilizar un método mucho más empleado que es el error cuadrático medio, más comúnmente llamado en inglés *'mse'(mean squared error)*.

El error cuadrático medio de un estimador mide el promedio de los errores al cuadrado, es decir, la diferencia entre el estimador y lo que se estima [26].

En la siguiente figura lo podemos observar de manera gráfica.



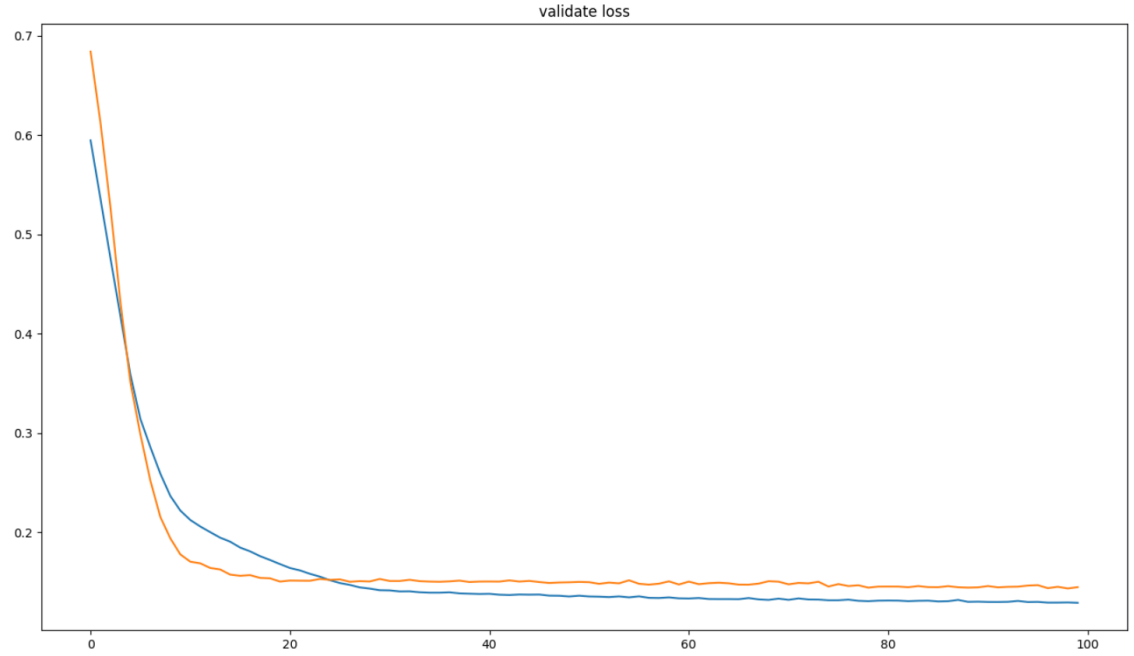
Gráfica 9: Gráfica *'mse' [26]*.

En nuestro caso, los puntos serían por ejemplo el nivel de hipoclorito en el depósito, la línea azul serían los valores predichos por el modelo y la línea roja por tanto el error cuadrático medio.

Ajustándose a nuestro modelo, podemos comprobar el error de manera gráfica como aparece en la siguiente imagen.

Aquí se puede comprobar el error que hay en nuestros datos reales de entrenamiento(línea azul) y el error que existe en nuestro conjunto de datos predichos(línea naranja).

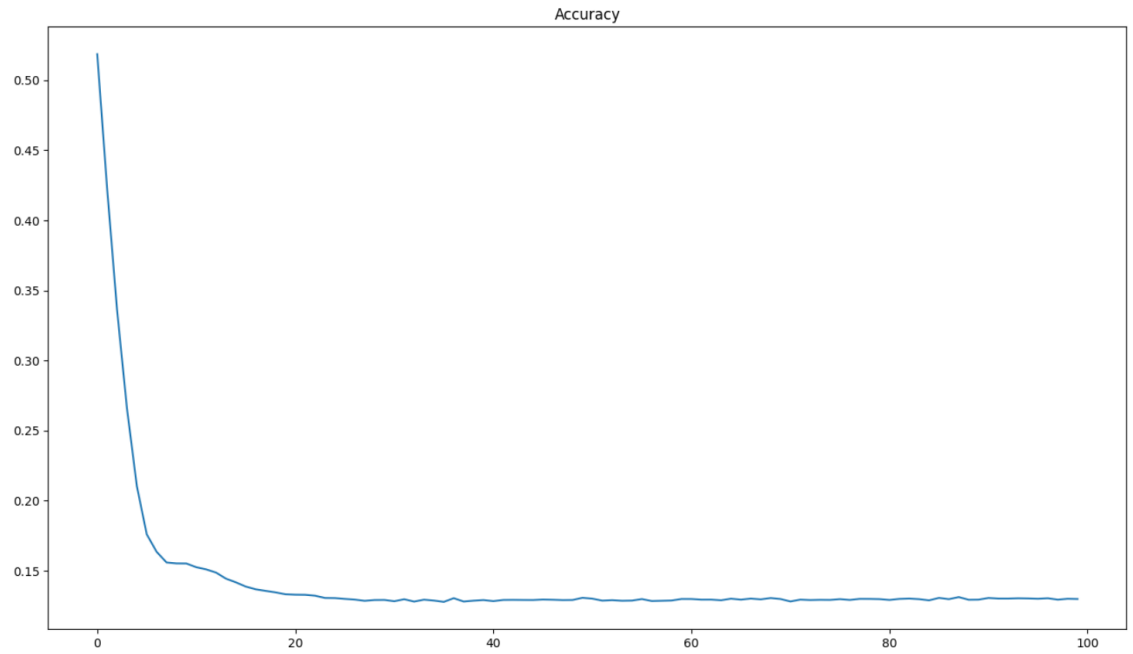
```
plt.plot(history.history['loss'])
plt.title('loss')
plt.plot(history.history['val_loss'])
plt.title('validate loss')
plt.show()
```



Gráfica 10: Gráfica de pérdidas en nuestro modelo. Elaboración propia.

Una vez observadas las pérdidas en el modelo en la gráfica anterior, se puede obtener la precisión que hemos obtenido del error cuadrático medio. En el eje x se observan las 100 épocas que nuestro modelo ha realizado y en el eje Y el valor del error.

```
plt.title('Accuracy')
plt.plot(history.history['mse'])
plt.show()
```



Gráfica 11: Gráfica de precisión en nuestro modelo. Elaboración propia.

4.2.4 Predicciones

Una vez analizado el error y visto que la pérdida del modelo con la realidad es significativamente poca, podemos adentrarnos en la predicción y compararlos con los datos reales.

Para crear el modelo y que los valores predichos se ajusten mucho más a la realidad, hemos comentado anteriormente que se han escalado los datos empleando la función *'MinMaxScaler'* de la librería *'scikit-learn'*.

A la hora de la predicción final, los valores no pueden estar en el rango de 0 y 1 puesto que esa compresión no la podríamos extrapolar a la realidad, por ello hay que volver a ajustar los valores comprimidos a su escala real.

En la siguiente figura, hemos escalado los valores y a parte hemos mostrado una tabla con una serie de valores predichos, reales y su diferencia.

```
compara = pd.DataFrame(np.array([y_val, [x[0] for x in results]])).transpose()
compara.columns = ['real', 'prediccion']

inverted = scaler.inverse_transform(compara.values)

compara2 = pd.DataFrame(inverted)
compara2.columns = ['real', 'prediccion']
compara2['diferencia'] = compara2['real'] - compara2['prediccion']

compara2.head()
```

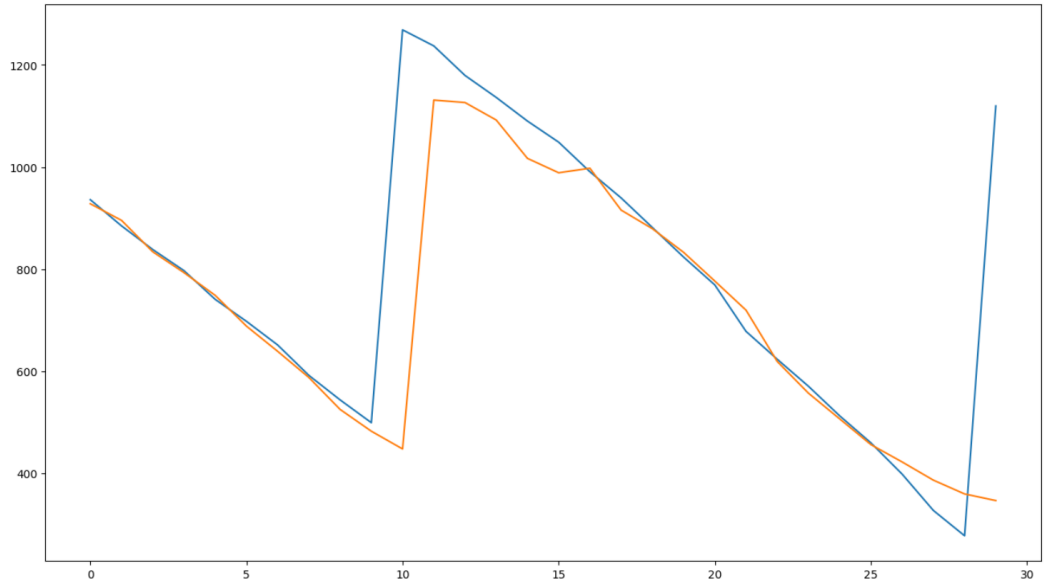
	real	prediccion	diferencia
0	936.119995	928.158936	7.961060
1	884.840088	895.634094	-10.794006
2	838.539978	834.107422	4.432556
3	797.179993	793.315186	3.864807
4	740.940002	748.772095	-7.832092

Tabla 10: Escalado original de los valores y tabla de predicción. Elaboración propia.

Ahora podemos mostrar gráficamente nuestra predicción y observar de manera visual si nuestro algoritmo predictivo está funcionando correctamente.

```
compara2['real'].plot()
compara2['prediccion'].plot()
```

```
<Axes: >
```



Gráfica 12: Gráfica de realidad(línea azul) vs predicción(línea naranja). Elaboración propia.

Se observa que la predicción del nivel de hipoclorito a lo largo de un mes se asemeja bastante a la realidad, aumentando su nivel prácticamente el mismo día que se repone esta sustancia en el depósito y disminuyendo de manera que se consume.

5. Conclusiones

Tras el desarrollo, ejecución y finalización del presente proyecto hay varias conclusiones que se pueden extraer.

La reposición de hipoclorito en la mayoría de depósitos de cada municipio no es un proceso digitalizado, más bien lo contrario. Desde la empresa se decide actuar en Paterna y L'Eliana debido a que hace unos años sus instalaciones cuentan con un sensor, que aunque presenta unos costes elevados, evita que un operario tenga que ir todos los días a comprobar el nivel de esta sustancia, como pasa en muchos municipios.

Gracias a este avance, se han podido extraer los datos a priori más relevantes en cuanto al consumo de hipoclorito. De todas formas, a la vista de los resultados, son insuficientes.

5.1 Discusión de resultados

En cuanto al desarrollo del *forecasting*, el primer modelo que hemos realizado, nos hemos centrado en analizar y predecir el consumo de hipoclorito diario. Podría parecer visualmente que las predicciones no son del todo acertadas, sin embargo y si nos centramos en el consumo mensual (figura 37) se puede concluir que los consumos reales no difieren prácticamente de los predichos. Esto nos hace pensar que una implantación supervisada podría ser viable.

En cuanto al desarrollo de la Red neuronal recurrente, el segundo modelo que hemos realizado, nos hemos centrado en analizar y predecir el nivel del depósito de hipoclorito en vez del consumo. Parece que la decisión de escalar los datos con el *'MinMaxScaler'* ha resultado fundamental a la hora de llevar a cabo el algoritmo. También la elección de las capas ocultas de la red neuronal en modo *'Dense'* y la función de activación *ReLU* han dado resultados. El error cuadrático medio presenta un valor relativamente bajo y la gráfica de predicción se ajusta a la realidad.

A pesar de la plena implicación por mi parte y por parte de la empresa, se concluye que los resultados son favorables pero se podrían ajustar más a la realidad con un *input* de datos mayor.

5.2 Líneas futuras

Este proyecto de investigación ha tenido una duración de aproximadamente 3 meses. En este tiempo se ha involucrado mucha gente en la realización del trabajo debido a que es un tema revolucionario que podría reportar grandes beneficios.

El almacenamiento, dosificación y reposición de hipoclorito es un tema muy importante para las empresas de gestión de agua debido a que sin este compuesto no se podría potabilizar el agua y no habría agua para el consumo en la población.

Aunque el tiempo para la realización hasta la fecha de entrega de este trabajo de investigación ha sido escaso, se pretende por ambas partes continuar en el desarrollo del algoritmo predictivo para buscar una mayor precisión.

Si se quisiera llevar a la etapa de producción, lo más importante sería implementar un modelo ficticio, es decir, una estación de tratamiento de aguas residuales a pequeña escala con un pequeño depósito que dosificara hipoclorito, en base a nuestras predicciones, al agua bruta para potabilizarla. Este modelo contaría con un medidor de nitratos lo que nos indicaría que el agua es potable y apta para el consumo humano con un valor menor al 50%.

Finalmente, en caso de funcionar este modelo, se podría aplicar el algoritmo a los municipios analizados de Paterna y L'Eliana bajo una supervisión por un operario debido a que en caso de error las consecuencias serían fatales. Una vez supervisado durante un tiempo y habiendo recogido datos para captar posibles imprecisiones en el modelo, se podría ajustar el algoritmo predictivo en base a esta recolección.

De esta manera, se podría operar en cualquier municipio bajo este procedimiento e implantar el modelo a gran escala. Se conseguiría la completa digitalización para un proceso totalmente esencial como es el abastecimiento de agua potable a la población.

Bibliografía

[1] Sandra Blakeslee y Jeff Hawkins, 2005. *Sobre la inteligencia*. Espasa libros.

[2] Agua de Consumo Humano, Ministerio de Sanidad, Consumo y Bienestar Social.
Available:https://www.sanidad.gob.es/profesionales/saludPublica/saludAmbLaboral/calidadAguas/consumoHumano.htm

[3] HIDRAQUA. Available: https://www.hidraqua.es/

[4] Descubre Dinapsis Valencia (2023). Available:
https://www.dinapsis.es/conoce-la-red/valencia/

[5] Segura, C. (2019). Big Data, Introducción a conceptos y terminología, Linkedin.
Available:https://es.linkedin.com/pulse/big-data-introducci%C3%B3n-conceptos-y-terminolog%C3%ADa-cristian-segura

[6] El verdadero padre de la inteligencia artificial, OpenMind (2018). Available:
https://www.bbvaopenmind.com/tecnologia/inteligencia-artificial/el-verdadero-padre-de-la-inteligencia-artificial/

[7] ¿Qué es el test de Turing? (2015). Available:
http://introduccionalainformaticaforense.blogspot.com/2015/08/que-es-el-test-de-turing-el-de-turing.html

[8] Inteligencia Artificial y uno de sus pioneros, A.L. Samuel. (2015). Available:
https://www.unocero.com/videojuegos/inteligencia-artificial-y-uno-de-sus-pioneros-a-l-samuel/

[9] ¿Qué es reinforcement learning?, MATLAB & Simulink. Available:
https://es.mathworks.com/discovery/reinforcement-learning.html

[10] Vélez, J. Aprendizaje Automático frente a IOT: Entender la diferencia (2022). Available: https://barbaraiot.com/es/blog/diferencia-machine-learning-inteligencia-artificial

[11] Systems (2020) Redes Neuronales o el Arte de imitar el Cerebro Humano, magiquo creamos inteligencia. Available: https://magiquo.com/redes-neuronales-o-el-arte-de-imitar-el-cerebro-humano/

[12] Clasificación de Redes neuronales artificiales (2017), Diego Calvo. Available:https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[13] González, L. (2022) ¿Qué es el perceptrón? perceptrón simple y multicapa, Aprende IA. Available: https://aprendeia.com/que-es-el-perceptron-simple-y-multicapa/

[14] Red neuronal convolucional CNN, Diego Calvo (2017). Available: https://www.diegocalvo.es/red-neuronal-convolucional/

[15] Neural networks: ¿De qué son capaces las redes Neuronales Artificiales? (2020) IONOS Digital Guide. Available: https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-búsqueda/que-es-una-neural-network/

[16] NumPy. Available: https://numpy.org/

[17] Pandas. Available: https://pandas.pydata.org/

[18] Matplotlib. Available: https://matplotlib.org/

[19] TensorFlow. Available: https://www.tensorflow.org/?hl=es-419

[20] Keras. Available: https://keras.io/

[21] Scikit-learn. Available: https://scikit-learn.org/stable/

[22] El Dato: La Gasolina para el motor del data science (2018) Máster en Data Science. Available: https://www.master-data-scientist.com/el-dato-motor-data-science/

[23] Amat, J. , Escobar, J. Skforecast: Forecasting series temporales con python y scikitlearn (2021). Available: https://www.cienciadedatos.net/documentos/py27-forecasting-series-temporales-python-scikitlearn.html

[24] Morante, S. Precauciones a la hora de normalizar datos en data science, Think Big. Available: https://empresas.blogthinkbig.com/precauciones-la-hora-de-normalizar/#:~:text=Escalado%20de%20variables%20(Feature%20Scaling%20o%20MinMax%20Scaler)&text=El%20problema%20de%20este%20tipo,%C3%A9ste%20va%20a%20ser%20ampliado.

[25] Activation functions in neural networks. Study Machine Learning. Available: https://studymachinelearning.com/activation-functions-in-neural-network/

[26] Torres, A. Aprendizaje Automático: Una Introducción al Error Cuadrático medio y las líneas de regresión (2021). Available: https://www.freecodecamp.org/espanol/news/aprendizaje-automatico-una-introduccion-al-error-cuadratico-medio-y-las-lineas-de-regresion/