E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Detección de outliers en series temporales de contaminantes

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Javier Lasheras Navas

Director: Mikel Galar Idoate

Pamplona, 30 de abril de 2020

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Me gustaría agradecer en este trabajo a mi tutor de prácticas Carlos por todo tu apoyo en este último año de grado, y a mis compañeros del departamento de I+D+I: Christian, Rubén, Mikel, Asier y Christian por ayudarme en este trabajo todo lo que he necesitado.

Resumen

La contaminación es un problema que nos toca a todos muy a fondo. La temperatura de la tierra sube cada vez más y más, los polos se están deshelando y las enfermedades que estaban extintas están resurgiendo por esta subida de calor. Es por eso, que se han creado agencias gubernamentales que velan por la seguridad medioambiental del planeta. En Europa esta agencia de medio ambiente se llama EEA (European Environment Agency).

Para proteger Europa de niveles de contaminación excesivos, esta agencia observa de manera periódica los niveles de contaminantes en ciertas zonas. Para ello construyen sensores y los distribuyen por todos los rincones del continente y recopilan esa información en bases de datos que posteriormente analizan.

Sin embargo, estos sensores no son perfectos. Los valores recopilados por estos pueden fallar por múltiples razones como, por ejemplo, un pico de tensión, un problema en la transmisión de los datos a la base de datos, por obstrucción del sensor o por cualquier otra eventualidad. Estos fallos, a pesar de ser esporádicos, son un problema muy grande. Al analizar los datos automáticamente y tratarlos no puede haber fallos asiduamente, ya que si así fuera se encenderían alertas de contaminación en múltiples países dados estos fallos, llegando *in extremis* a poner en marcha planes contra esta contaminación a nivel nacional.

Es por esto, que desde Tracasa Instrumental y en colaboración con la EEA vemos necesario desarrollar técnicas de machine learning y de deep learning para detectar estos fallos y solucionarlos en la medida de lo posible, para no encender falsas alarmas y para evitar datos corruptos que fallen cualquier estudio sobre los datos.

Lista de palabras clave

Algoritmo, Outliers, Clustering, Machine Learning, Modelo Autorregresivo, Deep Learning, k-means, Isolation Tree

Contenidos

1.Introducción

Desde el comienzo de los tiempos, el ser humano ha tratado de entender su mente. Aristóteles[1] fue el primero en crear un conjunto de reglas, llamadas silogismos, que describen el funcionamiento lógico de la mente humana y que a partir de premisas se generan conclusiones razonables. No fue hasta 1847 cuando George Boole continuo su trabajo creando la lógica proposicional. Sin embargo, no fue hasta 1950, cuando otro gran genio, Alan Turing, nacido en Inglaterra, formulo una pregunta que marcaría el futuro: "¿Puede pensar una máquina?". Alan había dado con la clave de la inteligencia artificial planteándose si, lo que nos hace humanos, nuestro cerebro, se podría replicar mediante bobinas y circuitos. [1]

Desde este momento, al que se le considera el momento de la concepción de la inteligencia artificial, el mundo ha cambiado mucho. Las antiguas máquinas de computo, a las que ni siquiera se le llamaba todavía ordenadores, han pasado de ocupar edificios enteros a caber en la palma de la mano. Todos estos avances en el desarrollo de las máquinas físicas han venido acompañados, como no podía ser de otra manera, de avances en los cálculos que realizaban estas, desarrollando nuevas técnicas o algoritmos para obtener nuevos resultados de manera más eficiente.

La pregunta de nuestro genio inglés también ha evolucionado hasta pasar a preguntarnos "Si los seres humanos aprendemos… ¿Lo podrán hacer también las maquinas? Así surgió el machine learning, máquinas que aprenden solas a desempeñar tareas sin que haya nadie detrás programándolas para tales propósitos.

Juntando todo esto con los avances en neurociencia y en entender el cerebro, se han desarrollado algoritmos que aprenden como el cerebro humano. Simplificándolo mucho, el cerebro es un conjunto de circuitos eléctricos que dejan pasar la electricidad con una mayor o menor intensidad de manera ordenada para crear conocimiento. Así pues, nacieron las redes neuronales, algoritmos de aprendizaje que se basan en simular este funcionamiento.

Hoy en día el conocimiento de computo es tan grande, que podemos utilizar las redes neuronales "a lo bestia". Podemos crear capa tras capa de neuronas de manera que la abstracción del conocimiento del algoritmo sea muy profunda. Así nació el Aprendizaje profundo (o Deep Learning).

Estos algoritmos de Deep Learning (DL a partir de ahora) se han especializado, al igual que hay ciertas partes de nuestro cerebro que están especializadas en ver imágenes o en analizar patrones en series de números o de palabras. De hecho, uno de los mayores usos del DL es el de analizar imágenes, ya sea para detectar caras o para contar coches que pasan por un túnel. Las aplicaciones son infinitas.

Las series temporales se pueden analizar por ejemplo para reconocer el lenguaje. Pensándolo bien, el lenguaje no es más que una serie de palabras puestas una tras otra. Un ejemplo de interpretación de series temporales para reconocer el lenguaje podría ser dispositivos como

[1] Filósofo y científico de la antigua Grecia.

Google Home o Amazon Echo, que graban tus palabras en orden y las analizan para sacar el significado de las frases y realizar tus órdenes.

Por otra parte, el análisis de series temporales también se puede llevar a cabo con estas técnicas, siempre y cuando sigan un patrón. Por ejemplo, el número de lotería que va a salir en el próximo sorteo de lotería de Navidad es un fenómeno aleatorio, así que no se puede analizar. En yuxtaposición con estos fenómenos, en nuestro día a día se pueden observar fenómenos que siguen una rutina, una tendencia. Se puede ver en el tráfico diario en una carretera o en la cantidad de usuarios conectados a la red wifi de la oficina, todos estos fenómenos son predecibles.

Otro ejemplo podría ser la contaminación de una ciudad. Es notable que, a las mañanas, a plena hora punta, los niveles de CO_2 o de NO_2, van a ser muy elevados debido a los gases de efecto invernadero de los coches. También se podría pensar que durante los fines de semana la polución debida a estos contaminantes será mucho menor, ya que hay mucha gente que no trabaja. O que en San Fermín estos niveles también serán mucho más reducidos. Por otra parte, no solo hay contaminación producida por los coches. También puede haber una tormenta que arrastre micro partículas del suelo o de otros continentes como África, que son también peligrosos.

Para medir estos contaminantes, se usan sensores. Estos sensores no son infalibles. Si un camión se pone cerca de uno de los sensores de CO_2 durante 2 minutos puede dañar el sensor o cuanto menos crear mediciones erróneas. También se pueden generar errores al enviar los datos, por una corrupción en el medio de transmisión, o por problemas del propio sensor que se haya averiado. A todos estos errores se les denomina outliers.

En este contexto, se necesitan métodos para detectar estos errores para su posterior corrección. Un ser humano detectaría sin problemas estos errores mirando las gráficas con suficiente detalle, pero necesitaríamos demasiado personal y tiempo para hacerlo. Así que la mejor solución es poner a aprender a las máquinas para que desempeñen el papel de un humano.

La motivación de este trabajo de fin de grado desarrollar técnicas de machine learning y de Deep learning para detectar estos errores.

El objetivo general del trabajo es ayudar es desarrollar técnicas para detectar los errores en las mediciones de los sensores de contaminación que controla la Agencia Europea de Medioambiente. Este objetivo se enmarca en un proyecto europeo que se desarrolla en la empresa Tracasa Instrumental, lugar donde he realizado las prácticas.

Para ello, definiremos como objetivos particulares:

- Estudiar técnicas de análisis de series temporales para entenderlas mejor.
- Realizar un análisis exploratorio de los datos, para, por ejemplo, detectar correlaciones entre dos zonas con tipos de contaminación similares.
- Estudiar y aplicar técnicas de detección de outliers en series temporales mediante la estadística.
- Estudiar y aplicar técnicas de detección de outliers en series temporales mediante machine learning.

- Estudiar y aplicar técnicas de detección de outliers en series temporales mediante deep learning.
- Desarrollar un método mediante estos estudios previos para hacer un análisis efectivo de estos outliers.

2.Descripción del problema

Como ya he introducido en el apartado anterior, nuestro problema es el de detectar anomalías en series temporales de sensores de contaminantes de la agencia europea de medioambiente, pero vamos a precisar un poco más.

La Agencia Europea de Medio Ambiente (Las siglas son EEA en inglés) es un organismo de la Unión Europea. Su trabajo es ofrecer información sólida e independiente sobre el medio ambiente. Es la fuente principal de información para los responsables de las políticas medioambientales europeas así como de cualquier interesado en las mismas.[2]

Para ofrecer esta información sólida, obtienen información de distintos sensores que tienen distribuidos por todo Europa. Hay unos 500 sensores y recogen información de 5 contaminantes de los que contamos con 4: NO2, O3, PM10 Y PM25. Posteriormente se profundizará más en ellos. Con esta información, crean mapas informativos de contaminación por zonas como los de la Figura 1. Estos sirven a las agencias medioambientales nacionales, para tomar medidas donde sea oportuno.

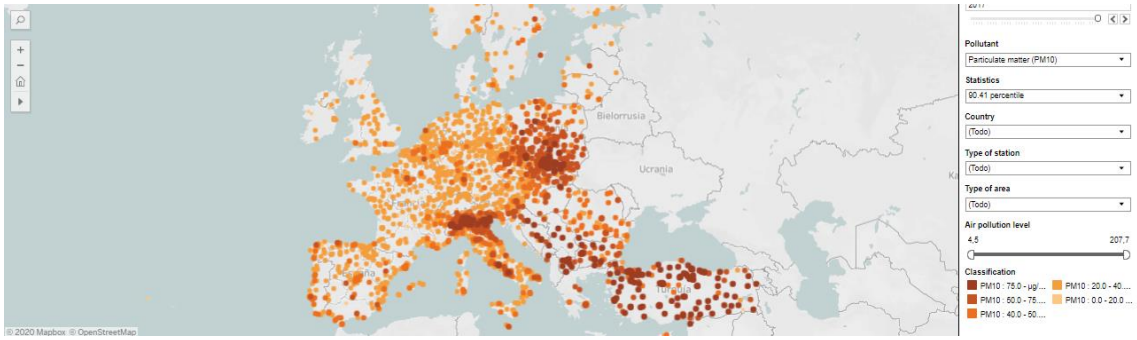


Figura 1: Mapa de contaminación en Europa. Fuente: EEA.

Estos sensores recopilan información periódicamente (cada hora) y la envían a una base de datos. La información que se envía en un primer lugar no tiene ningún tipo de filtrado ni de tratamiento, tal cual se recoge se envía. Estos datos se denominan UTD (Up-to-date).

Sin embargo, estos datos muchas veces tienen errores. Es por eso, que cada país corrige sus datos cada cierto tiempo y los sube a otra colección que se llama CDR (Central data repository). Al llevarse a cabo la corrección por país, se puede observar que hay países que corrigen más sus datos y otros que corrigen menos. Aproximadamente el 30% de los datos tienen alguna corrección, generalmente usan la desviación para corregirlos.

Nosotros disponemos de la información UTD y de la información CDR. Contamos con esta información en formato CSV de más de 1000 estaciones. Además, también disponemos de otros dos CSV, uno para UTD y otro para CDR, con la información geográfica y características estáticas de cada una de las estaciones.

No todas las estaciones están en ambos datasets. Hay ciertos países que no recopilan información UTD dentro de la base de datos de la EEA, pero que sin embargo si envían datos en formato CDR, por ejemplo, Turquía.

Por otra parte, a nosotros también se nos entrega en esos mismos CSVs, una serie de medidas climatológicas que pueden ser oportunas para el estudio de los outliers. Estos son los siguientes:

- **Presión en la superficie (sp):** Se mide en pascales y representa la presión atmosférica sobre la superficie de la tierra en el lugar donde se esté midiendo.
- **Radiación solar neta en la superficie (ssr):** Se mide en metros elevado a la inversa de dos $\left(\frac{1}{m^2}\right)$ y representa la cantidad de radiación del sol que llega a la tierra a pesar de los filtros de la atmosfera.
- **Temperatura a 2 metros (t2m):** Se mide en grados kelvin y representa la temperatura que hay a 2 metros de la superficie de la tierra en el lugar de la medición.
- **Cobertura de nubes total (tcc):** Se mide en un porcentaje y representa cuantas nubes hay en el lugar y momento de la medición.
- **Precipitación total (tp):** Se mide en metros y representa la cantidad de agua que ha caído en el lugar y momento de la medición.

Como La Agencia Europea de Medioambiente también cuenta con estos datos, sabe que necesita unos métodos más eficaces a la hora de arreglar los datos con errores. Por ello, ha lanzado un proyecto europeo financiado para que cualquier entidad pueda entrar en el concurso y tratar de conseguir esta financiación. Entre estas entidades involucradas se encuentra Tracasa Instrumental, que es la empresa en la que hemos desarrollado este TFG.

Para solucionar este problema tenemos la siguiente información acerca de los datos:

- La serie sigue una relación autoregresiva con la propia serie. En el siguiente capítulo, entraremos en el estudiaremos más en detalle que significa este término, pero como resumen significa que cada observación depende de observaciones pasadas.
- Hay una cierta correlación entre los diferentes contaminantes. El NO2 y el O3 están inversamente correlacionados, es decir, cuando aumenta uno disminuye el otro. Por otro lado, los PM10, PM25 y NO2 están directamente correlacionados, es decir, cuando aumenta uno aumentan los otros.
- Las mediciones dependen de las condiciones meteorológicas. Estas condiciones pueden ser la radiación solar, la presión ambiental, la precipitación o humedad en el ambiente, la cobertura nubosa o la temperatura.

Como ya hemos enumerado antes tenemos 4 contaminantes de los que se realizan las mediciones. Estos son:

- **PM10:** Son partículas microscópicas de 10 a 2.5 micrómetros. Se produce por tormentas de polvo, pero también por los coches.
- **PM25:** Son, como el PM10, partículas microscópicas, pero en este caso de menos de 2.5 micrómetros. Al ser más pequeñas estas partículas, son más peligrosas ya que se filtra más fácilmente. Excepto el tamaño, tienen las mismas características que el PM10.
- **NO2:** Su nombre es dióxido de nitrógeno. Lo producen los coches así que su concentración es mucho más alta en zonas con tráfico denso.
- **O3:** Este es el Ozono. En regiones altas de la atmosfera sirve para filtrar la radiación solar, pero en la superficie de la tierra es nocivo para la vida. Se produce a partir del NO2 a través de reacciones fotoquímicas.

Por último, el proyecto tiene como requerimiento hacer un análisis de las muestras, en el plano geográfico. Esto se debe a que quizás las muestras de Paris se parezcan más a las de Londres, debido a por ejemplo la cantidad de habitantes o a los horarios de tráfico, que a otras y se pueda aplicar esta relación para solucionar los outliers de una mejor manera. Además, como ya hemos expuesto previamente hay países que corrigen mejor los datos y por tanto son más fiables que otros. Estos son Austria, Bélgica, Gran Bretaña o países similares a estos. Por otra parte, también hay series temporales que presentan muchos outliers o que prácticamente la serie no corregida y la corregida parecen sacadas de diferentes fuentes. Este es el ejemplo de Turquía, cuyos datos parecen incluso estar falseados.

En el ámbito de lo geográfico, también hay que dejar claro que un país no solo cuenta con una estación, sino con muchas. Por tanto, esta correlación de la que hablábamos anteriormente entre ciudades similares de distintos países, puede ser mucho más útil si se da entre dos ubicaciones cuya distancia no sea grande. Por ejemplo, en París hay diversas estaciones que pueden tener una relación mucho más obvia que cualquier asociación de esta misma ciudad con la capital inglesa.

3.Preliminares

En este capítulo se explicarán los fundamentos necesarios para entender el trabajo de fin de grado. Se tratarán aspectos como el machine learning aplicado a la detección de outliers, el tratamiento de las series temporales y el uso del Deep learning para tratar estas series.

3.2 Machine learning

Machine learning es el proceso por el cual los ordenadores consiguen "aprender" a partir de los ejemplos que les están disponibles. Hablando de manera vulgar, aprender es el proceso que convierte la experiencia en conocimiento. La entrada de un algoritmo de aprendizaje son **datos de entrenamiento**, **experiencia representada** y la salida es alguna **habilidad**, que normalmente toma forma de un programa informático que puede realizar alguna tarea. Buscando una forma más matemática de entender este concepto, tenemos que ser más explícitos sobre a que nos referimos con cada uno de los términos involucrados: ¿Cuáles son los datos de entrenamiento a los que puede acceder nuestro programa? ¿Cómo podemos automatizar el proceso de aprender automáticamente? ¿Cómo podemos evaluar el éxito de este proceso (La calidad de la salida de un algoritmo de aprendizaje)?[3]

3.2.1 Uso del machine learning

Pero… ¿Para qué necesitamos machine learning? ¿No se puede hacer un algoritmo que resuelva las cosas y sea mucho más simple? Para solucionar cualquier problema de manera computacional hay que tener en cuenta dos aspectos: La complejidad del problema y la necesidad de adaptabilidad:

- Tareas que son demasiado complicadas de programar: Estas a su vez se dividen en dos.
    - Tareas realizadas por animales/humanos: Hay muchas tareas que los humanos realizamos de manera rutinaria pero que no comprendemos suficientemente bien para programarlas. Por ejemplo, conducir, reconocer el habla y entender imágenes.
    - Tareas que están más allá de las capacidades humanas: Hay otra familia igualmente grande de tareas que se benefician de las técnicas de machine learning, ya que se basan en analizar grandes cantidades de datos que además son complejos: Datos astronómicos, predicción del tiempo o analizar datos genómicos.
- Adaptabilidad. Una limitación de las herramientas programadas es su rigidez. Una vez que el programa ha sido escrito e instalado, permanece igual para siempre. Sin embargo, hay muchas tareas que cambian a lo largo del tiempo o de un usuario a otro. Las herramientas que se basan en machine learning tratan de solucionar este problema, ya que, por naturaleza, son herramientas que basan su comportamiento en el entorno que le rodea. Por ejemplo, los programas que descodifican texto escrito a mano, que se ajustan a la letra de cada usuario, los detectores de spam, que se ajustan a que el spam cambia a lo largo del tiempo, y los programas de reconocimiento del habla.

En nuestro problema particular, la detección de outliers en series temporales de contaminantes, cumple los dos aspectos: Es una tarea que está más allá de nuestras capacidades ya que tiene que lidiar con grandes cantidades de datos y además tiene que ser

adaptable, ya que puede ser que los hábitos de tráfico de una ciudad fluctúen con el tiempo, haciendo así que un algoritmo que vale para hoy no valga dentro de un año.

3.2.2 Tipos de aprendizaje

Aprender es un terreno muy amplio. De hecho, el campo del machine learning se ha desglosado en diferentes subtipos dependiendo de la manera en la que la máquina aprende. Para ilustrar esto, vamos a dar una pequeña clasificación de los diferentes paradigmas de aprendizaje, con el objetivo de que posteriormente se entiendan mejor nuestras explicaciones sobre el trabajo realizado.

Describimos 3 parámetros sobre los que el aprendizaje puede ser clasificado, aunque puede haber más:

- **Supervisado o No Supervisado.** El aprendizaje define una interacción entre el aprendiz y el entorno, así que se puede clasificar el aprendizaje según esta relación. Para ejemplificar esta clasificación, vamos a usar como ejemplos el sistema de detección de spam en contraposición detección de anomalías. Para la detección de Spam, consideramos un escenario en el que el aprendiz recibe correo etiquetado como spam o no spam. Basándose en este entrenamiento, el aprendiz tiene que obtener una regla para etiquetar el siguiente correo. Por otra parte, para la tarea de detección de anomalías, el aprendiz tiene como entrenamiento un montón de correos electrónicos, sin etiquetas, y el aprendiz tiene que detectar los mensajes que no son normales.

  Cambiando de punto de vista, viendo el aprendizaje como un proceso por el cual usamos experiencia para obtener una habilidad, el aprendizaje **supervisado** describe un escenario donde la experiencia, cada ejemplo de entrenamiento, contiene información significativa (En nuestro ejemplo, Spam o no spam) que falta en los ejemplos de test, donde la habilidad aprendida tiene que ser demostrada. En resumen, el aprendizaje supervisado trata de predecir la información que falta para los datos de test. Podemos pensar en este tipo de aprendizaje como un profesor que supervisa el aprendizaje del alumno dándole información extra (Las etiquetas).

  Por otra parte, en el aprendizaje **no supervisado** no hay distinción entre los datos de entrenamiento y los de test. El aprendiz procesa los datos de entrada con el objetivo de obtener algún resumen, o alguna versión comprimida de los datos. Clusterizar unos datos en subconjuntos de objetos similares es un ejemplo típico de este aprendizaje.

- **Aprendizaje activo o pasivo.** Las formas de aprender pueden variar en función del rol que juega el aprendiz. Distinguimos entre aprendices activos o pasivos. Un aprendiz activo interactúa con el entorno en el tiempo de entrenamiento, haciendo consultas o haciendo experimentos, mientras que un aprendiz pasivo solo observa la información que se le proporciona por el entorno o por el profesor sin influenciarla o cambiarla. En nuestro ejemplo de la detección de spam, normalmente es pasiva porque espera a que los usuarios marquen los correos que les llegan. En un escenario activo, el propio cliente de correo electrónico podría preguntar al usuario si un correo en particular es spam o no, mejorando así su comprensión de lo que es el spam.

El aprendizaje que vamos a llevar a cabo en este trabajo de fin de grado va a ser tanto supervisado como no supervisado (detallaremos en cada apartado cual estamos llevando a cabo) y siempre pasivo.

Además de estos tipos de aprendizaje que hemos enumerado arriba, hay subclases. Por ejemplo, dentro del aprendizaje supervisado podríamos distinguir dos tipos:

- La **clasificación** que trata de resolver el problema de decidir si el conjunto de datos que se le pasan al modelo como entrada pertenece a un tipo discreto concreto o a otro. En este caso se podría encontrar el caso del detector de Spam del que hemos hablado antes.
- La **regresión** que trata de predecir un valor continuo a partir de los datos de entrada. Dentro de este grupo se podría encontrar un modelo que, a partir de la ciudad, el tamaño, el número de ventanas y el número de baños de una casa estimase su precio.

Lo importante de esta clasificación, y por lo que hablo de ella es porque en nuestro trabajo de fin de grado vamos a tratar de clasificar los sucesos anómalos como tales y es por eso que es una clasificación, clasificación de los puntos “buenos” contra los puntos “malos”.

Sin embargo, esta clasificación tiene un nombre particular dentro de machine learning que es *one-class classification* y que es lo que vamos a explicar en el siguiente apartado.

3.2.3 One-class classification

Este concepto trata de explicar que únicamente se parte de un conjunto de datos en el que todas las instancias pertenecen a una sola clase. Por tanto, todos los puntos que parezcan más alejados de esta única clase se consideran casos anómalos y son estos los que se pretende detectar mediante el sistema clasificador.

Para explicar mejor este tipo de situación, vamos a poner un ejemplo. Imaginemos que estamos en una cadena de fabricación de baterías para móviles. Esta cadena tiene varios sensores que miden diferentes características del estado de la batería en distintos instantes de tiempo. La cadena quiere descartar las baterías que son defectuosas, para que no exploten en los móviles de los usuarios. En este contexto, prácticamente todas las baterías se crean correctamente, pero una batería defectuosa puede provocar, en un caso extremo, la explosión de un terminal móvil con la consiguiente mutilación de una mano del usuario final. Por esto, la dificultad de no tener, o tener muy pocos casos en los que la batería es defectuosa, hace que el sistema clasificador esté muy sesgado hacia los casos correctos y que, por tanto, no sea capaz de discernir entre una batería correcta y una defectuosa.

Para hacer una clasificación al uso, se ha investigado acerca de cómo balancear ambas clases, pese a tener muy pocos casos clasificados como anómalos. Sin embargo, este trabajo no ha profundizado en este balanceo, ya que, en nuestro caso, no hay dos clases sino solo una.

Estos algoritmos tratan de encontrar fronteras de decisión que aíslen el conjunto de datos para que, si se encuentra un outlier, se encuentre fuera de este espacio.

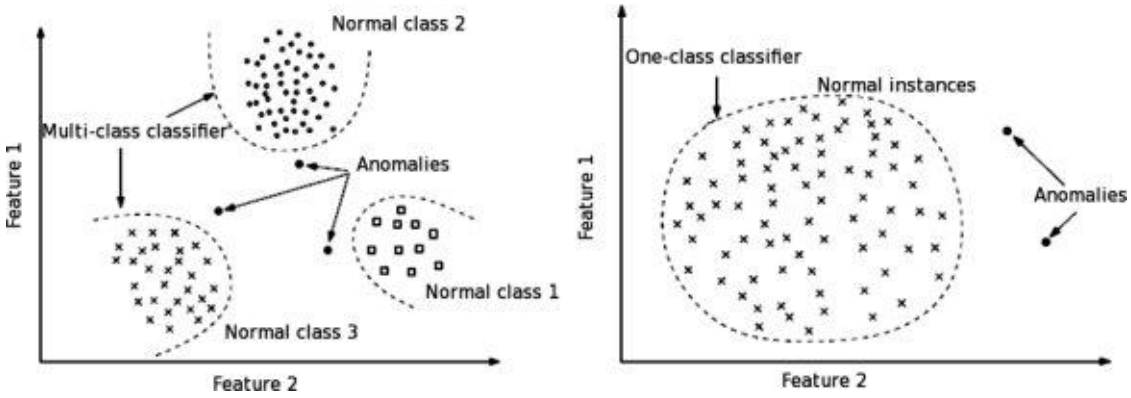


Figura 2: One Class Clasiffication

Por último, aclarar que *one-class classification* entra dentro del aprendizaje supervisado. Esto se debe a que tenemos datos de ambos datasets así que se pueden poner etiquetas a los datos buenos y a los datos malos, de tal manera que el algoritmo aprenda.

A partir de aquí, vamos a seguir explicando los modelos que se usan en este trabajo y aclarando por cada uno si se basa en *one-class classification*[4].

3.2.4 Modelos considerados en este trabajo

Ahora vamos a hablar en particular de los modelos que se usan en este trabajo de fin de grado, explicando qué son, cómo funcionan y para qué se suelen utilizar. No vamos a entrar en cómo los hemos usado en nuestro proyecto porque eso vendrá en otros capítulos, pero es interesante esta parte para la posterior comprensión del resto del documento.

3.2.4.1 Clustering

El clustering es un tipo de aprendizaje no supervisado cuyo objetivo es encontrar agrupamientos de tal forma que los objetos de un grupo sean similares entre sí y diferentes de los objetos de otros grupos [clústeres].

Los resultados obtenidos dependerán:

- El algoritmo de agrupamiento seleccionado, de los que hablaremos más tarde.
- El conjunto de datos disponible.
- La medida de similitud utilizada para comparar objetos. (Normalmente, definida como medida de distancia.)

Sus aplicaciones son muchas, entre ellas se encuentran la **detección de anomalías**, el reconocimiento de formas, el marketing, la clasificación de documentos…

A la hora de obtener la similitud entre dos objetos hay que tener en cuenta que:

- No tienen por qué utilizarse todos los atributos disponibles en nuestro conjunto de datos.

- Hay que tener cuidado con las magnitudes de cada variable.

Por otra parte, hay que tener en cuenta los valores continuos, ya que, si un valor es mucho más grande que otro puede pesar en el algoritmo mucho más, cuando simplemente es que la escala es diferente. Para llevar a cabo este escalado, se realiza un proceso que se llama normalización.

Una manera muy usada de normalizar es mediante la medida estandarizada. La fórmula para dicha normalización es la siguiente.

$$Z_f = \frac{x - \mu}{\sigma} \ (1)$$

Siendo μ la media y σ la desviación típica cuyas formulas son:

$$\mu = \frac{\sum_{i=0}^{N} x_i}{N} \ (2)$$

$$\sigma = \frac{\sum_{i=0}^{N} (x_i - \mu)}{N} \ (3)$$

Este método normaliza errores cuándo los parámetros de población son conocidos. Sobretodo trabaja bien para poblaciones que están normalmente distribuidas.

Otra manera de normalizar, que también es muy usada es normalizar por el rango. La fórmula de esta es la siguiente:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \ (4)$$

Esta característica suele traer todos los valores en el rango de [0,1]. Esta propiedad se suele llamar normalización basada en la unidad. Puede ser generalizado para restringir la gama de valores en el conjunto de datos entre cualesquiera puntos arbitrarios a y b utilizado la siguiente fórmula:

$$X' = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}} \ (5)$$

Volviendo a la clusterización, hemos hablado de que los resultados dependerán entre otras cosas de las medidas de similitud que utilicemos. Estas se suelen expresar en términos de distancias:

$$d(i,j) > d(i,k)$$

Esto nos indica que el objeto i es más parecido a k que a j.

Se suelen usar medidas de distancia como medidas de similitud porque verifican las siguientes propiedades:

- Propiedad reflexiva: $d(i,j) = 0 \Leftrightarrow i = j$
- Propiedad simétrica: $d(i,j) = d(j,i)$
- Desigualdad triangular: $d(i,j) \leq d(i,k) + d(k,j)$

La métrica de distancia más usada para atributos continuos es l**a distancia de Minkowski**

$$d_r(x,y) = \left(\sum_{j=1}^{J} |x_j - y_j|^r\right)^{\frac{1}{r}}, r \geq 1 \ (6)$$

Hay que tener en cuenta que esta fórmula si $r = 1$ tenemos la **distancia de Manhattan** y si $r = 2$ tenemos la **distancia euclidea.** Si la r tiende a infinito nos quedamos con la fórmula:

$$d_\infty(x,y) = máx_{j=1..J}|x_j - y_j| \ (7)$$

A esta distancia se le conoce como **la distancia de chebyshev** [5]**.**

Cómo los datos que vamos a tratar en este trabajo de fin de grado van a ser todos continuos, no vamos a entrar en otros tipos de distancias que sirven para cadenas o para datos discretos.

Por otra, parte algo que influye mucho en el resultado de estos algoritmos para hacer grupos, es la manera que tenemos para agruparlos. Dentro de esto, han surgido distintos paradigmas a partir de los cuales se pueden unir distintos elementos en un subconjunto. Vamos a presentar todos estos ya que en un apartado del trabajo realizado veremos que se realizan diversos experimentos, con el fin de encontrar cual es el mejor método de agrupamiento para nuestros datos.

Estos métodos se clasifican en:

- **Agrupamiento por particiones:** En los que se podría encontrar k-means, que es posiblemente el más usado. En este tipo de agrupamiento, se tiene que fijar el número que queremos tener. Existen diversos métodos para optimizar este número de particiones, que exploraremos más adelante.
- **Agrupamiento por densidad:** En el que podemos encontrar DBSCAN.
- **Clustering jerárquico:** En el que podría estar por ejemplo el método BIRCH. En este caso, a diferencia del agrupamiento por particiones, no se fija el número de particiones. Esta distinción se verá clara posteriormente al explicar el algoritmo.

Ahora vamos a explicar brevemente un ejemplo de cada tipo de método.

3.2.4.1.1 k-means

El primero que vamos a explicar es el **k-means**, un ejemplo de agrupamiento por particiones.[5]

Este algoritmo se caracteriza por tener un número de clústeres conocido (k), cómo ya hemos explicado para todos los métodos de agrupamiento por particiones. Para realizar esta división en subclases, se asocia a cada clúster un centroide, que es el centro geométrico del clúster. De esta manera, los puntos se asignan al clúster cuyo centroide esté más cerca, utilizando la métrica de distancia que hayamos decidido. Para afinar los clústeres y que se ajusten mejor a los datos, básicamente la parte de machine learning, se van actualizando los centroides en función de las asignaciones de los puntos a los clústeres, hasta que los centroides dejen de cambiar.

Para inicializar este algoritmo, se escogen al azar K centroides y se forman K grupos en función de estos centroides previamente elegidos. Para ajustar el modelo, como hemos dicho antes, se calculan las distancias de todos los puntos a los K centroides, se forman K grupos en función del

centroide más cercano y se recalcula el centroide, para que sea el centro de todos los puntos de su grupo. Para recalcular los centroides, hay que elegir una medida global (Por ejemplo, la media) y aplicarla.

La complejidad de **K-means** es en principio NP si el número de clústeres no es fijo. Sin embargo, si fijamos el número de puntos de nuestro espacio (n), el número de clústeres que queremos hacer (k), el número de iteraciones que vamos a hacer para afinar este modelo (I) y el número de atributos de cada uno de los vectores de nuestro espacio (d) la complejidad quedaría:

$$O(n * k * I * d) \ (8)$$

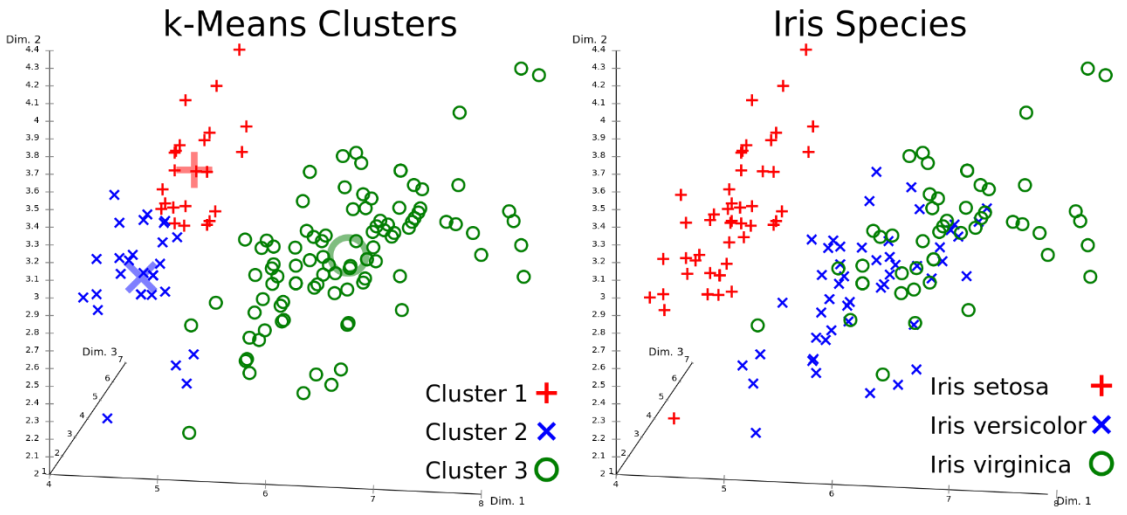


Figura 3: K-means. Fuente: https://es.wikipedia.org/wiki/K-medias

K-means es un algoritmo que funciona normalmente bien, pero no por ello se libra de tener algún problema:

- Es sensible a la elección inicial de los centroides. Es decir, si cambias los centroides iniciales varía la distribución de los grupos al final de la ejecución del algoritmo. Para solucionar esto hay varias ideas. Una de ellas, es hacer varias ejecuciones del algoritmo con diferentes centroides iniciales, comparar resultados y quedarte con la mejor ejecución. Otra solución, es fijar desde el principio unos buenos centroides con algún algoritmo determinista.
- Hay que elegir a priori el valor de K, es decir, estimar a priori el número de grupos que va a haber, cosa que muchas veces no sabemos. Para solucionar esto, podemos usar un método jerárquico para estimar el valor de k, aunque es mucho más ineficiente, o podemos usar algún método para estimar este valor.

  Un método muy famoso para estimar el valor de k, que usamos en este trabajo de fin de grado, se llama *"la regla del codo"*. Para seguir este método, se realiza el k-means con distintos números de clústeres. A continuación, se representa en una gráfica el error cuadrático medio entre el centro de cada clústeres y los puntos que se encuentran en ese clúster. Esta cuenta se denomina inercia.

$$Inercia = \sum_{i=0}^{n} \|x_i - \mu\|^2 \ (9)$$

Por último, se observa la gráfica y se busca lo que pueda ser su *"codo"*. Podemos encontrar un ejemplo de esto en la Figura 4. Se puede observar que el valor *"codo"* puede ser 5. Así que elegiríamos 5 como número de clases

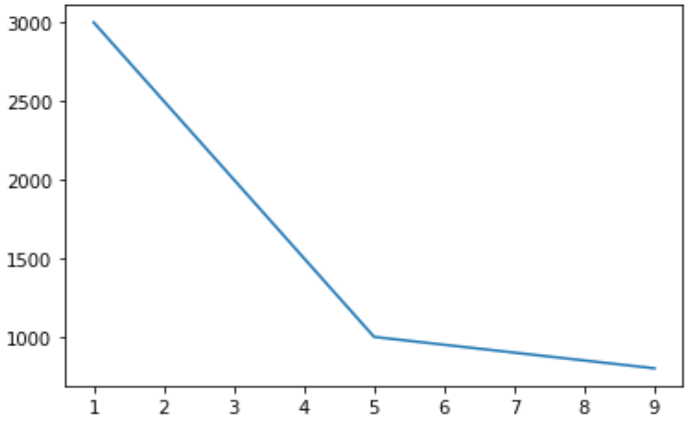


Figura 4: Regla del codo

- Cuando se usa la media para calcular los centroides el método es sensible a outliers.
- Tiene problema para manejar los tributos no numéricos.
- Por último, K-Means no funciona bien cuando los clústeres son de distinto tamaño, de diferente densidad o no convexos.

3.2.4.1.2 DBSCAN

Una vez expuesto el funcionamiento de **k-means** y expuestos también sus problemas, cambiamos de paradigma de agrupamiento al agrupamiento por densidad. En este grupo, vamos a explicar a su representante, **DBSCAN**[5]**.**

La concepción de **DBSCAN** acerca de clúster es diferente a la que podíamos ver con K-Means. En este caso, los grupos no son puntos muy cercanos, sino que un clúster es una región densa de puntos, separada por regiones poco densas de otras regiones densas.

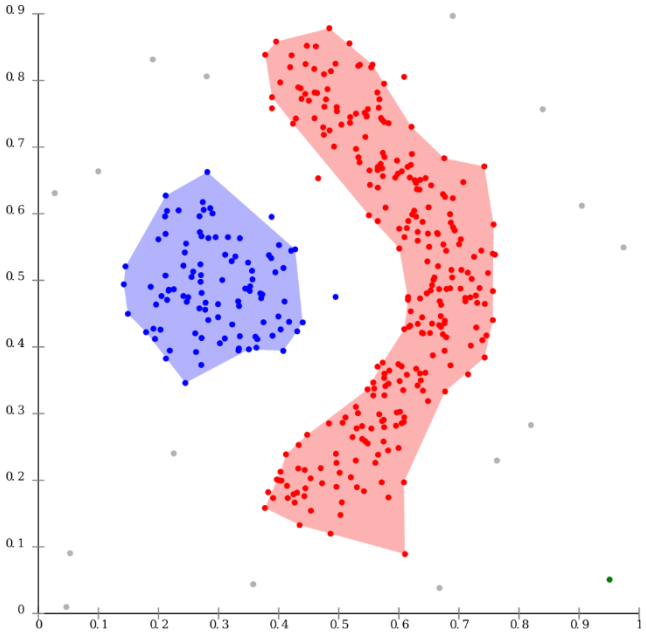


Figura 5: DBSCAN. Fuente:https://es.wikipedia.org/wiki/DBSCAN

Este algoritmo viene a resolver dos de los problemas que tenía **k-means**:

- En este caso, si el conjunto de datos describe grupos de distinto tamaño, diferente densidad o mostrando formas convexas, el algoritmo no tiene ningún problema en identificar estos grupos.
- Además, no tiene problema con los outliers o con el ruido, ya que lo podrá incluso detectar y filtrar
- Por finalizar, la eficiencia de este algoritmo es mejor que **k-means**. En este caso, la complejidad es $O(n \log n)$.

Sin embargo, a pesar de que parece que todo son ventajas, en la mayoría de los casos funciona mejor **K-means.** Esto es debido a que las distancias son un criterio mucho más robusto que la densidad para formar grupos.

3.2.4.3 Clustering jerárquico

Para terminar con el clustering, vamos a explicar por encima las técnicas que se aplican en el clustering jerárquico. En primer lugar, para entender el concepto de similitud en este tipo de clustering hay que saber el concepto de dendrograma. Un dendrograma es un gráfico en el que se reflejan los objetos como ramas de un árbol binario, de tal forma que la similitud de dos objetos se puede ver observando la "altura" del nodo común más cercano. Se puede observar esto en la Figura 6[2].

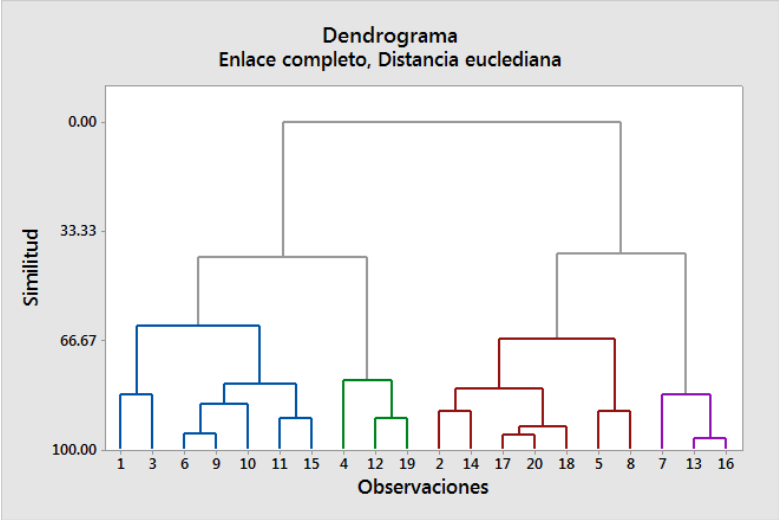


Figura 6: Dendrograma.

Para jerarquizar en este tipo de clustering, hay dos técnicas:

- **Técnicas aglomerativas:** En este tipo de jerarquía, se comienza con cada caso como clúster individual. En cada paso de ajuste, se van combinando el par de clústeres más cercano hasta que sólo queda uno o el número de clústeres que queramos.
- **Técnicas divisivas:** En yuxtaposición con el caso de arriba, se comienza con un único clúster que engloba todos los casos de nuestro conjunto de datos. Iteración tras iteración, se parte el clúster hasta que cada clúster contiene un único caso.

Como se puede ver en la Figura 4, es bastante fácil observar cuantos clústeres son los óptimos, así que no hace falta definirlo desde el principio como sí que era necesario en el **k-means,** sino que se puede ver al final.

Todas las ventajas traen una desventaja y esta no es una excepción. Así como solucionamos el problema de tener que fijar el número de clústeres desde el principio que tenía el **k-means,** empeoramos su complejidad computacional hasta $O(n^2)$. Esto lo hace muy poco escalable.

Conclusión

Para concluir el apartado del clustering, hay muchas técnicas y muchas medidas, pero la más usada suele ser **k-means** usando la **distancia euclidea** como métrica. Esto no significa que sea la mejor técnica que se pueda usar, ya que esto dependerá del tipo de problema a considerar.

---

[2] https://support.minitab.com/es-mx/minitab/19/help-and-how-to/modeling-statistics/multivariate/how-to/cluster-observations/interpret-the-results/all-statistics-and-graphs/dendrogram/

La siguiente técnica en la que vamos a profundizar, se enmarca dentro de *one-class classification* y se llama isolation forest.

3.2.4.2 Isolation forest

Como ya hemos dicho en el apartado anterior, este algoritmo es un algoritmo de detección de anomalías que se enmarca dentro del paradigma de *one-class classification* [6].

Los outliers son aquellos datos que son diferentes a los que consideramos normales. Es por eso, que normalmente los modelos lo que hacen es crear una definición de lo que consideran normal, para clasificar como anomalías todas aquellas que no se ajustan a dicha definición. Esto puede ocasionar varios problemas:

- Como están optimizados para datos normales, es posible que a la hora de detectar anomalías el modelo no sea muy eficaz.
- Estos modelos aumentan mucho su complejidad con la dimensión y el tamaño de los datos.

Para evitar esto, el algoritmo de **Isolation Forest** aísla las anomalías en vez de definir los datos normales. Para esto utiliza una estructura basada en árboles, ya que permite construir una estructura que aísle todos los puntos. Como las anomalías son muy diferentes a los puntos normales, quedarán aislados muy cerca de la raíz del árbol mientras que los puntos comunes no serán separados hasta zonas más profundas. A estos árboles se les llama **Isolation Trees**.

El algoritmo de Isolation Forest usa muchos Isolation Trees para ver cuáles son los puntos que de media son separados del resto de puntos más cerca de la raíz. Algunas características de los Isolation Forests son:

- Usar muestras pequeñas da buenos resultados porque reduce los efectos de identificar erróneamente datos buenos como outliers. Este error se llama empantanamiento o swamping. También se reducen los efectos al usar muestras pequeñas de datos de producir masking o enmascaramiento, que es detectar un grupo grande de anomalías como normal.
- No es necesario construir completamente el árbol ya que las anomalías se quedan en la parte superior. Esto reduce el coste computacional.
- Otra cosa que reduce el coste computacional es no usar medidas basadas en la distancia o en la densidad, como sí que se hacía en el clustering.
- Permite el uso de muestras grandes de datos con una gran cantidad de características.

El funcionamiento de los **isolation trees** es el siguiente. Estos van haciendo particiones de forma aleatoria hasta que todas las instancias se quedan aisladas. Esto provoca que las anomalías se encuentren cerca de la raíz del árbol, ya que, al haber pocas, se pueden aislar con pocas particiones.

Es por esto, que, en un bosque repleto de estos árboles, si se producen de media profundidades bajas para ciertos puntos, estos puntos probablemente sean anomalías.

Denominamos T a un nodo de un Isolation Tree, que puede ser un nodo externo sin hijos o un nodo interno con exactamente dos nodos hijos (T_l, T_r). Cada uno de los nodos internos está

compuesto por un atributo q y un valor de división p tal que $q < p$ divide los puntos que se encuentran en el nodo de T_l y T_r.

Si disponemos de una muestra de datos $X = \{x_1, \dots, x_n\}$ con n elementos pertenecientes a una distribución de dimensión d, entonces, utilizaremos una submuestra $X' \subset X$ con ψ puntos para construir un isolation tree. De forma recursiva iremos dividiendo X' seleccionando de forma aleatoria un atributo q y un valor p hasta que:

1. Todos los datos del nodo tengan el mismo valor que el atributo q.
2. Únicamente quede una instancia en un determinado nodo.

A partir de esto definimos **profundidad** h(x) de un punto x al número de ramas que hay que atravesar hasta llegar desde el nodo raíz hasta el nodo externo en el que se encuentra x.

Según esta definición, cuanta mayor profundidad sea la de un punto, menor es la probabilidad de que sea un outlier.

Sin embargo, decir que un punto tiene poca profundidad por lo que es un outlier, no nos sirve, es ambigua. Necesitamos **algún tipo de puntuación en función de h(x) que nos ayude a determinar si un punto es un outlier sin lugar a dudas.** Encontrar esta medida es problemática ya que la altura máxima del isolation tree crece en el orden de ψ, mientras que la altura media crece en el orden de $\log \psi$. Por tanto, si tratamos de normalizar el valor de h(x) en función de alguno de estos valores, nos encontramos con que no está limitado o que no es comparable directamente entre isolation trees.

La solución recae sobre la observación de que la estructura de un isolation tree es similar a la de un árbol binario de búsqueda, así que el valor medio de $h(x)$ para un nodo externo es la misma que en una búsqueda fallida. Para un conjunto de datos ψ, este valor es:

$$c(\psi) = \begin{cases} 2H(\psi - 1) - \left(2(\psi - 1)/n\right), & para\ \psi > 2, \\ 1, & para\ \psi = 2, \\ 0, & para\ cualquier\ otro\ caso, \end{cases} \quad (10)$$

Donde $H(i)$ es el i-ésimo número armónico, que puede ser estimado como $\ln i + 0{,}5772156649$ (constante de Euler)

Como $c(\psi)$ es la media de h(x) dado ψ, podemos utilizarlo para normalizar h(x), obteniendo así una puntuación o anomaly score $s(x, \psi)$ para un punto x,

$$s(x, \psi) = 2^{\frac{E(h(x))}{c(\psi)}} \quad (11)$$

Donde $E(h(x))$ es la media de $h(x)$ en un conjunto de isolation trees.

Gracias a esta medida podemos realizar las siguientes observaciones:

- Si el valor de s es cercano a 1, entonces estamos frente a una anomalía.
- Si el valor de s es menor a 0.5, podemos decir con seguridad que se trata de un dato normal.
- Si todos los puntos tienen un valor de $s \approx 0{,}5$, el conjunto de datos no presenta prácticamente anomalías y estas no son muy grandes. Esta observación se da porque

todos los puntos se separan aproximadamente a la misma distancia de la raíz del árbol, así que $E(h(x))\sim c(\psi)$.

Explicado ya el funcionamiento a grandes rasgos del algoritmo y sus medidas de profundidad, pasamos a explicar su funcionamiento con más detalle. Lo primero a destacar es que este algoritmo, como cualquier algoritmo de machine learning, tiene dos etapas:

- Entrenamiento, en la que construimos el modelo a partir de diferentes isolation trees.
- Evaluación, en la que decidimos si un punto es un outlier o no a partir del modelo previamente construido.

En el **entrenamiento** nos encontramos dos partes, el entrenamiento de cada uno de los árboles que conforman el bosque y el entrenamiento del bosque en general. Para entenderlo mejor hemos escrito un par de algoritmos, el algoritmo 1 y el algoritmo 2. Los diferentes isolation Trees se crean dividiendo el conjunto de ejemplos de entrenamiento dado hasta que cada uno de ellos este de manera independiente. Esto lo logramos cogiendo para cada árbol una muestra $X' \subset X$ extraída de forma aleatoria sin reemplazamiento de X.

Algorithm 1: iForest(X,t,ψ)

Entradas: X - datos de entrada, t - numero de arboles, ψ - tamao de la muestra

Salidas: Un conjunto de isolation trees

Inicializar Forest
for $i = 1$ *hasta* t **do**
    $X' \leftarrow muestra(X, \psi)$
    $Forest \leftarrow Forest \cup iTree(X')$
return Forest

Algoritmo 1:Iforest

Algorithm 2: iTree(X,t,ψ)

Entradas: X' - datos de entrada
Salidas: Un Isolation Tree
if *X' no puede ser dividido* **then**
    return $exNode\{Size \leftarrow |X'|\}$
else
    Siendo Q una lista de atributos de X'
    Se elige un atributo de manera aleatoria $q \in Q$
    Se elige de manera aleatoria un punto de corte q entre los valores maximos y minimos de q
    $X_t \leftarrow filtrar(X', q < p)$
    $X_r \leftarrow filtrar(X', q \geq p)$
    return $inNode\{Izquierda \leftarrow iTree(X_t), Derecha \leftarrow iTree(X_r), AtributoCorte \leftarrow q, ValorCorte \leftarrow p\}$

Algoritmo 2: iTree

En la **etapa de evaluación**, para cada punto x se calcula su profundidad h(x) contando el número de ramas e que hay que pasar desde la raíz hasta la hoja en la que se encuentra y sumando un valor c(size) que se calcula con la ecuación 8, con size el tamaño del isolation tree. Esto se realiza

en cada uno de los isolation trees del isolation forest, obteniendo finalmente la anomaly score del punto en cuestión. El algoritmo en cuestión se puede observar en el algoritmo 3.

Algorithm 3: $PathLength(x, T, hlim, e)$

Entradas: x - una instancia, T - un isolation tree, hlim - el lmite de altura, e - la longitud del camino actual que esta inicializado a 0 cuando se llama la primera vez

Salidas: La longitud del camino de x

```
1 if T es un nodo externo o e ≥ hlim then
      /* c(.)  esta definido en la ecuacion 8                    */
2     return e + c(T.size)
3 a ← T.AtributoCorte
4 if x_a < T.ValorCorte then
5     return PathLength(x, T.Izquierda, hlim, e + 1)
6 else if x_a ≥ T.ValorCorte then
7     return PathLength(x, T.Derecha, hlim, e + 1)
```

Algoritmo 3: PathLength

Es necesario mencionar que para evaluar si un punto es un outlier o no hace falta construir el árbol. No se puede decidir si un punto es outlier o no si no estaba en la etapa de entrenamiento porque el árbol de outliers hubiese cambiado si se hubiese considerado tal punto. Por tanto, este método es eficaz, pero hay que reentrenar el modelo cada vez que queramos decidir si un nuevo punto es un error o no lo es.

3.2.4.3 One class SVM

Las máquinas de vectores soporte (SVM son las siglas del inglés Support Vector Machine) tienen su origen en trabajos sobre la teoría del aprendizaje estadístico y fueron desarrolladas en los años 90 por Vapnik y sus colaboradores. Al principio, las máquinas de vectores soporte se pensaron para la clasificación binaria de clases. Sin embargo, ahora se han ajustado para poder resolver otros tipos de problemas (regresión o clasificación no binaria). Han sido usadas con éxito en campos muy diversos tales como la visión artificial hasta la clasificación de proteínas pasando por el procesamiento de lenguaje natural. De hecho, desde su introducción, han ido ganando reconocimiento gracias a sus sólidos fundamentos teóricos.

Las máquinas de vectores soporte son clasificadores lineales, ya que crean separadores lineales o hiperplanos, ya sea en el espacio original de los datos, si estos son linealmente separables, o en un espacio transformado, si no son linealmente separables en el espacio original. Para hacer esta separación en espacios trasformados, se usan unas funciones que se denominan kernel, que, al estar de manera implícita en el modelo, son muy eficientes.

A grandes rasgos la diferencia entre las SVM y el resto de modelos, es que las SVM no tratan de reducir el error cometido por los ejemplos de entrenamiento. La función de coste de las máquinas de vectores soporte se basa en la minimización del denominado *riesgo estructural.* Este *riesgo estructural*, tiene como idea seleccionar un hiperplano de separación que equidiste de los ejemplos más cercanos de cada clase para conseguir el *margen máximo* a cada lado del

hiperplano. Como se usan solo estos ejemplos para definir el hiperplano, al entrenar el modelo solo se consideran estos lo que aumenta en gran medida la eficiencia.[7]

Nuestro algoritmo en particular, **one class svm**, se basa como su propio nombre indica en una SVM, pero considerando en vez de dos clases para separar solo una. Para ello, queremos que nuestro modelo nos dé una función que devuelva +1 en la región en la que se encuentran la mayoría de puntos y -1 en el resto del espacio. La estrategia principal para conseguir esto es transformar el espacio de entrada R a un espacio de mayor dimensión F, mediante una función $\Phi(x_i)$ para poder separar los datos mediante el hiperplano de mayor margen. Estas transformaciones se pueden llevar a cabo mediante un kernel, como puede ser el gaussiano,

$$k(x,y) = e^{-\frac{\|x-y\|^2}{c}} \ (12)$$

Nuestra estrategia es la de convertir los datos en el espacio de características correspondiente al kernel, y separarlos del origen con el máximo margen posible mediante el kernel. Para cualquier nuevo punto x, saber si es un outlier o no estará determinado evaluando en qué lado del hiperplano cae, en el espacio de características.

Para encontrar un hiperplano que separe los datos hay que obtener la solución del siguiente problema:

$$\min_{w\in F,\xi\in\mathbb{R}^n,\rho\in\mathbb{R}} 0.5\|w\|^2 + {}^1\!/_{\nu n}\sum_i \xi_i - \rho_i \ (13)$$

Siendo $w \cdot \Phi(x_i) \geq \rho - \xi_i,\ \xi_i \geq 0\ y\ \nu \in (0,1]$. Como las variables ξ_i no nulas son penalizadas en la función objetivo, podemos esperar que si w y ρ solucionan este problema, entonces la función de decisión $f(x) = sgn((w \cdot \Phi(x_i)) - \rho)$ será positiva para la mayoría de ejemplos x contenidos en el conjunto de entrenamiento, siempre que el termino de regularización $\|w\|$ sea pequeño. El balance entre la regularización y la función de decisión es controlada por ν. Derivando la función de decisión con nuestro kernel, expuesto en la ecuación 10, saldría la siguiente ecuación:

$f(x) = sign(\sum_i \alpha_i k(x_i,x) - \rho)\ siendo\ k\ el\ kernel\ gaussiano$ *(14)*

Donde los coeficientes se consiguen como solución al problema dual:

$$min_\alpha \frac{1}{2}\sum_{ij} \alpha_i\alpha_j k(x_i,x_j)\ \ siendo\ 0 \leq \alpha_i \leq \frac{1}{\nu l}, \sum_i a_i = 1 \ (15)$$

Para llegar a estos resultados, hay que aplicar el Lagrangiano del problema inicial, pero al no ser este un trabajo de fin de grado de desarrollo de algoritmo, sino simplemente de su aplicación, no detallaremos su desarrollo teórico [7].

Mediante estas ecuaciones lo que conseguimos es un hiperplano en otro espacio de características a partir del cual separamos los valores más “raros” del conjunto de entrenamiento, del resto de valores. Así, cuando queramos predecir si un valor es outlier o no, simplemente tenemos que reproyectarlo en el nuevo espacio de características y ver en qué lado del hiperplano se encuentra.

De hecho, al tener la función de decisión presentada en la ecuación 12, simplemente con introducir los valores en ella ya sabríamos si un determinado punto es un valor anómalo o no [6].

3.3 Deep learning

3.3.1 Introducción

En este apartado trataremos un tipo especial de machine learning que se llama Deep learning. Para explicar que es el Deep Learning, primero hay que entender cómo funciona el cerebro humano.

En 1888, Ramón y Cajal demuestra que el sistema nervioso está compuesto por una red de células individuales que están interconectadas entre sí. Esto es un paso muy grande a la hora de empezar a entender nuestro cerebro. También demuestra que la información fluye desde las dendritas hasta el axón atravesando el soma.

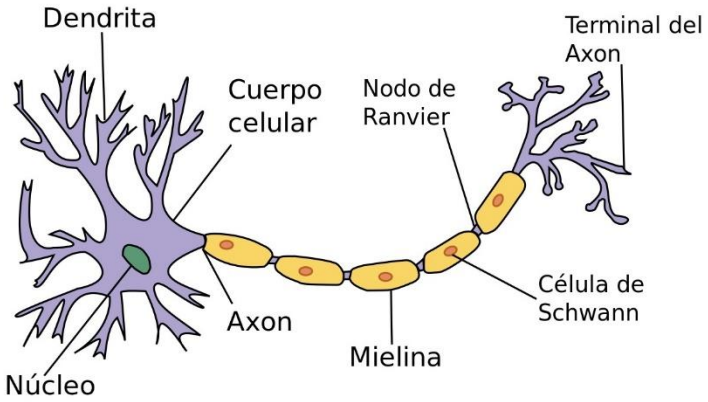


Figura 7: Neurona humana. Fuente: https://ast.wikipedia.org/wiki/Neurona

Nuestro cerebro está compuesto por 10^8 neuronas. Cada neurona se comporta de manera individual como un pequeño procesador sencillo. Se puede imaginar que las dendritas podría ser el canal de entrada de la información, el núcleo es el órgano de cómputo y el axón es el canal de salida que envía impulsos a otras neuronas.

El cerebro de un ser vivo se modela durante su desarrollo. Sí que hay algunas cualidades que son innatas, pero también hay otras muchas que son adquiridas por la influencia de la información que entra por sus sensores. Esta información transforma el sistema nervioso mediante creación de nuevas conexiones, ruptura de otras, mediante el modelado de las intensidades en las que se pasa la información de una neurona a otra o mediante la muerte y creación de nuevas neuronas.

Este simple funcionamiento permite que nuestro cerebro tenga una cantidad de computo inimaginable. Es por eso, que distintos expertos en computación han tratado de emular esta estructura en un ordenador con el fin de alcanzar una funcionalidad similar.

Para hacer esta simulación, hacen falta 3 conceptos clave:

- **Procesamiento en paralelo:** Cuando vemos una imagen las neuronas no actúan de manera secuencial, sino que actúan todas simultáneamente para procesarla lo más rápido posible.
- **Memoria distribuida:** Nuestro cerebro no guarda la información en un solo punto, sino que los datos están distribuidos por todo nuestro cerebro de manera redundante, para evitar pérdidas de información.
- **Adaptabilidad al entorno:** Nuestro cerebro es capaz de coger ejemplos individuales, que se ganan a través de la experiencia y generalizarlos de manera que podamos aplicar este aprendizaje en otros casos.

La idea de estos científicos fue copiar nuestro cerebro de la manera más exacta que pudieran. Para ello, observaron que nuestro cerebro era un conjunto de neuronas conectadas organizadas por capas. Así pues, un sistema que simulase un cerebro humano tendría que estar construido por un conjunto de neuronas, en este caso artificiales, organizadas en capas con una entrada, que simularía nuestros sentidos, y una salida [8].

3.3.2 Perceptrón

Estás neuronas artificiales se denominaron perceptrones. Un Perceptrón es un dispositivo de computación con umbral U y n entradas reales $X_1, \dots, X_n$ a través de arcos con pesos $W_1, \dots, W_n$ y que tiene salida 1 cuando $\sum_i w_i x_i \geq U$ y 0 en caso contrario.

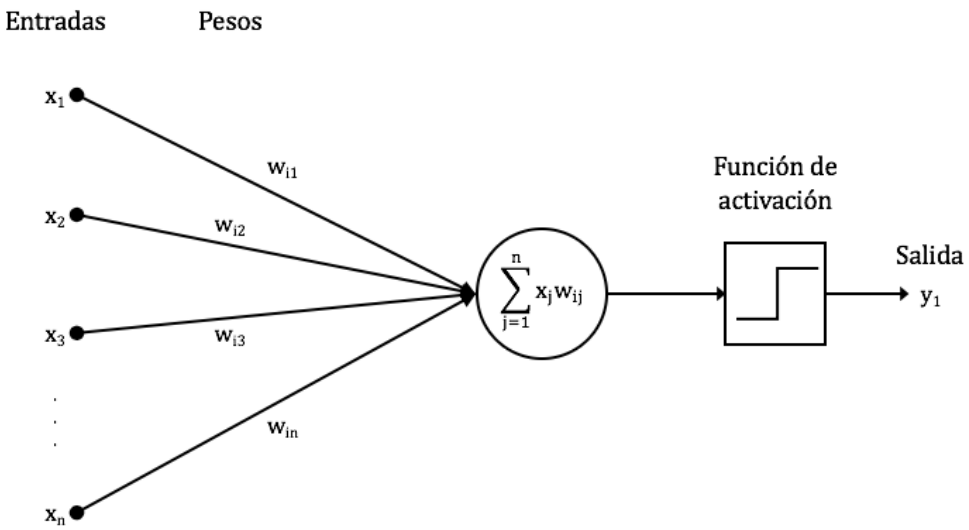


Figura 8: Neurona artificial. Fuente: https://www.avansis.es/inteligencia-artificial/que-son-las-redes-de-neuronas-artificiales-parte-i/

El origen de las entradas es irrelevante. Puede venir de otros perceptrones o de cualquier clase de unidad de computación. Geométricamente, es fácil identificar cual es la forma que dibuja un Perceptrón en el espacio. Crea un hiperplano que separa los puntos cuya agregación no supera el umbral de los que sí lo superan.

La salida de la neurona responde a una función de activación que depende de la agregación de todos los valores de entrada y del valor del umbral. Normalmente esta función devuelve un 0 o un 1 dependiendo de si el valor de esta agregación es positiva o negativa, pero puede dar

diferentes valores. Estas funciones de activación se suelen utilizar en las redes neuronales multicapa de las que ya hablaremos posteriormente.

Un Perceptrón simple como este puede separar dos conjuntos de puntos linealmente separables. Dos conjuntos de puntos A y B en un espacio n-dimensional son linealmente separables si existen n+1 números reales $w_1, \dots, w_{n+1}$ tales que cada punto $(x_1, \dots, x_n)$ A satisface $\sum_i w_i x_i \geq w_{n+1}$ y que cada punto $(x_1, \dots, x_n)$ B satisface que $\sum_i w_i x_i < w_{n+1}$.

Si las clases a separar son linealmente separables, el algoritmo del Perceptrón converge a una solución correcta en un número finito de pasos para cualquier elección inicial de pesos.

Sin embargo, algo de lo que no tenemos que olvidarnos es que esto es machine learning, luego detrás tiene que haber un algoritmo de entrenamiento. Este algoritmo es supervisado y usa n vectores que se pueden clasificar en dos conjuntos P y N en el espacio extendido n+1-dimensional. Se considera que el umbral es una entrada de tendencia con valor fijo 1, ya que $\sum_i w_i x_i < w_{n+1}$ equivale a $\sum_i w_i x_i - w_{n+1} < 0$ [9].

El algoritmo de entrenamiento es el siguiente:

Algorithm 1: PerceptronFit(X,P,N)

Entradas: X - vectores de entrada con al menos un elemento, P - indices de los vectores x cuya salida tiene que ser 1, N - indices de los vectores x cuya salida tiene que ser 0

Salidas: w - vector que separa los dos conjuntos de vectores

```
1 w ← random(0, 1, len(X_0))
2 while queden vectores mal clasificados do
3     r ← randint(0, len(X))
4     aux ← x_r
5     if x ∈ P ∧ w * x < 0 then
          /* Hay un error */
6         w ← w + aux
7     if x ∈ N ∧ w * x ≥ 0 then
          /* Hay un error */
8         w ← w − aux
9 return w
```

Algoritmo 4: Algoritmo de entrenamiento del Perceptrón

3.3.3 Adaline

Por otro lado, a la vez que surgía el perceptrón, también aparecía el adaline y su regla de aprendizaje llamada *least mean square*.

El adaline es prácticamente idéntico al perceptrón, excepto en su función de transferencia, la cual es una función de tipo lineal en vez hacer un corte fuerte como sí que hace el perceptrón. El adaline presenta el mismo problema que el perceptrón, no puede resolver problemas cuyos datos no son linealmente separables. Sin embargo, el algoritmo de aprendizaje de este tipo de neuronas es mucho más potente que el de los perceptrones ya que se basa en el error cuadrático medio. De hecho, esta regla sirvió de inspiración para más algoritmos posteriores.

El termino Adaline es una sigla, aunque su significado ha cambiado ligeramente desde los años sesenta cuando decayó el estudio de las redes neuronales. Inicialmente se llamaba *ADAptive Linear NEuron* (Neurona lineal adaptativa), para pasar después a ser *adaptative linear element* (Elemento lineal adaptativo). Este cambio se produjo ya que este tipo de neuronas son dispositivos que constan únicamente de un único elemento de procesamiento y por tanto no es como tal una red neuronal.

Esta unidad de procesamiento de lo que se encarga es de realizar las sumas y los productos entre los valores de entrada y los pesos, y aplica una función de salida, la cual es una función lineal en este caso, para obtener un único valor de salida.

El adaline es **adaptativo** ya que existe un procedimiento bien definido para modificar los pesos con el objetivo de hacer posible que el dispositivo consiga obtener la salida esperada dada la entrada correspondiente. Estos valores dependerán de la función de salida que tenga la neurona. No será lo mismo si tiene la función identidad, en el que los valores de la entrada son sumados y multiplicados por los pesos y expulsados por la neurona, que si tiene otra función más complicada. En redes neuronales más complejas, la función de salida o de activación será clave para la correcta adaptación de las redes.

Por otra parte, el adaline es **lineal** porque la salida es una función lineal sencilla de los valores de la entrada.

Por último, es una **neurona** tan solo en el sentido del elemento de procesamiento. Como ya hemos explicado, quizás sería más correcto llamarlo elemento lineal, evitando por completo la definición de neurona [9].

3.3.4 Redes neuronales multicapa y algoritmo backpropagation

Lo interesante de estos dos modelos previamente expuestos es que no permiten más que el tratamiento de datos lineales. Es por eso, que se pensó como mejorar el funcionamiento de las neuronas para poder tratar la no linealidad, que se encuentra en la mayor parte de lugares de la naturaleza. Para ello contábamos con dos estrategias:

- **El aumento del número de neuronas:** Al aumentar el número de unidades de procesamiento, a pesar de que cada una de ellas tuviese individualmente un funcionamiento lineal, conseguimos que el conjunto tenga un funcionamiento no lineal.
- **Introducir funciones de activación no lineales:** En el modelo adaline se contaba con funciones de activación lineales. Esto como es obvio, provocaba que la salida fuese lineal. Sin embargo, en este caso contamos con más funciones que podemos incluir como funciones de activación y que no llevan consigo este lastre.

Mediante estas redes, a pesar de que constan de una morfología diferente a las que se encuentran en el interior de nuestro cerebro, conseguimos las siguientes características:

- **Auto-organización y adaptabilidad:** Utilizan algoritmos de aprendizaje adaptativo y auto-organización, por lo que ofrecen mejores posibilidades de procesado robusto y adaptativo.
- **Procesado no Lineal:** Aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumenta su inmunidad frente al ruido.

- **Procesado paralelo:** Normalmente se usa un gran número de nodos de procesado, con alto nivel de interconectividad.

La morfología de las redes depende de los datos a tratar. No usaremos el mismo tipo de redes en el caso de usar como datos de entrada imágenes que si usamos series temporales, como es nuestro caso [10][11].

Como ya hemos dicho en el caso del adaline, la función que se usa en este caso para mejorar el rendimiento de la red neuronal y hacerla "aprender", es minimizar el error cuadrático medio entre los valores esperados y los valores de salida. Esta etapa se llama **fase de entrenamiento** y se basa en determinar los pesos que definen el modelo de red neuronal de manera iterativa. Para entrenar la red hace falta un algoritmo especial que se llama *backpropagation.*

Este algoritmo surgió ante la necesidad de un algoritmo eficiente que nos permita adaptar todos los pesos de una red multicapa y no solo los de la capa de salida, como en el caso del adaline o del perceptrón simple. En estos casos simples, era fácil saber cuáles eran los pesos que se tenían que poner en la capa de salida, pero al aumentar el número de capas y el número de neuronas, se hace mucho menos evidente cuales son las características que hay que poner para que la red neuronal funcione correctamente.

Entonces si realmente no sabemos cuáles son las características que tienen que aprender las capas ocultas de la red… ¿Cómo entrenamos estas redes? La respuesta es que hay que cambiar de prisma nuestra observación. En vez de fijarnos en los cambios de los pesos, nos fijaremos en los cambios de las salidas. Nuestro objetivo será acercar lo más posible las salidas de nuestra red a las salidas esperadas. Esta estrategia funcionará en la mayoría de los casos, solo que empezará a ser más problemática en los problemas no convexos.

Para explicar este algoritmo, usaremos el filtro lineal, aunque es aplicable a cualquier filtro interno de las neuronas, aunque elegiremos este porque es el más "visual" y el más sencillo de ejemplificar.

$$y = \sum_i w_i x_i = W^T X \quad (16)$$

Nuestro objetivo es minimizar la suma de los errores de la y real contra la y predicha por la red en los ejemplos de entrenamiento. Se podría resolver el problema analíticamente, pero la solución sería difícil de generalizar además de ser poco eficiente. Es por tanto que diseñaremos un algoritmo iterativo. Para explicar el algoritmo iterativo vamos a poner un ejemplo de la vida real.

Imaginemos que todos los días desayunamos en la cafetería de la universidad. Tomamos para desayunar café, zumo y tostadas, pero sólo nos pasan la factura al final de la semana, sin decirnos cuánto cuesta cada producto. ¿Seríamos capaces de saber cuánto vale cada uno de los productos? La respuesta es que si, tras varias semanas.

Lo primero que tendríamos que hacer es empezar con una estimación, que puede ser aleatoria, de los precios, y posteriormente irlos ajustando de tal manera que encajen con los importes facturados

Cada semana, el total nos impone una restricción lineal sobre los precios de las cantidades consumidas:

$$total = w_{cafe}x_{cafe} + w_{zumo}x_{zumo} + w_{tostada}x_{tostada} (17)$$

Los precios son los pesos $w = (w_{cafe}, w_{zumo}, w_{tostada})$ y las entradas corresponderían a las cantidades consumidas de cada producto.

La primera semana, la cuenta nos ha costado 8,50 euros. Lo que nos deja la situación de la Figura

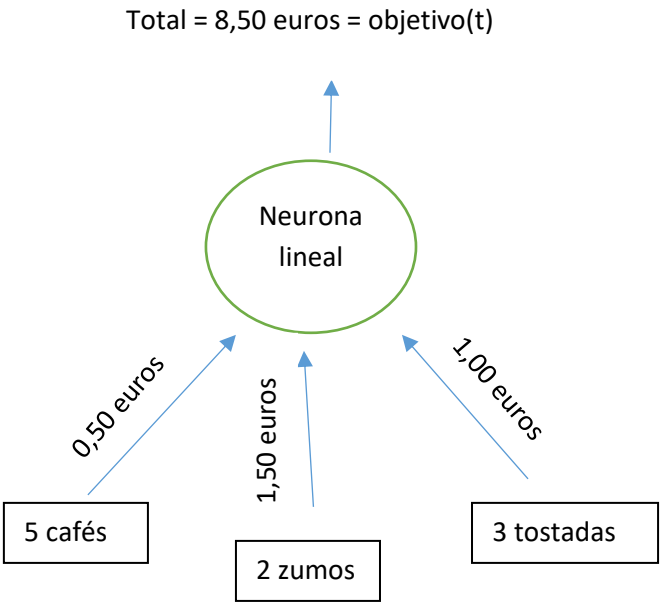


Figura 9: Neurona que nos hace la cuenta de la cafetería

9, estos serían los pesos reales que debería tener nuestra red para predecir la cuenta.

Sin embargo, en la primera semana los precios que habíamos estimado eran 0.50 euros por cada producto. Lo que nos deja con un error residual de 3 euros y medio. Esto se puede observar en la Figura 10. Para corregir los pesos de nuestra calculadora de facturas usamos la regla delta:

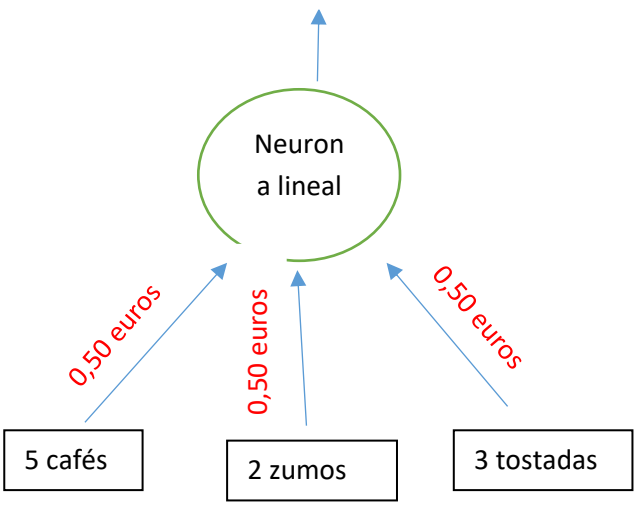


Figura 10: Precios que estimamos la primera semana desayunando en la universidad

$$\Delta w_i = \eta x_i (t - y)\ (18)$$

η es la tasa de aprendizaje utilizada.

Entonces para ajustar los pesos de nuestro calculador de facturas, usaremos la regla delta. Como el error residual es 3,50 euros (esto es la parte de la regla delta $(t - y)$) y siendo la tasa de aprendizaje $\eta = \frac{1}{35}$, los nuevos pesos son los que se muestran en la Figura 11.

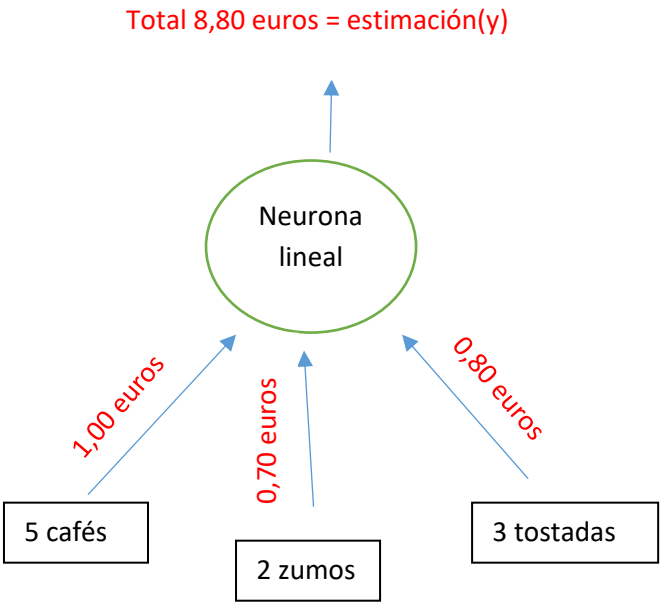


Figura 11: Precios que estimamos la segunda semana desayunando en la universidad

De esta manera, de manera iterativa y tras una serie finita de pasos, se llegaría a unos pesos que se corresponderían con los precios originales de los alimentos.

Si escribimos esto de una manera más formal nos queda:

- $Error = \frac{1}{2}\sum_{n \in training}(t^n - y^n)^2\ (19)$
- Derivada del error: $\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_n \frac{\partial y^n dE^n}{\partial w_i dy^n} = -\sum_n x_i^n (t^n - y^n)\ (20)$
- Ajuste de los pesos en proporción al error:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_n x_i^n (t^n - y^n)\ (21)$$

Con estas fórmulas, como hemos ejemplificado anteriormente, nos iremos acercando poco a poco, iteración tras iteración a la mejor solución posible. La tasa de aprendizaje tiene que ser pequeña para que no nos pasemos el punto mínimo de la función que se puede observar en la Figura 12 [12].

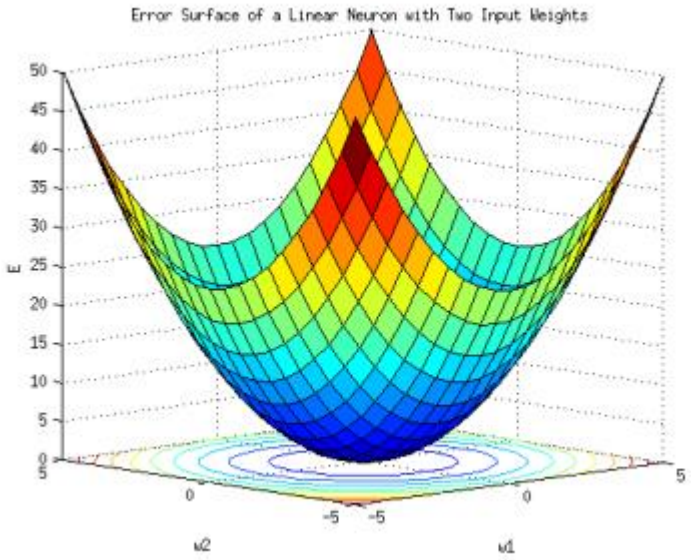


Figura 12: Descenso por gradiente del error. Fuente: Fernando Berzal, Backpropagation.

Después de que la red "aprenda" o se entrene mediante el algoritmo que acabamos de explicar, hay que hacer **una fase de prueba**. Esta fase hay que hacerla ya que en ocasiones el modelo se ajusta demasiado a las peculiaridades presentes en los patrones de entrenamiento, perdiendo su habilidad de generalizar el aprendizaje a casos nuevos.

Para realizar esta separación en dos fases, hay que dividir el conjunto de los datos en dos datasets, uno de train y uno de test. Con el dataset de train se realiza la fase de entrenamiento y con el dataset de test la fase de prueba. En ocasiones, el dataset de train se divide a su vez en dos, en train y en validación. El dataset de validación sirve para ajustar los híper-parámetros.

Normalmente, la fase de prueba consiste en probar a predecir o clasificar con el dataset de test y sacar una métrica, que simboliza la capacidad de generalización de la red. Sin embargo, en nuestro caso la métrica no nos sirve para testear la red porque lo que estamos buscando son outliers que no están etiquetados.

3.3.5 Funciones de base y activación

Una red neuronal se puede clasificar por la función de base y de activación que usa. Cada neurona suministra un valor y_j a su salida. Este valor se propaga a través de la red mediante conexiones unidireccionales hacia otros nodos de la red. Asociada a cada conexión hay un peso sináptico denominado $\{w_{ij}\}$, que determina el efecto de la neurona j-ésima sobre la neuronal i-ésima.

La función base es la que se encarga de tratar todas las entradas provenientes de otras neuronas junto con umbral θ_i obteniendo u_i. La salida final y_i se obtiene aplicando la función de activación sobre u_i.

La función base suele ser una de dos:

- **Una función lineal de tipo hiperplano**, en la que el valor de red es una combinación lineal de las entradas,

$$u_i(w,x) = \sum_{j=1}^{n} w_{ij}x_j \ (22)$$

- **Una función radial de tipo hiperesférico,** que es una función de base de segundo orden no lineal,

$$u_i(w,x) = \sqrt{\sum_{j=1}^{n} \left(x_j - w_{ij}\right)^2} \ (23)$$

Por otra parte, como ya hemos dicho la función de activación transforma este valor que expulsa la función de base, para añadir no linealidad a la salida. Las funciones de activación más comunes son las siguientes:

- Función sigmoide

$$f(u_i) = \frac{1}{1 + e^{-u_i}} \ (24)$$

- Función gaussiana

$$f(u_i) = c * e^{-{u_i}^2} \ (25)$$

3.3.6 Estructuras de conexión de las redes neuronales multicapa

Todas las redes neuronales tienen dos componentes principales: Las neuronas y una matriz de pesos.

Además, dentro de una red neuronal se pueden distinguir tres tipos de capas:

- La capa de entrada
- La capa oculta
- La capa de salida

Asimismo, las capas de la red neuronal se pueden conectar de distintas maneras, creando así diferentes tipos de pesos entre las neuronas:

1. **Conexiones hacia delante:** Estás son las más comunes, los valores de las neuronas de las capas anteriores pasan a las capas siguientes por medio de conexiones neuronales hacia delante.
2. **Conexiones hacia atrás:** Al contrario que las conexiones hacia delante, estas propagan los valores de las capas posteriores hacia las anteriores.
3. **Conexiones con retardo:** Los elementos con retardo se incorporan en las conexiones para implementar modelos dinámicos y temporales, es decir, modelos que precisan de memoria. Este tipo de conexiones no son muy usuales en redes neuronales convencionales, pero, en nuestro caso, veremos que en Deep learning se han desarrollado alternativas que las usan de manera efectiva.
4. **Conexiones laterales:** Las neuronas de una misma capa se conectan entre sí.

Por último, las neuronas pueden estar completamente conectadas a sus vecinas o solo conectadas de manera parcial.

En síntesis, las redes neuronales son estructuras complejas que se desarrollaron hace bastantes años (en los años entre 1960 y 1980), que se abandonaron una temporada pero que han resurgido con otro nombre: Deep Learning.

3.3.7 Introducción al Deep learning

Como ya hemos dicho, las redes neuronales han sido el precedente a una revolución dentro de la computación: El Deep Learning. Una posible definición del Deep Learning es decir que es una red neuronal con abundantes capas ocultas, de tal forma que el aprendizaje sea profundo en todas sus capas [8].

Uno de los problemas que tuvieron las redes neuronales cuando surgieron fue la capacidad de computo. Al tener tantas unidas de procesamiento, una por neurona, necesitaba muchos cálculos rápidos y paralelos para poder entrenarse y para poder predecir. Asimismo, el Deep Learning no se podía desarrollar en este entorno ya que son redes neuronales muy grandes.

Además, a la vez que surgieron las redes neuronales, también surgieron las máquinas de soporte vectorial, las cuales ya hemos mencionado, que tenían muy buen rendimiento y sin embargo no tenían tanto coste computacional.

A pesar de esto, con el paso de los años y la evolución de los ordenadores, los expertos se han dado cuenta que a pesar de la robustez de las SVM para según qué cálculos y objetivos, el Deep learning representa una buena alternativa para aprovechar toda la potencia del hardware, ya que se puede paralelizar, lo que conlleva que, para redes no muy grandes, la velocidad de entrenamiento de una SVM o de una red neuronal sea similar.

Como último punto a favor de este tipo de machine learning, se han desarrollado distintas variantes de redes neuronales dentro del ámbito del Deep Learning que permiten el tratamiento de datos específicos de mejor manera. Por ejemplo, redes convolucionales para detección de objetos en imágenes o redes recurrentes para el reconocimiento del habla. En este trabajo de fin de grado, se usan solo las redes recurrentes por lo que no vamos a hablar de ninguna otra para no extender demasiado la longitud de este [13].

3.3.8 Redes neuronales recurrentes

Estas redes son especiales para tratar series numéricas. Para optimizar este propósito se disponen de distintas técnicas:

- Presentan retro-alimentación, esto significa que la salida de una neurona se usa como entrada de sí misma o como entrada de otra neurona que está conectada a sí misma.
- La salida de la neurona se calcula usando valores de entrada y salida obtenidos en tiempos anteriores. Es decir, de predicciones pasadas.

Las diferencias de estas redes con respecto a las redes neuronales habituales es que estas segundas aceptan una entrada de tamaño fijo (por ejemplo, una imagen) y producen un vector de salida de tamaño fijo (por ejemplo, probabilidades de las distintas clases), mientras que las redes recurrentes operan sobre secuencias de vectores que no tienen tamaño fijo. La transformación recurrente es fija y se puede aplicar tantas veces como sea necesario.[13]

Además, otra cosa que tienen de especial las redes neuronales recurrentes es que cuentan con un estado, y para hacer la siguiente predicción usan el estado anterior. Para calcular, el estado de la neurona recurrente se usa la siguiente formula:

$$h_t = f_W(h_{t-1}, x_t) \ (26)$$

Donde h_t es el estado actual, h_{t-1} el estado anterior, x_t es la entrada en el momento actual y f_w es una función cualquiera. Una función muy usada para esto es la tangente hiperbólica.

Por tanto, las redes neuronales recurrentes (RNNs) se podrían definir como unas redes neuronales especiales que aceptan como entrada un vector x, que tienen internamente un vector de estados h, y que combinan x y h mediante una función (fija pero cuyos parámetros se van aprendiendo) y que producen como salida un vector y [14].

Lo más importante que hay que ver en esta definición es que la salida no solo está influenciada por la entrada x, sino por toda la historia de las entradas que hubo en el pasado y que afectan al estado h.

Este tipo de redes se pueden utilizar, por ejemplo, para predecir la siguiente letra de una palabra. Esto se puede observar en la Figura 13.

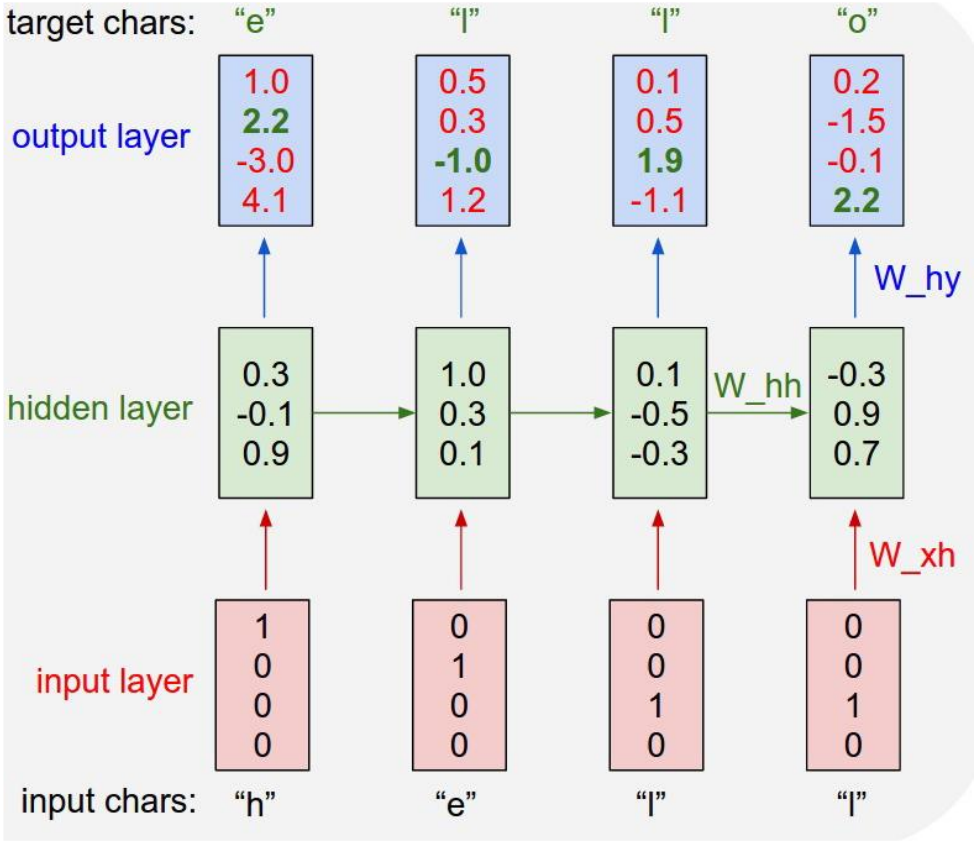


Figura 13: Predicción de una red neuronal recurrente.

Para inicializar una red recurrente, hay que definir 3 matrices W de manera aleatoria, que afectan a la entrada, al estado oculto y a la salida. El estado oculto h es inicializado con el vector nulo, ya que el estado es nulo al no tener ninguna historia la red.

Para predecir el siguiente valor, primero se actualiza el vector de estados con el estado anterior y la entrada actual y después se calcula la salida actual.

Para hacer entrenar la red, se usa el mismo algoritmo que en el caso de las redes neuronales multicapas normales: El algoritmo *backpropagation*.

Sin embargo, este tipo de redes neuronales recurrentes convencionales tienen un problema. ¿Qué pasaría si un ejemplo en particular es muy relevante, pero ha pasado hace muchas iteraciones? La red va olvidando su estado de la historia más lejana.

Para entenderlo mejor voy a poner un ejemplo. Imaginemos un sistema de texto predictivo. Si tratamos de predecir la siguiente palabra en una frase como "las nubes están en el", es obvio que la siguiente palabra va a ser "cielo". Si estamos en un caso así las redes neuronales recurrentes convencionales no necesitan más ayuda para adivinar la siguiente palabra.

Pero imaginemos una biografía, un texto más largo. En una de las primeras frases pone "Yo crecí en Francia", y hacia la mitad de la biografía escribimos "Hablo". Una red neuronal recurrente convencional sabrá que la siguiente palabra tendrá que ver sobre el lenguaje, pero si tiene que decidir que lenguaje necesitamos el contexto de que crecí en Francia, de mucho más atrás, al ser este hueco tan grande, las redes neuronales recurrentes no son capaces de recordar esa información.

Es por esto que existen otro tipo de redes que se ajustan a esto: las redes neuronales recurrentes de segundo orden. En la siguiente sección vamos a ver un ejemplo de redes neuronales recurrentes de segundo orden que se usa posteriormente en nuestro trabajo de fin de grado: las LSTM.

3.3.9 LSTM

Estas redes son una red neuronal recurrente especial que es capaz de almacenar dependencias de tiempos más largos. Fueron introducidas por Hochreiter y Schmidhuber y fueron refinadas y popularizadas por mucha gente en trabajos posteriores. Trabajan de manera muy buena en una gran cantidad de problemas, y ahora están muy usadas en reconocimiento de lenguaje natural y otros análisis de series [14].

Las LSTM están diseñadas explícitamente para evitar el problema de las dependencias de espacios largos de tiempo. Recordar información por periodos largos de tiempo es prácticamente su comportamiento por defecto, no algo complicado para aprender.

Una red neuronal recurrente tiene la forma de un conjunto de módulos de redes neuronales. En una red neuronal recurrente convencional, este módulo que se repite tendría una estructura muy simple con solo una capa tanh (tangente hiperbólica) [15].

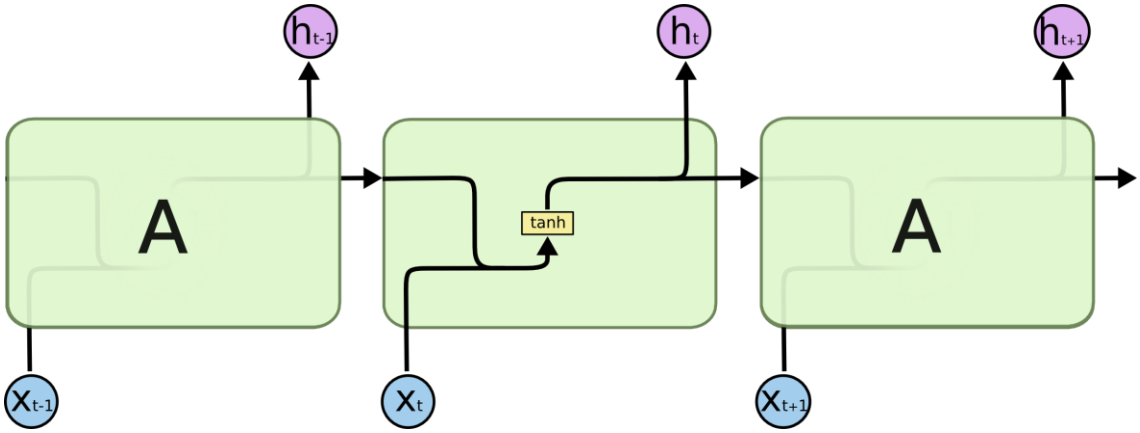


Figura 14: RNN convencional. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Las LSTM también tienen esta estructura de bloques, solo que el modulo que se repite cambia un poco. En vez de tener solo una capa de red neuronal, tiene 4, interactuando de una forma muy especial como se puede observar en la Figura 15.

La clave de las LSTM es el estado de la celda, la línea horizontal que corre por encima de la Figura 15. Esta línea es como un cinturón de toda la red, une toda la red solo con alguna interacción mínima lineal.

Las LSTM son capaces de añadir y eliminar información del estado de la celda, mediante estructuras llamadas puertas. Estas puertas son una manera de dejar pasar información cuando sea necesario. Están compuestos por una capa de red neuronal sigmoide (σ) junto con un punto con una operación de multiplicación [15].

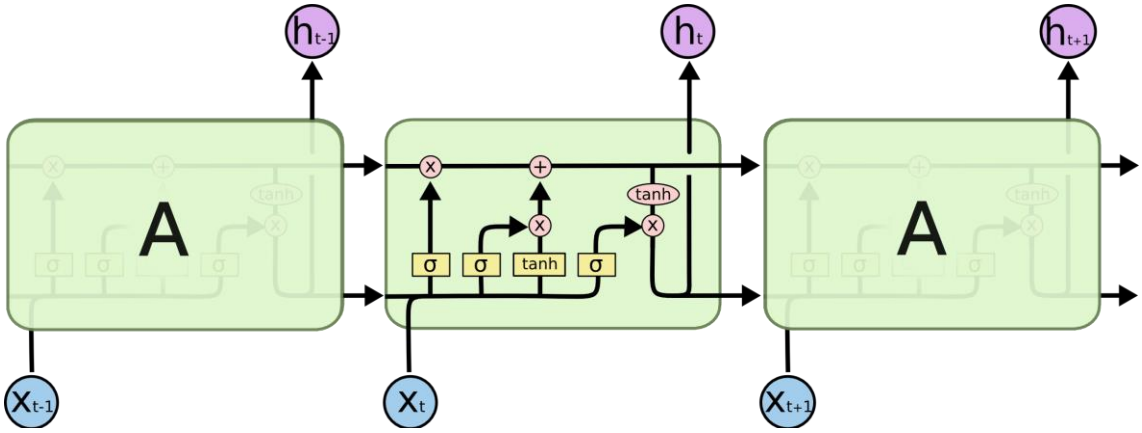


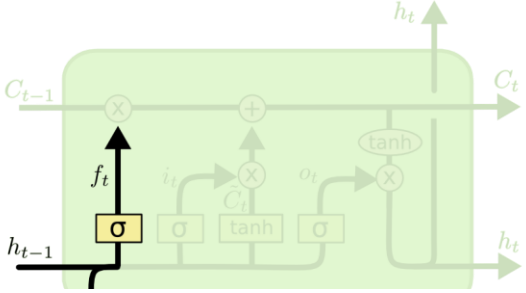
Figura 15: LSTM. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Las capas sigmoide (σ) devuelven un valor entre 0 y 1 que dice cuanto de un determinado componente puede pasar o no.

Una LSTM tiene 3 de estas puertas, para proteger y controlar el estado de la celda.

Lo primero que tiene que hacer una LSTM es decidir qué información vamos a tirar del estado de la celda. Esto se lleva a cabo en una capa de sigmoide que se llama "forget gate layer". Mira a h_{t-1} y a x_t, y saca como salida un número entre 0 y 1 para cada número en el estado de la celda C_{t-1}. Un 1 significa "Esto es muy útil, no lo olvides" mientras que un 0 significa "Esto no sirve para nada, olvídalo".

Volviendo al ejemplo del predictor de texto, cuando vemos un pronombre sabemos el género de las palabras que hay que poner. Sin embargo, cuando vemos otro pronombre esta información ya no sirve y hay que recordar el género del nuevo pronombre [15].

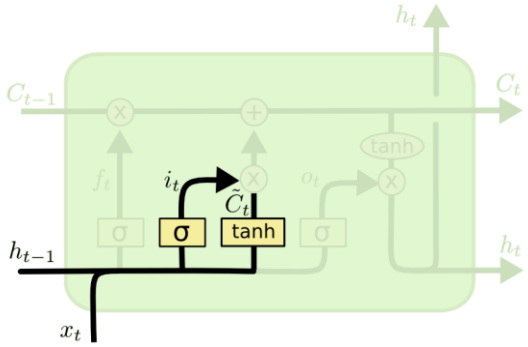


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 16: Forget gate layer. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Lo siguiente que tenemos que decidir es que nueva información vamos a guardar en el estado de la celda. Esto tiene 2 partes. Primero, una capa sigmoide que se llama “input gate layer” que decide que valores hay que actualizar. Posteriormente, una capa tanh crea un vector de nuevos valores candidatos que pueden ser añadidos al estado. Esto se puede ver en la Figura 16. En el siguiente paso, combinamos estas dos puertas para crear una actualización al estado. Esto se puede ver en la Figura 17.

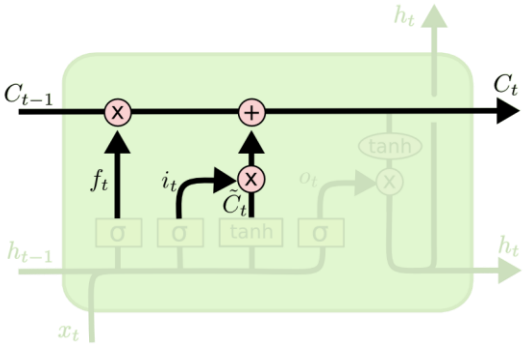
En nuestro ejemplo, queremos recordar el nuevo género del sujeto al estado de la celda, reemplazando el anterior [15].



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 17: Input gate layer. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Ahora toca actualizar el estado de la celda. Los pasos previos nos han dicho lo que tenemos que hacer, ahora simplemente hay que hacerlo [15].

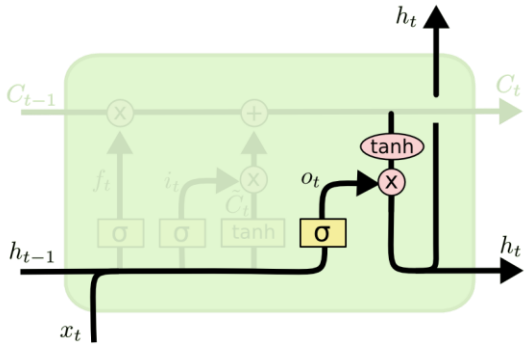


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 18: Actualizando los pesos de una celda LSTM. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Por último, tenemos que decidir qué vamos a ofrecer como salida. La salida estará basada en el estado de la celda, pero será una versión filtrada. En primer lugar, usaremos una capa sigmoide que decide que partes del estado de la celda vamos a sacar por la salida. Después, pondremos el estado de la celda a través de una capa tanh (Para hacer que los valores se encuentren entre -1 y 1) y lo multiplicaremos por la salida de la puerta sigmoide, de manera que salga de la celda solo las partes que hayamos decidido.

En nuestro ejemplo, como hemos visto un sujeto puede ser que queramos sacar información relevante para un verbo, en caso de que sea lo que viene después. Por ejemplo, puede que la salida diga si el sujeto es singular o plural para que podamos conjugar de manera correcta el verbo [15].



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 19: Creando la salida de una celda LSTM. Fuente: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Este sería el funcionamiento esencial de una capa LSTM. Para regular los pesos con capas posteriores, habría que usar el algoritmo de backpropagation que hemos usado anteriormente con las funciones de activación correspondientes.

3.3.10 Conclusión

Las redes neuronales es un campo muy extenso dentro del Deep Learning. Hemos explicado la parte correspondiente a este trabajo de fin de grado, pero se podría profundizar mucho más, aunque por concisión no lo vamos a hacer.

En la siguiente sección vamos a hablar de series temporales, las cuales relacionaremos con las redes LSTM en la exposición de resultados del trabajo de fin de grado.

3.4 Time series

3.4.1 Introducción

En nuestro trabajo de fin de grado, vamos a analizar series de números o datos que se presentan cada cierto tiempo en un periodo de tiempo determinado. Es por esto, que en los preliminares vemos necesario explicar al menos por encima las series temporales, para que en la posterior exposición de los resultados se entiendan los métodos realizados.

Una serie temporal es una sucesión de observaciones de una variable tomadas en varios instantes de tiempo. Estas observaciones se toman porque nos interesa estudiar los cambios en esta variable respecto al tiempo y puede que también predecir sus valores futuros [16].

Tenemos ejemplos de series temporales en muchos campos. En economía con el producto interior bruto o la tasa de inflación, en demografía con los nacimientos anuales o la tasa de dependencia, en meteorología con las temperaturas máximas o las precipitaciones diarias y en medio ambiente como en las emisiones anuales de CO2. Para ilustrarlo, se puede ver en la Figura 20, una serie temporal en azul de CO2 cada hora. En azul están los datos reales y en rojo estaría la gráfica corregida.

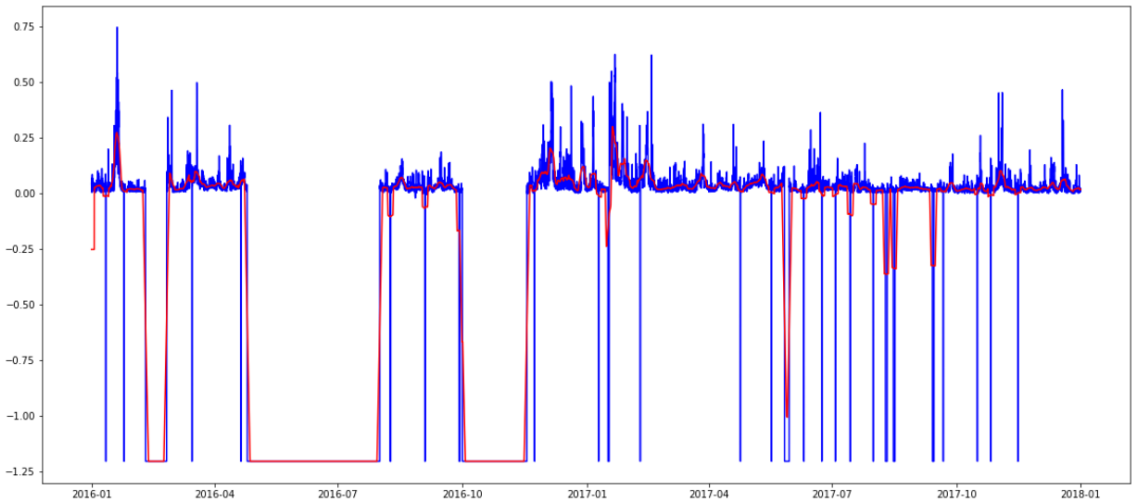


Figura 20: Serie temporal con outliers

3.4.2 Clasificación de las series temporales

No somos capaces de tratar cualquier tipo de serie temporal. Hay series temporales, como puede ser los números de la bonoloto, que no siguen ningún tipo de patrón y cuya media y variabilidad cambia mucho a lo largo del tiempo. Sin embargo, también hay series temporales, como la media de la temperatura diaria, que siguen un patrón y una media similar en cada zona, pero cuya media cambia en cada temporada [17].

Es por eso que es necesario clasificar las series temporales en función de si son tratables o de si, por el contrario, no lo son. La clasificación es la siguiente:

- Una serie es **estacionaria** si la media y la variabilidad se mantienen constantes a lo largo del tiempo. Por ejemplo, la temperatura media diaria durante el verano.
- Una serie es **no estacionaria** si la media y/o la variabilidad cambian a lo largo del tiempo.
    - Las series no estacionarias pueden mostrar **cambios de varianza.**
    - Además, las series no estacionarias pueden mostrar una **tendencia**, es decir que la media crece o decrece a lo largo del tiempo.
    - Por último, pueden presentar **efectos estacionales**, es decir que el comportamiento de la serie es parecido en ciertos tiempos periódicos en el tiempo. Estos efectos estacionales están muy presentes en nuestro ejemplo de temperatura diaria durante el año.

Para definirlo de manera correcta, una serie es estacionaria si presenta las siguientes características:

$$E[X_t] = \mu \; para \; todo \; t$$

$$Var(X_t) = \sigma^2 \; para \; todo \; t$$

$$Cov(X_t, X_{t+k}) = \gamma_k \; para \; todo \; t \; y \; k$$

Un posible ejemplo podría ser el **ruido blanco,** cuando la media y la covarianza son siempre cero [18].

Las series estacionarias presentan una serie de características que son muy beneficiosas para el estudiante de estas. En primer lugar, en las series temporales estacionarias se pueden obtener predicciones muy fácilmente. Además, como la media no varía, podemos estimarla con todos los datos, y utilizar este valor para predecir un valor posterior. Otra característica interesante es que también se pueden obtener intervalos de predicción o confianza para las predicciones, asumiendo que siguen, por ejemplo, una distribución normal o una distribución conocida.

Una serie no estacionaria no presenta ninguna de estas propiedades, lo cual hace que sea muy tedioso estudiarlas. Un ejemplo de serie no estacionaria podría ser el número mensual de pasajeros de avión.

3.4.3 Desglose de una serie temporal

Para entender las series temporales, a veces es bueno descomponerlas en n series temporales diferentes, para poder ver así características particulares de la serie temporal en particular.

Un ejemplo de esto sería la descomposición en tendencia, estacionalidad y residuo. Se establece que el valor observado es la suma de la tendencia, la estacionalidad y el residuo en un momento concreto.

La **tendencia** es el comportamiento o movimiento suave de la serie a largo plazo. En algunos casos, se puede suponer una relación determinista entre la tendencia y el tiempo, por ejemplo, una tendencia lineal, que se pude estimar mediante los mínimos cuadrados.

$$T_t = a + bt \ (27)$$

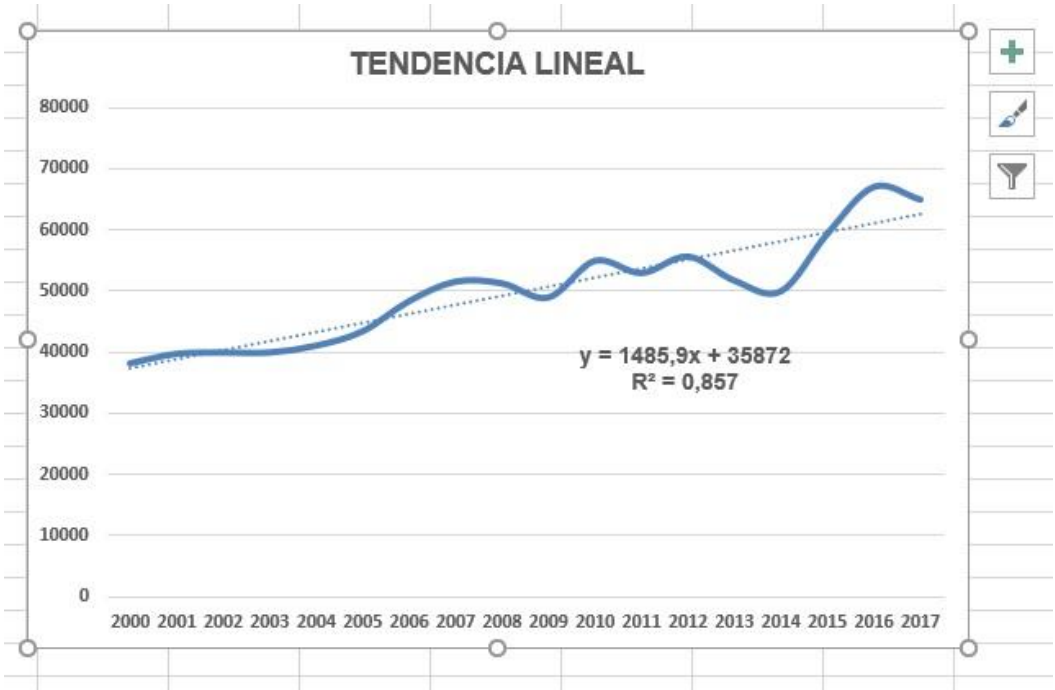


Figura 21: Tendencia lineal. Fuente: https://estamatica.net/analisis-de-tendencia-lineal/

Sin embargo, este tipo de tendencia es muy simple. Hay muchas veces que la tendencia no sigue una recta, sino que evoluciona a lo largo del tiempo.

Para estos casos, un método que se suele aplicar para estimar T_t es suponer que evoluciona lentamente en el tiempo, y que se puede aproximar con una función simple para intervalos cortos de tiempo. Un ejemplo de lo que se suele aplicar es la media móvil de orden 3.

$$m_t = \frac{x_{t-1} + x_t + x_{t+1}}{3} \quad (28)$$

Esta fórmula se puede generalizar. De esta manera si k es impar:

$$m_t = \frac{1}{k}\sum_{j=-m}^{m} x_{t+j}, m = (k-1)/2 \quad (29)$$

Por otra parte, si k es par:

$$m_t = \frac{1}{2k}x_{t-k/2} + \frac{1}{k}\left(x_{t-k/2+1} + \cdots + x_t + \cdots + x_{t+k/2-1}\right) + \frac{1}{2k}x_{t+k/2} \quad (30)$$

Ecuación 14: Media móvil si k es par

Con esto suponemos que la tendencia satisface:

$$T_t = m_t - \frac{I_{t-1} + I_t + I_{t+1}}{3} \quad (31)$$

Siendo I el residuo de la serie temporal. Como la media del componente irregular es cero, podemos suponer que la media de los tres valores es pequeña, así que de esta manera m_t recoge fundamentalmente la tendencia de la serie en el instante t.

Además de la tendencia, como ya hemos dicho previamente existen la estacionalidad y el residuo. La **estacionalidad** son movimientos de oscilación dentro del año o de un periodo de tiempo n. Por ejemplo, midiendo las emisiones de CO2 de una ciudad habrá más concentración los días laborables que los no laborables porque la gente va a trabajar. Esto es a lo que se denomina estacionalidad.

Por otra parte, el **residuo** son las variaciones aleatorias alrededor de los componentes anteriores. Esto es lo que podríamos considerar outliers o ruido y lo que habría que quitar en una posible eliminación de outliers, mediante un método de descomposición. Por ejemplo, en nuestro caso [19].

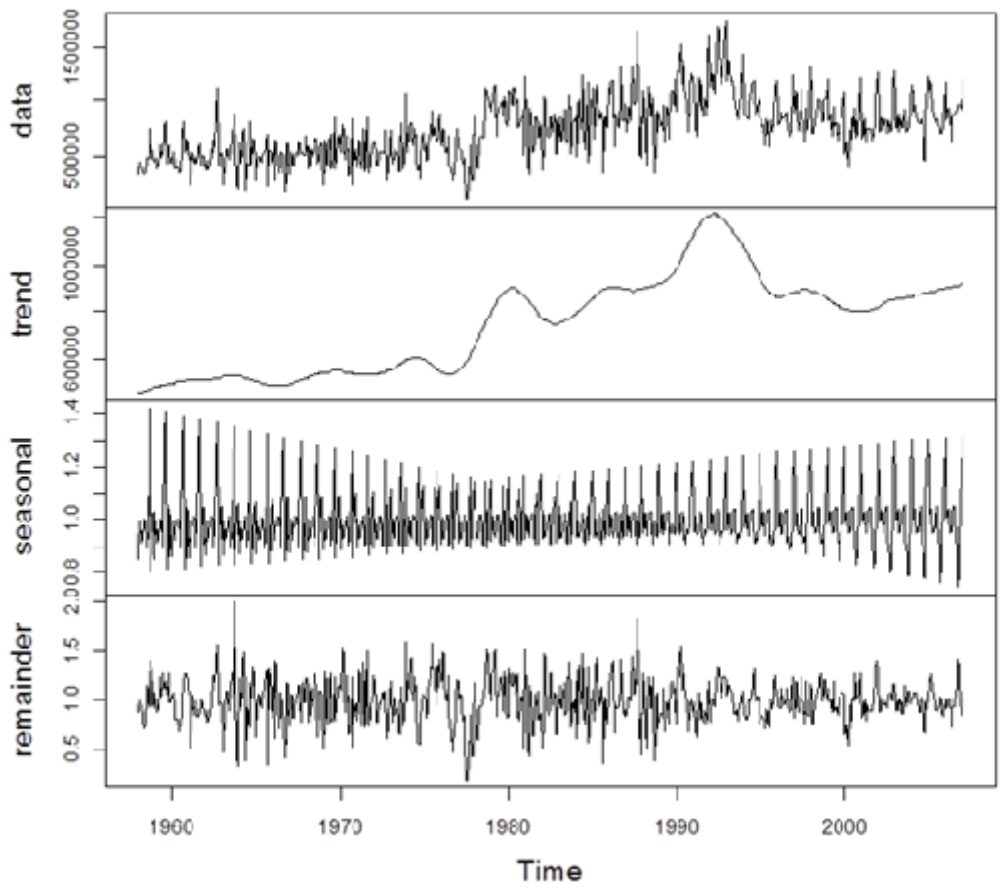


Figura 22: Descomposición en tendencia, estacionalidad y residuo de una serie temporal. Fuente: https://www.researchgate.net/publication/299547311/figure/fig1/AS:350516048613380@1460580946894/Figura-1-Descomposicion-de-la-serie-estacional-en-tendencia-ciclico-y-residual.png

3.4.4 Otras descomposiciones de series temporales

La descomposición que hemos visto en el apartado previo es muy simple. Se han desarrollado otros modelos para descomponer las series temporales de maneras más efectivas. Para explicarlas primero tenemos que desarrollar la diferencia entre los modelos aditivos y multiplicativos.

En los modelos **aditivos** las magnitudes en las que se descompone la serie temporal se suman. De este tipo es la descomposición estudiada en el apartado previo. Es apropiado si la magnitud de las fluctuaciones estacionales no varía con el nivel.

Por otro parte, si la estacionalidad es proporcional al nivel de la serie, **el modelo multiplicativo** es más apropiado. De hecho, este modelo se usa muy habitualmente en series económicas como en la bolsa.

Un método de descomposición muy usado es el método **STL** "Seasonal and Trend decomposition using Loess". Es muy versátil y robusto además de manejar muy bien la estacionalidad. La maneja muy bien ya que el propio modelo entiende que el componente estacional puede cambiar con el tiempo, así que esta ratio se puede controlar por el usuario. Además de este parámetro, el usuario puede ajustar también el suavizado de la tendencia-ciclo. Lo que más nos interesa de este modelo a nosotros es que es **muy robusto a los outliers.**

Para entender cómo funciona esta descomposición basada en Loess, primero como es obvio hay que entender qué es el Loess. Supongamos que x_i y y_i para $i = 1\ a\ n$ son medidas de una variable independiente y dependiente respectivamente. La curva de regresión de Loess, $\hat{g}(x)$, es un suavizado de y dado x que puede ser calculado para cualquier valor dado de x sobre la escala de la variable independiente. Esto significa que el loess puede ser definido donde sea y no solo en x_i. Esto nos viene muy bien ya que esta característica nos permite tratar falta de valores y descomponer el componente estacional de una manera homogénea. De hecho, el loess puede ser usado para suavizar y como una función de cualquier número de variables independientes, es decir es una función multivariable, pero en el caso de la descomposición STL, solo se necesita una variable independiente.

$\hat{g}(x)$ se calcula de la siguiente manera. En primer lugar, se elige un numero q entero positivo. Los q valores de x_i que están más cerca a x son seleccionado y a cada uno se le da un peso de vecindad basado en su distancia a x. Sea $\lambda_q(x)$ la distancia del q-ésimo valor más lejano de x. Sea W la función de peso tricubica:

$$W(u) = \begin{cases} (1-u^3)^3 & para\ 0 \le u \le 1 \\ 0 & para\ u \ge 1 \end{cases} \quad (32)$$

Entonces el peso de vecindad para cualquier x_i es:

$$v_i(x) = W\left(\frac{|x_i - x|}{\lambda_q(x)}\right) \quad (33)$$

Como se puede observar de la formula, el peso es mayor cuando más cercano sea el vecino del punto origen, haciéndose 0 la distancia en el q-ésimo vecino.

Explicado que es la función Loess vamos a pasar a explicar el funcionamiento interno del algoritmo STL. Este consiste en dos bucles: Uno interno y otro externo. En cada uno de las pasadas del bucle interno, el componente de tendencia y de estacionalidad son actualizados, de manera que se ajusten poco a poco a la solución que queremos obtener. Cada una de las pasadas del bucle externo consiste en la actualización de los pesos del bucle interno, junto con un cálculo de pesos robustos. Estos pesos sirven para que la siguiente vez que se ejecuta el bucle interno se reduzca la influencia del comportamiento anómalo de la tendencia o de la estacionalidad.

A partir de ahora vamos a explicar en profundidad el funcionamiento del bucle interior y posteriormente del bucle exterior, para una mejor comprensión de este tipo de descomposición.

Cada paso del bucle interno consiste en un suavizado del componente estacional que actualiza el componente estacional, seguido de un suavizado de la tendencia que actualiza el componente de la tendencia. Supongamos que $S_v^{(k)}$ y $T_v^{(k)}$ para $v = 1 \dots N$ son los componentes estacionales y de tendencia al final de k pasadas. Estos dos componentes son definidos en todos los tiempos de $v = 1 \dots N$, incluso en los momentos en los que falta Y_V. Para calcular $S_v^{(k+1)}$ y $T_v^{(k+1)}$, realizamos los siguientes pasos:

1. **Creamos una serie sin tendencia**. Para ello se calcula $Y_V - T_v^{(k)}$. Si Y_v falta en una posición determinada, entonces la serie sin tendencia falta también en esa posición v.

2. **Suavizado de las subseries**. Mediante el Loess con $q = n_{(s)}$ y $d = 1$, se suaviza cada una de las subseries de la serie sin tendencia. Los valores suavizados se calculan en todos los puntos de la serie temporal, incluso en aquellos en los que falta el valor. También se calculan una posición antes del inicio de la serie y una posición junto al final de la serie. Como salida se obtiene una serie suavizada estacional que denominaremos $C_v^{(k+1)}$.
3. **Filtro de paso bajo a los valores suavizados de las subseries**. Se aplica un filtro de paso bajo a cada una de las subseries suavizadas. Este filtro consiste en una serie de medias móviles con diferentes longitudes seguido de un loess de suavizado. En este paso hay que darse cuenta que al hacer la media móvil se pierden los valores a los extremos. Es por esto, que en el paso 2 hemos hablado de realizar la serie suavizada 1 valor más allá de los extremos, para anticiparnos a esto. La serie que sale de este filtro la denominaremos $L_v^{(k+1)}$.
4. **Quitar la tendencia de la serie suavizada.** El componente estacional del bucle $k + 1$ es $S_v^{(k+1)} = C_v^{(k+1)} - L_v^{(k+1)}$.
5. **Quitar la estacionalidad.** Se calcula una serie sin estacionalidad mediante la fórmula $Y_v - S_v^{(k+1)}$. Si falta Y_v en un determinado momento, entonces la serie sin estacionalidad también falta en ese momento.
6. **Suavizado de la tendencia.** La serie sin estacionalidad es suavizada con otro loess con $q = n_{(l)}$ y $d = 1$. Los valores suavizados se calculan en todas las posiciones temporales $v = 1 \dots N$, incluso en aquellas en las que falta el valor Y_v. El producto de este paso es $T_v^{(k+1)}$, que es uno de los productos que queríamos desde el principio.

Mediante estos pasos, el bucle interior para calcular los valores de descomposición estaría calculado. Sin embargo, como ya hemos dicho anteriormente, el bucle interior está envuelto en un bucle exterior que calcula una serie de pesos de robustez. Para explicar con detalle esta robustez, primero hay que decir que el bucle exterior también calcula el componente residual:

$$R_v = Y_v - T_v - S_v \ (34)$$

Los pesos de robustez nos dicen como de extremo es el residuo. Un outlier en los datos tendrá un $|R_v|$ muy elevado y su peso tiene que ser muy bajo. Sea

$$h = 6 * mediana(|R_v|)$$

Entonces los pesos de robustez en un determinado tiempo v son

$$\rho_v = B(|R_v|/h\ ) \ (35)$$

Donde B es una función bicuadrática de pesos:

$$W(u) = \begin{cases} (1-u^2)^2 & para\ 0 \le u \le 1 \\ 0 & para\ u \ge 1 \end{cases} (36)$$

Ahora el bucle interior se repite, pero cada uno de los valores en el tiempo v son multiplicados por un peso de robustez ρ_v.

Este es el mecanismo para calcular la descomposición basada en Loess [20]. Este par de bucles se repiten hasta que converjan los valores de tendencia y estacionalidad, de tal forma que varíen de manera mínima de una iteración a otra. Quedaría por explicar los valores de los que hemos

hablado para hacer el Loess y las medias móviles, pero eso es profundizar mucho en un tema que no es el principal del trabajo de fin de grado, así que no vamos a entrar.

En el siguiente apartado, hablaremos del filtrado de series temporales para eliminar el ruido, que puede ser nuestro caso de los outliers.

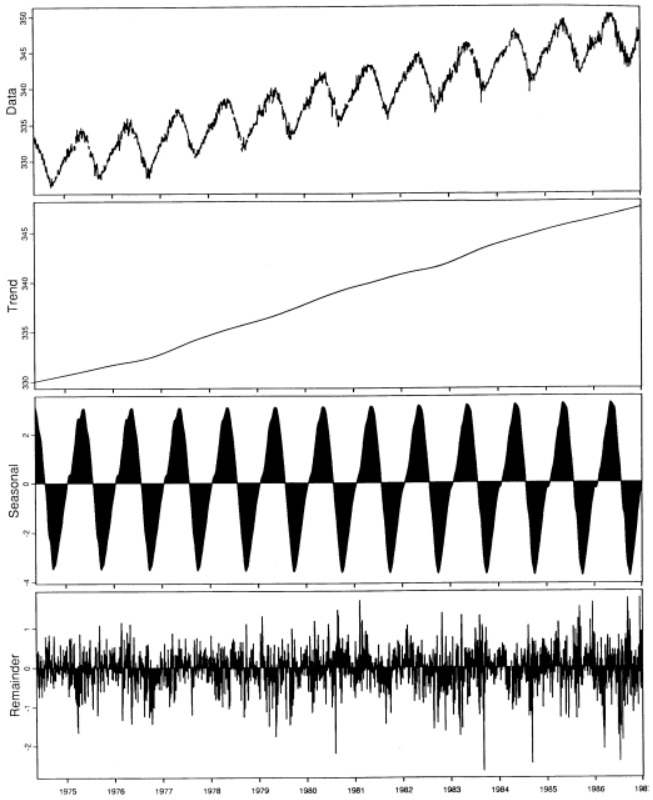


Figura 23: Descomposición STL de muestras de dióxido de carbono. Fuente: Journal of Official Statistics Vol. 6 No. 1. 1990. Pp. 3-73

3.4.5 Filtrado de las series temporales

Aquí hablare de los métodos que se pueden usar en series temporales para eliminar el ruido. Uno de los métodos más utilizados y del que ya hemos hablado es utilizar las medias móviles [17].

Sin embargo, este método presenta graves limitaciones:

- Las medias móviles son costosas de calcular. Hay que hacer una media móvil por cada valor así que tiene un coste computacional muy grande.
- Problemas en los extremos de las series de los datos. Dada la anchura de la ventana, no se pueden extender hasta el final de la serie, que suele ser lo más interesante.
- No se pueden definir fuera de la serie temporal, por lo que no se pueden sacar predicciones.

Para solucionar estos problemas, se propone el suavizado exponencial. Este proporciona un filtrado fácil de calcular y se divide en:

- **Suavizado exponencial simple:** Para series sin tendencia ni estacionalidad.
- **Suavizado exponencial doble:** Para series con tendencia, pero no estacionalidad.

- **Suavizado exponencial triple:** Para series con tendencia y estacionalidad.

El **suavizado exponencial simple** actualiza el resultado del anterior valor con el último dato de la serie original (Combinando la información ya disponible con la aportada por el nuevo dato mediante un parámetro, 0 < α < 1). Se calcula mediante la fórmula:

$$S_i = \alpha x_i + (1-\alpha)s_{i-1} \ (37)$$

¿Pero por qué se llama suavizado exponencial si no lleva exponente? Si expandimos la recurrencia obtenemos

$$S_i = \alpha \sum_{j=0}^{i} (1-\alpha)^j x_{i-j} \ (38)$$

Todas las observaciones contribuyen al valor suavizado, pero su contribución se suprime por el exponente creciente del parámetro alfa.

El problema de este tipo de suavizado es que, si extendemos el suavizado más allá de los datos disponibles, la predicción es extremadamente simple, lo que hace que no sea un modelo robusto para la predicción de nuevos valores.

Es por esto, que pasamos al segundo tipo de suavizado, al **suavizado exponencial doble.** Este suavizado es un suavizado más complejo ya que tiene en cuenta la tendencia de la serie. Se calcula mediante la siguiente ecuación:

$$s_i = \alpha x_i + (1-\alpha)(s_{i-1} + t_{i-1}) \ (39)$$

$$t_i = \beta(s_i - s_{i-1}) + (1-\beta)t_{i-1} (40)$$

Este suavizado retiene información acerca de la tendencia mediante estas fórmulas: La señal suavizada se encuentra en s_i y la tendencia suavizada se encuentre en t_i.

El parámetro beta se utiliza para realizar un suavizado exponencial sobre la tendencia.

En este caso, su uso en predicción de futuros valores es mejor. La fórmula que seguiría sería la siguiente:

$$x_{i+h} = s_i + ht_i \ (41)$$

Sin embargo, este modelo pese a ser mejor que el suavizado simple, se queda corto para la mayoría de los casos. Es por eso que nace el **suavizado exponencial triple** al que también se le conoce como el método de Holt-Winters. En este método, una tercera cantidad se utiliza para describir la estacionalidad, que puede ser aditiva o multiplicativa según nos interese.

En este trabajo de fin de grado solo se usa la estacionalidad aditiva por lo que solo pondré las ecuaciones que modelan este tipo de método de Holt-Winters. Son las siguientes:

$$s_i = \alpha(x_i - p_{i-k}) + (1-\alpha)(s_{i-1} + t_{i-1}) \ (42)$$

$$t_i = \beta(s_i - s_{i-1}) + (1-\beta)t_{i-1} \ (43)$$

$$p_i = \gamma(x_i - s_i) + (1-\gamma)p_{i-k} \ (44)$$

Así pues, la predicción de los valores siguientes se realiza mediante la siguiente ecuación:

$$x_{i+h} = s_i + ht_i + p_{i-k+h} \ (45)$$

En este caso, la predicción es mucho más robusta teniendo muchos más parámetros en cuenta. A pesar de esto, hay muchas series temporales que se le resisten debido a su complejidad. Es por esto por lo que vemos necesario explorar otros métodos de predicción de futuros valores [21].

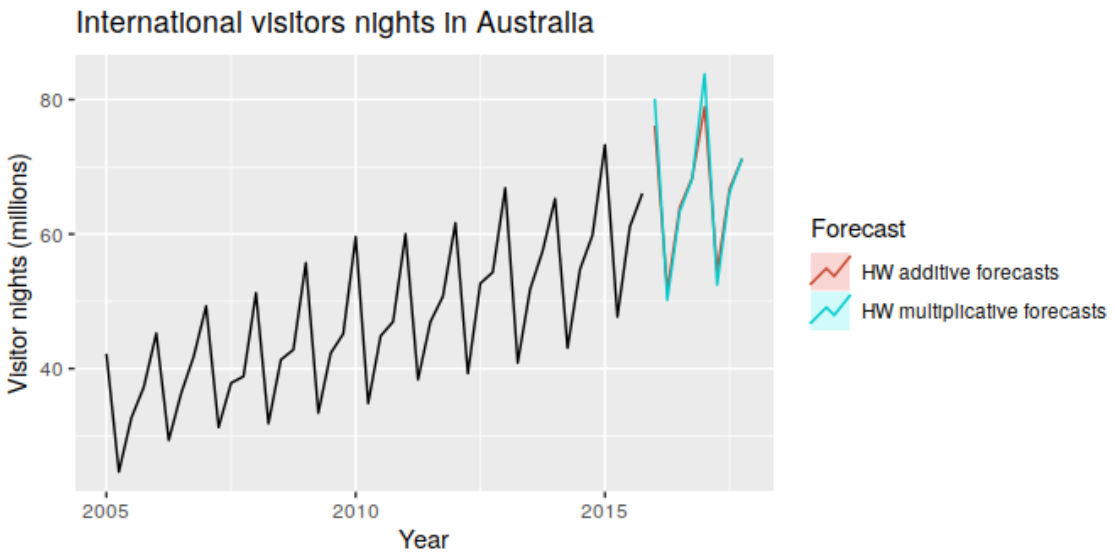


Figura 24: Holt winters aplicado a los visitantes nocturnos en Australia.

3.4.6 Modelación ARIMA

Para definir qué cual es la modelación, primero tenemos que definir que es un proceso estocástico. Un **proceso estocástico** es una secuencia de variables aleatorias, ordenadas y equidistantes cronológicamente, referidas a una o a varias características de una unidad observable en diferentes momentos.

Según esta definición, se puede ver que todos los procesos estocásticos son series temporales, aunque el reciproco no es cierto. Por tanto, todo lo que hemos hablado previamente acerca de las series temporales también se puede aplicar a los procesos estocásticos.

La estructura que existe en la dependencia entre los datos en uno de estos procesos, puede ajustarse a seis modelos que pueden describir una gran cantidad de fenómenos reales. La principal característica de dichos modelos es que no involucran a las variables independientes en su construcción, es decir, a la variable del tiempo t, como se hace en el análisis clásico. En este caso, se usan las observaciones pasadas para inferir la observación presente.

Los modelos son los siguientes:

Proceso de ruido blanco – (a_t): Es un proceso formado por una secuencia de variables aleatorias mutuamente independientes e idénticamente distribuidas. Este modelo puede ajustarse a proceso aleatorios de la naturaleza, como a las interferencias en una onda de radio.

Modelo no estacionario de corrido aleatorio – (I): Corresponden a aquellas situaciones en las que hay impulsos aleatorios residuales que tienden a sumarse o a integrarse en el tiempo. Este tipo de modelos podrían ser las series temporales con outliers que tratamos en este trabajo de fin de grado.

Modelo autorregresivo de orden p – AR(p): Se trata de un modelo en el que se puede predecir una observación a partir de la observación anterior (modelo autorregresivo de primer orden) o a partir de p observaciones que le preceden (modelo autorregresivo de orden p).

Un modelo autorregresivo tiene la siguiente forma:

$$Y_t = \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \varphi Y_{t-3} + \cdots + \varphi Y_{t-3} + \cdots + \varphi_p Y_{t-p} + a_t \ (47)$$

En donde Y_t es la variable dependiente y el resto de Y son las variables independientes.

El modelo se denomina autorregresivo porque se asemeja a la ecuación de regresión:

$$Y = a + b_1 X_1 + b_2 X_2 + b_3 X_3 + \cdots + b_p X_p + e \ (48)$$

Las variables independientes son simplemente valores de la variable dependiente en valores pasados.

Modelo de Medias Móviles de orden q – MA(q): En este modelo, se presupone que una determinada observación está condicionada por la aleatoriedad de momentos pasados. De esta manera, la observación actual es la suma del valor actual y de la aleatoriedad de momentos pasados con un determinado peso.

El modelo de medias móviles tiene la siguiente forma:

$$Y_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \ (49)$$

Donde a_t es el error en el periodo t y $a_{t-1}, a_{t-2}, a_{t-3}, \ldots, a_{t-q}$ son los valores anteriores del error. La ecuación es similar a la del modelo autorregresivo, solo que la variable dependiente depende más del error de las observaciones pasadas que de estos valores en sí mismos.

Modelo autorregresivo de medias móviles de orden p,q – ARMA(p,q): Este modelo es la combinación de las estructuras que hemos definido previamente: Un modelo autorregresivo junto con un modelo de medias móviles. Así se puede decir que una observación en concreto está afectada tanto por observaciones del pasado como por la aleatoriedad o los errores pasados.

La forma general de un modelo autorregresivo de medias móviles es

$$Y_t = \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \varphi Y_{t-3} + \cdots + \varphi Y_{t-3} + \cdots + \varphi_p Y_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \ (50)$$

Se puede ver que esta ecuación es simplemente una combinación de las ecuaciones del modelo AR y MA.

Un modelo como este se puede ajustar a cualquier patrón de datos. A pesar de eso, los valores de p y de q hay que especificarlos, para saber hasta qué valor hay que tener en cuenta para obtener el valor actual.

Una limitación de este modelo es que solo puede lidiar con series estacionarias. Esto significa que, si la media y la desviación cambia a lo largo del tiempo, este modelo no es capaz de ajustarse.

Modelo autorregresivo integrado de medias móviles de orden p,d,q – ARIMA(p,d,q): Este es un modelo ARMA con una peculiaridad: Incluye un proceso de restablecimiento (que se denomina integración) de inestabilidad original presente en una serie de tiempo. Esto significa, que a diferencia del modelo ARMA, este modelo es capaz de tratar también con series no estacionarias.

La forma general de un modelo ARIMA es la siguiente:

$$Y'_t = \varphi_1 Y'_{t-1} + \varphi_2 Y'_{t-2} + \varphi Y'_{t-3} + \cdots + \varphi Y'_{t-3} + \cdots + \varphi_p Y'_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \quad (51)$$

Donde Y'_t es la serie inducida a la estabilidad. Esta serie inducida a la estabilidad es una serie en completamente estacionaria. Para ello, se puede eliminar la tendencia y el residuo mediante un desglose de series temporales como hemos visto arriba.[22]

Mediante este modelo ARIMA, somos capaces de descomponer cualquier serie, aunque no sea estacionaria.

4.Análisis exploratorio de los datos

4.1 Análisis preliminar

En el capítulo anterior, hemos explicado los fundamentos previos para entender este trabajo de fin de grado. Con este capítulo, comenzamos con la exposición de los resultados del mismo. Como en cualquier análisis de datos, lo primero que hay que llevar a cabo es un análisis exploratorio de los datos para poder ver cuál es el problema al que nos enfrentamos.

Lo primero hay que entender es que nos enfrentamos a una gran cantidad de datos. Lo cual hace muy complicados tratarlos simultáneamente. Es por eso que, en primer lugar, se crea un archivo resumen con todos los datos de relevancia para un estudio global preliminar, mediante todos los archivos que tenemos. Este archivo global lo vamos a realizar sobre los valores de CDR y sobre los valores de UTD, aunque en este primer estudio nos centraremos sobre los valores de CDR, ya que se supone que son los que no tienen errores y queremos ver la distribución de los valores válidos.

En este archivo se encuentran todas las series temporales recopiladas, para poder realizar posteriormente estudios de la media o realizar boxplots y ver entre que valores oscilan estas series en formato general.

Para ello, usamos la función describe() de la librería pandas sobre los datos de CDR como se puede observar en la Figura 25.

	NO2	O3	PM10	PM25	sp	ssr	t2m	tcc	tp
count	1.694063e+07	1.620774e+07	1.188128e+07	9.838069e+06	1.828584e+07	1.828584e+07	1.828584e+07	1.828584e+07	1.828584e+07
mean	-3.211064e+02	-4.352656e+02	-1.253670e+02	-8.079691e+02	9.980958e+04	1.513673e+06	2.841483e+02	7.896663e-01	7.831993e-03
std	2.086257e+03	2.580101e+03	2.179415e+03	3.348408e+03	2.502028e+03	3.126137e+05	7.681642e+00	2.169061e-01	2.982403e-03
min	-9.999000e+03	-9.999000e+03	-9.999000e+03	-9.999000e+03	8.876457e+04	9.471855e+05	2.655699e+02	4.999924e-01	4.734921e-03
25%	5.000000e+00	1.800000e+01	8.718550e+00	3.021991e+00	9.890560e+04	1.150446e+06	2.777210e+02	4.999924e-01	5.714091e-03
50%	1.204588e+01	4.625000e+01	1.600000e+01	7.850000e+00	1.004081e+05	1.600008e+06	2.835204e+02	8.946790e-01	6.904020e-03
75%	2.600000e+01	7.000000e+01	2.800000e+01	1.500000e+01	1.014341e+05	1.765093e+06	2.914423e+02	1.000000e+00	9.931413e-03
max	9.999900e+04	9.999900e+04	9.999900e+04	9.999900e+04	1.051303e+05	3.413114e+06	3.132890e+02	1.000000e+00	1.580643e-02

Figura 25: Tabla descriptiva de los datos

De la Figura 25, se puede observar una serie de cosas:

- El valor mínimo es muy negativo en las series temporales de NO2, O3, PM10 y PM25. Esto nos deja ver que ese es probablemente el valor que se le da a los momentos de la serie temporal sin ningún valor.
- La media en las series temporales de NO2, O3, PM10 y PM25 no es válida, ya que sale negativo por los valores nulos y al estar hablando de concentraciones no pueden salir valores negativos.
- Lo que pasa con el valor mínimo también pasa con el valor máximo, es un valor tan elevado que no parece real.
- Sin embargo, los valores de cuartiles salen mucho más razonables por lo que podemos ver que los datos están bien excepto en los extremos.
- El resto de series temporales (sp,ssr,t2m,tcc,tp,…), salen demasiado bien. Es decir, no tienen valores negativos ni valores máximos muy extraños. Esto nos hace pensar que

realmente estos valores hayan sido sacados de otra fuente. De hecho, en reuniones con la gente que se encarga de este proyecto, les preguntamos acerca de este hecho y confirmaron nuestras sospechas.

Por otra parte, algo que queríamos ver si era cierto era si los datos de diferentes series tenían una correlación entre ellos. Esto significa que quizás si una serie temporal aumentaba sus valores, otra los disminuía. O quizás iban a la par y si una serie temporal aumentaba otra también lo hacía. Por ello, computamos tres tipos de correlaciones de los datos. Para no alargar mucho este trabajo de fin de grado, no hemos explicado en detalle estas correlaciones. Además, vemos claramente que una de las tres correlaciones da información mucho más visual, más significativa y con más sentido que el resto, así que solo vamos a mostrar esa. Esta correlación es la de Spearman.

	NO2	O3	PM10	PM25	sp	ssr	t2m	tcc	tp
NO2	1	-0,3214	0,3374	0,4551	0,0568	-0,2007	-0,1862	0,0185	-0,1015
O3	-0,3214	1	-0,2665	-0,007	-0,0023	0,3448	0,2648	-0,083	0,0388
PM10	0,3374	-0,2665	1	0,5034	-0,0236	-0,0743	-0,1134	0,0385	-0,0469
PM25	0,4551	-0,007	0,5034	1	0,1339	-0,1224	-0,1291	-0,0102	-0,0929
sp	0,0568	-0,0023	-0,0236	0,1339	1	0,0528	0,1124	-0,1164	0,058
ssr	-0,2007	0,3448	-0,0743	-0,1224	0,0528	1	0,759	-0,1586	0,2549
t2m	-0,1862	0,2648	-0,1134	-0,1291	0,1124	0,759	1	-0,1329	0,634
tcc	0,0185	-0,083	0,0385	-0,0102	-0,1164	-0,1586	-0,1329	1	-0,0602
tp	-0,1015	0,0388	-0,0469	-0,0929	0,058	0,2549	0,634	-0,0602	1,0000

Tabla 1: Correlaciones de Spearman

De la Tabla 1, se puede observar con facilidad que series temporales presentan correlaciones, tanto positivas como negativas. Por ejemplo, se puede observar que el PM10 y el PM25 tienen una correlación positiva muy fuerte, de más de 0,5, la mayor de toda la tabla entre contaminantes. Esto tiene sentido ya que al final son ambas partículas de polvo, de diferente tamaño, pero partículas de polvo.

Por otra parte, se puede ver fácilmente que el NO2 y el O3 tienen una correlación negativa bastante notable. Esto es que cuando el NO2 o el O3 aumenta el otro disminuye o viceversa.

Continuando con estas comparativas, se puede ver que el NO2 y los PMX tienen una correlación directa bastante considerable, de más de 0,3. También se puede ver que el O3 tiene una correlación directa también considerable con la radiación solar también de más de 0,3.

Esta última comparativa nos hace ver que las correlaciones tienen sentido, ya que el O3 se forma por radiación solar que rompe las moléculas de oxígeno y se reagrupan en moléculas de ozono.

Siguiendo con nuestro análisis de los datos, vamos a continuar con un boxplot de los contaminantes para ver sobre qué rangos se mueven. Para hacer este estudio, hemos eliminado los valores nulos de estas series temporales, los valores que hemos visto anteriormente que estaban sobre -999, y también eliminaremos los valores negativos, ya que, al estar hablando de concentraciones, si los valores salen negativos suponemos que están mal.

Después de esta purga de valores, al realizar el boxplot nos sale lo de la Figura 26.

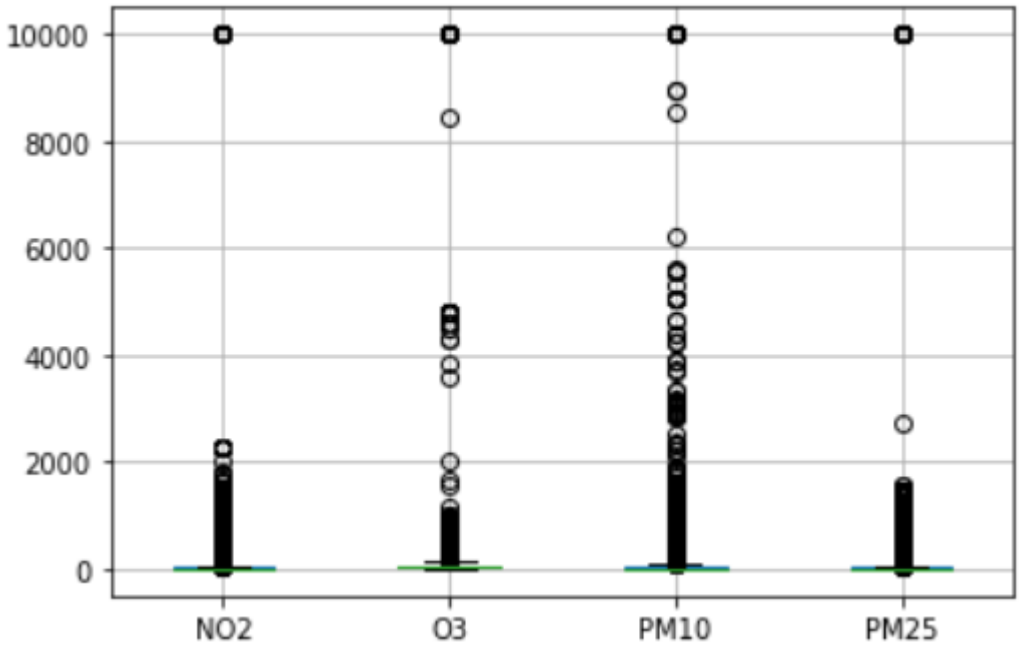


Figura 26: Boxplot

Se puede observar en la Figura 26, que los valores de los contaminantes oscilan normalmente entre 0 y 2000 unidades, aunque hay valores que sobrepasan estos valores. Estos podrían ser los denominados outliers.

También se puede observar que el PM10 tiene valores más altos de normal que el resto de contaminantes. Esto habrá que observarlo posteriormente con más atención, pero probablemente signifique que tengamos que realizar una normalización, para evitar que el PM10 pese más que el resto de contaminantes al realizar modelos.

Ahora hemos visto de manera global todos los datos, pero nos falta algo. A pesar de que hemos visto la distribución de los datos de manera general, no hemos visto aún ninguna serie temporal, aunque sea de lo que trate este trabajo de fin de grado.

Las series temporales tienen el problema de que analizarlas de manera conjunta es muy complicado. Esto es porque los tiempos pueden no coincidir o los lapsos de tiempo entre los que se toman mediciones pueden variar de una dataset a otro. Incluso aunque estas dos características encajen, no tiene demasiado sentido realizar un estudio global de series temporal ya que cuando en Londres haya contaminación por tráfico, puede que en Pamplona no haya ningún coche en la calle.

Sin embargo, sí que vemos necesario mostrar alguna serie temporal modelo para observar su comportamiento y para comparar los datos corregidos de series temporales (CDR) de los datos sin corregir.

Esta estación modelo que hemos elegido está en Alemania y su código de estación es *STA.DE_DEBB065.* Para poder observar mejor la serie de UTD y poder compararla mejor con su homóloga en CDR, vamos a realizar un proceso de normalización y de relleno de huecos por interpolación. Los huecos se provocan por momentos en los que no hay medición pero que debería haber por tiempos.

A continuación, mostramos la comparativa entre series temporales de contaminantes de UTD y de CDR.

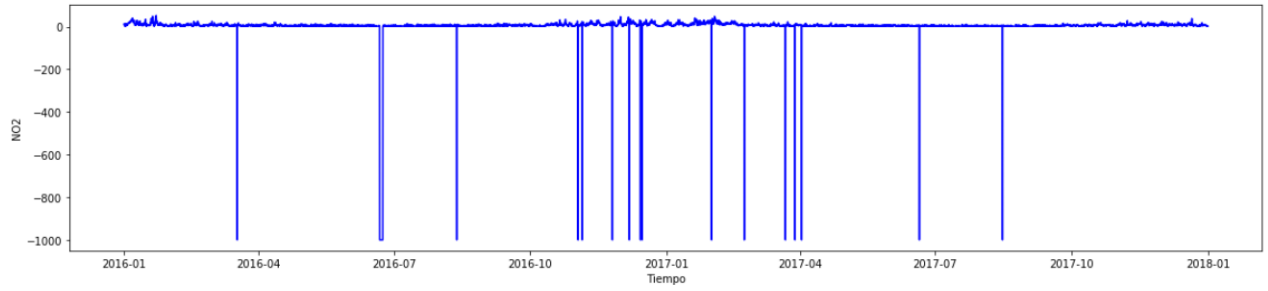


Figura 27:NO2 UTD STA.DE_DEBB065

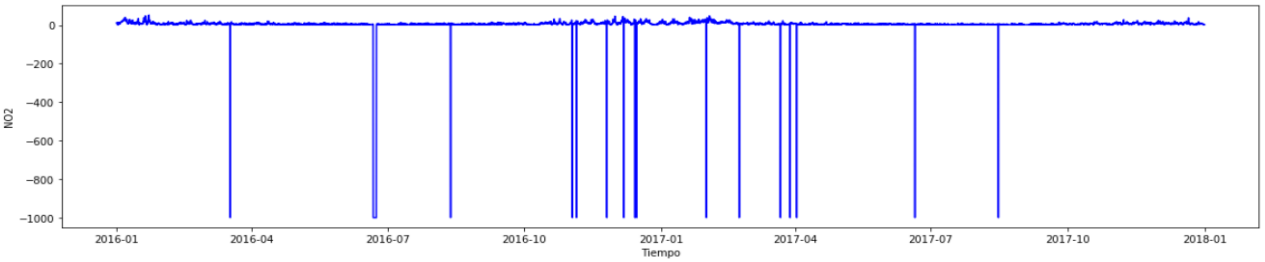


Figura 28:NO2 CDR STA.DE_DEBB065

En las Figuras 27 y 28 se puede observar que de UTD a CDR prácticamente no ha cambiado nada. Tiene los mismos picos hacia abajo una serie como la otra. Puede ser que, si ajustásemos la escala, quitando los valores negativos, se viese alguna diferencia más. Pese a esto, no he realizado tal ajuste porque veía más significativo ver toda la serie al completo, incluyendo los picos bruscos hacia abajo.

Asimismo, los cambios entre la serie corregida y no corregida de ozono también son inapreciables. Dado esto, no voy a adjuntar las imágenes correspondientes ya que la comparativa es similar a la del NO2. Esto probablemente sea debido a que los sensores de O3 y de NO2 sean los que mejor funcionen y, por tanto, los que menos correcciones necesitan.

En contraposición a estos dos contaminantes, nos encontramos con el PM25 y el PM10. Estos dos reciben una gran cantidad de actualizaciones y mejoras entre la serie UTD y la serie CDR. Dado que ambas están correlacionadas, por lo que las correcciones son similares, y que en esta parte del trabajo de fin de grado estamos haciendo un análisis visual y no un análisis exhaustivo, solo voy a mostrar las semejanzas y diferencias con la serie del PM25, en las Figuras 29 y 30.

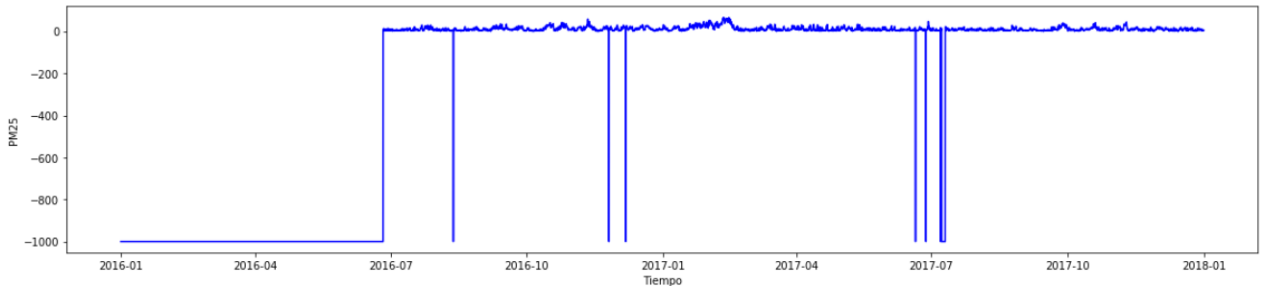


Figura 29:PM25 CDR STA.DE_DEBB065

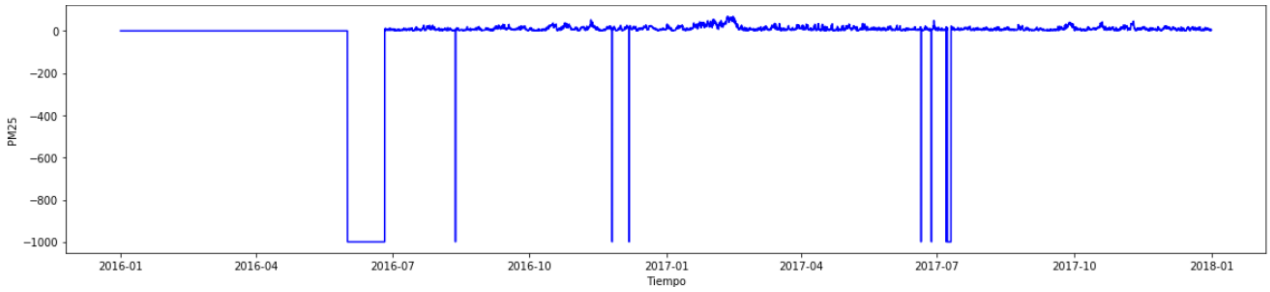


Figura 30:PM25 UTD STA.DE_DEBB065

Se puede ver que de la Figura 29, que se corresponde con los datos no validados, a la Figura 23, se es la que muestra los datos validados, hay diferencias. La más notable es que los primeros datos de la Figura 30 quedan anulados en la Figura 29. Esto se debe a que el sistema ha detectado que son datos con poca validez y los ha convertido al valor mágico. Además de esta corrección, probablemente en la serie con pocas variaciones (entre 0 y 1) haya también alguna corrección, aunque no sea tan notable.

Después de este análisis, vamos a continuar analizando con más profundidad las diferencias entre UTD y CDR, y viendo que países son los que más y los que menos corrigen sus series de contaminantes.

4.2 Análisis de diferencias entre UTD y CDR

Para analizar mejor las diferencias entre estos dos datasets, hemos creado un script que computa las diferencias entre ambas series y muestra una gráfica para que sea más fácil para nosotros observar esas diferencias. Con este script, hemos notado que hay series temporales que se corrigen más que otras, por ejemplo, el PM25 frente al NO2 o el O3, aunque hay excepciones como la siguiente que se muestra en la Figura 31.

Figura 31: O3 UTD vs CDR STA_ES1799A

En esta Figura se puede observar en azul la serie UTD, y en naranja las correcciones que hacen en la serie CDR. Como se puede ver, la cantidad de correcciones que se generan son inmensas. Todas estas correcciones son las que podríamos denominar outliers, además de otras que pueden no estar detectadas.

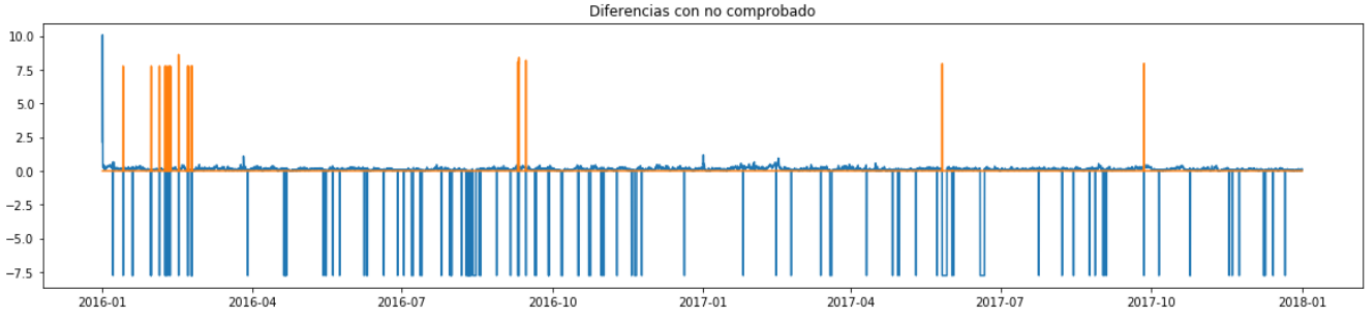


Figura 32: PM10 UTD vs CDR STA.DE_DEHH033

En la Figura 32 podemos ver otro ejemplo de correcciones. En este caso, en la serie temporal de PM10 se pueden ver una gran cantidad de errores que han sido corregidos. A su vez, en el PM25 también había errores. De hecho, no existía serie corregida de PM25, porque habían anulado la serie al completo. Esto verifica lo que he dicho antes de que los sensores de PM25 y PM10 funcionan en líneas generales peor que el resto de sensores.

Por otra parte, la gente del proyecto de la EEA tenía las sospechas de que había países que corregían mejor los datos que otros. Es por eso que realizamos un estudio al respecto. Mediante ese estudio, hemos sacado una tabla, la Tabla 2, con los países que más corrigen sus datos.

	country	NO2	O3	PM10	PM25	complete
1	Bosnia y Herzegovina	57934,8821	18612,7355	41522,4774	80885,4753	49738,8925
2	Turquía	3254,75445	1001,34764	449,602755	4594,13933	2324,96104
3	Kosovo	2095,76283	1063,4807	1168,71735	4836,30515	2291,06651
4	Grecia	3539,80689	630,112558	762,695781	1593,44762	1631,51571
5	Hungría	1632,93528	80,2468727	5,23E-05	0	428,295551
6	Bélgica	579,78976	184,502204	117,806954	237,285167	279,846021
7	Croacia	16,859705	124,415703	267,050701	257,22291	166,387255
8	Chequia	206,434788	36,6627022	28,8430576	63,9782978	83,9797115
9	Macedonia	66,3635312	3,54235138	2,44959062	7,76954352	20,0312542
10	Polonia	14,7169619	2,12609763	3,93541446	8,60231213	7,34519652

Tabla 2: Países que más corrigen sus datos

Para calcular la Tabla 2, hemos cargado los datos de CDR y se los hemos restado a los datos de UTD. En cada columna se muestra la diferencia entre los datos corregidos y sin corregir por contaminante, y en la columna más a la izquierda se muestra la suma de todos los errores. A primera vista, puede parecer que corregir mucho los datos significa que hay mucho trabajo detrás. Sin embargo, nos hemos dado cuenta que hay países como Bosnia y Herzegovina, que solo sube los datos corregidos y no sube ninguno de sus datos sin corregir.

De todas formas, mediante este pequeño estudio de la Tabla 2 nos hemos dado cuenta cuales son los países que no tenemos que tomar de referencia para hacer una corrección de outliers, los países que aparecen en dicha tabla, porque tienen más datos outliers que cualquier otro y

por tanto el estudio quedaría sesgado. Esto se deduce de que al corregir muchos datos, que es lo que estamos verificando en la Tabla 2, es porque tienen muchos datos mal. Esto quiere decir que tienen muchos datos que son anomalías u outliers.

En la siguiente sección, continuaremos con el análisis geográfico de las distintas estaciones para ver si hay una correlación geográfica real entre ellas.

4.3 Análisis geoespacial de los datos

Una de las cosas que nos pidieron la gente de la EEA era buscar relaciones espaciales entre las series temporales de diferentes lugares de contaminantes, como ya hemos mencionado anteriormente. Sin embargo, buscar estas relaciones en series tan largas y que no tienen por qué coincidir los sucesos en el mismo tiempo, es decir, puede que haya un lag, es un trabajo muy arduo. Por eso, nuestra aproximación a esto es crear una serie de características estáticas de las series temporales, para un análisis más sencillo.

Estas características extraídas de las series temporales son:

- **La energía absoluta de la serie:** Esta es la suma de los valores al cuadrado.
- **El número absoluto de la suma de los cambios:** Devuelve la suma del valor absoluta de los cambios consecutivos en el eje x.
- **La media.**
- **La varianza.**
- **La tendencia lineal:** Calcula una regresión lineal de mínimos cuadrados para los valores de la serie temporal desde el valor 0 hasta la longitud de la serie menos uno.
- **Auto correlación agregada:** Calcula el valor de una función de agregación, como, por ejemplo, la media o la mediana.
- **Skewness:** Esta es una medida de asimetría que permite establecer el grado de simetría que presenta una distribución de probabilidad de una variable aleatoria.
- **Curtosis:** Es una característica de forma de su distribución de frecuencias o probabilidad.
- **La transformada de Fourier agregada:** Devuelve la media, la varianza, el sesgo y la curtosis del espectro absoluto de la transformada de Fourier.

Mediante estas características se crea un diccionario de todas las estaciones y todas las series de cada una de las estaciones para su posterior procesamiento. Estos diccionarios se encuentran disponibles para las series CDR y UTD, aunque usaremos CDR ya que son los datos “buenos”.

A continuación, con todos estos datos aplicamos un modelo de machine learning del que ya hemos hablado anteriormente, el algoritmo k-means. Pero para su aplicación, primero escalamos los valores con un escalado normal.

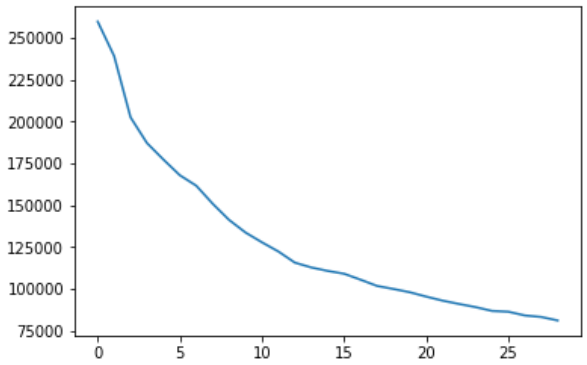


Figura 33: Gráfica del codo

Después de este escalado, buscamos el número de grupos o clusters que se deben hacer por optimización. Para ello, lo hemos representado en una gráfica para tratar de ver de mejor manera el codo, aunque no se puede apreciar correctamente como se puede ver en la Figura 33.

A pesar de esto, hemos creado una función para calcular cuales son los puntos contiguos en los que la inercia cambia más. Este "codo" se encuentra cuando el número de clústeres es 11.

A partir de ahí, creamos los 11 clústeres y posteriormente los mostramos en un mapa con el resultado de la Figura 34.

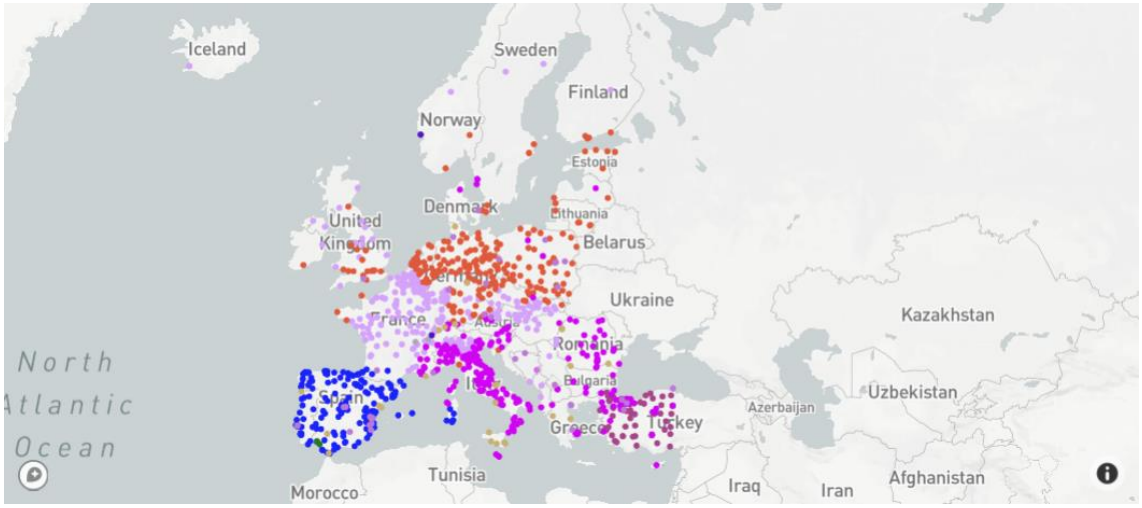


Figura 34: Distribución de las estaciones según los 11 clústeres

Como se puede observar, sí que parece haber una correlación geográfica con las características de las series temporales. Por ejemplo, en España sale prácticamente todo del mismo color, lo mismo pasa con Francia y con la parte norte de Alemania. Esto podría sernos de utilidad en estudios posteriores de las estaciones, para poder correlacionar distintas estaciones y obtener más información.

Después de buscar un patrón geográfico, también hemos tratado de ver si el clustering era capaz de inferir el país, el tipo de estación y el tipo de área. Como ya hemos mencionado anteriormente, las estaciones están clasificadas por país, por tipo de estación y por tipo de área en la que se encuentra la estación.

Para ver si esta inferencia se lleva a cabo, solo vamos a forzar el número de clústeres al número de tipos de área y posteriormente al número de tipos de estación y al número de países.

A continuación, vamos a mostrar en primer lugar un mapa con los clústeres creados por el algoritmo y posteriormente el mapa que queríamos que infiriese. Así en los tres supuestos ya mencionados.

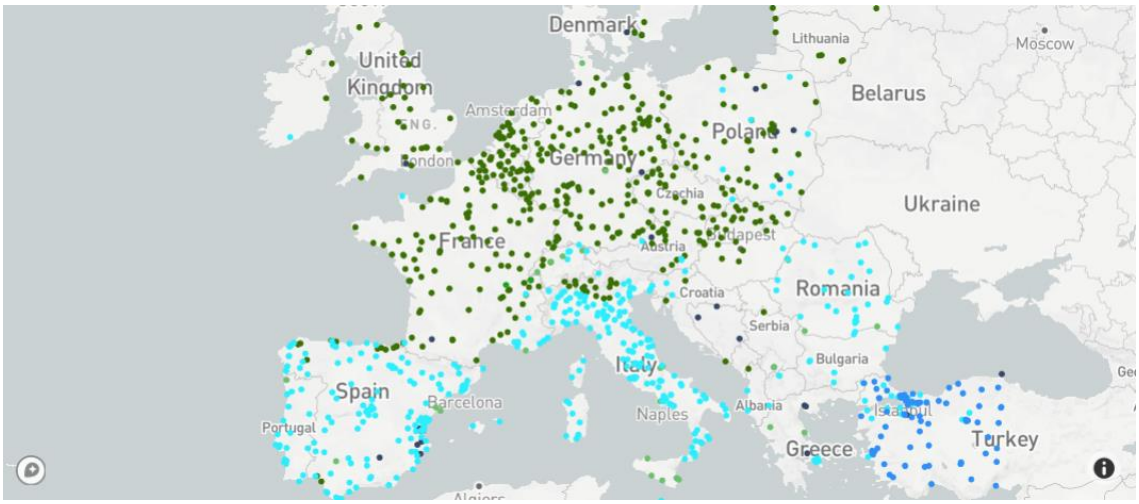


Figura 35: 6 clústeres todas las series temporales

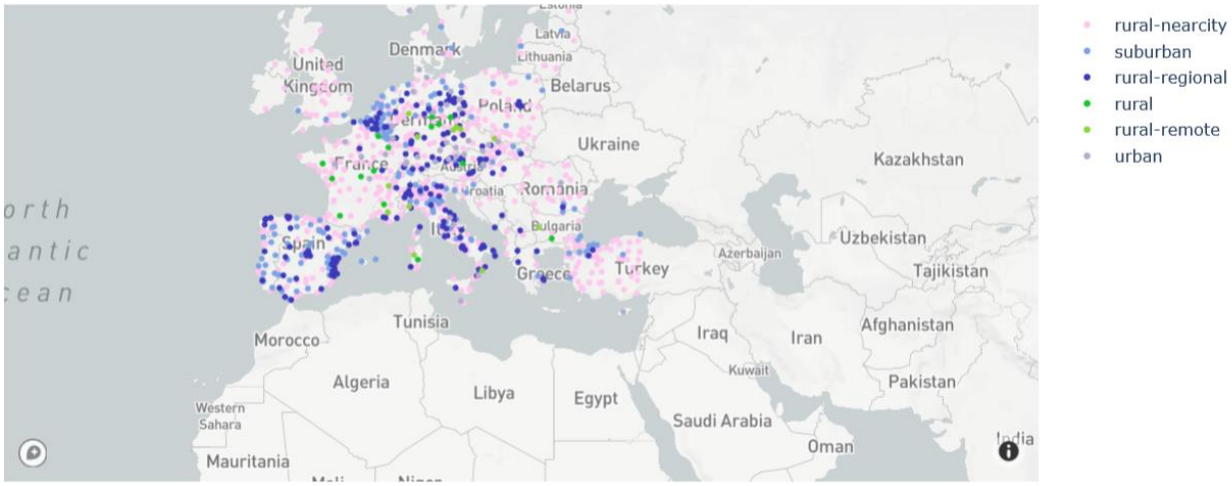


Figura 36: Tipos de estación

Cómo se puede observar en las Figuras 35 y 36, la inferencia no es válida. Los clústeres no se parecen. Por tanto, podemos observar que realmente el tipo de estación no tiene ninguna correlación con los propios datos de las series temporales, y, por tanto, tiene más prioridad la localización geográfica que esta.

Por otra parte, voy a realizar el mismo proceso con los tipos de área para ver si en este caso los clústeres se parecen más, en las Figuras 37 y 38.

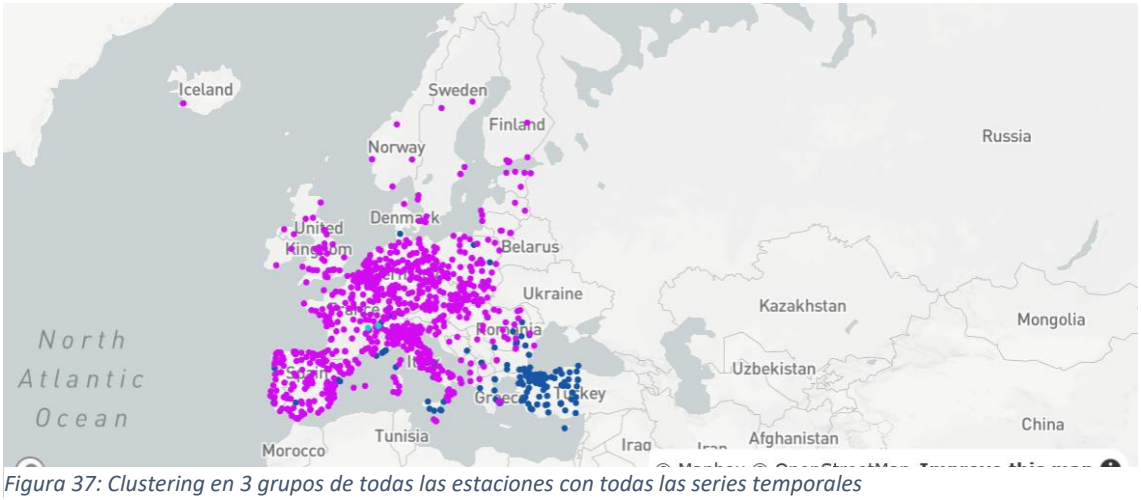


Figura 37: Clustering en 3 grupos de todas las estaciones con todas las series temporales

Figura 38: Tipos de área

Como se puede ver en las Figuras 37 y 38, los tipos de área no tienen nada en común. Esto es como ya hemos dicho anteriormente, significa que los tipos de área tienen poca correlación con los datos que estamos tratando de series temporales.

Por último, vamos a realizar el mismo análisis, pero por país, para comprobar si en este caso sí que hay una mayor correlación.

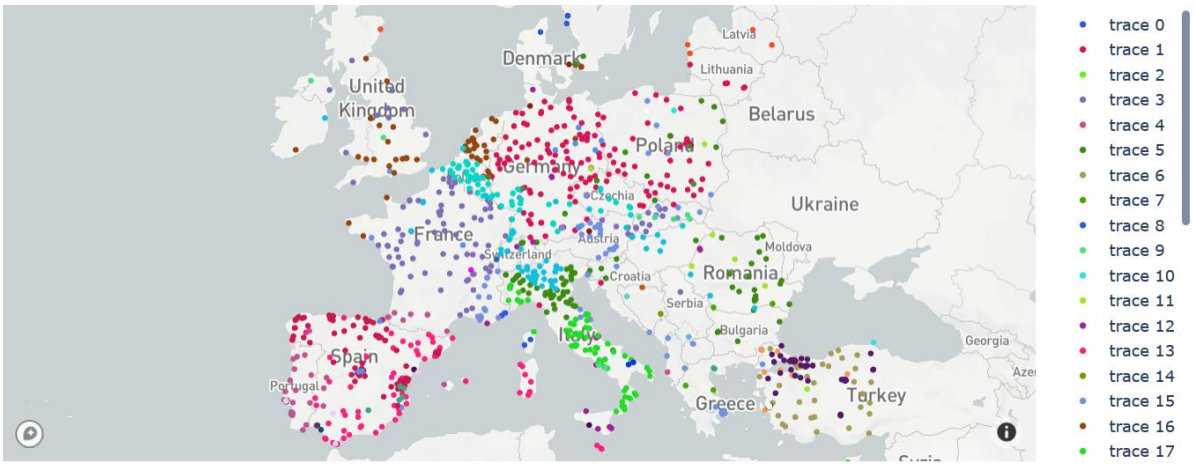


Figura 39: Clustering en 38 grupos de todas las estaciones y con todas las series

En el caso de los países, vemos innecesario mostrar otro mapa con las estaciones clasificando las estaciones por países, porque se ve claro qué estaciones pertenecen a qué país. En este caso, sí que se ve una mayor correlación entre los países y los datos, ya que sí que se pueden apreciar países con el mismo tipo de color en general.

Este análisis que hemos expuesto arriba también ha sido llevado a cabo por cada serie temporal de contaminante de manera independiente, pero al no dar resultados relevantes y ser prácticamente igual que el estudio anterior, hemos decidido no incluirlo en el trabajo de fin de grado.

4.4 Conclusión

Este análisis previo de los datos, nos ha permitido entender los datos un poco mejor, lo cual nos va a venir bien en el siguiente proceso que vamos a realizar.

Por otra parte, hemos visto las correcciones que se llevan a cabo de UTD a CDR, que en algunos casos son simplemente un suavizado, pero en otros con lleva la invalidación de una parte de la serie. Esto nos permite hacernos a la idea de que es lo que la EEA considera un outlier.

Por último, también hemos visto también cual es la relación espacial de los datos, cosa que nos permitirá en un futuro si nos conviene combinar la información de distintas estaciones para realizar un estudio más completo y ahorrarnos tiempo de computación, al combinar distintos resultados en uno.

En el siguiente capítulo, vamos a comenzar a realizar un estudio estadístico de la serie temporal para ver si hay una manera sencilla de corregir con eficiencia los errores de estas.

5.Detección de outliers mediante métodos estadísticos

5.1 Introducción

Para hacer cualquier tipo de análisis sobre las series temporales, tenemos que comprobar si las series son o no estacionarias. Si las series no son estacionarias, significa que las características de la serie van variando de manera aleatoria a lo largo del tiempo, lo cual nos dificultaría en gran medida sacar ningún patrón de ellas.

Por otra parte, después de saber si las series son o no estacionarias, empezaremos a analizar los outliers. Para llevar a cabo este análisis, usaremos cosas que hemos visto en la sección de Time Series de los preliminares, como las medias móviles.

Además, si logramos predecir los outliers, vendría ver si hay alguna manera automática de corregirlos. Para esto intentaremos usar modelos ARIMA, que usan una propiedad de las series temporales que también tendremos que comprobar, denominada auto regresividad, junto con las medias móviles para predecir el siguiente valor. Ajustaremos este algoritmo para tratar de solucionar los outliers.

5.2 Descomposición de la serie

En primer lugar, hay que hablar sobre los datos que vamos a tomar para realizar el estudio. Vamos a realizar los estudios solo sobre una serie ejemplo del conjunto de series CDR. Estos datos están normalizados y los huecos están rellenados para tener muestras siempre cada hora.

A continuación, vamos a mostrar la serie sin tratarla. De esta manera, podremos observar en la Figura 40 que, aunque sea del conjunto de datos que se supone corregido, sigue habiendo

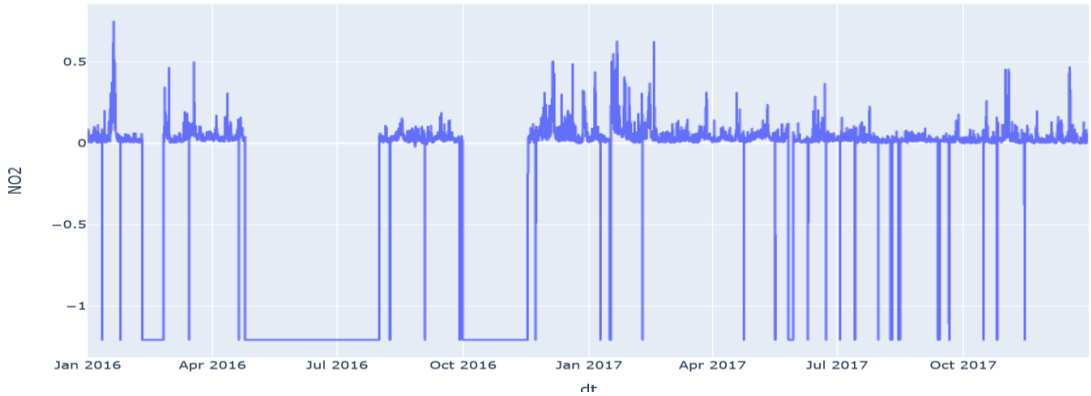


Figura 40: Serie temporal de NO2 sin tratar

muchos errores. Debido a que en las estaciones se recogen 4 series temporales, vamos a centrarnos en el NO2, debido a que es la que se encuentra en más estaciones.

Además de los errores, podemos observar valores mágicos muy negativos. Estos valores van a hacer que la descomposición de la serie se realice de una manera mala. Es por eso, que, para la descomposición de la serie, como ya sabemos de primera mano que esos valores no son buenos, los vamos a quitar. Los huecos que dejan los vamos a rellenar de la misma manera que anteriormente.

Después de observar la serie, vamos a descomponerla en sus 3 elementos, que ya hemos explicado en los preliminares, pero que vamos a refrescar:

- **La tendencia:** Es un comportamiento o movimiento suave de la serie a largo plazo.
- **La estacionalidad:** Son movimientos de oscilación dentro del periodo temporal que estamos estudiando.
- **El residuo:** Son variaciones aleatorias alrededor de los componentes anteriores.

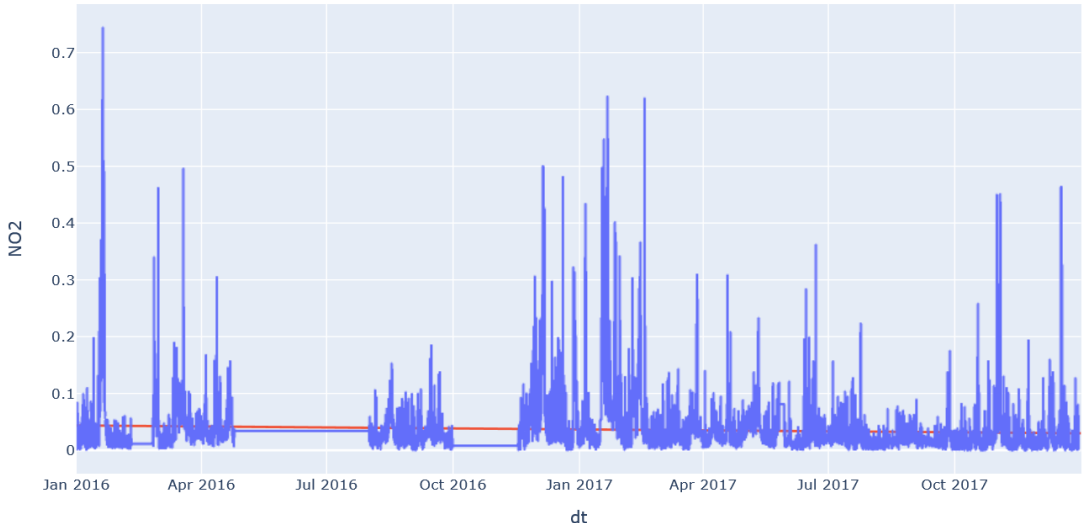


Figura 41: Tendencia lineal en el NO2

Mediante la suma de las 3, se obtiene la serie original. En Python, existe una función que se llama seasonal_decompose que genera los 3 componentes. Sin embargo, vamos a profundizar más en el cálculo de cada uno de ellos.

Para la tendencia tenemos varias formas de calcularla. La primera y más sencilla es la **tendencia lineal**. Esta simplemente asume que la serie sigue una recta y calcula esta recta. La ecuación de esta es la ecuación de una recta.

En nuestro ejemplo, la tendencia lineal es la que se muestra en la Figura 41, la tendencia lineal sería la línea roja y en azul estaría representada la serie original. Como se puede observar, es demasiado simple para que tenga ninguna validez. Solo es una línea que va del primer punto de la serie al punto final.

Otra tendencia posible, es **la tendencia evolutiva**. En esta, el valor en un punto concreto es la media móvil de orden n en ese punto. De esta manera obtenemos un suavizado. En nuestro ejemplo, para hacer la prueba vamos a hacer la media móvil de orden 168, que se puede ver en la Figura 42.

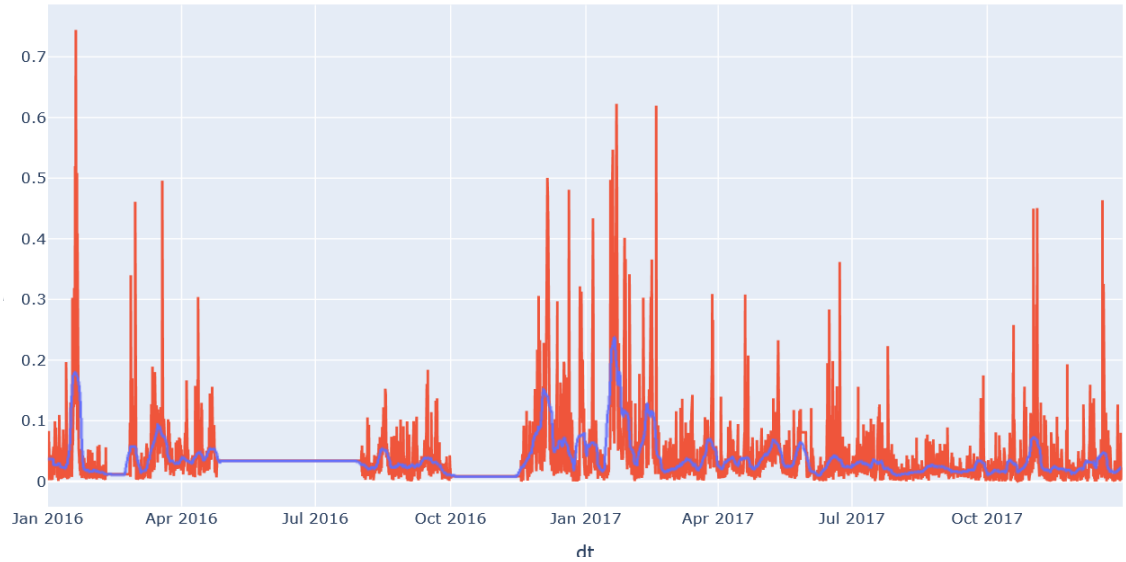


Figura 42: Tendencia evolutiva

Como se puede observar en la Figura 42, esta tendencia es un poco más compleja y refleja mejor la verdadera tendencia.

Sin embargo, estas tendencias no funcionan demasiado bien así que hay que buscar un método más complejo. Este método más complejo nos lo proporciona la función que hemos mencionado anteriormente que se llama seasonal_decompose.

Esta función nos da la descomposición clásica de las series. Dado m, que es la frecuencia de la estacionalidad, se calcula la descomposición de la siguiente manera:

- La tendencia se calcula como 2*m-MA (media móvil) si m es par. Si por el contrario es impar, se calcula como m-MA.
- Se calcula la serie sin tendencia como $y_t - T_t$.
- Se calcula el componente estacional para cada estación, definida por m. Simplemente se calcula la media de los valores sin tendencia de esa estación. Se denomina S_t.
- El residuo se calcula como $R_t = y_t - T_t - S_t$.

Vamos a suponer que m, la estacionalidad, es igual a 168 horas que son las horas de la semana, como se muestra en la Figura 43.

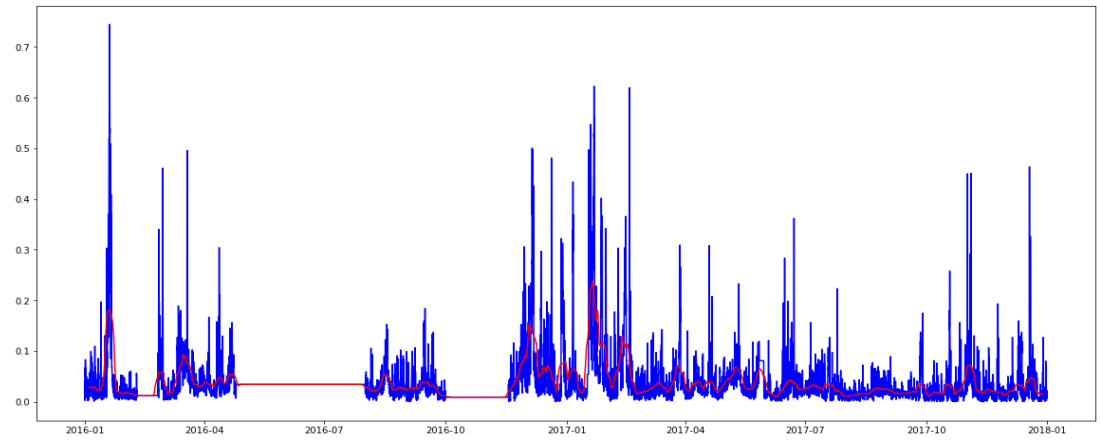


Figura 43: Tendencia de la serie descompuesta por seasonal_decompose

En la Figura 43, se representa la serie original en azul, mientras que la tendencia se encuentra en rojo. Como se puede observar, es una tendencia que podría estar bien descompuesta. Por otra parte, este método también muestra la estacionalidad y el residuo. En la Figura 44, se puede observar la tendencia y la estacionalidad. El residuo no lo hemos incluido para que la Figura sea más clara y porque no aporta demasiado. Además, para verlo con más claridad, hemos cogido solo un trozo de la serie.

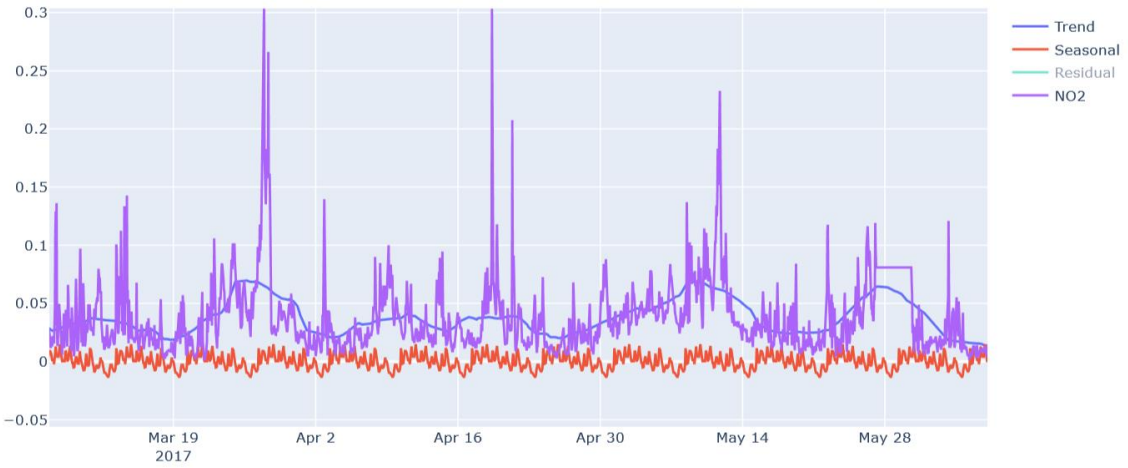


Figura 44: Tendencia y estacionalidad clásica sobre el NO2

Se puede observar en la Figura 44, en morado la serie original, como el rojo el componente estacional de la serie y en azul la tendencia de la serie. Se puede observar que la tendencia suaviza la serie y la estacionalidad se repite periódicamente, de manera que juntas representarían la serie sin ruido.

Este podría ser un método para eliminar el ruido de las series temporales de contaminantes. A pesar de esto, hay demasiados picos que no se preceden con una ascendencia. Es decir, hay picos abruptos. Esto podrían ser perfectamente outliers y, sin embargo, se recogen dentro de la

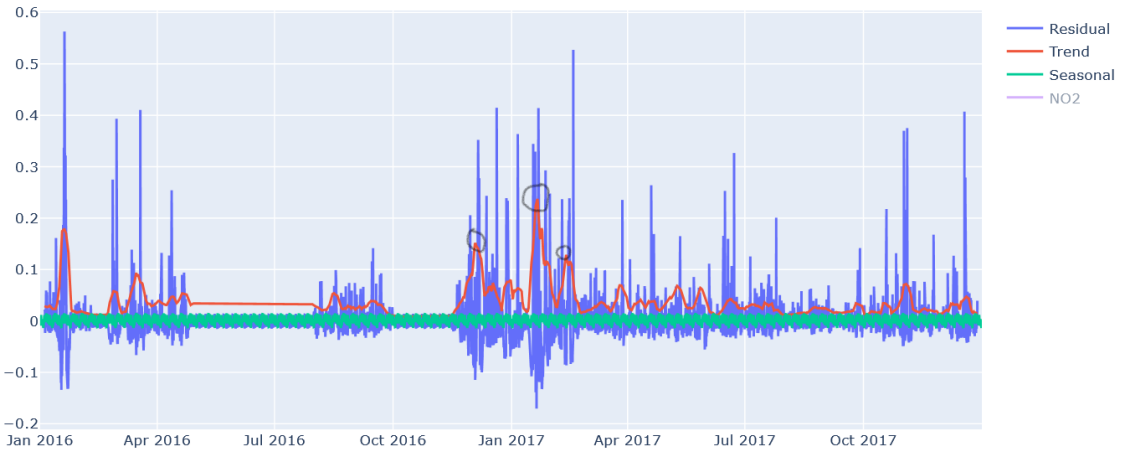


Figura 45: Tendencia, estacionalidad y residuo de una serie temporal de NO2 con los picos marcados en negros

tendencia. Se puede observar esto en la Figura 45. Para solucionar esto se podría cambiar la estacionalidad o buscar otros métodos que suavicen mejor este tipo de casos: Nosotros nos inclinamos por esta segunda opción.

Por tanto, hay que buscar una descomposición que recoja de una mejor manera la tendencia, de una manera más “suave”. Para esto, vamos a usar la descomposición STL, la cual hemos desglosado en los preliminares. Se puede ver la descomposición STL de la serie temporal de NO2 en la Figura 46.

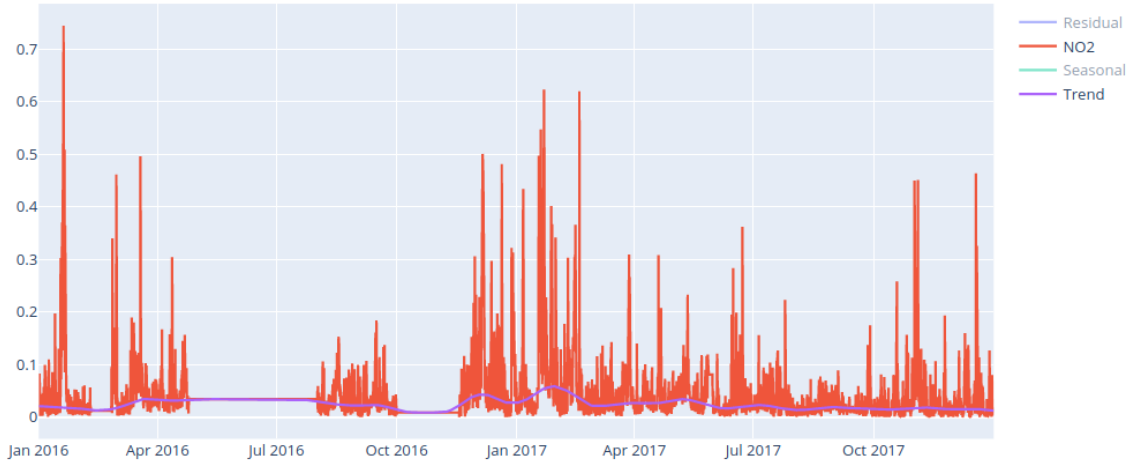


Figura 46: Tendencia STL del NO2

Se puede observar en la Figura 46, que, en este caso, la tendencia sí que es robusta a esos picos. En rojo se puede ver la serie original y en morado la tendencia. Por tanto, podemos decir que este tipo de descomposición funciona mejor que la anterior. Además, este tipo de descomposición cuenta con un parámetro, que es la fracción de Loess, que permite ajustar si queremos que la tendencia se ajuste más o menos a la serie. En la Figura 47, se puede observar un fragmento ampliado con todos los componentes de la descomposición STL.

En la Figura 47, se puede observar en morado la tendencia; en verde la estacionalidad, la cual

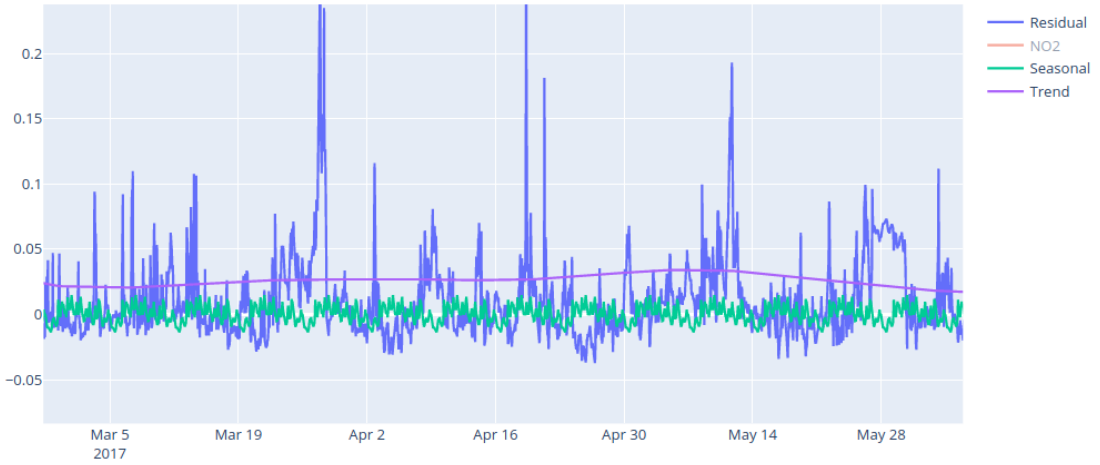


Figura 47: Descomposición STL completa de un fragmento de la serie del NO2

hemos fijado en 168 horas que son las horas de la semana; y en azul el residuo. Observándolo así quizás podemos darnos cuenta de que probablemente se suavice demasiado la serie y se detecte demasiado como residuo, aunque como no tenemos la definición de outlier no lo sabemos.

Para concluir, vamos a hablar de cómo podríamos usar este método para detectar outliers. Bastaría con poner un threshold al residuo, a partir del cual, todos los valores de la serie temporal serían detectados como outliers.

Este tipo de detección de outliers, tienen sus ventajas e inconvenientes. Como ventajas podemos observar que:

- Es una aproximación bastante simple, excepto en el caso de la descomposición STL, y bastante robusta. Puede manejar muchas situaciones diferentes y todas las anomalías pueden ser interpretadas intuitivamente.
- Si los outliers son aditivos, es decir, en un momento puntual se le suma un valor exagerado a la serie, este tipo de métodos lo detectarían con facilidad.

Por otra parte, también existen desventajas como:

- Es una aproximación muy rígida. Es decir, no se pueden ajustar parámetros para que se ajusten mejor al modelo.
- Si hay valores mágicos, este método no los detecta ni los corrige, sino que se ajusta demasiado a ellos. Esto se podría solucionar haciendo una descomposición un poco más

compleja, como la descomposición STL, o eliminándolos previamente como es nuestro caso. Sin embargo, creemos que pueden existir mejores métodos, que son los que vamos a seguir explorando.

5.3 Suavizado de la serie temporal

5.3.1 Introducción

En esta sección vamos a ver maneras de suavizar la serie temporal. Los errores, además de ser picos de valores anómalos, también pueden ser por errores de medición muy sutiles. En ese caso, un suavizado puede ser la mejor opción.

Para realizar este estudio de suavizado, vamos a cambiar de estación, aunque vamos a seguir con el NO2. Esto lo vamos a hacer porque queremos un ejemplo que tenga menos valores mágicos, así que no tengamos que corregirlos y la serie sea natural, y además que, de esta forma, se pueda ver mejor el suavizado.

5.3.2 Media móvil

El primer suavizado que vamos a realizar es el de la media móvil. Al hacer una media móvil con n valores, hacemos que el valor actual tenga menos peso, lo que hace que se suavicen los errores. Este hecho se ve acentuado de que los valores cercanos al valor actual tienen cierta información sobre este, por mera proximidad. Se puede observar el suavizado por la media móvil en la Figura 48.

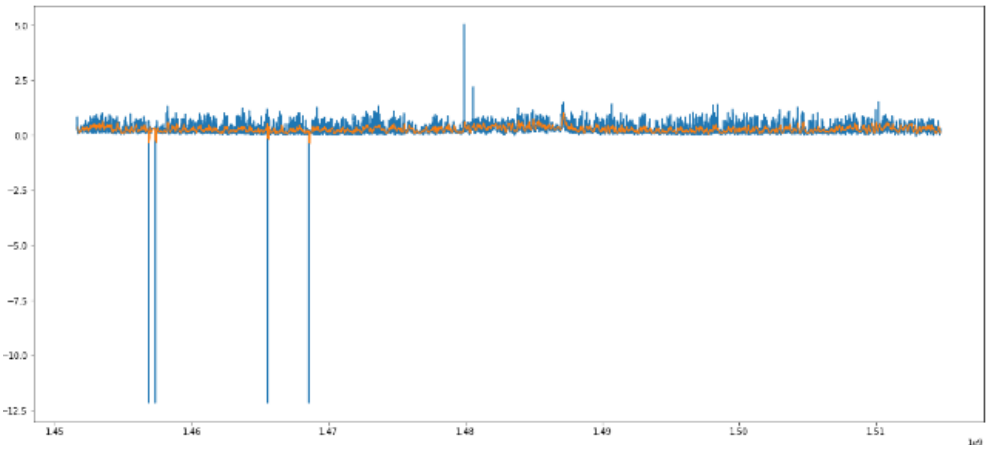


Figura 48: Media móvil con n igual a 24

Se puede observar en naranja, la media móvil que no sigue los picos de la serie original, en azul, sino que sigue una trayectoria más rectilínea, aunque ajustándose al momento temporal así que sí que sube y baja dependiendo del momento. Es notable que este método es un suavizado bastante bueno para lo simple que es, así que es una opción a tener en cuenta.

5.3.3 Media móvil ponderada linealmente

Otra opción de suavizado es usar un promedio móvil ponderado linealmente. Esto es una media móvil, pero mejorada de tal forma que a los valores más cercanos al punto en el que estamos calculando la media tienen más peso que los más lejanos. Se puede observar en la Figura 49, aunque sale una gráfica muy similar a la gráfica de la media móvil anterior.

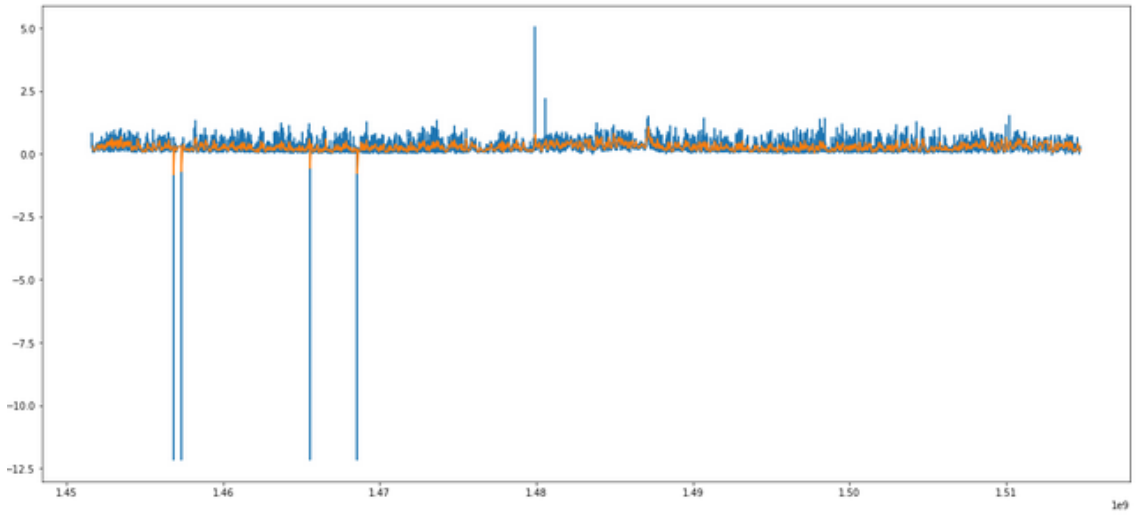


Figura 49: Media móvil ponderada linealmente con n igual a 24 sobre el NO2

Se puede observar que la media móvil en este caso suaviza los puntos más gordos, pero se ajusta más a la serie y adopta como propios valores más altos. Esto se debe a los pesos que se les asignan a los diferentes valores.

5.3.4 Anchor based

Otro método que se puede usar para suavizar es el denominado anchor based. Lo que hace es utilizar un peso, que se le pasa como parámetro, para ponderar el valor anterior y el actual y dar un resultado, que sería la serie suavizada. En el fondo es como una media móvil, es decir, usas valores vecinos de manera directa para suavizar la gráfica, y, por tanto, los resultados son parecidos como se pueden ver en la Figura 50.

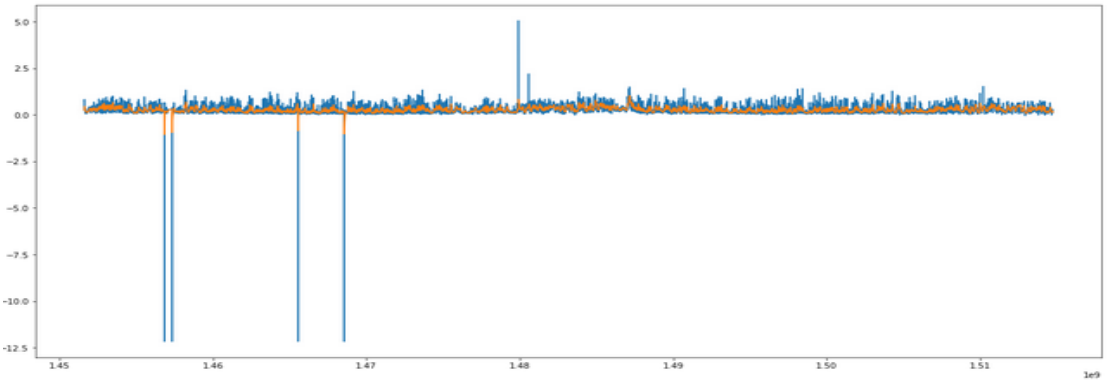


Figura 50: Anchor based con 0.9 sobre el NO2

El valor de la proporción de este método es muy importante. Cuanto más cercano a 0 sea, más se ajustará la serie al valor original, y cuanto más lejano sea este de 0, más se ajustará a la serie movida 1 posición hacia la derecha.

5.3.5 Holt winters

Como ya hemos explicado Holt-winters es un triple suavizado exponencial. Por recordar, un suavizado exponencial es un método de predicción de series temporales con datos univariables.

Los modelos de series temporales como ARIMA, que hemos explicado y exploraremos más adelante, hacen que la predicción sea una suma ponderada de las últimas observaciones. Un suavizado exponencial hace algo similar. Usa una suma ponderada de las últimas observaciones, pero el modelo usa una reducción exponencial de los pesos para observaciones más antiguas.

Específicamente, las observaciones pasadas son ponderadas con una ratio de reducción exponencial. Existen múltiples tipos de suavizados exponenciales, que vamos a explorar para poner solución a nuestro problema de detección y corrección de outliers.

Uno de estos suavizados es **el suavizado exponencial único o simple**, que es un método de suavizado para predecir datos univariables sin una tendencia o estacionalidad. En Python, este suavizado esta implementado en SimpleExpSmoothing en la clase Statsmodels. Para probar qué tal funciona, vamos a entrenar el suavizado exponencial con el 80% de los datos y vamos a ver cómo se las apaña con el resto de los valores. Se puede ver en la Figura 51.

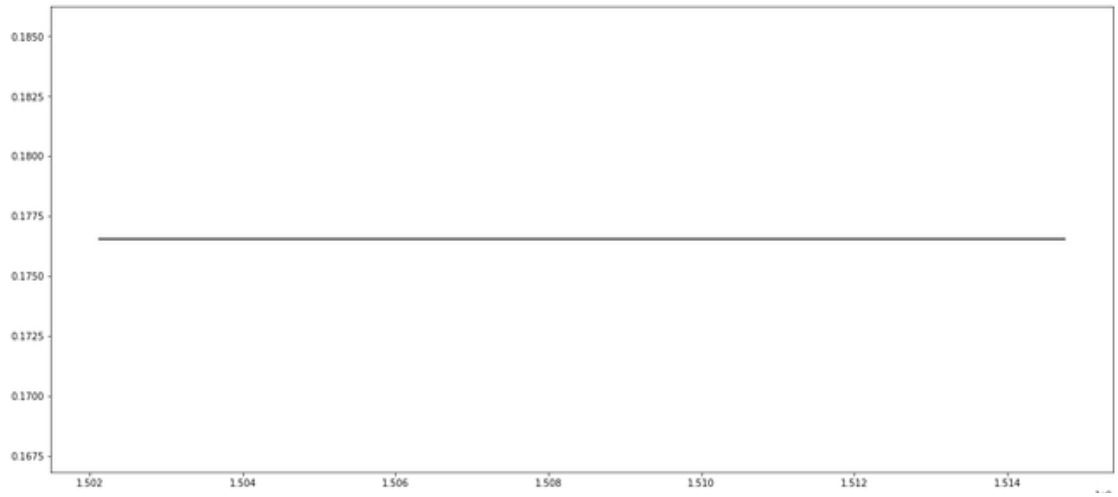


Figura 51: Suavizado exponencial simple

Como se puede observar este suavizado exponencial, no aporta demasiado porque se basa solo en una línea recta, así que no sirve para nada. Solo he mostrado el resultado del suavizado exponencial porque si la mostraba junto a la serie original no se apreciaba el suavizado.

Por suerte, existen más tipos de suavizados exponenciales, como el doble o el triple. El doble es una extensión del simple que añade soporte para tendencias en series temporales univariables. El triple, por otro lado, es una extensión del suavizado exponencial doble, que añade la capacidad de modelar la estacionalidad en series temporales univariables.

A estos dos métodos se les llama Holt-Winters, debido a sus dos desarrolladores. Todo esto está explicado con más detalle en los preliminares.

La librería que vamos a usar para realizar todos estos suavizados es Statsmodels.tsa.holtwinters.

Para aplicar Holt-winters hay que ajustar una serie de paramentos, los cuales voy a explicar y justificar mi decisión acerca de su ajuste:

- **La tendencia:** Tiene que ser modelada como aditiva o multiplicativa. He decidido suponer que es aditiva porque en la descomposición que hemos hecho antes, no daba rastros de ser multiplicativa.
- **La amortiguación:** Si la tendencia de debe estar amortiguarla o no. Para una primera versión, no voy a amortiguarla, aunque igual luego se hacen otros casos de uso.
- **La estacionalidad:** El tipo de componente estacional, es decir, al igual que con la tendencia, si es aditiva o multiplicativa. Voy a poner en primer lugar aditiva, por lo observado en la descomposición anterior, aunque igual posteriormente se cambia.
- **Los periodos de estacionalidad:** Saber cada cuanto se repite la estacionalidad. Voy a poner cada semana, porque tiene sentido que los mismos días de la semana haya los mismos coches y fabricas funcionando, que son los principales emisores de contaminación.

Se puede observar en la Figura 52 el resultado de este suavizado.

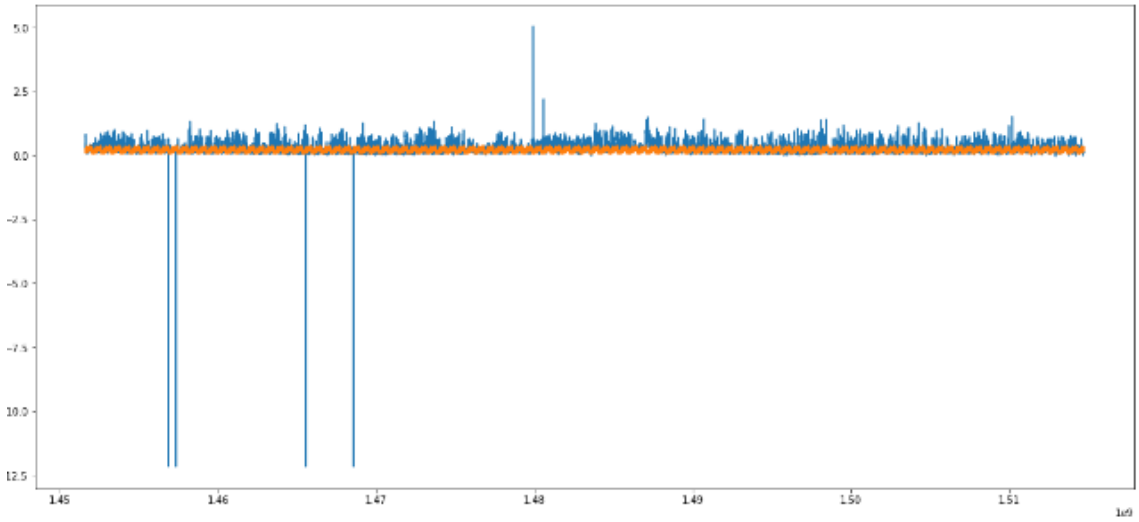


Figura 52: Suavizado Holt-Winters

Se puede observar en naranja en la Figura 52, el resultado de este suavizado. Sin embargo, pese a ser un modelo más complejo que, por ejemplo, la media móvil, no se comporta mejor.

5.3.6 Conclusión

Hemos visto muchos modelos en esta sección para suavizar las series temporales. Según mi criterio, el que mejor funciona para nuestro caso es la media móvil ponderada linealmente, que además coincide que es un modelo simple.

Este modelo por sí mismo no es capaz de detectar los ouliers, pero combinado con otros sí que puede ser capaz de solventarlos. Lo veremos en posteriores estudios.

5.4 Detección de outliers con pruebas estadísticas básicas

En esta sección, vamos a explorar la manera habitual de detectar outliers. Normalmente se suele suponer que los valores de la serie siguen una distribución normal. A partir de ahí, se dice que

los valores que superen n veces la desviación son outliers. En series temporales también se puede usar, aunque solo detectará outliers que sean desviados con respecto a todos los valores, sin tener en cuenta la temporalidad. Los valores anómalos según esta aproximación están marcados con cruces rojas en la Figura 53.

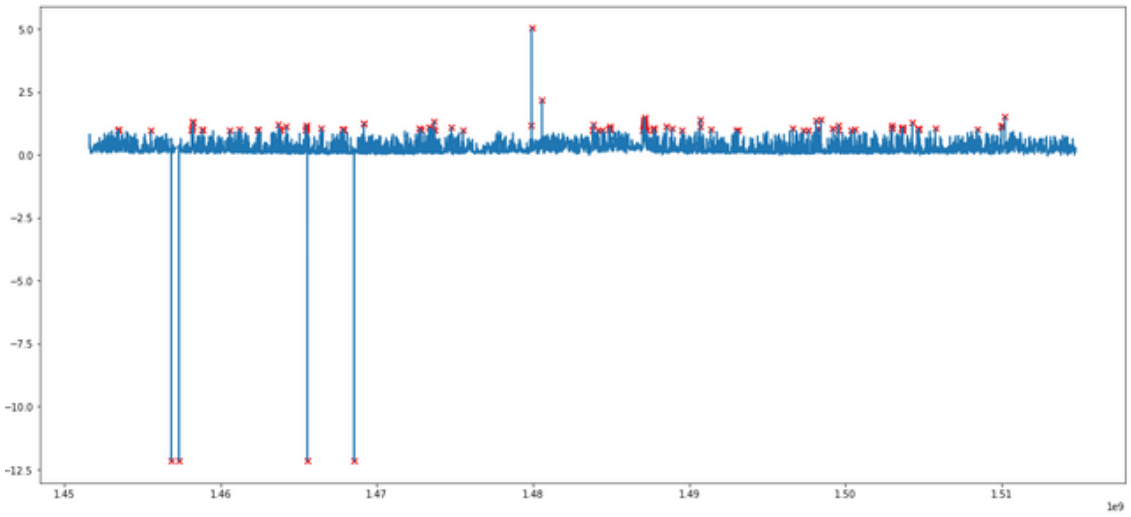


Figura 53: Outliers detectados por superar 2 veces la desviación.

Como se puede observar en la Figura 53, se detectan los outliers que superan la media por mucho, pero puede que haya falsos positivos porque en un día concreto hay más tráfico y también falsos negativos, porque un valor se ponga a cero por un error de medición y como la media está cerca de cero no lo detecte como outliers.

A pesar de esto, es un método que funciona bastante bien y hay que tenerlo en cuenta para estudios posteriores.

5.5 Noise filtering

En esta sección, vamos a filtrar el ruido. Para eso vamos a definir un par de funciones: La primera que va a eliminar el ruido y la segunda que va a obtener el ruido que hemos eliminado anteriormente.

Pero... ¿Cómo eliminamos el ruido? Teóricamente, si la serie es autoregresiva, el valor de un punto es el punto medio del punto que le antecede y que le precede. Es decir, si se hace interpolación de la media, tendría que dar el valor intermedio. La resta del valor teórico y el valor experimental, es el ruido. Se puede ver en la Figura 54, los valores sin ruido y el ruido.

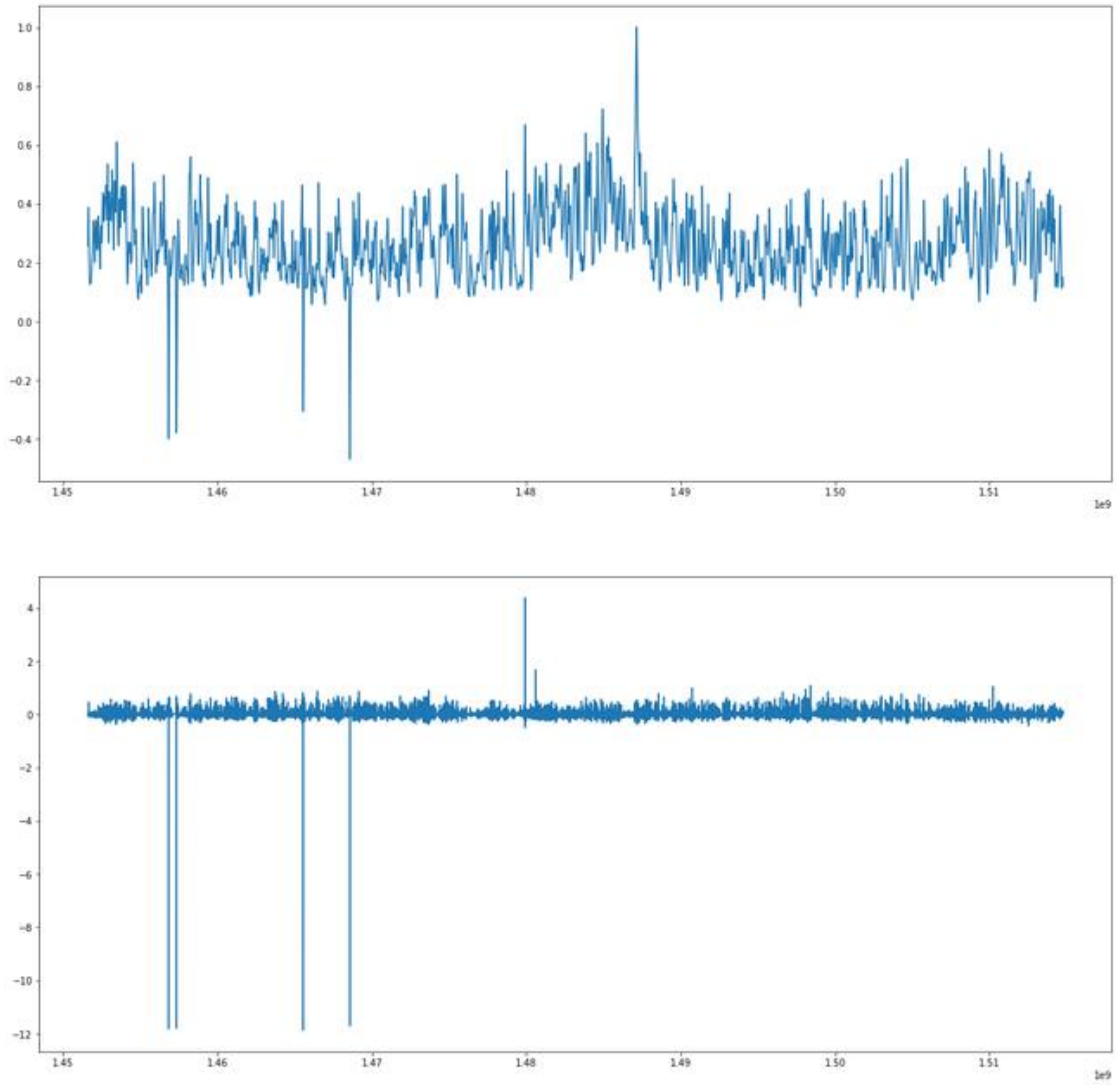


Figura 54: Serie filtrada de ruido. Arriba: Serie sin ruido. Debajo: Ruido

Como se puede observar en la Figura 54, la serie sin ruido es mucho más coherente que la serie normal. Sin embargo, no nos vale como detección de outliers porque está eliminando demasiada información de la serie original.

5.6 ARIMA, Autoregressive Integrated Moving Average

Como ya hemos mencionado y explicado anteriormente, ARIMA es un modelo que se usa para predecir valores de las series temporales. Vamos a probar a usarlo para eliminar los outliers.

Para ARIMA hay 3 parámetros fundamentales:

- **p**: Es el número de términos autorregresivos. Permite incorporar el efecto del pasado en nuestro modelo. Un ejemplo de esto sería decir que va a hacer calor mañana porque ha hecho calor los últimos 3 días.
- **d**: Es el número de diferencias no estacionales. Sería como decir que va a hacer la misma temperatura mañana porque la diferencia de temperatura de los tres últimos días ha sido muy pequeña.

- **q**: Es el número de términos de la media móvil. Esto nos permite decir que el error de nuestro modelo es una combinación lineal de los últimos puntos.

Para buscar los mejores parámetros que se ajustan a nuestros datos, vamos a usar dos librerías de ARIMA que buscan los mejores y los aplica. Según ese método los mejores valores que se aplican a nuestro ejemplo son con p igual a 1, d igual a 0 y q igual a 2. Para demostrar que se han elegido los mejores parámetros, se crean las gráficas que se pueden observar en la Figura 55.

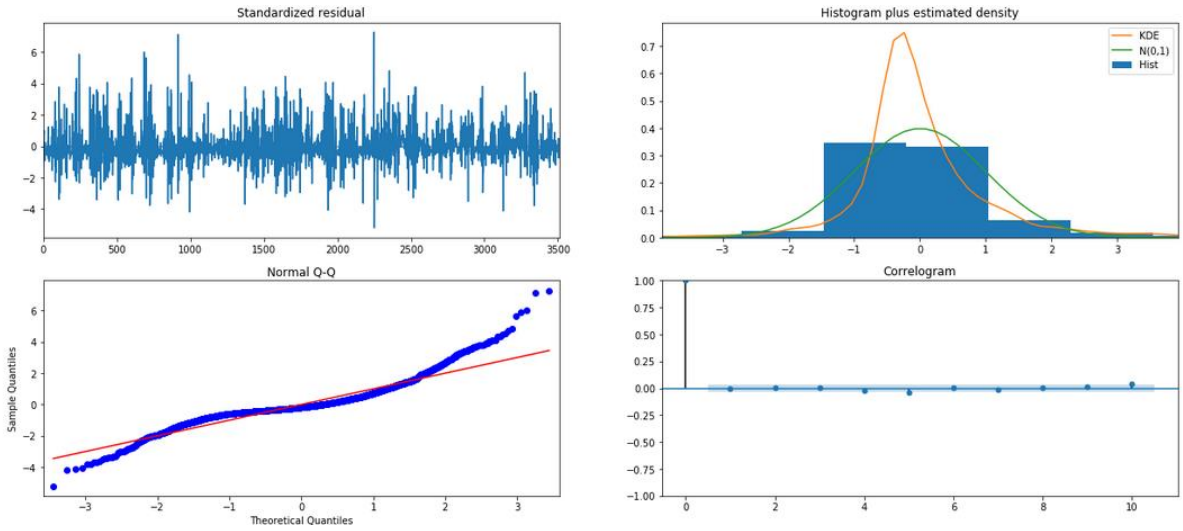


Figura 55: Diagnósticos de los parámetros ARIMA

En la Figura 55 se pueden observar 4 gráficas cuyo significado no está muy claro, así que nos disponemos a explicarlo:

- **La de arriba a la izquierda:** Los errores residuales fluctúan alrededor de una media de cero y casi todos tienen la misma varianza. Hay algunos que tienen mucha más varianza, que probablemente serán outliers.
- **La de arriba a la derecha:** El histograma no se ajusta a la normal de media 0 y varianza 1.
- **La de abajo a la izquierda:** Todos los puntos deberían estar alineados con la línea roja. Que no lo estén significa que la muestra esta sesgada.
- **La de abajo a la derecha**: Es el correlograma, muestra los errores residuales que no están autocorrelados.

Después del análisis de los parámetros de la Figura 55, vamos a predecir para ver qué tal funciona el modelo ARIMA a la hora de la verdad. Se muestra en la Figura 56.

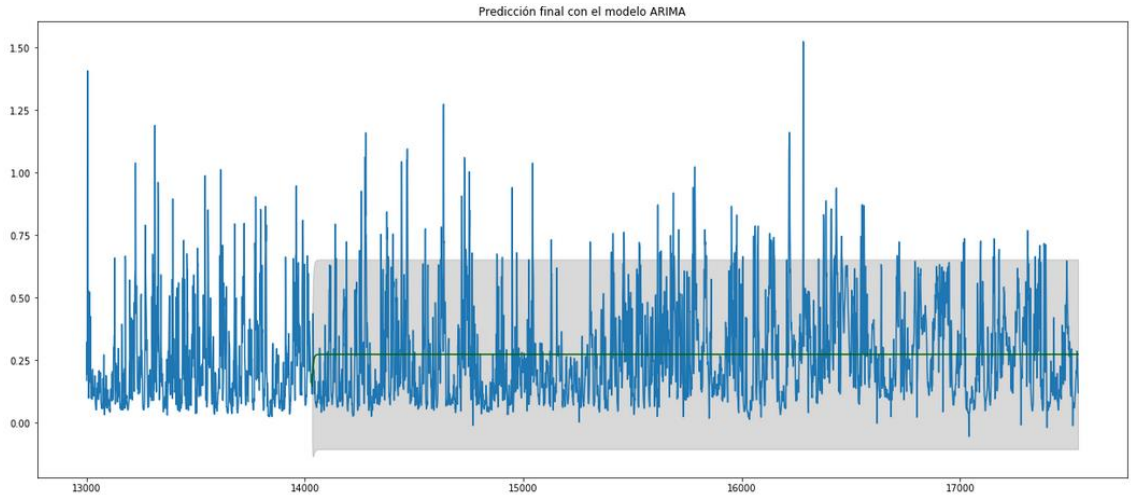


Figura 56: Predicción ARIMA

Como se puede observar en la Figura 56, la línea verde que representa la predicción ARIMA funciona correctamente en los primeros valores, pero luego se establece en una línea recta que es la tendencia. Esto se debe a que el modelo ARIMA no tiene en cuenta los efectos de la estacionalidad. Para eliminar este problema, se desarrolló el modelo SARIMA, que es un modelo ARIMA que también modela los efectos estacionales.

Calculamos los parámetros de este modelo de la misma manera que en el caso del ARIMA convencional y mostramos los resultados en la Figura 57.

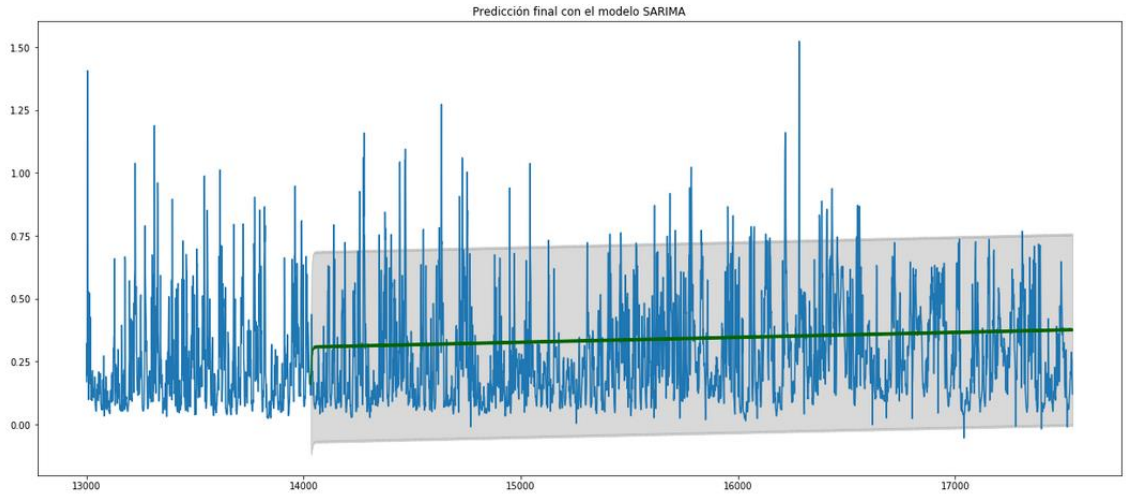


Figura 57: Predicción SARIMA

Cómo se puede observar, es prácticamente el mismo resultado que en el caso del ARIMA convencional, aunque se modela mejor el cambio de la tendencia hacia arriba.

5.7 Conclusiones

En este capítulo, hemos visto métodos estadísticos para la detección de outliers. Por los resultados obtenidos, hemos llegado a la conclusión que una buena aproximación para la detección de buena parte de los outliers y para su posterior corrección, es aplicar el método de

la desviación a la serie temporal, posteriormente eliminar los outliers y predecir estos valores mediante ARIMA.

Hemos implementado un ejemplo de esto sobre la misma serie y podría ser algo como lo mostrado en la Figura 58.

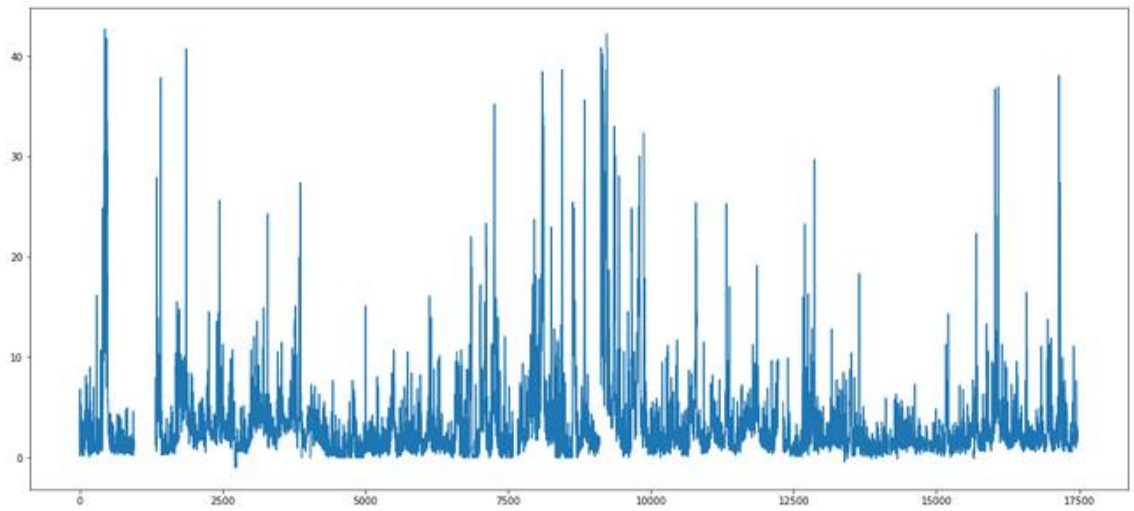


Figura 58: Detección y corrección con ARIMA

Se puede observar que los outliers con respecto a la serie temporal que estamos tratando todo el rato, han disminuido notablemente. Es decir, el método funciona razonablemente bien. Sí que es verdad que hay espacios en blanco, lo cual habría que solucionar, por ejemplo, mediante interpolación de la media.

Sobre los suavizados y la eliminación del ruido, no es una buena opción en la mayoría de los casos porque hace que pierda la serie temporal mucha información. Sin embargo, si la serie está plagada de outliers sí que puede ser una buena opción.

Por último, la descomposición de la serie temporal para el análisis de outliers puede ser una buena opción, aunque en el capítulo siguiente exploraremos métodos que están mucho más pensados para detección de outliers y que es probable que funcionen mejor.

6.Detección de outliers con machine learning

6.1 Introducción

En este capítulo, vamos a aplicar los métodos de detección de outliers que hemos explicado en los preliminares. Hay que destacar que son métodos que no se aplican usualmente a series temporales, sino que se aplican a datos cuya relación consigo mismos no es temporal.

Esto nos desfavorece, ya que parte de la riqueza de los datos de los que disponemos es que están relacionados entre distintos momentos, y que el instante en el que se ha tomado la medición es muy relevante.

Sin embargo, en las siguientes secciones comprobaremos realmente su validez y veremos qué tal funcionan.

6.2 Clustering

En esta sección vamos a utilizar k-means para determinar si un valor es o no un outlier. Vamos a utilizar este algoritmo de distintas maneras que veremos a continuación.

Lo primero que vamos a hacer es quitar los valores mágicos. Así como, en el análisis estadístico, no eliminábamos los outliers para buscar un método lo suficientemente robusto que pudiese detectar todos los outliers incluidos los valores mágicos, en este caso sabemos que el k-means es muy delicado así que es mejor borrar lo que sabemos que está mal.

Para eliminar los valores mágicos, eliminamos todos los valores que sean menores que 0, ya que son valores imposibles, y rellenamos estos huecos con interpolación de la media. Se puede ver el resultado en la Figura 59.

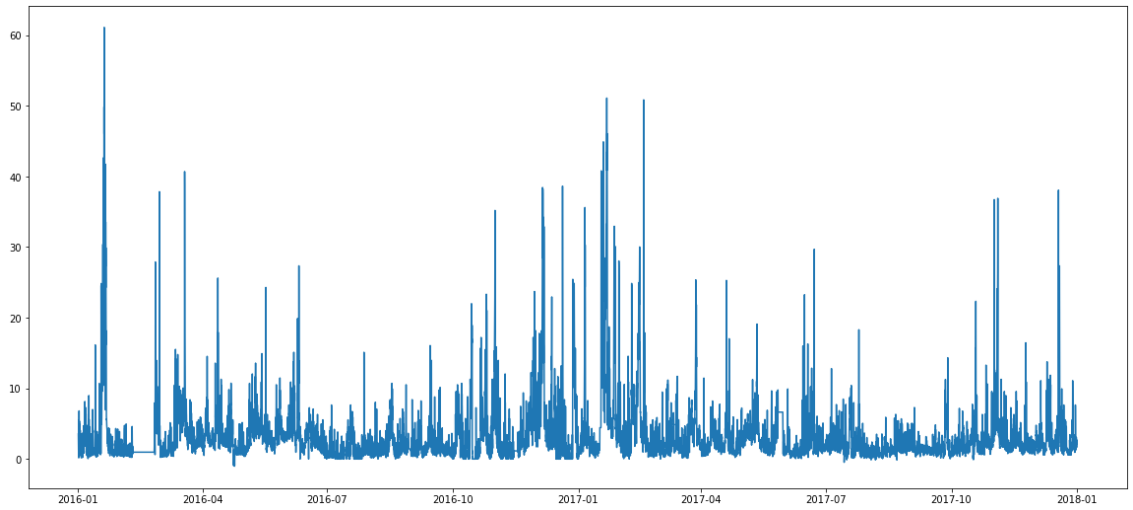


Figura 59: Análisis preliminar antes del clustering

Para detectar outliers con clustering, podemos hacerlo de 3 formas:

- **Utilizando ventanas temporales:** Podemos coger ventanas de tiempo para representar cada dato como él mismo y n datos hacia el pasado. De esta manera, modelamos la auto regresividad de la serie.
- **Utilizando los datos de todas las series de la misma estación:** De esta manera vemos si conjuntamente hay outliers del sensor.

- **Hibridando**: Si usamos ambas maneras escritas anteriormente, podrá mejorar el resultado.

En nuestro caso solo explicaremos el caso de solo utilizar ventanas temporales y el de hibridar.

6.2.1 Uso de ventanas temporales

Dentro de este método vamos a usar dos maneras. La primera es una que hemos pensado nosotros. Al final, k-means es un algoritmo de machine learning no supervisado que se encarga de hacer grupos de puntos parecidos dentro de un hiperplano multidimensional. Entonces, si forzamos al algoritmo a hacer dos grupos, tiene sentido pensar que uno de ellos será de outliers y el otro será el de los valores propiamente de la serie.

Vamos a usar durante toda esta subsección el NO2 para realizar el estudio, ya que hacerlo sobre todas las series se haría muy largo y pesado.

Para comenzar vamos a usar ventanas de 3 valores. Se puede observar esto en la Figura 60.

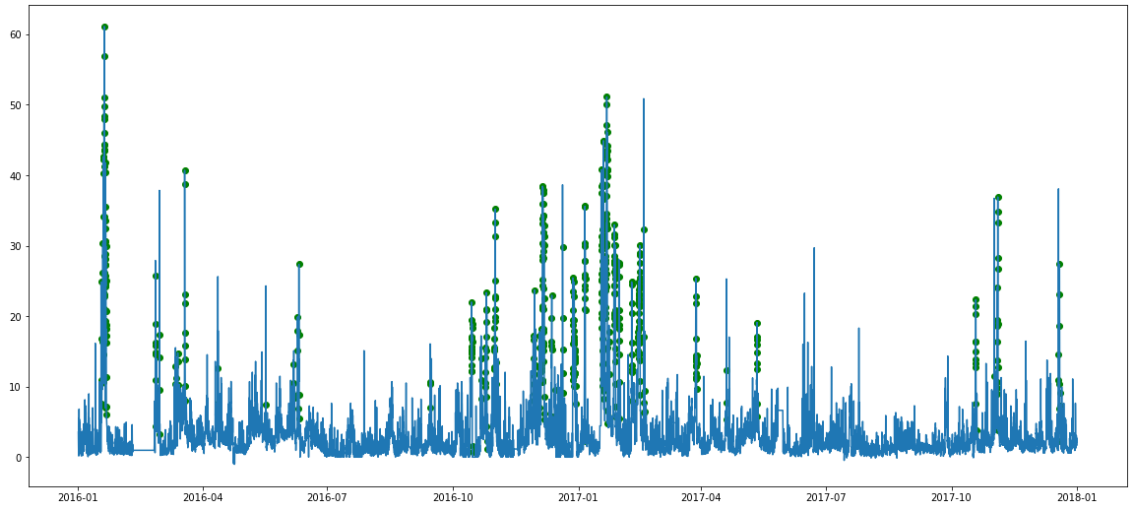


Figura 60: El resultado obtenido con ventanas temporales de 3 valores

Se puede observar en la Figura 60, que parece que funciona bien. El único inconveniente es que no sabemos cuál es el clúster de outliers, ya que no están etiquetados, pero nos quedamos con el clúster con menos puntos.

Ahora vamos a probar con ventanas temporales de 24 valores, para ver cómo se comporta en este caso y si mejoramos el resultado anterior. Podemos observar esto en la Figura 61.

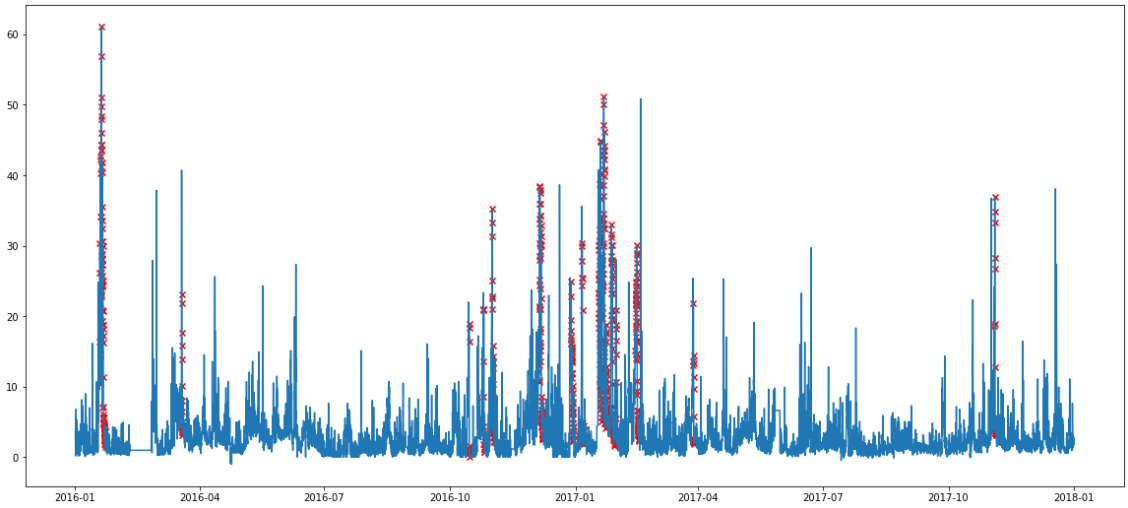


Figura 61: El resultado obtenido con ventanas temporales de 24 valores

Podemos observar en la Figura 61, que si usamos 24 valores se detectan picos, pero menos que si usamos 3 valores temporales.

Por último, vamos a probar a usar 100 valores. Podemos observar esto en la Figura 62. Como se puede observar, al poner un número de valores como ventana temporal, podríamos estar teniendo algo parecido a la **estacionalidad**, el número de valores en los que la serie sigue un patrón determinado.

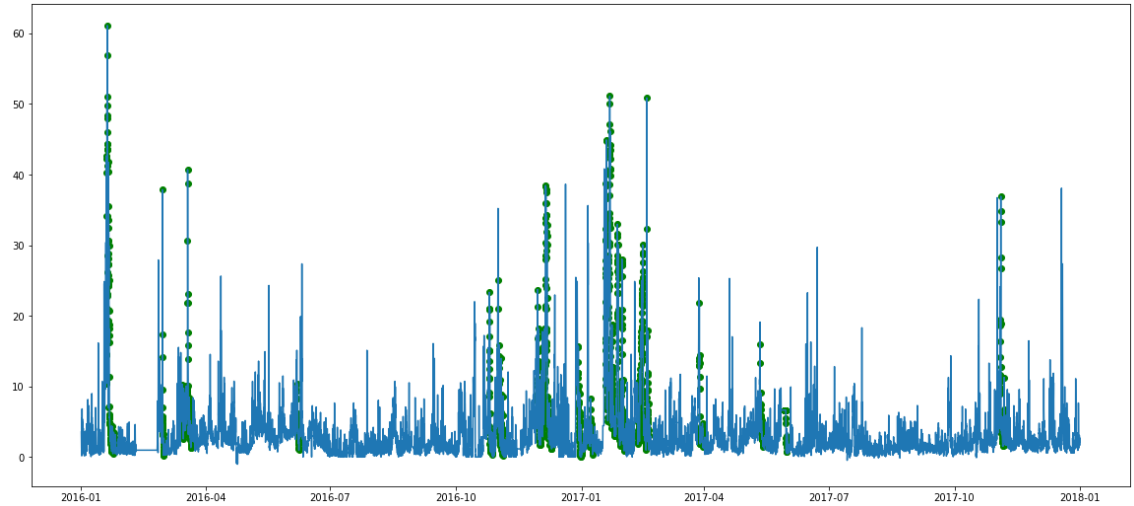


Figura 62: Ventanas temporales de 100 valores

Se puede ver en la Figura 62, que con 100 valores detecta bien los outliers, pero hay falsos positivos.

Esto puede ser porque la influencia de un valor muy cercano es la misma que la de uno muy lejano. Para ello vamos a hacer una función que asigne más valor a los momentos del pasado más cercanos y menos valor a los momentos del pasado más lejano. Este peso se asigna de manera lineal.

Se puede observar en las Figuras 63 y 64, el clustering tal como estamos haciéndolo previamente, pero con esta nueva asignación de pesos. El tamaño de la ventana es 24 y 100 respectivamente. Hemos decidido no realizarlo con una ventana de 3 valores, porque es tan pequeña que el resultado no va a ser muy diferente.

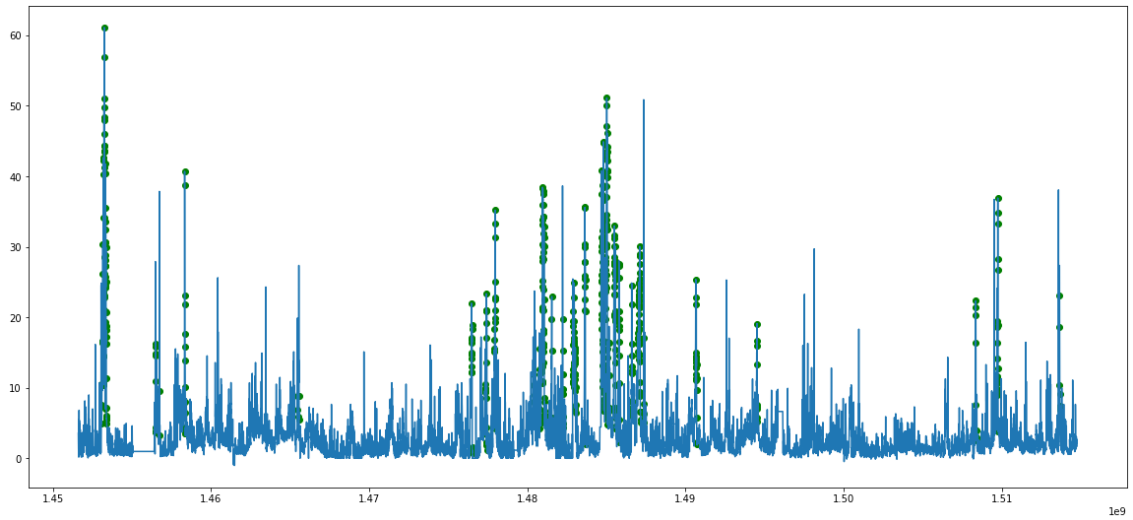


Figura 63: Ventanas temporales de 24 valores con decrecimiento lineal de los pesos

Como observamos en la Figura 63, la detección de outliers funciona mejor que en los casos anteriores. Sin embargo, sigue habiendo muchos outliers que sigue sin detectarse, y que el método de la desviación sí que detectaría a pesar de ser mucho más simple. Sigamos con la Figura 64, en la que se puede observar la misma forma de hacer clustering, pero con ventanas de 100 valores temporales.

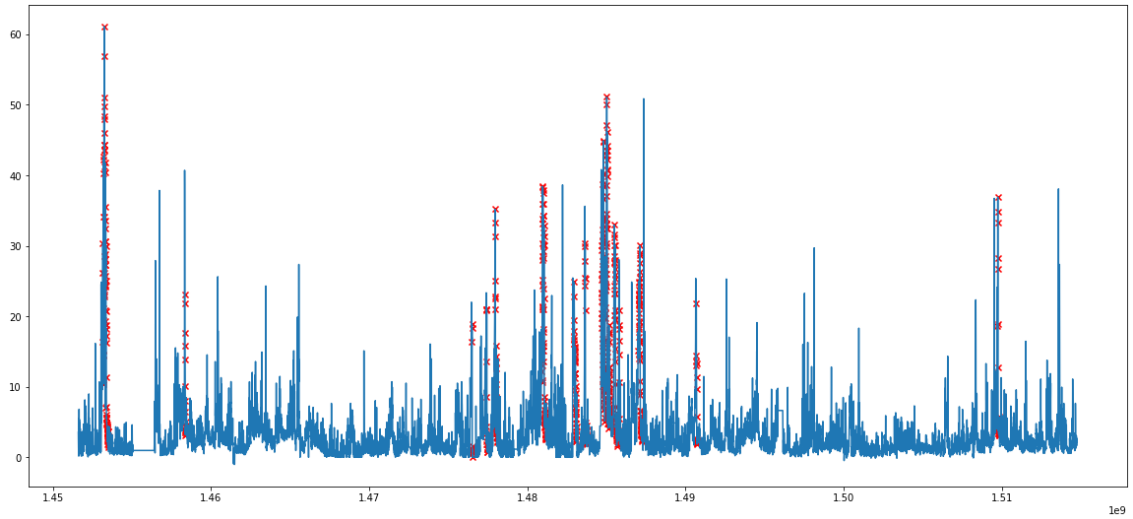


Figura 64: Ventanas temporales de 100 valores con decrecimiento lineal de los pesos

En el caso de la Figura 64, se puede observar que han desaparecido algunos falsos positivos, pero también han desaparecido detecciones que eran correctas.

A pesar de esto, creemos que hacer un decrecimiento de los valores a medida que se aleja el instante medido del instante en el que estamos haciendo la detección es una buena aproximación. Quizás habría que estudiar distintas maneras de hacer decrecer los pesos.

Como hemos dicho anteriormente, el uso de ventanas temporales tiene dos maneras de aplicarse. La primera es la que hemos explorado anteriormente y ahora vamos a probar con otra.

La que vamos a usar ahora consiste en hacer n clústeres y ver los elementos más lejanos de los centroides. Estos elementos lejanos a los centroides serán los elementos que el algoritmo no consigue clasificar correctamente y, por tanto, outliers.

Algo interesante de este método es que se puede fijar cual es el porcentaje de ouliers que hay en la muestra. Esto es un problema en algunos casos porque puede que no sepamos cuando no sepamos cual es este porcentaje. Nosotros lo hemos fijado en un 5% por defecto.

Para calcular el número de clústeres optimo, vamos a usar la regla del codo como hemos explicado anteriormente. No vamos a incluir las imágenes por longitud y porque no las hemos usado, sino que hemos usado una función que encuentra automáticamente el codo sin ver la imagen.

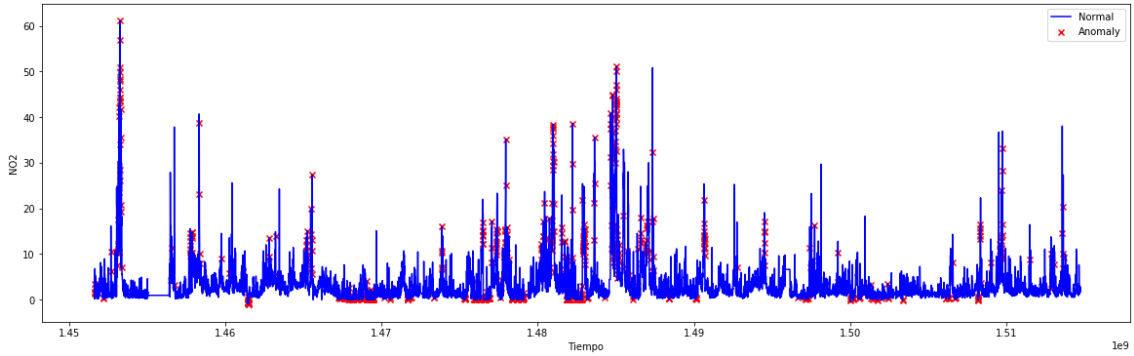


Figura 65: Ventanas temporales de 3 elementos siendo outliers los elementos más alejados del centro del clúster

Para 3 elementos en la ventana temporal, el número óptimo de clústeres es 6. Se muestra en la Figura 65, el resultado de este procedimiento. Se detectan los outliers más notables y también detecta algunos no observables a simple vista. En contraposición a esto, nos da la impresión de que detecta falsos outliers.

A continuación, en la Figura 66, se muestra el mismo procedimiento con ventanas temporales de 24. En este caso, el número óptimo de clústeres era 5.

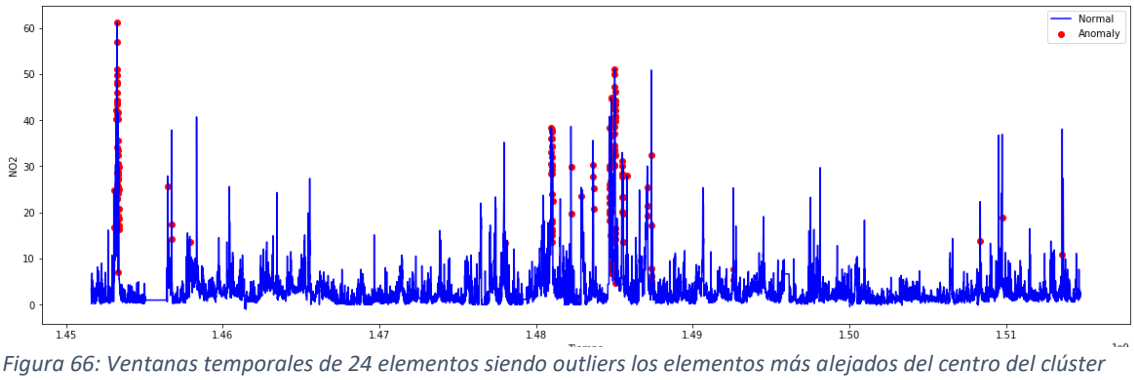


Figura 66: Ventanas temporales de 24 elementos siendo outliers los elementos más alejados del centro del clúster

Con ventanas temporales de 24, parece que detecta algo mejor y no detecta tantos falsos positivos. Sin embargo, hay outliers no detectados. Esto puede ser porque al introducir datos

tan del pasado, a pesar de haber introducido un decrecimiento del peso de los componentes temporales más viejos, no se pondera correctamente el punto en sí mismo.

Por último, probemos este método con ventanas temporales de 100 elementos que se puede ver en la Figura 67.

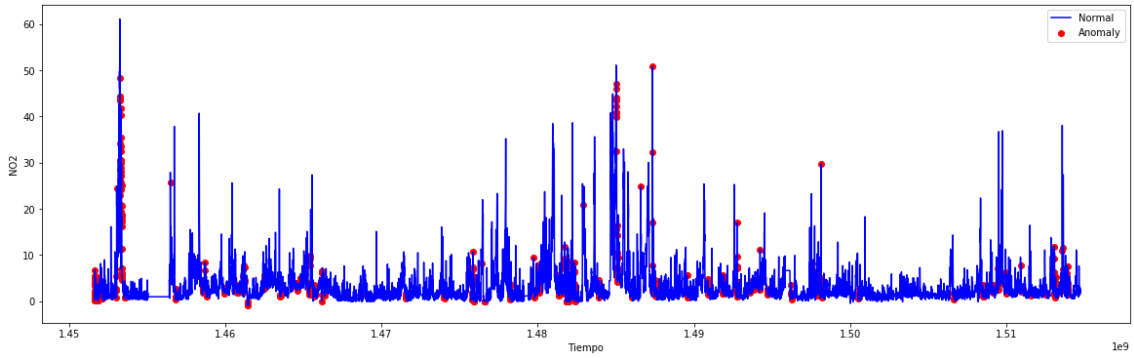


Figura 67: Ventanas temporales de 100 elementos siendo outliers los elementos más alejados del centro del clúster

No parece que dé buenos resultados, según la Figura 67. Hay demasiados falsos positivos y hay muchos outliers claros que no están siendo detectados.

En síntesis, usar ventanas temporales es bastante útil al detectar los outliers. La manera que mejor ha funcionado según mi criterio es usar ventanas temporales de 3 y hacer dos clústeres, uno de outliers y otro de valores buenos.

6.2.2 Uso de K-Means con varias series temporales simultáneamente para detectar outliers y varios datos de carácter temporal

En este caso, vamos a aprovecharnos de que las series entre ellas están relacionadas para tratar de detectar outliers. Como en el dataset que tenemos no se encuentran todos los contaminantes, vamos a tratar de encontrar otro que contenga todos los datos. También

quitamos en este caso los valores mágicos. Se pueden observar las series temporales en la Figura 68. El dataset que usamos en este caso es STA.DE_DEBB065 de UTD.

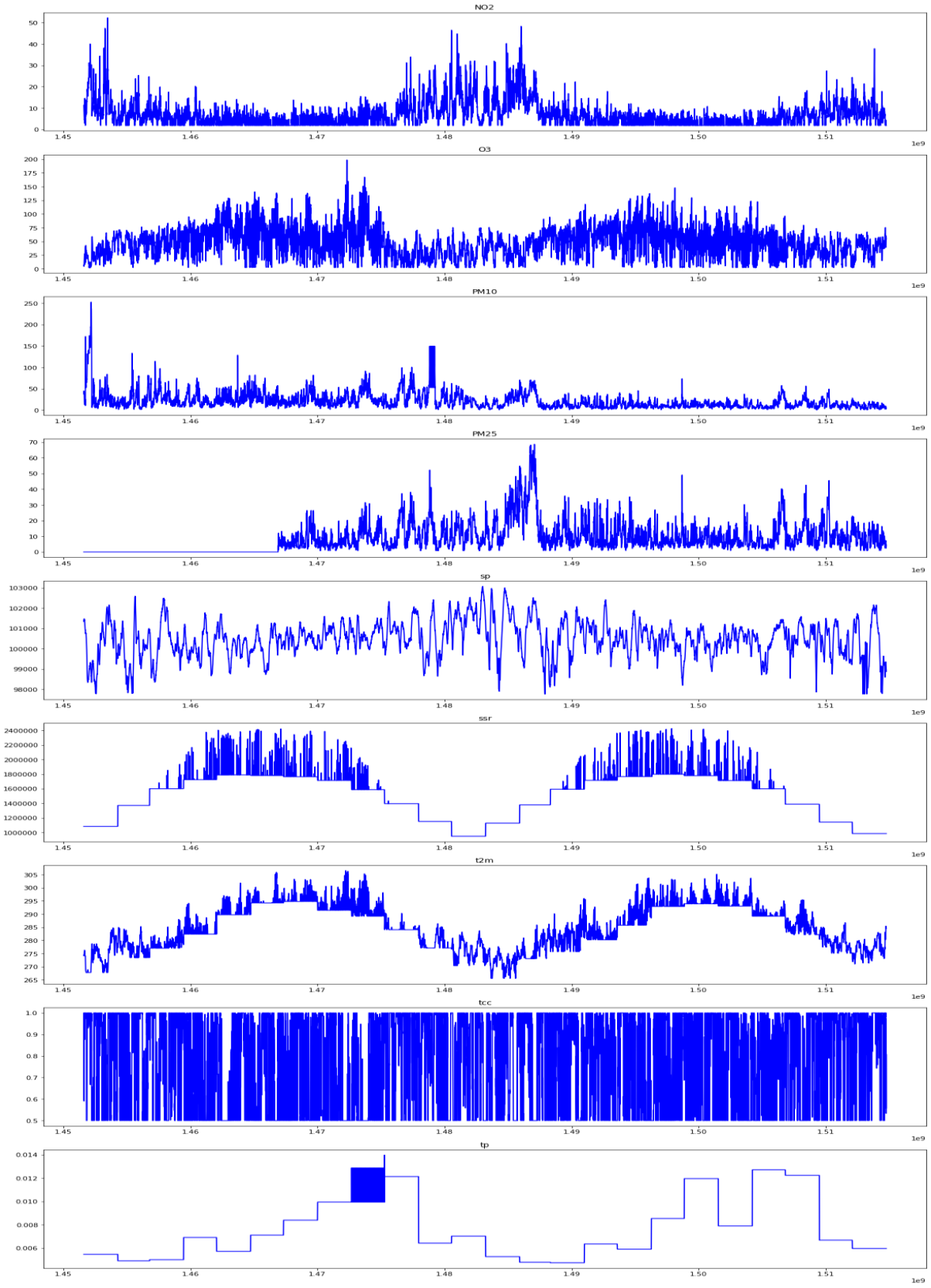


Figura 68: Todas las series temporales antes de aplicar ningún procedimiento de detección de outliers

Como en el caso de las ventanas temporales, vamos a usar el método de usar solo 2 clústeres, uno de outliers y otro de datos buenos, y el método de los vecinos más lejanos al centro. Además de todas las series simultáneamente, también vamos a usar ventanas temporales de 3 elementos.

En primer lugar, en la Figura 69 se puede observar el método de los 2 clústeres. Solo vamos a mostrar los contaminantes porque en las medidas meteorológicas no estamos tratando de encontrar outliers.

Figura 69: Series temporales de contaminantes con detección de outliers conjuntos con el método de los dos clústeres

Como se puede observar, al detectar los outliers conjuntamente, no se detectan correctamente. Esto se debe a que un outlier en una serie no tiene por qué ser necesariamente un outlier en otra serie. Esto da como resultado muchos falsos positivos.

Veamos ahora el método de los vecinos más alejados del centro del clúster en la Figura 70. También hemos puesto en este caso ventanas temporales de 3 elementos. Hemos fijado el porcentaje de outliers a 1% porque si no aparecían demasiados falsos positivos.

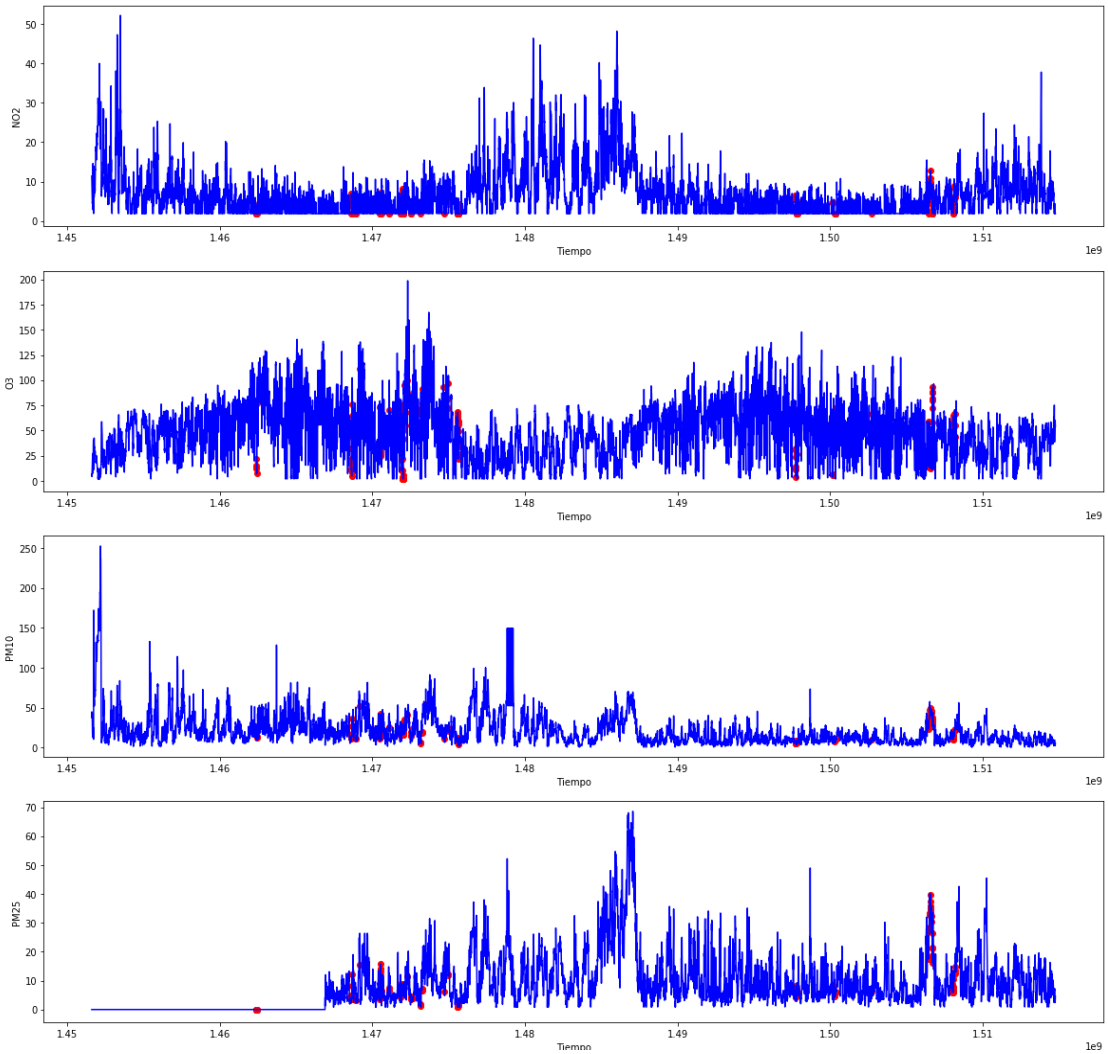


Figura 70: Series temporales de contaminantes con detección de outliers conjuntos con el método de los vecinos más lejanos al centro

Cómo se puede observar en rojo en la Figura 70, hay outliers que se detectan, pero hay mucho que no se detectan.

Como conclusión de esta subsección, podemos decir que añadir todas las series temporales para la detección de outliers no es efectiva, a pesar de que tengan cierta correlación entre ellas.

6.2.3 Conclusión

Podemos observar, que la mejor manera de aplicar el clustering para detectar outliers es por tripletas de valores, usando el método de los dos grupos y sin usar más de una serie temporal a la vez. Este método es útil y podría ser usado para la verdadera detección de los outliers en un futuro.

6.3 Isolation forest

Como hemos explicado en los preliminares, isolation forest trata de separar cada punto de los datos. De esta manera, los puntos que sea más fácil separar serán más outliers que los que sean más difícil de hacerlo. Vamos a aplicar este método sobre todas las series temporales simultáneamente, sin aplicar ningún tipo de ventana temporal como sí que hacíamos en el clustering, para ver cuál es el resultado. Este resultado se muestra en la Figura 71. En esta Figura, las diferentes series tienen diferentes escalas según los valores que toman.

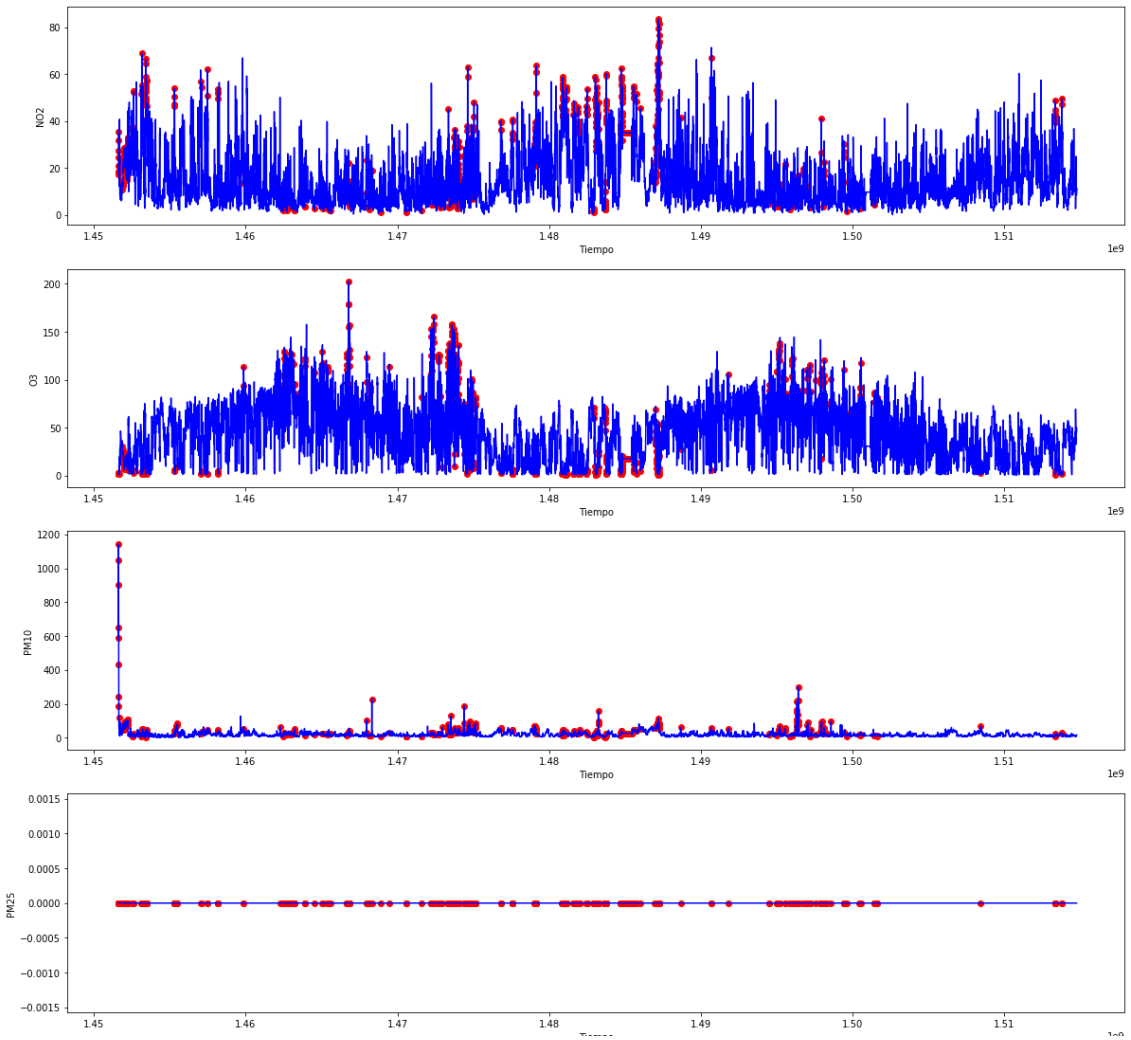


Figura 71: Series temporales de contaminantes con detección de outliers por isolation forest

Antes de aplicar el método, vamos a eliminar los valores mágicos, ya que no queremos que el resultado se desvirtúe por estos valores tan anómalos que sabemos de antemano que son malos. Los rellenaremos con interpolación de la media.

Cómo se puede ver, hay muchos falsos positivos y también se detectan outliers que los son en una serie, pero no en todas, y, por tanto, se detectan de manera errónea en algunos contaminantes.

Para solucionar esto, vamos a usar solo una serie temporal sin utilizar ventanas temporales ni nada. Esta serie temporal va a ser el NO2 y se puede ver en la Figura 72.

Como se puede comprobar en la Figura 72, este método hace lo mismo que el método de la desviación típica, ya que tiene muy pocos datos sobre los que operar. Solo detecta como outliers los datos que están muy alejados de la distribución de los datos atemporales de la serie.

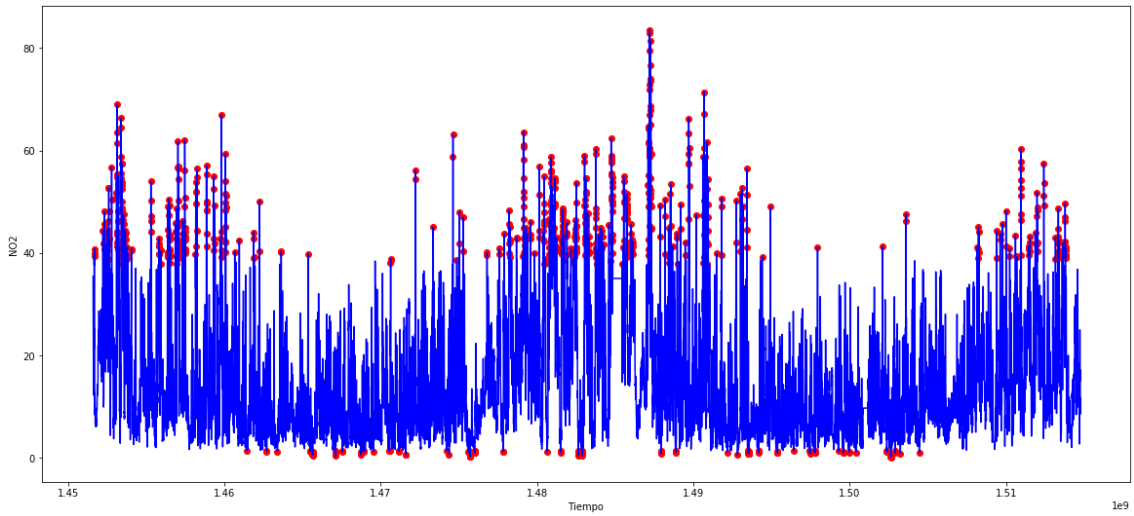


Figura 72: Isolation Forest sobre el NO2

Para ampliar los datos sobre los que puede operar además de incluir temporalidad, usábamos ventanas de tiempo. Así que vamos a ver cómo funcionan las ventanas de tiempo con el Isolation Forest sobre el NO2.

Lo primero que vamos a probar es a hacer ventanas temporales de 3 instantes, para ver el rendimiento, se puede observar en la Figura 73.

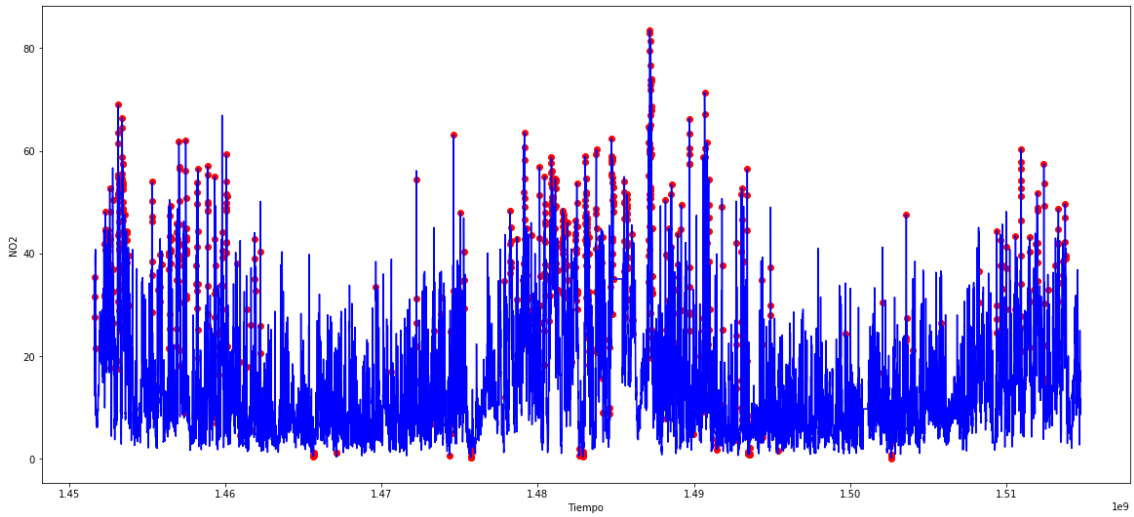


Figura 73: Isolation Forest sobre el NO2 con ventanas de tiempo de 3 instantes

Se puede observar en la Figura 73, que hay muchos puntos outliers que no se han detectado. Sin embargo, hay muchos otros que sí ha detectado. Es decir, el modelo con ventanas de tiempo funciona razonablemente bien. A esto hay que sumarle que no tenemos definido qué es un outlier así que no podemos decir objetivamente si está bien o si no lo está. Para solucionar los pocos outliers detectados, se puede aumentar el porcentaje de outliers a un 5%.

También se han realizado pruebas con ventanas de tiempo de 24 y 100 instantes, las cuales se pueden observar en las Figuras 74 y 75. Sin embargo, estas pruebas no han dado mejor resultado que usar 3 instantes.

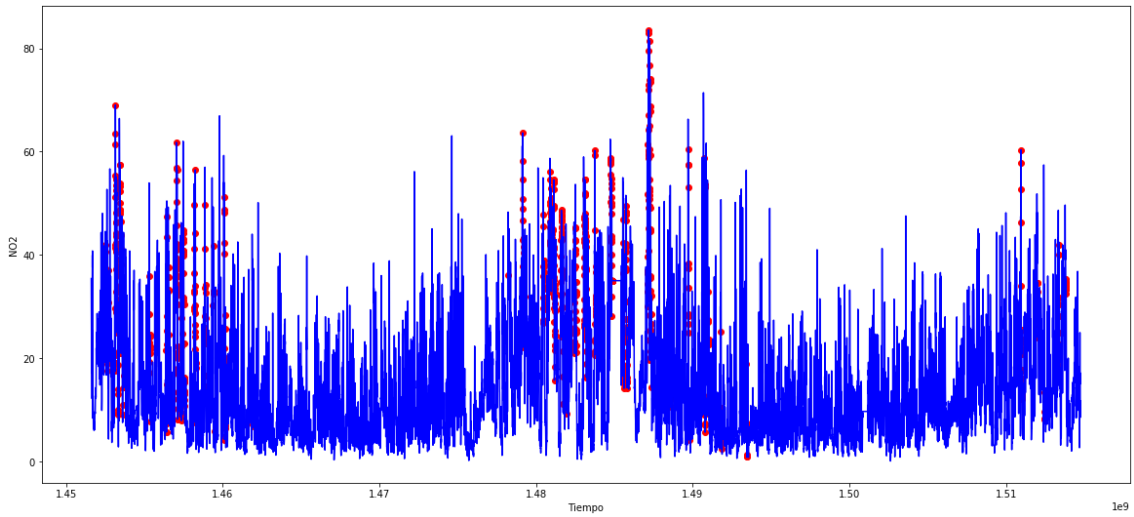


Figura 74: Isolation Forest sobre el NO2 con ventanas de tiempo de 24 instantes

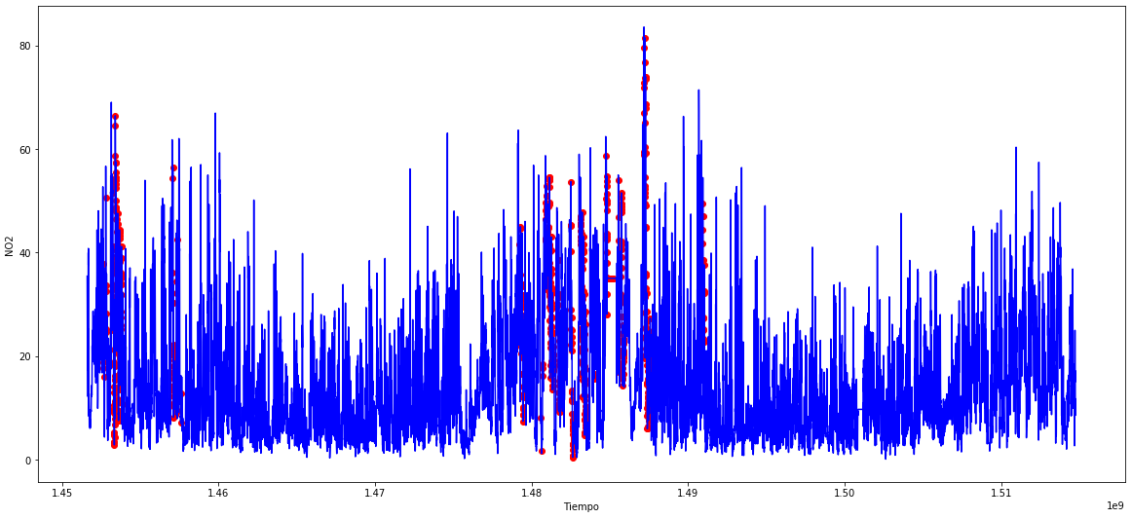


Figura 75: Isolation Forest sobre el NO2 con ventanas de tiempo de 100 instantes

De algo que podemos darnos cuenta viendo las Figuras 73, 74 y 75, es que cuanto más aumentamos el tamaño de la ventana, encontramos outliers más globales y más zonales. Es decir, en vez de encontrar puntos distribuidos por toda la serie, nos encontramos con zonas con muchos más puntos. Esto es interesante de cara a implementar un método de detección de outliers de producción, se podría crear un ensemble de un algoritmo como este con diferentes tamaños de ventana, de manera que se mezclasen los resultados de la ejecución de algoritmo con varios tamaños de ventana temporal, mejorando su eficiencia.

Por otra parte, no hemos hecho un análisis con ventanas temporales de más de una serie viendo que ni en el clustering ni en este mismo algoritmo ha dado muy buen resultado hacer un análisis multiserie.

En conclusión, este método puede ser válido usando ventanas temporales y una sola serie temporal a la vez. Además, no merece la pena usar ventanas temporales de más de 3 instantes, ya que la detección si solo usamos una única instancia de isolation forest no se ve visualmente mejor. Algo que hay que tener en cuenta también es que, a pesar de la complejidad de este

algoritmo, no da muchos mejores resultados que el clustering, cuya implementación es mucho más simple.

6.4 One-Class SVM

Como hemos explicado anteriormente, las máquinas de soporte vectorial son un típico ejemplo de machine learning. Se usan para diversas aplicaciones. En esta sección, vamos a usar una de estas aplicaciones, la de detectar outliers.

La idea de una SVM para detectar anomalías o outliers, es encontrar una función que sea positiva para las regiones con alta densidad de puntos y negativa para las densidades pequeñas.

Para realizar este modelo, vamos a usar una función que se llama OneClassSVM que está dentro del paquete de sklearn y que cuenta con los siguientes parámetros:

- **Nu=outliers_fraction**, es decir, ahí tenemos que poner un valor entre 0 y 1, que va a ser la proporción de outliers que vamos a encontrar. Nosotros vamos a poner una proporción de 0.01.
- También hay que usar **el kernel** que se usa en el algoritmo. Por defecto es rbf. Esto permite al SVM a usar una función no lineal para proyectar los datos con más dimensiones. A diferencia de la regresión logística, al usar kernels, las svm gestionan el aumento de dimensiones en el hiperespacio mucho mejor.
- **Gamma** es un parámetro del tipo kernel RBF que controla la influencia de las muestras individuales de entrenamiento. Esto consigue un efecto de suavizado en el modelo.

Por otra parte, para predecir los outliers después de entrenar el modelo, se usa la función **predict(data)** la cual realiza la clasificación de los datos. Nuestro modelo es un modelo de una clase porque devuelve 1 o -1, -1 si es un outlier y 1 si es normal, si pertenece a la clase.

El dataset sobre el que vamos a aplicar este método es STA.DE_DEBB065. Antes de aplicarlo, vamos a eliminar los valores que claramente son valores mágicos para ver mejor los resultados. Para ello, eliminaremos los valores negativos y lo sustituiremos por la interpolación de la media.

Lo primero que vamos a hacer es entrenar el modelo con las 4 series de contaminantes: NO_2, O_3, PM25 y PM10, sin ventanas temporales como en otras ocasiones. De esta manera, trataremos de encontrar outliers comunes a todas las series y que, por ello, se puedan detectar

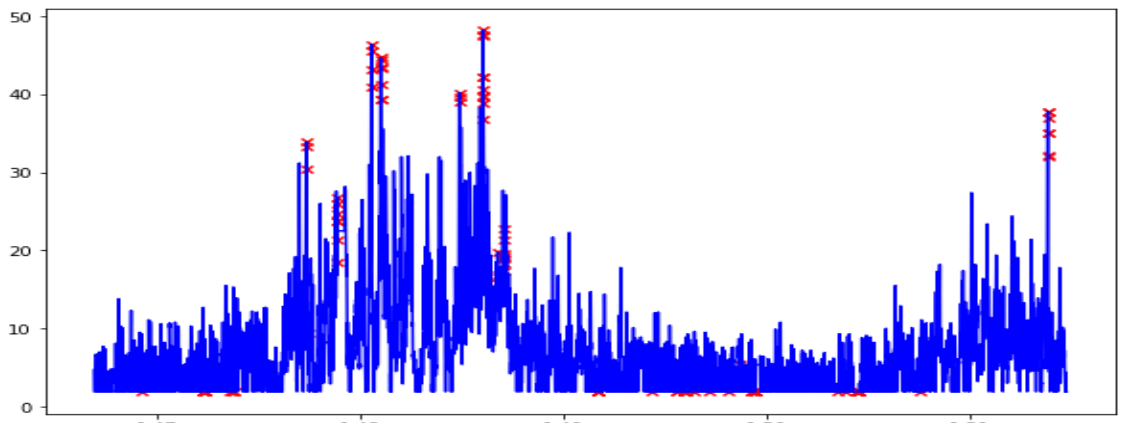


Figura 76: Detección de outliers con One-Class SVM en el NO2 con todas las series

de esta manera. En las Figuras 76, 77, 78 y 79 se pueden ver las series anteriormente mencionadas en el orden previamente expuesto.

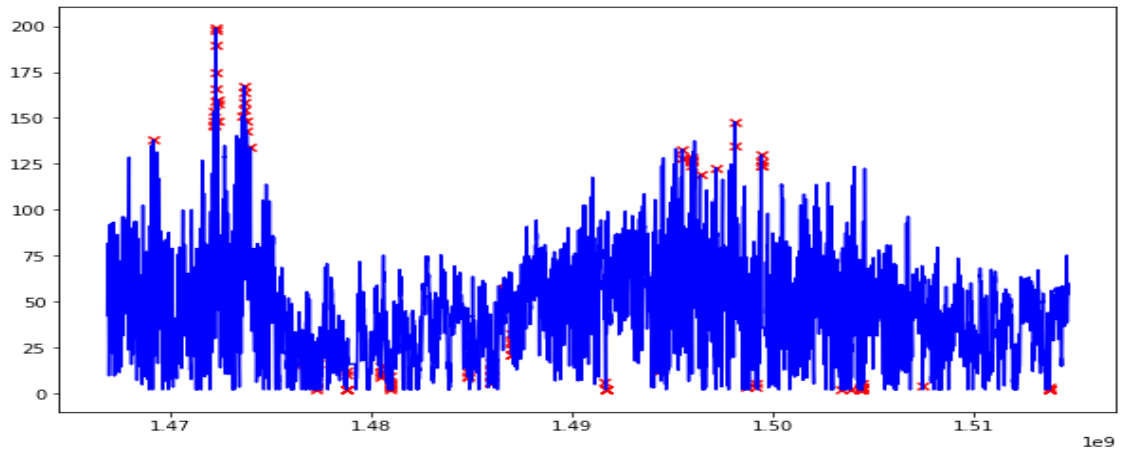


Figura 77: Detección de outliers con One-Class SVM en el O3 con todas las series

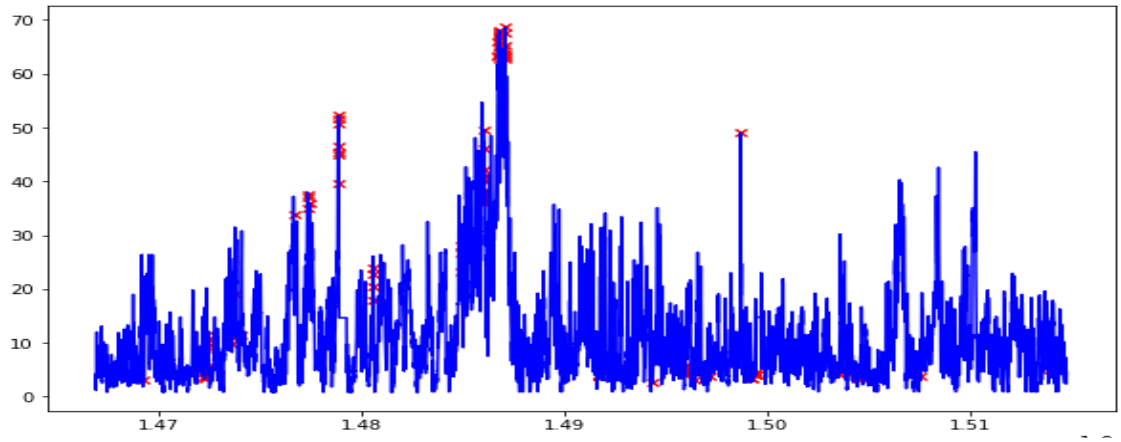


Figura 78: Detección de outliers con One-Class SVM en el PM25 con todas las series

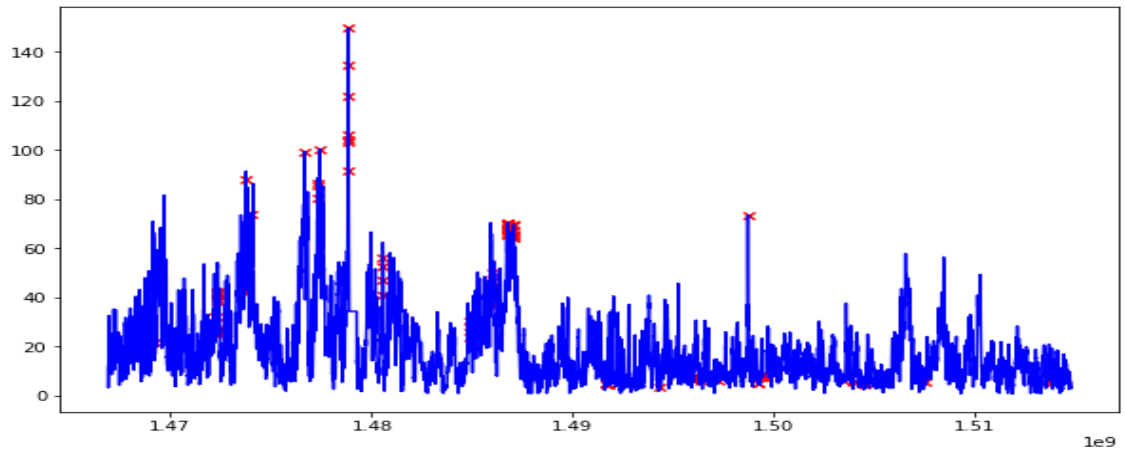


Figura 79: Detección de outliers con One-Class SVM en el PM10 con todas las series

A simple vista, parece que funciona muy bien. Si bien es cierto que hay outliers que no detecta o falsos positivos si vemos cada serie de manera independiente, sí que detecta de manera bastante precisa todos los outliers comunes a todas las series.

Viendo lo bien que funciona este método simplemente usando las series temporales conjuntamente de manera atemporal, comprobemos que tal funciona si usamos todas las series temporales de contaminantes a la vez que usamos ventanas temporales de 24 instantes.

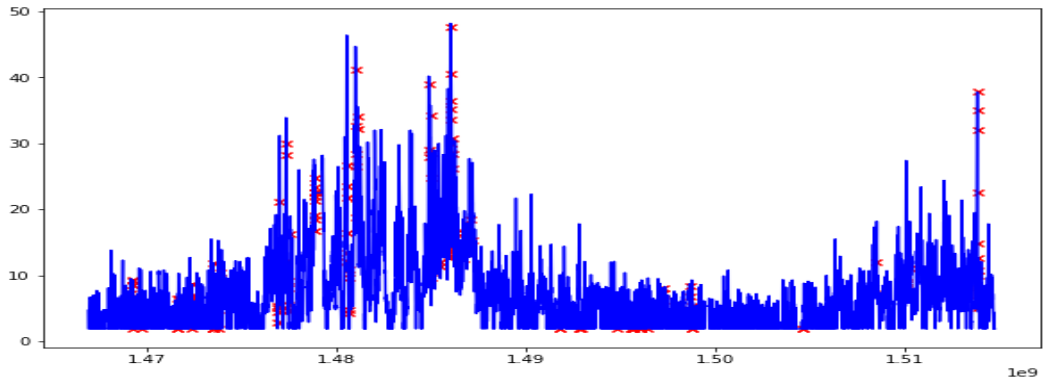


Figura 80: Detección de outliers con One-Class SVM en el NO2 con todas las series y ventanas temporales de 24

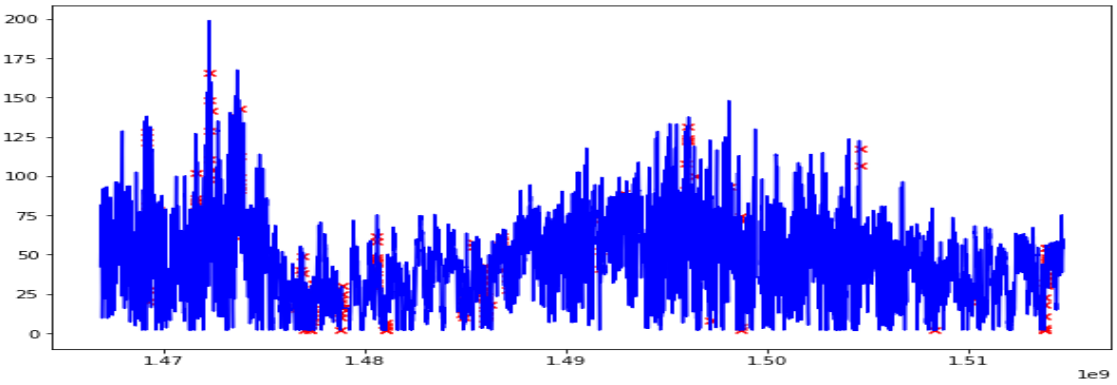


Figura 81: Detección de outliers con One-Class SVM en el O3 con todas las series y ventanas temporales de 24

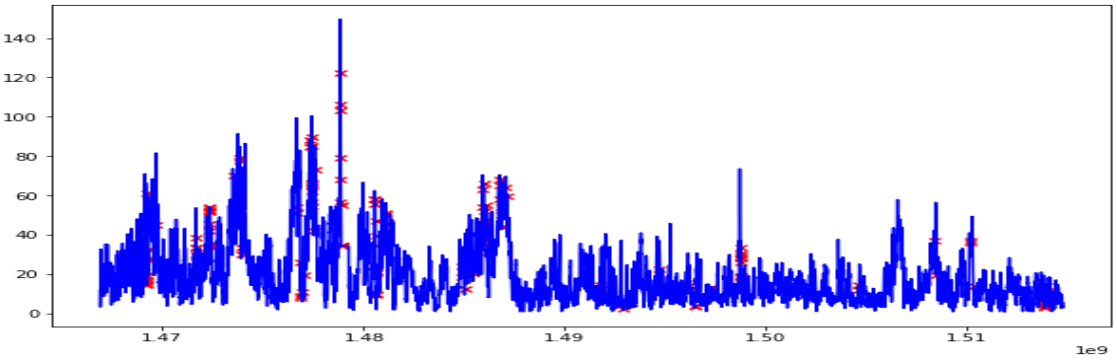


Figura 82: Detección de outliers con One-Class SVM en el PM10 con todas las series y ventanas temporales de 24

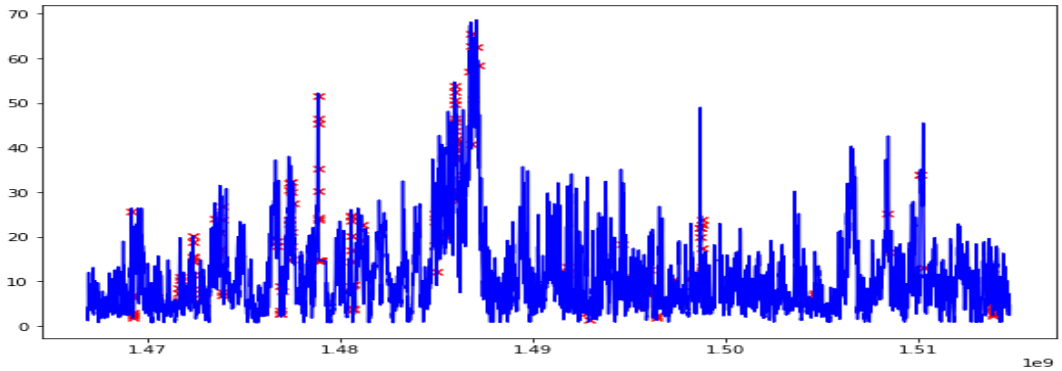


Figura 83: Detección de outliers con One-Class SVM en el PM25 con todas las series y ventanas temporales de 24

Como se puede observar de las Figuras 80 a la 83, ambas incluidas, es que usando ventanas temporales también se detectan correctamente los outliers, aunque no son los mismos que si no se usan ventanas temporales. De hecho, comparando ambas formas de detectar outliers con varias temporales simultáneamente, nos hemos dado cuenta de un par de cosas:

- Los outliers aislados, es decir, diseminados de manera aleatoria por la serie temporal, los detecta mejor si no usa ventanas temporales.
- Sin embargo, usando ventanas temporales nos damos cuenta de que hay zonas que no son outliers obvios, pero no podemos decir que no sean outliers, ya que no siguen el mismo patrón que zonas anteriores de la serie temporal.

A continuación, voy a usar el método de las ventanas temporales, de 5 valores concretamente, pero únicamente con una serie temporal a la vez. Para no alargar demasiado esta sección incluyendo demasiadas fotos, vamos a hacer pruebas únicamente con el NO_2. Como la proporción de outliers anteriormente era muy baja, vamos a subirla a 0.05 para realizar esta prueba. Se puede ver en la Figura 84.

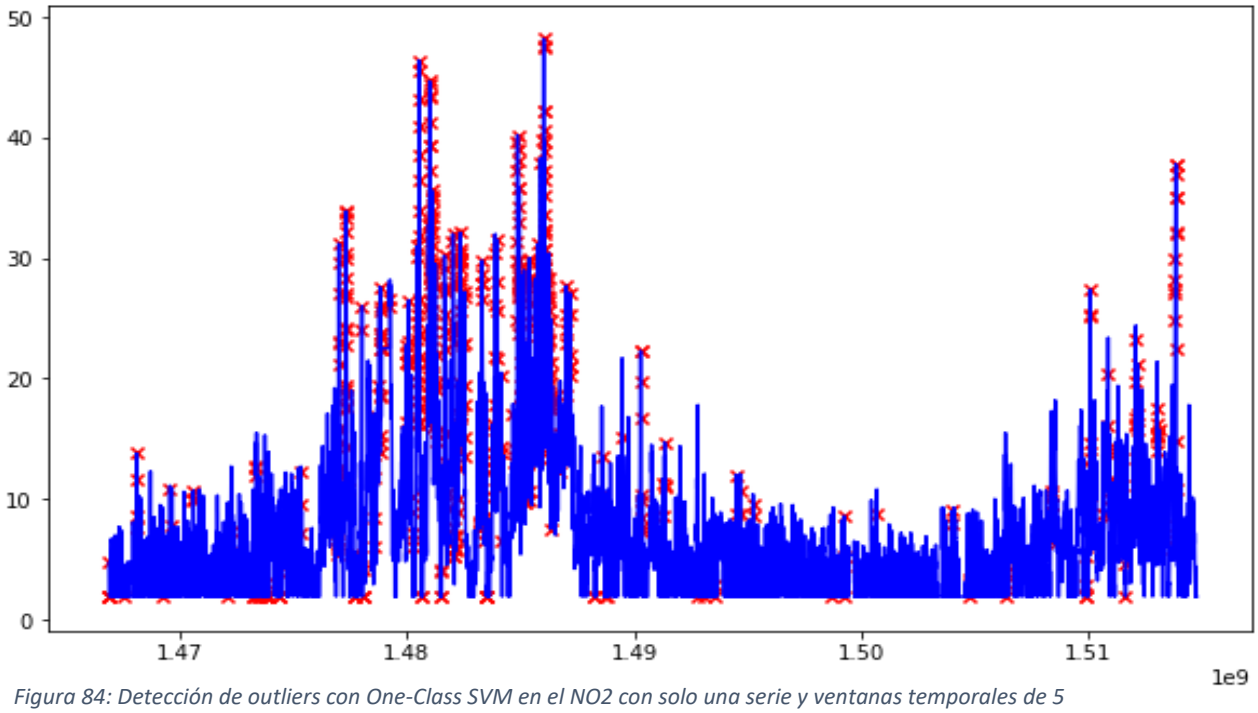


Figura 84: Detección de outliers con One-Class SVM en el NO2 con solo una serie y ventanas temporales de 5

Se puede ver en la Figura 84, que los outliers que detecta tienen mucho sentido. También se puede ver que quizás detecte demasiados outliers. Esto puede ser porque la proporción de outliers este sobreestimada y quizás habría que reducirla un poco.

De todas formas, en nuestra opinión y únicamente de manera visual, funciona mejor si se usan todas las series simultáneamente. Lo que podemos decir de esto es que este método es el que mejor usa la correlación entre las series temporales. Además, podemos decir que, para este método, la comparación entre todas las series temporales es mucho más útil que las ventanas temporales.

En conclusión, el método de One-Class SVM es uno de los mejores métodos para detectar outliers que hemos probado hasta ahora, por no decir el mejor. Detecta perfectamente de donde tiene que sacar los outliers y no maneja mal si se usan datos del pasado, usando ventanas temporales.

A pesar de esto, hay métodos como K-Means que usan mejor los datos del pasado, aunque hay que atribuirle a este método que es el que mejor usa la relación entre las series.

6.5 Conclusión

Todos los métodos que hemos visto en este capítulo de Machine Learning funcionan razonablemente bien en determinadas ocasiones. Sin embargo, hay tener claro que en ningún momento nadie ha definido que es un outlier de manera objetiva. Esto significa que todo lo que estamos diciendo es según nos parece de manera visual, pero no podemos obtener una medida objetiva para evaluar un método con respecto a otro.

Por otra parte, hay algunas cosas que nos hemos podido dar cuenta a lo largo de este capítulo que pueden ser interesantes a la hora de realizar un método final de detección de outliers:

- Hay algunos outliers que son comunes a todos los contaminantes. Estos pueden ser causados, por ejemplo, por una obstrucción del sensor, de un fallo de comunicación, por un pico de tensión, etc… Estos outliers los podemos detectar con métodos que usen todas las series temporales de contaminantes de manera simultánea. En este caso, el método que mejor ha funcionado ha sido el One-Class SVM, usando solamente los datos de los contaminantes, sin usar datos temporales y sin tener en cuenta las medidas meteorológicas.
- Por otra parte, también hay outliers que se pueden dar solo en un contaminante por una emisión extrema de un determinado contaminante en el sensor, por ejemplo, por el humo de una fábrica que por las corrientes de aire colisione directamente con el sensor o por un camión que ha estacionado justo debajo del sensor y echa todo el humo en él. En este caso, la mejor manera es usar detección de una única serie con ventanas temporales no muy grandes, para que no se desvirtué el método. En este caso, las mejores opciones son Isolation Tree o usar clustering de una determinada manera que hemos explicado anteriormente.
- Otra posible opción es usar todas las series temporales a la vez que se usan ventanas temporales. Sin embargo, este método no da buenos resultados experimentalmente así que recomendamos no utilizarlo en un método final de detección de outliers.

En el siguiente capítulo, usaremos Deep Learning y trataremos de definir que es un outlier usando la comparación entre los datos validados y los datos no validados.

7.Detección de outliers mediante Deep Learning

En este caso, vamos a usar redes neuronales recurrentes para tratar de detectar y reparar los outliers. Lo primero que vamos a hacer es cargar un dataset de UTD y eliminar los valores

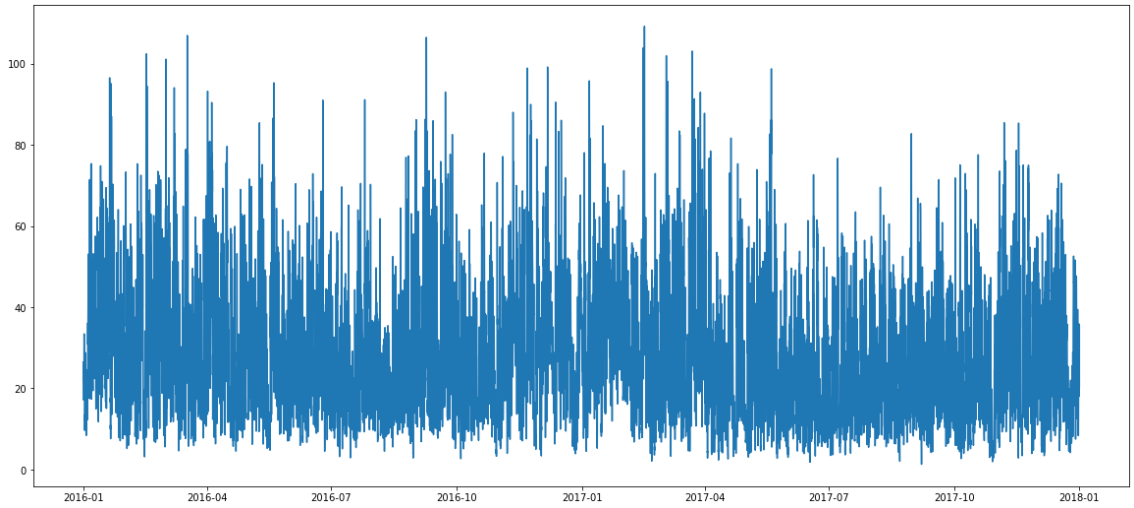


Figura 85: NO2 sin valores mágicos antes del Deep Learning

mágicos mediante la interpolación de la media. Como en otros casos, mostraremos todo el rato solo el NO2 para no alargar en exceso este trabajo y porque poner más contaminantes no aporta nada más, al ser el mismo proceso para todos ellos. La estación que estamos analizando es STA.DE_DEMV022. El NO2 sin valores mágicos podemos verlo en la Figura 85.

Para entrenar el modelo de Deep Learning, vamos a usar datos derivados de los datos que tenemos, además de usar propiamente los datos que tenemos. Esta técnica se denomina **aumentación**. Vamos a ir explicando poco a poco como obtenemos estas características. Después de obtenerlas, tendremos que obtener medidas para evaluar la calidad de los datos que tenemos y para reducir la dimensionalidad si es muy grande.

Los datos con los que vamos a aumentar los datos provienen del análisis técnico. El análisis técnico es una técnica que se usa en el análisis de bolsa para ayudar a los inversores a ganar más dinero. En el fondo, son técnicas de análisis de series temporales, así que podemos usarlas en este caso:

- **Media móvil:** Ya la hemos estudiado anteriormente en el análisis estadístico pero la traslado aquí brevemente para luego utilizarla.
- **Media móvil exponencial:** Es una media móvil ponderada que otorga más peso a los datos más recientes, mediante un método exponencial. Para calcularla solo hace falta usar un multiplicador y empezar con la media móvil normal.
- **Bandas de bollinger:** Simplemente usan la media móvil y la desviación para decir que todos los valores que superen una desviación son anomalías y no son predecibles ni normales.
- **Media móvil de convergencia/divergencia:** Tiene 3 componentes, pero nos vamos a quedar solo con el MACD. Es la diferencia entre 2 medias móviles exponenciales de

diferente longitud. El primero es cuánto de sensible es la serie en periodos de tiempo pequeños y el segundo es la sensibilidad en periodos medianos.

Voy a incluir el código de como he obtenido estas características para que se vea con más claridad. Se puede ver en la Figura 86.

```
def get_technical_indicators(dataset,c):
    # Media movil de 12 horas, 24 horas y 48 horas
    dataset['ma12'+c] = dataset[c].rolling(window=12).mean()
    dataset['ma24'+c] = dataset[c].rolling(window=24).mean()
    dataset['ma48'+c] = dataset[c].rolling(window=48).mean()

    # Create MACD
    dataset['26ema'+c] = dataset[c].ewm(span=26).mean()
    dataset['12ema'+c] = dataset[c].ewm(span=12).mean()
    dataset['MACD'+c] = (dataset['12ema'+c]-dataset['26ema'+c])

    # Create Bollinger Bands
    dataset['24sd'+c] = dataset[c].rolling(window=24).std()
    dataset['upper_band'+c] = dataset['ma24'+c] + (dataset['24sd'+c]*2)
    dataset['lower_band'+c] = dataset['ma24'+c] - (dataset['24sd'+c]*2)

    # Create Exponential moving average
    dataset['ema'+c] = dataset[c].ewm(com=0.5).mean()
```

Figura 86: Obtención de indicadores técnicos

Otra opción para aumentar, es utilizar el ARIMA que hemos usado en el análisis estadístico. Sin embargo, su computo es muy costoso y tampoco aporta gran cosa por lo que hemos visto así que no lo vamos a añadir.

Como ya hemos mencionado anteriormente, teniendo tantas características tenemos que considerar si todas ellas son realmente indicativas de la dirección que va a tomar el contaminante. Por ejemplo, hemos incluido los datos derivados del resto de contaminantes, pero no sabemos si realmente tienen algo que ver. Hay muchas maneras de comprobar la importancia de las características, pero la que vamos a aplicar usa XGBoost, porque da uno de los mejores resultados tanto en regresión como en clasificación.

El XGBoost consiste en un árbol de decisión que puede realizar regresión o clasificación, y nos permite ver cuáles han sido los parámetros que más usa para predecir, que es lo que queremos.

Así hemos probado para ver entre todas las características aumentadas junto con los parámetros iniciales, cuáles son las características más importantes. Como es obvio, no hemos usado las

características aumentadas del NO2, ya que al estar prediciendo el NO2, tienen tanta información de la serie a predecir que son las que va a usar el modelo.

El XGBoost en este caso lo entrenamos para un problema de regresión en el que a partir del resto de características tiene que predecir el valor del NO2, que es el caso que vamos a tener posteriormente con el Deep Learning.

Veamos pues la importancia de cada una de las características en la Figura 87.

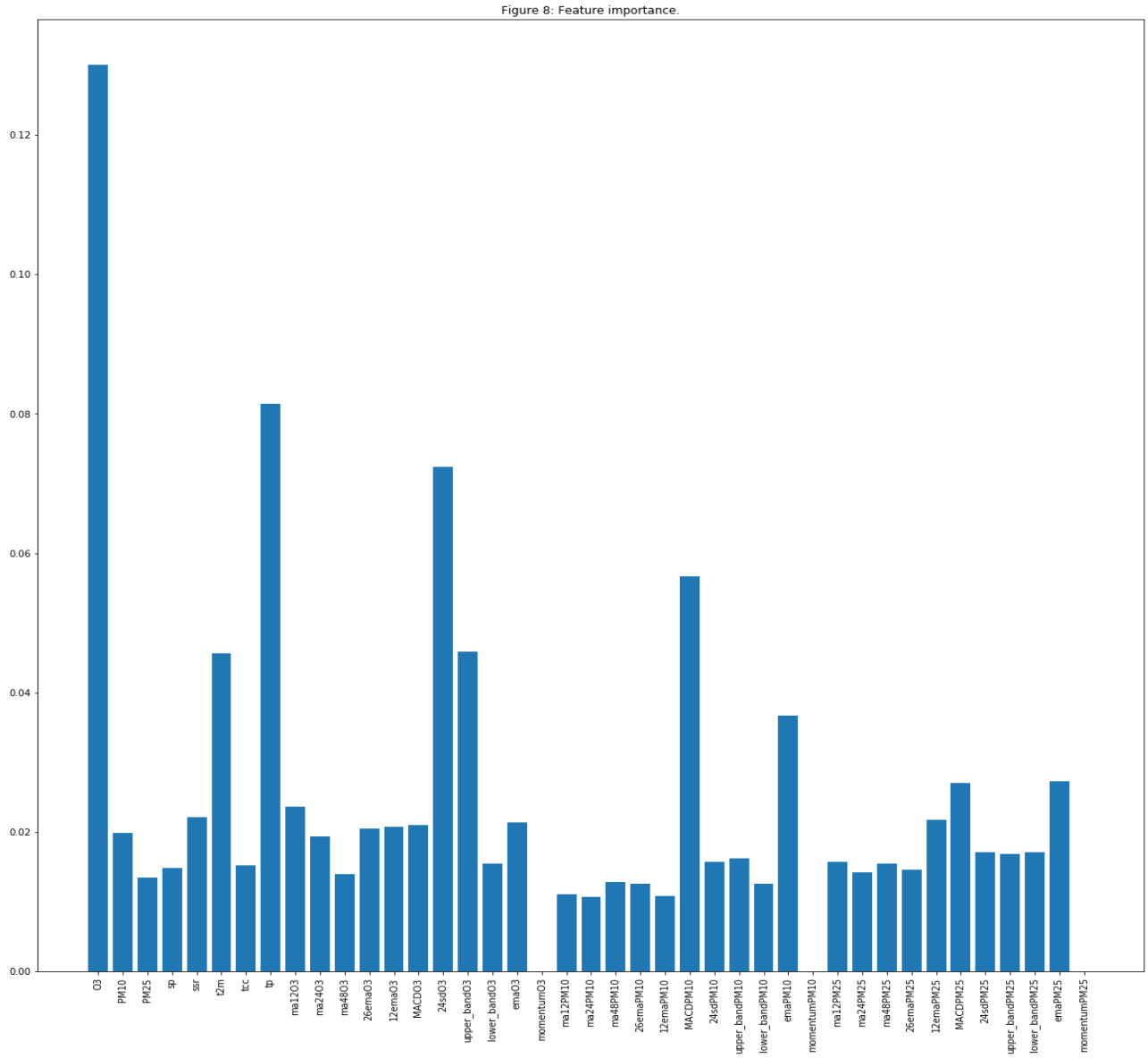


Figura 87: Ranking de características con XGBoost junto con las aumentadas

Como se puede ver en la Figura 87, el O3 es la característica que más está usando para predecir el NO2. Hay que decir que la suma de todas las proporciones de importancia de todas las características es 1, por eso el O3 solo tiene 0.12 de importancia.

Veamos de entre las características originales cuales son las más importantes en la Figura 88.

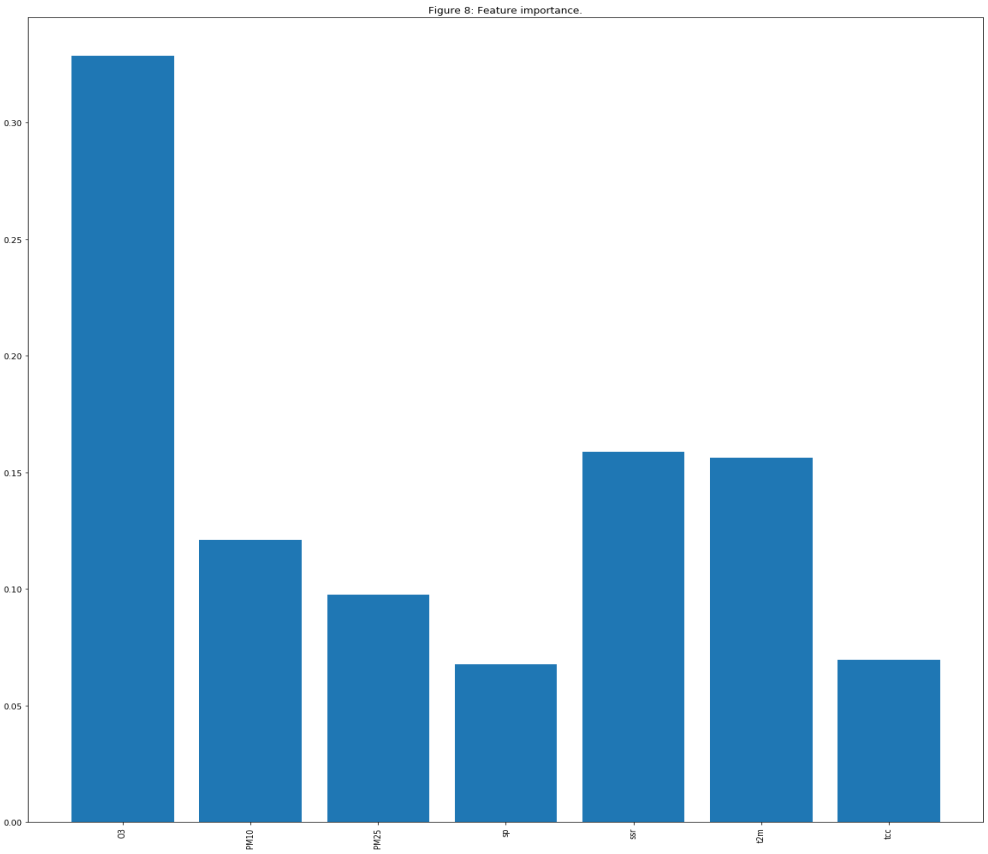


Figura 88: Ranking de características con XGBoost sin las aumentadas

En la Figura 88, vemos claramente que la característica más usada es el 03 seguida de dos de las características ambientales. Esto podemos interpretarlo como que el O3 es la característica más correlacionada con el NO2. A pesar de que vemos claramente que hay algunas características que son más importantes que otras, vamos a usar todas las características para predecir ya que el Deep Learning nos lo permite.

La técnica que vamos a usar con Deep Learning para detectar los outliers va a consistir en predecir la serie y luego ver las diferencias con la serie original. Si las diferencias son muy grandes, entonces diremos que hay un outlier en ese instante. En este caso, vamos a entrenar la red con los datos sin validar y posteriormente vamos a predecir también datos sin validar. Posteriormente, calcularemos la diferencia entre lo predicho y los datos originales y si la diferencia supera un umbral marcado por la desviación, declararemos ese valor concreto como outlier. Usaremos ventanas temporales de 100 valores porque el Deep Learning puede hacer buen uso de tanta información y además así evitamos el sobre entrenamiento.

La red que vamos a usar para este propósito va a ser una red compuesta por celdas LSTM. Vamos a adjuntar el código de construcción de la red para que se vea con más detalle cual ha sido la

implementación, aunque adelantamos que no tiene una gran complicación. Se puede ver dicha programación en la Figura 89.

```
def build_model(layers):
    d = 0.2
    init = glorot_uniform(seed = 69)

    model = Sequential()
    model.add(LSTM(100, input_shape=(layers[0], layers[1]), return_sequences=True, kernel_initializer = init))
    model.add(Dropout(d))
    model.add(LSTM(100))
    model.add(Dropout(d))
    model.add(Dense(100,kernel_initializer= init ,activation='linear'))
    model.add(Dense(4,kernel_initializer= init ,activation='linear'))
    model.compile(loss='mse',optimizer='adam',metrics=['accuracy'])
    return model
```

Figura 89: Modelo de Deep Learning

El error tanto en train como en test es muy bajo, de 0.03 RMSE para train y de 0.02 para test.

Train y Test son dos particiones diferentes que se realizan para que el modelo no sobre aprenda. Con Train se entrena el modelo y con Test se evalúa. Hemos hablado de esto anteriormente en los preliminares. RMSE (Root Mean Square Error) es la raíz cuadrada del error cuadrático medio.

Veamos cómo se ven las gráficas de Train y de Test predichas. Para ver con más detalle las diferencias, cogeremos fragmentos aumentados. Se puede observar Train en la Figura 90 y Test en la Figura 91.

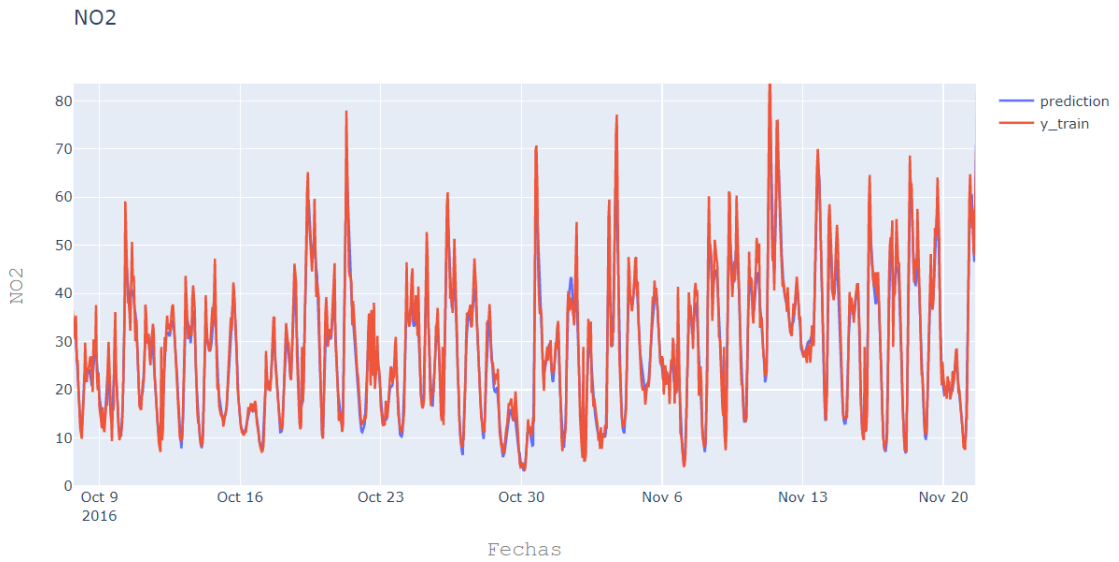


Figura 90: Train de predicción de Deep Learning con x e y de UTD

Como se puede ver en la Figura 90, las diferencias en train entre la predicción y los datos originales son muy pequeñas, pero existen.

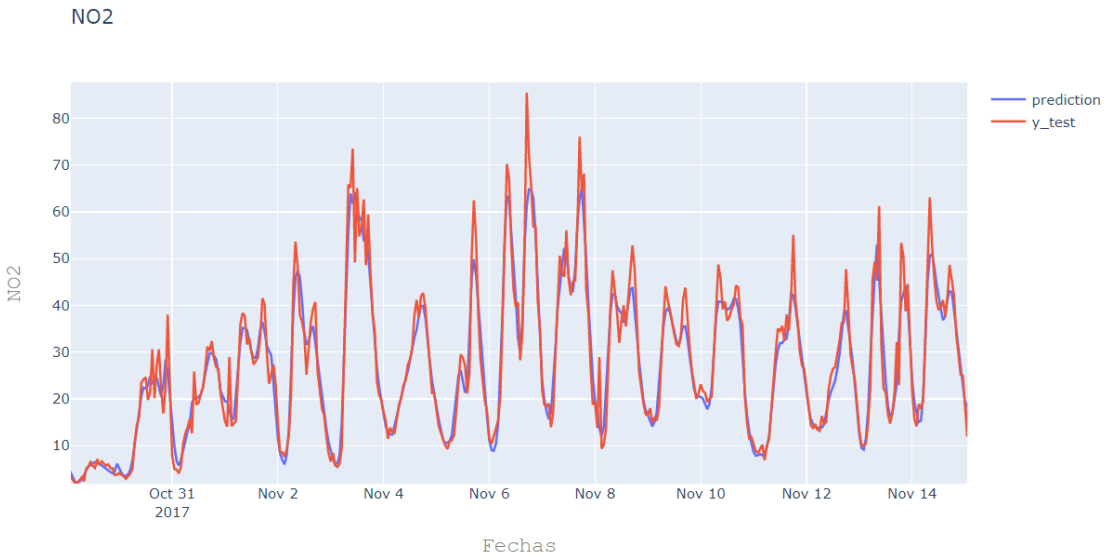


Figura 91: Test de predicción de Deep Learning con x e y de UTD

Como se puede ver en la Figura 91, las diferencias entre los datos originales y la predicción son más grandes que en el caso de train.

Como hemos dicho, podemos usar estas predicciones para detectar los outliers. Para ello, en primer lugar, predecimos los valores de train y test. En segundo lugar, calculamos las diferencias entre los datos reales y los datos predichos. En este segundo paso es muy importante normalizar los datos, para que luego la detección por medio de umbrales sea correcta. Por último, se detectan los outliers como los valores que superan un cierto umbral, en nuestro caso hemos puesto que el umbral es la desviación típica de las diferencias. Este umbral se puede observar en la Figura 92 las diferencias entre los datos originales y los datos predichos con los outliers marcados.

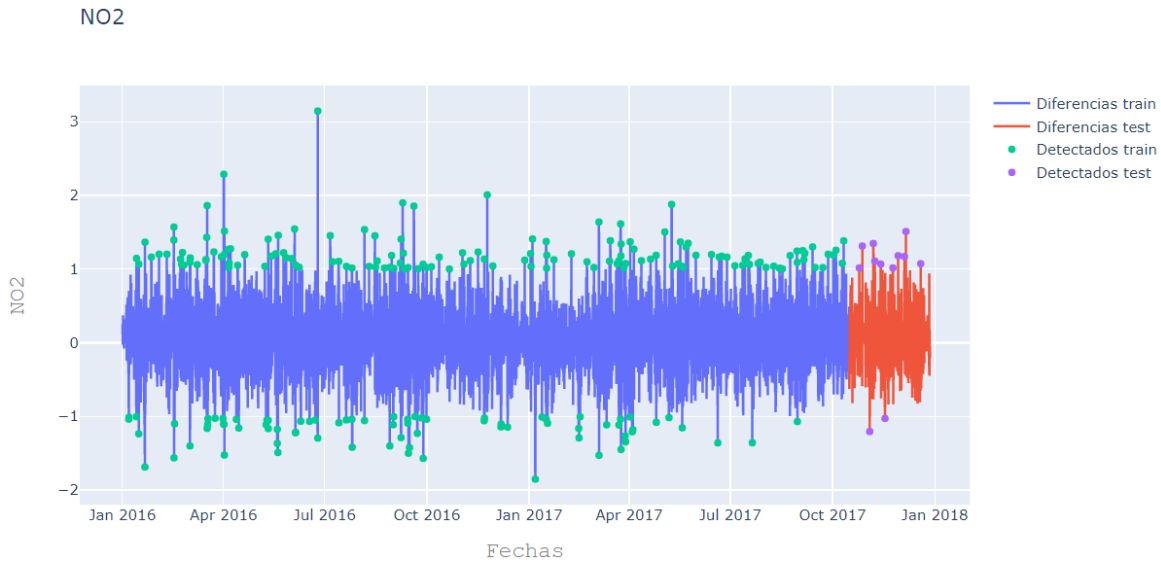


Figura 92: Diferencias entre lo predicho y los datos originales con x e y de UTD y con los outliers marcados

Como se puede ver en la Figura 92, al seguir los datos una distribución normal por estar normalizados, todos los valores que superan el valor 1 se detectan como outliers. Podríamos

cambiar el umbral a otro más elevado, pero creemos apropiado ponerlo así para detectar una mayor proporción de outliers. En la Figura 93, vemos la serie original con los outliers marcados.

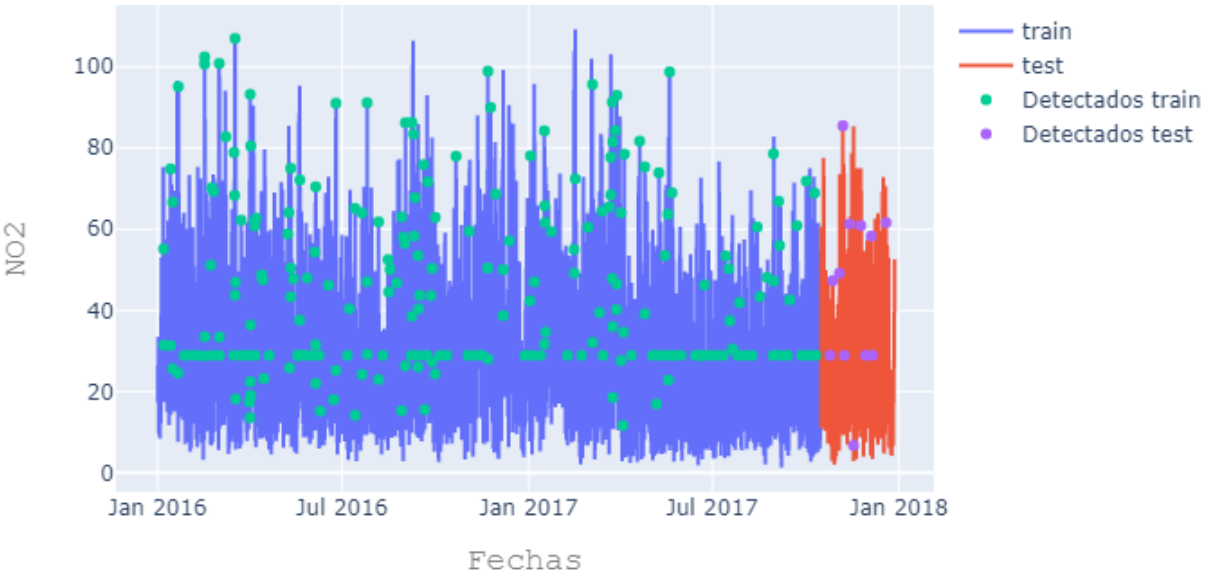


Figura 93: Outliers detectados con x e y de UTD sobre la serie original

Se puede ver en la Figura 93, que hay muchos outliers cuyo valor está más o menos sobre la media. Quizás este detectando la interpolación de la media que hemos realizado antes, detectando todos estos valores como outliers, porque al final estos valores no son ciertos así que están bien detectados. Por lo demás, los outliers parecen bien detectados.

Posteriormente a esta detección, los outliers se pueden corregir reemplazando su valor por el valor predicho por la red. Pintamos con puntos verdes y morados, los valores actuales de los puntos que previamente eran outliers. Se puede ver esto en la Figura 94.

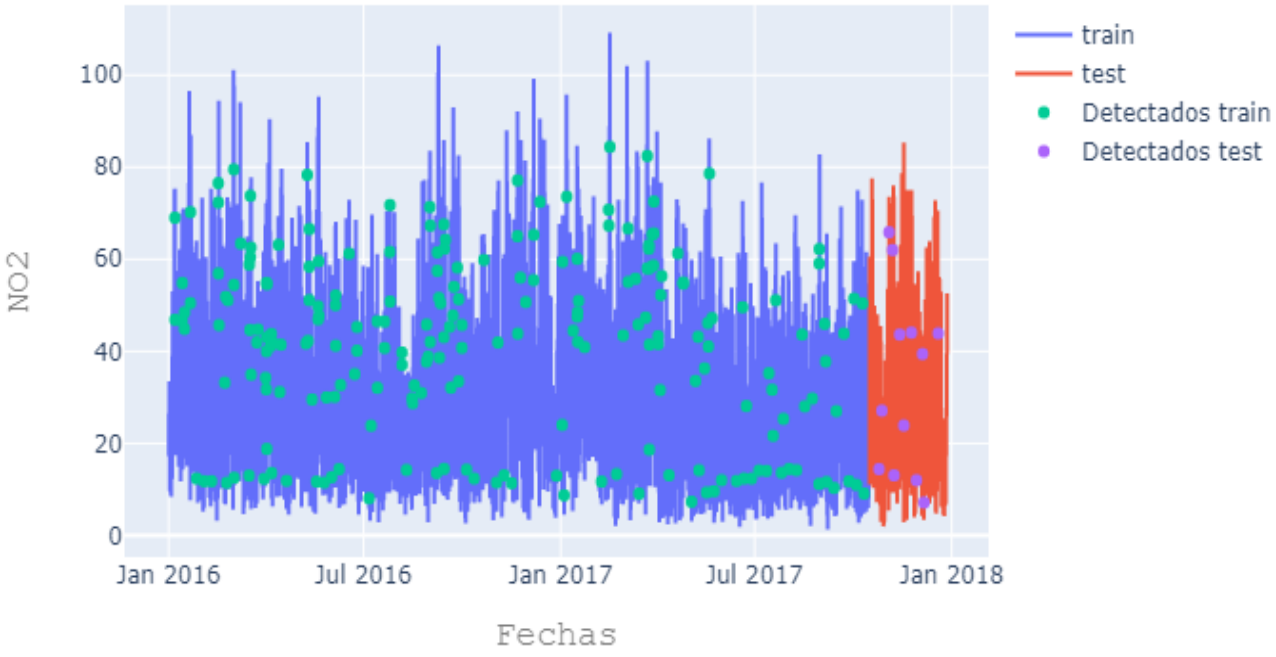


Figura 94: Outliers corregidos con x e y de UTD sobre la serie original

En la Figura 94, se ve cómo se han solucionado los outliers que habían sido detectados mediante el método del umbral. De esta manera, la serie parece mucho más natural que anteriormente, si se compara con la Figura 93.

Además de esa técnica, se nos ha ocurrido otra. Esta consiste en coger los datos no solo del dataset de datos no validados, sino que se pueden coger de los datos también validados. Es decir, hacemos que la entrada de la red neuronal sean los datos no validados pero que la salida tenga que ser la de los datos validados. Todo lo demás es exactamente igual que en el caso anterior. En la Figura 95 se muestra un fragmento del resultado en train.

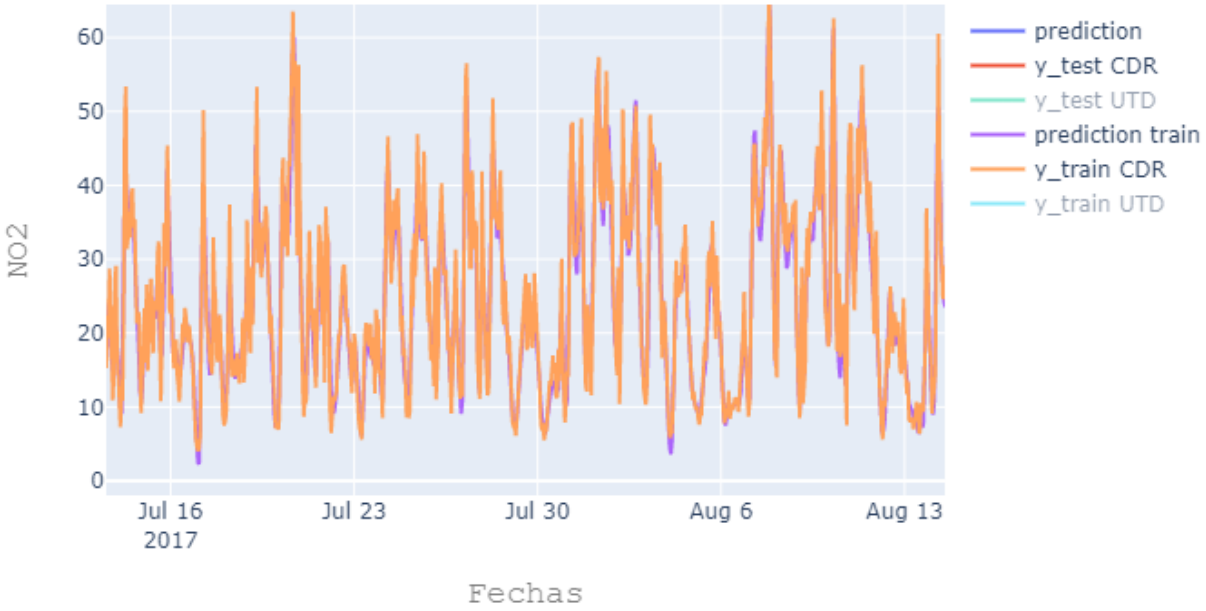


Figura 95: Train de predicción de Deep Learning con x de UTD e y de CDR

Los resultados, como se puede ver en la Figura 95, es muy similar al caso en el que cogíamos los datos de entrada y de salida del mismo dataset de datos sin validar. Esto es porque los datos han sido corregidos muy poco o han sido corregido siempre de la misma manera. Veamos en la Figura 96 el resultado para test.

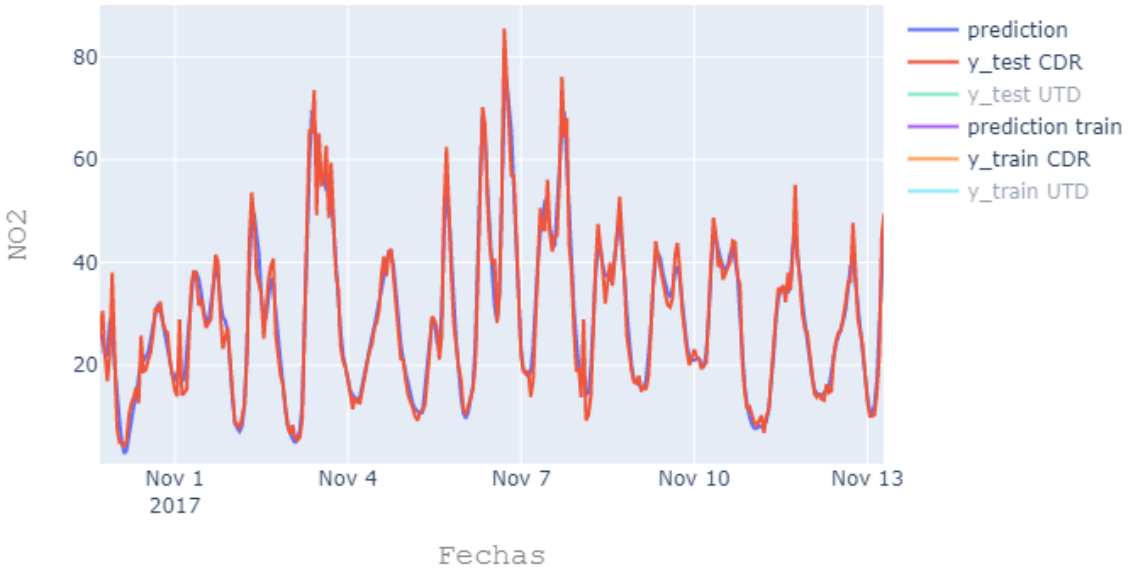


Figura 96: Test de predicción de Deep Learning con x de UTD e y de CDR

Como se puede observar en la Figura 96, estos resultados se asemejan mucho a los resultados que obteníamos en la Figura 91. Es decir, no ganamos nada de obtener los datos de distintos datasets.

Veamos ahora como se detectan los outliers. Para ello vamos a utilizar los mismos pasos que hemos mencionado anteriormente, así que no los vamos a volver a explicar y pasamos directamente a los resultados. Podemos ver las diferencias entre los datos predichos y los datos originales en la Figura 97.

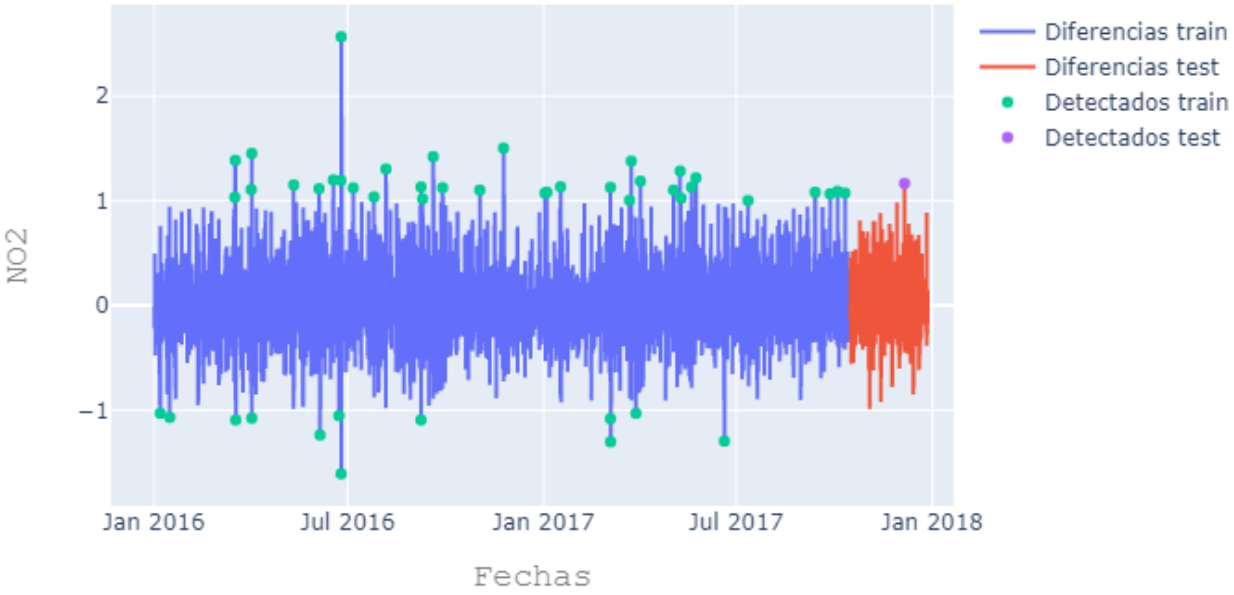


Figura 97: Diferencias entre lo predicho y los datos originales con x de UTD e y de CDR y con los outliers marcados

Como podemos ver en la Figura 97 y como llevamos explicando unas cuantas veces, los resultados entre la técnica anterior y esta son muy similares. Se puede observar comparando la Figura 97 con la Figura 92. Sí que es verdad que algunos outliers que estaban en la frontera de ser considerados como tales, en este caso sí que son considerados valores anómalos. O viceversa. Pero en la gran mayoría de los casos ha permanecido inmutable.

Veamos ahora en la Figura 98 estos outliers marcados en la serie original, para así observar que representan estos outliers en la realidad.

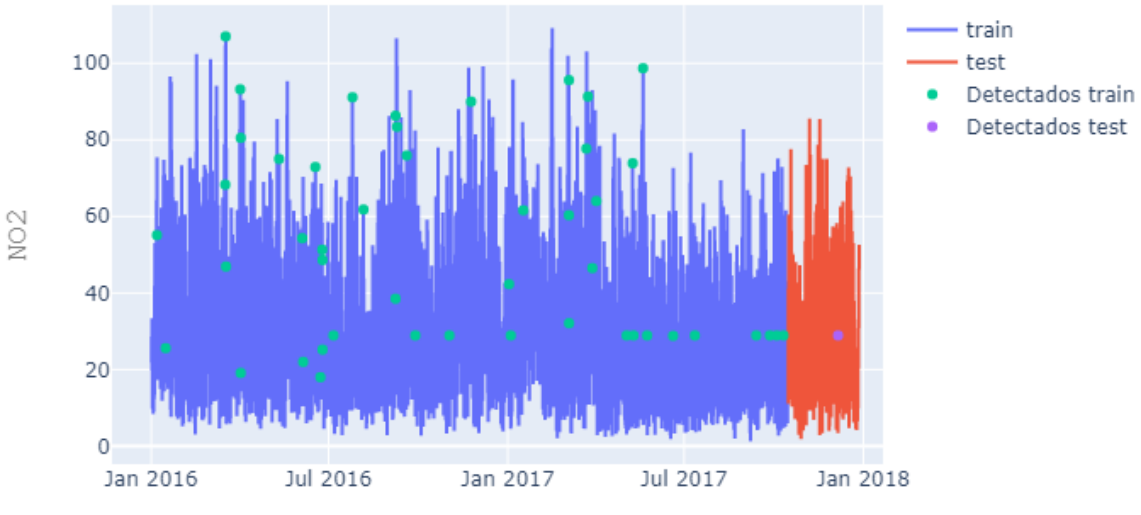


Figura 98: Serie original con los outliers marcados cogiendo x de UTD e y de CDR

Como se puede observar en la Figura 98, sí que se detectan menos valores que están por donde la media, pero la distribución de los datos detectados como outliers más o menos es igual que en el caso anterior, aunque se detectan algunos menos.

Como en el caso anterior, aquí también vamos a corregir los datos sustituyéndolos por los valores predichos por la red. Se puede ver esto en la Figura 99.

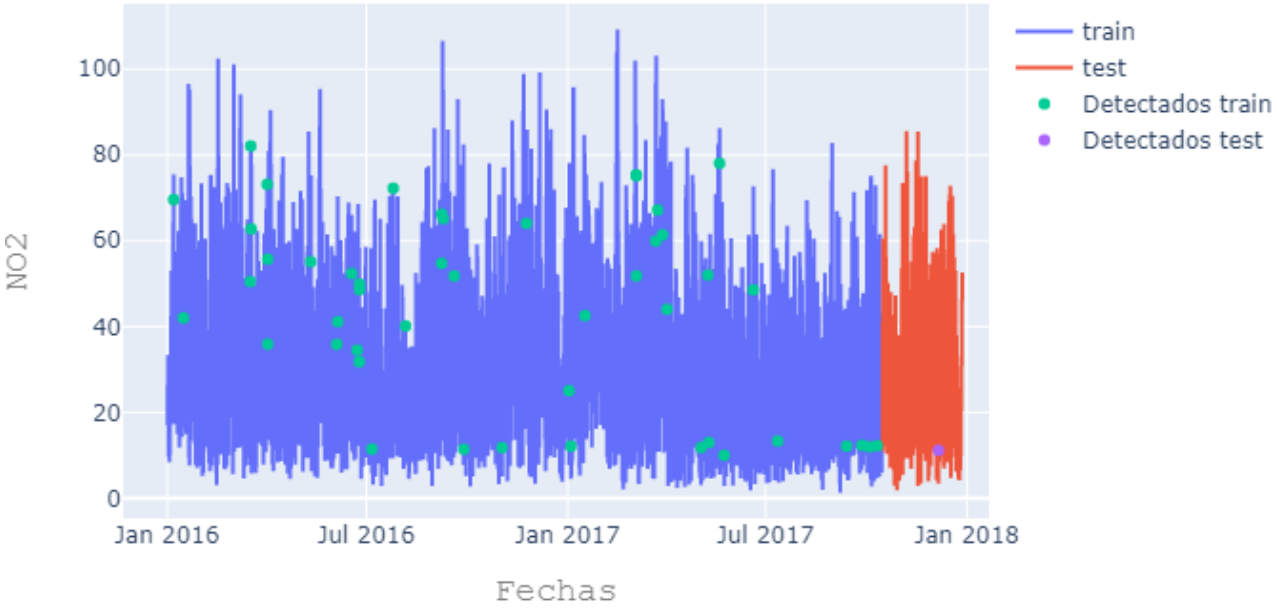


Figura 99: Serie corregida con los outliers marcados cogiendo x de UTD e y de CDR

Como se puede ver en la Figura 99, los resultados son muy buenos como en el caso anterior.

Como hemos podido ver, los resultados del Deep Learning han superado con creces los resultados del Machine Learning. Dentro de este capítulo también hemos podido ver que si cogemos los datos de x y de y de los datos no validados funciona muy bien y no hace falta hacer ninguna mezcla más.

Para comprobar si este método es general o es solo el caso de esta estación, vamos a probar con alguna estación más. En este caso, vamos a probar con una estación que se encuentra en España y que se llama "STA_ES1529A" y los datos pertenecen a la categoría UTD, como en el caso anterior. La serie la hemos elegido al azar y hemos eliminado los valores mágicos reemplazándolos por la media de la serie. Se puede ver el NO_2 en la Figura 100.

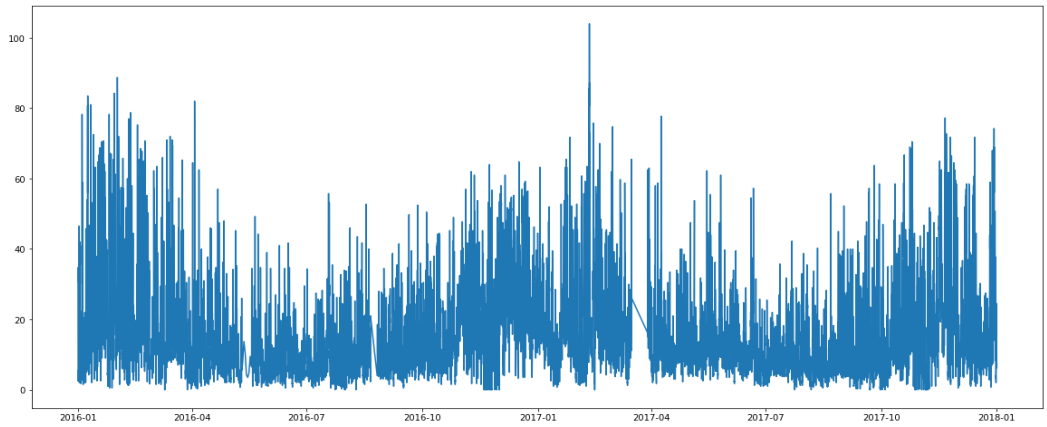


Figura 100: NO2 sin outliers de la serie STA_ES1529A de UTD

En este caso, además de hacer otra prueba para ver si funciona bien en otra serie la detección de outliers, no vamos a incluir los datos aumentados que en el caso anterior hemos usado. Esto lo vamos a realizar para ver si nos podemos ahorrar eso, ya que si no sirven para nada al final es una pérdida de tiempo computacional.

Lo primero que hacemos antes de introducir los datos en la red es normalizarlos, algo que también hemos hecho en el caso anterior, aunque no lo hayamos mencionado. Si se meten datos sin normalizar dentro de la red neuronal, puede que funcione mal ya que están diseñadas para funcionar con valores entre 0 y 1. La normalización que vamos a llevar a cabo es una normalización max-min.

En este caso, vamos a usar una página web que nos proporciona Google que se llama *google collaboratory* para entrenar la red neuronal. Esto lo hacemos porque tiene un acelerador de redes neuronales mediante tarjeta gráfica. Esto hace que cada época del entrenamiento que antes realizábamos en nuestro ordenador con CPU en 1 minuto, ahora lo hacemos en 4 segundos.

En primer lugar, se pude ver en la Figura 101 el train predicho junto a los valores reales de la nueva serie que estamos evaluando.

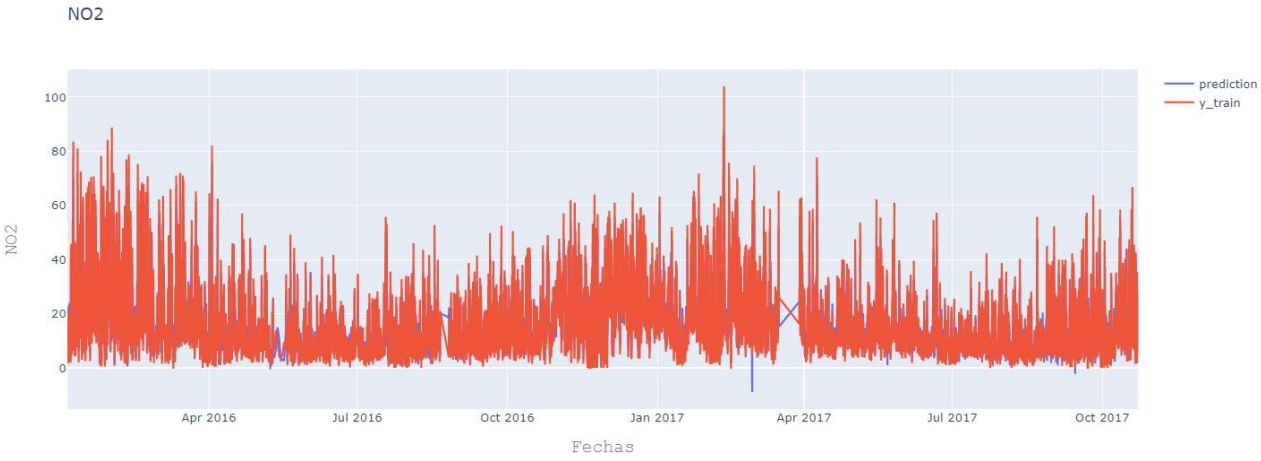


Figura 101: NO2 predicho de train junto con datos originales de la serie STA_ES1529A de UTD

Como se puede ver, los datos originales tapan a la predicción porque prácticamente son valores iguales. En la Figura 102 se puede ver una zona aumentada de la Figura 101, para observar mejor los valores predichos. En el caso de la Figura 102, he cambiado los colores respecto a la Figura 101 para que los valores predichos estén por encima de los valores originales.

Se puede observar en la Figura 102, que la serie predicha es una copia de la serie original pero más suavizada. De alguna forma, la red neuronal logra eliminar parte del residuo de la serie al tener en cuenta solo los valores pasados para averiguar el valor actual.

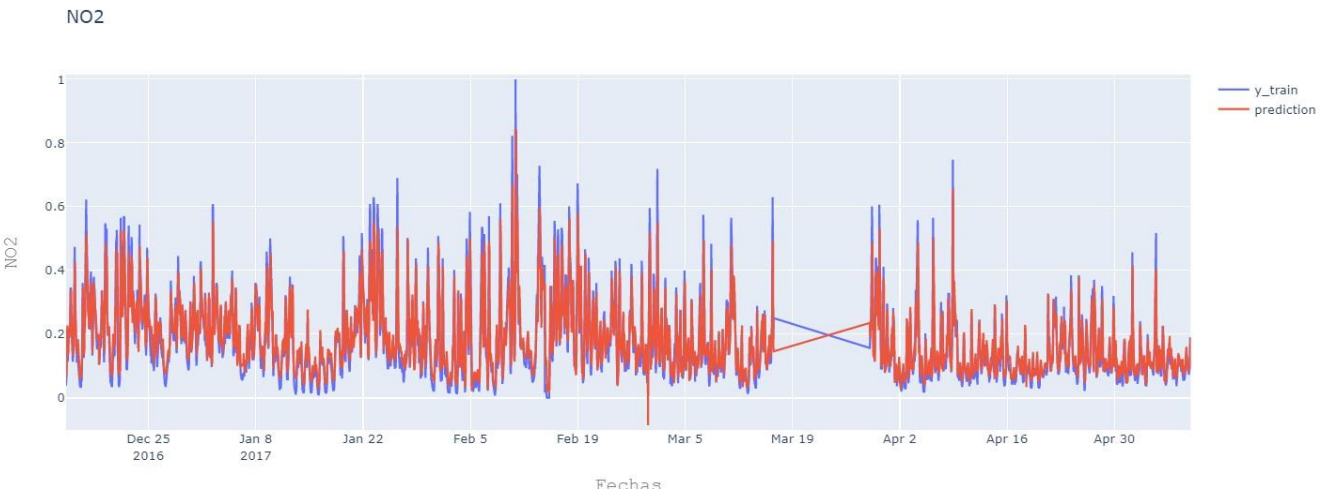


Figura 102: Trozo del NO2 predicho de train junto con datos originales de la serie STA_ES1529A de UTD

Como hemos dicho en los preliminares, con ver los resultados de train no es suficiente, también hay que observar los valores de test. Para observar estos valores, se puede ver la Figura 103.

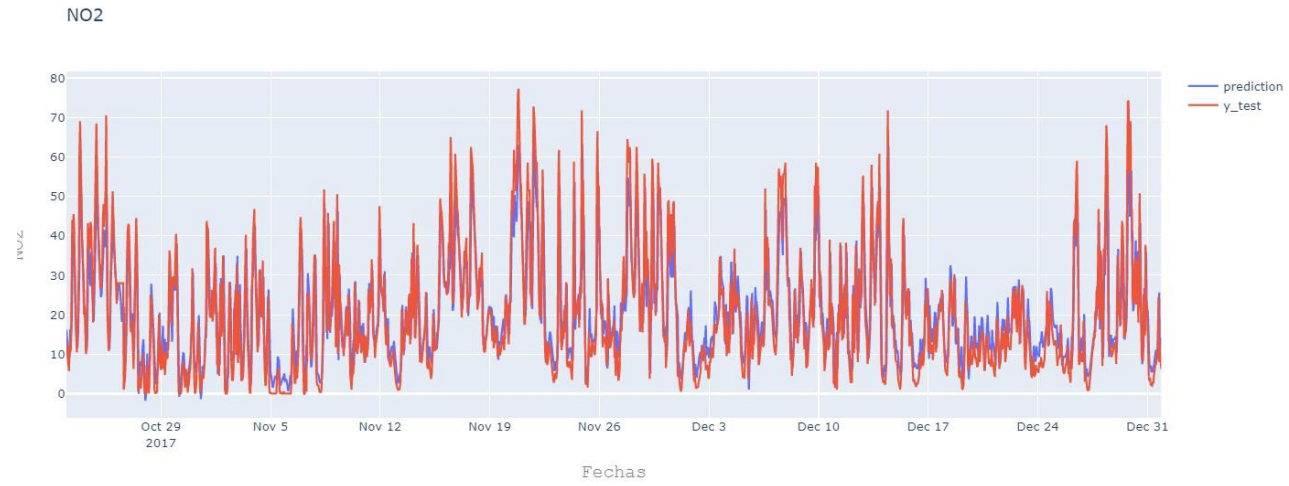


Figura 103: NO2 predicho de train junto con datos originales de la serie STA_ES1529A de UTD

En el caso de la Figura 103, se aprecia suficientemente bien la diferencia entre los datos predichos y los datos originales así que no vemos necesario extraer un trozo. Más o menos se comporta de igual manera que en train así que generaliza bastante bien.

Ahora vamos a utilizar la misma técnica de la desviación que hemos usado en el caso anterior para detectar los outliers. En primer lugar, restamos la serie predicha normalizada a la serie original normalizada. De esta manera, obtenemos una serie de diferencias que sigue una distribución normal. A esta serie le aplicamos un umbral de dos desviaciones típicas y detectamos todos los outliers que superen el umbral por encima o por debajo. Veamos el resultado en la Figura 104. Hay un total de 5.68% de outliers si seguimos este criterio.

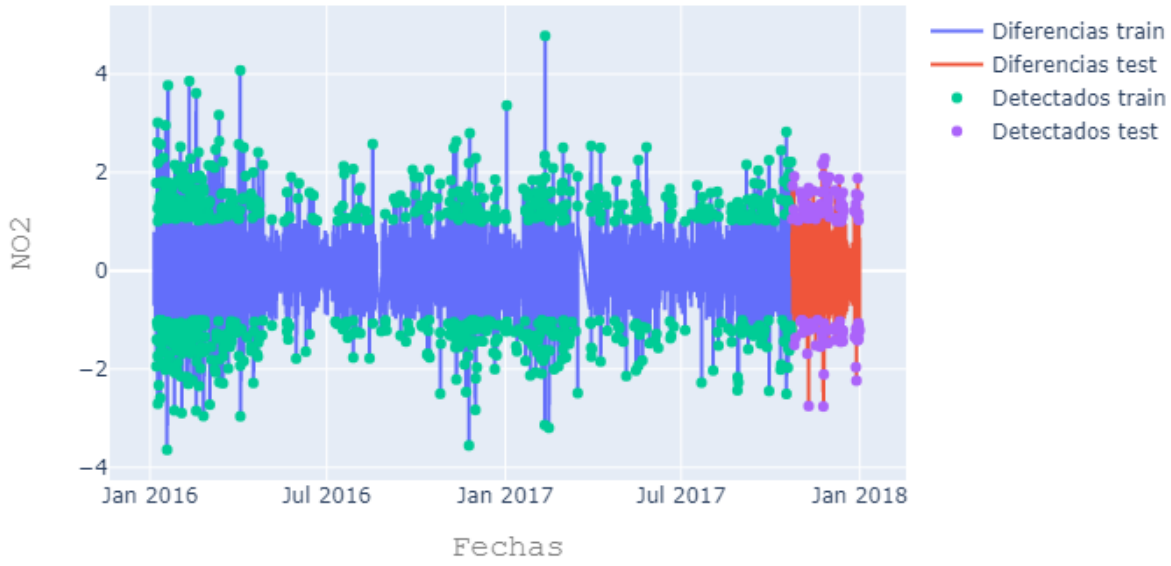


Figura 104: Diferencia del NO2 de la serie STA_ES1529A de UTD

Cómo se puede ver en la Figura 104, hay más outliers que en el caso anterior. Ahora veamos cómo se ven estos outliers en la serie original en la Figura 105.

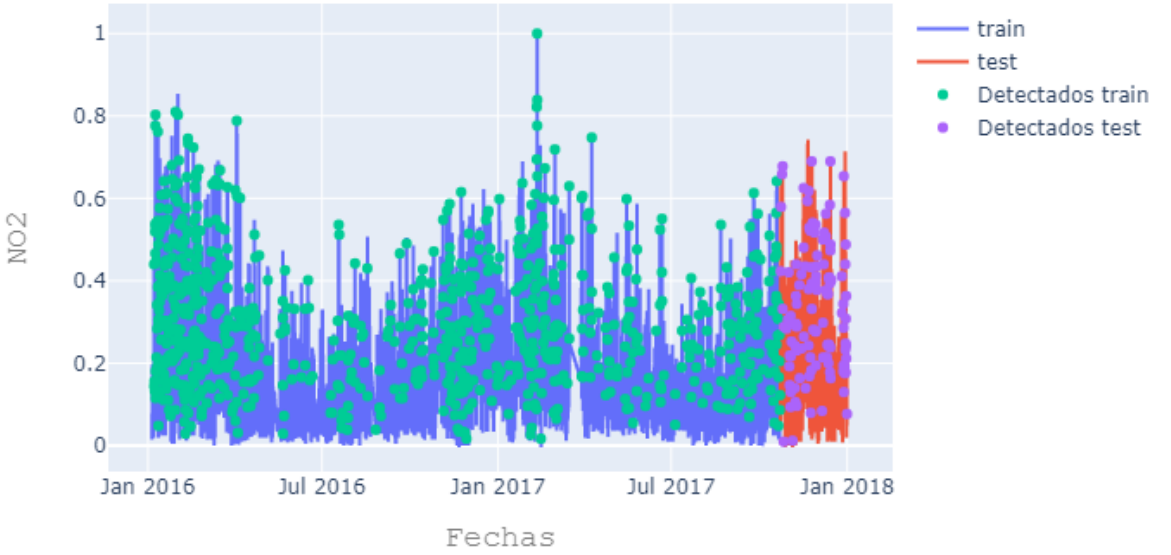


Figura 105: Serie original del NO2 con los outliers detectados de la serie STA_ES1529A de UTD

Vista la Figura 105, se puede ver que la cantidad de outliers que detecta es mayor que en el caso anterior como hemos remarcado antes. No obstante, por el tamaño de los puntos parece que hay más outliers de los que realmente hay. Realmente, hay menos de un 6% de outliers como hemos dicho anteriormente.

Como hemos hecho en el caso anterior, vamos a tratar de solucionar estos outliers sustituyéndolos por sus valores predichos. Se puede ver en la Figura 106 la serie del NO2 con los outliers corregidos.

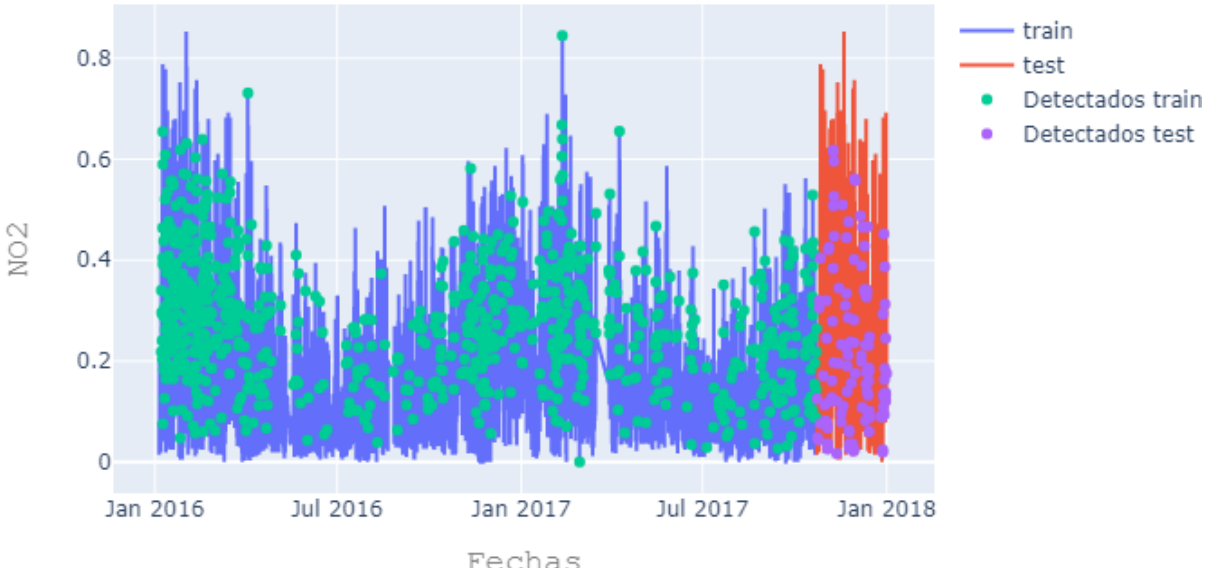


Figura 106: Serie corregida del NO2 con los outliers detectados de la serie STA_ES1529A de UTD

Como se puede observar entre la Figura 105 y la Figura 106, ha habido muchos outliers que se han solucionado y otros que se han suavizado. Por ejemplo, el pico que en la Figura 115 medía 1 y en este caso mide un poco más de 0.8. La escala de ambas Figuras esta entre 0 y 1 porque no hemos realizado el proceso inverso de la escala, ya que así se veía más claro.

También se puede ver entre ambas Figuras que el número de picos ha disminuido de la Figura 105 a la Figura 106, lo que puede ser indicativo de reducción de outliers.

Como vemos, este método funciona bastante bien a la hora de detectar los outliers. Hemos probado dos series radicalmente distintas y se comporta correctamente. Sin embargo, quizás el método de detección de outliers es demasiado simple. Para líneas futuras se podría aplicar alguna de las técnicas de machine learning que hemos visto en el capítulo anterior sobre las diferencias entre ambas series para detectar mejor los outliers. Por ejemplo, se podría usar un Isolation Forest con solo una serie y ventanas temporales de 3 momentos para ver si mejora los resultados. Además, con el Isolation Forest se puede ajustar el porcentaje de outliers, lo que nos permitiría ajustarlo sin tener que jugar con la desviación típica de la muestra, además de poder ahorrarnos la normalización de las series.

Siguiendo con el tema de detección de outliers mediante las diferencias de lo predicho y la serie original, también se podría usar clustering con ventanas temporales sobre cada una de las series para detectar estas anomalías y luego posteriormente corregirlas.

Con respecto al tamaño de la red neuronal y con la ventana temporal usada, hemos probado con diferentes tamaños y el mejor resultado se ha dado con este. No hemos expuesto estos resultados por no alargar de manera innecesaria este trabajo de fin de grado.

8.Conclusión y líneas futuras

Como hemos podido ver a lo largo de este trabajo, hay muchos métodos de detección de outliers que son dignos de mencionar. Debido a la duración limitada de este trabajo no hemos podido explorar todos y cada uno de estos métodos, pero hemos explorado aquellos que a nuestro parecer nos han resultado más interesantes.

Creemos que el mejor resultado que hemos obtenido ha sido gracias al Deep Learning y ese sería el resultado con el que tendríamos que quedarnos para este proyecto europeo por el cual hemos hecho este trabajo de fin de grado. Sin embargo, no tenemos ninguna métrica para valorar cómo de buenos han sido los resultados por lo que decidir entre un modelo u otro se queda en la subjetividad.

En el tema del análisis estadístico, podríamos probar otros métodos, como puede ser el filtro de Hampel o el filtro de Kalman, para ver si dan mejores resultados a la hora de analizar los outliers. Sin embargo, no creemos que sea necesario ya que el análisis que hemos realizado nos parece lo suficientemente extenso como para no explorar nuevas técnicas de índole estadística.

Por otra parte, en el tema de machine learning probablemente nos queden algunos ámbitos de detección de outliers que no hemos explorado. Por ejemplo, un modelo llamado gaussian distribution que funciona muy bien para valores que siguen una distribución normal. Este modelo consiste en ajustar una serie de parámetros a una distribución normal multivariable, y aquellos puntos que no se ajusten a esta distribución o que menos se ajustan son detectados como anomalías. A lo largo de este trabajo de fin de grado hemos visto que los valores sí que siguen en gran medida una distribución normal así que quizás sería una buena opción aplicarlo.

También puede que otra línea de investigación dentro del machine learning pueda ser usar KNN (K Nearest Neighbours) para detectar los outliers. En este caso, el método se basa en detectar que puntos están más cerca o más lejos de los vecinos más cercanos. De esta manera, los vecinos que estén más lejanos podrían ser outliers.

Continuando con líneas futuras, en Deep Learning hay un gran abanico de opciones. De hecho, una posible opción que está aún en fase de investigación es usar un GAN para detectar estos outliers. Un GAN (Generative Adversarial Networks) es un tipo de red neuronal que cuenta con un generador y un discriminador. Al entrenar la red, entrenas al generador para que cree muestras lo más similares a los datos que le estas pasando y a su vez, entrenas al discriminador para diferenciar los datos reales de los datos generados.

De esta manera, cuando el discriminador considere que un dato original no es original podemos obtener un posible outlier. Este modelo es muy complejo, pero puede que de buenos resultados. Se puede observar en la Figura 107.

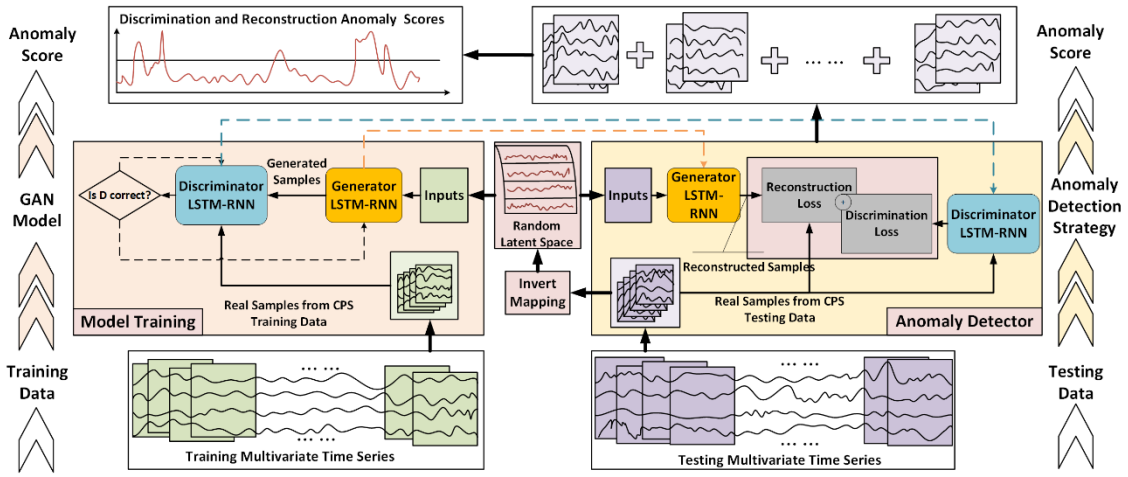


Figura 107: GAN para detección de outliers. Fuente: https://www.groundai.com/project/mad-gan-multivariate-anomaly-detection-for-time-series-data-with-generative-adversarial-networks/1

Como se puede ver en la Figura 107, esta red devuelve un valor por cada valor de la serie temporal que simboliza como de anómalo es cada valor medido de la serie. Sin duda una buena línea de investigación por donde seguir.

En el contexto real de la aplicación de este trabajo fin de grado, también habría que preocuparse por la implementación de estos modelos. Para ello, habría que optimizar al máximo el rendimiento. Por ejemplo, habría que hacer que el entrenamiento y la predicción de las redes neuronales profundas fuese más rápido y en vez de costar minutos, costase segundo. Para conseguir este objetivo, alguna de las opciones podría ser sustituir las LSTM por celdas GRU (Gated recurrents units), que al tener menos parámetros para entrenar son más rápidas y cumplen una función parecida. Sí que es verdad que habría que comprobar su rendimiento, pero puede que sea una buena alternativa.

Otra posible opción para optimizar el rendimiento podría ser utilizar tarjetas gráficas para entrenar los modelos de Deep Learning, en vez utilizar CPUs. Esto es porque las tarjetas gráficas consisten en CPUs especializadas en hacer operaciones con matrices, que al final es en lo que se basa las redes neuronales, en operaciones con tensores que en el fondo son matrices con otro nombre.

Para concluir este trabajo de fin de grado, decir que hemos aprendido mucho con él y creemos que ha sido un éxito al encontrar nuevas maneras para la detección de outliers en series temporales, que en este caso eran de contaminantes, pero podía ser de cualquier otro tema. Además, dentro del contexto de Tracasa Instrumental era importante aprender sobre series temporales para posteriores estudios que se quieran hacer en cualquier otro ámbito como, por ejemplo, en el estudio temporal de zonas por satélite.

Bibliografía

[1] "Historia de la inteligencia artificial." .

[2] A. D. E. L. A. Aema, "Quiénes somos."

[3] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, vol. 9781107057. 2013.

[4] A. Carreño López, "Detección de Sucesos Raros con Machine Learning," p. 95, 2017.

[5] F. Berzal, "Introducción Medidas de similitud," *Decsai*, vol. 1, no. 5, pp. 35–50, 2012.

[6] J. B. S. José Ramón Dorronsoro, *MÉTODOS DE APRENDIZAJE AUTOMÁTICO PARA DETECCIÓN DE ANOMALÍAS*. Madrid, 2019.

[7] B. Schölkopf *et al.*, "Support vector method for novelty detection," *Rev. en Telecomunicaciones e Informática*, vol. 3, no. 5, pp. 1–25, 1994.

[8] R. L. Briega, "Introducción a deep learning," *Introd. a Deep Learn.*, pp. 1–37, 2017.

[9] BIBING, "El Perceptrón," pp. 1–11, 2017.

[10] P. Larrañaga, "Tema 8. Redes Neuronales," pp. 1–19.

[11] J. M. Marín Diazaraque, "Introducción a las redes neuronales aplicadas," *Man. Data Min.*, pp. 1–31, 2007.

[12] F. Berzal, "Backpropagation Backpropagation Introducción Introducción."

[13] M. A. G. Naranjo, "Introducción a Deep Learning Deep Learning," 2018.

[14] R. Y. Lstm and R. Matuk, "Rosana Matuk (DC-FCEyN-UBA) RNN y LSTM Redes Neuronales Profundas," 2017.

[15] C. Olah, "Understanding LSTM Networks," 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[16] M. González, M. Inés, M. González, and M. Inés, "Tema 1 : Introducción a las series temporales Tema 1 : Introducción a las series temporales Definición y ejemplos."

[17] G. Corres, A. Esteban, J. García, and C. Zárate, "Análisis de series temporales," *Rev. Ing. Ind.*, vol. 8, no. 1, pp. 21–33, 2009.

[18] X. Barber, "Análisis clásico de las series temporales Índice de contenidos," 2019.

[19] A. M. Alonso, "Introducción al Análisis de Series Temporales. Cálculo de Tendencias y Estacionalidad," 2009.

[20] Robert B Cleveland, William S. Cleveland, Jean E. McRae, and Irma Terpenning, "STL: A Seasonal-Trend decomposition Procedure Based on Loess," *Journal of Official Statistics*, vol. 6, no. 1. pp. 3–73, 1990.

[21] R. J. H. George Athana-sopou-los, “Holt winters with R.”

[22] S. de la Fuente Fernández, “Series Temporales: Modelo Arima,” *Univ. Autónoma Madrid*, p. 53, 2016.