Identificación de datos anómalos en series temporales

TRABAJO FIN DE GRADO

Curso 2020/2021

UNIVERSIDAD COMPLUTENSE
MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

GRADO EN MATEMÁTICAS Y ESTADÍSTICA

Miguel Zabaleta Sarasa

Tutor: Daniel Vélez Serrano

Madrid, 2 de septiembre de 2021

Abstract

Este TFG aborda el problema de identificación de datos anómalos en series temporales. A tal fin se expondrán algunas de las metodologías más conocidas que se utilizan para hacerlo, como son: los clásicos algoritmos de detección de *outliers* que se aplican tras el ajuste de un modelo tipo ARIMA, los modelos *Isolation Forest* que suponen una variante no supervisada de los populares *Random Forest* y finalmente los métodos *Matrix Profile*, que suponen una de las técnicas más recientes y eficientes que existen en este contexto. Dichas metodologías serán aplicadas sobre las series temporales de libre acceso y diversa naturaleza vinculadas a la competición KDD 2021.

Abstract

This Bachelor's thesis addresses the problem of time series anomaly detection. To this end, some of the best-known methodologies used to do so will be exposed, such as: the classic outlier detection algorithms that are applied after adjusting an ARIMA-type model, the Isolation Forest models that represent an unsupervised variant of the popular Random Forest and finally the Matrix Profile methods, which are one of the most recent and efficient techniques that exist in this context. Such methodologies will be applied to the free access and diverse nature time series related to the KDD 2021 competition.

Tabla de contenido

1. Objetivo e introducción

Dentro del contexto de las series temporales, la detección de anomalías es un método de encontrar patrones que no se corresponden con el comportamiento esperado, estos patrones se denominan *outliers*. Hoy en día, esta cuestión es de gran importancia en el ámbito de la minería de datos, ya que identificando estos *outliers*, se puede analizar por qué ocurrieron. Algunos de los ámbitos donde la detección de *outliers* es más relevante son el médico (gráfico de un electrocardiograma), el energético (rotura de un patrón de demanda), el bancario (número o importe de operaciones retirada da un cajero), etc.

El objetivo de este trabajo se centra en primer lugar en presentar algunas de las metodologías más notables en la detección de *outliers* en series temporales, desde la más clásica metodología ARIMA, pasando el algoritmo *Isolation Forest*, versión no supervisada del popular *Random Forest*, hasta la reciente metodología basada en el *Matrix Profile*, focalizando nuestra atención en los algoritmos que sustentan esta última especialmente, debido a su rico fundamento matemático y a su alta eficiencia computacional. Además, aplicaremos los algoritmos correspondientes a las distintas metodologías sobre las series temporales ofrecidas en la competición KDD 2021, tratando de obtener la mayor tasa de acierto, discutiendo los resultados y obteniendo las conclusiones pertinentes.

En cuanto a la estructura del trabajo, se comienza elaborando un repaso a las propuestas más novedosas en la detección de *outliers* en series temporales, destacando las metodologías más usuales con un enfoque puramente estadístico, y también otras más recientes desde el ámbito del *machine learning* y las redes neuronales. A continuación de esto, comenzará la sección correspondiente al marco teórico de las metodologías. En primer lugar, se definen los conceptos previos necesarios para el correcto entendimiento del desarrollo constructivo realizado en la metodología *Matrix Profile*. Estos resultados incluyen definiciones, teoremas y el desarrollo en detalle a partir de éstos de la Transformada Rápida de Fourier, elemento clave en el éxito del algoritmo *Matrix Profile*. Una vez descritos estos resultados, se comienza con la exposición de los algoritmos que implementa la metodología *Matrix Profile*; de nuevo desde un enfoque constructivo, comenzando con los algoritmos que sustentan la computación de dicha metodología hasta los que computan el *Matrix Profile* propiamente dicho, presentando desde las versiones más rudimentarias hasta las más eficientes. En segundo lugar, se presenta el marco conceptual correspondiente a la metodología ARIMA y a la basada en el algoritmo *Isolation Forest*. Concluyendo esta sección, se hará una breve introducción a la teoría de *wavelets* dado que éstas han participado también en la metodología finalmente propuesta.

Por último, se aplicarán los algoritmos estudiados a las series temporales ofrecidas por la competición KDD 2021, detallando el objetivo de ésta, ofreciendo un primer acercamiento descriptivo para lograr cierta intuición sobre la naturaleza de los datos con los que trabajaremos y presentando los resultados obtenidos. Finalmente se incluye una sección, en la que se presentan las conclusiones adquiridas de los resultados y una breve nota a nivel personal.

2. Estado del arte

La detección de *outliers* en series temporales ha sido un campo de investigación importante durante mucho tiempo.

Sus orígenes se remontan al año 1975, cuando Box y Tiao [27] introducen el análisis de intervenciones como una técnica para reflejar el efecto de sucesos externos en una serie temporal. A tal fin, surgen algunos procedimientos iterativos como el de Chang y Tiao (1982, [28]) y Tsay (1986, [29]) en los cuales la detección de anomalías se hace a partir de los residuos del modelo ARIMA univariante especificado previamente para la serie temporal. Posteriormente, Chen y Liu (1993, [25]) hicieron propuestas en las que proponían un método de acuerdo al cual, la detección de *outliers*, la estimación conjunta de los parámetros y los efectos de las anomalías se realizaba en la fase de estimación del modelo univariante.

Este enfoque de tratar de predecir los valores que toma la serie temporal e identificar como *outliers* aquellos puntos en los que peor es la predicción se ha adoptado también en el más reciente ámbito del *machine learning*. Concretamente, con este fin, se han utilizado algoritmos muy potentes de predicción como las redes neuronales profundas totalmente conectadas (uso extendido en 2006 por el Hinton Lab al solucionar las dificultades del entrenamiento) y el *Extreme Gradient Boosting* (Tianqi Chen y Carlos Guestrin, agosto 2016). El procedimiento que siguen estas metodologías es, una vez realizadas las predicciones, se define un umbral del error cometido y, a partir de este umbral, se establece un percentil de observaciones que superen el umbral, obteniendo así el resultante conjunto de datos anómalos. [20]

Si analizamos el campo de investigación de detección de *outliers*, se puede apreciar que el trabajo seminal en este campo ha estado muy focalizado en enfoques puramente estadísticos. Sin embargo, en los últimos años se han desarrollado un número elevado de algoritmos basados en *machine learning* para detectar anomalías en series temporales [20], siendo bastante popular el *Subsequence Time Series Clustering* o STSC (Oates, 1998), basado en el popular *k-means*, el *Isolation Forest* (Liu, Fei Tony y Ting, Kai y Zhou, Zhi-Hua, 2009, referencia [2]), que una versión no supervisada del popular algoritmo *Random Forest* orientada a aislar explícitamente anomalías en una serie temporal y el RE-ADTS, propuesto por Amarbayasgalan et al. (junio 2020, [21]), basado en *deep learning*.

Sin embargo, es la metodología *Matrix Profile* propuesta por Eamonn Keogh en 2016 (referencia [9]) la que quizá haya permitido obtener mejores resultados. Dicha metodología se basa en calcular una medida de similitud entre los patrones presentes en una serie temporal. Mediante esta medida, se puede establecer que los patrones con muy poca similitud con el resto de subsecuencias de la serie temporal son anómalos.

Gran parte de este trabajo se enfocará en explicar en profundidad esta metodología, comenzando con los conceptos matemáticos previos necesarios para la construcción y deducción del método, siguiendo con los algoritmos que sustentan la base del *Matrix Profile* y por último detallando los algoritmos que computan el *Matrix Profile* propiamente dicho, desde sus versiones más rudimentarias hasta la que, en el momento de confeccionar este trabajo (julio de 2021), está implementada en el lenguaje de programación R (utilizada en el capítulo 4, “Presentación de resultados”).

Dicho esto, queremos mencionar algunas de las creaciones más novedosas que hacen posible el uso de la metodología *Matrix Profile* en los entornos más exigentes, como son la implementación en la nube, o el tratamiento de datos masivos.

El primer algoritmo se denomina SCAMP (SCAlable *Matrix Profile*, Eamonn et al. Nov 2019), que surge por la necesidad de, en campos como la sismología y la astronomía, tratar con mayor volumetría de datos de lo habitual, precisando de un algoritmo capaz de procesar datos que no entren en memoria. La clave de esta propuesta es su estructura, que permite ejecutar la computación del *Matrix Profile* en un *cluster* con un host y uno o varios "trabajadores" (*workers*). La referencia a este algoritmo puede encontrarse en [22].

La segunda propuesta aparece a la hora de implementar una solución que compute el *Matrix Profile* con datos que llegan al momento (*streaming data*). Existe una implementación que hace esto (llamada STOMP*I*) pero no es capaz de ofrecer buenos resultados ya que el tiempo que tarda en actualizar el *Matrix Profile* crece conforme se hayan procesado más datos. La solución es el algoritmo LAMP (*Learned Approximate Matrix Profile*, Eamonn et al. Nov 2019) que computa constantemente una aproximación del *Matrix Profile* para solventar el problema que mencionamos, y esto hace posible que se aplique en áreas como la neurociencia y la entomología. [23]

La última metodología basada en *Matrix Profile* que mencionaremos trata de aprovechar el hecho de que el Alineamiento Dinámico Temporal o DTW (*Dynamic Time Wraping*) se ha entendido a lo largo de los años como una medida de similitud superior a otras, como la distancia euclídea (utilizada por las implementaciones de *Matrix Profile* usuales). El alto coste computacional del DTW hacía inviable que fuera empleado en calcular el *Matrix Profile*. La referencia detallada en [24] supone la primera presentación de un método exacto, denominado SWAMP (*Scalable Wraping Aware Matrix Profile*, Eamonn et al. Nov 2020), que es eficiente y escalable para encontrar patrones en series temporales bajo el DTW y computando el *Matrix Profile*.

3. Marco teórico

La finalidad de la metodología presentada a continuación es la detección de datos anómalos o *outliers* en series temporales. Estos patrones fuera de lo común pueden tomar muchas formas; se incluye a continuación una figura con una representación de algunos de estos patrones.

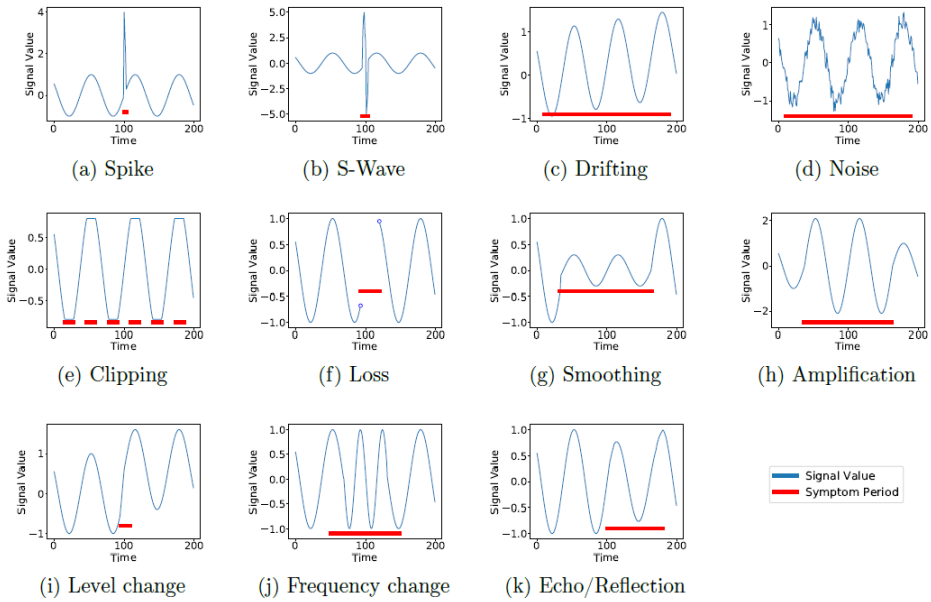

Figura 1: Ejemplos de tipología de *outliers* (extraída de [26])

En primer lugar, se presentarán una relación de conceptos preliminares necesarios para el mejor entendimiento del algoritmo *Matrix Profile*, que puede ser considerado como el referente que mejores resultados proporciona a tal fin. Es precisamente la metodología basada en dicho algoritmo la primera que se presenta a continuación. Sobre ella, se ha propuesto además un tratamiento posterior de elaboración propia basada en teoría de *wavelets*, a través del cual se ha obtenido una leve mejora de los resultados que ha permitido la identificación de algunos *outliers* adicionales de tipo *noise* (véase figura 1) no capturados por ella. Posteriormente se presentará otra más clásica consistente en el ajuste de un modelo de tipo ARIMA a la serie y la identificación de *outliers* sobre el proceso residual resultante de dicho ajuste y una más reciente denominada *Isolation Forest*. A su vez, en sus respectivos apéndices se presentarán una relación de conceptos preliminares necesarios para el mejor entendimiento de estas dos últimas metodologías.

3.1 Conceptos preliminares

La teoría de la señal es un campo que resulta muy conveniente cuando se trabaja con series temporales, ya que se puede estudiar una serie temporal como una señal, y viceversa. En esta área, son populares las metodologías basadas en análisis de Fourier, como la *Matrix Profile*, y en teoría de *wavelets*. A continuación se detallan los resultados previos necesarios para el correcto entendimiento de dichas metodologías.

3.1.1 Matrix Profile

Para ser capaces de comprender en profundidad la metodología en la que se basa el algoritmo *Matrix Profile*, especificaremos algunos resultados. En concreto, para que este algoritmo sea exitoso, esencialmente debe ser capaz de computar productos escalares de manera eficiente. Esto es posible

mediante el algoritmo de la Transformada Rápida de Fourier (FFT), que no es más que una forma de computar la transformada de Fourier discreta y su inversa (DFT e IDFT) en tiempo $\boldsymbol{O(n \log n)}$ (en lugar de $\boldsymbol{O(n^2)}$).

También es importante el Teorema de Convolución, resultado que relaciona la convolución, es decir, un producto escalar, con la transformada de Fourier. Una vez descritos estos resultados y sus definiciones correspondientes, procederemos a la explicación del algoritmo FFT, con la que finalizará esta sección.

Antes de presentar dichos resultados, es necesario dar una relación de conceptos preliminares necesarios para su mejor entendimiento. La referencia a la primera definición se encuentra en [8].

Definición 1 (Convolución): Sean f y g dos funciones, con $f, g: [0: \infty] \to \mathbb{R}$. Se define la convolución de f con g, y se denota $f * g$, por:

$$(f * g)(t) = \int_0^t f(x)g(t-x)dx = \int_0^t f(t-x)g(x)dx = (g * f)(t) \tag{1}$$

Obsérvese que otra forma de expresar lo anterior es considerar dos secuencias de números $x = \{x_k\}_{k=0}^{n-1}, y = \{y_k\}_{k=0}^{n-1}$, con $k = 1, 2, \dots, n-1$. Expresado de esta forma, la convolución de x con y queda:

$$x * y = \{x_k * y_k\}_{k=0}^{n-1} = \left\{\sum_{j=1}^{k} x_j y_{k-j}\right\}_{k=0}^{n-1} \tag{2}$$

Nótese la similitud de (2) con el producto escalar de x con y, $x \cdot y = \sum_{j=0}^{n-1} x_j y_j$.

Desarrollaremos la metodología del algoritmo FFT desde un enfoque de multiplicación de polinomios, ya que, como hemos introducido y detallaremos más adelante, multiplicar dos polinomios es equivalente a hacer su convolución, que es precisamente el cálculo que querremos computar eficientemente.

Como se puede encontrar en [6], existen varias formas de definir la transformada de Fourier y su inversa; cada definición es adoptada dependiendo del contexto en el que se use y de la conveniencia que presente. En nuestro caso, tomaremos la siguiente definición:

Definición 2 (Transformada de Fourier discreta y su inversa): Sea $x = \{x_t\}_{t=0}^{N-1}$ una sucesión de números reales. Se denota por $DFT_k(x)$ la transformada de Fourier discreta de x, y se define como:

$$DFT_k(x) = X_k = \sum_{n=0}^{N-1} x_n e^{i2\pi nk/N}, k = 0, 1, \dots, N-1 \tag{3}$$

Su inversa se define como:

$$IDFT_n(X_k) = x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{-i2\pi kn/N}, n = 0, 1, \dots, N-1 \tag{4}$$

Como adelantábamos al principio de la sección, existe un teorema que relaciona la convolución de dos vectores con sus transformadas de Fourier. Disponiendo del algoritmo FFT, podemos computar la DFT e IDFT de forma eficiente; por tanto, utilizando estos dos resultados seremos capaces de obtener la convolución de dos vectores, es decir, los productos escalares deseados, eficientemente.

El siguiente teorema nos permitirá computar la convolución de dos vectores en términos de sus transformadas de Fourier; la referencia de su enunciado y demostración se encuentra en [7].

Teorema 1 (Teorema de Convolución para funciones discretas): Sean x e y funciones discretas arbitrarias. Entonces:

$$DFT_k(x * y) = X_k Y_k \tag{5}$$

Alternativamente, aplicando la inversa de la transformada de Fourier discreta a ambos lados:

$$(x * y) = IDFT_k(X_k Y_k) \tag{6}$$

Demostración:

$$\begin{aligned}
DFT_k(x * y) &= \sum_{n=0}^{N-1} (x * y)_n e^{-i2\pi nk/N} \\
&= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(m) y(n-m) e^{-\frac{i2\pi nk}{N}} \\
&= \sum_{m=0}^{N-1} x(m) \sum_{n=0}^{N-1} y(n-m) e^{-\frac{i2\pi nk}{N}} \\
&= \sum_{m=0}^{N-1} x(m) \sum_{n=0}^{N-1} y(n-m) e^{-\frac{i2\pi nk}{N}} e^{\frac{i2\pi mk}{N}} e^{-\frac{i2\pi mk}{N}} \\
&= \sum_{m=0}^{N-1} x(m) \sum_{n=0}^{N-1} y(n-m) e^{-\frac{i2\pi(n-m)k}{N}} e^{-\frac{i2\pi mk}{N}} = \sum_{m=0}^{N-1} x(m) \sum_{n=0}^{N-1} Y_k e^{-\frac{i2\pi mk}{N}} \\
&= \left(\sum_{m=0}^{N-1} x(m) e^{-\frac{i2\pi mk}{N}} \right) Y_k = X_k Y_k \quad \blacksquare
\end{aligned}$$

El siguiente resultado se utiliza en la deducción de la complejidad computacional (o simplemente, complejidad) del algoritmo FFT, cuya referencia está detallada en [1].

Teorema 2 (Teorema Maestro):

El Teorema Maestro es un resultado que en el análisis computacional de algoritmos proporciona un análisis asintótico sobre la complejidad computacional de algoritmos de divide y vencerás. Dado un algoritmo de recurrencia, podemos describir el tiempo que tarda en realizar n operaciones como

$$T(n) = aT(n/b) + f(n) \tag{7}$$

donde $f(n)$ es el tiempo que se tarda en crear los subproblemas y combinar sus resultados según se sube en la recurrencia. En el caso que utilizaremos, el teorema garantiza que cuando $f(n) = \boldsymbol{O(n^c \log n^k)}$ para $k \geq 0$, entonces $T(n) = \boldsymbol{O(n^c \log n^{k+1})}$.

La demostración de este teorema puede ser encontrada en [1].

Transformada Rápida de Fourier

La transformada rápida de Fourier (FFT) es un algoritmo para computar la DFT e IDFT de una secuencia de números muy eficientemente (en tiempo $\boldsymbol{O(n \log n)}$) y esto lo convierte en uno de los algoritmos más importantes del siglo XX, cuya implementación más generalizada se atribuye a James Cooley y John Tukey, publicada en 1965.

La transformada de Fourier tiene multitud de aplicaciones: en procesamiento de señales, antenas, óptica, probabilidad, resolución de ecuaciones diferenciales, incluso física cuántica. En la era de digitalización en la que nos encontramos, todos estos problemas se han convertido en cuestiones en las que se trabaja con bits, es decir, con funciones discretas.

Por ello, la transformada de Fourier discreta es esencial en proponer soluciones y análisis a estos problemas. Sin embargo, el impedimento que esta técnica suponía era el alto coste computacional, ya que la complejidad de este método era de $\boldsymbol{O(n^2)}$**.**

La referencia a la introducción anterior puede ser encontrada en [5].

Como hemos descrito, la relevancia de la FFT en este trabajo radica en que uno de los algoritmos utilizados para la detección de anomalías (*Matrix Profile*) necesita calcular productos escalares muy rápidamente. Estos productos escalares son obtenidos mediante la convolución de dos vectores, que por el teorema de convolución sabemos que es equivalente a hallar la DFT e IDFT, las cuales son calculadas en tiempo computacional $\boldsymbol{O(n \log n)}$ mediante la FFT e IFFT.

De esta forma, conseguimos reducir la complejidad de la base del algoritmo *Matrix Profile* a $\boldsymbol{O(n \log n)}$.

A continuación, detallaremos el razonamiento detrás del algoritmo FFT, cuya referencia puede encontrarse en [1].

Sean dos polinomios de grado n:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} = \sum_{k=0}^{n-1} a_k x^k = \langle a_0, a_1, a_2, \dots, a_{n-1} \rangle \tag{8}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} = \sum_{k=0}^{n-1} b_k x^k = \langle b_0, b_1, b_2, \dots, b_{n-1} \rangle \tag{9}$$

Dados estos polinomios, partimos del objetivo de obtener el polinomio $C(x) = A(x) \cdot B(x)$.

Se tiene que $c_k = \sum_{j=0}^{k} a_j b_{k-j}, \forall\, k \in \{0, 1, \dots, n-1\}$. La complejidad computacional de hacer este cálculo es $\boldsymbol{O(n^2)}$, ya que hay n coeficientes que calcular, y cada uno tiene a su vez $\boldsymbol{O(n)}$ de complejidad por tener que realizar n multiplicaciones cada vez.

Como vimos en (2), la convolución de dos secuencias de números $x = \{x_k\}_{k=0}^{n-1}$ con $y = \{y_k\}_{k=0}^{n-1}$ es:

$$x * y = \{x_k * y_k\}_{k=0}^{n-1} = \left\{ \sum_{j=1}^{k} x_j y_{k-j} \right\}_{k=0}^{n-1}$$

que es igual a los coeficientes del polinomio $W(z) = X(z) \cdot Y(z)$, donde $x = \{x_k\}_{k=0}^{n-1}$ e $y = \{y_k\}_{k=0}^{n-1}$ son los coeficientes de $X(z)$ e $Y(z)$, respectivamente. Es decir, multiplicar dos polinomios es equivalente a hacer su convolución.

Recordemos que nuestro objetivo es realizar este cálculo en $\boldsymbol{O(n \log n)}$. Para ello, debemos introducir un par de formas de representación de polinomios: la de coeficientes y la de puntos.

Una representación de coeficientes de un polinomio $A(x) = \sum_{k=0}^{n-1} a_k x^k$ es el vector $a = (a_0, a_1, a_2, \dots, a_{n-1})$.

La multiplicación de polinomios en esta forma tiene complejidad $\boldsymbol{O(n^2)}$ ya que hay que multiplicar cada coeficiente de un polinomio por los coeficientes del otro, pero la ventaja está en que una vez obtenidos los coeficientes de $C(x) = A(x) \cdot B(x)$, directamente se tiene el polinomio $C(x)$.

Por otro lado, una representación de puntos de un polinomio $A(x) = \sum_{k=0}^{n-1} a_k x^k$ es un conjunto $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ tal que todos los x_k son distintos y tal que $A(x_k) = y_k$. El problema es que hallar esta representación simplemente seleccionando $\{x_0, x_1, \dots, x_{n-1}\}$ y evaluando $A(x_k), \forall\, k = 0, 1, \dots, n-1$ tiene una complejidad de $\boldsymbol{O(n^2)}$, por haber n puntos y cada uno con n operaciones que realizar. Sin embargo, veremos cómo podemos seleccionar los x_k de manera que consigamos la representación en $\boldsymbol{O(n \log n)}$.

Otra cuestión que surge es que $grado(C) = grado(A) + grado(B)$, es decir, necesitamos $2n$ pares de puntos para obtener una representación de puntos de $C(x)$. Por tanto, comenzaremos obteniendo representaciones de puntos de $A(x)$ y $B(x)$ consistentes de $2n$ pares de puntos cada una, $\{(x_0, y_0), (x_1, y_1), \dots, (x_{2n-1}, y_{2n-1})\}$ y $\{(x_0, y'_0), (x_1, y'_1), \dots, (x_{2n-1}, y'_{2n-1})\}$ respectivamente, y la representación de puntos de $C(x)$ será: $\{(x_0, y_0 y'_0), (x_1, y_1 y'_1), \dots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\}$.

En este caso, solo hay que computar una operación por cada punto, por tanto, la complejidad de hallar la representación de $C(x)$ es $\boldsymbol{O(n)}$, dadas las representaciones en puntos de $A(x)$ y $B(x)$.

Con esto, la pregunta que surge es si podemos usar la multiplicación en tiempo lineal de la representación de puntos para con estos puntos obtener la multiplicación de polinomios en forma de coeficientes. Esto dependerá de nuestra habilidad de convertir un polinomio rápidamente de forma de coeficientes a forma de puntos (evaluación) y viceversa (interpolación). El siguiente diagrama ilustra este proceso:

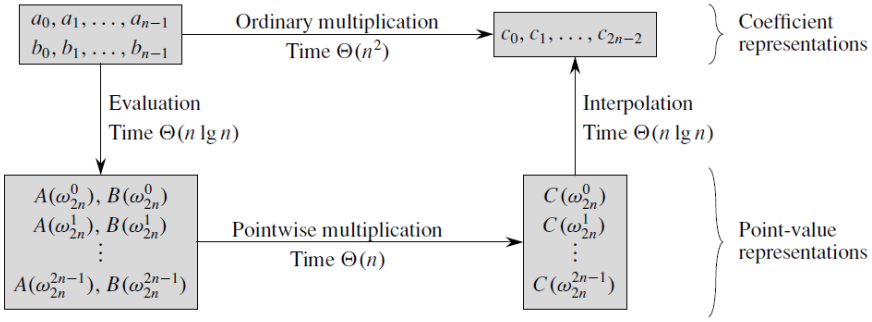

Figura 2: Diagrama multiplicación eficiente de polinomios (extraída de [1])

Como necesitamos una representación de $A(x)$ y $B(x)$ de $2n$ puntos, sus respectivas representaciones de coeficientes se extienden con ceros hasta tener longitud $2n$ y los x_k serán las raíces complejas $2n$-ésimas de la unidad (más adelante veremos el porqué de esos x_k).

Adelantamos ahora que mediante la DFT seremos capaces de realizar la evaluación y mediante la IDFT la interpolación, y la forma de computar esto en tiempo $\boldsymbol{O(n \log n)}$ será a través de la FFT e IFFT. De esta forma conseguiremos hallar la multiplicación de dos polinomios en tiempo $\boldsymbol{O(n \log n)}$.

A continuación, detallaremos el razonamiento detrás de elegir esos x_k y el desarrollo del algoritmo FFT como tal.

Sin pérdida de generalidad asumimos que $A(x)$ es un polinomio de grado n (en vez del polinomio de grado $2n$ que tendríamos en la práctica).

Sean $\omega_n^k = e^{i2\pi k/n}$ las raíces n-ésimas de la unidad, con $k = 0, 1, \ldots, n-1$.

El siguiente lema es necesario para demostrar el *Halving lemma*, uno de los resultados que hace posible la estructura del algoritmo FFT. Ambos, juntamente con sus demostraciones, han sido extraídos de [1].

Lema 1 (*Cancellation lemma*): Para todo entero $n \geq 0, k \geq 0, d \geq 0$:

$$\omega_{dn}^{dk} = \omega_n^k$$

Demostración:

Por definición,

$$\omega_{dn}^{dk} = \left(e^{2\pi i/dn}\right)^{dk} = \left(e^{2\pi i/n}\right)^k = \omega_n^k \quad \blacksquare$$

Corolario 1: Para todo entero par $n > 0$:

$$\omega_n^{n/2} = \omega_2 = -1$$

Lema 2 (*Halving lemma*): Si $n > 0$ es par, entonces los cuadrados de las n raíces complejas n-ésimas de la unidad son las $n/2$ raíces complejas $(\frac{n}{2})$-ésimas de la unidad.

Demostración:

Por el lema 1, tenemos $(\omega_n^k)^2 = \omega_{n/2}^k, \forall\, k > 0$. Nótese que si elevamos al cuadrado todas las n-raíces complejas de la unidad, entonces cada $\left(\frac{n}{2}\right)$-raíz de la unidad se obtiene exactamente dos veces, ya que

$$\left(\omega_n^{k+n/2}\right)^2 = \omega_n^{2k+n} = \omega_n^{2k}\omega_n^n = \omega_n^{2k} = (\omega_n^k)^2$$

Por tanto, ω_n^k y $\omega_n^{k+n/2}$ tienen el mismo cuadrado. ■

Comenzamos eligiendo como puntos x_k las raíces n-ésimas de la unidad, denotados por ω_n^k, para $k = 0, 1, \dots, n-1$. Recordamos que nos gustaría evaluar el polinomio $A(x) = \sum_{k=0}^{n-1} a_k x^k$ en estos puntos para obtener la representación de puntos de $A(x)$.

Es decir, queremos computar

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} = \sum_{j=0}^{n-1} a_j e^{i2\pi kj/n} \tag{10}$$

El vector $y = (y_0, y_1, \dots, y_{n-1})$ es precisamente la DFT del vector de coeficientes $a = (a_0, a_1, \dots, a_{n-1})$. Por tanto, computar la DFT equivale a tener una representación de puntos (evaluación).

Ahora, utilizando la FFT, que aprovecha las propiedades de las raíces complejas unitarias, podemos computar la DFT en tiempo $\boldsymbol{O(n \log n)}$.

El método FFT utiliza una estrategia de "divide y vencerás", separando los coeficientes de $A(x)$ en pares e impares, resultando en dos polinomios de grado $n/2$:

$$A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{\frac{n}{2}-1} \tag{11}$$

$$A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{\frac{n}{2}-1} \tag{12}$$

Nótese que

$$A(x) = A_{even}(x^2) + xA_{odd}(x^2) \tag{13}$$

entonces el problema de evaluar $A(x)$ en $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$, se reduce a evaluar $A_{even}(x)$ y $A_{odd}(x)$ en $(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n-1})^2$ y entonces combinar los resultados de acuerdo a (13).

Por el lema 2, el conjunto de puntos $(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n-1})^2$ no consiste de n puntos distintos, sino de $n/2$ puntos de las raíces $n/2$-ésimas de la unidad, donde cada raíz aparece exactamente dos veces. De esta forma hemos conseguido descomponer el problema anterior en dos subproblemas de la mitad de dimensión. Esta descomposición es la base del FFT, que realiza el proceso anterior recursivamente.

A continuación mostramos el planteamiento del algoritmo, donde se asume que n es una potencia de 2 y que A es el vector de coeficientes de $A(x)$. La referencia a este código puede encontrarse en [1].

Algoritmo 1 (Algoritmo FFT):

def $\boldsymbol{FFT(A)}$:

1. $n = len(A)$
2. $if\ n == 1$:
3. $\ \ return\ A$
4. $\omega = e^{i2\pi/n}$
5. $A_{even}, A_{odd} = [a_0, a_2, \dots, a_{n-2}], [a_1, a_3, \dots, a_{n-1}]$
6. $y_{even}, y_{odd} = FFT(A_{even}), FFT(A_{odd})$
7. $y = [0] * n$
8. $for\ j\ in\ range\left(\frac{n}{2}\right)$:
9. $\ \ y[j] = y_{even}[j] + \omega^j y_{odd}[j]$
10. $y[j + n/2] = y_{even}[j] - \omega^j y_{odd}[j]$
11. $return\ y$

Las líneas 2-3 representan la base de la recursión; cuando se llega a tener un elemento, su DFT es el elemento mismo, ya que tendríamos

$$y_0 = a_0\omega_1^0 = a_0 1 = a_0$$

En la línea 4 guardamos el valor de ω, que es actualizado como corresponde en las líneas 9-10. Continuando, en la línea 5 se definen las representaciones de coeficientes para A_{even}, A_{odd} (de acuerdo a la longitud del polinomio correspondiente n) y en la línea 6 se produce la recursión, realizándose hasta tener coeficientes de tamaño 1.

Una vez devuelto el valor del coeficiente (línea 3), se sube un nivel en la recursión y se continúa con la ejecución del código en la línea 7, donde se crea el vector DFT deseado, inicialmente con valores nulos.

Ahora, en las líneas 9-10 se calculan los coeficientes de la DFT; veamos esto en detalle.

Para $k = 0, 1, \dots, \frac{n}{2} - 1$, en la línea 9:

$$\begin{aligned} y[k] &= y_{even}[k] + \omega^k y_{odd}[k] \\ &= A_{even}(\omega^{2k}) + \omega^k A_{odd}(\omega^{2k}) \end{aligned}$$

Que por (13) es igual a $A(\omega^k)$.

Por otro lado, para $k = 0, 1, \dots, \frac{n}{2} - 1$, en la línea 10:

$$\begin{aligned} y\left[k + \frac{n}{2}\right] &= y_{even}[k] - \omega^k y_{odd}[k] \\ &\underset{(1)}{=} y_{even}[k] + \omega^{k+\frac{n}{2}} y_{odd}[k] \\ &\underset{(2)}{=} A_{even}(\omega^{2k}) + \omega^{k+\frac{n}{2}} A_{odd}(\omega^{2k}) \\ &= A_{even}(\omega^{2k+n}) + \omega^{k+n/2} A_{odd}(\omega^{2k+n}) \end{aligned}$$

Que por (13) es igual a $A\left(\omega^{k+n/2}\right)$.

(1): Por el corolario 1, se tiene que $\omega^{k+\left(\frac{n}{2}\right)} = -\omega^k$.

(2): Por definición, $\omega^n = 1$, por tanto $\omega^{2k} = \omega^{2k+n}$.

Es decir, de acuerdo a (10), el vector devuelto por la función $FFT(A)$ es en efecto la DFT del vector A.

En cuanto a la complejidad computacional de la recursión, en nuestro caso tenemos $T(n) = 2T\left(\frac{n}{2}\right) + \boldsymbol{O(n)}$, ya que recurrentemente dividimos el problema en dos subproblemas, cada uno con la mitad de operaciones a realizar, y se tarda $\boldsymbol{O(n)}$ en crear los subproblemas y combinar sus resultados.

Por el teorema 2, aplicando el caso en el que $f(n) = \boldsymbol{O(n^c \log n^k)}$ para $k \geq 0$ ($c = 1, k = 0$), tenemos que $T(n) = \boldsymbol{O(n^c \log n^{k+1})}$, por tanto, $T(n) = \boldsymbol{O(n \log n)}$.

Lo único que queda ahora es detallar el proceso de interpolación, es decir, cómo obtener la IDFT mediante el IFFT.

La diferencia entre la DFT y la IDFT es que, en la IDFT, los $\omega_n = e^{i2\pi/n}$ pasan a ser $\frac{1}{n}e^{-i2\pi/n}$, por tanto, simplemente cambiando $\omega = e^{i2\pi/n}$ por $\omega = \frac{1}{n}e^{-i2\pi/n}$ en el algoritmo 1, se obtiene la IFFT y por ende la capacidad de multiplicar dos polinomios en tiempo $\boldsymbol{O(n \log n)}$.

3.1.2 Breve introducción a las wavelets

Descritas a grandes rasgos, una *wavelet* (u ondícula) es una función ondulada construida de forma que presenta ciertas propiedades matemáticas. Un conjunto de *wavelets* es construido a partir de una sola función "*wavelet* madre", y este conjunto de *wavelets* proporciona información útil sobre la función sobre la que se aplica la *wavelet*, que suele ser una señal. Se han desarrollado diversas *wavelets* madre, cada una aportado diferentes ventajas e inconvenientes. [16]

Es adecuado relacionar el análisis de *wavelets* con el análisis de Fourier. La transformada de Fourier es un método de describir una señal (o función) en términos de las frecuencias que la componen. Sin embargo, los métodos basados en el análisis de Fourier fallan en describir correctamente una señal que cambia con el tiempo. Esto ocurre porque las funciones básicas de Fourier (basadas en senos y cosenos) son localizadas en frecuencia, y no en tiempo. Con "localizadas", se quiere decir que las oscilaciones de la función están concentradas en un intervalo pequeño. En cambio, las *wavelets* sí son localizadas en frecuencia y tiempo (referido como dilatación y traslación, respectivamente), permitiendo así describir cómo cambian las características importantes de la señal en estas dos dimensiones. [15][16]

Para ilustrar lo que acabamos de describir, la figura 3 muestra un ejemplo de un par de funciones de Fourier (izquierda) y de *wavelets* (derecha).

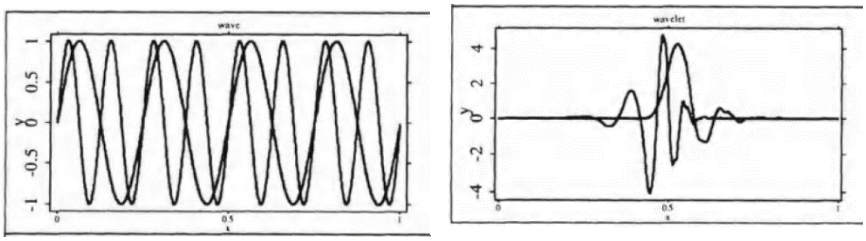

Figura 3: Ejemplo de funciones Fourier y *wavelets* (extraída de [15])

La propiedad de localización de las *wavelets* seguramente sea su característica más importante, pero también cabe destacar la propiedad de suavizado no lineal y la de ortonormalidad. Adicionalmente, las *wavelets* están intrínsicamente conectadas a la noción del análisis multirresolucional. Esto es, analizar diferentes objetos (señales, funciones, datos) utilizando distintos niveles de focalización. [16]

El análisis al que nos acabamos de referir se lleva a cabo estudiando los coeficientes *wavelets*. El primero de estos coeficientes es obtenido al pasar la transformada *wavelet* a la señal que queramos analizar, y los siguientes coeficientes se calculan pasando esta transformada a los sucesivos coeficientes *wavelets*; más adelante detallaremos este proceso.

Como campos en los que las *wavelets* son utilizados, podemos destacar el procesamiento de señales, análisis de imágenes, compresión de datos y más recientemente, análisis de series temporales. A continuación describiremos la obtención de los coeficientes *wavelets* para una transformada *wavelet* discreta general. [16]

Supóngase que observamos un conjunto de datos $\boldsymbol{x} = (x_1, \dots, x_n)$ y $n = 2^J$ para cierto $J \in \mathbb{N}$. La transformada *wavelet* discreta utiliza transformaciones ortogonales para descomponer el vector $\boldsymbol{x}$ en vectores de coeficientes *wavelets* $\boldsymbol{D}(J-1), \boldsymbol{D}(J-2), \dots, \boldsymbol{D}(0)$ y $\boldsymbol{C}(0)$. Cada conjunto de coeficientes *wavelets* contiene 2^j datos respectivamente, con $j = 0, 1, \dots, J-1$. En la práctica, estos coeficientes son computados directamente usando algoritmos de filtrado rápido basados en un filtro, $\mathcal{H}$. Este filtro está definido por los coeficientes $h(k)_{k\in\mathbb{Z}}$. [18]

Los sucesivos coeficientes *wavelets* son el resultado de aplicar el filtro $\mathcal{H}$ al coeficiente *wavelet* anterior, donde denotamos $\boldsymbol{x} = \boldsymbol{D}(J)$. Estos coeficientes contendrán el contenido de alta frecuencia, por lo tanto, serán extremadamente sensibles a características atípicas en la serie, es decir, *outliers*. [18]

Para entender qué representan estos coeficientes, mostramos a continuación la figura 4, extraída de [16].

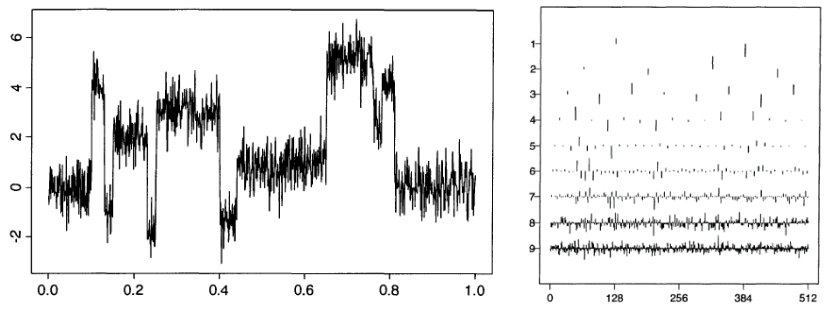

Figura 4: Serie temporal (izquierda) y coeficientes *wavelets* correspondientes (derecha)

Podemos deducir cómo los coeficientes *wavelets* son una manera de aproximar la señal original. Además, se aprecia que conforme aumenta el nivel de los coeficientes, lo hace también la precisión con la que se aproxima la señal.

En el caso de detección de *outliers* en series temporales, esto indica que los *outliers* más pronunciados serán más evidentes en los primeros órdenes de los coeficientes *wavelets*. Una ilustración del resultado que ofrecería un análisis de *wavelets* en una serie temporal viene representada por la figura 4 (derecha), donde se muestra el valor de estos coeficientes.

Como apuntábamos, efectivamente en los primeros niveles los coeficientes señalan los valores que pueden resultar más atípicos en la serie temporal.

Por último, resulta conveniente hacer una mención a un tipo concreto de *wavelets*, las *symlets*, pertenecientes a la familia Daubechies y utilizadas en el capítulo 4. Daubechies demostró que excepto por el sistema Haar, ningún sistema de *wavelets* puede ser a la vez compacto en el tiempo y simétrico. Con un enfoque práctico, las *symlets* surgieron con el fin de acercarse lo máximo posible a esta simetría, denominando así a esta familia de *wavelets* como la familia "menos asimétrica". [16][17]

Concluimos mostrando unos ejemplos de *wavelets symlets*, ilustrados en la figura 5.

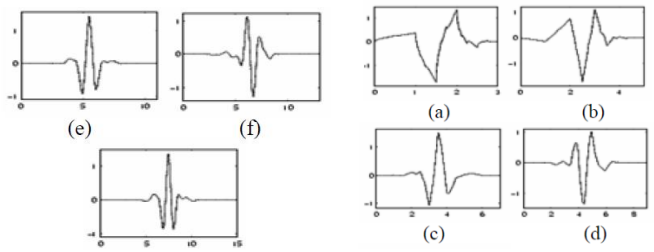

Figura 5: Ejemplos de *symlets* (extraída de [19])

3.2 Metodologías

Una vez vistos los conceptos matemáticos necesarios para comprender los métodos que utilizaremos, daremos comienzo a las explicaciones metodológicas de los algoritmos.

3.2.1 Metodología Matrix Profile

El problema de unión de similitud (cuya tarea básica consiste en, dado un conjunto de objetos de datos, obtener el vecino más cercano para cada objeto) ha sido un problema que hasta la fecha de publicación del algoritmo *Matrix Profile* solo había sido extensamente estudiado en problemas de texto y otros tipos de datos, pero no en series temporales. Esto no se debía a falta de interés, sino más bien a la desalentadora naturaleza del problema.

Cuando decimos que el problema es encontrar el vecino más cercano para cada objeto, en la práctica nos estamos refiriendo a calcular las distancias de cada subsecuencia de observaciones. Debido al gran volumen de datos propio de una serie temporal con datos reales, el proceso de calcular todas las distancias es extremadamente exhaustivo y computacionalmente muy demandante.

A continuación presentaremos las definiciones (véase [9]) que permitirán desarrollar la metodología *Matrix Profile*, comenzando con el concepto que sustenta el *Matrix Profile* como tal (*Distance Profile*) y posteriormente daremos la definición del *Matrix Profile*.

Definición 3 (*Distance Profile*): Consideremos una serie temporal de longitud n. Sea m el tamaño de las subsecuencias que tomamos; denotamos por d_{ij} la distancia de la subsecuencia i con la subsecuencia j, con $i, j = 1, 2, \dots, n - m + 1$. Se define el *Distance Profile* como $D_i = \left\{ d_{ij} : j \in \{1, 2, \dots, n - m + 1\} \right\}$.

La figura 6 ilustra el cálculo de un *Distance Profile*, obtenido calculando las distancias entre la ventana o *query* i y el resto de subsecuencias j a lo largo de la serie temporal. Esta imagen ha sido extraída de [12].

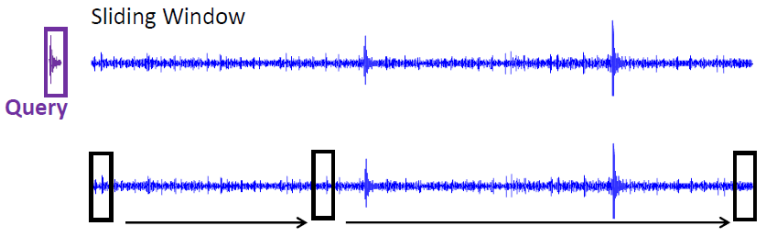

Figura 6: *Distance Profile*

Si agrupamos los d_{ij} en forma de matriz, conseguimos la matriz de distancias; a continuación se plantea la definición formal de *Matrix Profile*.

Definición 4 (*Matrix Profile*): Sea $P_i = \min_j \left\{ d_{ij} : j \in \{1, 2, \dots, n - m + 1\} \right\}$, para $i = 1, 2, \dots, n - m + 1$ una medida de la distancia entre la ventana i con su vecino más cercano en toda la serie temporal.

El conjunto formado por los P_i forma el *Matrix Profile*, y se puede ver como una meta-serie temporal.

La figura 7 ilustra la expresión y la interpretación a la que responde dicha matriz, ambas imágenes extraídas de [12].

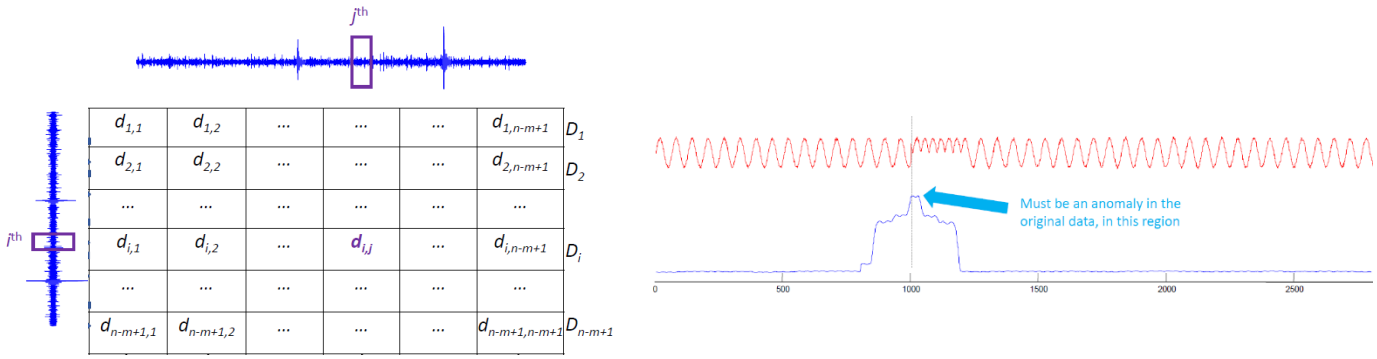

Figura 7: Matriz de distancias e interpretación

La línea en rojo de la figura 7 (derecha) representa una serie temporal y la azul la meta-serie resultante del *Matrix Profile*. Si pensamos en la serie original como la trayectoria de una señal, en las regiones donde el

patrón es muy repetitivo las distancias P_i son muy bajas ya que muchas otras ventanas de su misma longitud van a ser muy similares; por otro lado, en el tramo donde la frecuencia es más alta y más "aguda", las distancias P_i serán significativamente más altas y por tanto podemos identificar esa zona como anomalía. También mencionamos que la matriz de distancias es simétrica.

La metodología basada en el concepto de *Matrix Profile* consiste simplemente en calcular estas distancias de una manera muy eficiente, consiguiendo así reducir el tiempo para poder abordar problemas en series temporales que hasta la fecha no eran viables, como detección de patrones o detección de anomalías. Como ejemplo ilustrativo de la magnitud de mejora conseguida con este procedimiento, supongamos que después de un año de recoger registros cada minuto, tenemos una serie temporal de longitud 525600; se quiere calcular las distancias entre los vectores de longitud semanal (10080 de ellos). El bucle anidado tendrá que calcular más de 132000 millones de distancias euclídeas. Suponiendo que cada una se calcula en 0.0001 segundos, se tardaría 153.8 días en completar la tarea. El algoritmo de *Matrix Profile* es capaz de resolver este problema en 6.3 horas, más de 585 veces más rápido. [9]

En esencia, esto se consigue calculando las distancias euclídeas normalizadas como subrutina, aplicando la Transformada Rápida de Fourier (FFT) y explotando la superposición entre los vectores.

Del método de *Matrix Profile* podemos destacar las siguientes características: [9]

- Es exacto; es decir, la solución que ofrece no es una aproximación, sin ofrecer falsos positivos
- Es simple y libre de parámetros, en contraste al resto de algoritmos más generales
- Es fácilmente paralelizable, lo que lo hace abordable mediante técnicas de Big Data

En un principio, enfocaremos el desarrollo con el objetivo de computar un *Distance Profile*, es decir, computar las distancias de una misma ventana y_k con el resto de subsecuencias de la serie temporal $\boldsymbol{x}$.

En un paso posterior, se calcularán las distancias entre todas las ventanas y_k y las subsecuencias de $\boldsymbol{x}$, $\forall\, k = 1, 2, \dots, n - m + 1$, componiendo así la matriz de distancias, y siendo P_i el mínimo de un *Distance Profile*, el conjunto $P = \{P_i : i \in \{1, 2, \dots, n - m + 1\}\}$ será el *Matrix Profile*, cuya construcción buscamos.

3.2.1.1 Computación Distance Profile

A continuación, procederemos a detallar el desarrollo del algoritmo que computa un *Distance Profile* (algoritmo MASS), comenzando con las versiones menos optimizadas hasta conseguir la más eficiente. En cada versión se consigue reducir el tiempo de ejecución necesario; ilustraremos el rendimiento de todas ellas en un mismo gráfico después de ser introducidas. Una vez hecho esto, describiremos el algoritmo para computar el *Matrix Profile* (que utilizará a su vez el algoritmo MASS), de nuevo diferenciando entre sus versiones menos y más eficientes computacionalmente.

La referencia correspondiente a todo el desarrollo que se presenta en la sección 3.2.1.1 puede encontrarse en [9][12].

I. Algoritmo *Brute Force*

Por sencillez en la notación y sin pérdida de generalidad, supondremos que disponemos de dos series temporales $\boldsymbol{x} = x_1,\ x_2, \dots, x_n$ e $\boldsymbol{y} = y_1,\ y_2, \dots, y_n$; pero recuérdese que en la práctica estaremos trabajando con subsecuencias de una serie temporal y una ventana de la misma. Dicho esto, la distancia euclídea normalizada entre $\boldsymbol{x}$ e $\boldsymbol{y}$ se define por:

Definición 5 (Distancia euclídea normalizada entre dos vectores):

$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{\sum_{i=1}^{n}(\hat{x}_i - \hat{y}_i)^2}$, donde $\hat{x}_i = \frac{x_i - \mu_x}{\sigma_x}, \hat{y}_i = \frac{y_i - \mu_y}{\sigma_y}$

Podemos plantearnos simplemente computar todas las distancias entre la ventana y las subsecuencias. Esta es la versión cero del algoritmo, "a fuerza bruta". La complejidad computacional es $\boldsymbol{O((n - m + 1)m)}$, por tener que calcular $n - m + 1$ distancias y cada una de longitud m. Como ejemplo ilustrativo del tiempo que se tardaría en ejecutar, mencionamos que para una serie temporal de 100 millones de datos y con una ventana de longitud 100, se tardaría casi 1 hora en un ordenador convencional.

Nótese que siguiendo este algoritmo necesitamos calcular continuamente los vectores estandarizados de $\boldsymbol{x}$ e $\boldsymbol{y}$ llamando a una función que realice esta computación, llamémosla $Norm()$. Al llamar a $Norm()$, también se llamará internamente a otras funciones necesarias para la computación de la estandarización, como una función que calcule la media, y otra para la desviación típica.

Esto da paso a la siguiente versión del algoritmo, el *Just-in-time Normalization*.

II. Algoritmo *Just-in-time Normalization*

El primer paso para mejorar el resultado anterior es relacionar la distancia euclídea normalizada con el coeficiente de correlación de Pearson, y así desglosar la computación de los vectores estandarizados en cálculos de los estadísticos esenciales para la obtención de las distancias deseadas. No reduciremos la complejidad computacional, pero sí el número de funciones a las que llamamos, que resultará en una mejora del tiempo de ejecución.

Así pues, sean $\boldsymbol{x}$ e $\boldsymbol{y}$ dos vectores de dimensión *m*. El coeficiente de correlación de Pearson entre $\boldsymbol{x}$ e $\boldsymbol{y}$ se define como

$$corr(\boldsymbol{x}, \boldsymbol{y}) = \frac{(E(x) - \mu_x)(E(y) - \mu_y)}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{m} x_i y_i - m\mu_x \mu_y}{m\sigma_x \sigma_y} \tag{14}$$

Los estadísticos suficientes para calcular este estadístico son: [12]

$$\sum_{i=1}^{m} x_i y_i\,, \sum_{i=1}^{m} x_i\,, \sum_{i=1}^{m} y_i\,, \sum_{i=1}^{m} x_i^2\,, \sum_{i=1}^{m} y_i^2 \tag{15}$$

Es decir, una vez computados estos estadísticos, el coeficiente de correlación es una operación de tiempo constante.

Ahora, el teorema 3 nos permite relacionar la distancia euclídea normalizada de $\boldsymbol{x}$ e $\boldsymbol{y}$ con su coeficiente de correlación.

Teorema 3: Sean $\boldsymbol{x}$ e $\boldsymbol{y}$ dos vectores de igual dimensión, entonces:

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2m(1 - corr(\boldsymbol{x}, \boldsymbol{y}))}$$

Demostración:

Equivalentemente, basta demostrar que

$$\sum_{i=1}^{m} (\hat{x}_i - \hat{y}_i)^2 = 2m(1 - corr(x, y))$$

$$\begin{aligned}
\sum_{i=1}^{m} (\hat{x}_i - \hat{y}_i)^2 &= \sum_{i=1}^{m} \hat{x}_i^2 - 2\sum_{i=1}^{m} \hat{x}_i\,\hat{y}_i + \sum_{i=1}^{m} \hat{y}_i^2 = 2\left(m - \sum_{i=1}^{m} \hat{x}_i\,\hat{y}_i\right) \\
&= 2\left(m - \sum_{i=1}^{m} \left(\frac{x_i - \mu_x}{\sigma_x}\right)\left(\frac{y_i - \mu_y}{\sigma_y}\right)\right) \\
&= 2\left(m - \frac{1}{\sigma_x \sigma_y} \sum_{i=1}^{m} (x_i y_i - \mu_y x_i - \mu_x y_i + \mu_x \mu_y)\right) \\
&= 2\left(m - \frac{1}{\sigma_x \sigma_y} \sum_{i=1}^{m} (x_i y_i) - m\mu_y \mu_x - m\mu_x \mu_y + + m\mu_x \mu_y\right) \\
&= 2\left(m - \frac{\sum_{i=1}^{m} x_i y_i - m\mu_x \mu_y}{\sigma_x \sigma_y}\right) = 2m\left(1 - \frac{\sum_{i=1}^{m} x_i y_i - m\mu_x \mu_y}{m\sigma_x \sigma_y}\right) = 2m(1 - corr(x, y))
\end{aligned}$$ ■

Este es el cálculo que se plantea para computar las distancias, y para ello es preciso calcular la media y desviación típica de cada intervalo, además de los productos escalares. Podemos realizar las dos siguientes modificaciones para mejorar el algoritmo:

Supongamos que $\boldsymbol{y}$ se trata de la ventana que vamos moviendo; en vez de calcular cada vez su media y desviación típica para obtener el coeficiente de correlación, podemos hacer uso del hecho de que un vector normalizado tiene media 0 y desviación típica 1. Entonces, si a cada intervalo $\boldsymbol{y}$ le restamos su media y dividimos por su desviación típica, la expresión se reduce a:

$$d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \sqrt{2\left(m - \frac{\sum_{i=1}^{m} x_i y_i}{\sigma_x}\right)} \tag{16}$$

La segunda modificación consiste en computar todas las desviaciones típicas (pertenecientes a las distintas subsecuencias de $\boldsymbol{x}$) en un solo paso por la serie temporal, en vez de ser calculadas en cada vector.

Este es el procedimiento:

Algoritmo 2 (Cálculo desviaciones típicas en un paso, extraído de [10]):

1. Calcular $C = \sum x\,, C^2 = \sum x^2$.

2. Calcular $S_i = C_{i+m} - C_i,\ S_i^2 = C_{i+m}^2 - C_i^2$

3. Las desviaciones típicas deseadas son: $\sigma_i = \sqrt{\frac{S_i^2}{m} - \left(\frac{S_i}{m}\right)^2}$, donde $i = 1, 2, \dots, m$

Como habíamos adelantado, de esta forma no se reduce la complejidad computacional, pero sí el número de funciones que son llamadas. Esto resulta en más de 2 veces de reducción de tiempo de ejecución.

A continuación se presenta el algoritmo de Mueen para búsqueda de similitudes, conocido como MASS (*Mueen's Algorithm for Similarity Search*) que está enfocado a optimizar el cálculo de los productos escalares y es la primera versión que hace uso de la FFT.

III. Algoritmo MASS 1.0

El algoritmo MASS utiliza un método basado en convoluciones para calcular los productos escalares necesarios con complejidad computacional de $\boldsymbol{O(n \log n)}$.

Sea $\boldsymbol{x} = \{x_t\}_{t=1}^{n}$ la serie temporal y sea $y_k = \{x_t\}_{t=k}^{k+m-1}$ la ventana k-ésima de tamaño m que va deslizando conforme pasa por la serie temporal, calculándose las distancias entre $x_j = \{x_t\}_{t=j}^{j+m-1}$ e y_k, para todo $j = 1, 2, \dots, n - m + 1$.

En los conceptos preliminares de esta sección destacamos la similitud de la convolución discreta de dos secuencias de números con su producto escalar. En nuestro caso estamos interesados en calcular los productos escalares entre la serie temporal $\boldsymbol{x}$ y la ventana y_k. Estos productos se ilustran en la figura 8, que representa las asociaciones entre los coeficientes de $\boldsymbol{x}$ e y_k (suponemos $n = 4$, $m = 2$):

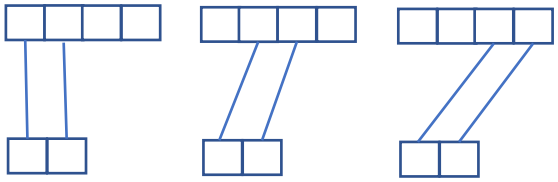

Figura 8: Productos escalares en función de coeficientes

La idea es utilizar la convolución discreta para calcular todos estos productos escalares.

Supongamos que x e y son dos vectores de dimensión 4. La imagen 9 (extraída de [12]) ilustra la convolución de x con y en términos de los productos de sus coeficientes:

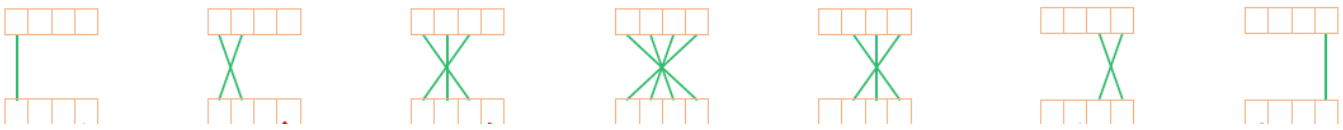

Figura 9: Convolución de dos vectores en función de sus coeficientes

En ella, queda de manifiesto que los productos buscados son un subconjunto de los términos calculados en la convolución de x con y.

La manera de conseguir el resultado deseado es transformar el vector y_k, revirtiéndolo y añadiendo ceros hasta que sea de la misma dimensión que $\boldsymbol{x}$; llamemos y_k^* al vector resultante de hacer esta transformación. De esta forma, al hacer $\boldsymbol{x} * y_k^*$ quedan exactamente los términos que queremos, además de algunos otros.

La figura 10 ilustra los coeficientes de los vectores $\boldsymbol{x}$ e y_k^*:

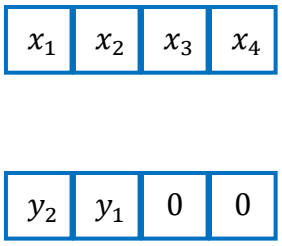

Figura 10: Coeficientes vectores $\boldsymbol{x}$ e y_k^*

Siguiendo en la figura 10 las líneas del segundo, tercero y cuarto gráfico (de izquierda a derecha) de la figura 9, llegamos a los términos $x_1y_1 + x_2y_2$, $x_2y_1 + x_3y_2$, $x_3y_1 + x_4y_2$, que corresponden con los productos escalares asociados a las distancias entre la ventana y las subsecuencias de la serie temporal, es decir, los productos escalares que queríamos de acuerdo a la figura 8.

Al computar los productos escalares mediante la convolución, por el Teorema de Convolución y la FFT conseguimos computar los productos escalares en tiempo $\boldsymbol{O(n \log n)}$ y, además, este cálculo no depende del tamaño m de la ventana, ya que independientemente del m que elijamos, añadiremos ceros al vector hasta hacerlo de la misma dimensión que $\boldsymbol{x}$. Esta última característica es importante ya que nos permite trabajar con datos muy grandes, donde sería razonable considerar un m relativamente grande.

Sin embargo, hemos visto que al hacer la convolución hay muchos términos que no son necesarios calcular, que corresponderían con los gráficos 1, 5, 6 y 7 de la figura 9. La siguiente versión del algoritmo tiene como objetivo optimizar el algoritmo actual haciendo que no calcule muchos de esos términos.

IV. Algoritmo MASS 2.0

Para ello, debemos observar cómo se calcula la convolución computacionalmente; concretamente, por el teorema 1:

$$(x * y_k^*) = IDFT(DFT(x)DFT(y_k^*)) \tag{17}$$

(recordemos que y_k^* es el vector y_k revertido y extendido con ceros hasta tener la misma dimensión que x) Aplicando la FFT e IFFT para calcular la DFT e IDFT respectivamente, tenemos:

$$(x * y_k^*) = IFFT(FFT(\hat{x})FFT(\widehat{y_k^*})) \tag{18}$$

donde $\hat{x}$ e $\widehat{y_k^*}$ denotan los vectores x e y_k^* extendidos a longitud $2n$ para poder realizar la FFT e IFFT (dado $C(x) = A(x) \cdot B(x)$, como $grado(C) = grado(A) + grado(B)$, se necesita extender los polinomios $A(x)$ y $B(x)$ hasta $2n$ para obtener la representación de puntos de $C(x)$).

Se tiene que la forma de reducir los términos que no necesitamos en la convolución es no extender con ceros los vectores x e y_k^*; llamamos a esta operación "media-convolución" (únicamente se calcula un término no necesario, el correspondiente con el gráfico 1 de la figura 9). Para realizar este ajuste necesitamos asumir que $\frac{n}{2} > m$, lo cual no supone un problema debido a que, por la naturaleza de las series temporales, podemos asumir que $n \gg m$.

V. Algoritmo MASS 3.0

La última versión del algoritmo MASS utiliza dos métodos habitualmente empleados en la implementación computacional eficiente de una convolución discreta en la que uno de los términos es mucho mayor en longitud que el otro, se trata de los métodos *overlap-add* y *overlap-save*. [11]

Una buena forma de explicar estos métodos es gráficamente, comenzaremos con el *overlap-add*, ilustrado en la figura 11 (izquierda).

La idea es dividir la serie temporal en segmentos de longitud K y cada segmento se extiende con $m-1$ ceros (representado por la línea negra). Después, se realiza la convolución usual entre la ventana y_k^* y cada segmento extendido. Es decir, en un segmento, los índices extendidos serán cero, pero en el siguiente segmento estos índices serán los de la serie temporal. Finalmente, el output final será el resultado de cada convolución, pero sumando los $m-1$ índices que se solapen entre cada segmento consecutivo (representado por la línea naranja).

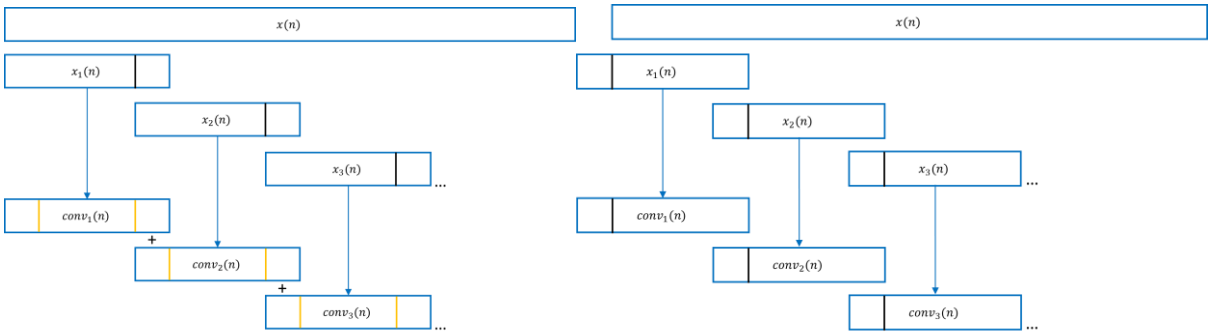

Figura 11: Esquemas algoritmos *overlap-add* y *overlap-save*

En el método *overlap-save*, representado en la figura 11 (derecha), la idea es muy similar, solo que esta vez los ceros se extienden por la izquierda de cada segmento. Una vez calculada la convolución de cada segmento extendido con la ventana y_k^*, se descartan los $m-1$ primeros coeficientes de cada convolución (línea negra), y el output final serán los coeficientes no descartados de cada convolución.

La eficiencia conseguida mediante las diferentes versiones que acabamos de detallar viene ilustrada por la figura 12.

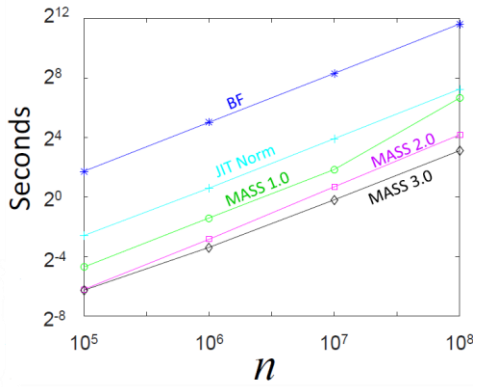

Figura 12: Tiempo de ejecución MASS 3.0 [12]

Comparando la primera y última versión, se tiene que el algoritmo a fuerza bruta necesita cerca de una hora (unos 2^{12} segundos) para calcular un *Distance Profile* de una serie temporal de 10^8 datos, y en cambio el algoritmo MASS 3.0 tan solo necesita unos 8 segundos.

Se ha calificado a este algoritmo de "ultra-eficiente" ya que comparándolo con el resto de algoritmos para computar eficientemente vecinos de una serie temporal, estos son más rápidos que computarlo a fuerza bruta en el mejor de los casos, pero en el peor de los casos son igual de rápidos que computarlos a fuerza bruta (y hay muchos de los llamados peores casos); como hemos visto, con este algoritmo se obtiene una mejora independientemente del tamaño de la ventana que se escoja, es decir, es considerablemente mejor independiente de los datos. [9]

3.2.1.2 Computación Matrix Profile

Recapitulando, hemos obtenido un algoritmo para calcular todas las distancias entre la serie temporal $\boldsymbol{x}$ y una misma ventana y_k en $\boldsymbol{O(n \log n)}$, que hemos optimizado hasta hacerlo "ultra-eficiente". Recordemos que $D_i = \left\{d_{ij} : j \in \{1, 2, \dots, n-m+1\}\right\}$ es el llamado *Distance Profile*, donde d_{ij} denota la distancia entre la subsecuencia i y la subsecuencia j, y denotamos por $P_i = \min_j \left\{d_{ij} : j \in \{1, 2, \dots, n-m+1\}\right\}$ (medida de la similitud más acentuada entre la ventana i-ésima y todas las subsecuencias j).

Ahora debemos calcular las distancias entre la ventana y_k y las subsecuencias de $\boldsymbol{x}$, $\forall\ k = 1, 2, \dots, n-m+1$, componiendo así la matriz de distancias, y el conjunto $P = \left\{P_i : i \in \{1, 2, \dots, n-m+1\}\right\}$ será el *Matrix Profile*.

El algoritmo del que disponemos hasta el momento con las herramientas que hemos descrito se llama STMP (*Scalable Time series Matrix Profile*), que consiste simplemente en calcular todos los *Distance Profile* para cada ventana, y tomar el mínimo de cada *Distance Profile*. Este algoritmo tiene una complejidad de $\boldsymbol{O(n^2 \log n)}$. La complejidad en realidad sería de $\boldsymbol{O((n-m+1)n\,log\,n)}$, por tener que computar $n - m + 1$ *Distance Profiles*, y cada uno tener una complejidad de $\boldsymbol{O(n\,log\,n)}$, pero como $n \gg m$, se aproxima $n - m + 1$ por n. [9]

I. Algoritmo STAMP

Un inconveniente que puede surgir es que, si trabajamos con un número muy grande de datos, el algoritmo lleve mucho tiempo en acabar de ejecutarse. Por tanto, sería conveniente poder parar la ejecución del algoritmo y que los resultados obtenidos hasta ese momento sean buenos.

El algoritmo STAMP (*Scalable Time series Anytime Matrix Profile*), soluciona precisamente esto, y se consigue simplemente escogiendo las ventanas y_k de forma aleatoria. [9]

La figura 13 muestra la eficiencia de esta característica, en la que podemos observar la raíz del error cuadrático medio entre la aproximación del *Matrix Profile* y su valor verdadero. Como se puede apreciar, en un número muy bajo de iteraciones (10% de ellas) se consigue una aproximación extremadamente similar.

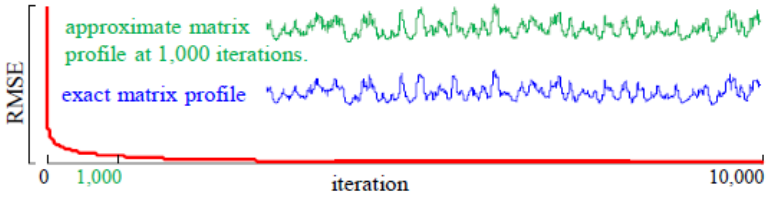

Figura 13: Eficiencia STAMP (extraída de [9])

Finalmente, los dos siguientes algoritmos tratan de aprovechar los cálculos realizados en computar los diferentes productos escalares, necesarios para el cálculo de las distancias.

II. Algoritmo STOMP

En concreto, el algoritmo STOMP (*Scalable Time series Ordered Matrix Profile*) no escoge las ventanas de forma aleatoria, pero aprovecha los cálculos realizados en computar el *Distance Profile* $(i-1)$-ésimo para calcular el i-ésimo. Su referencia puede encontrarse en [13].

Concretamente, si recordamos la expresión correspondiente al teorema 3, aplicándola en la distancia entre la subsecuencia i y j:

$$d_{i,j} = \sqrt{2m\left(1 - \frac{\sum_{k=0}^{m-1} t_{i+k}t_{j+k} - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right)} \tag{19}$$

Si denotamos por $T_iT_j = \sum_{k=0}^{m-1} t_{i+k}t_{j+k}$, entonces $T_{i-1}T_{j-1} = \sum_{k=0}^{m-1} t_{i-1+k}t_{j-1+k}$, por tanto, se tiene

$$T_iT_j = T_{i-1}T_{j-1} - t_{i-1}t_{j-1} + t_{i+m-1}t_{j+m-1} \tag{20}$$

Para la implementación de este algoritmo, es preciso tratar el caso especial en el que $i = 1$ o $j = 1$, esto se hace pre-computando los productos escalares asociados a la primera fila y columna de la matriz de distancias (que son idénticos, al ser la matriz de distancias simétrica).

También se pre-computan las desviaciones estándar y las medias de todas las ventanas en un solo paso mediante el algoritmo 2 (similarmente para las medias), ya que estos estadísticos deben estar guardados en

memoria para poder ser capaces de calcular la distancia entre las ventanas i y j en el momento en que hayamos computado sus productos escalares.

La figura 14 representa un esquema del planteamiento del algoritmo, extraída de [12].

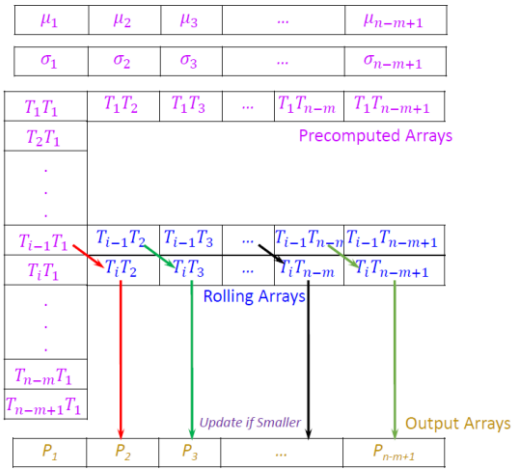

Figura 14: Esquema arquitectura algoritmo STOMP

Siguiendo el esquema ilustrado, primero se pre-computan los estadísticos necesarios y teniendo los productos entre las ventanas $i-1$ y j, para $j \in \{1, \dots, n-m+1\}$ (i fijo), se van calculando los productos escalares entre la ventana i y j; cada vez que se hace esto, se calcula la distancia entre las ventanas y se actualiza el valor de la distancia al vecino más cercano (P_j) si éste es menor al que había hasta ese momento. Iterando este proceso en las ventanas $i, i \in \{1, \dots n-m+1\}$, se forma el *Matrix Profile*.

La complejidad de STOMP es de $\boldsymbol{O(n^2)}$, en contraste con el $\boldsymbol{O(n^2 \log n)}$ de STAMP.

III. Algoritmo SCRIMP++

El último algoritmo que queremos mostrar es el SCRIMP++ (*Scalable Column Independent Matrix Profile*), ya que es el implementado en el paquete de R "*tsmp*", utilizado en el capítulo 4. En el momento de su publicación (2018), sus autores lo consideraron como "estado del arte". La referencia a este algoritmo y a las figuras mostradas puede encontrarse en [14].

Comparemos la eficiencia de los dos algoritmos desarrollados hasta el momento (STAMP y STOMP). Como se ilustra en la figura 15, existe un patrón claro en las secciones marcadas en verde; por tanto, los algoritmos deberían presentar un descenso pronunciado en esos tramos en el *Matrix Profile*, ya que, de identificarse el otro tramo, la distancia entre ambos tramos sería muy baja. Observamos como el algoritmo STAMP puede ofrecer buenos resultados con solo el 10% de sus computaciones, mientras que el algoritmo STOMP no consigue identificar el patrón con un 50% de las computaciones realizadas.

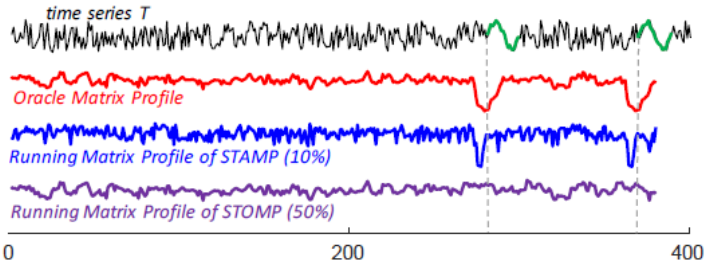

Figura 15: Representación acierto de STAMP vs STOMP

Por otro lado, cuando se trabaja con una serie temporal muy larga y los vecinos son raros, la probabilidad de que el algoritmo STAMP encuentre los k mejores vecinos en el 10% de sus computaciones se reduce en gran medida. El algoritmo SCRIMP++ consigue solventar este dilema entre usar STAMP o STOMP, incluyendo la ventaja de la aleatoriedad del algoritmo STAMP en el algoritmo STOMP.

En concreto, el algoritmo SCRIMP++ consiste de dos partes: PreSCRIMP y SCRIMP, como se muestra en la figura 16.

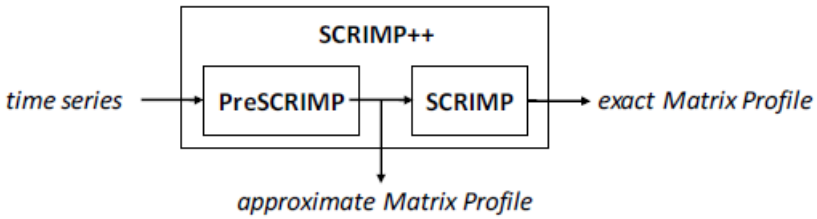

Figura 16: Esquema arquitectura algoritmo SCRIMP++

Comenzamos describiendo el algoritmo SCRIMP:

Nótese que por la ecuación (20), también tenemos las diagonales de la matriz de distancias. Por tanto, podemos elegir qué diagonales calculamos de forma aleatoria, e ir actualizando los valores correspondientes del *Matrix Profile* si son menores que los que teníamos.

El esquema del planteamiento del algoritmo SCRIMP viene representado por la figura 17.

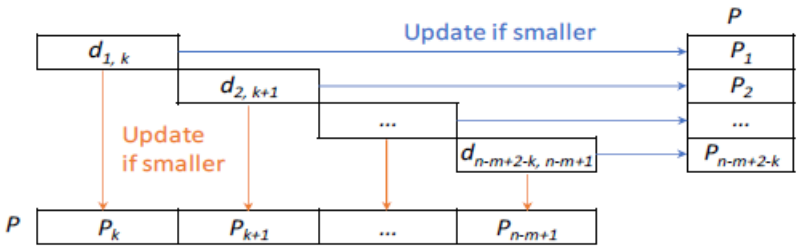

Figura 17: Esquema implementación algoritmo SCRIMP

Adicionalmente, como diferencia con el algoritmo STOMP podemos destacar que al estar iterando entre ambas dimensiones de las distancias $d_{i,j}$, se comprueban los elementos P_i y P_j del *Matrix Profile*. En cambio, en el algoritmo STOMP se itera solo en la dimensión j, por tanto es el valor P_j el que se va verificando en el *Matrix Profile*.

Sin embargo, la manera de iterar en SCRIMP presenta problemas, ya que podría darse una situación en la que solo existiese un patrón repetido a lo largo de la serie temporal y este patrón se encuentre en una de las últimas diagonales o en uno de los últimos elementos de una de las diagonales, lo que derivaría en unos resultados pobres si parásemos la ejecución rápidamente, que es la ventaja que supuestamente conseguimos incluyendo la aleatorización. Esta situación viene representada por la figura 18.

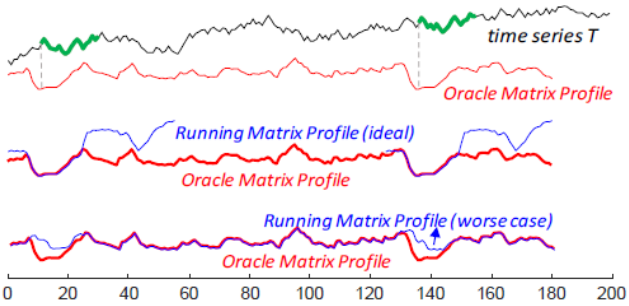

Figura 18: Comparación caso ideal y peor de SCRIMP

En la figura 18 podemos apreciar como existe un patrón en la serie temporal que debería ser identificado por el algoritmo, resultando en un descenso pronunciado en la meta-serie temporal del *Matrix Profile* obtenida. En el mejor de los casos, este patrón es calculado en la primera iteración porque casualmente la aleatorización comienza en la diagonal adecuada (se puede distinguir que se trata de una de las primeras iteraciones porque fuera de los tramos correspondientes a los patrones el *Matrix Profile* obtenido y el "real" (*Oracle*) son muy distintos); sin embargo, en el peor de los casos esta diagonal no es calculada hasta la última iteración (se observa que en este caso el resto de la meta-serie temporal es idéntica a la real).

Es cierto que este es el caso más difícil posible para el algoritmo SCRIMP, donde solo hay un patrón a descubrir, pero es un ejemplo que ayuda a hacerse a la idea de las limitaciones de este algoritmo, y por tanto la motivación detrás de mejorarlo en el algoritmo SCRIMP++, que detallaremos a continuación.

Recordemos que, como se indicó en la figura 17, el algoritmo SCRIMP++ está formado por el algoritmo PreSCRIMP, que recibe la serie temporal y el output generado por PreSCRIMP es el input de SCRIMP, cuyo output resulta en el *Matrix Profile* final de SCRIMP++.

Para describir el algoritmo PreSCRIMP, antes debemos mencionar una propiedad de las subsecuencias de una serie temporal, denominada por los autores como la "propiedad de la conservación consecutiva del vecindario", o CNP (*Consecutive Neighborhood Preserving property*). Esta propiedad consiste en la idea de que, dado que las subsecuencias consecutivas se solapan en muchos de sus datos, uno podría esperar que si la i-ésima subsecuencia es muy similar a la j-ésima, entonces con una probabilidad muy alta, la $(i+1)$-ésima será muy similar a la $(j+1)$-ésima subsecuencia. Esta propiedad puede verse reflejada en la figura 19.

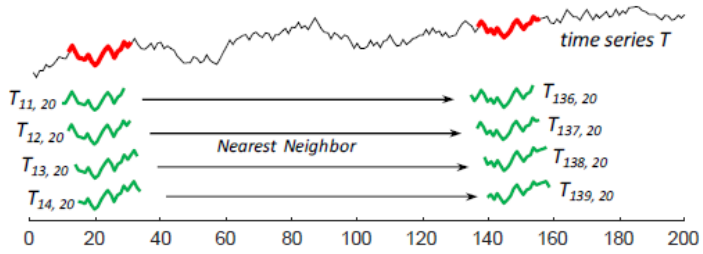

Figura 19: Ilustración propiedad CNP

El algoritmo PreSCRIMP trata de aprovechar la propiedad CNP, y lo hace primeramente muestreando subsecuencias de la serie temporal de tamaño fijo s. Para cada subsecuencia, elegida aleatoriamente, el algoritmo encuentra su vecino exacto más cercano. Supóngase que $T_{i,m}$ es una subsecuencia muestreada de tamaño m y que su vecino más cercano es $T_{j,m}$; ahora, de acuerdo a la propiedad CNP, existe una probabilidad muy alta de que el vecino más cercano de $T_{i+k,m}$ sea $T_{j+k,m}$, para $k = -s + 1, -s + 2, \dots, -2, -1, 1, 2, \dots, s - 2, s - 1$. Así pues, se calculan las distancias entre $T_{i+k,m}$ y $T_{j+k,m}$, actualizando el *Matrix Profile* si apareciese una distancia menor a lo largo de las subsecuencias.

Como cuestión práctica, se fija el valor de s a $s = m/4$ (donde m es el tamaño de la ventana), lo cual se ha demostrado empíricamente que ofrece buenos resultados.

Una vez hecho esto, se sigue refinando el *Matrix Profile* con el algoritmo SCRIMP hasta que el algoritmo converge a la solución exacta del *Matrix Profile*. De esta forma se consigue un algoritmo que pueda ser interrumpido antes de converger y siga ofreciendo buenos resultados.

Comparando los tres algoritmos para el cálculo del *Matrix Profile* desarrollados (STAMP, STOMP y SCRIMP++), se estudiaron cuatro conjuntos de datos, cada uno compuesto por 100 series temporales de longitud 40.000, en las que se incluyen varios patrones a identificar de longitud $m = 400$. En el primer conjunto de datos las series temporales son caminos aleatorios con solo un par de patrones; en el segundo conjunto, son caminos aleatorios pero con diez pares de patrones a identificar; el tercer conjunto lo componen series temporales de ruido sismográfico (con un par de patrones) y por último, el cuarto conjunto de datos es ruido blanco, donde el objetivo consiste en identificar su "patrón natural" de longitud 400.

En cada ejecución de los algoritmos, éstos son interrumpidos, se marca el tiempo t y entonces se extraen los k mejores patrones obtenidos, con k dependiendo del conjunto de datos; entonces se compara si los patrones han sido extraídos exitosamente (se considera éxito si el patrón obtenido solapa en al menos un 95% con respecto al verdadero). Las figuras 20 y 21 muestran el éxito de los diferentes algoritmos, representando el porcentaje medio de los patrones descubiertos en t.

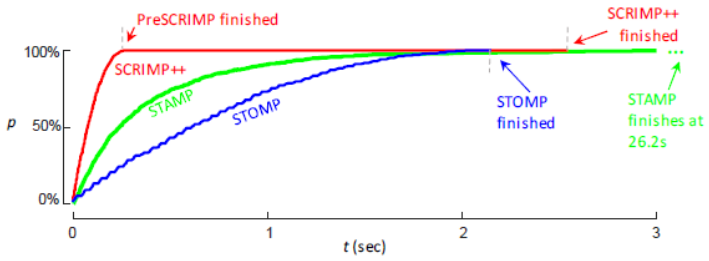

Figura 20: Acierto algoritmos en el segundo *dataset*

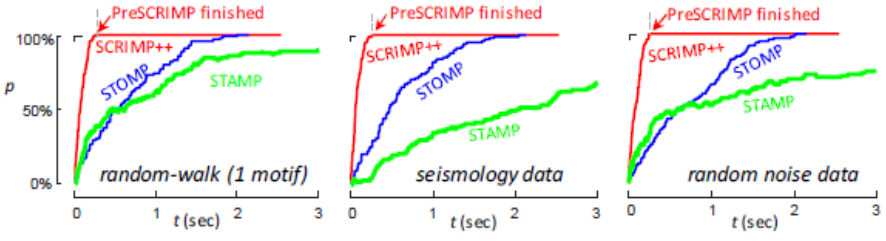

Figura 21: Acierto algoritmos en el resto de *datasets*

De esta forma concluimos esta sección, destacando el algoritmo SCRIMP++ como el mejor entre todos ellos en todos los conjuntos de datos.

3.2.2 Otras metodologías para la identificación de outliers en series temporales

Procedemos ahora a describir a un nivel más alto los marcos teóricos que acompañan a dos metodologías relevantes en la detección de anomalías en series temporales. La primera de ellas es la basada en un modelo ARIMA, que podemos categorizar de la metodología más clásica, y la segunda corresponde con el algoritmo *Isolation Forest*, que se puede entender como una versión no-supervisada del algoritmo *Random Forest*, donde las separaciones se realizan de manera aleatoria.

3.2.2.1 Identificación de outliers a partir del residuo de un modelo ARIMA

La metodología Box-Jenkins (Peña, [30]) es posiblemente la más popular dentro del análisis de series temporales. Dicha metodología consiste en realizar una relación de transformaciones a la serie buscando que el comportamiento de ésta sea estacionario y ajustar un modelo ARMA estacionario tras el cual:

- Los parámetros resulten significativos y no presenten correlación entre ellos que puedan derivar en problemas de multicolinealidad.
- El proceso residual resultante responda a las hipótesis de ruido blanco.

Los algoritmos de identificación de *outliers* dentro del contexto del análisis series temporales suelen aplicarse tras el ajuste de un modelo de este tipo. Dicho análisis consiste en identificar el momento en el que dicho *outlier* comienza y la forma a la que responde.

Continuamos esta sección con la explicación de la metodología de identificación de *outliers* implementada en la función *locate.outliers()* del paquete "*tsoutliers*" de R, utilizada en el capítulo 4. Esta metodología se basa en el enfoque propuesto por Chen y Liu (1993), cuya referencia puede encontrarse en [25].

La metodología en cuestión comienza definiendo un modelo ARIMA para la serie y_t^* sujeta a m *outliers* definidos como $L_j(B)$ con pesos ω_j:

$$y_t^* = \sum_{j=1}^{m} \omega_j \, L_j(B) I_t(t_j) + \frac{\theta(B)}{\phi(B)\alpha(B)} a_t \tag{21}$$

donde $I_t(t_j)$ es la función indicatriz de *outlier* j-ésimo, presente en el instante t_j y $\frac{\theta(B)}{\phi(B)\alpha(B)}$ es el ajuste ARIMA habitual. La presencia de *outliers* es probada por medias de estadísticos *t* aplicados en los residuos de la serie. Se computan todos los estadísticos *t* para cada tipo de *outlier*, en cada punto de la serie. Después, los que superen el valor crítico son considerados como *outliers* (*cval* es un parámetro que denota este valor crítico y que la función *locate.outliers()* admite).

A continuación se incluyen las expresiones estandarizadas de los estadísticos mencionados para cada tipo de *outlier*, que pueden ser encontradas en [25].

$$\hat{\tau}_{IO}(t_1) = \hat{\omega}_{IO}(t_1)/\hat{\sigma}_a \tag{22}$$

$$\hat{\tau}_{AO}(t_1) = (\hat{\omega}_{AO}(t_1)/\hat{\sigma}_a)\left(\sum_{t=t_1}^{n} x_{2t}^2\right)^{1/2} \tag{23}$$

$$\hat{\tau}_{LS}(t_1) = (\hat{\omega}_{LS}(t_1)/\hat{\sigma}_a)\left(\sum_{t=t_1}^{n} x_{3t}^2\right)^{1/2} \tag{24}$$

$$\hat{\tau}_{TC}(t_1) = (\hat{\omega}_{TC}(t_1)/\hat{\sigma}_a)\left(\sum_{t=t_1}^{n} x_{4t}^2\right)^{1/2} \tag{25}$$

En el artículo referenciado se proponen tres maneras de estimar σ_a; mencionamos la primera de ellas, dada por la expresión $\hat{\sigma}_a = 1.483 * median(|\hat{e}_t - \tilde{e}|)$, donde $\tilde{e}$ es la mediana de los residuos estimados.

Finalizamos la explicación con el procedimiento para encontrar los *outliers*. Primero se calculan los estadísticos mencionados anteriormente para cada t y se halla el máximo de ellos en cada instante de tiempo. Después se calcula el máximo de éstos y si en valor absoluto resulta mayor que el valor crítico determinado, se considera como *outlier*. Entonces se elimina su efecto en la serie de acuerdo al tipo de *outlier*, se vuelve a ajustar el modelo y se repite el proceso de búsqueda del *outlier* más evidente.

También creemos conveniente mencionar el algoritmo desarrollado por Peña (véase [30]), el cual sigue un procedimiento similar al descrito anteriormente en cuanto a la manera iterada de identificar los *outliers*, corrigiendo su efecto en la serie, pero con diferentes expresiones de los estadísticos.

Ambos algoritmos permiten identificar *outliers* de tipo aditivo (véase tipología a en la figura 1), de tipo escalón (véase tipología i en la figura 1) y de tipo rampa (véase tipología c en la figura 1).

3.2.2.2 Isolation Forest

El método *Isolation Forest* o *iForest* es fundamentalmente diferente a muchos de los enfoques basados en modelos que había en el momento de su publicación. Estos enfoques esencialmente construyen un perfil de datos no anómalos, y a partir de este perfil, deciden los que son anómalos. Ejemplos muy relevantes de este planteamiento son métodos basados en clasificación, en *clustering* o métodos estadísticos. [2]

Estos métodos presentan dos desventajas: el detector que resulta está optimizado para detectar datos normales, y no para detectar datos anómalos. Esto suele resultar en peores resultados ya que tiende a haber demasiados falsos negativos (asignar anomalía cuando no lo es) o demasiadas pocas anomalías detectadas. La segunda desventaja es que, debido a su alta complejidad, muchos de estos algoritmos están restringidos a conjuntos de una baja dimensionalidad. [2]

El algoritmo *iForest*, por el contrario, es capaz de separar directamente las observaciones anómalas de las no anómalas eficientemente. La idea detrás del algoritmo es que las anomalías, al ser pocas y estar diferenciadamente separadas del resto de observaciones, son muy susceptibles a ser fácilmente aisladas por un método basado en un árbol de aislamiento, concepto explicado anteriormente en los conceptos preliminares. Las figuras mostradas en esta sección pueden ser encontradas en [2].

El desempeño de este algoritmo para aislar una observación no anómala y una que si lo es se muestra en la figura 22.

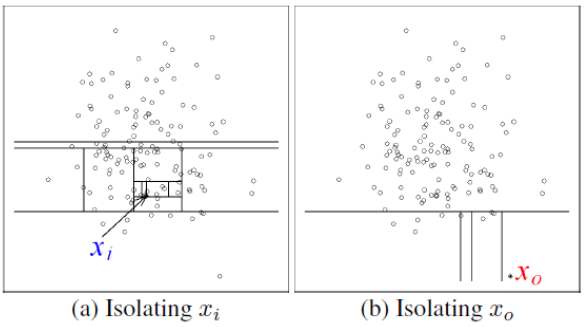

Figura 22: Aislamiento de dos observaciones mediante *iForest*

En el gráfico de la izquierda podemos ver como para aislar la observación x_0 se necesitan muchas menos separaciones que para aislar la observación x_i.

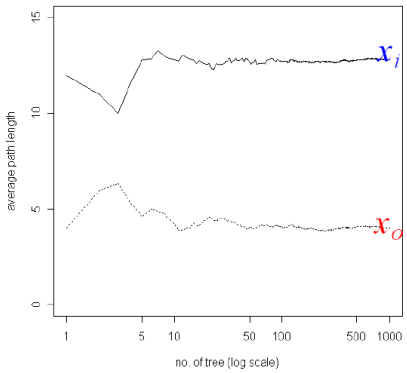

Figura 23: Número medio de separaciones para aislar una observación no anómala y una anómala

En la figura 23 se aprecia como el número de separaciones necesarias para aislar las observaciones converge conforme aumenta el número de árboles utilizados en el *Isolation Forest* y este número de separaciones es mucho menor en x_0 que en x_i (4.02 vs 12.82).

En este momento, nos parece conveniente referenciar las dos definiciones pertenecientes a esta sección, encontradas en el anexo, Apéndice B.

Para garantizar su convergencia, y de la misma forma que en el algoritmo *Random Forest*, se cogen submuestras de los datos y se calcula para cada uno de ellos su árbol de aislamiento. De esta forma se construye el *Isolation Forest*.

La cuestión que queda pendiente es establecer qué criterio seguir para determinar si una observación es anómala o no. Para ello, se define una "puntuación de anomalía", que va a depender del número de separaciones necesarias para el aislamiento de cada observación, llamado "longitud del camino" y denotado por $h(x)$. Como los *iTree* tienen una estructura equivalente a la de un árbol binario de búsqueda, la estimación de la longitud media del camino en los nodos externos será la misma que para las búsquedas fallidas, cuya estimación (véase [3]) viene dada por:

$$c(n) = 2H(n-1) - \left(\frac{2(n-1)}{n}\right) \tag{26}$$

donde n es el número de observaciones en el árbol y $H(n)$ es el número armónico, que puede estimarse por $ln(i) + 0.57721$ (constante de Euler-Mascheroni). Como $c(n)$ es la media de $h(x)$ dado n, puede ser utilizado para normalizar $h(x)$. La puntuación de anomalía de una observación x está definida por:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{27}$$

donde $\mathrm{E(h(x))}$ es la media de $h(x)$ en una colección de árboles de aislamiento.

Las características destacables de $s(x,n)$ son:

- cuando $\mathrm{E(h(x))} \to c(n),\ s \to 0.5$.
- cuando $\mathrm{E(h(x))} \to 0,\ s \to 1$.
- cuando $\mathrm{E(h(x))} \to n-1,\ s \to 0$.

Las conclusiones que obtenemos de estas condiciones son que si el número de divisiones medio de una observación tiende al número de observaciones totales, es decir, se necesitan muchas divisiones para ser aislada, entonces la puntuación de anomalía es casi 0 y es muy seguro decir que esa observación no es anómala; si el número de divisiones necesarias tiende a 0, la puntuación de anomalía tiende a 1 y por tanto definitivamente será una anomalía y si todas las observaciones tienen una puntuación de anomalía cercana a 0.5, entonces no hay una anomalía clara en todo el conjunto de datos.

Algoritmo *iForest*

El algoritmo *iForest* para detección de anomalías es un proceso de dos etapas: en la primera etapa (de entrenamiento) se construyen los árboles de aislamiento utilizando submuestras del conjunto de entrenamiento aleatorias sin reemplazamiento. La segunda etapa (de testeo) se pasan las observaciones del conjunto de test por los árboles de aislamiento para determinar la puntuación de anomalía.

En la etapa de entrenamiento, los árboles son construidos particionando las submuestras hasta que o bien todas las observaciones están aisladas o hasta que cierta profundidad del árbol es alcanzada. Esta profundidad l viene estimada por la altura media del árbol aproximadamente, que es automáticamente

determinada por el tamaño de la submuestra ψ: $l = ceiling(\log_2 \psi)$ (véase [4]). Desarrollar los árboles únicamente hasta esa profundidad es importante ya que solo estamos interesados en aquellas observaciones que tengan una longitud del camino menor que la media, así no será necesario desarrollar por completo el árbol y ganaremos eficiencia computacional.

En cuanto al valor óptimo ψ, se ha determinado empíricamente que a partir de cierto valor el algoritmo no mejora y solo aumenta el tiempo computacional. Generalmente se tiene que fijando el valor de ψ a 256 suele ofrecer buenos resultados. Por otra parte, el parámetro del número de árboles t que conforman el *iForest* suele converger antes de $t = 100$. La complejidad de entrenar este algoritmo es $\boldsymbol{O(t\psi \log \psi)}$.

Una vez hecho esto, se procede a la etapa de testeo. Recordemos que nuestro objetivo es calcular la ecuación (27) para cada observación. Lo único que falta que estimar es $E(h(x))$; esto se hace aplicando para cada observación *x* en el conjunto de test, la función $PathLength(x, T, e)$ a cada *iTree* T, donde e es el valor actual de $PathLength()$, que comienza en cero.

Concretamente, cuando en el siguiente árbol T, x pertenece a un nodo terminal, $e' = e + c(n_T)$. Si x no pertenece a un nodo terminal, se calcula $PathLength(x, T_{sup}, e + 1)$, donde T_{sup} denota el subárbol resultante de podar T en el nivel del nodo hoja más profundo. La complejidad de la etapa de testeo es $\boldsymbol{O(nt \log \psi)}$.

Para encontrar m anomalías, simplemente se seleccionan los m mayores valores de $s(x, n)$ siempre y cuando estas puntuaciones sean lo suficientemente cercanas a 1.

Otro dato que será de interés en el capítulo 4 es que, para un entrenamiento sin observaciones anómalas, se recomienda aumentar el valor del parámetro ψ en varios factores de potencias de 2.

4. Presentación de resultados

Comenzamos la sección de Presentación de resultados con un ejemplo aplicado de los algoritmos que hemos descrito anteriormente. El caso práctico que vamos a tratar corresponde con una competición publicada en 2021 por HexagonML en colaboración con KDD.

4.1 Competición

En los últimos años se ha visto una explosión en el número de artículos académicos ligados a la detección de anomalías en series temporales, mayoritariamente debido al éxito de técnicas de *deep learning* en otros campos y en otras tareas de series temporales. La mayoría de estos artículos basan sus resultados en *datasets "benchmark"*, proporcionados por NASA, Yahoo, ... entre otros. Sin embargo, estos *datasets* tienen por lo menos uno de cuatro defectos [31]; estos pueden ser la trivialidad de resolverlos, la densidad de *outliers* irrealista, la mala etiquetación de los datos, o el hecho de que la mayoría de *outliers* se encuentren al final de los *datasets* de test.

Por tanto, es muy difícil otorgar validez a los resultados obtenidos por algoritmos en estos datos. La competición de HexagonML (referencia [32]) surge con el objetivo de mitigar este problema; su propósito es encontrar una metodología o algoritmo que de forma automática y general sea capaz de detectar datos

anómalos en series temporales, proporcionando 250 *datasets* estudiados a lo largo de más de 20 años. Cada uno de estos *datasets* contiene una anomalía únicamente, situada en el conjunto de test (cuya posición en los datos conocemos) y asignada mediante un rango de valores; para obtener una respuesta correcta se permite un margen de error de +/- 100 posiciones. Es decir, si en una serie temporal la anomalía viene dada por las posiciones (1000-1100), entonces cualquier predicción entre 900 y 1200 sería correcta.

Comenzamos la parte descriptiva del análisis con algunas de las 25 primeras series para obtener cierta intuición sobre la naturaleza de los datos con los que estamos trabajando. En los gráficos posteriores, el número de la serie viene incluido como título, la línea azul determina la tendencia de la serie y la posición de comienzo del conjunto de test viene marcada por la línea roja.

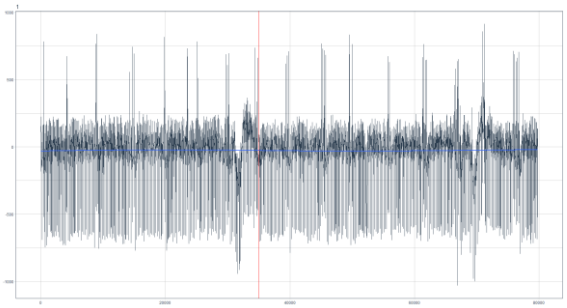

Figura 24: Serie temporal 1

En la primera serie, es evidente que es muy complicado tratar de deducir a simple vista dónde se podría encontrar el *outlier*; esto es debido en parte al gran número de datos que conforma la serie. Es cierto que se observa un descenso abrupto en forma de rampa alrededor de la posición 68000, pero como esto también es presente en la zona de train (donde no existen *outliers*), uno debería descartar esta rampa como posible *outlier*.

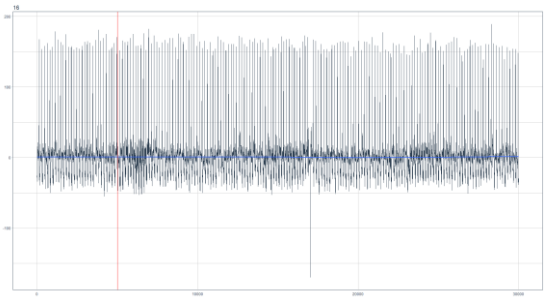

Figura 25: Serie temporal 16

Por contraste, podemos destacar la serie número 16, en la que se observa una clara presencia de un *outlier* de tipo aditivo (figura 25) en la posición 17000 aproximadamente.

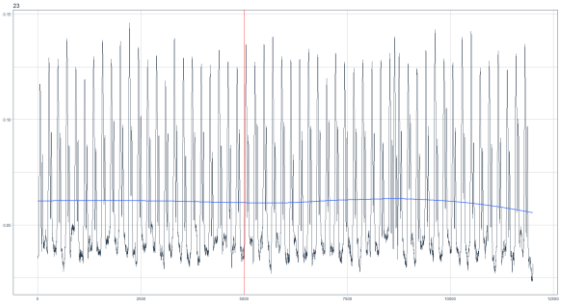

Figura 26: Serie temporal 23

Por otro lado, en la serie 23 destacamos una zona (entre 8500-9250 aproximadamente) en la que se aprecia una zona con una variación en los datos considerablemente más alta que en el resto de la serie. Podemos concluir que seguramente sea un *outlier* de tipo "ruido" (figura 26).

En el resto de series, por lo general, no es nada clara la presencia de un *outlier*, debido al gran número de observaciones, la alta variabilidad, y la poca apreciación de periodicidad en las series. En cuanto a las tendencias, si una fuese predominantemente creciente o decreciente, podríamos deducir que un valor muy diferenciado de esta tendencia podría ser un buen candidato a *outlier*; esto no ocurre ya que todas las series tienen una tendencia por lo general constante.

4.2 Software

En cuanto a los recursos con los que se ha desarrollado este trabajo, cabe destacar que el lenguaje de programación utilizado ha sido R y en un ordenador portátil HP EliteBook 840 G5 con un procesador Intel(R) Core(TM) i5-8250U (CPU @ 1.60GHz 1.80 GHz) y 7.85 GB de RAM usables. En el Apéndice D se pueden encontrar las funciones de R que han sido usadas, con sus respectivos paquetes.

4.3 Aplicación

En este capítulo se presentan los resultados de aplicar las metodologías propuestas en el marco teórico a las series temporales proporcionadas por la competición, además de una comparativa entre todos ellos. Hemos decidido trabajar con las primeras 200 series en vez del total de 250 debido a la limitante capacidad computacional del equipo mencionado anteriormente (algunas de las series tenían más de un millón de datos). En el Apéndice D se incluyen las rutinas utilizadas que presentamos a continuación.

4.3.1 Resultados obtenidos con Matrix Profile

En primer lugar, mostramos el producto de aplicar el algoritmo *Matrix Profile*. El único parámetro de este algoritmo es el *window* (tamaño de los vectores con los que se calculan las distancias). Primeramente, se asignó un valor a este parámetro de 100 (asignación razonable considerando que la competición admitía un rango de predicción de +/- 100 posiciones). La tasa de acierto lograda fue de 0.505 sobre 1, acertando en 101 de las 200 series.

Tratamos de mejorar este resultado construyendo un algoritmo de elaboración propia, implementando una *grid* del hiperparámetro *window* para valores entre 100 y 200 de cinco en cinco (de nuevo debido a las condiciones ofrecidas por la competición, parecía lógico considerar que los mejores resultados se

obtendrían buscando *outliers* de longitud entre 100 y 200). A continuación se muestra el pseudo-código de este algoritmo y posteriormente incluimos su explicación.

Algoritmo 3 (Implementación metodología Matrix Profile):

$def\ MatrixProfile(datos, len\_100\_interval)$:

1. $for\ i\ in\ 1:200:$
2. $data = datos[i]$
   $for\ k\ in\ seq(100, 200, 5)$:
   $posicion\_outlier = matrixprofile\_outlier(data, windows = k)$
   $posiciones = append(posiciones, posicion\_outlier)$
3. $for\ j\ in\ len\_100\_interval$:
   $contador = count(posiciones\ in\ j)$
   $num\_outliers\_intervalos = append(num\_outliers\_intervalos, contador)$
4. $intervalo\_optimo = \max(num\_outliers\_intervalos)$
   $prediccion = \max(intervalo\_optimo)$

Para cada serie temporal, se ve dónde se encuentra el *outlier* predicho para cada valor de *window* (paso 2) y en intervalos de longitud 100 se busca aquel intervalo con mayor número de *outliers* predichos, según el valor de *window* (paso 3). Entonces, se toma como predicción final el mayor *outlier* de este intervalo (paso 4). De esta forma, conseguimos obtener una tasa de acierto de 0.715 (143/200).

4.3.2 Resultados obtenidos con ARIMA

Continuamos este capítulo con los resultados ofrecidos por la metodología ARIMA. En esta sección analizaremos la estacionariedad de algunas de las series y posteriormente realizaremos un estudio del acierto a priori que presentará el algoritmo basándonos en la asignación que da en ciertas de las series con *outlier* más y menos evidente.

Test Dickey-Fuller

El test de Dickey-Fuller determina si es necesario o no diferenciar la serie para hacer que sea estacionaria en media. En R, este contraste se lleva a cabo mediante la función *adf.test()*.

Los resultados obtenidos de aplicar esta función a las series son un p-valor de 0.01 en todos los casos; por tanto, a un nivel de significación de 0.05, rechazamos la hipótesis nula del contraste y concluimos que no es necesario diferenciar ninguna de las series.

Como el test de Dickey-Fuller es poco potente (el resultado que ofrece muchas veces no concuerda con la realidad sugerida por un estudio más concienzudo de la serie) cuando ajustemos la serie con la función *auto.arima()*, se dará la opción de que el algoritmo decida si es necesario diferenciar las series para optimizar los coeficientes AIC, BIC y AICc.

Box-Cox

En cuanto a la necesidad de realizar una transformación para conseguir estacionariedad en varianza, los resultados obtenidos en el test de Box-Cox sugieren que unas pocas series precisan de una transformación logarítmica ($\lambda = 1$) o transformación de raíz cuadrada ($\lambda = 0.5$) y en la gran mayoría de casos no es necesario realizar ninguna transformación ($\lambda = 0$).

Esto será utilizado más adelante cuando hagamos la rutina de identificación de los *outliers*, transformando la serie apropiadamente antes de realizar el ajuste automático.

Estacionalidad

La estacionalidad de una serie temporal puede ser obtenida en R mediante la función *findfrequency()*, que no tiene más parámetros que la serie temporal en cuestión.

Los resultados obtenidos para las series varían mucho, tomando valores de periodo como 1, 24, 91, 250 o 499. Esto da a entender que hay series con una irregularidad muy clara y presente (valores de periodo como 1, 250 o 499) y otras series que pueden ser más regulares (un valor de 24 es un indicador de serie horaria). A continuación, se evaluará si esto es cierto con un par de ejemplos; decidimos mostrar la serie número 5 con periodo igual a 24 como serie regular, y la serie número 1 con periodo igual a 250 como serie irregular.

Las gráficas se han obtenido cogiendo intervalos de datos de longitud k, siendo k el periodo obtenido. En primer lugar estudiaremos la serie número 5, con los periodos de datos 1-24, 25-48, 49-72 y 73-96.

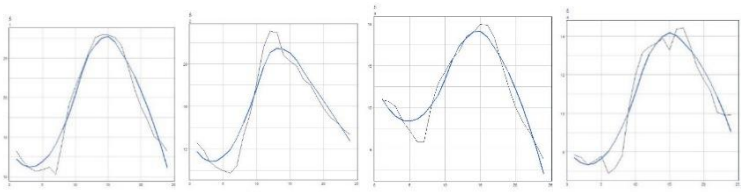

Figura 27: Patrones serie temporal 5

Como vemos, se observa un patrón con forma de parábola muy similar en los 4 intervalos, indicando que en efecto la serie tiene un periodo donde el patrón en los datos se repite cada 24 datos.

A continuación, mostramos los intervalos de la serie número 1, de periodo 250.

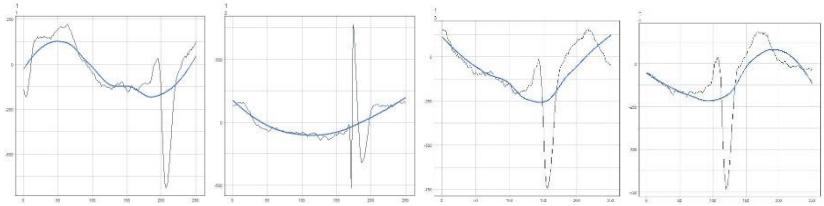

Figura 28: Patrones serie temporal 1

En este caso, es claro que el patrón en los distintos intervalos es mucho menos evidente. Puede apreciarse cierta similitud en los datos de la tercera y cuarta figura; este patrón no está tan presente en la primera figura por el claro crecimiento en el comienzo de la serie, y desde luego el patrón es muy poco reconocible en la segunda gráfica.

Por tanto, los resultados obtenidos sugieren que las series con periodo 24 son series horarias con un patrón regular y las series con otros periodos son series más irregulares.

Tipos de outliers

Para estudiar si a priori el modelo ARIMA podría ofrecer buenos resultados, mostraremos algunos de los *outliers* que esta metodología identifica e interpretaremos los resultados cuando sea posible (sin tener en cuenta la posición del *outlier* real).

En el análisis descriptivo previo habíamos destacado la serie número 16 por presentar un *outlier* de tipo aditivo muy claro; en las figuras siguientes se marcará con una línea de color la posición en la que se encuentra en *outlier*, con color dependiendo del tipo que sea, veamos si en este caso lo identifica:

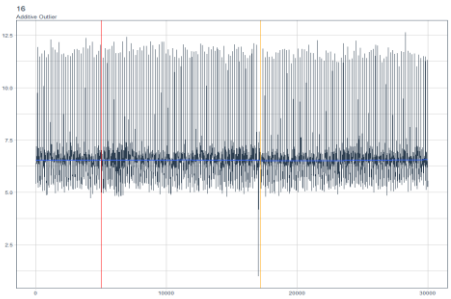

Figura 29: Serie temporal 16 con *outlier*

Como podemos observar, identifica correctamente este *outlier* aditivo. Un caso no tan claro de esta identificación la podemos encontrar por ejemplo en la serie número 1:

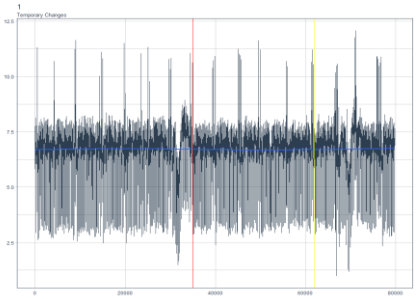

Figura 30: Serie temporal 1 con *outlier*

En este gráfico podemos apreciar que no es muy fácil concluir si la identificación del *outlier* se ha realizado correctamente o no, ya que no está claro que los puntos que sobresalen mucho después del *outlier* seleccionado sean en realidad el verdadero *outlier*. Deducimos por tanto que es probable que este algoritmo sí consiga asignar correctamente la posición del *outlier* en los casos más evidentes, pero no podemos decir lo mismo del resto de series. Asumiendo que en las series con un gran número de datos los *outliers* serán menos claros y teniendo en cuenta la tasa de acierto de la primera versión de la rutina del *Matrix Profile* (0.505) podemos intuir que el ARIMA no logrará un éxito sorprendente.

En la rutina de este algoritmo se estandarizan los datos, se realiza la transformación adecuada de acuerdo con la metodología Box-Cox, se ajustan los parámetros del modelo ARIMA (función *auto.arima()*), y posteriormente se calculan los *outliers* de acuerdo a la función *locate.outliers()*. Esta función por defecto busca *outliers* de tipo “AO”, “LS”, y “TC” y depende del parámetro *cval*, que determina el valor crítico que tienen que superar los estadísticos de los contrastes que se realizan para ser considerados como *outliers*.

La cuestión con este parámetro es que para un valor de *cval* muy pequeño, habrá muchos candidatos a *outliers* y por tanto el tiempo de ejecución del programa será muy largo; y para un valor de *cval* muy grande, correremos el riesgo de que no se encuentre ningún candidato a *outlier*. Por tanto, para determinar un valor de *cval* correcto, seleccionaremos algunas series y veremos a partir de qué valor no se identifica ningún *outlier* en alguna de las series.

Se escogieron nueve series para este análisis. El resultado obtenido fue que con un valor de *cval* de 3.5 se identifica un outlier en todas las series, pero con un valor de 4 no. Para asegurarnos que se encuentre algún *outlier* en todas las series, fijaremos a partir de ahora el valor de *cval* a 3.

Hay dos motivos por los cuales se ha decidido realizar el ajuste de los parámetros del modelo ARIMA automáticamente: el primero es la naturaleza de la competición en la que estamos trabajando, cuyo objetivo es encontrar una metodología que de forma automática y generalizada encuentre *outliers* en series temporales; el segundo motivo es el volumen de datos de las series, que como mencionamos anteriormente, era demasiado alto para ser computado por el equipo informático disponible.

Realizando el ajuste automático de ARIMA como se ha indicado previamente, localizando el *outlier* predicho y con la posición válida de *outlier* sugerida por la competición, obtuvimos una tasa de acierto de 0.405 (81/200).

4.3.3 Resultados obtenidos con Isolation Forest

La función en R que realiza este algoritmo pertenece al paquete *h2o* y en particular es la función *h2o.isolationForest()*. Esta función recibe como argumentos los datos con los que se va a entrenar el modelo, el parámetro *contamination*, que indica la tasa de anomalías en el conjunto de datos de entrenamiento y el número de árboles que conforman el *Isolation Forest*.

El algoritmo de *Isolation Forest* ha sido desarrollado entrenando el modelo con todos los datos y prediciendo en el conjunto de test, con el parámetro de *contamination* indicando que solo hay 1 *outlier* y entrenado 500 árboles. En ciertos conjuntos de datos, debido a su alto volumen, no podían entrar en memoria entrenando 500 árboles; por tanto, se decidió que para aquellos datos con más de 100.000 observaciones, no se hiciese predicción (se cuentan como fallo). Se ha decidido entrenar el modelo con todos los datos ya que entrenando en el conjunto de entrenamiento hubiera sido muy importante indicar que no había ninguna anomalía en este conjunto, y la función no ofrecía esta posibilidad. Esta rutina no ofrece buenos resultados, acertando solo en 31 de las 200 series (tasa de 0.155).

4.3.4 Resultados obtenidos con wavelets

Por último, decidimos incluir un algoritmo propio basado en la metodología de las *wavelets* con el fin de localizar aquellos *outliers* más evidentes, que, de acuerdo a la teoría, corresponderían con los datos atípicos presentes en la *wavelet* de nivel 1.

Algoritmo 4 (Implementación metodología wavelets):

$\boldsymbol{def\ wavelets(datos, num\_puntos, ventana)}$:

1. $for\ i\ in\ 1{:}200:$
2. $data = datos[i]$
   $wavelet\_nivel1 = wavelet(datos)[1]$
3. $Q1 = quantile(wavelet\_nivel1, prob = 0.25)$
   $Q3 = quantile(wavelet\_nivel1, prob = 0.75)$
   $a = Q1 - 3(Q3 - Q1)$
   $b = Q3 + 3(Q3 - Q1)$
   $intervalo\_atipico = (a, b)$
4. $outliers = (wavelet\_nivel1 < a)\ or\ (wavelet\_nivel1 > b)$
   $contador\_outliers = count(outliers)\ in\ ventana$
5. $if\ \max(contador\_outliers) \geq \min(num\_puntos, length(outliers)/2)$:
   $prediccion = outliers[which.\max(contador\_outliers)]$
   $else$:
   $prediccion = 'Dont\ know'$

En primer lugar, se cargan los datos y se calcula la *wavelet* de primer nivel (paso 2). Después, se define el intervalo de valores no atípicos proporcionado por el intervalo intercuartílico (paso 3). En el paso 4 se define como *outliers* los puntos fuera de este intervalo, y se cuentan el número de observaciones anómalas en una ventana de longitud "ventana". Por último, se impone una condición restrictiva en la que se considera si el mayor número de *outliers* obtenido en una ventana suficientemente alto como para ser asignado como predicción; en caso de no cumplirse la condición, el algoritmo no predice un *outlier* (paso 5).

Esta rutina se incluyó en las rutinas anteriores más prometedores (ARIMA, *Matrix Profile* "estándar" y *Matrix Profile* propio). En la siguiente sección se encuentran dichos resultados obtenidos y su comparativa.

4.3.5 Discusión de resultados

A continuación, mostramos una tabla comparando el acierto en las 200 series proporcionado por el ajuste ARIMA, las rutinas del *Matrix Profile* y sus respectivas versiones incluyendo la metodología de *wavelets*. En el Apéndice C se encuentran los aciertos y fallos de cada modelo en cada una de las 200 series.

	ARIMA	Matrix Profile	Matrix Profile propio
Sin wavelet	0,405	0,505	0,715
Con wavelet	0,42	0,54	0,72

Figura 31: Resultados obtenidos en las 200 series

Como se puede observar, el mejor algoritmo lo constituye el algoritmo propio de *Matrix Profile* unido con la metodología de *wavelets*, con un acierto de 0.72 (una serie de mejora sin incluir *wavelets*). La mejora es más considerable en el *Matrix Profile* estándar, aumentando el acierto en siete series. Para el caso del ARIMA, resulta en tres series más.

Comparando las tres metodologías sin *wavelets* entre sí, se tiene que el ARIMA acertó en 81/200 series, el *Matrix Profile* en 101 (veinte series más), y a su vez el algoritmo propio de *Matrix Profile* identificó el *outlier* exitosamente en 143 de las 200 series, resultando en 42 series más que su predecesor.

5. Conclusiones

Se han presentado y descrito (detalladamente en algunos casos) algunas de las metodologías más relevantes en la detección de *outliers* en series temporales, mostrando las matemáticas que sustentan el desarrollo de éstas y aplicando los conocimientos adquiridos en la KDD 2021.

Dicha competición tenía como objetivo representar un nuevo *benchmark* en la detección de *outliers* en series temporales. Valorando la tasa de acierto obtenida con el algoritmo ARIMA (0.405), podemos afirmar que no era fácil identificar el dato anómalo en la gran mayoría de las series. Teniendo esto en cuenta, la tasa de acierto final obtenida ha resultado ser considerablemente alta (0.72), en especial si se compara con la del algoritmo *Matrix Profile* estándar (0.54).

Por otro lado, también se ha conseguido corroborar la alta eficiencia de los algoritmos basados en la metodología *Matrix Profile* que sugiere la teoría. Si bien la ejecución del código era más lenta que la del ARIMA, no hubo problemas de espacio de memoria, cosa que sí ocurrió en el *Isolation Forest*, cosa aún más remarcable si se tiene en cuenta que en un principio el *Matrix Profile* está calculando distancias euclídeas entre vectores, lo cual debería ser mucho más demandante que realizar separaciones de los datos y ver cuál de ellos necesita menos divisiones para ser aislado.

Querría concluir este trabajo con una nota personal, en la que expreso mi satisfacción por el trabajo realizado, el resultado final obtenido y el esfuerzo que ha conllevado. Para mí ha sido una experiencia muy enriquecedora haber ampliado de esta manera mis conocimientos en el ámbito de la detección de *outliers* en series temporales, trabajando en temas tanto de análisis de variable compleja, estadística y computación y, como resultado a destacar, profundizando en uno de los algoritmos más importantes del siglo XX, la Transformada Rápida de Fourier. Me gustaría además agradecer a mi tutor por sus oportunas consideraciones y atento seguimiento.

Referencias bibliográficas

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. The MIT Press, 2001.

[2] Liu, Fei Tony & Ting, Kai & Zhou, Zhi-Hua. Isolation Forest. ICDM 2008, 413 – 422. 2009

[3] B. R. Preiss. Data Structures and Algorithms with Object-Oriented Design Patterns in Java. Wiley, 1999.

[4] Donald E. Knuth. The Art of Computer Programming, Volume 3, Sorting and Searching. Addison-Wesley Professional, 1998.

[5] E. Oran Brigham. The Fast Fourier Transform, An Introduction to Its Theory and Application. Prentice Hall, 1973.

[6] Ronald N. Bracewell. The Fourier Transform And Its Applications. McGraw Hill, 2000.

[7] Julius O. Smith. Mathematics of the Discrete Fourier Transform. W3K Publishing, 2003.

[8] Steven B. Damelin, Willard Miller. The Mathematics of signal processing. Cambridge University Press, 2012.

[9] Yeh, Chin-Chia Michael & Zhu, Yan & Ulanova, Liudmila & Begum, Nurjahan & Ding, Yifei & Dau, Anh & Silva, Diego & Mueen, Abdullah & Keogh, Eamonn. *Matrix Profile* I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. IEEE ICDM 2016.

[10] T. Rakthanmanon et al. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. 2012.

[11] Lawrence R. Rabiner, Bernard Gold. Theory and Application of Digital Signal Processing. Prentice Hall. 1975.

[12] MASS: http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

[13] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Berisk and Eamonn Keogh. *Matrix Profile* II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. EEE ICDM 2016.

[14] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar and Eamonn Keogh. Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speed. ICDM 2018.

[15] W. Hardle et al. *Wavelets*, Approximation and Statistical Applications. New-York: Springer-Verlag. 1998.

[16] Todd Ogden. Essential *Wavelets* for Statistical Applications and Data Analysis. Birkhauser, Basel. 1997.

[17] Donald B. Percival, Andrew T. Walden. *Wavelets* Methods for Time Series Analysis. Cambridge University Press. 2013.

[18] Canan Bilen, S. Huzurbazar. *Wavelet*-Based Detection of Outliers in Time Series. Journal of Computational and Graphical Statistics 11, no. 2, 311-327. 2002.

[19] Chavan, Mahesh & Mastorakis, Nikos & Chavan, Manjusha & Gaikwad, M. Implementation of SYMLET *wavelets* to removal of Gaussian additive noise from speech signal. 2011.

[20] Braei, Mohammad & Wagner, Dr.-Ing. ANOMALY DETECTION IN UNIVARIATE TIME-SERIES: A SURVEY ON THE STATE-OF-THE-ART. 2020.

[21] Amarbayasgalan, Tsatsral & Pham, Van-Huy & Theera-Umpon, Nipon & Ryu, Keun. Unsupervised Anomaly Detection Approach for Time-Series in Multi-Domains Using Deep Reconstruction Error. Symmetry, 12, 1251. 2020.

[22] Zimmerman, Zachary & Kamgar, Kaveh & Shakibay Senobari, Nader & Crites, Brian & Funning, Gareth & Brisk, Philip & Keogh, Eamonn. *Matrix Profile* XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond. SoCC. 2019.

[23] Zachary Zimmerman, Nader Shakibay Senobari, Gareth Funning, Evangelos Papalexakis, Samet Oymak, Philip Brisk, and Eamonn Keogh. Matrix Profile XVIII: Time Series Mining in the Face of Fast Moving Streams using a Learned Approximate *Matrix Profile*. IEEE ICDM 2019.

[24] Sara Alaee, Ryan Mercer, Kaveh Kamgar, Eamonn Keogh. *Matrix Profile* XXII: Exact Discovery of Time Series Motifs under DTW. ICDM 2020.

[25] Chung Chen, Lon-Mu Liu. Joint Estimation of Model Parameters and Outlier Effects in Time Series. Journal of the American Statistical Association, Vol. 88, No. 421, 284-297. 1993.

[26] Kauffman, Sean & Dunne, Murray & Gracioli, Giovani & Khan, Waleed & Benann, Nirmal & Fischmeister, Sebastian. Palisade: A Framework for Anomaly Detection in Embedded Systems. Journal of Systems Architecture, 113. 2020.

[27] Box, G., Tiao, G. Intervention Analysis with Applications to Economic and Environmental Problems. Journal of the American Statistical Association. Vol. 70, No. 349. 70-79. 1975.

[28] Chang, I., Tiao, G. Estimation of Time Series Parameters in the Presence of Outliers. Technical Report 8, University of Chicago, Graduate School of Business. 1982

[29] Tsay, R. Time Series Model Specification in the Presence of Outliers. Journal of the American Statistical Association. Vol. 81, No. 393, 132-141. 1986.

[30] Daniel Peña. Análisis de Series Temporales. Alianza Editorial. 2010.

[31] Wu, Renjie, Keogh, Eamonn. Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. 2020.

[32] Keogh, E., Dutta Roy, T., Naik, U. & Agrawal, A (2021). Multi-*dataset* Time-Series Anomaly Detection Competition, SIGKDD 2021. https://compete.hexagon-ml.com/practice/competition/39/

Anexo

Apéndice A

Definiciones ARIMA

Se dan a continuación las definiciones de proceso estacionario, ruido blanco y modelo ARIMA.

Definición (Proceso débilmente estacionario):

Dado un proceso estocástico X_t, éste es estacionario en sentido débil (desde ahora, estacionario) si:

1. $E[X_t] = \mu < \infty, \forall\, t$

2. $V[X_t] = \gamma_0 < \infty, \forall\, t$

3. $Cov[X_t, X_{t+k}] = \gamma_k, \forall\, t, \forall\, k$

Un caso particular de proceso estacionario de interés en las series temporales es el de Ruido Blanco, cuya definición se incluye seguidamente.

Definición (Ruido Blanco):

Un proceso estocástico a_t, se dice que es Ruido Blanco si cumple:

1. $E[a_t] = 0, \forall\, t$

2. $V[a_t] = \sigma_a^2 < \infty, \forall\, t$

3. $Cov[a_t, a_{t+k}] = 0, \forall\, t, \forall\, k \neq 0$

A continuación, damos la definición de un modelo ARIMA.

Definición (Modelo ARIMA):

Dado una serie temporal X_t, un modelo ARIMA(p,d,q) viene representada por la ecuación con la siguiente estructura:

$$\phi(B)(1-B)^d X_t = \theta(B) a_t$$

Donde

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q$$

Apéndice B

Definiciones Isolation Forest

Se presentan las definiciones de árbol binario de búsqueda y árbol de aislamiento.

Definición (Árbol binario de búsqueda):

Un árbol binario de búsqueda o BST es un árbol cuya división de nodos se construye recursivamente de forma que el nodo siguiente izquierdo tiene un valor más pequeño que el anterior, y el nodo siguiente derecho tiene un valor más grande que el anterior; se muestra un ejemplo en la siguiente figura:

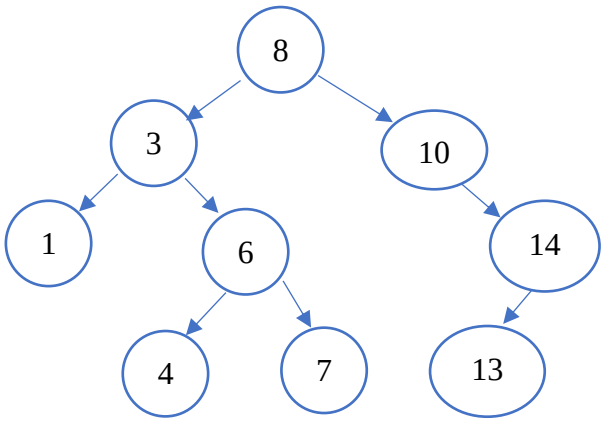

Figura: Ejemplo de árbol binario de búsqueda

Este tipo de estructura de datos sirve para optimizar muchos algoritmos informáticos, como algoritmos de ordenación, búsqueda, inserción o eliminación, por citar algunos ejemplos.

Otro concepto que se utilizará es que, en esta estructura de datos, un nodo externo se denomina “búsqueda fallida”.

Definición (Árbol de aislamiento):

Un árbol de aislamiento o *iTree* es el resultado de aplicar un algoritmo de separación de datos a cierto conjunto de observaciones. Este algoritmo tiene el objetivo de identificar datos anómalos y lo consigue haciendo que las separaciones sean aleatorias.

Veamos la intuición detrás de este enfoque:

Figura: Representación

Supongamos que estos círculos representan los valores unidimensionales de un conjunto de datos. Nuestro objetivo es identificar exitosamente la observación anómala (que en este caso es la primera) mediante separaciones consecutivas de los datos. Al ser la separación aleatoria, es decir, uniformemente distribuida, será más probable que la primera observación quede separada del resto en un menor número de

separaciones. Por tanto, se determinarán como observaciones anómalas aquellas que hayan sido separadas en un menor número de iteraciones.

Es importante destacar en este momento la similitud en la estructura de un árbol de aislamiento con un árbol binario de búsqueda; ambos realizan sucesivos cortes de manera que a cada lado de cada nodo quedan observaciones menores que el nodo superior al lado izquierdo, y mayores que el nodo superior en el lado derecho. Por tanto, cuando calculemos más adelante la estimación de la longitud media del camino en los nodos externos de un *iTree*, la expresión será equivalente a la de un nodo externo en un árbol binario de búsqueda.

Apéndice C

Tabla de resultados completa

Modelos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
ARIMA	0	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1	1	1	1
Matrix Profile estándar	0	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	0	0	0	0	0	1	1
Matrix Profile propio	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ARIMA+wavelets	0	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1	1	1	1
MP estándar+wavelets	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	1	1	1
MP propio+wavelets	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
0	1	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1
1	0	1	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0	1	0	0	1	0	1	1	1	0	0	1	1	0	0
0	1	1	1	1	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1
0	1	1	1	0	0	1	0	0	1	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0
0	1	1	1	1	1	1	0	0	1	0	1	0	1	1	0	1	1	1	0	0	1	1	0	0

51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	1	1
1	1	0	1	1	0	1	1	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	1	1
1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	1	1
1	1	0	1	1	0	1	1	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	1	1

76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	1	0	1
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	1	0	1
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0
0	0	1	0	0	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125
0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	0	1	0	1	1
0	0	1	0	0	0	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	0	1	0	1	1
0	0	1	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1
1	0	1	1	0	1	0	0	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1

126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
1	0	0	0	0	0	1	1	0	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	0
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1	0	1	0	0	0	0	1	0	1	0	0	0
1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0
1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1	0

151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
1	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	0
0	1	0	1	1	1	1	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	1	0
0	1	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1	1	1	0	1	0
1	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	0
0	1	0	1	1	1	1	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	1	0
1	0	1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	0	1	1	1	0	1	0	1

176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1
1	1	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Apéndice D

Paquetes y funciones utilizadas

```
$`c("package:h2o", "package:base")`
[1] "colnames"   "is.numeric" "log"        "round"

$`c("package:h2o", "package:stats")`
[1] "sd"

$`c("package:plotly", "package:dplyr")`
[1] "arrange"

$`character(0)`
[1] "analisisWavelet"

$`package:base`
 [1] "abs"           "append"        "as.character"  "as.data.frame" "as.integer"    "as.vector"     "c"
 [8] "cbind"         "data.frame"    "dim"           "file.path"     "length"        "library"       "list.files"
[15] "ls"            "matrix"        "max"           "mean"          "min"           "nchar"         "numeric"
[22] "paste"         "print"         "rep"           "return"        "rm"            "row.names"     "seq"
[29] "seq_along"     "setwd"         "sort"          "strsplit"      "subset"        "substr"        "sum"
[36] "which.max"

$`package:dplyr`
[1] "desc"

$`package:forecast`
[1] "auto.arima"    "findfrequency"

$`package:ggplot2`
[1] "geom_vline" "labs"

$`package:grDevices`
[1] "dev.off" "jpeg"

$`package:h2o`
[1] "as.h2o"              "h2o.init"            "h2o.isolationForest" "h2o.predict"

$`package:MASS`
[1] "boxcox"

$`package:NCmisc`
[1] "list.functions.in.file"

$`package:stats`
[1] "as.ts"    "quantile"

$`package:timetk`
[1] "plot_time_series"

$`package:tsmp`
[1] "analyze"

$`package:tsoutliers`
[1] "coefs2poly"      "locate.outliers"

$`package:utils`
[1] "read.csv" "View"

$`package:wmtsa`
[1] "wavDWT" "wavMRD"
```

Código utilizado

ARIMA

```
result <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header=FALSE)
  datos$ID <- seq_along(datos[, 1])
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)
  datos$V1 <- datos$V1 + 1+abs(min(datos$V1))

  freq <- findfrequency(datos$V1)

  box_cox <- boxcox(V1 ~ ID,
                    data = datos,
                    lambda = c(0, 0.5, 1))
  lambda <- box_cox$x[which.max(box_cox$y)]

  if (1-lambda < 0.5-lambda) {
    datos$target_transformado <- log(datos$V1)
  }
  else if (-lambda < 0.5-lambda) {
    datos$target_transformado <- datos$V1
  }
  else {
    datos$target_transformado <- (datos$V1)^lambda
  }

  datos.ts <- as.ts(datos$target_transformado, frequency = freq)


  ajuste_automatico <- auto.arima(datos.ts,
                                  max.d = 1, max.D = 1,
                                  max.p = 2, max.P = 2,
                                  max.q = 2, max.Q = 2,
                                  seasonal = TRUE,
                                  ic = "aic",
                                  allowdrift = FALSE)
  lista_outliers <- locate.outliers(ajuste_automatico$residuals,
                                    pars = coefs2poly(ajuste_automatico),
                                    types = c("AO", "LS", "TC"), cval = 3)

  lista_outliers$abststat <- abs(lista_outliers$tstat)
  lista_outliers <- arrange(lista_outliers,desc(lista_outliers$abststat))

  outlier_pos <- lista_outliers[lista_outliers$ind >= pos_test[i],][1, "ind"]
  resultado <- (outlier_pos < target_matrix[i, 2]) & (outlier_pos > target_matrix[i, 1])
  total[i] <- resultado
}
```

Matrix Profile estándar

```
result <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header = FALSE)
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)
  model <- analyze(datos$V1, windows = 100)
  discords <- model$discord

  for (j in discords$discord_idx) {
    if (j >= pos_test[i]) {
      prediccion <- j
      break
    }
    else{
      prediccion <- 'no encuentra outlier en test'
    }
  }

  if ((prediccion <= target_matrix[i, 2]) & (prediccion >= target_matrix[i, 1])) {
    result[i] <- 1
  }
  else{
    result[i] <- 0
  }
}
```

Matrix Profile propio

```
result <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header = FALSE)
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)

  posiciones <- numeric()
  for (k in seq(100,200,5)){
    model <- analyze(datos$V1, windows = k)
    discords <- model$discord
    for (j in discords$discord_idx) {
      if (j >= pos_test[i]) {
        prediccion <- j
        break
      }
      else{
        prediccion <- 'no encuentra outlier en test'
      }
    }
    if (is.numeric(prediccion) == TRUE){
      posiciones <- append(posiciones, prediccion)
    }
  }
  if (length(posiciones) == 0) {
    result[i] <- 0
  }
  else {
    posiciones <- sort(posiciones)
    numero_outliers <- numeric()
    a = min(posiciones)
    b = max(posiciones)
    w = 0
    while (a+w*100 < b){
      numero_outliers = append(numero_outliers,
                               sum(posiciones >= a+w*100 & posiciones <= a + (w + 1)*100))
      w = w+1

    }

    indice <- which.max(numero_outliers)
    posiciones_validas <- posiciones[(posiciones >= a + (indice-1)*100) & (posiciones <= a + indice*100)]
    prediccion <- max(posiciones_validas)

    if ((prediccion <= target_matrix[i, 2]) & (prediccion >= target_matrix[i, 1])) {
      result[i] <- 1
    }
    else{
      result[i] <- 0
    }
  }
}
```

Wavelets

```
analisis_wavelet <- function(serie, numero_puntos, longitud_ventana){
  serie.dwt <- wavDWT(serie, n.levels = 6)
  MRD <- wavMRD(serie.dwt)
  D_1 <- MRD[["D1"]]

  Q_1 <- quantile(D_1, probs=0.25)
  Q_3 <- quantile(D_1, probs=0.75)
  sup_outliers <- Q_3 + 3*(Q_3 - Q_1)
  inf_outliers <- Q_1 - 3*(Q_3 - Q_1)

  D1_df <- as.data.frame(D_1)
  D1_df <- cbind(D1_df, seq(1:length(serie)))
  resultado <- subset(D1_df, D_1 > sup_outliers | D_1 < inf_outliers)
  resultado <- arrange(resultado, desc(abs(D_1)))
  colnames(resultado) <- c("Valor","Observacion")

  contador_atipicos <- rep(-1, min(numero_puntos, length(resultado$Valor)))

  if (length(resultado$Valor) != 0) {
    for (i in 1:min(numero_puntos, length(resultado$Valor))) {
      for (j in 1:min(numero_puntos, length(resultado$Valor))) {
        if ((resultado$Observacion[i] <= resultado$Observacion[j] + longitud_ventana)
            & (resultado$Observacion[i] >= resultado$Observacion[j] - longitud_ventana)) {
          contador_atipicos[i] <- contador_atipicos[i]+1
        }
      }
    }
  }

  if (length(resultado$Valor) != 0) {
    if (max(contador_atipicos) >= min(numero_puntos, length(resultado$Valor))/2) {
      return(resultado$Observacion[which.max(contador_atipicos)])
    }
    else{
      return(c("Dont know"))
    }
  }
  else{
    return(c("Dont know"))
  }
}
```

ARIMA con wavelets

```
total <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header=FALSE)
  datos$ID <- seq_along(datos[, 1])
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)
  datos$V1 <- datos$V1 + 1+abs(min(datos$V1))

  x <- datos[,1]
  prediccion_wavelet <- analisis_wavelet(x, numeroPuntos=20, longitudVentana=200)

  freq <- findfrequency(datos$V1)

  box_cox <- boxcox(V1 ~ ID,
                    data = datos,
                    lambda = c(0, 0.5, 1))
  lambda <- box_cox$x[which.max(box_cox$y)]

  if (1-lambda < 0.5-lambda) {
    datos$target_transformado <- log(datos$V1)
  }
  else if (-lambda < 0.5-lambda) {
    datos$target_transformado <- datos$V1
  }
  else {
    datos$target_transformado <- (datos$V1)^lambda
  }

  datos.ts <- as.ts(datos$target_transformado, frequency = freq)

  ajuste_Automatico <- auto.arima(datos.ts,
                                  max.d = 1, max.D = 1,
                                  max.p = 2, max.P = 2,
                                  max.q = 2, max.Q = 2,
                                  seasonal = TRUE,
                                  ic = "aic",
                                  allowdrift = FALSE)
  lista_Outliers <- locate.outliers(ajuste_Automatico$residuals,
                                    pars = coefs2poly(ajuste_Automatico),
                                    types = c("AO","LS","TC"), cval = 3)

  lista_Outliers$abststat <- abs(lista_Outliers$tstat)
  lista_Outliers <- arrange(lista_Outliers,desc(lista_Outliers$abststat))

  outlier_pos <- lista_Outliers[lista_Outliers$ind >= pos_test[i],][1, "ind"]

  if (is.numeric(prediccion_wavelet) == TRUE) {
    outlier_pos <- prediccion_wavelet
  }

  resultado <- (outlier_pos < target_matrix[i, 2]) & (outlier_pos > target_matrix[i, 1])
  total[i] <- resultado
}
```

Matrix Profile estándar con wavelets

```
result <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header = FALSE)
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)

  x <- datos[,1]
  prediccion_wavelet <- analisis_wavelet(x, numeroPuntos=20, longitudVentana=200)

  model <- analyze(datos$V1, windows = 100)
  discords <- model$discord

  for (j in discords$discord_idx) {
    if (j >= pos_test[i]) {
      prediccion <- j
      break
    }
    else{
      prediccion <- 'no encuentra outlier en test'
    }
  }
  if (is.numeric(prediccion_wavelet) == TRUE) {
    prediccion <- prediccion_wavelet
  }
  if ((prediccion <= target_matrix[i, 2]) & (prediccion >= target_matrix[i, 1])) {
    result[i] <- 1
  }
  else{
    result[i] <- 0
  }
}
```

Matrix Profile propio con wavelets

```
result_final <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header = FALSE)
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)

  x <- datos[,1]
  prediccion_wavelet <- analisis_wavelet(x, numeroPuntos=20, longitudVentana=200)

  posiciones <- numeric()
  for (k in seq(100,200,5)){
    model <- analyze(datos$V1, windows = k)
    discords <- model$discord
    for (j in discords$discord_idx) {
      if (j >= pos_test[i]) {
        prediccion <- j
        break
      }
      else{
        prediccion <- 'no encuentra outlier en test'
      }
    }
    if (is.numeric(prediccion) == TRUE){
      posiciones <- append(posiciones, prediccion)
    }
  }

  if (length(posiciones) == 0) {
    if (is.numeric(prediccion_wavelet) == TRUE){
      prediccion <- prediccion_wavelet
      if ((prediccion <= target_matrix[i, 2]) & (prediccion >= target_matrix[i, 1])) {
        result_final[i] <- 1
      }
      else{
        result_final[i] <- 0
      }

    }
    else {
      result_final[i] <- 0
    }
  }

  else {
    posiciones <- sort(posiciones)
    numero_outliers <- numeric()
    a = min(posiciones)
    b = max(posiciones)
    w = 0
    while (a+w*100 < b){
      numero_outliers = append(numero_outliers,
                               sum(posiciones >= a+w*100 & posiciones <= a + (w + 1)*100))
      w = w+1
    }

    indice <- which.max(numero_outliers)
    posiciones_validas <- posiciones[(posiciones >= a + (indice-1)*100) & (posiciones <= a + indice*100)]
    prediccion <- max(posiciones_validas)

    if (is.numeric(prediccion_wavelet) == TRUE) {
      prediccion <- prediccion_wavelet
    }

    if ((prediccion <= target_matrix[i, 2]) & (prediccion >= target_matrix[i, 1])) {
      result_final[i] <- 1
    }
    else{
      result_final[i] <- 0
    }
  }

}
```

Isolation Forest

```
result <- numeric()
for (i in 1:200) {
  datos <- read.csv(files[i], header = FALSE)
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)

  if (dim(datos)[1] >= 100000){
    result[i] <- 0
  }

  else{
    train <- as.data.frame(datos[1 : (pos_test[i] - 1),]); colnames(train) <- c("V1")
    test <- as.data.frame(datos[pos_test[i] : dim(datos)[1],]); colnames(test) <- c("V1")
    datos.h2o <- as.h2o(datos)
    train.h2o <- as.h2o(train)
    test.h2o <- as.h2o(test)

    model <- h2o.isolationForest(x = "V1", training_frame = datos.h2o,
                                 seed = 1234, stopping_metric = "anomaly_score",
                                 score_each_iteration = TRUE,
                                 stopping_rounds = 3,
                                 sample_rate = 0.1,
                                 max_depth = 50, ntrees = 500,
                                 contamination = 1/dim(datos)[1])

    score <- h2o.predict(model, test.h2o)
    result_pred <- as.vector(score$score)
    score_test <- cbind(RowScore = round(result_pred, 4), test)
    outlier <- score_test[which.max(score_test$RowScore),]
    posicion <- as.integer(row.names(outlier))
    posicion <- posicion + dim(train)[1]

    if ((posicion <= target_matrix[i, 2]) & (posicion >= target_matrix[i, 1])) {
      result[i] <- 1
    }
    else{
      result[i] <- 0
    }
  }
}
```

Estudio descriptivo

Las siguientes son las rutinas utilizadas en las secciones 4.1 y 4.3.2 para crear las representaciones de las series temporales, indicando la posición de inicio de test y el outlier predicho, además de los gráficos mostrando los patrones de ciertas series según su periodograma.

Outliers

```
for (i in 1:25) {
  print(c("vuelta", i))
  datos <- read.csv(files[i], header=FALSE)
  datos$ID <- seq_along(datos[, 1])
  datos$V1 <- (datos$V1 - mean(datos$V1)) / sd(datos$V1)
  datos$V1 <- datos$V1 + 1+abs(min(datos$V1))

  box_cox <- boxcox(V1 ~ ID,
                    data = datos,
                    lambda = c(0, 0.5, 1))
  lambda <- box_cox$x[which.max(box_cox$y)]

  if (1-lambda < 0.5-lambda) {
    datos$target_transformado <- log(datos$V1)
  }
  else if (-lambda < 0.5-lambda) {
    datos$target_transformado <- datos$V1
  }
  else {
    datos$target_transformado <- (datos$V1)^lambda
  }

  datos.ts <- as.ts(datos$target_transformado, frequency = freq)

  ajuste_Automatico <- auto.arima(datos.ts,
                                  max.d = 1, max.D = 1,
                                  max.p = 2, max.P = 2,
                                  max.q = 2, max.Q = 2,
                                  seasonal = TRUE,
                                  ic = "aic",
                                  allowdrift = FALSE)
  lista_Outliers <- locate.outliers(ajuste_Automatico$residuals,
                                    pars = coefs2poly(ajuste_Automatico),
                                    types = c("AO","LS","TC"), cval = 3)


  lista_Outliers$abststat <- abs(lista_Outliers$tstat)
  lista_Outliers <- arrange(lista_Outliers,desc(lista_Outliers$abststat))

  outlier_pos <- lista_Outliers[lista_Outliers$ind >= pos_test[i],][1,c(1:2)]
```

```
  if (outlier_pos$type == 'TC') {
    ts_plot <- plot_time_series(
      .data = datos,
      .value = datos$V1,
      .date_var = 1:dim(datos)[1],
      .interactive = FALSE)
    print(ts_plot +
            geom_vline(xintercept = pos_test[i], color = "red") +
            geom_vline(xintercept = outlier_pos$ind, color = "yellow") +
            labs(title = as.character(i), subtitle = 'Temporary Changes'))
  }
  else if (outlier_pos$type == 'AO') {
    ts_plot <- plot_time_series(
      .data = datos,
      .value = datos$V1,
      .date_var = 1:dim(datos)[1],
      .interactive = FALSE)
    print(ts_plot +
            geom_vline(xintercept = pos_test[i], color = "red") +
            geom_vline(xintercept = outlier_pos$ind, color = "orange") +
            labs(title = as.character(i), subtitle = 'Additive Outlier'))
  }
  else{
    ts_plot <- plot_time_series(
      .data = datos,
      .value = datos$V1,
      .date_var = 1:dim(datos)[1],
      .interactive = FALSE)
    print(ts_plot +
            geom_vline(xintercept = pos_test[i], color = "red") +
            geom_vline(xintercept = outlier_pos$ind, color = "green") +
            labs(title = as.character(i), subtitle = 'Level Shift'))

  }
}
```

Patrones periodograma

```
for (j in c(1,15,5,6)) {
  datos <- read.csv(files[j], header=FALSE)
  for (i in 0:4) {
    mypath <- file.path("C:","Matemáticas","TFG","Time Series Anomaly Detection",
                        "images","plot periodos",
                        paste("plot", j,"_", i+1, ".jpg", sep = ""))
    periodo <- freqs[j]
    a <- periodo * i+1
    b <- periodo * (i+1)
    sub_plot <- plot_time_series(
      .data = data.frame(datos[a:b,]),
      .value = datos$V1[a:b],
      .date_var = 1:periodo,
      .interactive = FALSE)
    jpeg(filename = mypath)

    print(sub_plot + labs(title = as.character(j), subtitle = as.character(i+1)))
    dev.off()
  }
}
```