Técnicas de clasificación de series temporales basadas en medidas de distancia elásticas

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Mourad Abbou Aazaz
Tutor/es:
Manuel Campos Mártinez

Facultad
Informática
Universidad
Murcia

7 de Septiembre de 2020

Técnicas de clasificación de series temporales basadas en medidas de distancia elásticas

Comparación de benchmarks de clasificación de series temporales basadas medidas de distancia elásticas

Autor
Mourad Abbou Aazaz

Tutor/es
Manuel Campos Mártinez
Departamento de Informática y Sistemas

Facultad de Informática

Grado en Ingeniería Informática

UNIVERSIDAD DE MURCIA

Murcia, 7 de Septiembre de 2020

Declaración firmada sobre originalidad del trabajo

D./Dña. **Mourad Abbou Aazaz**, con DNI **49699519S**, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TF titulado "**Técnicas de clasificación de series temporales basadas en medidas de distancia elásticas**".

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa delos Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 7 de Septiembre de 2020

Fdo.: Mourad Abbou Aazaz
Autor del TF

Resumen

El objetivo de este Trabajo de Fin de Grado es implementar, evaluar y comparar dos nuevas técnicas de clasificación de series de tiempo de distinto tipo basadas en distancias elásticas. Tenemos la técnica *Proximity Forest*, basada en árboles de decisión, y *Nearest Neighbours*, que utiliza la medida de distancia *Lower Bounds Enhanced*. Ambas técnicas se van a comparar otro modelo de Nearest Neighbours pero que hace uso de la medida de distancia *Dynamic Time Warping*, y es actualmente la técnica que mejores resultados ofrece en términos de relación entre precisión y tiempo de predicción. El trabajo consiste en explicar que son series de tiempo, sus aplicaciones y las medidas de distancias elásticas. Luego, presentaremos las técnicas de clasificación innovadoras que vamos a evaluar. Nuestro desarrollo se ha concentrado en implementar las técnicas de Proximity Forest y LbEnhanced en Python basándonos en los proyectos desarrollados en Java por los investigadores que publicaron estas técnicas. Se evalúan empíricamente los hiperparámetros y su influencia en los proyectos desarrollados y se comparan los modelos idóneos de cada técnica con el otro modelo de Nearest Neighbours. El resultado obtenido durante la comparación fue que Proximity Forest ofrece unos resultados mucho mejores que los otros dos benchmarks. Nearest Neighbours con Dynamic Time Warping, sigue ofreciendo los mejores resultados de precisión, pero Proximity Forest, aunque ofrece unos resultados de precisión ligeramente más bajos, ofrece un tiempo de ejecución muchísimo menor y una relación precisión/tiempo bastante mejor.

Extended Abstract

Time Series are arrays of observations arranged chronologically across time line. They represent an important percentage of all the data around the world, due to its use for monitoring devices, the Internet Of Things, procesing time data and clinical analysis. Time Series processing needs special techiques due to its particular structure and classic algorithms for data science and clustering may not be useful to work on time stamps where each stamp is correlated to the previous and the next ones. One of most useful data processing uses for time series is classification, even though there are other powerful uses like Clustering. Time series classification consists on, given a time serie, what data group it belongs to or what group of time series it shares common patterns with. The main difference between standard classification and time series clasification is that time series stamps order matterns and iteractions among time positions is not independent. In time series classification, in order to properly determine what group a time series belongs to, we must compare the sequence to rest of the group sequences in order to check if they are similar or if they follow the same patterns. That's why we need a distance measure to compare two sequence and see how similar the are across time path. The most popular distance measure or similarity measure is the Euclidean distance.

$$D_{euclidean}(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})(\vec{x} - \vec{y})'}$$

This one is pretty simple and easy to implement. However, it tends to be very weak and most of the times when two similar sequences have different length and shape across time line. One of the sequences needs some "warping" in order to have a correct comparison, otherwise results will be incorrect. That's why we need Elastic Distance Measures. Elastic distance measures can warp, compress or stretch a time sequence in order to align its time stamps so that time patch in both sequences can match. The most popular elastic distance measure is the **Dynamic Time Warping** or DTW. This distance measure, based on the dynamic programming technique, consists on, given two time sequences S=$S_1, ..., S_n$ and R=$R_1, ..., R_n$ and a matrix M_{DTW}, having an $m \cdot m$ size, in which each position of the matrix $Mi, j = (S_i - R_j)^2$, the DTW technique finds the best warping path inside the matrix M_{DTW}. Each cell $M_{DTW}(i, j)$ represents the current cost from position $(0, 0)$ to (i, j). Each cell $M_{DTW}(i, j)$ calculates de distance between S_i and R_j and the result is added to the minimum value of its neighbouring cells.

$$\mathcal{D}_{DTW}(i, j) = (R_i - S_j)^2 + min \left\{ \begin{array}{c} \mathcal{D}_{DTW}(i-1, j-1) \\ \mathcal{D}_{DTW}(i, j-1) \\ \mathcal{D}_{DTW}(i-1, j) \end{array} \right\}$$

This process is repeated untill the cell (m, m) is reached. Then, we obtain a *warping path* $\mathcal{P}$, which is form by pairs of points.

$$\mathcal{P} =< (r_1, s_1), (r_2, s_2), ..., (r_n, s_n) >$$

Each pair p_i of the warping path $\mathcal{P}$ represents de distance between the time stamp value of the position r_i of the sequence R and the time stamp value of the position s_i of the sequence S.

DTW, as it's based on dynamic programming, can find the optimal path to compare two non-synchronized time sequences. However, it takes a huge computation cost, $O(m \cdot m)$, being m the sequence length. That's why it's very practical to use a warping window w. This window helps limiting the number of cells to explore and it reduces the computation cost to a $O(m \cdot w)$ complexity. However, there might be cases in which DTW can't find an optimal path in some circumstances. A simple DTW measure can be very effective when it comes to aligning two time sequences across the time axis, but it doesn't work if we want to align a sequence in the Y axis. That's why the Derivate Dynamic Time Warping was introduced. This distance measure, also based on DTW, collects information of a sequence shape by applying a derivate, that is, given a sequence S, there's another sequence S' in which each point of S_i equals the average of the slopes between S_{i-1} and S_{i+1}. This distance measure can also use a warping window and its computational cost is the same as DTW complexity. Another distance measure based on DTW is the Weighted Dynamic Time Warping. There are some cases in which two sequences, S and R, there is a point S_i which is too far way from its matched point Ri. WDTW tries to calculate to cost of aligning these two points using a cost function. The main purpose is to penalize large distances between to points due the distorsion of the warping path they could cause.

Time Series Classification techniques to compare

In these thesis we compare two time series classification techniques, which are Proximity Forest and Nearest Neighbours using LbEnhanced and DTW measure.

Proximity Forest This is a new classification technique for time series which consist on a set of proximity trees. A Proximity Tree is a decision tree and each node of the tree contains a time series. So, a time series S matches to the node which contains the most similar series, using a random distance measure, even though in this case we will focus only on the DTW measure. A Proximity Forest model is compouned by k trees and c candidates per split. Each node is built recursively from the root to the leaves. If a node is considered *pure*, that is, all the information collected in that node belongs to the same class, the node becomes a leaf-node. We start having a only a tree having formed by a root node and then we split our nodes to c candidates. Once the splits are expanded, we the select the ones having the greatest difference between their parent-nodes' gini impurity and the sum of children nodes' gini impurities. To classify a series Q, we start at the root of the main tree. Each subnode applies a distances measure to the Q sequence and the most similar subnode collects that serie. Each node contains a class and a set of series which belong to that class. The Proximity Forest complexity is $O(k \cdot \log n \cdot c \cdot l^2)$, being k the number of trees, $\log n$ the depth of each tree, c the distance between two sequences y l^2 the square series length.

Nearest Neighbour and Lower Bounds Enhanced This is a common algorithm, also known as KNN, used for time series classification. It consists on classifying a time series applying a distance measure, normally DTW, to the k time series previously classified. In this case we will focus on a new distance measure that will be used in the Nearest Neighbours technique to classify time series. This distance measure, Lower Bounds Enhanced or LbEnhaced, establishes some lower bounds to the DTW warping path. These restriction requires that, having two sequences S and T, and matrix M, where the number of rows represents S , the number of columns represents T and both have an L length, the value of S_1 is (1,1), the value of S_2 is one of the cells $\{(1,2),(2,1),(2,2)\}$ and so on.

This is known as left bands and picks the maximum value of each candidate position starting from (1,1).

$$L_i^W = (max(1, i-W), (max(1, i-W)+1, i)...(i,i), (i, i-1)...(i, max(1, i-W))$$

The sum of all those minimum values is what established the lower bounds to the DTW measure. On the other side of the matrix, starting from the position (L, L), we apply the same restriction but using the minimum value. This set of values is known as *right bands*.

$$R_i^W = (min(L, i+W), (min(L, i+W)+1, i)...(i,i), (i, i-1)...(i, min(L, i+W))$$

The main hyperparameters of this distance measure are the warping window, which works as threshold inside de matrix calculation path, and the speed-tightness which limits the number cells or bands we can reach on the right and on the left from a certain position inside the matrix. A small speed-tightness value requires less computations but the result wouldn't be as good. A high speed-tightness value offers a good accuracy value, but requires more computations that can affect the prediction time. In the end, if we have two time series S and T, we sum all the minimum values of L and R of the sequence S, and if the result of the sum is higher than current distance to the nearest neighbour, we discard this result and use another lower-bounded distance measure called LB_{KEOGH}. If it's not, we calculate calculate un upper and lower envelope of T. Then, we iterate the bands we established with speed-tightness value. For every band i, if $S_i > U_i$, we add the distance from U_i^T to S_i to the result of the sum calculated previously. Otherwise, if $S_i < L_i^T$, we add the distance from L_i^T to S_i to the result.

Objectives

The aim of this thesis is two compare both techniques Proximity Forest and KNN using LbEnhanced to KNN with DTW because this one is to offer the best trade-off performance, in terms of accuracy and execution time. Proximity Forest and KNN LbEnhanced will be evaluated in terms on hyperparameter and their influences on accuracy and execution time. After analysing their best hyperparameter values, we'll choose the best models of both techniques and then compare them to KNN DTW. Each model will be submited to a cross validation to be sure there's no discrepancy in the results and the algorithm implemented is correct. Both techniques will be developed in Python.

Experiments

Proximity Forest is originally developed in Java by the authors of this technique. What we did is to implement it in Python based on the project in Java. After analyzing the influence of the values of the hyperparameters on accuracy and execution time, we decided to chose a Proximity Forest model having 20 trees and 2 candidates per split because our experiments showed that the impact on accuracy wasn't very high with respect to other models having higher number of trees and candidates, but the execution time was definitely much lower. Then, we compared both results in python and Java using the same model we selected, and even though both languages might draw different results due to the structure of each language, the result were pretty similar. For KNN LbEnhanced we analyzed the hyperparameters, such as the speed-tighness V, the number of neighbours and the warping window. The results we extracted from experiments is that, even though we authors of this new distance measure are still researching for the best value for the speed-tightness, we decided to select value of V = 20 because it offers a good trade-off. Then, experiments results showed us that

having a warping window = 1 and having only 1 neighbour will be enough to give us good results in terms of accuracy and time execution. Both selected models were submited to a cross validation to verify the implementation is correct and solid.

After selecting the best models, acording to the results extracted from the experiments, we decided to compare them to a KNN DTW model. The results we found in the comparison were that KNN DTW still offered good accuracy results, but Proximity Forest presents a execution time way lower than the rest of the benchmarks having a lower accuracy, but not too much. Proximity Forest definitely offers the best trade-off because the number of trees and the number of candidates per split are very low. If we chose another Proximity Forest model using more trees and candidates per split, time would be even greater than KNN DTW. On the other hand, KNN LbEnhanced also has better results than KNN DTW in terms of time and the accuracy results are might be lower but not too much. But execution time it's still much higher than the Proximity Forest model.

Future work

The conclusion we've drawn during the analysis is that Proximity Forest is a good choice when the main goal is to have fast predictor. Even though accuracy results might not be very high as high as KNN DTW they present way better trade-off and reaching the highest accuracy might not be affordable if it takes a lot of time.

For this work we compared three benchmarks based on elastic distances, two of them are based on DTW and the other one based on LbEnhanced. The results showed KNN using DTW still presented good accuracy values, but Proximity Forest proves to have a better perfomance trade-off. What would be interesting for the future is comparing these techniques using another elastic distances such as DDTW or WDTW, which try to solve some problems DTW can't solve. There are some other new classification techniques like FastEE, which consists on a set of classifiers, but the authors, who are also the ones who published the Proximity Forest Classifier, stated it was not as efficent as Proximity Forest is.

Índice general

1 Introducción **1**
- 1.1 Series de tiempo y sus aplicaciones . . . 1
- 1.2 Medidas de distancia elásticas . . . 3
  - 1.2.1 Medida de Distancia Euclídea . . . 3
  - 1.2.2 Medidas de distancia elásticas: Dynamic Time Warping . . . 5
- 1.3 Técnicas de clasificación de series de tiempo . . . 8
  - 1.3.1 Clasificación basada en distancias . . . 8
  - 1.3.2 Clasificación basada en vecinos próximos . . . 8
  - 1.3.3 Shapelets independientes de la fase . . . 9
  - 1.3.4 Fast Shapelets . . . 10
  - 1.3.5 Bag of Words . . . 10
  - 1.3.6 Clasificación basada en Ensembles . . . 11
  - 1.3.7 Clasificación basada en árboles de decisión . . . 11
- 1.4 Técnicas de estado de arte . . . 12
  - 1.4.1 Proximity Forest . . . 12
  - 1.4.2 Lower Bounds Enhanced . . . 13

2 Objetivos y metodologías **17**
- 2.1 Objetivos . . . 17
  - 2.1.1 Estudio de técnicas de clasificación a comparar . . . 17
  - 2.1.2 Estudio de técnicas de clasificación a comparar . . . 17
  - 2.1.3 Comparación de las técnicas anteriores con KNN con DTW . . . 18
- 2.2 Metodología y herramientas de trabajo . . . 18

3 Experimentación, datos utilizados y resultados **19**
- 3.1 Datos utilizados para los experimentos . . . 19
- 3.2 Proximity Forest: Análisis y validación . . . 21
  - 3.2.1 Sktime y motivo de implementación propia . . . 21
  - 3.2.2 Mejores hiperparámetros, tiempo y limitaciones . . . 22
  - 3.2.3 Evaluación con K-Fold Crossvalidation . . . 26
  - 3.2.4 Comparación con Proximity Forest original . . . 26
  - 3.2.5 Comparación con t-test . . . 28
- 3.3 KNN con Lower Bounds Enhanced y DTW: Análisis y validación . . . 30
  - 3.3.1 Hiperparámetros y análisis de rendimiento . . . 30
  - 3.3.2 Evaluación con K-Fold Crossvalidation . . . 36
- 3.4 Comparación con Nearest Neighbours - DTW . . . 37

4 Conclusiones **42**

Bibliografía **45**

5 Anexo **49**

1 Introducción

Este capítulo de introducción tiene como objetivo explicar los conceptos básicos de la definición de una serie de tiempo, así como comentar sus diversas aplicaciones en distintos campos, explicando con más detalle los puntos sobre los que se centra este trabajo.

1.1 Series de tiempo y sus aplicaciones

Definimos una serie temporal T como un conjunto de observaciones ordenadas cronológicamente en el tiempo, con una longitud m, asociado a una etiqueta o clase y. En muchas ocasiones aparece como un par *(T, y)* y las observaciones pueden contener una o varias variables que son históricas y, por tanto, sus valores son irrepetibles [1]. Las series temporales componen una parte muy importante del total de todos los conjuntos de datos que existen. Su importancia en el mundo de las ciencias de datos ha ido creciendo debido al crecimiento de la producción masiva de conjuntos secuenciales de datos como el Internet de las Cosas (IoT), la digitalización de los procesos sanitarios, el aumento de las *Smart Cities* y en general cualquier tipo de tecnología que necesite monitorear grandes cantidades de datos que se generan en cada instante de tiempo. [2]. Tan solo en el campo de la medicina existen enormes volúmenes de datos de amplia diversidad como puede ser el campo de la ingeniería genética [3] y aplicaciones en otros campos de la industria, las finanzas, la meteorología [4]. Es bastante común que un conjunto de datos experimentales tenga una cierta correlación con una línea temporal y que, por tanto, sea complicado analizar el conjunto de datos utilizando técnicas algorítmicas tradicionales de *Machine Learning* y *Data Mining* debido a su peculiar estructura. Los algoritmos clásicos asumen que los datos tienen una baja dimensionalidad, cosa que no ocurre con las series temporales, que se caracterizan por tener una alta dimensionalidad y un alto contenido de ruido, lo que provoca que sea difícil llevar a cabo tareas de minería de datos [4]. Es por eso que los algoritmos de minería de datos para series temporales evitan operar con datos originales, y consideran darles una mejor representación [4]. Antes de continuar con los objetivos de este trabajo de fin de grado, vamos comentar de manera resumida la utilidad de las series de tiempo ciertos campos de las ciencias e ingeniería de datos.

Clasificación La clasificación, como su nombre indica, consiste en categorizar o clasificar una serie datos en un grupo predefinido que comparte características y patrones comunes. Para ello se utiliza dicho grupo para entrenar el reconocimiento de dichos patrones que nos interesan. Cabe destacar que para los problemas de clasificación tradicionales, el orden de los atributos de un conjunto no importa y la interacción entre variables suele ser independiente de las posiciones en las que se encuentren [24]. Sin embargo, para una serie temporal es imprescindible el orden a la hora de clasificar y reconocer los patrones de interés. Las técnicas de búsqueda de patrones más populares en la clasificación de series temporales son los Árboles de Decisión (*Decision Tree*) y la técnica de vecinos próximos (*Nearest Neighbors*). Luego trataremos más en detalle este tipo de enfoques de clasificación.

- **Decision Trees**: En los decision trees, o árboles de decisión, se infiere un conjunto de reglas del conjunto de datos de entrenamiento, y dichas reglas se aplican a cualquier nuevo conjunto de datos para ser clasificado. Los árboles de decisión están formado por tres elementos básicos:

- **Nodo raíz**
- **Nodo Hoja o terminal**: Contiene la etiqueta o el identificador de una clase.
- **Nodo no terminal**: Cada nodo no terminal representa una condición y a partir de un hecho que ofrece el dato se pueden ramificar distintos sub-nodos que representas distintas hipótesis a partir de ese hecho.

Para explicar como funcionan los árboles de decisión, nos apoyaremos en este ejemplo básico 1.1.

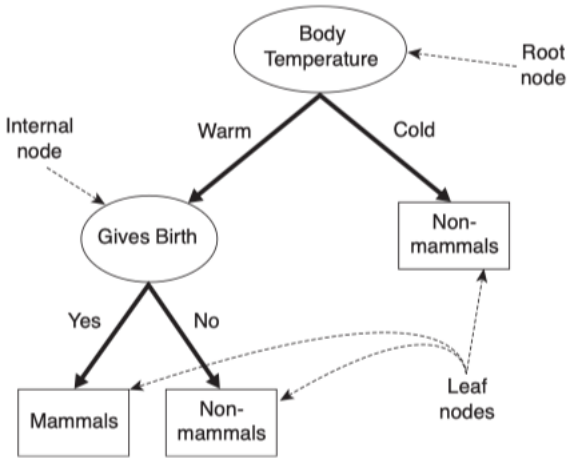


Figura 1.1: Árbol de decisiones para determinar si un animal es mamífero o no. Imagen extraída del siguiente enlace: `https://www.researchgate.net/figure/A-decision-tree-for-the-mammal-classification-problem_fig16_282866516`

La Figura 1.1 representa un conjunto de reglas para saber si un animal es mamífero o no. El nodo raíz es el `Body Temperature` que representa la temperatura corporal del animal en cuestión y que separa dos posibles casos: Fría o Caliente. Dicho nodo raíz se ramifica en dos sub-nodos en función de dos posibles hechos. A la derecha tenemos un nodo hoja que infiere que un animal no es un mamífero si su temperatura corporal es fría. A la izquierda tenemos un nodo no terminal al que se accede cuando la temperatura corporal de un animal es no es fría. Este subnodo representa una condición que separa dos posibles casos: Si el animal da a luz o no. Para ambos casos, se accede a un nodo hoja que determina si un animal es mamífero, en función de si da a luz o de que no. Por tanto, podemos concluir que un animal se considera mamífero solo si su temperatura corporal es de sangre caliente y da a luz. Como podemos ver, la construcción de un clasificador se hace de forma casi inmediata una vez se haya construido el árbol de decisión. Comenzando desde el nodo, aplicamos una serie de hechos y cada hecho nos conducirá al sub-nodo correspondiente en función de las condiciones hasta llegar al nodo hoja que contiene el tipo de clasificación para nuestro conjunto. Más adelante explicaremos con más detalle como funciona y como se construye el árbol de decisiones, que vamos a utilizar ya que este trabajo de fin de grado tratará de analizar dos técnicas de clasificación distintas.

Agrupación La agrupación o *clustering* es similar a la clasificación, ya que intenta agrupar la información por categorías o grupos. Sin embargo, los grupos no están predefinidos sino que se tienen buscar patrones comunes basados en la similitud de las distancias temporales. Los datos que

presentan mayor similitud se agrupan en clústers. Los métodos generales para las series de tiempo son:

- **Whole Clustering**: Dado un conjunto de series individuales de tiempo, el objetivo es agrupar las series de tiempo que son similares en el mismo clúster [5].
- **Clustering de subsecuencia**: Se aplica a cada serie individual de tiempo extraída de una serie larga [5].

Detección Como se comentó en los apartados anteriores, las series de tiempo pueden ser útiles para aplicaciones de clasificación y/o agrupamiento por detección de patrones de interés. Sin embargo, también se pueden detectar secuencias cuyos patrones de comportamiento son desconocidos o cuyas características no encajan con las del resto de secuencias del mismo tipo. Por tanto, las series temporales pueden utilizarse para detectar anomalías a la hora de realizar labores de monitorización, control en tiempo real, etc. El problema de la detección de anomalías es que su uso se ha generalizado para intentar detectar patrones de interés, en vez de detectar patrones anómalos [6].

Predicción Consiste en poder predecir un valor o estado en el futuro en vez de hacerlo en el presente, como ocurre con la clasificación o la agrupación. Los modelos de predicción, utilizando series temporales, resultan de mucha utilidad para poder estimar un resultado futuro a partir de datos históricos y su uso se puede encontrar en ejemplos como predecir la tasa de natalidad de un hospital en una ciudad, o el precio de la gasolina que se establece cada día en una región [7].

1.2 Medidas de distancia elásticas

La mayoría de las técnicas utilizadas para la clasificación de series temporales se basa en las medidas de semejanza. Se dice que dos series son semejantes si sus puntos forman un recorrido similar en el eje del tiempo. Por ejemplo, en la Figura 1.2 podemos observar un gráfico de la disipación de energía de los ciclones tropicales ocurridos en distintos puntos en el Océano Atlántico y Pacífico desde 1960 hasta 2006. Como se puede observar, las series tienen un recorrido similar según transcurre el tiempo dibujando lineas curvas con la misma topología. Las variaciones deben al ruido causado en la observación y también a los posibles desajustes que se puedan provocar en algunas fases en el eje temporal.

1.2.1 Medida de Distancia Euclídea

Una de las medidas más simples para medir la semejanza entre series temporales es la Distancia Euclídea [8]. Dicha medida de distancia es bastante simple de entender y fácil de implementar, y por eso es una medida ámpliamente utilizada para búsquedas de similitud [9].

$$D_{euclidean}(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})(\vec{x} - \vec{y})'}$$

Sin embargo, la distancia euclídea tiene una gran desventaja y es que tiende a ser muy frágil y no permite que dos series similares estén desfasadas y/o desajustadas en el tiempo. Tampoco permite que alguna de las dos series sufra variaciones en su recorrido [9]. Estos problemas que presenta la distancia euclídea pueden resolverse trasladando una de las series a la posición de partida de la otra y aplicando medidas de escalado y normalizado (Figura 1.4).

Sin embargo, incluso después de aplicar el escalado y normalizado de las series, la distancia euclídea sigue siendo insuficiente ya que no permite que una serie tenga variaciones, aceleraciones o

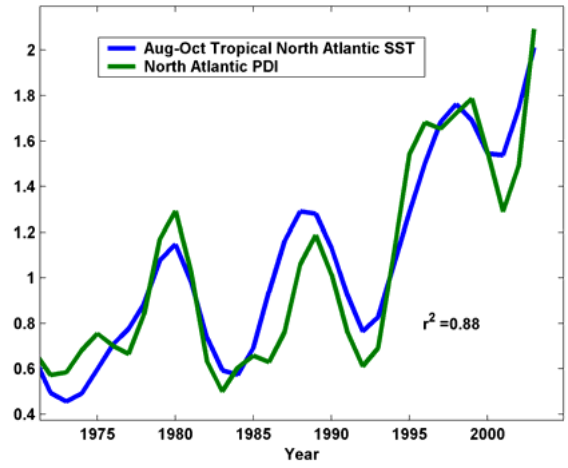


Figura 1.2: Recorrido de la energía disipada por ciclones en el Océano Atlántico y Pacífico desde 1960 hasta 2006. Fuente: `https://www.researchgate.net/figure/Power-dissipation-index-from-the-ERA-40-re-analysis-data-set-for-global-tropical-cyclones_fig2_265290245`

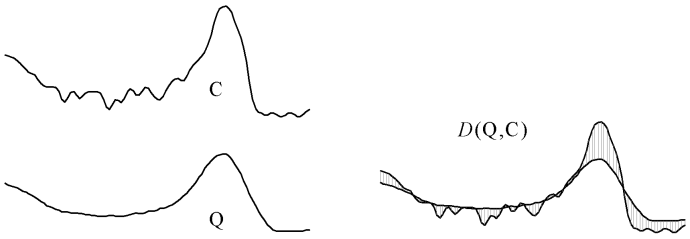


Figura 1.3: En esta imagen se observa como se comparan dos secuencias de tiempo utilizando una medida de distancia euclídea [8]

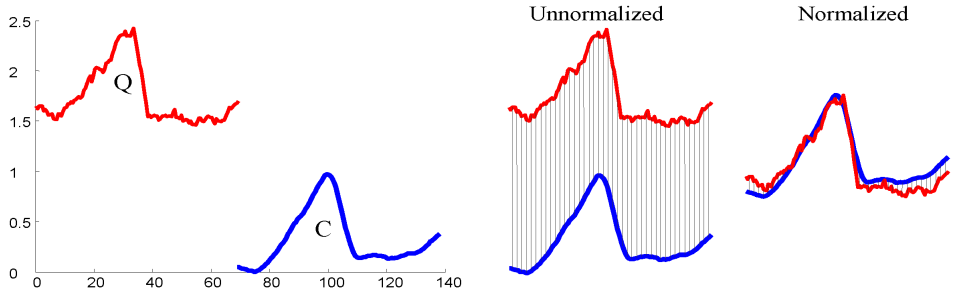


Figura 1.4: La imagen, extraída del artículo *Efficient similarity search in sequence databases* [9], muestra dos secuencias desfasadas con el mismo tipo de recorrido pero que necesitan un escalado y normalizado para poder ser comparadas

desaceleraciones en el eje del tiempo y, por tanto, no calculará de manera correcta la distancia entre series. Por este motivo, para solventar este problema se utilizan medidas de distancia elásticas.

1.2.2 Medidas de distancia elásticas: Dynamic Time Warping

Como comentamos en la sección anterior, es muy común que dos secuencias tengan la forma y recorrido parecidos pero que no estén alineados en el eje X. Es por eso por lo que hace falta "estirar" o "comprimir" una de las secuencias o ambas en el eje temporal. Eso es lo que hacen las medidas de distancia elásticas. Una de las medidas de distancia elásticas más conocidas es la medida *Dynamic Time Warping* o *DTW* [10]. Esta medida de distancia basada en programación dinámica trata de deformar las secuencias en el eje temporal para que se asemejen. Dicha medida se corresonde con la suma de cada par de puntos p_k conectados en el camino conectados en un camino P [1].

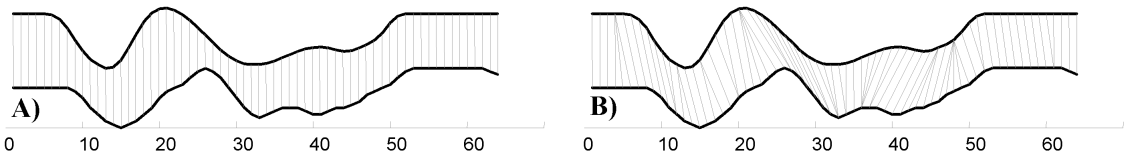


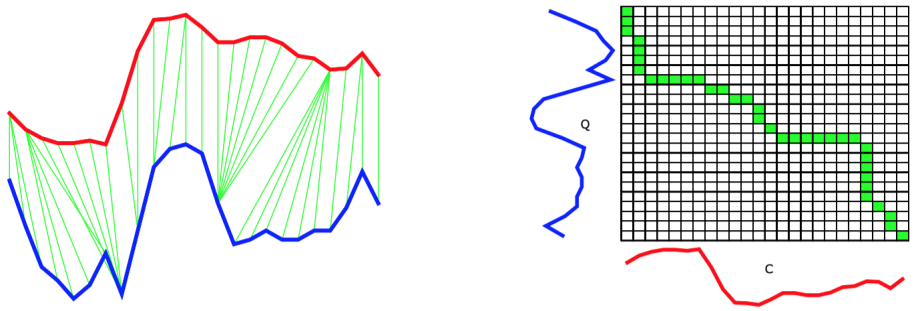
Figura 1.5: Aqui podemos ver dos secuencias temporales parecidas, pero que una sufre una variación con respecto a la otra en un intervalo. A la izquierda podemos observar cómo se calcula la distancia euclídea en el que i-ésimo punto de una secuencia se alinea con el i-ésimo punto de la otra. A la derecha podemos ver como se calcula una distancia elastica DTW. La imagen en cuestión fue extraída del artículo *Dynamic programming algorithm optimization for spoken word recognition* [10]

Su funcionamiento es el siguiente: Considerando dos secuencias $R = \{R_1, ..., R_n\}$ y $S = \{S_1, ..., S_n\}$ y una matriz M_{DTW}, de tamaño $m \cdot m$, que representa las distancias entre R y S donde cada posición $M_{i,j} = (R_i - S_j)^2$, DTW intenta encontrar el camino óptimo dentro de la matriz M_{DTW}. Cada celda $M_{DTW}(i, j)$ de la matriz representa el coste acumulado actual desde el la posición $(0, 0)$, como se describe en la ecuación que aparece a continuación. En dicho punto, $M_{DTW}(i, j)$, se calcula la distancia de entre R_i y $_j$ más el mínimo valor que existe entre sus vecinos. Este proceso se repite hasta llegar hasta el punto (m, m) [11]. Tras recorrer la matriz, obtenemos el camino de distorsión o *Warping path* formado por pares de puntos.

$$\mathcal{P} =< (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) >$$

Cada elemento p_i del Warping Path $\mathcal{P}$, siendo i el i-ésimo par de puntos, corresponde a la distancia que existe entre la posición a_i de la secuencia R y b_i de la secuencia S.

$$\mathcal{D}_{DTW}(i, j) = (R_i - S_j)^2 + min \left\{ \begin{array}{c} \mathcal{D}_{DTW}(i-1, j-1) \\ \mathcal{D}_{DTW}(i, j-1) \\ \mathcal{D}_{DTW}(i-1, j) \end{array} \right\}$$



(a) (b)

Figura 1.6: (a) Alineamiento entre dos series mediante DTW. (b) La matriz de coste M_{DTW} en la que aparecen los puntos del camino de distorsión W en color verde. Cabe destacar que si aplicáramos una medida de distancias euclídea, el camino de distorsión o *Warping path* formaría una diagonal dentro de la matriz. Esta imagen fue extraída del siguiente enlace: `https://www.researchgate.net/publication/337355704_FastEE_Fast_Ensembles_of_Elastic_Distances_for_time_series_classification`

La técnica DTW consigue encontrar el camino óptimo para conseguir asemejar varias secuencias no sincronizadas, pero su principal desventaja es su alto coste computacional de orden $\mathcal{O}(m \cdot m)$, que dificulta su escalabilidad. Para solventar este problema se establece una restricción, limitando el valor máximo permitido de distancia entre pares de puntos. Dicha restricción se conoce como *Warping Window* o ventana de distorsión. De esta forma, dada una ventana w, se consigue un tiempo de ejecución de orden $\mathcal{O}(w \cdot m)$ [8].

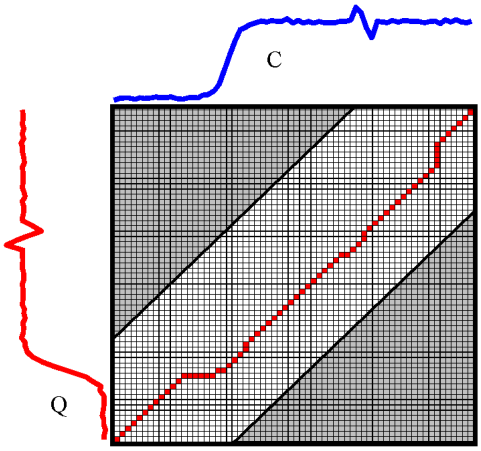


Figura 1.7: Dadas dos secuencias C y Q, para alinear ambas secuencias se construye una matriz de distorsión para poder encontrar el camino óptimo. Como se puede observar, los puntos de las zonas que están en color gris oscuro son descartadoss ya que se ha aplicado una ventana w se estable como límite en el recorrido de la matriz. Fuente: `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=7847D05EC0213AC8A9E67F990796B451?doi=10.1.1.23.6686&rep=rep1&type=pdf`

Derivative Dynamic Time Warping

La técnica *DTW* es bastante buena para alinear dos secuencias que son similares pero que están desajustadas en el eje temporal X. Sin embargo, es posible que también existan desajustes en ciertas regiones o intervalos en el eje Y. Por ejemplo, dadas dos secuencias R y S, existe un punto R_i que

está en un intervalo ascendente pero que, al mismo tiempo, está emparejado con un punto S_i que está dentro de un intervalo descendente (Figura 1.8). El problema que tiene *DTW* es que no resuelve los desajustes o anomalías que existen en el eje Y, ya que solo recopila los valores de los puntos en dicho eje.

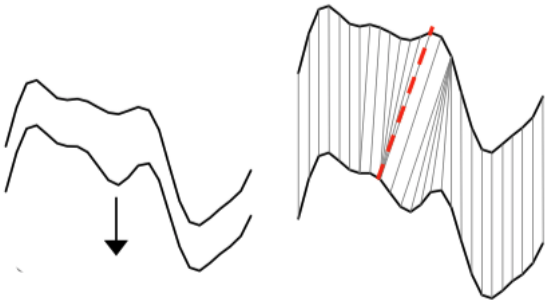


Figura 1.8: Se puede observar como dos secuencias tienen puntos emparejados (linea roja) pero dichos puntos pertenecen a intervalos que tienen un recorrido distinto en el eje Y. Uno es ascendente y el otro es descendente. Fuente: `https://www.researchgate.net/publication/273447766_Mineria_de_datos_para_series_temporales`

Es por eso por lo que los investigadores Eammon Keogh y Michael J. Pazzan introducen una nueva variante llamada *Derivative Dynamic Time Warping* o DDTW [12]. Esta técnica se enfoca en la forma característica de la secuencia, en lugar de limitarse solo a tomar los valores de los puntos del eje Y como hace normalmente. Los autores proponen obtener información de la forma de las secuencias aplicándoles una derivada. Por tanto, la derivada de una secuencia $Q = \{q_1, ...q_n\}$ es otra secuencia $Q' = \{q'_2, ...q'_{n-1}\}$ en la que cada punto q_i se define como la media de las pendientes entre los puntos q_{i-1}, q_i y q_{i+1} para cada $1 < i < n$.

$$q'_i = \frac{(q_i - q_{i-1}) + \frac{(q_{i+1} - q_{i-1})}{2}}{2}$$

Empíricamente, según los autores, la técnica *DDTW* tiene un tiempo de ejecución de orden $O(m\dot{n})$ que es el mismo que *DTW* y también se le puede aplicar restricciones como una ventana de distorsión, que ya mencionamos en el apartado anterior [12].

Weighted Dynamic Time Warping

Otra variante de *DTW*, introducida por los investigadores Jeong y Omitaomu, es el algoritmo *Weighted Dynamic Time Warping* o WDTW [13]. Existen casos en los que, dadas dos secuencias Q y R con tamaño n, puede que un punto Qi se encuentre muy lejos de su emparejado Ri y, por tanto, tiendA a formar una distorsión muy grande. Para prevenir este problema la técnica *WDTW* mide el coste de alinear un punto Qi con su punto homólogo Ri usando una función de coste. El objetivo es añadir una penalización a aquellas distancias largas entre puntos. Por eso, cuando construimos una matriz $\mathcal{M}$, a cada celda $\mathcal{M}_{i,j}$ de la matriz se le aplica un coste junto a la distancia calculada.

$$\mathcal{M}_{i,j} = w_{|i-j|} \cdot d_w(Q_i, R_j)$$

$w_{|i-j|}$ es un valor del coste que existe entre Qi y Rj. Este valor estará determinado por la distancia de distorsión entre estos dos puntos: Si la distancia entre los dos puntos es pequeña, tendrá un valor de penalización menor.

$$w_i = \frac{W_{max}}{1 + e^{-g \cdot (i - n/2)}}$$

W_{max} es el límite máximo para el parámetro de coste, teniendo normalmente el valor 1 [13]. El parámetro g controla el nivel de penalización para las distorsiones largas. Cuanto mayor sea el valor de g, mayor es la penalización entre para las distorsiones largas formadas por dos puntos [13] y su valor óptimo de suele variar entre 0.01 y 0.6 según los estudios empíricos [13].

1.3 Técnicas de clasificación de series de tiempo

En esta sección repasaremos los distintos tipos de técnicas de tiempo existen actualmente. Para cada tipo haremos un breve resumen y luego explicaremos con más detalle las dos técnicas que queremos analizar en este trabajo de fin de grado.

1.3.1 Clasificación basada en distancias

Una de las propiedades de las series temporales es que una marca t_i va a tener un valor muy parecido a su predecesora t_{i-1}. Esto ha llevado a que se desarrollen múltiples medidas de distancias que ayuden a comparar, de manera efectiva, series de un mismo tipo. Muchas series suelen tener un recorrido similar, pero frecuentemente padecen distorsiones en el eje temporal debido a fenómenos que han provocado retardos o se han producido en momentos diferentes a los esperados. Por esta razón es importante contar con medidas de distancia que ayuden a comparar, de manera efectiva, series de un mismo tipo. Esta categoría de algoritmos se caracteriza por utilizar una medida de distancia entre series que cuantifica la distancia entre dos secuencias tras ajustarlas para eliminar posibles desfases o distorsiones [14]. A parte de la medida de distancia DTW, y sus derivados, que hemos visto en la sección Medidas de distancia elásticas, también podemos encontrar otras medidas de distancia como *Longest Common Subsequence* (LCSS) [40] o *Time Warp Edit Distance* (TWE) [39], entre otras. El artículo publicado por investigadores de la Universidad de East Aglia, *The great time series classification bake off* [15] ofrece una descripción detallada de las medidas de distancia, así como una comparativa de cada una de ellas. El enfoque basado en series puede ser confundido por el hecho de que existan algunas dilataciones o distorsiones en algunos intervalos con altibajos. Es por eso que se utilizan medidas de distancias elásticas que sean capaces de ajustar las secuencias y poder calcular las distancias de forma efectiva. Dichas medidas de distancia se emplean con un clasificador *Nearest Neighbours* y son utilizadas para encontrar desajustes que existen entre dos series, como comentábamos en apartados anteriores, como por ejemplo, la técnica *DTW* que suele utilizarse en este tipo de clasificadores [14].

1.3.2 Clasificación basada en vecinos próximos

La técnica basada en vecinos próximos o Nearest Neighbours es la más común para clasificar series temporales. La técnica Nearest Neighbours busca clasificar un conjunto de datos aplicando medidas de similitud a partir de K conjuntos de datos existentes que ya fueron clasificados. Su funcionamiento es sencillo, aunque su complejidad computacional aumenta conforme aumenta el tamaño de conjunto de datos. Lo que se hace es calcular la distancia entre un elemento e a clasificar y el resto de elementos del conjunto de datos de entrenamiento D, y selecciona aquellos K elementos que menor distancia tienen con el elemento e. Esos K elementos decidirán que cuál será la clasificación final para el elemento e. Por tanto, elegir un valor K es de vital importancia pues determinará a que grupo pertenecerá un conjunto de puntos [16].

Algorithm 1 K-Nearest Neighbours

1: **function** Classify(X, Y, x)▷ where X is the training data with size m, Y is the class labels of X, x is an unknown new sample
2: **for** $i = 1$ to m **do**
3: *Compute distance* $d(X_i, x)$
4: **end for**
5: Compute set I containing indices for the k smallest distances d(X_i, x)
6: **return** majority label for Y_i, where $i \ \ I$
7: **end function**

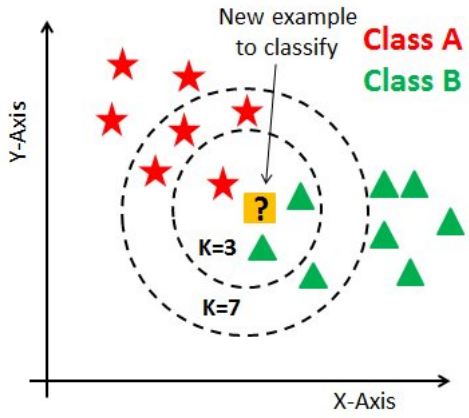


Figura 1.9: Modelo K-NN. Imagen extraída de DataCamp (`https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn`)

El problema que tiene esta técnica es que es muy simple y consume muchos recursos para que un conjunto de datos se mantenga cargado y disponible para evaluar nuevos elementos. Es por eso que no suele ser recomendado para conjuntos muy grandes [16]. Sin embargo, esta técnica combinada con **DTW** ofrece muy buenos resultados y en muchas ocasiones es difícil encontrar técnicas que mejores que esta [29].

1.3.3 Shapelets independientes de la fase

Los *Shaplets* son subseries que capturan un tipo característico dentro de la serie [17]. Ayudan a mejorar la detección de similitudes entre series de una misma clase localizadas independientemente de su fase. La necesidad de nuevos algoritmos aparece porque los algoritmos que tratan series de cierta longitud resultan tener un rendimiento muy pobre y los algoritmos basados en intervalos, aunque son buenos eliminando el ruido causado por la longitud de las series, dependen de las características discriminatorias que ocurren en determinadas zonas un intervalo.
Los investigadores Ye y Keogh [18] introducen el algoritmo original basado en shapelets, que hace una búsqueda de shapelets, haciendo un enumerado de todos los shapelets disponibles, y selecciona el mejor para expandirlo a cada nodo del árbol [18]. Dos problemas que presenta este algoritmo es que la enumeración de los shapelets es muy lenta ($O(m^2 \cdot n^4)$) y el uso de shapelets embebidos en un árbol de decisión no implican necesariamente una mejora de la precisión a la hora de construir un clasificador.

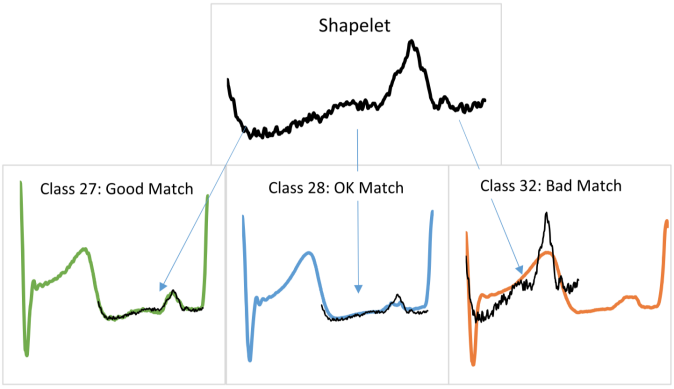


Figura 1.10: Un ejemplo de emparejamiento o *matching* entre un shapelet y varias series. Como comentamos, un shapelet es una subserie que contiene unas características. En cada serie se va haciendo buscando una subserie que se asemeje al shapelet de la parte de arriba. Esta imagen fue extraída del artículo *The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances*

1.3.4 Fast Shapelets

Los autores Rakthanmanon y Keogh proponen una extensión del árbol de decisiones basado en shapelets que acelera la búsqueda de shapelets [19]. Este algoritmo, en vez de hacer una búsqueda enumerada para cada nodo, utiliza una aproximación simbólica agregada (SAX). SAX es un método que consiste en dar una representación a las series temporales que permita reducir sus dimensiones convirtiéndola en cadenas de caracteres y luego indexarlas aplicando una medida de distancia [20]. Tal y como se muestra en la figura 1.11, el algoritmo Fast Shapelets forma un diccionario de palabras SAX para cada posible longitud de shapelet. La dimensión del diccionario se reduce aplicando una máscara a las letras de la cadena de caracteres de manera aleatoria. Se generan múltiples proyecciones aleatorias y se crea un histograma para cada clase. Luego se calcula una puntuación para cada palabra SAX basada en cómo se discriminan las tablas de frecuencias entre clases. Las k mejores palabras SAX son seleccionadas y luego mapeadas con los shapelets originales [19] [20].

Algorithm 9 buildClassifierFS(A list of n cases of length m, $\mathbf{T} = \{\mathbf{X}, \mathbf{y}\}$)

Parameters: The SAX word length l; alphabet size α and window length w; number of random projections r; number of SAX words to convert back, k

```
Let b be an empty shapelet with zero quality
for l ← 5 to m do
    SAXList ← createSaxList(T, l, α, w)
    SAXMap ← randomProjection(SAXList, r)
    ScoreList ← scoreAllSAX(SAXList, SAXMap)
    s ← findBestSAX(ScoreList, SAXList, k)
    if quality(b) < quality(s) then
        b ← s
{T1, T2} ← splitData(T, b)
if ¬ isLeaf(T1) then
    buildClassifierFS(T1)
if ¬ isLeaf(T2) then
    buildClassifierFS(T2)
```

Figura 1.11: Algoritmo Fast Shapelets

1.3.5 Bag of Words

Esta tipo de técnicas es bastante similar a los Shapelets con la diferencia de que busca las diferencias entre clases utilizando la frecuencia relativa de las subseries. Estas frecuencias se calculan

discretizando los valores en series de símbolos y cada símbolo representa una subserie [21]. Actualmente, el estado de arte de este tipo de técnicas es la técnica *Bag of SFA* (BOSS) [22]. BOSS extrae las palabras de las series temporales y les aplica un filtro para reducir el ruido y permitir que se apliquen adecuadamente algoritmos de emparejamiento de cadenas de caracteres. Entonces es cuando dos series se comparan en base a sus diferencias pero dentro de un conjunto de patrones de ruido reducido.

1.3.6 Clasificación basada en Ensembles

La clasificación basada en *ensembles* (conjuntos) es una combinación de múltiples clasificadores, en el que cada algoritmo implicado puede evaluarse para ofrecer una precisión óptima mientras tienen un bajo tiempo de ejecución. Un ejemplo es clasificador Collective of Transformation-based ensembles (COTE) [23]. Se trata de un conjunto de 35 clasificadores sobre 4 dominios de series temporales: Tiempo, frecuencia, cambio y transformación de *shapelet*. Este conjunto es el que mayor precisión ofrece dentro de la clasificación basada en conjuntos, aunque también existen otros conjuntos como Elastic Ensembles (EE) [24], o Shapelet Transform (ST) [25].

1.3.7 Clasificación basada en árboles de decisión

Los árboles de decisión funcionan mediante la inferencia de un conjunto de reglas del conjunto de datos de entrenamiento, y dichas reglas se aplican a cualquier nuevo conjunto de datos para ser clasificado. Los árboles de decisión están formado por tres elementos básicos:

- **Nodo raíz**
- **Nodo Hoja o terminal**: Contiene la etiqueta o el identificador de una clase.
- **Nodo no terminal**: Cada nodo no terminal representa una condición y a partir de un hecho que ofrece el dato se pueden ramificar distintos sub-nodos que representas distintas hipótesis a partir de ese hecho.

Se han desarrollado varios algoritmos cuyos nodos a expandir dependen la simitud con los ejemplares. Existe el Time Series Forests [26]. Este algoritmo busca solventar el problema de largos intervalos empleando un enfoque basado en *Random Forests* [26]. Entrenando solo un árbol implica elegir $\sqrt{m}$ intervalos aleatorios, generando la media, la desviación típica y una pendiente para cada serie creada, por lo que el árbol acaba teniendo $3 \cdot \sqrt{m}$ características. En la figura 1.12 se muestra como se construye un clasificador TSF.

Algorithm 6 buildClassifierTSF(A list of n cases of length m, $\mathbf{T} = (\mathbf{X}, \mathbf{y})$)

Parameters: the number of trees, r and the minimum subseries length, p.

Let $\mathbf{F} = (F_1 \dots F_r)$ be the trees in the forest.
for $i \leftarrow 1$ to r **do**
  Let $\mathbf{S}$ be a list of n cases $(\mathbf{s}_1, \dots, \mathbf{s}_n)$ each with $3\sqrt{m}$ attributes
  for $j \leftarrow 1$ to $\lfloor \sqrt{m} \rfloor$ **do**
    $a = rand(1, m - p)$
    $b = rand(a + p, m)$
    for $k \leftarrow 1$ to n **do**
      $s_{k,3(j-1)+1} = \text{mean}(\mathbf{x}_k, a, b)$
      $s_{k,3(j-1)+2} = \text{standardDeviation}(\mathbf{x}_k, a, b)$
      $s_{k,3(j-1)+3} = \text{slope}(\mathbf{x}_k, a, b)$
  $F_i.buildClassifier(\{\mathbf{S}, \mathbf{y}\})$

Figura 1.12: Algoritmo de TSF

El aprendizaje de TSF tiene una complejidad de orden $O(n \cdot \log n \cdot l \cdot k)$ donde k es el número de árboles, n el número de series y l la longitud de serie. También existe el Generalized Random Shapelet Forest [27] que extrae un shapelet de una serie aleatoria de tiempo y busca la distancia entre esa serie de tiempo y el resto. Después, los datos se expanden en función de un umbral de distancia. Sin embargo, el problema de esta técnica es su alto coste computacional. También existe otro que es *Similarity Forests* [41] que añade un valor de corte a la diferencia entre distancia entre dos ejemplares, y optimiza ese valor utilizando un coeficiente *Gini*.

1.4 Técnicas de estado de arte

1.4.1 Proximity Forest

Esta nueva técnica, introducida en 2019 y basada en árboles de decisión, fue introducida como algoritmo de clasificación que mejora la escalabilidad en conjuntos de datos muy grandes [28]. Existen varias técnicas de clasificación, vistas anteriormente, como ST, EE o COTE que ofrecen unos resultados muy buenos, pero que no escalan en la fase de aprendizaje. Otras técnicas, como BOSS-VS [22], ofrecen resultados muy pobres a medida que escalan. La escalabilidad ha sido, después de la precisión, el asunto más importante a tratar en el desarrollo y avance con respecto al estado de arte [29]. Hasta ahora, la técnica COTE era el estado de arte prevalente en términos de precisión, pero tiene un coste computacional muy grande debido a que la técnica EE no abstrae mucha información durante el aprendizaje y, por tanto, hay más procesos pendientes de completarse en el periodo de la validación o *testing*. Por tanto, se necesita un nuevo algoritmo clasificador que sea escalable en el tiempo y mantenga unos buenos resultados de predicción. Los investigadores plantearon un algoritmo basado en árboles de decisión para mejorar la escalabilidad ya que estos hacen uso de la estrategia de *Divide y vencerás* [28]. Para mejor la escalabilidad, se combina la estrategia de aprendizaje de un árbol de decisión, donde se eligen aquellos que más se asemejan entre sí, con la estrategia de formación de conjuntos de árboles en los que los nodos se eligen de manera aleatoria.

Proximity Forest: Método de aprendizaje

Se puede decir que un bosque de proximidad es un conjunto de k árboles de proximidad. Un árbol de proximidad o *Proximity Tree* es similar a un árbol de decisión pero con formas de validación distintas. Mientras un árbol de decisión aplica un tipo de validación o *testing* sobre el valor de un atributo, en los árboles de proximidad cada rama de un nodo interno del árbol tiene asociada una serie. Entonces, si queremos clasificar una serie Q, esta se corresponderá con aquel nodo cuya serie la más parecida en base a criterios de medidas de similitud. Cada nodo del árbol está formado por una medida de distancia y un conjunto de ramas, en el que cada rama está formada a su vez por una serie de tiempo y un subárbol.

Criterios de expansión La técnica se construye a partir un árbol. Cada nodo se construye recursivamente desde el nodo raíz hasta las hojas. Si un nodo es puro, esto es, toda información en ese nodo pertenece a la misma clase, entonces el nodo se convierte en un nodo-hoja. Para cada nodo se expande un conjunto de r candidatos, y para cada candidato se elige una medida de distancia de manera aleatoria. No obstante, nosotros para este trabajo utilizaremos siempre la medida de distancia DTW. Una vez expandidos y creados los candidatos, seleccionamos aquel que mayor diferencia

tenga entre la impureza de *Gini* [1] del nodo padre y la suma de las impurezas de *Gini* de los nodos hijos. Entonces se construye el árbol de manera recursiva con los nodos candidatos seleccionados. Según los investigadores, el incremento del número de candidatos mejora la calidad de la expansión, aunque incrementa también el tiempo de entrenamiento. Sin embargo, según los experimentos realizados, el incremento de candidatos por nodo no necesariamente mejora la precisión.

Proximity Forest: Método de clasificación

El proceso de clasificación de una serie Q se inicia en el nodo raíz del árbol principal. Cada subnodo le aplica a la serie Q una medida de distancia con el resto que series que tiene almacenadas. Luego se pasa dicha serie Q a la rama del nodo cuya serie más se le asemeja. Entonces la serie Q va recorriendo el árbol y repitiendo el mismo proceso hasta llegar a un nodo hoja que es el que contiene la clase a la que pertenece Q. Este proceso se repite para cada árbol del bosque de proximidad y el elige el aquella clase mayoritaria entre los árboles resultantes.

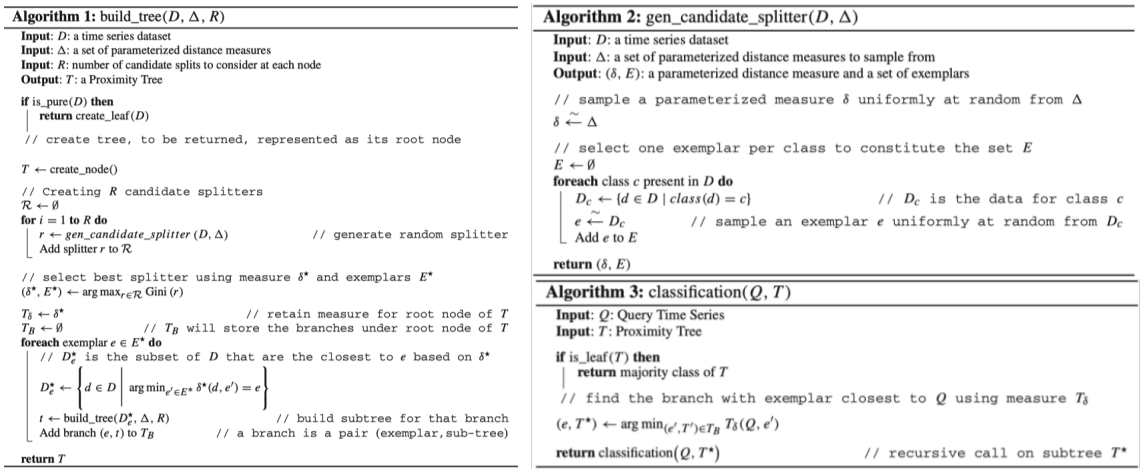


Figura 1.13: En la parte izquierda se encuentra el pseudocódigo de construcción de un árbol de proximidad. En la parte superior derecha se muestra el algoritmo de expansión de un nodo hoja y en la parte inferior derecha se muestra el algoritmo de clasificación [28].

Proximity Forest: Complejidad

Según los autores del artículo que presenta esta nueva técnica [28], la complejidad que tiene esta técnica es de orden $O(k \cdot \log n \cdot c \cdot l^2)$, siendo k el número de árboles, $\lg n$ la profundidad de cada árbol, c la distancia entre dos secuencias y l^2 la longitud de serie al cuadrado. Dicho artículo también detalla el análisis de la complejidad así la influencia del número de árboles y candidatos por nodo.

1.4.2 Lower Bounds Enhanced

Se trata de una medida de distancia elástica que consiste en aplicar a DTW unas cotas inferiores o *Lower Bounds* para acelerar el análisis de las series temporales y minimizar el número de cálculos. Hasta ahora, la técnica que ofrece una buena relación *precisión / tiempo de ejecución* es el *Nearest Neighbours* con DTW [30], pero solo es competitiva cuando se usan ventanas de distorsión o *warping windows* [15]. El motivo de la publicación de esta medida de distancia se debe a que buscar un tamaño

[1] Aquí se puede consultar una breve explicación de la impureza de Gini: `https://victorzhou.com/blog/gini-impurity/`

de ventana efectivo puede consumir un alto coste de tiempo al analizar su valor desde el 0% al 100% del tamaño de serie. Por tanto, la estrategia clave era aplicar una medida de distancia basada en cotas que pueda descartar vecinos próximos sin que haga falta realizar cálculos DTW. Las medidas de distancia basada en cotas inferiores más utilizadas son LB_{KIM} y LB_{KEOGH}. LB_{KIM} [31] es el mas rápido para aquellas medidas que utilizan un tamaño de ventana pequeño pero su precisión es muy baja y apenas aumenta conforme aumenta el tamaño de venta. En cambio, LB_{KEOGH} [32] presenta una mejor relación entre la precisión y el tiempo de ejecución. Sin embargo, a medida que aumenta el tamaño de ventana, puede tener una precisión mucho más baja que LB_{KIM}. Otra medida basada en costas, $LB_{IMPROVED}$ [33], ofrece mejores resultados para tamaños de ventana más grandes.

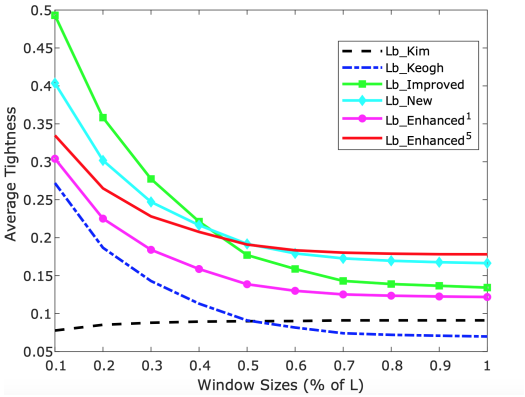


Figura 1.14: Comparativa de medidas de distancia basadas en cotas inferiores (Lower Bounds). Gráfica extraída del artículo *Elastic bands across the path: A new framework and method to lower bound DTW* [30]

Esta nueva medida de similitud que se presenta, conocida como $LB_{ENHANCED}$ y publicada por investigadores de la Universidad de Monash [30], se basa en restricciones del camino de distorsión o *Warping Path* que se obtiene aplicando la medida de distancia DTW [30]. Recordemos que el *Warping Path*, como se mencionó en la sección de medidas de distancia, es una array de pares de valores en los que cada elemento p_i, siendo i el i-ésimo par de puntos, corresponde a la distancia que existe entre la posición a_i de una secuencia R y b_i de una secuencia S. Las restricciones requieren que, dada una matriz M, donde el número de filas representa a una secuencia S y el número de columnas representa a una secuencia R, el valor de S_1 debe ser el que hay en la posición (1,1). Luego, las restricciones aseguran el valor S_2 depende de sus vecinos. Así pues, se asegura que $S_2 \in \{(1,2),(2,1),(2,2)\}$ y así su sucesivamente (Recordemos el recorrido de la matriz de distorsión en la anterior sección, Medidas de distancia elásticas: Dynamic Time Warping). Si continuamos con esta secuencia de conjuntos, obtenemos lo que conoce como *left bands*, que funciona como cota .

$$\mathcal{L}_i^W = (max(1, i-W), (max(1, i-W)+1, i)...(i,i), (i, i-1)...(i, max(1, i-W))$$

La suma de todos los valores mínimos de cada posición es lo que se aplica como una cota inferior o *lower bound* en la medida distancia DTW. En el otro extremo de la matriz de distorsión, las restricciones de cota requieren que S_L, que se encuentra en la casilla en la posición más extrema a la derecha, debería ser (L, L) siendo L la longitud de serie. Esta secuencia de conjunto se conoce como *right bands*, que el número de bandas a la derecha.

$$\mathcal{R}_i^W = (min(L, i+W), (min(L, i+W)+1, i)...(i,i), (i, i-1)...(i, min(L, i+W))$$

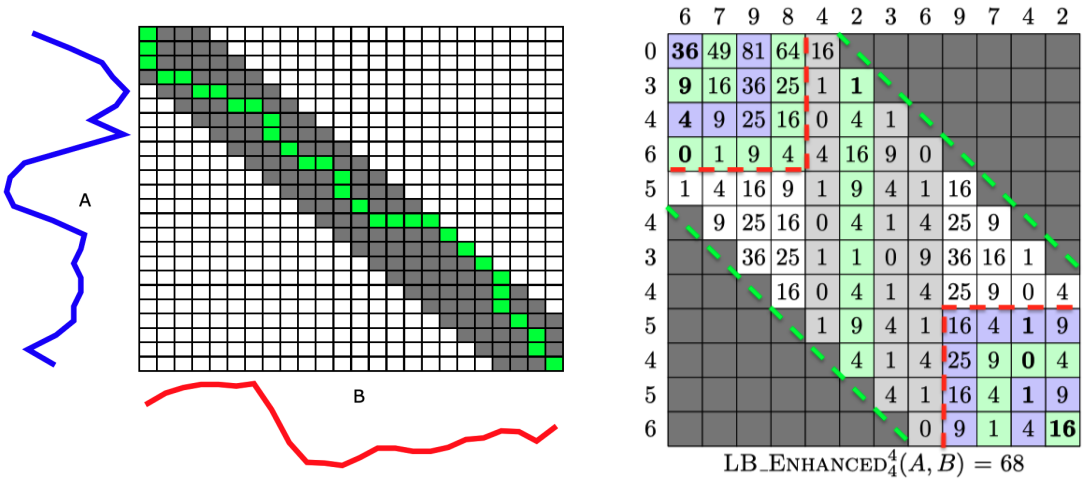


Figura 1.15: A la izquierda, se muestra la matriz de distorsión de DTW, que muestra el *warping path* delimitado por una ventana W. A la derecha, la matriz de distorsión de $LB_{ENHANCED}$ con la aplicación de cotas inferiores y superiores. Se puede ver como un valor V = 4 hace que el recorrido, en la parte izquierda, no descienda de cuatro celdas. En la parte derecha, tampoco se pasa de 4 celdas, estableciendo el límite superior. Todo el recorrido está a su vez delimitado por W, que es la zona delimitada por la linea discontinua verde. Las imágenes fueron sacadas del artículo *Elastic bands across the path: A new framework and method to lower bound DTW* [30]

Lower Bounds Enhanced: Algoritmo

Como se muestra en la Figura **??**, primero se establecen el número de bandas a utilizar. Antes de continuar es muy importante recalcar que se pasan como argumentos dos series A y B, los envoltorios inferiores y superiores de B, que son $\mathbb{U}^B$ y $\mathbb{L}^B$. Cada punto $\mathbb{U}_i^B$ contiene el valor máximo entre las distintas posiciones B_j siendo j el rando de valores entre el máximo entre 1 y $i-W$, y el mínimo entre L y $i+W$. Cada punto $\mathbb{L}_i^B$ contiene el valor mínimo entre las distintas posiciones B_j siendo j el rango de valores entre el máximo entre 1 y $i-W$, y el mínimo entre L y $i+W$.

$$\mathbb{U}_i^B = \max_{\max(1,i-W)\leq j\leq \min(L,i+W)}(B_j)$$
$$\mathbb{L}_i^B = \min_{\max(1,i-W)\leq j\leq \min(L,i+W)}(B_j)$$

Cabe destacar que el algoritmo no contempla los cálculos realizados para obtener las cotas. De hecho, los autores de esta medida de distancia afirman que el cálculo de las cotas no está incluido dentro del estudio teórico del algoritmo y la complejidad [30]. La línea 3 hasta la 11 calculan la suma de las distancias mínimas entre las posiciones de las de las cotas $\mathcal{L}$ y $\mathcal{R}$ de A. Si nos fijamos, el número de bandas calculado al principio, es el que determina la cota inferior que puede recorrer la matriz. El número de bandas viene condicionado por el valor del parámetros *speed-tighness* y la longitud de serie. Un valor de V pequeño requiere menos cálculos, pero genera cotas muy frágiles y una capacidad de predicción baja. Si el valor de V es grande, aumentará la precisión en la predicción, pero conllevará hacer más cálculos y, por tanto, más tiempo. En total, calculan la suma de las distancias mínimas entre las posiciones de las de las cotas $\mathcal{L}$ y $\mathcal{R}$ de una A en función de V. Lo que hace el algoritmo es hacer un recorrido de 1 hasta la longitud de las series. Mientras el índice del recorrido i es menor o igual que el V, se suma el mínimo valor de $\mathcal{L}_i^W$. Si el índice es mayor que V, se suma el mínimo valor de $\mathcal{R}_i^W$. En caso de que el resultado de la suma sea mayor que

la distancia actual al vecino más próximo, se dejan de realizar cálculos y se procede a ejecutar el algoritmo LB_{KEOGH}. Sin embargo, si resultado de la suma es menor que la distancia con el vecino más próximo, se recorre la secuencia A desde la posición establecida por el número de bandas, hasta la posición dada por la longitud de la serie menos el número de bandas establecido. En dicho recorrido, si una marca temporal A_i es mayor que la cota superior de B, $\mathbb{U}_i^B$, entonces añadimos al resultado la distancia que hay entre A_i y $\mathbb{U}_i^B$. En caso de que se sea menor que la cota inferior de B, $\mathbb{L}_i^B$, entonces añadimos al resultado la distancia que hay entre A_i y $\mathbb{L}_i^B$.

Según las conclusiones sacadas por los investigadores que publicaron esta técnica, todavía no se ha encontrado un valor óptimo de V ya que es algo que está en proceso de investigación, pero según sus experimentos un valor de $V = 5$ ofrece resultados más rápidos y con mejor precisión que el resto de técnicas basadas en distancias con acotamiento o *Lower Bounds* en la mayoría de los datasets utilizados [30]. Esta técnica tiene un tiempo de ejecución $O(L)$, siendo L la longitud de serie. Sin embargo, hay un pequeño matiz: El cálculo de las cotas inferiores y superiores no está incluido dentro del estudio teórico, según señalan los autores [30].

2 Objetivos y metodologías

En esta sección detallaremos los objetivos que queremos alcanzar en este trabajo de fin de grado, así como las herramientas que utilizaremos para ello.

2.1 Objetivos

El objetivo en este trabajo de fin de grado es implementar, analizar y comparar distintas técnicas de clasificación de series de tiempo basadas en distancias elásticas, más concretamente en DTW. Dicha comparativa estará formada por la modelos de técnicas Proximity Forest, KNN con Lower Bounds Enhanced y KNN con Dynamic Time Warping respectivamente. El trabajo de fin de grado se compondrá de una fase de análisis de precisión y tiempo de ejecución de estas dos técnicas, y luego una comparativa de estos dos con KNN con DTW ya que está considerada como la técnica que mejor ofrece una relación entre la precisión a la hora de predecir y el tiempo de ejecución [30]. Para alcanzar nuestro objetivo, vamos a descomponerlo en nuestros

2.1.1 Estudio de técnicas de clasificación a comparar

Validación de la implementación de Proximity Forest Comprobaremos que el desarrollo que hemos realizado para implementar esta técnica sea correcto. Cabe mencionar que la implementación se ha realizado replicando el proyecto que los investigadores desarrollaron en Java. En nuestro caso, lo hemos replicado en lenguaje Python con ligeras modificaciones y utilizaremos el proyecto de Java para realizar comparaciones y verificar que la implementación es correcta. Asimismo, realizaremos una validación basada en una crossvalidación para comprobar que nuestra implementación es robusta.

Validación de la implementación KNN con LbEnhanced Comprobaremos que el desarrollo que hemos realizado para implementar esta técnica sea correcto. Cabe mencionar que la implementación se ha realizado replicando el proyecto que los investigadores desarrollaron en Java pero, al contrario de lo que hicimos con Proximity Forest, hemos replicado, también en lenguaje Python, solo la parte que nos interesaba, que fue la medida de distancia. Luego, hemos integrado esa medida de distancia en un proyecto de KNN realizamos en Python que encontramos en un repositorio público y accesible en `GitHub`.

2.1.2 Estudio de técnicas de clasificación a comparar

En este subobjetivo nos centraremos en el análisis de la precisión y el tiempo de ejecución de cada técnica de estado de arte que hemos mencionado. Cabe destacar que en cada artículo en donde se publica cada una de estas técnicas, sus respectivos autores mencionan que ciertos valores de los hiperparámetros de cada técnica ofrecen resultados óptimos en la mayoría de conjuntos de datos utilizados. Sin embargo, la implementación propia desarrollada en Python no ofrece los mismos resultados óptimos que afirman los autores y, por tanto, se debe realizar un análisis profundo de los hiperparámetros que mejores resultados ofrecen para nuestro desarrollo.

Evaluación del tiempo de ejecución y la precisión de Proximity Forests Los autores del artículo que presenta este nuevo clasificador [28], ofrecen de manera pública su implementación en java. Nuestro trabajo ha sido implementar el mismo proyecto en Python con ciertas modificaciones. Analizaremos los mejores hiperparámetros y seleccionaremos y validaremos un modelo para la comparativa.

Evaluación del tiempo de ejecución y la precisión de la implementación KNN con LbEnhanced Al igual que Proximity Forest, hemos implementado LBEnhaced en Python replicando el proyecto realizado por los investigadores en Java y analizaremos los hiperparámetros que ofrezca un mejor ajuste. Validaremos nuestra implementación realizando una cross validación para asegurarnos de que nuestra implementación es correcta y robusta.

2.1.3 Comparación de las técnicas anteriores con KNN con DTW

Habiendo analizado, tanto Proximity Forest como KNN-LbEnhanced, y habiendo sacado los mejores hiperparámetros de cada uno, procederemos a realizar una comparación en términos de precisión y tiempo de ejecución con KNN-DTW, ya que este se considera como la técnica de clasificación que ofrece una mejor relación entre la precisión y el tiempo de ejecución.

2.2 Metodología y herramientas de trabajo

Lenguaje de desarrollo ¿Por qué Python? Python está adquiriendo notoriedad en el campo de las ciencias de datos a pasos agigantados. Este lenguaje es sencillo, muy intuitivo y ofrece muy buenas librerías para visualización, cálculos, análisis de datos y diversas herramientas y frameworks para el Machine Learning [1]. También existe el lenguaje `R`, que es un lenguaje basado en programación funcional, mientras que Python sirve de muchos más propósitos y ofrece mayor facilidad para integrarlo con otros lenguajes. Además, para algoritmos de clasificación aplicados a series temporales, ya existen técnicas y algoritmos disponibles en librerías que podremos utilizar para nuestro trabajo de fin grado.

Entorno de desarrollo La parte experimental de este trabajo se realiza en **PyCharm**. PyCharm es un potente entorno de desarrollo en Python bajo licencia pero cuyo uso está permitido para estudiantes de centros académicos [2]. También se realiza el tratamiento y visualización de resultados obtenidos en la herramienta Jupyter, que es una herramienta web que permite crear documentos, mostrar gráficos resultantes de trozos de código, entre otras posibilidades. [3]

Entorno de sistema operativo La parte de desarrollo se realiza bajo el Sistema Operativo OSX, que se basa en UNIX. Cuenta con un procesador Intel Core i5 de 2.7 Ghz, memoria RAM de 8 GB y una tarjeta gráfica Intel Iris Graphics 6100 1536 Mb. La parte de ejecución de pruebas y experimentos se realiza en un servidor Linux con 4 procesadores x86_64 de 32 y 64 bits de 3 Ghz, y una memoria RAM de 8 Gb.

[1] Lenguage Python para Machine learning: `https://iartificial.net/librerias-de-python-para-machine-learning/`

[2] Más información sobre PyCharm en: `https://www.jetbrains.com/es-es/pycharm/`

[3] Más información sobre Jupyter en: `https://jupyter.org`

3 Experimentación, datos utilizados y resultados

En este capítulo nos centramos en el análisis y estudio de las técnicas de clasificación que vamos a comparar y trataremos de evaluar los hiperparámetros que ofrezcan mejores resultados en cuanto a tiempo y precisión se refiere. Es importante recalcar que los datos de partida que utilizaremos no pasarán por ningún tipo de preprocesado o *Data Cleansing* ya que este trabajo de fin de grado se enfocará en la comparación de los distintos clasificadores de series temporales en el tiempo y la precisión, y no en el tratamiento de conjuntos de datos para ser entrenados. Por tanto, los datasets estarán listos para poder utilizarse sin tener que realizar ninguna labor previa de preprocesado.

3.1 Datos utilizados para los experimentos

Antes realizar los experimentos necesarios que nos ayuden a alcanzar nuestros objetivos, primero vamos a detallar que conjuntos de datos o datasets vamos a utilizar y la medida de performance que vamos a utilizar. Todos estos dataset se encuentran disponibles en la plataforma `http://www.timeseriesclassification.com/dataset.php`. En dicho en enlace se puede encontrar una amplia lista de datasets de series temporales con distinto tamaño.

Datasets utilizados En la plataforma donde se encuentran estos datasets, hay dos conjuntos para cada tipo: Uno para realizar el entrenamiento y otro para realizar el testing.

Dataset	Training Size	Testing Size	Length	Num. Classes
Chinatown	20	345	24	2
ERing	30	270	65	6
FacesUCR	200	2050	131	14
FreezerRegularTrain	150	2850	301	2
ItalyPowerDemand	67	1029	24	2
MelbournePedestrian	1194	2439	24	10
Plane	105	105	144	7
RacketSports	151	152	30	4
SharePriceIncrease	965	965	60	2
SmoothSubSpace	150	150	1570	3
SonyAIBORobotSurface1	20	601	70	2

Tabla 3.1: Datasets utilizados para entrenar y validar nuestros modelos

Datasets para validación Utilizaremos los datasets que se muestran abajo para realizar la validación necesaria para comprobar que nuestro algoritmo es correcto

Dataset	Training Size	Testing Size	Lenght	Num. Classes
ElectricDeviceDetection	623	3767	256	2
ItalyPowerDemand	67	1029	24	2
Wafer	1000	6164	152	2
DodgerLoopGame	20	138	288	2
FreezerRegularTrain	150	2850	131	14
SmoothSubspace	150	150	1570	3
Chinatown	20	345	24	2
InsectWingbeatSound	30000	20000	30	10
FacesUCR	200	2050	131	14
RacketSports	151	152	30	4
ERing	30	270	65	2

Tabla 3.2: Datasets utilizados para realizar una cross-validación

Medida de performance Para la métrica de performance utilizaremos la precisión o `Accuracy` [1], ya que esta es la más directa en la calidad de los clasificadores, su valor se encuentra entre 0 y 1, y es la más sencilla.

Método de validación Para validar nuestras técnicas, utilizaremos una crossvalidación repetida de 10 pliegues. Esta se conoce como `Repeated 10-Fold crossvalidation`. Esta técnica divide el conjunto de entrenamiento en 10 pliegues, de los cuales 9 se utilizarán para entrenar el modelo y 1 para validarlo. La ventaja es que estas observaciones se utilizan para entrenar y validar, y en cada iteración se utiliza un pliegue de validación diferente. Al final, se extrae la media de resultados de precisiones obtenidas. En este enlace se puede ver una comparativa más detallada de las distintas medidas de validación: `https://towardsdatascience.com/validating-your-machine-learning-model-25b4c8643fb7`.

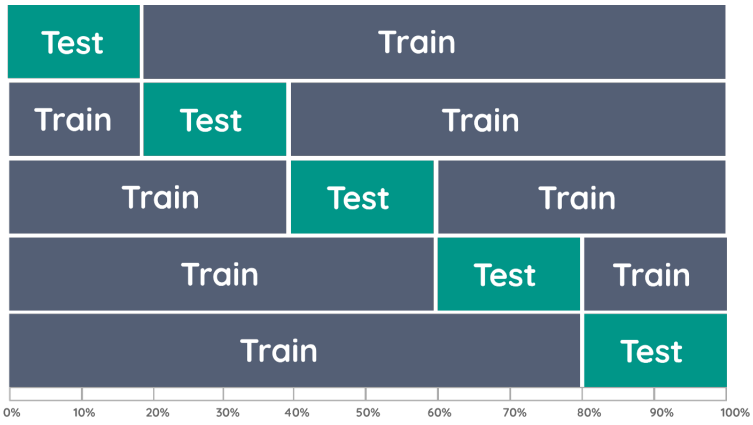


Figura 3.1: Esquema de una crossvalidation. Esta imagen fue extraída del enlace `https://towardsdatascience.com/validating-your-machine-learning-model-25b4c8643fb7`

[1] Para ver una comparativa entre las distintas medidas de performance, podemos acceder al siguiente enlace: `https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3`

3.2 Proximity Forest: Análisis y validación

Con el fin de justificar los resultados y conclusiones obtenidas, vamos a realizar un estudio para analizar el rendimiento y la efectividad de esta técnica implementada íntegramente en lenguaje `Python`. Sin embargo, la implementación de esta técnica ha sido una réplica, con ciertas variaciones, de la publicada por los autores del artículo que presenta este algoritmo, que estaba implementada en lenguaje `Java` [2]. El motivo de su implementación en `Python` era poder comparar, con el mismo lenguaje, los clasificadores que se quieren analizar en este trabajo. Por tanto, se ha procedido a implementar esta técnica basándonos en la implementación realizada por los investigadores y luego comparando los resultados en ambas plataformas. Cabe recordar que el tiempo de ejecución en este caso puede variar en función de cada compilador e intérprete para cada lenguaje.

Como se ha comentado anteriormente, la técnica `Proximity Forest` se basa en un conjunto de árboles de decisión que mide la similitud entre series a partir de medidas de distancia elegidas de manera aleatoria. En nuestro caso, solo utilizamos una sola medida de distancia, la técnica *Dynamic Time Warping* o DTW. Dicha medida de similitud, como comentamos en secciones anteriores, se utiliza para comparar de manera efectiva dos secuencias del mismo tipo y recorrido pero que presentan deformidades y/o intervalos dispares en el eje del tiempo. A lo largo de esta sección, se van a tomar los distintos conjuntos de datos seleccionados y se realizará un estudio sobre los valores de los hiperparámetros que influyan en la precisión a la hora de predecir, así como el tiempo de ejecución a la hora de clasificar una serie.

Según la conclusión que sacaron los investigadores, los valores idóneos, en la mayoría de los datos utilizados, para los hiperparámetros eran de 100 árboles y 5 candidatos por ramificación [34]. Sin embargo, debemos comprobar si esos valores son idóneos para nuestra implementación.

3.2.1 Sktime y motivo de implementación propia

Durante la implementación de nuestro benchark se encontró una herramienta en `Python` bastante útil para comparar series temporales llamada `sktime`. La herramienta `sktime` [3] es una *toolkit* que se utiliza para comparar distintas técnicas de clasificación de series temporales y que suele ser muy útil a la hora de analizar el rendimiento en el tiempo y la precisión de varias técnicas de distinto tipo. En un principio se pensaba utilizar el benchmark `Proximity Forest` ofrecido por `sktime` al estar ya implementado y listo para su uso. Sin embargo, como se verá a continuación, tiene un tiempo de ejecución enorme.

Sktime: Comparación con implementación propia

Para esta comparativa se seleccionaron varios datasets para justificar nuestra implementación frente a la de `sktime`. Cabe destacar que los hiperparámetros elegidos en ambos benchmarks son de 100 árboles y 5 candidatos por expansión, que son los valores que presentan los mejores resultados para el *Proximity Forest*, según las conclusiones que sacaron los autores [29].

[2] El proyecto realizado por los investigadores se puede encontrar a través de este enlace: `https://github.com/fpetitjean/ProximityForest`

[3] La herramienta `sktime` está disponible a través de este enlace: `https://github.com/alan-turing-institute/sktime`

Name	Accuracy PF Propio	Accuracy PF sktime	T.ejec. Propio	T.ejec. sktime
Chinatown	0.8921	0.9737	2.85	220.928
Plane	0.9904	1.000	1.00	332.233
RbtSurface	0.8174	0.886	13.47	818.131
RckSports	0.5921	0.394	23.30	413.199
ERing	0.6259	0.522	6.68	465.354
ItalyPower	0.9543	0.962	12.71	903.302

Tabla 3.3: Comparación de los resultados de precision entre nuestra implementación y la de Sktime

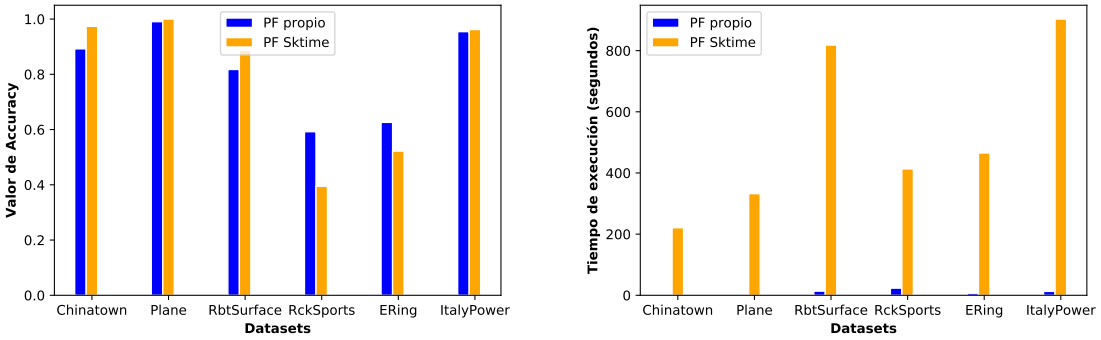


Figura 3.2: Comparativa de la precisión y el tiempo de ejecución utilizando varios datasets. Imagen de elaboración propia.

Como se puede ver en la Figura 3.2, tras ejecutar varios datasets, podemos ver claramente que nuestra implementación ofrece una mejora del rendimiento más que evidente y en algunos casos, como `ItalyPowerDemand`, ofrece el mismo `accuracy` pero con un tiempo de ejecución 80 veces menor. Otros datasets fuera de esta gráfica (Figura 3.2) han tenido que ser descartados debido al altísimo consumo de tiempo de ejecución que conllevaban en `sktime` y no han podido terminar de ejecutarse, demostrando su escasa escalabilidad. Por tanto, en base a los resultados mostrados en la Figura 3.2 y en la Tabla 3.3, muestran que la técnica ofrecida por `sktime` no es escalable en el medio plazo y, por tanto, este benchmark queda totalmente descartado, quedándonos con el benchmark propio que hemos desarrollado.

3.2.2 Mejores hiperparámetros, tiempo y limitaciones

Según los investigadores que publicaron este nuevo algoritmo de clasificación, un modelo de Proximity Forest formado por 100 árboles y 5 nodos candidatos por expansión ofrece buenos resultados en la mayoría de los datasets utilizados [34]. Sin embargo, es posible que para nuestra implementación, y para los datasets que se utilizan en este trabajo, sea necesario ver qué valores son lo idóneos para alcanzar un mejor rendimiento. Es necesario comprobar si los valores que sugieren los investigadores son correctos o, si en nuestro caso existen otros valores que mejoran la precisión de nuestro benchmark.

Parámetros de partida Para cada dataset, probaremos con **20, 35, 50, 75** y **100 árboles** respectivamente, y de 2 a 7 candidatos por expansión de nodo,

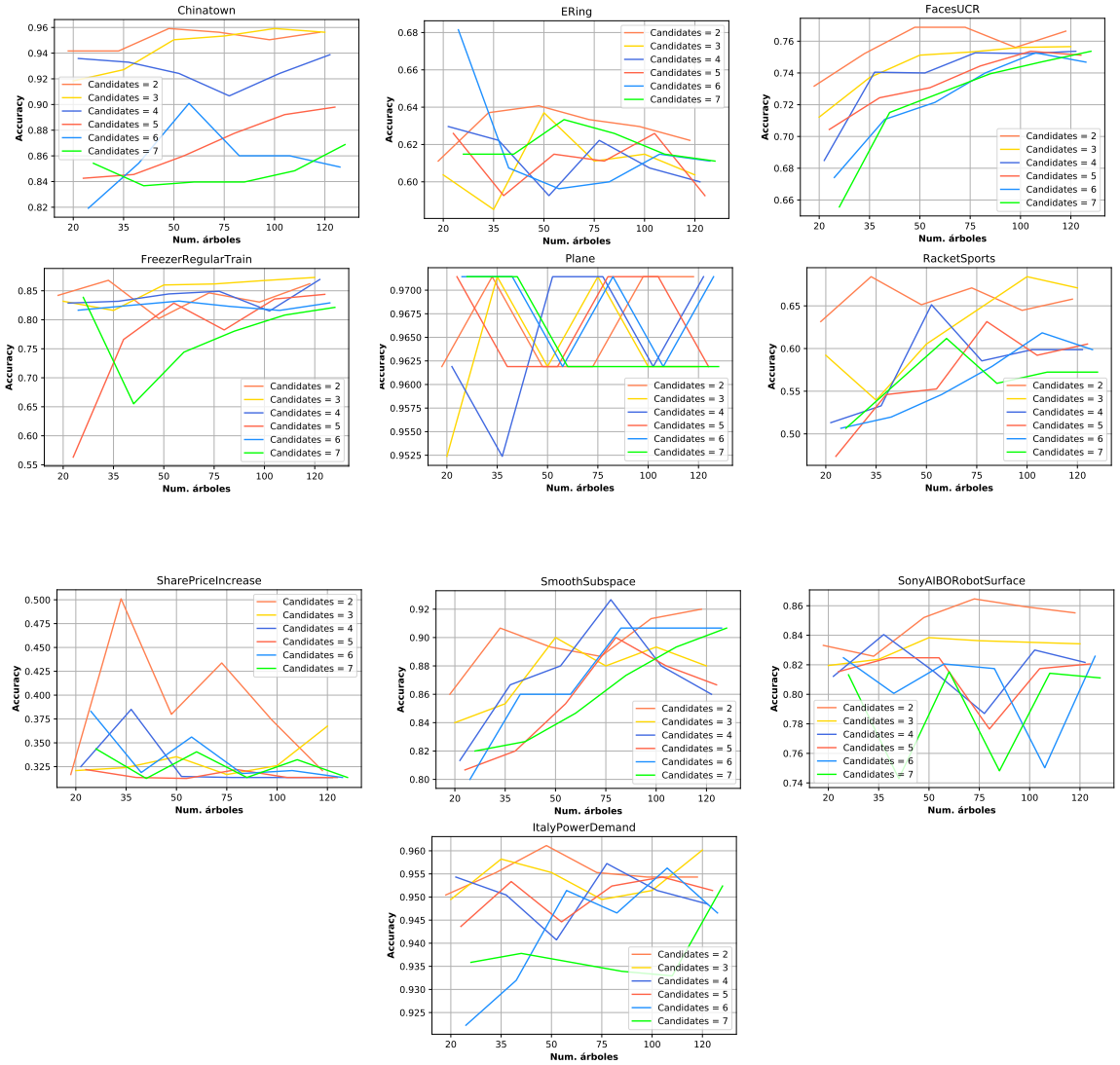


Figura 3.3: Gráficos, de elaboración propia, que muestran la precisión calculada para cada dataset utilizado.

Según podemos ver en los gráficos (Figura 3.3), los resultados no tienen grandes diferencias entre sí, pero sin embargo, las ejecuciones que presentan mejores resultados suelen aquellas cuyo número de árboles es de 35 y 120. En caso de tener 35 árboles, podemos observar que el valor de `accuracy` suele ser bueno cuando existen 2 candidatos para cada nodo. En caso de contar con 120 árboles, las diferencias de precisión suelen ser mínimas a la hora de elegir el número de candidatos por nodo, aunque se puede ver una ligera mejora cuando este número oscila entre 2 y 3. El resto de ejecuciones con distinto número de árboles y candidatos presenta resultados similares con ciertas variaciones. Sin embargo, tenemos que tener en cuenta el tiempo de ejecución, y este es directamente proporcional al número de árboles y candidatos [4]. Por tanto, es necesario elegir los valores adecuados, ya que es mejor elegir el número mínimo de árboles y candidatos para tener una buena relación entre la predicción y el tiempo de ejecución, que elegir otros que solo posiblemente tendrán, como mucho, unas décimas de mejora en la capacidad predicción pero que conllevarán un tiempo de ejecución mucho mayor.

[4]Recordemos que Proximity Forest tiene una complejidad de orden $O(k \cdot \log n \cdot c \cdot l^2)$ [28]

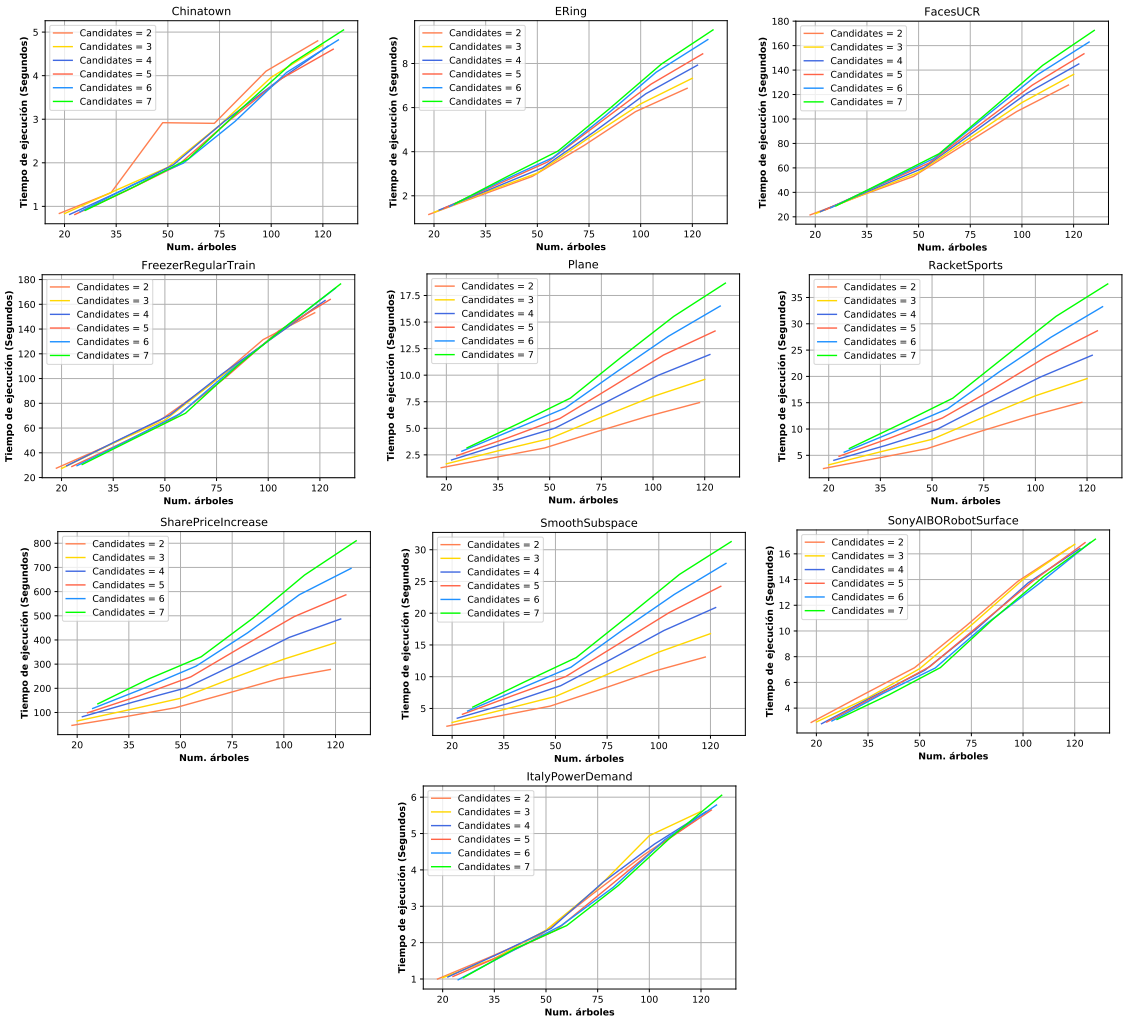


Figura 3.4: Gráficos, de elaboración propia, que muestran el tiempo que ha llevado ejecutar cada dataset utilizado.

Si nos fijamos en los gráficos de tiempo de ejecución (Figura 3.4), podremos observar que el número de árboles y candidatos incrementa el tiempo que tarda un algoritmo en realizar una predicción. Por tanto, no interesa escoger unos hiperparámetros que encarezcan el tiempo si ofrecen una capacidad predictiva que apenas mejora frente a otros modelos con valores menores que requieran menos tiempo y su precisión se decremente en apenas centésimas. El tiempo de ejecución tiende a duplicarse conforme se duplica el número de árboles y por regla general suele tardar más al disponer de más candidatos por nodo. Recordemos que, según los investigadores, el tiempo de ejecución de orden $O(k \cdot \log n \cdot c \cdot l^2)$, por lo que el tiempo se verá severamente incrementado conforme existan más árboles y candidatos.

Tras observar los resultados caso por caso, vamos a enfocarnos, dentro del rango de valores seleccionado, en los modelos cuyo número de árboles de 20 y 35, y de 2 a 4 candidatos para cada nodo a expandir.

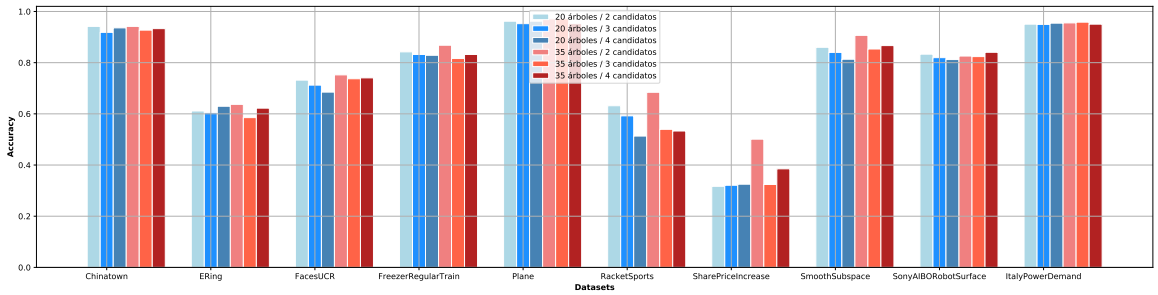


Figura 3.5: Comparación de modelos de Proximity con 20 y 35 árboles, y 2 a 4 candidatos por nodo. Imagen de elaboración propia.

Las observaciones que podemos sacar aquí son que las diferencias, a priori, no parecen muy grandes. Para cada modelo, el valor de la precisión es bastante similiar y apenas se diferencia en centésimas. Parece claro que con un Proximity Forest de 35 árboles y 2 candidatos se obtiene una ligera mejora. Sin embargo, puede que esta mejora no nos interese teniendo en cuenta que puede tardar el doble de tiempo que un modelo con 20 árboles y 2 candidatos. Por tanto, la cuestiones a abordar son cuántos árboles debemos utilizar para construir un Bosque de proximidad y cuántos candidatos por nodo.

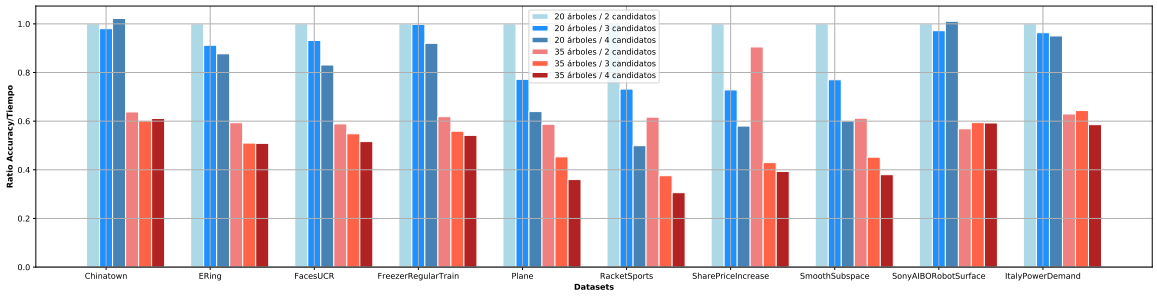


Figura 3.6: Ratio de precisión/tiempo de ejecución. Esta gráfica, elaborada durante el análisis, nos ayuda visualizar que modelo ofrece mejor relación entre la capacidad de predicción y el tiempo de ejecución

Según se ve en la Figura 3.6, el modelo compuesto por 20 árboles y 2 candidatos tiene una mejor relación entre la precisión y el tiempo de ejecución. La diferencia es evidente y, como se muestra también en la Figura 3.5, la diferencia de precisión entre los distintos modelos no es muy grande, por lo que no interesa escoger un modelo que tenga mejor capacidad predicitiva si este conlleva mucho más tiempo de ejecución.

Modelo final para comparar Según podemos concluir, si el conjunto de datos a entrenar no es muy grande o el tiempo de ejecución no es decisivo, se puede construir un bosque de proximidad formado por **35 árboles** y 2 **candidatos** por nodo para la expansión. Por otra parte, si el tamaño del conjunto de datos es muy grande, lo ideal sería construir un bloque de proximidad formado por **20 árboles** y **2 candidatos** por expansión. Por tanto, y basándonos en los resultados que hemos obtenido, elegimos utilizar un modelo **20 árboles** y **2 candidatos**.

3.2.3 Evaluación con K-Fold Crossvalidation

Para evaluar la capacidad de nuestra implementación para predecir una respuesta variable, realizamos una crossvalidación, en nuestro caso es una (`Repeated 10-Fold cross-validation`), que dividen el conjunto de datos de entrenamiento en 10 pliegues, en la que un pliegue se utiliza como conjunto de validación y los otros 9 como conjuntos de entrenamiento. Este proceso se itera para cada pliegue. La crossvalidación se repite 10 veces y vamos a utilizar el conjunto de datasets disponibles para validación que mencionamos en el apartado de datos de partida [5]. En este caso, al ver que el modelo que ofrece un mejor ajuste es aquel que tiene 20 árboles y 2 candidatos por nodo, vamos a validar dicho modelo para comprobar que su capacidad predictiva es sólida.

Dataset	1	2	3	4	5	6	7	8	9	10	media
EDevDet	0.95	0.95	0.935	0.90	0.967	0.95	0.95	0.967	0.919	0.909	0.9409
ItalyPowerDd	0.85714	1.0	1.0	1.0	1.0	1.0	0.714	1.0	1.0	1.0	0.949
Wafer	0.94	1.0	0.97	1.0	1.0	1.0	0.99	0.99	1.0	1.0	0.9624
DdgLoopGame	0.5	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.5	1.0	0.9343
FRgTrain	0.66	1.0	0.933	1.0	1.0	0.93	0.933	0.533	0.733	0.933	0.9207
SmoothSpce	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.934
Chinatown	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9434
InsectWing	0.409	1.0	1.0	0.954	1.0	1.0	0.909	1.0	1.0	0.954	0.9408
FacesUCR	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9452
RcktSports	0.66	1.0	1.0	1.0	1.0	1.0	0.866	1.0	0.933	1.0	0.9453
ERing	0.33	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9442

Tabla 3.4: Tabla de resultados de precisión de cada iteración y la media de la `crossvalidation`.

Los resultados de la Tabla 3.4 nos muestran que en los datasets apenas se pueden apreciar valores dispares en algunos datasets y las predicciones realizadas en cada iteración son bastante parecidas. Cae destacar que algunos datasets son pequeños y su división en 10 pliegues hace que el pliege de validación sea muy pequeño y por tanto la precisión en muchos casos pueda ser de 1.0 o del 100%. Luego de ver la tabla de crossvalidación, hacemos una comparación con la predicción realizada con el conjunto de validación que tenemos para cada dataset y la media de crossvalidación. La Tabla 3.5 muestra que los resultados de la crossvalidación son mayores que los del conjunto de validación. Este *underfitting* se puede deber a que el conjunto de testing es mucho mayor que el conjunto de entrenamiento, y por tanto es evidente que las predicciones sean menores ya que no es lo mismo validar con un conjunto, por ejemplo, de 10 series que con otro de 2000 series. sin embargo, datasets como `ElectricDeviceDetection` o `Wafer`, cuyos conjuntos de entrenamiento son grandes, presentan resultados de validación con test dataset parecidos a los de la cross-validación.

Con esto, se podría decir la evaluación muestra unos resultados favorables de nuestro proyecto, ya que los resulatados de la cross-validación no muestran disparidades tan pronunciadas en las gráficas de los distintos datatsets, cada uno de distinto tamaño y longitud de serie. Podríamos afirmar que nuestro benchmark es sólido y aceptable para un uso normal.

3.2.4 Comparación con Proximity Forest original

Tras hacer una cross-validación y tener unos buenos resultados, es necesario hacer la comparación de nuestra implementación de Proximity Forest en Python con la implementación original en

[5] Todos los datos utilizados para este trabajo se pueden encontrar en este enlace `http://www.timeseriesclassification.com/dataset.php`

Dataset	Media Accuracy Crossvalidación	Accuracy Testing dataset
ElectricDeviceDetection	0.9409	0.8635
ItalyPowerDemand	0.9409	0.961
Wafer	0.9624	0.986
DodgerLoopGame	0.9343	0.81
FreezerRegularTrain	0.9207	0.842
SmoothSubspace	0.934	0.86
Chinatown	0.9434	0.941
InsectWingBeat	0.9408	0.577
FacesUCR	0.9452	0.731
ERing	0.9453	0.61
RacketSports	0.9453	0.644

Tabla 3.5: Comparación entre la precisión ofrecida por el dataset de validación y la media de la crossvalidación

Java, desarrollada por los autores de esta técnica, para poder comprobar que hace una predicción correcta y efectiva. Para ambas implementaciones se utilizan los hiperparámetros que hemos visto que presentan una mejor relación entre la precisión y el tiempo de ejecución: (a) Número de árboles = 20 (b) Número de candidatos por nodo = 2.

Comparativa de valores de precisión para cada lenguaje

Dataset	**Java accuracy**	**Python Accuracy**
Chinatown	0.965	0.9708
ERing	0.7296	0.6259
FacesUCR	0.8902	0.8059
FreezerRegularTrain	0.9091	0.8119
ItalyPowerDemand	0.9456	0.9621
MelbournePedestrian	0.9016	0.9106
Plane	0.9905	0.9524
SharePriceIncrease	0.6398	0.5383
SmoothSubspace	0.7933	0.9467
SonyAIBORobotSurface	0.8143	0.8227

Tabla 3.6: Diferencia de precisión entre Proximity Forest desarrollado en Java por los investigadores que publican este nuevo algoritmo de clasificación, y su réplica en Python desarrollada para este trabajo de fin de grado.

Según puede observarse en la Tabla 3.6, ambos modelos presentan valores muy similares, en algunos casos PF en Java es mejor que en Python, y viceversa. Se ha hecho una revisión rigurosa del desarrollo en Python para comprobar que las estructuras y algoritmos son correctos pero tras revisar varias veces el código que hemos implementado, no hemos podido saber por qué hay notables diferencias entre distintas implementaciones. Una hipótesis sería el tiempo que conlleva utilizar ciertas librerías puede variar en función del lenguaje y entorno de desarrollo, pero tampoco podemos asegurar tal información. En todo caso, los resultados son bastante parecidos, en algunos casos nuestra implementación muestra una mejora de la precisión, y nos ayuda a consolidar la robustez y la fiabilidad de nuestra implementación en Python.

3.2.5 Comparación con t-test

Se han comparado los resultados en la sección anterior y se ha comprobado que los resultados son similares, en términos de precisión, y que se sirven para defender la efectividad de nuestro benchmark en **Python** a la hora de realizar una predicción. Sin embargo, es necesario realizar un `t-test` para comprobar que la media de los resultados obtenidos en ambos clasificadores, Java y Python, no son por casualidad y tienen una correlación. En nuestro caso aplicaremos un `Student's t-test` que es una validación de una hipótesis estadística que se utiliza para ver si dos conjuntos tienen el mismo tipo de poblaciones [35]. El estudio consiste en verificar las valores medios de dos muestras para ver si diferentes entre sí calculando error de desviación típica, que puede interpretarse como el grado de diferencia entre dos conjuntos o muestras que tienen una media muy similar o idéntica [35]. Para realizar un t-test debemos realizar los siguientes pasos:

Condiciones Tomamos como muestras los valores de precisión que hemos sacado para cada dataset de ambas técnicas (Java y Python). La figura 3.7 nos muestra que existe una evidencia de que los valores de precisión de ambos clasificadores son similares y tienen un recorrido similar.

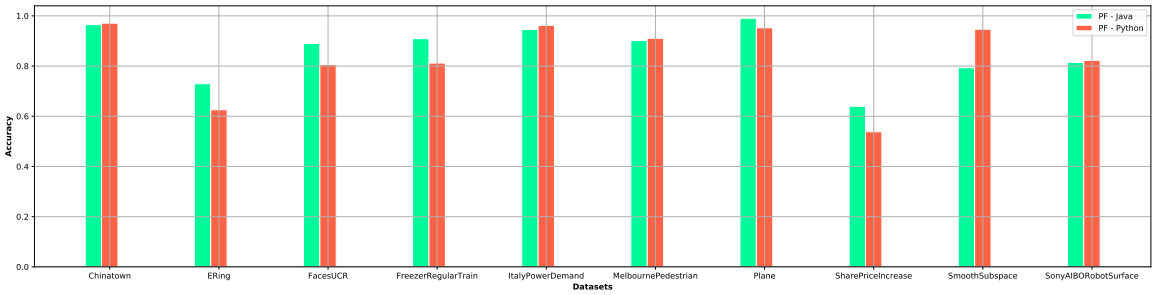


Figura 3.7: Gráfico de los valores de `accuracy` para cada dataset en ambas implementaciones (Imagen de elaboración propia).

Establecemos la hipótesis nula Esto significa que establecemos una hipótesis H_o en la que asumimos que no existe una diferencia significativa, en términos estadísticos, entre las dos muestras [6]. Para cada dataset, extraemos el valor de `accuracy` tanto para el clasificador en Java como en Python. Una vez extraídos, hacemos un listado de para cada tipo de lenguaje.

```
>>> java_accuracies = {0.965 , 0.7296, 0.8902, 0.9091, 0.9456, 0.9016,
0.9905, 0.6398, 0.7933, 0.8143}

>>> python_accuracies = {0.9708, 0.6259, 0.8059, 0.8119, 0.9621, 0.9106,
0.9524, 0.5383, 0.9467, 0.8227}
```

Establecemos el nivel de significancia Partimos de un valor de significancia α de 0.05, que es el que se utiliza para estudios científicos. El nivel de significancia determina la probabilidad de error que se quiere asumir a la hora de rechazar la hipótesis.

[6] Afirmación sacada de esta fuente: `https://www.statisticssolutions.com/manova-analysis-one-sample-t-test/`

Calculo de p-value y comparación de nivel de significancia El objetivo es calcular el `t-statistic`, que puede ser interpretado comparándolo con el valores críticos de las distribuciones [36]. Un valor crítico (cv) se puede calcular usando grados de libertad o *degrees of freedom* .

Primero, calculamos el grado de libertad. Para ellos debemos sumar la longitud de las dos muestras y restarle 2.

$$df = n_1 + n_2 - 2$$

Luego, calculamos el estándar de error.

$$SE = \sqrt{\frac{S_1}{n_1} + \frac{S_2}{n_2}}$$

siendo S la desviación típica y n la longitud de la lista de valores de `accuracy`.

Con el valor de significancia $\alpha = 0.05$ y el grado de libertad $df = 24$, obtenemos un `critical value` de 1.717. El `critical value` determina la similitud entre dos muestras. Si el valor de `t-statistics` es mayor que el `critical value`, significa que dos muestras son diferentes,

Ahora, calculamos el `t-statistic`:

$$t - statistics = \frac{(X_1 - X_2) - (\mu_1 - \mu_2)}{SE} - dt$$

Por último, calculamos el $p - value$:

$$p - value = P(|T_{calculada}| >= t_{df}, 1 - \alpha/2)$$

Estos son los resultados obtenidos:

	Java	Python
Media de valores de accuracy	`0.834`	`0.857`
Error de desviación típica (SE)	`0.022`	`0.034`
Degrees of Freedom	18	
Critical Value	`1.734`	
`t-statitics`	`-0.39`	
`p-value`	`0.69`	

Tabla 3.7: Resultados del `t-test`

Con los resultados obtenidos en la tabla 3.7 podemos observar que el valor $t - statistics$ es mayor que la de `critical value`. Si `t-statistics` fuera menor, quería refutada la hipótesis nula (H_o) que habíamos asumido anteriormente y ambas muestras serían estadísticamente diferentes [36]. Sin embargo, al ser mayor, sí hay evidencia de que tanto los resultados del proyecto en `Java` como los de `Python` son estadísticamente similares. Para reforzar esta evidencia, el valor de $p - value$ es mayor que el valor de significancia α. En caso de que fuese menor, la hipótesis nula (H_o) también quedaría refutada.

3.3 KNN con Lower Bounds Enhanced y DTW: Análisis y validación

En esta sección trataremos de analizar la técnica *K Nearest Neighbours* o KNN con una medida distancia elástica mejorada llamada *Lower Bound Enhanced* o `LbEnhanced`, sugerida por los investigadores de la Universidad de Monash (Australia) [30]. Al igual que hicimos con Proximity Forest, hemos hecho una réplica en Python del proyecto realizado en Java por los investigadores que publicaron esta nueva medida de distancia, extrayendo solamente la parte que nos interesaba para este trabajo de fin de grado. El código original desarrollado por los investigadores se encuentra disponible en `GitHub` a través de este enlace: `https://github.com/ChangWeiTan/LbEnhanced`. Luego, hemos combinado la parte de desarrollo propio y lo hemos combiando con un proyecto de KNN con DTW. Dicho proyecto se encuentra en un repositorio público en `GitHub` [7] y lo que se ha hecho ha sido modificar el algoritmo sustituyendo la medida DTW por la medida que `LbEnhanced` propuesta con la que queremos realizar el análisis. Este repositorio también no será útil para hacer la comparación de los benchmarks evaluados con la técnica KNN con DTW. Recordemos que KNN busca clasificar un conjunto de datos aplicando medidas de similitud a partir de K vecinos próximos que ya fueron clasificados. Procederemos en este caso a analizar los hiperparámetros tanto del algoritmo de clasificación como de la medida de similitud, así como evaluar la precisión y el tiempo de ejecución que conllevan.

3.3.1 Hiperparámetros y análisis de rendimiento

Según los investigadores que publican esta nueva medida de distancia, con un tamaño de ventana W igual a 1 y estableciendo un `speed-tightness` V entre 1 y 2, se obtiene un resultado mejor que otras medidas de distancia con acotamiento [30] aunque, según apuntan los investigadores, la mayoría de los experimentos presentan un buen resultado de predicción si se establece el grado de estrechamiento en 5 y un tamaño de ventana cuyo valor oscila entre 6 y 10, dependiendo de la longitud de serie [30]. Sin embargo, con los datasets utilizados nos hemos dado cuenta de que no es así. Utilizando esos valores, los resultados que se presentan no son buenos. Por tanto, es necesario hacer un análisis de los hiperparámetros que ofrezcan resultados óptimos en nuestro benchmark.

La medida de similitud LbEnhanced consta de varios hiperparámetros:

- **Warping Window** o Ventana de deformación. Como se comentó en la sección de DTW, este hiperparámetro establece un umbral para no recorrer toda la matriz sino hasta un cierto punto de esta (Figura 3.8).

- **Speed-Tightess** (V). Este hiperparámetro se encarga de limitar el número de posiciones que se puede avanzar, tanto a la derecha como a la izquierda, dada una posición actual. Es decir, si tenemos un grado de estrechamiento V = 4, tal y como se muestra en la Figura 3.8, una posición actual solo podrá recorrer como máximo 4 posiciones a la derecha y a la izquierda dentro de los límites de la matriz. Un valor de V pequeño requiere menos cálculos, pero genera cotas muy frágiles y una capacidad de predicción baja. Si el valor de V es grande, aumentará la precisión en la predicción, pero conllevará hacer más cálculos y, por tanto, más tiempo.

[7] La fuente del proyecto se encuentra a través del siguiente enlace `https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping`

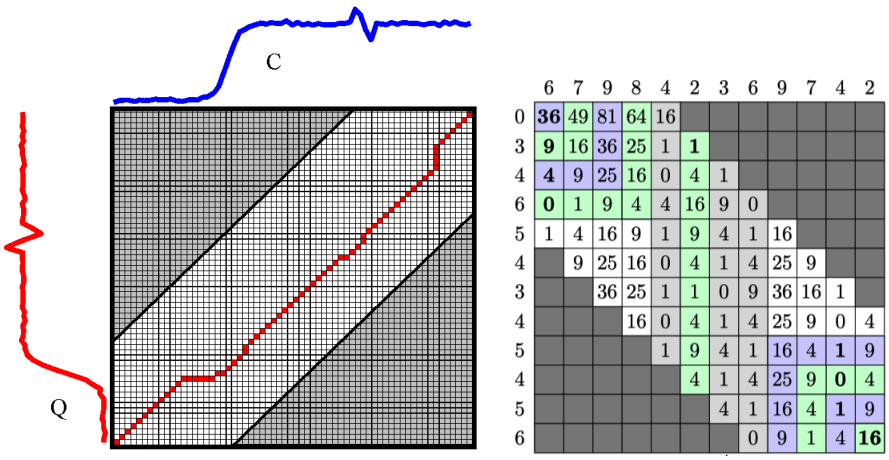
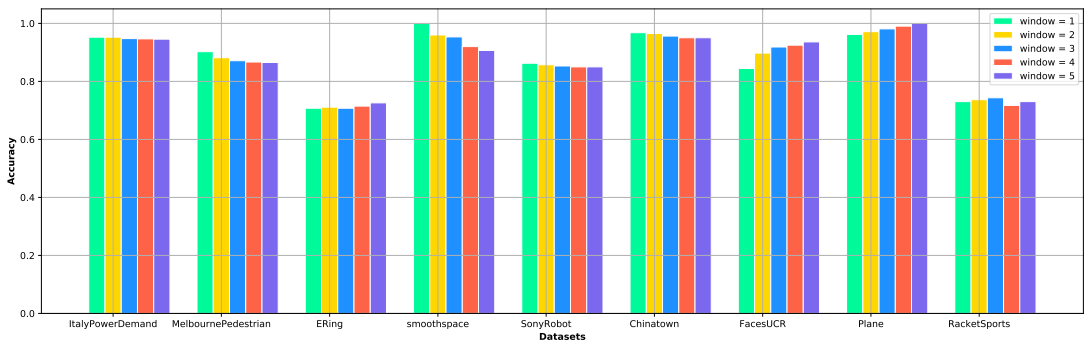


Figura 3.8: A la izquierda , Dadas dos secuencias C y Q, para alinear ambas secuencias se construye una matriz de distorsión para poder encontrar el camino óptimo. Como se puede observar, los puntos de las zonas que están en color gris oscuro son descartados ya que se ha aplicado una ventana w se establece como límite en el recorrido de la matriz (Imagen extraída de la siguiente fuente `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=7847D05EC0213AC8A9E67F990796B451?doi=10.1.1.23.6686&rep=rep1&type=pdf`. A la derecha se muestra el speed-tightness V. Dado un valor de estrechamiento V = 4, dada una determinada posición, solo se podrá recorrer hasta 4 celdas, tanto por la derecha como por la izquierda [30].

Análisis de ventana de deformación

Tamaño de ventana de DTW KNN con DTW solo funciona si se establece una ventana. Por eso hemos establecido el rango de valores entre 1 y 5.



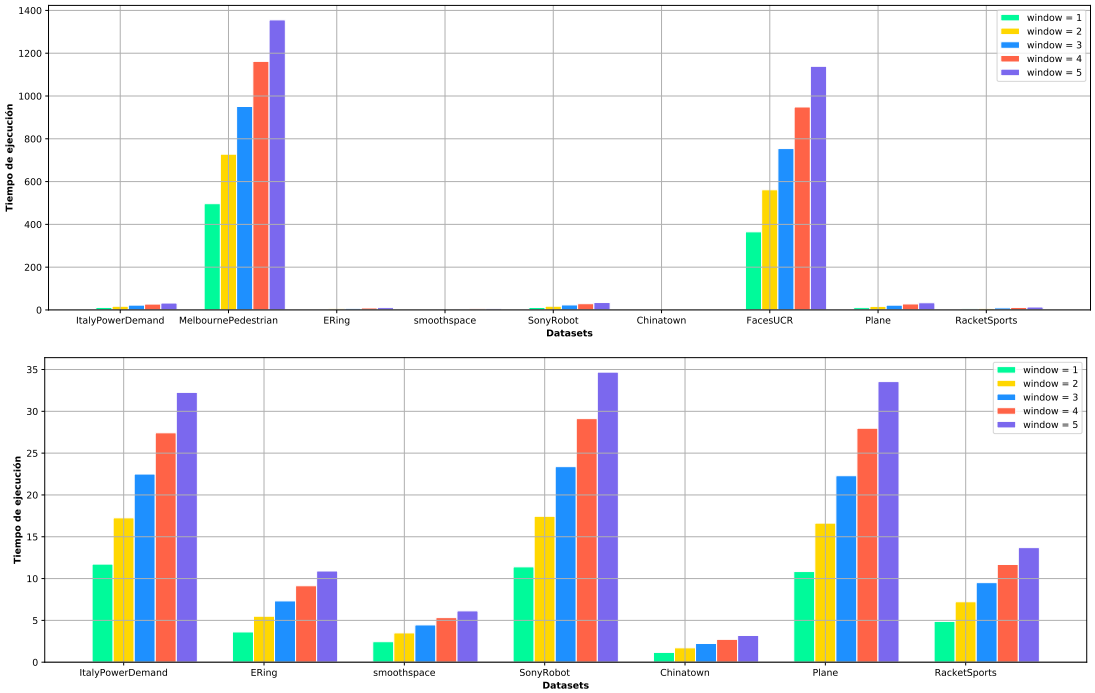
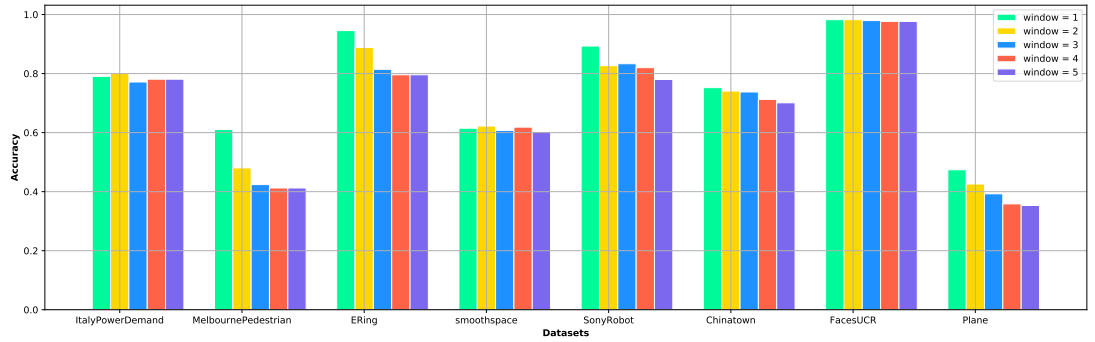


Figura 3.9: Gráfico, de elaboración propia, de comparación de tamaños de ventana para DTW con KNN.

Como se muestra en la Figura 3.9, a simple vista el tamaño no tiene una gran influencia en la precisión, aunque si la tiene sobre el tiempo de ejecución. Por tanto, decidimos elegir un tamaño de ventana $\mathbf{W = 1}$ para DTW.

Tamaño de ventana de LbEnhanced Procedemos a establecer tamaños de ventana pequeños. El resultado es que la precisión mejora cuando el tamaño de ventana de este no es muy grande. Independientemente del `speed-tightness` y el número de vecinos, el algoritmo de clasificación ofrece mejores resultados cuando el valor de Warping Window oscile entre 0.25 y 1. Se ha hecho la prueba en varios datasets y estos han sido los resultados.



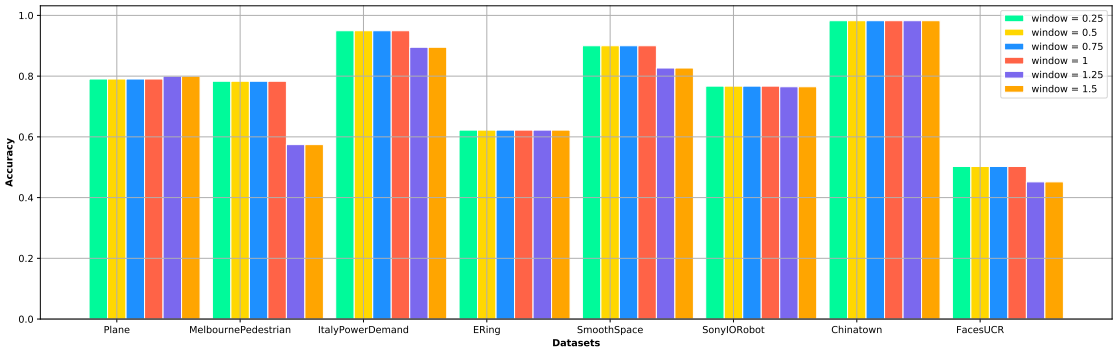


Figura 3.10: Comparación de tamaños de ventana. En la imagen superior, se muestra el valor de la `accuracy` en función de tamaños de ventana entre 1 y 5. En la imagen inferior, se muestra el valor de la `accuracy` en función de los tamaños de venta entre 0.25 y 1.5 (Imagen de elaboración propia durante el análisis de los hiperparámetros).

La Figura 3.10 muestra los valores de `accuracy` en función del tamaño de ventana. En la imagen de arriba, el valor de `accuracy` en función del tamaño de ventana de 1 a 5. Como se puede apreciar, los mejores resultados se dan cuando el tamaño de ventana no pasa de 1. En la imagen de abajo, se enfoca el tamaño de ventana entre 0.25 y 1.5. Conforme el tamaño de ventana pasa de 1, la precisión se decrementa. Sin embargo, esta se mantiene estable en el intervalo [0.25, 1].
En cuanto al tiempo de ejecución, según podemos observar en la Figura **??**, un tamaño de ventana igual a 1 provoca un tiempo de ejecución sensiblemente menor, por lo que no existe mucha diferencia entre él y los valores más pequeños. Donde sí comienza a existir una diferencia notoria es cuando el tamaño de ventana es superior a 1.

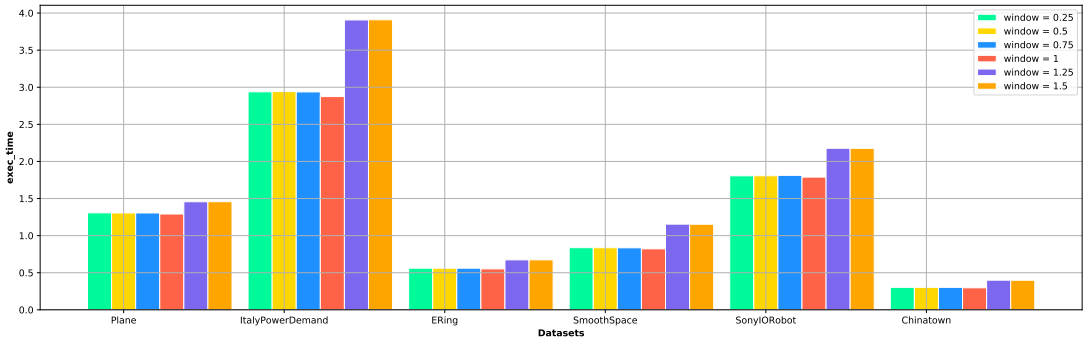


Figura 3.11: Gráfico, elaborado durante el análisis, en donde se muestra el tiempo de ejecución, en segundos, en función de tamaños de ventana entre 0.25 y 1.5 para cada dataset.

Análisis del Speed-Tightness

Según se muestra en el gráfico de la Figura 3.13, podemos destacar que la precisión, a la hora de predecir, suele ser mayor conforme crece el tamaño de `Speed-Tightness`, siempre que su tamaño no sea mayor a la mitad de la longitud de una serie. Sin embargo, también tiende a aumentar el tiempo de ejecución.

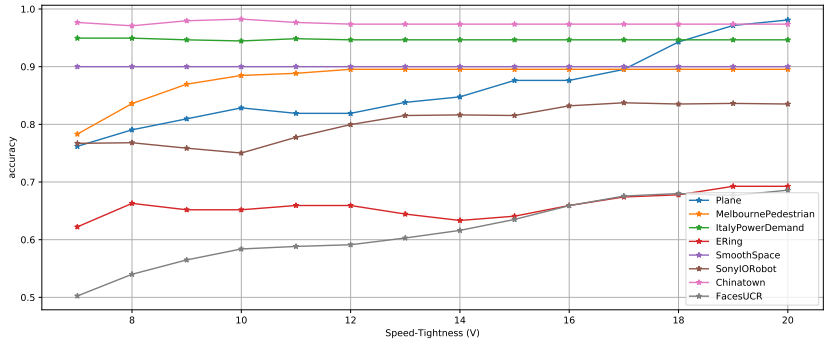


Figura 3.12: Gráfico, de elaboración propia, que muestra la influencia del *speed-tightness* V, con valores entre 5 y 20, en la precisión para cada dataset. Utilizamos un tamaño de ventana W = 1 y un número de vecinos próximos = 1

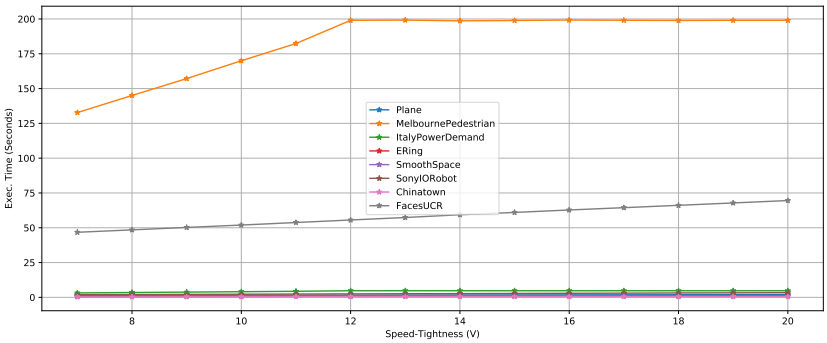


Figura 3.13: Gráfico, de elaboración propia, que muestra la influencia del *speed-tightness* V, con valores entre 5 y 20, en el tiempo de ejecución. Utilizamos un tamaño de ventana W = 1 y un número de vecinos próximos = 1

Número de vecinos próximos

Ahora vamos a analizar el hiperparámetro de la técnica KNN, que es el número de vecinos próximos de quien basarse a la hora de clasificar. KNN no solo se utiliza para clasificar sino que también se usa para clustering, es decir, aprendizaje no supervisado en la que se buscan puntos o patrones comunes entre distintos conjuntos de datos no clasificados.

Vecinos próximos en LbEnhanced En este caso se han ejecutado varios datasets con distinto número de vecinos próximos, específicamente de 1 a 10 vecinos que son los valores que normalmente se suelen utilizar para esta técnica. Los resultados han sido estos:

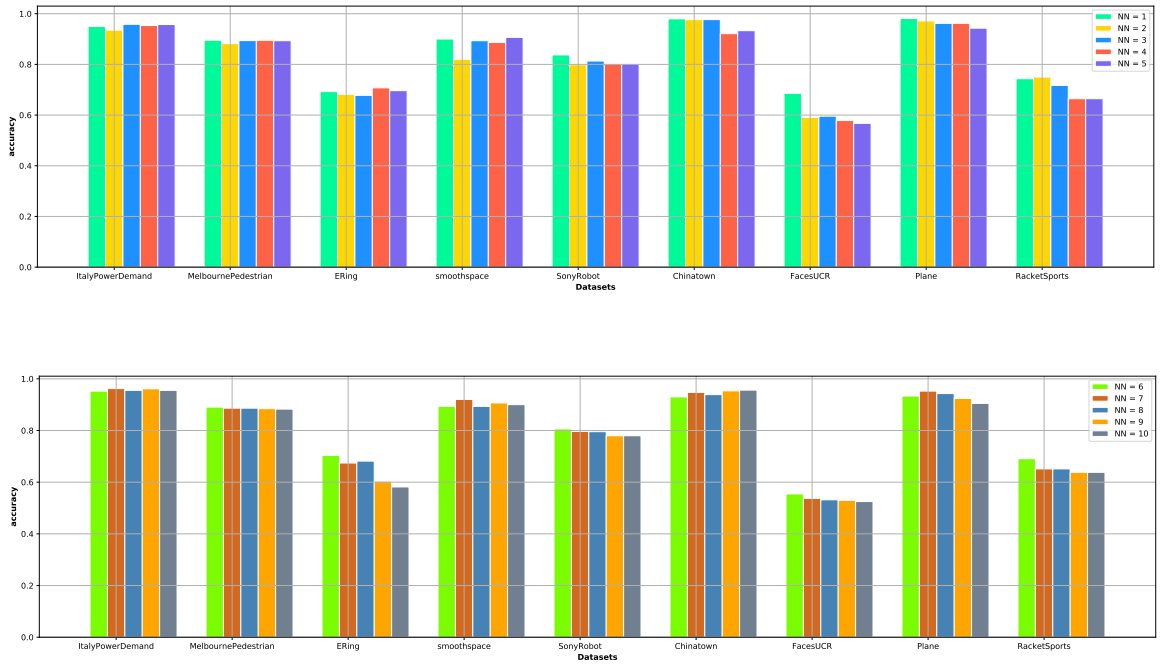


Figura 3.14: Imágenes, de elaboración propia, que muestran la influencia del número de vecinos próximos en la precisión de las predicciones. En la imagen superior, se muestra la influencia del número de vecinos entre 1 y 5. En la imagen inferior, se muestra el número de vecinos entre 6 y 10.

Como podemos observar en el histograma 3.14, incrementar el número de no solo no altera significativamente el resultado de la predicción, sino que incluso lo decrementa.

Vecinos próximos en DTW El gráfico que nos muestra la Figura 3.15, al igual que con LbEnhanced, el número de vecinos no altera el demasiado la precisión.

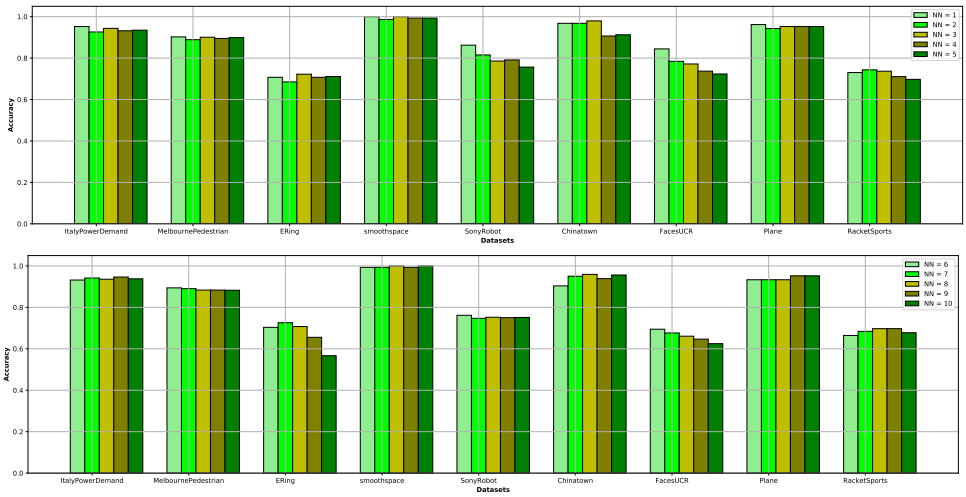


Figura 3.15: Imágenes, de elaboración propia, que muestran la influencia del número de vecinos próximos en la precisión de las predicciones en KNN con DTW. En la imagen superior, se muestra la influencia del número de vecinos entre 1 y 5. En la imagen inferior, se muestra el número de vecinos entre 6 y 10.

Conclusiones y modelo final

Los autores que publicaron esta medida de distancia afirman un valor V = 5 y un tamaño de ventana inferior a 10 ofrece unos resultados mejores en términos de distancia entre secuencias [30], pero nuestros análisis muestran resultados distintos: El tamaño de ventana que muestra un mejor resultado, según se muestra en la Figura **??** y Figura 3.10, suele ser aquel que se encuentra entre 0.25 y 1. Los investigadores aún no han encontrado el tamaño óptimo de V [30], pero según podemos ver en la Figura 3.8, un tamaño de V = 20 ofrece mejores resultados en la mayoría de los datasets y en algunos casos no influye en el tiempo de ejecución. En cuanto al número de vecinos próximos, el tamaño de vecinos próximos más idóneo es de 1 basándonos en la Figura **??**.
Por tanto, nuestro modelo final que utilizaremos para realizar una comparación de benchmarks es de un tamaño de ventana = 1, una $V = 20$, 1 vecino próximo.

3.3.2 Evaluación con K-Fold Crossvalidation

Para comprobar que nuestra implementación es correcta y robusta, vamos a realizar una cross-validación de 10 pliegues con 10 repeticiones, igual que en Proximity Forest. Para cada iteración habrá un pligue distinto que usaremos para validar y los otros 9 para entrenar.

Tabla de resultados de cross-validación por dataset

Dataset	1	2	3	4	5	6	7	8	9	10	Media
DdLoopGame	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.95
RacketSports	0.66	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9583
EDevDetction	0.919	0.984	1.0	1.0	0.984	1.0	1.0	1.0	1.0	0.9697	0.9674
ItPowerDem	0.857	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.972
Wafer	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9776
ERing	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9813
Plane	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9826
Chinatown	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9848
MelbPedestrian	0.5042	0.966	0.907	0.99	0.991	0.983	0.991	0.873	0.823	1.0	0.9757
FacesUCR	0.45	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9726
FRegTrain	0.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9697

Tabla 3.8: Tabla de resultados de la precisión en cada iteración y la media de valores de precisión para cada dataset en una `crossvalidation`

Dataset	Accuracy media Cross-validación	Accuracy Testing Dataset
DdLoopGame	0.95	0.855
RacketSports	0.9583	0.743
EDevDetction	0.9674	0.87
ItPowerDem	0.972	0.946
Wafer	0.977	0.968
ERing	0.9813	0.696
Plane	0.982	0.981
Chinatown	0.984	0.973
MelbPedestrian	0.9757	0.895
FacesUCR	0.9726	0.69
FRegTrain	0.9697	0.756

Tabla 3.9: Comparación entre la media de precisión sacada de la cross-validación y la precisión obtenida del dataset de validación.

Como comentamos en la sección de validación Proximity Forest, es posible que hayan notables diferencias entre la precisión media de la crossvalidación y la precisión sacada del conjunto de validación debido a que el tamaño los datasets utilizados son pequeños y, al dividirlo en 10 partes, cada pliegue utilizado para la validación puede ser demasiado pequeño y fácil de predecir. Sin embargo, el dataset dedicado para el testing puede ser mucho más grande y, por tanto, puede contener muchas más series. Sin embargo, la Tabla 3.8 muestra resultados similares sin mucha diferencia entre los resultados de cada iteración, por lo que se podría afirmar que nuestro benchmark es fiable y funciona correctamente.

3.4 Comparación con Nearest Neighbours - DTW

Tras analizar las dos técnicas de clasificación en secciones anteriores, así como evaluar los hiperparámetros idóneos para cada una de ellas, vamos a proceder a realizar una comparativa entre ambos benchmarks con la técnica de clasificación Nearest Neighbour con la medida de distancia DTW. Para realizar la comparativa se va hacer uso del mismo proyecto de KNN que se utilizó para integrar la medida elástica `LbEnhanced`.

Tras analizar los dos técnicas de clasificación por separado, así como evaluar los hiperparámetros idóneos para cada una de ellas, vamos a proceder a realizar una comparativa entre ambos benchmarks con la técnica de clasificación Nearest Neighbour con la medida de distancia DTW. Para realizar la comparativa se va a utilizar mismo proyecto de KNN que se utilizó para integrar la medida elástica `LbEnhanced` [8] . Hasta la fecha, la técnica KNN usando DTW presentaba muy buenos resultados tanto en tiempo como en precisión. Esta técnica no aplica una medida de distancia con unos valores por defecto, sino que estos son aprendidos durante el entrenamiento. Por tanto, procedemos a comparar los tres benchmarks utilizando los datasets de partida, y cada benchark tendrá los valores que mejores resultados ofrecen y que se han sacado durante el análisis.

[8]Se puede consultar la fuente del proyecto de KNN con DTW en Python a través de este enlace: `https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping`

	Proximity Forest	KNN con LbEnhanced	KNN con DTW
Hiper parámetros	Num. de árboles = 20 Num. de candidatos por nodo = 2 Num. de repeticiones = 1	Warping Window = 1 Speed-Tightness = 20 NN = 1	Warping Window = 1 NN = 1

Probamos los tres benchmarks en cada uno de los datasets utilizados anteriormente y estos son los resultados:

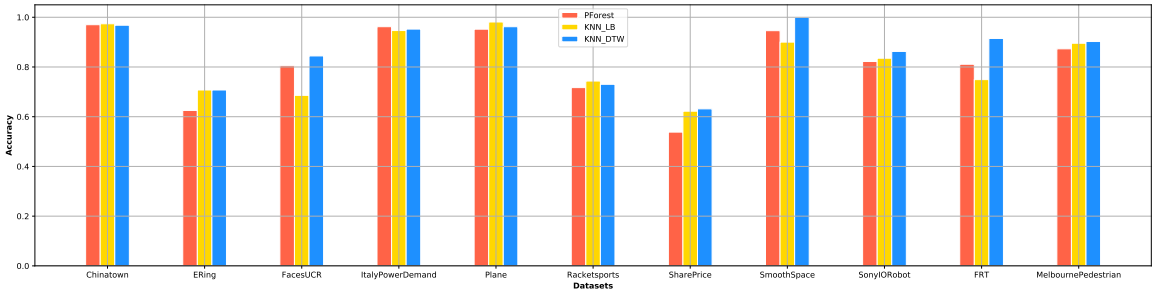


Figura 3.16: Resultado, extraído del análisis, de valores de `accuracy` de cada benchmark para cada dataset.

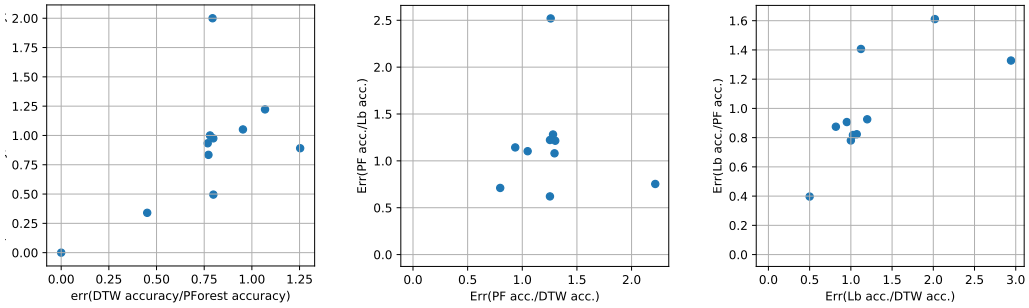


Figura 3.17: Ratio de tasas de error de los tres modelos. A de la izquierda muestra la tasa de error de DTW frente a los demás, el del centro muestra la de PForests y el de la derecha la de LBEnhanced. Cada punto representa las tasas de error de predicción de un dataset. Si un punto se encuentra dentro del rango [0, 1] en ambas dimensiones significa que favorece al benchmark que del que se muestra la tasa de error frente a los demás

Como se puede ver en la Figura 3.16, en términos de precisión, KNN con DTW sigue presentando unos resultados muy competitivos frente a los demás benchmarks. Si nos fijamos solamente en los dos benchmarks que presentamos, LbEnhanced presenta unos resultados ligeramente mejores que Proximity Forest, y en algunos casos las diferencias entre estos dos benchmarks suele ser muy significativa como ocurre en dataset `SharePrice`. Si echamos un vistazo al ratio de error de predicción (Figura 3.17), observamos en el gráfico de la izquierda que hay más puntos dentro de la franja entre 0 y 1, lo que significa que hay más tasas predicción favorables a KNN con DTW. Sin embargo, la predicción no es el único factor a la hora de elegir qué benchmark utilizar.

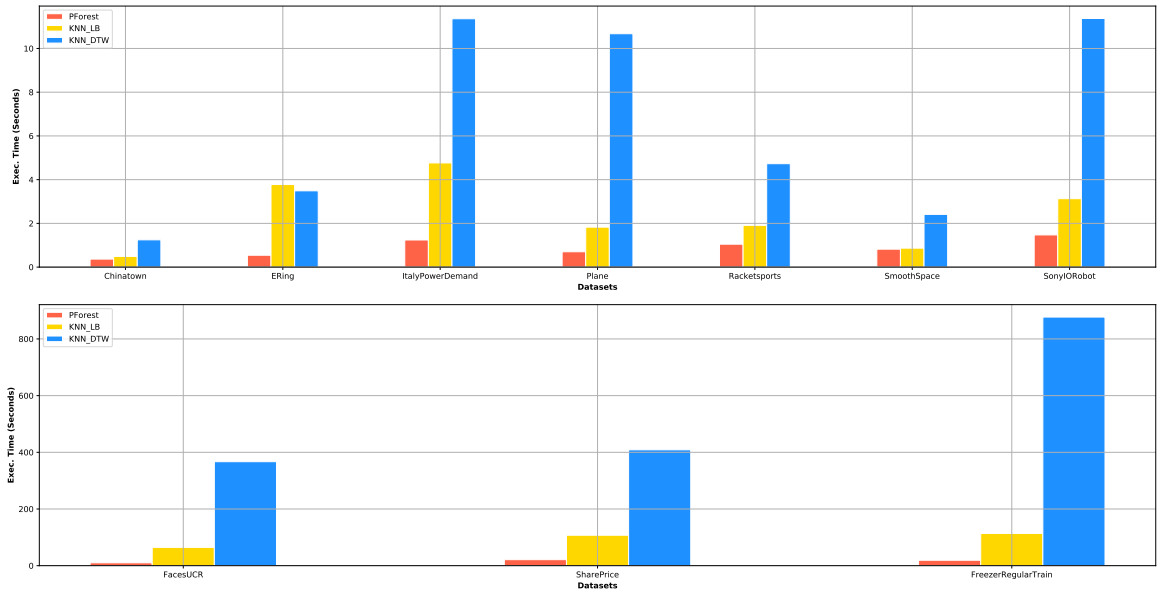


Figura 3.18: Resultado de valores de tiempo de ejecución de cada benchmark para cada dataset

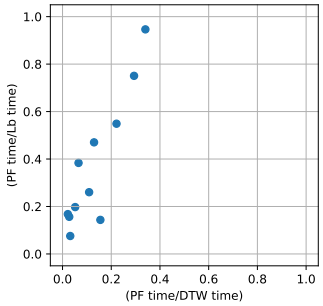


Figura 3.19: Gráfico, extraído de los resultados de los análisis, que muestra el ratio del tiempo de ejecución del modelo de Proximity Forest entre el modelo DTW en el eje X y el model *LbEnhanced* en el eje Y.

La cuestión que habría que hacerse ahora es ¿Cual es mejor en términos de tiempo?. La figura 3.18 refleja a un claro ganador en términos de tiempo, Proximity Forest. Sin duda este algoritmo presenta un tiempo de ejecución mucho mejor que los otros dos benchmarks debido al bajo número de árboles y nodos candidatos que se establecen. Por otro lado, el modelo de KNN con DTW refleja un tiempo de predicción muy descompensado con respecto a los demás predictores. El gráfico de la Figura 3.19 representa el ratio de tiempo de ejecución entre Proximity Forest y KNN DTW en el eje X, y LbEnhanced en el eje Y. El gráfico muestra los datasets como puntos y los que se encuentran dentro de la zona del intervalo horizontal y vertical [0, 1] indican que son mejores con Proximity Forest que con los otros dos predictores. Pues se puede observar que todos los puntos están dentro de la zona y que el modelo de Proximity Forest es el más favorable en términos tiempo.

Si echamos un vistazo a lo que hubiera pasado si tuvieramos un modelo de Proximity Forest de 35 árboles y 3 candidatos, el resultado, como se muestra en la Figura 3.21, sería muy diferente. De hecho, Proximity Forest muestra resultados mucho peores en términos de tiempo sin ninguna

mejora notable de la precisión. Esto nos indica que la elección de tener un modelo de 20 árboles y 2 candidatos ha sido bastante acertada y consolidada.

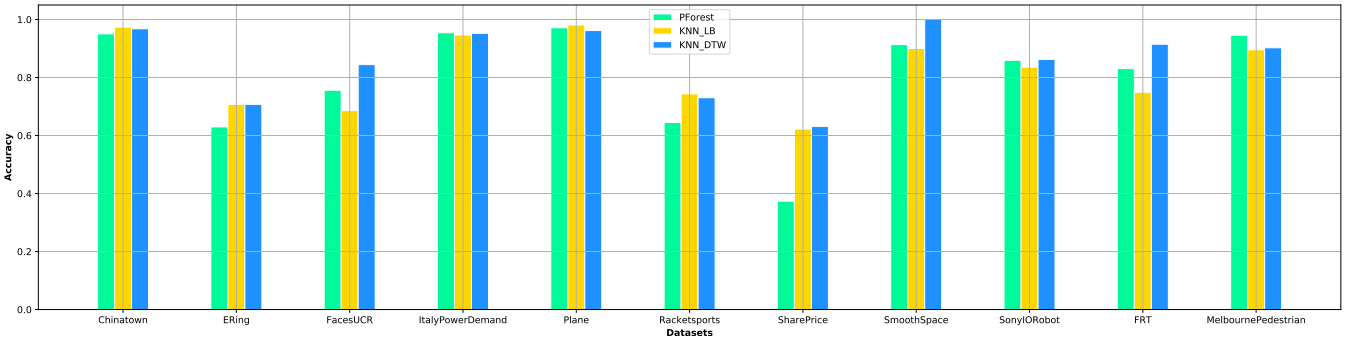


Figura 3.20: Resultado, de elaboración propia, de valores de precisión de cada benchmark para cada dataset, en el que nuestro modelo para *Proximity Forest* es de 35 árboles y 3 candidatos.

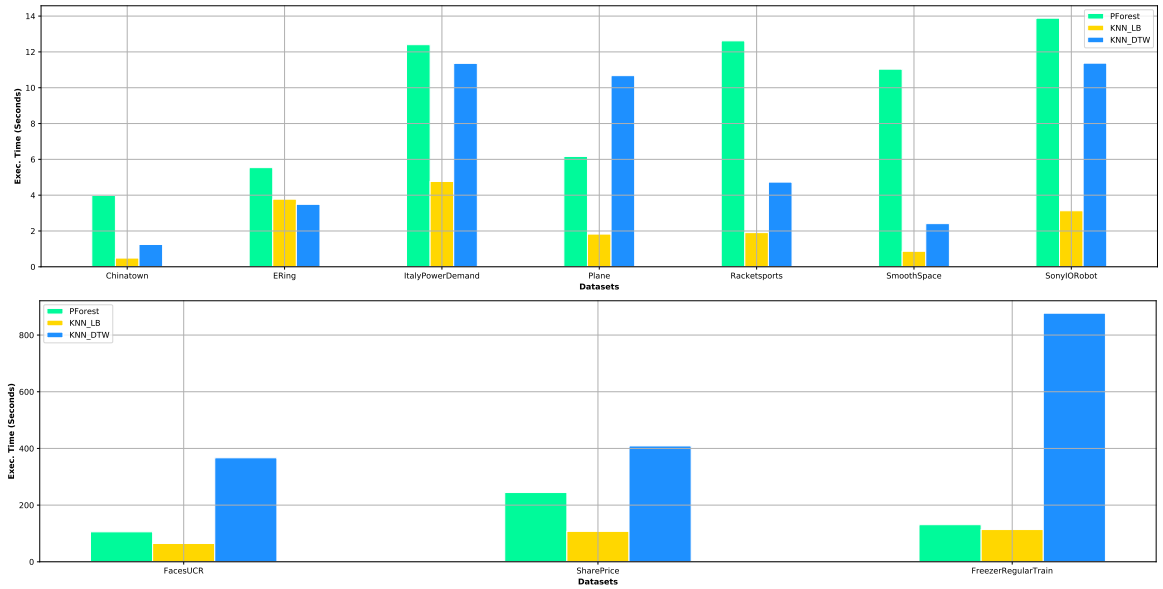


Figura 3.21: Resultado, de elaboración propia, de valores de tiempo de ejecución de cada benchmark para cada dataset, en el que nuestro modelo para *Proximity Forest* es de 35 árboles y 3 candidatos.

Discusión de resultados

Tras analizar y comparar cada uno de los benchmarks, es hora de discutir si los resultados son aceptables con respecto al estado de arte. La técnica KNN con DTW es, hasta la fecha, la técnica que ofrece una mejor relación entre la capacidad de predicción y el tiempo necesario para llevarla a cabo. Sin embargo, partiendo de nuestros resultados (Figura 4.1 y Figura 4.2), Proximity Forest ofrece un buen ajuste en un tiempo mucho menor. Por otra parte, **LbEnhanced** también ofrece resultados mejores, en términos de tiempo, pero queda bastane lejos de Proximity Forest. No obstante, cabe recalcar que KNN-DTW sigue ofreciendo los mejores resultados de precisión. La cuestión, en términos de eficiencia, sería la prioridad de uso de cada benchmark. Si lo que se busca es una buena `accuracy` y el tiempo no es un problema, entonces KNN-DTW sigue siendo la mejor opción, además de otras técnicas como COTE [23]. Sin embargo, si buscamos un modelo que pueda realizar un entrenamiento y una predicción en poco tiempo, aunque la precisión sea ligeramente menor que la ofrecida por KNN, entonces *Proximity Forest* es, según nuestro estudio, la mejor opción.

4 Conclusiones

Nuestro trabajo de fin de grado concluye con que el modelo de Proximity Forest es el que mejor ofrece una relación entre la precisión y el tiempo de ejecución. Tanto en Proximity Forest como en LbEnhanced, hemos analizado los mejores hiperparámetros y su influencia en el tiempo y la precisión. Para Proximity Forest, el modelo que ofrece un mejor ajuste fue aquel con 20 árboles y 2 candidatos, ya que ofrecía una buena precisión con un tiempo de ejecución mucho menor que con los demás modelos. Para LbEnhaced, concluimos que el mejor modelo para comparar era aquel con un `speed-tighness` V = 20, y un tamaño de ventana $W = 1$.

	Proximity Forest	KNN con LbEnhanced	KNN con DTW
Hiper parámetros	Núm. árboles = 20 Núm. candidatos = 2	Warping Window = 1 Speed-Tightness = 20 Núm. vecinos = 1	Warping Window = 1 Núm. vecinos = 1

Tabla 4.1: Valores de los hiperparámetros seleccionados para cada modelo a comparar

Tras elegir los mejores hiperparámetros, cada modelo fue sometido a una cross-validación para comprobar que ambos algoritmos desarrollados son robustos. Luego, los modelos fueron comparados la técnica KNN con DTW, que es la más efectiva en términos de precisión y tiempo [28]. El resultado obtenido en la mayoría de datasets utilizados fue que el modelo de Proximity Forest ofrecía resultados de precisión similares a los demás benchmarks en menos tiempo.

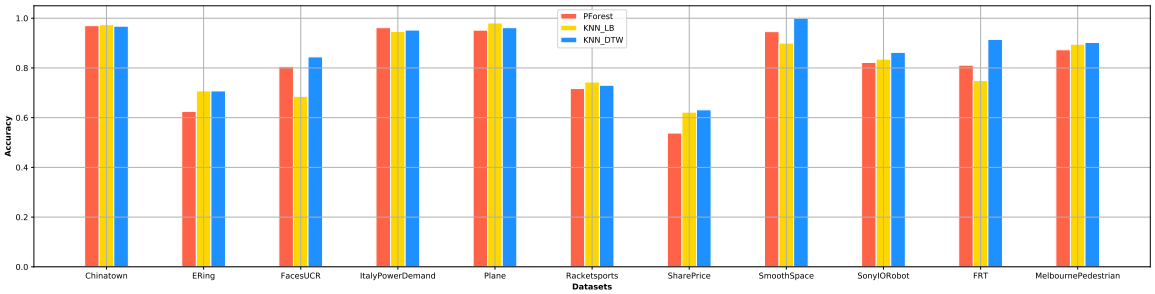
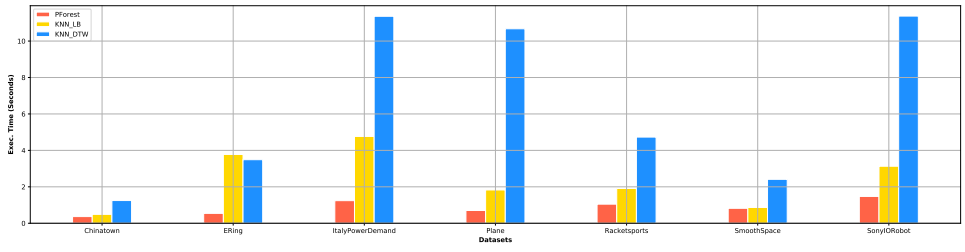


Figura 4.1: Resultado, extraído del análisis, de valores de `accuracy` de cada benchmark para cada dataset.



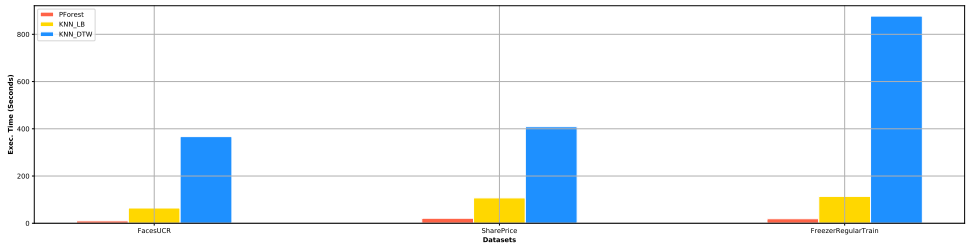


Figura 4.2: Resultado de valores de tiempo de ejecución de cada benchmark para cada dataset (Gráficos de elaboración propia).

Como podemos observar en la Figura 4.1 y 4.2, aunque es evidente que el modelo KNN con DTW ofrece mayores valores de precisión, lo cierto es que el tiempo empleado por el modelo Proximity Forest es indudablemente mejor que los otros dos benchmarks en todos los datasets utilizados. Sin embargo, como suele ocurrir en el Machine Learning, la idoneidad de cada modelo o técnica a utilizar depende del contexto en el que se quiere usar, es decir, algunos modelos con ciertos hiperparámetros serán mejores si lo que se pretende es alcanzar una precisión óptima y otros serán idóneos si lo que se busca es mejorar el tiempo de predicción aunque se reduzca un poco la precisión. La realización de este trabajo de fin de grado ha conllevado algunas dificultades durante el desarrollo. La parte de implementación ha sido la que más tiempo ha llevado y en la que se incluyen el desarrollo, búsqueda y reparación de *bugs*, testeo de casos extremos, etc. Otra fase en la que hubieron dificultades fue el tratamiento de los resultados de la ejecuciones para visualizarlos ya que la ejecución de varios modelos de la misma técnica, seguida de una comparación y validación de los mismos, conlleva mucho tiempo. En este trabajo de fin de grado, también hemos conseguido optimizar la técnica de Proximity Forest frente a la ofrecida por la herramienta `sktime`, consiguiendo unos resultados iguales, incluso mejores en algunos casos, en un tiempo muchísimo menor, tal y como mostramos en la sección 3.2.1 en referencia a esta herramienta en Python.

Name	Accuracy PF Propio	Accuracy PF sktime	T.ejec. Propio	T.ejec. sktime
Chinatown	0.9708	0.9416	0.3755	35.8863
Plane	0.9523	1.0	0.7098	899.7863
RbtSurface	0.8226	0.8709	1.4691	798.1022
RckSports	0.5921	0.3947	23.304	413.199
ERing	0.6259	0.5185	0.5412	383.9367
ItalyPower	0.9543	0.9563	1.292	162.02

Tabla 4.2: Tabla de resultados de la precisión y el tiempo de ejecución de cada dataset utilizando el modelo de Proximity Forest con 20 árboles y 2 candidatos en cada proyecto.

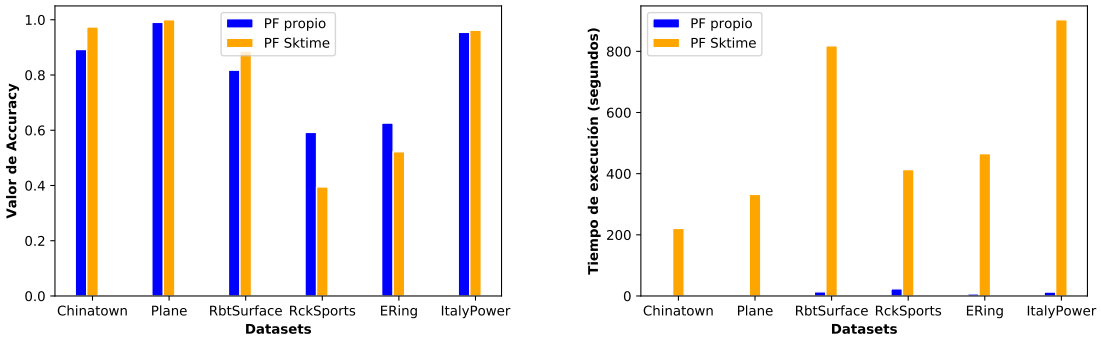


Figura 4.3: Comparativa de la precisión y el tiempo de ejecución, utilizando varios datasets, del modelo de Proximity Forest con 20 árboles y 2 candidatos. Imagen de elaboración propia.

Vías Futuras En este trabajo de fin de grado se ha realizado una comparación de técnicas basadas en la medida de distancia DTW y LbEnhanced. Sin embargo, puede que esta medida no sea la más efectiva y sea necesario seguir investigando con otras medidas de distancia elásticas que deriven de ella. Como comentamos en la secciónes 1.2.2 y 1.2.2, existen técnicas que ayudan a solventar los problemas que puede acarrear la medida de distancia DTW y sería interesante poder analizar y comparar los benchmarks tratados en este trabajo de fin de grado con estas medidas de distancia, *DDTW* y *WDTW*, en un futuro. También sería interesante analizar y comparar la técnica `FastEE` [37]. Esta nueva técnica, basada en cotas inferiores (Como LbEnhanced), fue publicada en 2019 y ofrece una alternativa a la técnica Elastic Ensembles [24] que ya mencionamos en la sección 1.3.6, aunque los autores también afirman que esta técnica es menos efectiva que Proximity Forest [38] y aún continúa en proceso de investigación.

Bibliografía

[1] Mabel González Castellanos and César Soto-Valero. *Minería de datos para series temporales*, pages 5–6. 01 2013. ISBN 978-959-250-924-5. doi: 10.13140/RG.2.1.2571.9841.

[2] A. Nielsen. *Practical Time Series Analysis: Prediction with Statistics and Machine Learning*. O'Reilly Media, 2019. ISBN 9781492041627. URL `https://books.google.es/books?id=odCwDwAAQBAJ`.

[3] John Aach and George Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics (Oxford, England)*, 17:495–508, 07 2001. doi: 10.1093/bioinformatics/17.6.495.

[4] Chotirat Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. *Mining Time Series Data*, pages 3–4. 07 2010. doi: 10.1007/978-0-387-09823-4_56.

[5] Chotirat Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. *Mining Time Series Data*, chapter Time Series Data Mining, pages 13–14. 07 2010. doi: 10.1007/978-0-387-09823-4_56.

[6] Chotirat Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. *Mining Time Series Data*, chapter Time Series Data Mining, pages 18–19. 07 2010. doi: 10.1007/978-0-387-09823-4_56.

[7] "Davide Burba". "an overview of time series forecasting models". `https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb`.

[8] Chotirat Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. *Mining Time Series Data*, chapter Time Series Data Mining, pages 5–6. 07 2010. doi: 10.1007/978-0-387-09823-4_56.

[9] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. volume 730, pages 69–84, 01 1993.

[10] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[11] Chotirat Ratanamahatana and Eamonn J. Keogh. Three myths about dynamic time warping data mining. In *SDM*, 2005.

[12] Eamonn Keogh and Michael Pazzani. Derivative dynamic time warping. *First SIAM International Conference on Data Mining*, 1:5–6, 01 2002. doi: 10.1137/1.9781611972719.1.

[13] Youngseon Jeong, Myong Jeong, and Olufemi Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44:2231–2240, 09 2011. doi: 10.1016/j.patcog.2010.09.022.

[14] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):610–612, nov 2016. doi: 10.1007/s10618-016-0483-9. URL https://doi.org/10.1007%2Fs10618-016-0483-9.

[15] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31, 11 2016. doi: 10.1007/s10618-016-0483-9.

[16] "Juan Ignacio Bagnato". "clasificar con k-nearest-neighbor ejemplo en python". https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/.

[17] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):619–622, nov 2016. doi: 10.1007/s10618-016-0483-9. URL https://doi.org/10.1007%2Fs10618-016-0483-9.

[18] Lexiang Ye and Eamonn Keogh. Time series shapelets: A novel technique that allows accurate, interpretable and fast classification. *Data Min. Knowl. Discov.*, 22:149–182, 01 2011. doi: 10.1007/s10618-010-0179-5.

[19] Cun Ji, Chao Zhao, Shijun Liu, Chenglei Yang, Pan Li, Lei Wu, and Xiangxu Meng. A fast shapelet selection algorithm for time series classification. *Computer Networks*, 148, 01 2019. doi: 10.1016/j.comnet.2018.11.031.

[20] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: A novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15:107–144, 08 2007. doi: 10.1007/s10618-007-0064-z.

[21] Jessica Lin, Rohan Khade, and Yuan Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315, 2012. doi: 10.1007/s10844-012-0196-5. URL https://doi.org/10.1007/s10844-012-0196-5.

[22] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31, 11 2016. doi: 10.1007/s10618-016-0483-9.

[23] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27:1–1, 09 2015. doi: 10.1109/TKDE.2015.2416723.

[24] Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29, 06 2014. doi: 10.1007/s10618-014-0361-2.

[25] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28 (4):851–881, 2014. doi: 10.1007/s10618-013-0322-1. URL https://doi.org/10.1007/s10618-013-0322-1.

[26] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 08 2013. doi: 10.1016/j.ins.2013.02.030.

[27] Isak Karlsson, Panagiotis Papapetrou, and Henrik Boström. Generalized random shapelet forests. *Data Mining and Knowledge Discovery*, 07 2016. doi: 10.1007/s10618-016-0473-y.

[28] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 02 2019. doi: 10.1007/s10618-019-00617-3.

[29] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, page 7, 02 2019. doi: 10.1007/s10618-019-00617-3.

[30] Chang Wei Tan, François Petitjean, and Geoffrey Webb. *Elastic bands across the path: A new framework and method to lower bound DTW*, pages 522–530. 05 2019. ISBN 978-1-61197-567-3. doi: 10.1137/1.9781611975673.59.

[31] Sang-Wook Kim, Sanghyun Park, and Wesley Chu. An index-based approach for similarity search supporting timewarping in large sequence databases. pages 607–614, 02 2001. ISBN 0-7695-1001-9. doi: 10.1109/ICDE.2001.914875.

[32] Eamonn Keogh and Chotirat Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7:358–386, 01 2005. doi: 10.1007/s10115-004-0154-9.

[33] Daniel Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, 42:2169–2180, 11 2008. doi: 10.1016/j.patcog.2008.11.030.

[34] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, pages 20–23, 02 2019. doi: 10.1007/s10618-019-00617-3.

[35] Antoine Al-Achi. The student's t-test: A brief description. 5:1–3, 02 2019.

[36] Tae Kim. T test as a parametric statistic. *Korean Journal of Anesthesiology*, 68:540, 11 2015. doi: 10.4097/kjae.2015.68.6.540.

[37] Chang Wei Tan, François Petitjean, and Geoffrey I. Webb. Fastee: Fast ensembles of elastic distances for time series classification. *Data Mining and Knowledge Discovery*, 34(1):231–272, 2020. doi: 10.1007/s10618-019-00663-x. URL `https://doi.org/10.1007/s10618-019-00663-x`.

[38] Chang Wei Tan, François Petitjean, and Geoffrey I. Webb. Fastee: Fast ensembles of elastic distances for time series classification. *Data Mining and Knowledge Discovery*, 34(1):37, 2020. doi: 10.1007/s10618-019-00663-x. URL `https://doi.org/10.1007/s10618-019-00663-x`.

[39] Pierre-François Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:306–318, 2009.

[40] Michalis Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multidimensional time-series. *VLDB J.*, 15:1–20, 07 2006. doi: 10.1007/s00778-004-0144-2.

[41] Saket Sathe and Charu C. Aggarwal. Similarity forests. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 395–403, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098046. URL https://doi.org/10.1145/3097983.3098046.

[42] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):617–618, nov 2016. doi: 10.1007/s10618-016-0483-9. URL https://doi.org/10.1007%2Fs10618-016-0483-9.

[43] Juan Rodríguez, Carlos Alonso, and José Maestro-Prieto. Support vector machines of interval-based features for time series classification. *Knowledge-Based Systems*, 18:171–178, 08 2005. doi: 10.1016/j.knosys.2004.10.007.

[44] Mustafa Baydogan and George Runger. Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30:1–34, 07 2015. doi: 10.1007/s10618-015-0425-y.

[45] Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29, 06 2014. doi: 10.1007/s10618-014-0361-2.

[46] Anthony Bagnall, Franz Király, Markus Löning, Matthew Middlehurst, and George Oastler. A tale of two toolkits, report the first: benchmarking time series classification algorithms for correctness and efficiency, 2019.

[47] Xiaoyue Wang, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *CoRR*, abs/1012.2789, 2010. URL http://arxiv.org/abs/1012.2789.

Anexo

Para consultar el código fuente de cada proyecto y los datos elaborados:

- Repositorio del código fuente de Proximity Forest de desarrollo propio: `https://github.com/moradisten/PForests.git`.
- Repositorio del código fuente de LBEnhaced desarrollado: `https://github.com/moradisten/ElasticBand.git`
- Repositorio del desarrollo de KNN con LbEnhaced: `https://github.com/moradisten/KNN-LB.git`
- Datos y gráficos elaborados durante el desarrollo de este trabajo: `https://drive.google.com/drive/folders/1S8OfpuJDHt9AXlwOHKh2ZEVB6a8-VDKz?usp=sharing`

Dataset	**classifer**	**Accuracy**	**Tiempo de ejecución (Segundos)**
Chinatown	PForest	0.9728	**0.366**
	KNN_LB	**0.97376**	0.48766
	KNN_DTW	0.96793	1.24655
ItalyPowerDemand	PForest	**0.962**	**1.240**
	KNN_LB	0.94655	4.76474
	KNN_DTW	0.95238	11.35909
ERing	PForest	0.6251	**0.542**
	KNN_LB	**0.70741**	3.77614
	KNN_DTW	**0.70741**	3.48818
FacesUCR	PForest	0.805	**10.10**
	KNN_LB	0.68585	64.55547
	KNN_DTW	**0.84439**	366.86865
FreezerRegularTrain	PForest	0.811	**19.10**
	KNN-LB	0.74912	113.7929
	KNN-DTW	**0.91474**	877.11518
MelbournePedestrian	PForest	0.873	**15.39**
	KNN_LB	0.89545	203.48557
	KNN_DTW	**0.90242**	485.7252
Plane	PForest	0.952	**0.701**
	KNN_LB	**0.98095**	1.8274
	KNN_DTW	0.9619	10.67351
RacketSports	PForest	0.717	**1.048**
	KNN_LB	**0.74342**	1.90945
	KNN_DTW	0.73026	4.72935
SharePriceIncrease	PForest	0.538	**21.202**
	KNN_LB	0.62215	107.34919
	KNN_DTW	**0.63147**	408.75386
SmoothSubspace	PForest	0.946	**0.82**
	KNN_LB	0.9	0.8665
	KNN_DTW	**1.0**	2.41062
SonyAIBORobotSurface	PForest	0.822	**1.4716**
	KNN_LB	0.83526	3.1293
	KNN_DTW	**0.86254**	11.37098

Tabla 4.1: Tabla de resultados de la comparación entre los Benchmarks