Librería para la predicción usando el k-NN: paralelización y visualización de resultados

Trabajo de fin de grado del Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Daniel Francisco Bastarrica Lacalle
Javier Berdecio Trigueros
Director: Javier Arroyo Gallardo
Codirector: Albert Meco Alías

2017/2018

Índice

Resumen

Desde hace unos años, cada vez se oye hablar más y más de Redes Neuronales, y toda clase de métodos complejos de clasificación y predicción dando la apariencia de que se restringen a estos exclusivamente.

Sin embargo, existen otros muchos métodos de predicción y clasificación que sin ser tan complejos como los anteriores son igualmente válidos y puede dar resultados igual de buenos.

Este trabajo tiene por objetivo utilizar un método de los del segundo grupo. Concretamente utilizaremos el método de los *k vecinos mas cercanos*, el cuál será explicado en detalle más adelante. El principal objetivo será arrojar luz sobre este método, dada su naturaleza de tipo "caja negra", para poder entender mejor sus resultados.

El trabajo tendrá dos partes:

- Desarrollar una serie de funciones para implementar el algoritmo y optimizar sus parámetros
- Desarrollar una interfaz para visualizar sus resultados e interpretar los parámetros más facilmente

En esta memoria se describe todo el proceso que se ha llevado a cabo, las distintas iteraciones de las funciones, los paquetes utilizados, las conclusiones del desarrollo y una serie de mejoras que no se han planteado pero no han podido llevarse a cabo en el tiempo que dura el curso.

Palabras clave

k-NN , series temporales, aprendizaje vago, pronóstico estadístico, aprendizaje basado en instancias

Abstract

In the last years, we have heard more and more about Neural Nets, and all kinds of complex methods for classification and prediction. This gives the appearance that these are the only available options.

Nevertheless, there exists a variety of other methods for prediction and classification without being of the complexity of the before mentioned and they are just as valid and give as good results.

This work has the objective of utilizing a method within the second group. Our focus is on the k-nearest neighbors method. This will be explained in detail later. The main objective will be to shine a light on this method, given its "black-box" nature, in order to better understand the results.

This work will have two parts:

- Developing a series of functions to implement the algorithm and to optimize the parameters.
- Developing a graphical user interface for viewing its results and easily interpreting the parameters.

In this text we describe the whole process that was carried out, the packages utilized, the conclusions, and a series of improvements that could be done in the future.

Key words

k-NN , time series, lazy learning, statistical forecasting, instance-based learning

1 Introducción

En el mundo actual, predecir es una tarea que se realiza en múltiples ámbitos, ya sea para tomar decisiones políticas relacionadas con el aumento de las emisiones de carbono dada la tendencia de crecimiento actual, determinar la estrategia de negocio de una empresa en base a cómo se comportará el mercado, o para la detección de posibles desastres naturales a partir de un modelo que capture fielmente el movimiento de las placas tectónicas.

Para poder realizar las tareas previamente descritas se utilizan distintos métodos que se pueden englobar en dos grupos [1]:

- Los métodos de predicción cualitativos, que se aplican cuando no hay datos disponibles o sí los hay pero no son relevantes.

- Los métodos de predicción cuantitativos, que se utilizan cuando hay datos númericos sobre el pasado con patrones que podemos asumir que se repetirán en el futuro. La mayoría de estos métodos se aplican a problemas con datos de corte transversal, es decir, que han sido tomados en un momento concreto; o con datos capturados secuencialmente en el tiempo, lo que se conoce como series temporales.

Este trabajo está enfocado a una técnica del campo de la predicción de series temporales. Por lo tanto es necesario presentar el concepto de serie temporal. En términos generales, una serie temporal es un fenómeno observado secuencialmente a través del tiempo cuya representación se compone de dos partes:

- Los instantes temporales que abarca, y que para una serie de tamaño n se expresan de esta manera: $\{t_1, t_2, \ldots, t_{n-1}, t_n\}$.

- Los valores asociados a los instantes temporales: $\{y_1, y_2, \ldots, y_{n-1}, y_n\}$ en el caso de ser una serie temporal ya observada, o $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$ si se trata de una serie temporal predicha.

También es necesario exponer los patrones básicos que se pueden identificar en una serie temporal. Estos son:

- Tendencia: esto sucede si los datos de la serie temporal decrementan o incrementan a largo plazo [2].

- Estacionalidad: se tiene este patrón si la serie temporal se ve influenciada por un factor estacional (mes, año, semestre, etc) [2].

Sobre los métodos de predicción aplicados al campo de las series temporales, se puede hablar de dos tipos: los métodos estadísticos y los métodos basados en inteligencia artificial. Dentro del primer grupo se encuentran los modelos ARIMA, que pretenden describir las autocorrelaciones de los datos; y los modelos de alisado exponencial, que se basan en una descripción de la tendencia y la estacionalidad de la serie temporal [3]. Estas técnicas basadas en la estadística se caracterizan por tener parámetros que son interpretables.

En cuanto a las técnicas basadas en la inteligencia artificial, la interpretabilidad de los parámetros no es sencilla. En este grupo, se tiene a las redes neuronales; por ejemplo, al perceptrón multicapa. Al ser éste un aproximador universal de funciones, es una buena herramienta para predecir; sin embargo, esto provoca que sus parámetros sean de difícil interpretación. Además de las redes neuronales, existen otras técnicas que se pueden incluir en este grupo. Una de ellas es el método del k-NN, técnica sobre la cual trata este trabajo.

1.1 Los k vecinos más cercanos

El algoritmo de los k *vecinos más cercanos* (k-NN por sus siglas en inglés), es un método de clasificación y regresión sencillo e intuitivo que por su versatilidad también se puede utilizar para la predicción de series temporales. En este campo este método consiste en:

1. Hallar los k periodos de tamaño d más cercanos al periodo más reciente.

2. Obtener la media de los valores siguientes a estos periodos. Éste será el valor predecido.

En el presente trabajo, denominamos *elemento* a cada periodo de tamaño d. Representamos un *elemento* por su tamaño y los instantes que lo componen de la siguiente manera: $\{t_{i-d+1}, t_{i-d+2}, \ldots, t_{i-1}, t_i\}$.

A modo de exponer la dinámica de funcionamiento del k-NN se expone el Ejemplo 1.1.

Ejemplo 1.1. Se quiere predecir la cantidad de ventas de helados de un establecimento. Para ello se tiene un registro de ventas de los últimos nueve días tal y como se observa en la Tabla 1. Gráficamente, la serie temporal queda representada con la Figura 1. Se aplicará el k-NN con $k = 2$ y $d = 2$ para predecir el valor correspondiente al instante t_{10}.

Día	S	D	L	M	X	J	V	S	D
t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
x_i	32	34	17	15	34	30	20	34	36

Tabla 1: Serie temporal del Ejemplo 1.1

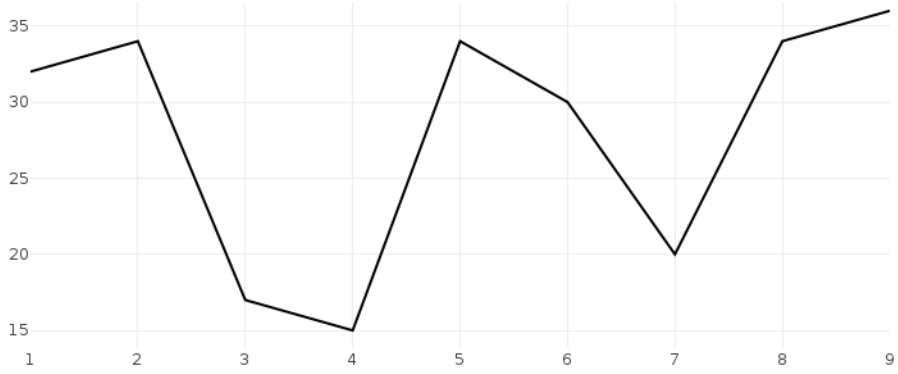

Figura 1: Gráfica del Ejemplo 1.1

Aplicando el k-NN obtenemos que el *elemento* más reciente $\{t_8, t_9\}$ tiene como *elementos* más cercanos a $\{t_1, t_2\}$ y $\{t_5, t_6\}$. Los instantes siguientes a estos *elementos* son $\{t_3\}$ y $\{t_7\}$. Por lo que el valor predicho se calcula de la siguiente manera:

$$\frac{x_3 + x_7}{2} = \frac{17 + 20}{2} = 18.5$$

Como observamos en el Ejemplo 1.1, los valores de k y d se han elegido arbitrariamente. En realidad, para elegir estos valores es necesario entrenar el k-NN para la serie temporal a predecir. Normalmente, la serie se divide en dos partes: una primera, para entrenamiento;

y otra segunda, para validación. Para la fase de entrenamiento es fundamental probar un amplio rango de valores de k y d de manera que la combinación óptima de k y d que se obtenga sea fiable. Este proceso puede llegar a ser computacionalmente muy costoso si la serie temporal es de gran tamaño.

Un aspecto importante a tener en cuenta sobre el k-NN es la interpretabilidad de sus parámetros. El k-NN es un método de predición no paramétrico, esto quiere decir, que a priori no especifica un modelo. Es necesario aclarar que esto no signfica que no tenga parámetros, sino más bien que no pertenecen a un modelo estadístico subyacente y que por tanto su interpretación es en principio más compleja. Este trabajo pretende también facilitar la misma.

1.2 Antecedentes

El método de *k-nearest neighbors*, usualmente llamado k-NN, aparece por primera vez en un reporte no publicado de la US Air Force School of Aviation Medicine en 1951. En dicho documento Fix and Hogdes presentaron este método no paramétrico de reconocimiento de patrones [4]. Ya en 1967 se descubrieron algunas propiedades formales inherentes a este método (Cover & Hart, 1967), este evento fue un importante precursor de las futuras investigaciones.

En general, esta técnica ha sido utilizada en problemas de clasificación o regresión, sin embargo su versatilidad le permite también ser empleada en tareas de estimación de densidades y predicción de series temporales [5]. En el ámbito de las series temporales es posible encontrar trabajos realizados durante los últimos años. En concreto, se procede a nombrar dos de ellos: Forecasting histogram time series with k-nearest neighbours methods (Javier Arroyo & Carlos Maté, 2009) y Combining nearest neighbor predictions and model-based predictions of realized variance: Does it pay? (Julián Andrada Félix, Fernando Fernández-Rodríguez & Ana-Maria Fuertes, 2016). Sin embargo, a pesar de haberse tratado este tema en el ámbito teórico, el k-NN no suele ser una técnica que esté implementada en el software estadístico de predicción. Esto provoca que los usuarios de este tipo de herramientas se vean obligados a llevar a cabo algunas de las siguientes acciones:

- Implementar su propia versión del k-NN.

- Adaptar el k-NN aplicado a problemas de regresión a la predicción de series temporales.

Si bien es posible realizar la última acción mencionada, la predicción de series temporales requiere el manejo de ciertos aspectos que no son tomados en cuenta para los problemas de regresión. Estos son:

- La secuencialidad de las variables de entrada.
- La existencia de características intrínsecas en la serie, como pueden ser la estacionalidad y la tendencia.
- El hecho de que para predecir un valor para un instante dado solamente se pueden buscar instancias en el pasado de dicho instante, lo que hace que la fase de entrenamiento no se realice como en un problema de regresión cualquiera.

1.3 Objetivos

El objetivo de este trabajo es implementar el método del k-NN para la predicción de series temporales. Nuestra principal motivación es suplir la carencia de implementaciones de este método en el ámbito de la predicción de series temporales. Por una parte se desarrollarán herramientas dedicadas únicamente a la predicción mediante este método. Por otro lado, se implementarán una serie de funciones que permitan obtener los valores óptimos de los parámetros k y d a partir de una serie temporal y un conjunto de *k's* y *d's* a analizar. Se pretende que las herramientas que se desarrollen sean homologables con el paquete `forecast` del repositorio CRAN (The Comprehensive R Archive Network), el cual ha sido desarrollado principalmente por el estadístico Rob J. Hyndman.

También se proporcionará una interfaz gráfica con el objetivo de poder interpretar de manera más clara los parámetros k y d. La interfaz contendrá dos pestañas: la primera, con el objetivo de observar cómo se desempeña el k-NN con la k y d óptimas en comparación con otros métodos; y la segunda, con vistas a estudiar los valores de *k's* y *d's* analizados en la etapa de optimización.

1.4 Plan de trabajo

Para la realización del proyecto se llevarán a cabo las fases que a continuación se exponen:

- Familiarización con el lenguaje de programacion R, así como con los fundamentos teóricos detrás de la predicción de series temporales, las técnicas más comunes de predicción y la dinámica de funcionamiento del método del k-NN.

- Implementación de una primera función que lleve a cabo la predicción de un instante futuro utilizando el método del k-NN para una cierta serie temporal y unos determinados parámetros.

- Implementación de una función encargada de predecir una serie temporal observada utilizando el k-NN con unos parámetros k y d. Esto quiere decir que las predicciones que se realicen se corresponderán a un espacio temporal pasado. Esta función tiene como objetivo estudiar la calidad de la predicción en espacios de tiempo ya conocidos. De la misma manera, será útil para acercarnos a una futura implementación de una serie de funciones dedicadas a la optimización de los parámetros k y d del k-NN.

- Implementación de una función dedicada a la optimización de los parámetros k y d del k-NN. Se indicarán como entrada las *k's* y las *d's* a analizar, y se obtendrá el par con el que el k-NN rinde mejor.

- Análisis de partes del código que son optimizables, planteamiento de las posibles soluciones e implementación de las mismas. A priori, se prevé optimizar dos puntos: el cálculo de distancias y el cómputo de las predicciones. Esto último es necesario para garantizar que la optimización de los parámetros sea escalable a series temporales de mayor tamaño.

- Realización de una interfaz gráfica. Para esto, primero se investigará sobre las herramientas disponibles para poder graficar, después de desarrollarán los componentes básicos y por último, se pasará a una fase de integración de todos ellos en la misma interfaz.

2 Introduction

In today's world, predicting is done in multiple areas, whether it is for making political decisions related to the carbon emissions due to the current growth trend, determining the business strategy of a company according to the financial market behavior, or to detect possible natural disasters based on a model that precisely captures tectonic plates movements.

In order to perform the before mentioned actions, there are different kinds of methods that can be used. These ones can be divided in two groups [1]:

- Qualitative forecasting methods which can be used when there are no data available or if the available data are not relevant.
- Quantitative forecasting method that are used when there are numeric data about the past with patterns that we can assume to be repeated in the future. The majority of these methods can be applied to cross-sectional data problems, which means that the data have been collected at a single point; or data collected at regular intervals over time, which is known as time series.

This work is focused on a technique of the time series forecasting field. Therefore, it is necessary to present the concept of time series. In general terms, a time series is a phenomenon observed at regular intervals over time. A time series is represented by the following two:

- The time instants that it covers, $\{t_1, t_2, \ldots, t_{n-1}, t_n\}$, for a time series of length n.
- The values regarding the time instants: $\{y_1, y_2, \ldots, y_{n-1}, y_n\}$ in case of an observed time series, or $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$ if the time series has been forecasted.

It is also necessary to present the basic patterns that can be found in a time series. These are:

- Trend: this occurs if there is a long-term increase or decrease in the data [2].
- Seasonality: this exists when a time series is influenced by seasonal factors (e.g., month, year, semester, etc) [2].

Regarding forecasting of time series, there are two kinds of methods: the statistical methods and the methods based on artificial intelligence. In the first group there are the ARIMA models and the exponential smoothing models. These first ones attempt to describe the autocorrelations within the data. The second models are based on a description of the trend and the seasonality of the time series [3]. These techniques based on statistics are known for having non interpretable parameters.

In the second kind of techniques that are based on artificial intelligence the interpretability of the parameters is not simple. Within in this group, there is the neural nets. A example of neural nets is the multilayer perceptron: this is good for forecasting as it is an universal function approximator, although its parameters are not interpretable. Besides the neural nets there are other techniques which can be included in this group. One of them is the k-NN method, this is the technique that this work is based on.

2.1 The k nearest neighbors

The *k-nearest neighbors algorithm* (k-NN) is a simple and intuitive method for classification and regression. Nevertheless, due to its versatility it can also be used for forecasting time series. In this field, this technique consists of:

1. Finding the k periods, nearest to the most recent period. Each of these has length n.
2. Calculating the mean of these periods next values. This will be the predicted value.

In this work, each period of length d we will address as *element*. We represent an *element* by its length and the instants that compose it in the following way: $\{t_{i-d+1}, t_{i-d+2}, \ldots, t_{i-1}, t_i\}$.

In order to present how the dynamics of the k-NN is showcased, Example 2.1 is provided.

Ejemplo 2.1. We want to predict the amount of sells of an ice cream establishment. The data available is a record of sells of the last nine days as we can see in Table 2 . Graphically, the time series is represented in the Figure 2. In order to predict the value corresponding to instant t_{10} the k-NN method will be applied with $k = 2$ and $d = 2$.

Day	S	Sn	M	T	W	Th	F	S	Sn
t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
x_i	32	34	17	15	34	30	20	34	36

Tabla 2: Time series of Example 2.1

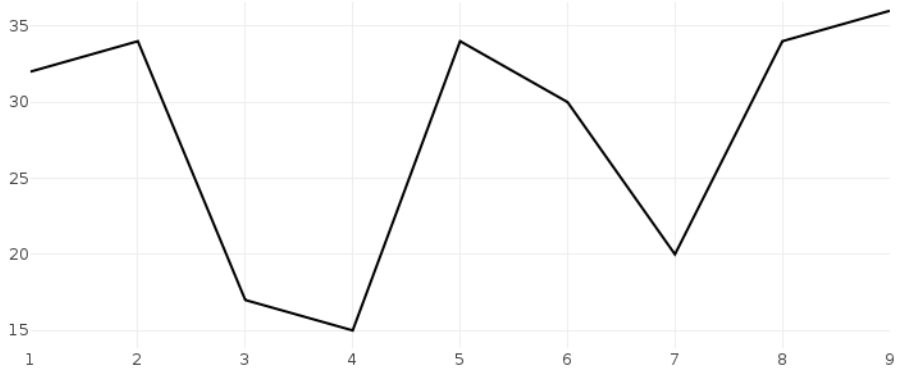

Figura 2: Graphical representation of the time series

Applying the k-NN method we find the the most recent *element* $\{t_8, t_9\}$ has as its most nearest neighbors, *elements* determined by instants $\{t_1, t_2\}$ and $\{t_5, t_6\}$. The next instants of these *elements* are $\{t_3\}$ and $\{t_7\}$. Therefore, the predicted value is calculated in the following manner:

$$\frac{x_3 + x_7}{2} = \frac{17 + 20}{2} = 18.5$$

As we observed in Example 2.1, the values of k and d have been chosen arbitrarily. In reality, in order to choose these values it is necessary to train the k-NN method for a a given time series. Normally, the series is divided in two parts: a first one for training and a second one for validation purposes. In the training phase it is fundamental to test a wide range of values of k and d so that we can rely on the calculated optimal combination of these two. This process can have a large computational cost.

Another important aspect to consider about the k-NN method is the interpretability of its parameters. The k-NN is a non parametric prediction method, this means that a priori a model is not specified. It is needed to mention that this does not mean that it does not have parameters but that they belong to an underlying statistical model. Therefore its interpretation is in theory more complex. This work intends to facilitate this.

2.2 Background

The *k-nearest neighbors* method, usually known as k-NN, appears for the fist time in a non published report of the US Air Force School of Aviation Medicine in 1951. In that document Fix and Hodges presented this non parametric method of pattern recognition [4]. In 1967 some formal properties were discovered regarding this method (Cover & Hart, 1967), this event was an important predecessor of the future researches.

In general, this technique has been used in classification and regression problems, however its versatility has allowed it to be used in density estimation and time series forecasting [5]. In time series forecasting it is possible to find works made during the last years. In specific, two of them are: Forecasting histogram time series with k-nearest neighbours methods (Javier Arroyo & Carlos Maté, 2009) and Combining nearest neighbor predictions and model-based predictions of realized variance: Does it pay? (Julián Andrada Félix, Fernando Fernández-Rodríguez & Ana-Maria Fuertes, 2016). Nevertheless, despite the research that has been made in the theoretical field, the k-NN is not usually implemented in the standards statistical software of prediction. This causes users of these kinds of tools, to be forced to do one of the following actions:

- To implement their own version of the k-NN.
- To adapt the k-NN for regression problems to time series forecasting.

While it is possible to do the last action mentioned, time series forecasting requires the management of certain aspects that are not considered for regression problems. These are:

- The sequentiality of input variables.
- The existence of intrinsics characteristics of the time series (e.g. seasonality and trend).
- The fact that forecasting a value for a specific instant is only doable if there are enough instances to look for in the past. This makes the training phase not possible to be executed as a normal regression problem.

2.3 Objectives

The objective of this work is to implement the k-NN method for forecasting time series. Our main motivation is to provide an alternative as there is a lack of implementations of this method in the field of forecasting time series. For this purpose, first we will develop a series of functions dedicated solely to prediction. Secondly we will implement a series of functions that allow us to obtain the optimal values of parameters k and d. A time series and two sets of k's and d's to analyze will be provided as input. It is intended that the tools to be developed are in coherence with the `forecast` package of the CRAN (The Comprehensive R Archive Network) repository. This package has been developed by Rob J. Hyndman.

Furthermore, a graphical user interface will be developed with the objective of interpreting in a more clear manner the parameters k and d. The interface will have two tabs: the first with the objective of observing how well the k-NN method performs in comparison to others; and the second one in order to study the values of k's and d's analyzed in the optimization phase.

2.4 Project plan

In order to achieve the project's objectives we have to establish some evident phases:

- Getting familiar with R programing language, the theoretical fundaments of time series forecasting, the most commonly used techniques and the dynamics of the k-NN.

- Implementation of a first function that forecasts an instant future using the k-NN method with a time series and other parameters.

- Implementation of a function that forecasts a time series using k-NN with parameters k and d. This means that the predictions that are calculated, correspond to a past-time space. The objective of this function is to study the quality of predictions that belong in the past-time spaces. Therefore, it will also be useful to get closer to a future implementation of a series of functions dedicated to optimize the parameters k y d of the k-NN method.

- Implementation of a function dedicated to optimize the parameters k y d of the k-NN. Values of k's and d's will be provided as input and then analyzed, and the pair of these values that produces a better performance of the k-NN will be returned.

- Analysis of the fragments of the code that are optimizable, approach to the possible solutions and the implementation of them. It is expected to optimize two main parts of the code: calculation of distances and calculation of predictions. This last part is critical, and it needs to be optimized in order to guarantee scalability to bigger data.

- Development of a graphical user interface. In order to achieve this, first we will research about the available tools to plot the results, after that we will develop all the basic components and finally we will integrate all of them in a single interface.

3 k-NN para predicción de series temporales

El algoritmo del k-NN es un método de aprendizaje automático basado en instancias. En breves palabras, este algoritmo con consiste en buscar entre todos los elementos de un conjunto de datos los que sean más parecidos de acuerdo a una función de distancia, y generar un resultado en función del valor de salida.

Otra característica del k-NN es que se considera un método de aprendizaje perezoso. Ésto debido a que no aproxima la función por completo, sino que únicamente realiza una aproximación local y deja la realización de los cálculos para el momento de la asignación del resultado [5].

Como se mencionó, el k-NN se suele utilizar para problemas de clasificación. En este caso, lo que se hace es buscar los vecinos que más se asemejan al elemento a clasificar, y en función de qué tipo sean la mayoría se decide el tipo del elemento. Sin embargo, también se puede aplicar a problemas de regresión, de entre los cuales la regresión es un subtipo.

3.1 Descripción del k-NN

El algoritmo del k-NN para la predicción de series temporales se puede describir de la siguiente manera [6]:

1. La serie temporal proporcionada, $\{y_1, \ldots, y_{n-1}, y_n\}$, es transformada a una serie de *elementos* d-dimensionales tal que $y_t^d = \{y_{t-d+1}, \ldots, y_{t-1}, y_t\}$ donde d es el tamaño del *elemento*. Cada uno de los *elementos* puede ser representado con un punto en un espacio d-dimensional.

2. Se calculan las distancias entre el *elemento* más reciente $y_n^d = \{y_{n-d+1}, \ldots, y_{n-1}, y_n\}$ y cada uno del resto de *elementos*.

3. Se identifican los k *elementos* más cercanos al *elemento* más reciente. Estos *elementos* se denotan por $y_{t_1}^d, y_{t_2}^d, \ldots, y_{t_k}^d$.

4. En función de los k *elementos* vecinos, $y_{t_1}^d, y_{t_2}^d, \ldots, y_{t_k}^d$, la predicción se calcula como la media ponderada de sus valores siguientes:
$$\hat{y}_{n+1} = \frac{\sum_{i=1}^{k} w_i * y_{t_i}^d}{\sum_{i=1}^{k} w_i}$$

3.2 Parámetros *k* y *d*

El k-NN consta de dos parámetros que se explican a continación:

- k: indica el número de vecinos a considerar, es decir, el número de *elementos* más cercanos al *elemento* más reciente y_n^d. La k es un parámetro de suavizado, mientras más grande sea la k más se reduce el ruido. Sin embargo, esto hace que las predicciones sean más homogéneas. Si tiende a n, se tendría la media de la serie.

- d: número de retardos a considerar para cada *elemento*. Es necesario determinar un valor adecuado para este parámetro para poder determinar qué longitud de las secuencias es la apropiada para caracterizar vecinos que resulten útiles para predecir.

3.3 Distancia

Como se menciona en la descripción del k-NN, para calcular las distancias entre *elementos* se utiliza una función distancia o métrica. A continuación, se listan y explican brevemente algunas de las métricas que se pueden utilizar:

- Euclídea: longitud del segmento que une dos puntos. Para un espacio de n dimensiones y los vectores p y q, se define con la siguiente fórmula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}.$$

- Manhattan: suma de las diferencias absolutas de las coordenadas cartesianas de dos puntos. Para un espacio de n dimensiones en el que se encuentran los vectores p y q, se define con la siguiente fórmula:

$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

- Canberra: es la versión ponderada de la distancia Manhattan. Para un espacio de n dimensiones y los vectores p y q, se define con la siguiente fórmula:

$$d(p,q) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|}$$

3.4 Ponderación

Como se aprecia en el cálculo de la media ponderada, para cada valor siguiente a cada *elemento* se asocia un determinado peso. La selección de estos pesos se puede realizar de distintas maneras. Algunos de lo métodos que se pueden emplear para el cálculo de estos pesos son:

- Proximidad: asigna a cada *elemento* un peso proporcional a su distancia. Por lo tanto, los pesos quedan asignados de la siguiente manera:

$$w_1 = 1/d_1$$

$$\vdots$$

$$w_{k-1} = 1/d_{k-1}$$

$$w_k = 1/d_k$$

  donde d_i es la distancia del *elemento* $y^d_{t_i}$ con el *elemento* más reciente.

- Igual: establece el mismo peso para todos los *elementos*. De esta manera, cada uno de los pesos queda establecido así:

$$w_1 = 1$$

$$\vdots$$

$$w_{k-1} = 1$$

$$w_k = 1$$

- Lineal: asigna el valor k al *elemento* más cercano, $k-1$ al segundo *elemento* más cercano, y así hasta fijar el valor 1 al *elemento* más lejano de los seleccionados. Por lo tanto, los pesos quedan asignados de esta forma:

$$w_1 = k$$

$$\vdots$$

$$w_{k-1} = 2$$

$$w_k = 1$$

donde w_1 se corresponde al peso que se le asigna al *elemento* más cercano, y w_k al *elemento* más lejano.

3.5 Optimización de los parámetros k y d

Para poder seleccionar los valores de los parámetros k y d de manera adecuada es necesario entrenar el k-NN con la serie temporal que se quiere trabajar. Para ello se suele dividir la serie temporal en dos partes: una para entrenamiento y otra para validación.

En la fase de entrenamiento se hacen pruebas del k-NN con todas las posibles combinaciones de k y d a partir de dos conjuntos de $k's$ y $d's$ que se proporcionan. Estas pruebas consisten en predecir valores que se corresponden a una sección del espacio temporal al que pertenece la serie para la que se entrena. De esta manera, se puede saber qué tan buenas son las predicciones con respecto a la serie temporal original. En definitiva, el objetivo de esta fase es encontrar la combinación de estos valores que derivan en un mejor rendimiento del k-NN sobre la serie temporal dada.

Hay que tener en cuenta que al tratarse de series temporales, las predicciones que se realicen solo pueden utilizar valores asociados a instantes pasados al del instante para el que se predice. Además se requiere que exista un pasado suficiente en el que se busquen vecinos. Debido a esto último, se hace uso de un parámetro llamado *init* que define el instante temporal que delimita el pasado. Éste garantiza que haya pasado suficiente si se escoge un valor adecuado. En general se recomienda que su valor sea de $0.7 * n$, donde n es la longitud de la serie temporal.

El procedimiento que se realiza se puede describir de la siguiente manera:

1. Se aplica el algoritmo del k-NN para cada una de las combinaciones de k y d obteniéndose para cada una de ellas un vector de valores predichos $\{\hat{y}_{init+1}, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$ correspondientes a los instantes $\{t_{init+1}, \ldots, t_{n-1}, t_n\}$. Aquí observamos como *init* delimita el pasado conocido. Para entender de manera clara su cometido y el de esta fase del proceso, en la Tabla 3 observamos cómo partes de la serie temporal y son tomadas como conocidas para predecir cada uno de los valores del vector mencionado.

2. Se calcula el error entre cada uno de los vectores de predicciones $\{\hat{y}_{init+1}, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$ y los valores reales de la serie temporal $\{y_{init+1}, \ldots, y_{n-1}, y_n\}$. De esta manera se obtiene un valor de error para cada combinación de k y d.

3. Se obtiene la combinación de valores de k y d con los que se minimiza el error. Cabe destacar que hay distintas maneras de realizar el cálculo del error.

Valor predicho	Pasado conocido
$\hat{y}_{init+1}$	$\{y_1, \ldots, y_{init-1}, y_{init}\}$
$\hat{y}_{init+2}$	$\{y_1, \ldots, y_{init}, y_{init+1}\}$
$\vdots$	$\vdots$
$\hat{y}_{n-1}$	$\{y_1, \ldots, y_{n-3}, y_{n-2}\}$
$\hat{y}_n$	$\{y_1, \ldots, y_{n-2}, y_{n-1}\}$

Tabla 3: Pasado conocido asumido para predecir

Posteriormente, en la fase de evaluación, se pone a prueba la combinación de k y d obtenida. Esto de cara a poder observar cómo de bien se predice en la sección de la serie temporal que no se ha usado para entrenar.

3.6 Error

Al hacer el cálculo de errores en la fase de entrenamiento es necesario hacer uso de una métrica concreta. Ésta sirve para cuantificar la diferencia entre los valores que se predigan $(\hat{y})$ y los valores reales de la serie temporal (y). A continuación se listan algunas de ellas:

- Error medio:

$$\frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i}}{\mathrm{n}}$$

- Raiz cuadrada del error cuadrático medio:

$$\sqrt{\frac{\sum_{\mathrm{t}=1}^{\mathrm{n}} (\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2}{\mathrm{n}}}$$

- Error medio absoluto:

$$\frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} |\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i}|}{\mathrm{n}}$$

- Error medio porcentual:

$$\frac{100}{\mathrm{n}} \sum_{t=1}^{n} \frac{y_i - \hat{y}_i}{y_i}$$

- Error porcentual absoluto medio:

$$\frac{100}{\mathrm{n}} \sum_{t=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

3.7 Efecto de la estacionalidad y tendencia de una serie temporal en el k-NN

Una característica que suelen tener algunas series temporales es la estacionalidad. Esto hace que los patrones que definen el comportamiento respondan a una determinada frecuencia de aparición de los datos y por tanto que la forma de predecir mejor la serie sea detectando la estacionalidad. Métodos como el *Ingenuo* (predecir usando el último valor) o el de la *Media histórica* tienen formas de adaptarse a esa estacionalidad de cara a mejorar la predicción, como son tomar el último valor correspondiente al intervalo anterior de la estación o la realización de la media de todas las estaciones anteriores, respectivamente. En el caso del k-NN, la ventaja de buscar entre todos los elementos es que si la serie responde a una estacionalidad, los vecinos que estén más próximos serán los que mejor definan la parte de la estacionalidad en la que se encuentra el elemento actual de la serie.

Por otro lado hay que tener en cuenta que algunas series puede presentar una tendencia, ya sea al alza o a la baja. Es importante destacar que en el caso del k-NN si la serie presenta este fenómeno, la predicción no podrá realizarse correctamente. Esto es debido a que no encontrará vecinos similares relevantes, ya que no están en el mismo nivel. Por este motivo, es importante diferenciar sobre qué tipo de serie estamos realizando la predicción.

Teniendo todo lo comentado anteriormente presente, podemos ver que aparte de la k y la d, hay más aspectos importantes a tener en cuenta a la hora de predecir una serie temporal. Esto da lugar a que en nuestra implementación decidiésemos incluir también como parámetros todo ese conjunto de posibilidades.

3.8 Series temporales multivariable

Además de todo lo comentado anteriormente, cabe destacar que en muchas ocasiones se pueden presentar la ocasión de trabajar con series multivariable. Estas son series en las que en cada instante temporal existe una medición de más de un atributo de la misma. Un ejemplo sencillo de esto sería una estación meteorológica, en la que además de la temperatura se miden otros elementos como pueden ser la presión atmosférica o la humedad.

A la hora de tratar estas series, es importante tener en cuenta que se debe observar el valor de todas las variables a la hora de calcular las distancias. Sin embargo, a la hora de realizar la predicción se debe hacer individualmente para cada una de las variables, ya que cada una puede responder mejor a una determinada combinación de k y d.

4 Implementación del k-NN para series temporales

La URL del repositorio del código es: https://github.com/Dani-Basta/TFG

En esta sección se describen las funciones principales implementadas. Se pretende explicar en detalle cómo se desarrollaron los algoritmos descritos en la Sección 3. A continuación se listan las funciones:

- `knn_next`: implementa el agoritmo de predicción del k-NN.
- `knn_optim`: implementa el algoritmo de optimizado de los parámetros k y d.
- `knn_past`: predice valores correspondientes a un espacio temporal pasado.
- `knn_elements`: implementa la transformación de la serie temporal a una serie de *elementos* d-dimensionales.

Es importante aclarar que todos los parámetros de entrada de estas funciones han sido tratados en la Sección 3, a excepción del parámetro v. Dicho parámetro es necesario para poder trabajar con series multivariables. Éste indica el índice que ocupa en la serie temporal multivariable la variable que se quiere predecir. Como ya se comentó anteriormente, aunque en las series temporales multivariable se haga el cálculo de las distancias entre *elementos* con todas sus variables, sólo se realiza la predicción sobre una de ellas.

4.1 Predicción de valores desconocidos *(knn_next)*

Esta función se encarga de hacer el cálculo de la predicción correspondiente al instante $n+1$ para una serie temporal de tamaño n. La base teórica de esta función se encuentra en la Sección 3.1. A continuación se presenta el pseudocódigo de esta función seguido de una explicación textual de cada una de las instrucciones:

Pseudocódigo 1: Algoritmo del k-NN que predice el instante $n+1$ para una serie temporal de tamaño n

1 function knn_next $(y, k, d, v, distancia, pesos)$;

Input : Una serie temporal y, parámetros k y d del k-NN, variable a predecir (v), métrica para calcular distancias $(distancia)$, manera de asignar pesos $(pesos)$

Output: El valor predicho

2 $matriz\_de\_elementos \leftarrow knn\_elements(y, d)$

3 $matriz\_de\_distancias \leftarrow dist(matriz\_de\_elementos, distancias)$

4 $vector\_de\_distancias \leftarrow matriz\_de\_distancias[:, 1]$

5 $vector\_de\_distancias \leftarrow ordena(vector\_distancias)$

6 $k\_nn \leftarrow obtiene\_primeros\_k\_valores(vector\_de\_distancias, k).indices$

7 $vector\_de\_pesos \leftarrow crear\_vector\_de\_pesos(vector\_de\_distancias, pesos)$

8 **return** $media\_ponderada(y[k\_nn + 1][v], vector\_de\_pesos)$

La semántica de cada una de las intrucciones del pseudocódigo presentado es la siguiente:

1. Cabecera de la función.

2. Pre-procesado de la serie temporal y que genera una matriz de *elementos* en función de la d proporcionada. De esta manera se acomodan los datos para luego calcular una matriz de distancias. En la Sección 4.4 se explica el comportamiento de esta función.

3. Cálculo de la matriz de distancias a partir de la matriz de *elementos* y la métrica de distancias proporcionadas. Dicha matriz contiene las distancias que se han calculado entre cada par de elementos posible.

4. Extracción de la columna de la matriz de *elementos* que contiene las distancias entre el *elemento* más reciente y el resto de *elementos*. Como se explica en la Sección 3.1, el *elemento* más reciente para una serie temporal de tamaño n es aquel que está compuesto por los valores asociados a los instantes $\{t_{n-d+1}, \ldots, t_{n-1}, t_n\}$ y que se representa como $y_n^d = \{y_{n-d+1}, \ldots, y_{n-1}, y_n\}$.

5. Ordenación del vector de distancias extraido de manera ascendente.

6. Obtención de los índices de los k *elementos* más cercanos que son seleccionados de la parte inicial del vector de distancias.

7. Asignación de los pesos a cada *elemento* seleccionado en la fase anterior según el parámetro *pesos*.

8. Cálculo de la predicción como una media ponderada a partir de los valores siguientes a los *elementos* seleccionados y los pesos previamente asignados. El resultado de esta instrucción es la predicción para el instante $n + 1$.

4.2 Búsqueda de parámetros óptimos *(knn_optim)*

Esta función se puede considerar la más crítica del paquete desarrollado. La base teórica de esta implementación se encuentra en la Sección 3.5, en la cual se habla del proceso de optimización de los parámetros k y d. A grandes rasgos, su objetivo es obtener los valores de k y d con los que el k-NN aplicado a la serie temporal (y) proporcionada rinde mejor. Para ello se analizan todas las posibles combinaciones de k y d, a partir de los rangos de ambos valores que se reciben como entrada. Es necesario aclarar que esta función devuelve los valores óptimos de k y d como ya se mencionó, y una matriz con los errores de todas las combinaciones de k y d analizadas.

Sobre nuestra implementación en concreto, las combinaciones de k y d han sido generadas con bucles anidados. En total, se ha abordado el proceso descrito con tres bucles anidados, dos externos para generar las combinaciones de k y d, y un interno para calcular el vector de predicciones $\{t_{init+1}, \ldots, t_{n-1}, t_n\}$. A continuación, se describe cada uno de estos bucles:

- Bucle externo: itera sobre el conjunto de $d's$ a analizar. Se decidió ubicarlo exteriormente debido al coste que implica hacer el preprocesado de datos y calcular la matriz de distancias, fases que se realizan en función de d. En el diagrama de la Figura 3 observamos de manera general el proceso que se lleva a cabo en cada iteración de este bucle. A continuación explicamos cada una de las fases del proceso:

  a) Preprocesamiento: los valores de la serie temporal proporcionada son *acomodados* para el posterior cálculo de la matriz de distancias. Esto da origen a una matriz de *elementos*.

  b) Cálculo de distancias: a partir de la matriz de *elementos* y la métrica de distancias proporcionada se calculan las distancias entre cada par de *elementos*.

  c) Cálculo de las predicciones: se obtiene una matriz de predicciones luego de la ejecución del bucle intermedio. Ésta almacena los vectores de valores predichos $\{\hat{y}_{init+1}, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$ para cada k a analizar y la d de la iteración correspondiente.

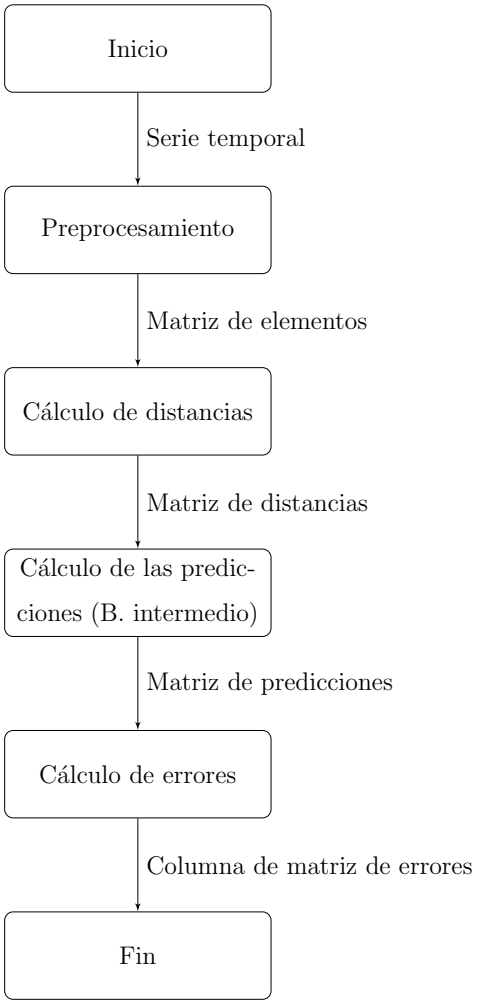

Figura 3: Diagrama de flujo del bucle externo

d) Cálculo de errores: se computan los errores entre cada uno de los vectores de valores predichos que se encuentran en la matriz de predicciones y los valores reales de la serie temporal. El tipo de error que se calcula viene indicado por el parámetro de entrada que indica que métrica de errores utilizar.

e) La *salida* de cada iteración consiste en una columna de la matriz de errores que se rellena. Esta columna contiene los errores para la d de la iteración correspondiente y todas las $k's$ a analizar.

f) Al finalizar la ejecución de este bucle la matriz de errores queda totalmente rellena.

Finalmente, en la Tabla 4 observamos la correspondencia de cada una de fases de este bucle con secciones del Pseudocódigo 2 proporcionado.

Fase	Pseudocódigo (línea/s)
Preprocesamiento	5
Cálculo de distancias	6
Cálculo de las predicciones	7-15
Cálculo de errores	17-20

Tabla 4: Correspondencia del Pseudocódigo 2 con el diagrama de la Figura 3

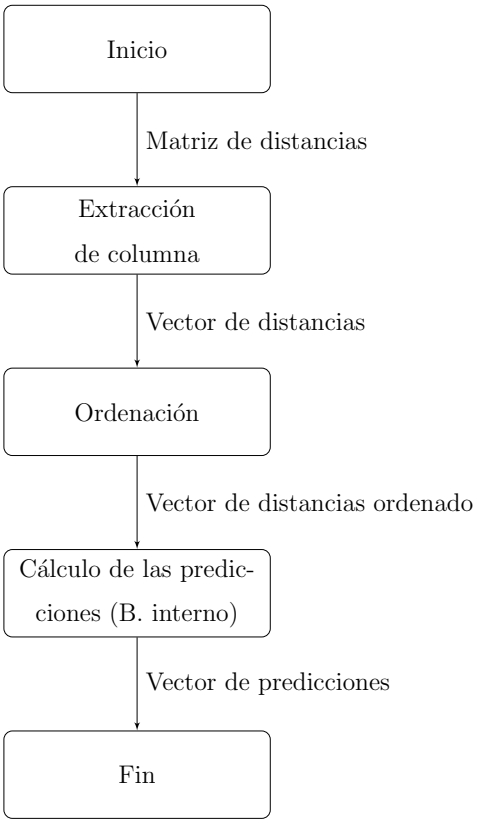

Figura 4: Diagrama de flujo del bucle intermedio

- Bucle intermedio: itera sobre los instantes $\{t_{init}, \ldots, t_{n-2}, t_{n-1}\}$. Cada uno de estos instantes delimita el pasado conocido para cada uno de los valores a predecir $\{\hat{y}_{init+1}, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$. En la Figura 4 podemos observar el proceso que se sigue en cada iteración. Las fases de dicho proceso vienen explicadas a continuación:

  a) Recibe como *entrada* del bucle externo, la matriz de distancias asociada a la d que se está analizando.

  b) Extracción de columna: se extrae una columna de la matriz de distancias en función del valor que se va a predecir. Para cada valor $\hat{y}_i$ que se desea predecir, se selecciona la columna correspondiente a las distancias entre el *elemento* determinado por los instantes $\{t_{i-d+1}, \ldots, t_{i-1}, t_i\}$ y el resto de *elementos* caracterizados por instantes más antiguos que éste.

  c) Ordenación: se ordena el vector de distancias extraído de forma ascendente.

d) Cálculo de las predicciones: se calculan predicciones para la d correspondiente a la iteración actual del bucle externo, el instante t_i asociado a la iteración actual de este bucle y todas las $k's$. Por lo tanto, se obtiene un vector de longitud número de $k's$ a analizar. Este vector pasa a rellenar una columna de la matriz de predicciones mencionada en el inciso *c)* de la explicación sobre el bucle externo.

e) Al completar todas las iteraciones de este bucle, se rellena la matriz de predicciones en su totalidad.

Para concluir con lo referente a este bucle, se presenta la Tabla 5. Ésta establece la corresponencia de cada una de las fases del proceso llevado a cabo por este bucle y las secciones del Pseudocódigo 2 proporcionado.

Fase	Pseudocódigo (línea/s)
Extracción de columna	8
Ordenación	9
Cálculo de las predicciones	10-15

Tabla 5: Correspondencia del Pseudocódigo 2 con el diagrama de la Figura 4

- Bucle interno: recorre cada una de las $k's$ a analizar. En el diagrama de la Figura 5 observamos las fases del proceso que se lleva a cabo en cada iteración del bucle. Seguidamente, se esclarecen algunos detalles sobre el proceso que realiza este bucle:

  a) Como *entrada* del bucle intermedio se proporciona un vector de distancias ordenado.

  b) Extracción de k elementos: se extraen k actual índices de los *elementos* más cercanos. Esto permite de alguna manera *reutilizar* el vector de distancias ordenado que *proporciona* el bucle intermedio. Debido a esto, se decidió ubicar a este bucle interamente.

  c) Ponderación: el vector de pesos se calcula de la forma especificada por el parámetro de entrada *pesos*.

d) Cálculo de predicción: se computa la media ponderada. Cada valor predicho pasa a rellenar una posición en concreto de la matriz de distancias. Dicha posición viene determinada por la d del bucle exterior, el instante temporal t_i del bucle intermedio y la k correspondiente a la iteración actual del bucle.

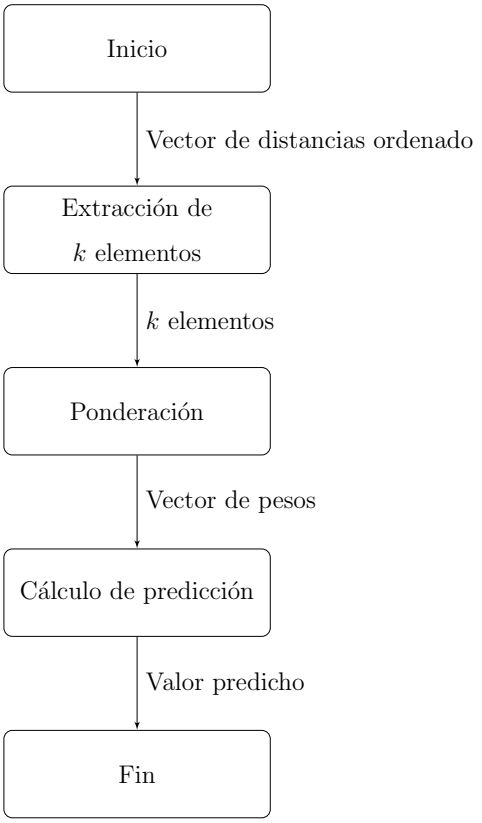

Figura 5: Diagrama de flujo del bucle interno

Finalmente, se presenta la Tabla 6 para observar cómo cada una de las fases que se realizan en cada iteración de este bucle queda correspondida con una sección del Pseudocódigo 2 proporcionado.

Fase	Pseudocódigo (línea)
Extracción de k elementos	11
Ponderación	12
Cálculo de predicción	13

Tabla 6: Correspondencia del Pseudocódigo 2 con el diagrama de la Figura 5

Pseudocódigo 2: Algoritmo de optimización de los parámetros k y d del k-NN para una serie temporal

```
1  function knn_optim (y, k, d, v, init, error, distancia, pesos);
   Input  : Una serie temporal y, conjuntos de k's y d's a analizar, variable a predecir
            (v), delmitador de pasado conocido (init), métrica para calcular errores
            (error), métrica para calcular distancias (distancia), manera de asignar
            pesos (pesos)
   Output: Matriz de errores, k y d óptimos
2  errores ← matriz(tam(k), tam(d))
3  for i ← 1 to tam(d) do
4      predicciones ← matriz(tam(k), n − init)
5      matriz_de_elementos ← knn_elements(y, d[i])
6      matriz_de_distancias ← dist(matriz_de_elementos, distancia)
7      for j ← 2 to n − init + 1 do
8          vector_de_distancias ← matriz_de_distancias[:, j]
9          vector_de_distancias ← ordena(vector_distancias)
10         for ind_k ← 1 to tam(k) do
11             k_nn ← obtiene_primeros_valores(vector_de_distancias, k[ind_k]).indices
12             vector_de_pesos ← crear_vector_pesos(vector_de_distancias, pesos)
13             valor_predicho ← media_ponderada(y[k_nn + 1][v], vector_de_pesos)
14             predicciones[k_index, n − init + 2 − j] ← valor_predicho
15         end for
16     end for
17     for ind_k ← 1 to tam(k) do
18         errores[ind_k, i] ← calcula_errores(predicciones[ind_k, :]), y[(init + 1) : n, v],
19         error)
20     end for
21 end for
22 return errores, mejor_d_y_k(errores)
```

4.3 Predicción de una serie ya conocida *(knn_past)*

Esta función recibe como parámetros de entrada los mismo que los de la función `knn_optim` a excepción de la métrica de error, ya que no se calculan errores. Su objetivo es generar un vector de predicciones $\{\hat{y}_{init+1}, \ldots, \hat{y}_{n-1}, \hat{y}_n\}$ aplicando el k-NN con una d y k concretas. Como podemos observar, estas predicciones se corresponden a una sección del espacio temporal de la serie proporcionada $\{y_1, \ldots, y_{n-1}, y_n\}$.

En una secuencia de uso normal, esta función se utiliza para generar las predicciones de las mejores combinaciones de k y d devueltas por `knn_optim`. Luego, utilizando la interfaz gráfica, se visualizan la serie temporal original y la predicha para poder analizar el error cometido.

En cuanto al proceso que se lleva a cabo para calcular el vector de predicciones existe una clara correspondencia con el `knn_optim`. En este último, se emplean tres bucles, dos de los cuales tienen la finalidad de generar todas las combinaciones posibles de $k's$ y $d's$. Como en esta función sólamente se aplica el k-NN para una k y d concretas, se prescinde de los dos bucles mencionados. Sin embargo, el bucle que itera sobre los instantes $\{t_{init}, \ldots, t_{n-2}, t_{n-1}\}$ se mantiene.

A continuación, se lista sin entrar en detalle cada una de las fases que se realizan con referencias al Pseudocódigo 3:

1. Reserva del vector de predicciones con longitud $n - init$. (2)
2. Preprocesamiento de la serie temporal y generación de la matriz de elementos. (3)
3. Se itera sobre los instantes $\{t_{init}, \ldots, t_{n-2}, t_{n-1}\}$. (6)
   a) Extracción de la columna que contiene los datos necesarios para predecir el valor correspondiente a la iteración actual. (7)
   b) Ordenación del vector de distancias extraído. (8)
   c) Extracción de los índices de los k *elementos* más cercanos. (9)
   d) Asignación de pesos a cada *elemento*. (10)
   e) Cálculo de la predicción como una media ponderada. (11)

Pseudocódigo 3: Algoritmo de predicción de pasado usando el k-NN

function knn_past $(y, k, d, v, init, distancia, pesos)$;

Input : Una serie temporal y, conjuntos de $k's$ y $d's$ a analizar, variable a predecir (v), delmitador de pasado conocido ($init$), métrica para calcular distancias ($distancia$), manera de asignar pesos ($pesos$)

Output: Vector de predicciones

$predicciones \leftarrow array(n - init)$

$matriz\_de\_elementos \leftarrow knn\_elements(y, d)$

$matriz\_de\_distancias \leftarrow dist(matriz\_de\_elementos, distancia)$

$i < -tam(predictions)$

for $j \leftarrow 2$ **to** $n - init + 1$ **do**

  $vector\_de\_distancias \leftarrow matriz\_de\_distancias[:, j]$

  $vector\_de\_distancias \leftarrow ordena(vector\_distancias)$

  $k\_nn \leftarrow obtiene\_primeros\_valores(vector\_de\_distancias, k).indices$

  $vector\_de\_pesos \leftarrow crear\_vector\_pesos(vector\_de\_distancias, pesos)$

  $valor\_predicho \leftarrow media\_ponderada(y[k\_nn + 1][v], vector\_de\_pesos)$

  $predicciones[i] \leftarrow valor\_predicho$

  $i \leftarrow i - 1$

end for

return predicciones

4.4 Preprocesado de datos *(knn_elements)*

Esta función recibe como entrada una serie temporal de tamaño n y una cierta d. Esta función se llama desde las demás para generar todos los elementos. Su objetivo es acomodar los datos para calcular una matriz de distancias. Para ello, crea una matriz de *elementos* de tamaño *nxd* en la cual la primera fila tenga al *elemento* más reciente; la segunda, al segundo *elemento* más reciente; y así, hasta tener al *elemento* más antiguo en la última fila.

De manera genérica, para una serie temporal de tamaño n como la de la Tabla 7 y una cierta d obtenemos la matriz de elementos de la Tabla 8 al aplicar esta función.

t_1	t_2	t_3	$\cdots$	t_i	t_{i+1}	$\cdots$	t_{n-2}	t_{n-1}	t_n
y_1	y_2	y_3	$\cdots$	y_i	y_{i+1}	$\cdots$	y_{n-2}	y_{n-1}	y_n

Tabla 7: Serie temporal genérica

y_{n-d+1}	$\cdots$	y_{n-1}	y_n
y_{n-d}	$\cdots$	y_{n-2}	y_{n-1}
y_{n-2}	$\cdots$	y_{n-3}	y_{n-d-1}
$\vdots$	$\ddots$	$\vdots$	$\vdots$
y_{i-d+1}	$\cdots$	y_{i-1}	y_i
y_{i-d}	$\cdots$	y_{i-2}	y_{i-1}
$\vdots$	$\ddots$	$\vdots$	$\vdots$
y_2	$\cdots$	y_d	y_{d+1}
y_1	$\cdots$	y_{d-1}	y_d

Tabla 8: Matriz de elementos genérica

En el Ejemplo 4.1 se muestra cómo se comporta la función para una serie temporal concreta.

Ejemplo 4.1. Se tiene una serie temporal univariable de tamaño 10 (Tabla 9) a la que se le aplica la función de preprocesado con una $d = 3$. La matriz resultante tiene la apariencia de la Tabla 10.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
9	2	3	7	4	1	5	8	6	0

Tabla 9: Serie temporal del Ejemplo 4.1

8	6	0
5	8	6
1	5	8
4	1	5
7	4	1
3	7	4
2	3	7
9	2	3

Tabla 10: Matriz de elementos del Ejemplo 4.1

5 Optimización de las funciones

Una vez que tenemos la función `knn_optim` funcionando correctamente pasamos a la etapa de optimización. Primero, se realizó un análisis de las partes del código que eran paralelizables. De dicho análisis concluimos que existían dos fragmentos de código optimizables:

- Cálculo de las matrices de distancias.
- Cálculo de las predicciones y de la matriz de errores.

En el caso de las matrices de distancias, la paralelización fue resuelta mediante el uso de la función `parDist` del paquete `parallelDist` del repositorio CRAN. Esta función se encarga de calcular las matrices de distancias usando un número determinado de hilos de ejecución. Utilizando esta función, se hicieron una serie de mediciones de los tiempos de ejecución de las distintas funciones desarrolladas. Dado que el tiempo que tardan en ejecutarse `knn_elements`, `knn_past` y `knn_next` es muy bajo, no hubo necesidad de intentar optimizar nada más. Sin embargo, al llegar a la función `knn_optim` esto ya no es así. Como se puede observar en la Tabla 11 a mayor cantidad de $k's$ y $d's$ que probemos, se amplía el espacio de búsqueda, y por tanto el cálculo varía enormemente. Las pruebas se hicieron en un portatil con 4 cores y Debian usando 3 threads en paralelo.

k \ ***d***	1:10	1:15	1:40
1:10	10,76	15,75	56,94
1:50	19,48	29,98	94,80

Tabla 11: Tiempos de ejecución de *knn_optim* en segundos

Esto dio lugar a la optimización de esta segunda parte, y para ello se fueron abordaron distintas estrategias sucesivas, refinando más en cada iteración, con el fin de reducir cada vez más los tiempos de ejecución.

5.1 Múltiples hilos en memoria

En una primera aproximación se paralelizó por el número de *d's* y por el número de instantes a predecir. Esta implementación es la relativa a la función `knn_optim_parallel`. En esta versión cada hilo calcula para una d concreta y un instante determinado tantas predicciones como *k's* se estén analizando. Otro detalle correspondiente a esta versión es que el cálculo de errores pasó a realizarse después de la ejecución de todos los hilos. Esto es debido a que los datos necesarios para el cómputo de este valor no están disponibles hasta la terminación de todos los hilos.

Como se puede ver en la Tabla 12, se redujeron los tiempos, pero seguía teniendo capacidad de mejora. Además se puede apreciar que en este momento los cambios en la cantidad de de $k's$ a probar no afectan demasiado al empezar a paralelizar, con lo que las mediciones se hicieron sin probar distintas $k's$ a partir de este momento.

k \ ***d***	1:10	1:15	1:40
1:10	10,76	15,75	56,94
1:50	19,48	29,98	94,80

k \ ***d***	1:10	1:15	1:40
1:10	19,88	29,01	*swap*
1:50	23,77	36,87	*swap*

Tabla 12: Tiempos de ejecución de *knn_optim* y *knn_optim_parallel* en segundos

Una segunda aproximación fue paralelizar el cálculo de las predicciones y errores solo por el número de *d's*. Esta versión se corresponde con la función `knn_optim_parallel2`. En este caso, cada hilo computa $n - init$ predicciones correspondientes a los instantes que van desde $init + 1$ a n, todo esto para cada k a analizar. Finalmente cada hilo calcula los errores para la d que tiene asociada y para todas las *k's* que se están explorando. Como se puede ver en la Tabla 13

d	1:10	1:15	1:40
knn_optim	19,48	29,98	94,80
knn_optim_parallel	23,77	36,87	*swap*
knn_optim_parallel2	17,59	30,65	*swap*

Tabla 13: Tiempos de ejecución en segundos con k de 1 a 50

El gran problema de estas dos soluciones fue el uso excesivo de memoria principal. Por un lado el hilo padre tenía en memoria todas las matrices de distancias. Esto, en el caso de series temporales de gran tamaño y un extenso número de *d's*, puede llegar a ocupar varios GBs (del orden de varias decenas). Por otra parte, no se encontraron herramientas dentro de R que permitiesen proporcionar a cada hilo únicamente los datos necesarios para su fracción de cálculos. Esto ocasionaba que cada hilo tuviese **copias efectivas** de todas las matrices de distancias. Por este motivo, se podía llegar a producir swapping por parte del sistema operativo, dado que la memoria RAM se ve desbordada, causando así la ralentización extrema de los cálculos.

5.2 Optimización con uso de archivos

A partir de los problemas de memoria se propuso una solución un tanto contraintuitiva: ***utilizar archivos***. El objetivo era realizar un cacheo de datos manual entre memoria principal y memoria secundaria. Para esto se decidió dividir el proceso de optimización de *k's* y *d's* en dos funciones:

- Una primera únicamente para el cálculo de las matrices de distancias y posterior guardado en ficheros.
- Otra dedicada al cálculo de las predicciones y de la matriz de errores.

Una primera aproximación consistió en guardar una matriz de distancias por archivo, implementándose para ello la función `knn_distances`. Luego, para la parte del cálculo de las predicciones y matriz de errores, se paralelizó por número de *d's* e instantes temporales a predecir con la función `knn_parallelf`. Con esta solución se ahorró no tener constantemente en memoria principal todas las matrices de distancias en todos los hilos. Sin embargo, como cada hilo cargaba de fichero la matriz de distancias entera en la cual se encuentran sus datos, se continuó haciendo un uso excesivo e innecesario de memoria principal. Cada hilo iba a tratar sólo una cierta cantidad de instantes y sin embargo cargaba todos. Esto provoca que esta función sólo se pueda utilizar en series temporales pequeñas, ya que de lo contrario genera *swapping* a disco.

Finalmente, la solución óptima fue guardar cada matriz de distancias en archivos que contuviesen como máximo un número determinado de columnas de las matrices de distancias. Esta forma de implementarlo es la que está presente en la función `knn_distances2`. Posteriormente, se realiza el cálculo de las predicciones paralelizando por número de *d's* e instantes de tiempo calculables con los datos de cada archivo, dando por tanto lugar a $d(n - init)/columnas$ hilos. Como se puede ver en la Tabla 14 la solución de guardar sólo una determinada cantidad de columnas por fichero aporta mejor rendimiento. Asimismo, es importante destacar que la cantidad de columnas por fichero también es un factor decisivo.

d	1:10	1:15	1:40
knn_optim	19,48	29,98	94,80
knn_optim_parallel2	17,59	30,65	*swap*
knn_optim_parallelf2	12	18	48

Tabla 14: Tiempos de ejecución en segundos con k de 1 a 50

columnas \ ***d***	1:10	1:15	1:40
1	12	18	48
10	22	25	80
100	66	100	275

Tabla 15: Tiempos de ejecución de $knn\_optim\_parallelf2$ en segundos

Por último cabe comentar que esta forma de implementación tiene una ventaja añadida: una vez calculadas las matrices de distancias, se pueden utilizar en tantas llamadas a la función de optimización como se quiera, sin necesidad de volver a recalcularlas. Además permite poder realizar la optimización en dos fases, sin necesidad de calcular las matrices y hacer las predicciones una a continuación de la otra.

A continuación se presentan dos tablas en las que resumen las estrategias que se siguen en las distintas funciones desarrolladas. En la Tabla 16 se pueden ver las funciones que hacer cálculos de las matrices de distancias y en la Tabla 17 las encargadas de hacer predicciones.

Tabla 16: Cálculo de distancias

Función	**Estrategia**	**Uso de memoria**
knn_optim	Uso de "parDist"	Solo una matriz de distancias a la vez
knn_optim_parallel	Uso de "parDist"	Todas las matrices de distancias $d * (n - init)$ copias
knn_optim_parallel2	Uso de "parDist"	Todas las matrices de distancias d copias
knn_distances	Uso de "parDist" y posterior guardado de las matrices de distancias en ficheros, uno por cada matriz	Solo una matriz de distancias a la vez
knn_distances2	Uso de "parDist" y posterior guardado de las matrices de distancias en ficheros, cada fichero contiene un máximo de columnas	Solo una matriz de distancias a la vez

Tabla 17: Cálculo de predicciones

Función	**Estrategia**	**Uso de memoria**
knn_optim	Secuencial	Matrices de distancias
knn_optim_parallel	Paralelización por número de d's y n-init instantes a predecir	Hilo padre - matrices de distancias Cada hilo - copia de las matrices de distancias
knn_optim_parallel2	Paralelización por el número de d's	Hilo padre - matrices de distancias Cada hilo - copia de las matrices de distancias
knn_optim_parallelf	Paralelización por número de d's y n-init instantes a predecir	Hilo padre - resultados Cada hilo - matriz de distancias leída de fichero
knn_optm_parallelf2	Paralelización por número de d's y n-init/columnas instantes a predecir	Hilo padre - resultados Cada hilo - columnas de la matriz de distancias leídas de fichero

6 Herramienta para la visualización del k-NN

Dada la naturaleza de este proyecto de análisis de datos, consideramos necesario aparte de hacer las funciones de procesamiento de series temporales, crear una interfaz para visualizar el resultado de las mismas y su utilidad. Como se ha comentado anteriormente, el k-NN es un método no paramétrico, y por tanto no es fácil de interpretar. De esto surge la motivación de crear una buena interfaz que ayude al usuario a entender las ventajas y desventajas de utilizar este método para una determinada serie con la que esté trabajando. Algunas de las posibles dudas a la hora de estar trabajando con el k-NN son:

- Una vez obtenida la combinación óptima de k y d, ¿a qué responde esa combinación concreta? ¿Existen otras combinaciones que sean igual de buenas? Y si es así, ¿en qué se diferencian?
- ¿Algunas combinaciones de k y d predicen sensiblemente peor?
- ¿Existen mínimos locales en el espacio de búsqueda de las combinaciones de k y d probadas? De ser así, ¿a qué responden dichos mínimos? De manera análoga puede ocurrir con máximos locales.
- ¿Hay propiedades inherentes a las series que se puedan descubrir al utilizar el k-NN?
- Puede ocurrir que determinadas combinaciones de k y d predigan mejor ciertas etapas de una serie. En caso de que las haya, ¿cuáles son y por qué?

En resumidas cuentas, lo que se busca con la interfaz es "abrir" la "caja negra" que puede suponer el método del k-NN. Para ello era necesario tener una forma de generar gráficas e interactuar con ellas fácilmente, así como tener la posibilidad de modificarlas de forma dinámica. De esto surge la necesidad de tener una forma de combinar gráficas de distintos tipos y de hacer que el usuario pueda interactuar con ellas sin necesidad de modificar líneas de código. Se buscaba generar dos tipos de vistas:

- Una debía mostrar distintas series temporales. El principal objetivo sería mostrar una determinada serie temporal junto a predicciones de la misma. Esto con el fin de poder ver cómo se ajustan las diferentes predicciones a la serie y ver dónde se

comete el error. Por ello, debe ser capaz de mostrar varias gráficas a la vez sin que esto suponga una superposición de las mismas que impida la correcta visualización.

- La otra debía reflejar de una forma sencilla e intuitiva qué combinaciones de k y d funcionan mejor. Esto con el objetivo de permitir ver mínimos locales de la función de error. Además, debía permitir ver de forma cómoda qué predicción genera una determinada combinación que no sea la óptima.

6.1 Gráficas interactivas *(plotly)*

Una primera aproximación a la generación de gráficas fue con `ggplot2`, paquete que es prácticamente un estándar en muchos desarrollos en *R*. Se utiliza para generar gráficas de diversos tipos de forma bastante sencilla. Sin embargo, después de varias pruebas no sentimos que cubriese las necesidades del proyecto, ya que no poseía el dinamismo e interactividad buscados (figura 6).

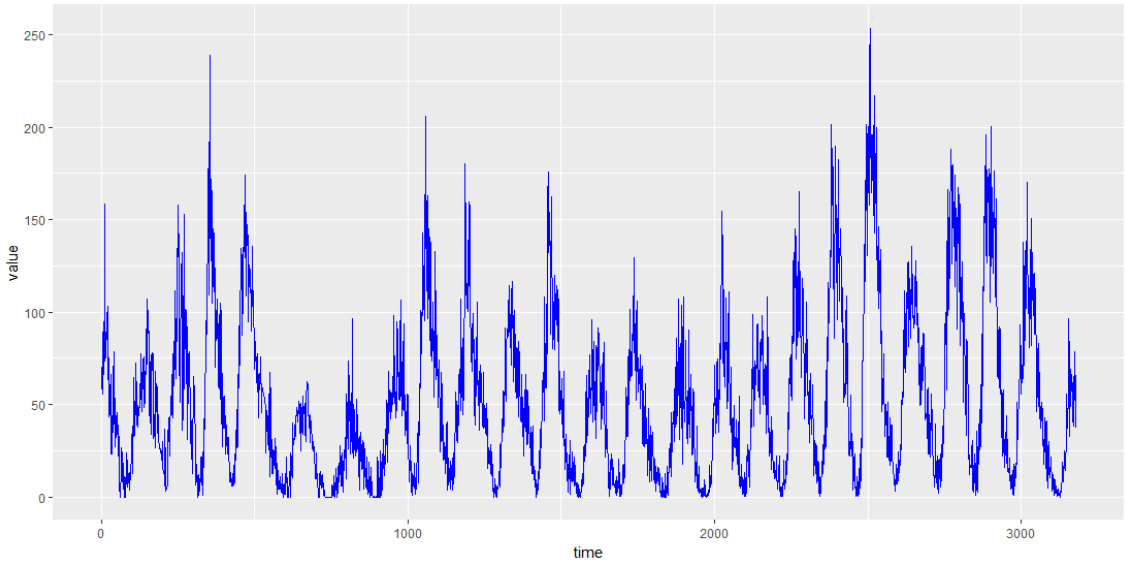

Figura 6: Gráfico de una serie temporal con *ggplot2*

Se optó por probar *Plotly*, un paquete de R que se basa en el desarrollo de ggplot2 y lo lleva más allá generando una intersección con Javascript y HTML5. Permite de esta manera una visualización nativa en navegadores web, así como una gran flexibilidad de uso por su variedad de tipos de gráficas y grado de personalización de las mismas.

6.1.1 Visualización de series temporales

El primer tipo de gráfica parecía necesitar un patrón muy estándar: una gráfica clásica que tenga en el eje X el tiempo y en el eje Y el valor de la serie temporal en cada instante (figura 7).

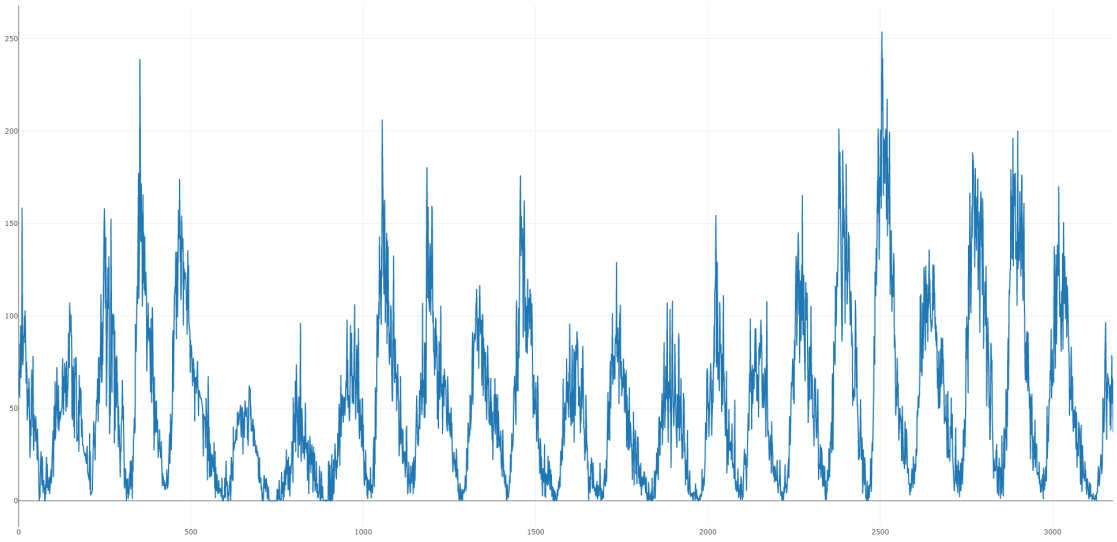

Figura 7: Gráfico básico en plotly

Como ya se ha comentado anteriormente, el principal problema que puede encontrarse es cuando se grafican distintas series de forma simultánea. Esto podía requerir la necesidad de eliminar y repintar una determinada serie u otra. Sin embargo, para hacer esto en *plotly* basta con hacer click en la leyenda de una determinada serie y esta se eliminará de la visualización, sin necesidad de volver a generar un gráfico desde cero. De manera análoga, al hacer click en dicha serie se vuelve a mostrar (figura 8).

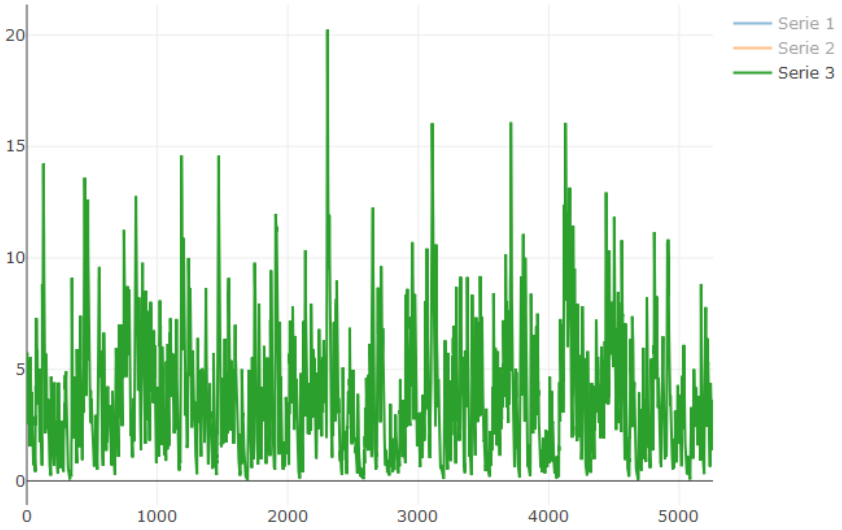

Figura 8: Gráfico mostrando 1 sola serie

Esto aporta mucha flexibilidad a la hora de implementarlo y una usabilidad sencilla y relativamente intuitiva para el usuario. Además, dada la presencia de varias series a la vez, se corría el riesgo perder precisión en el momento de visualizar las diferencias entre ellas. De nuevo, para evitar esto *plotly* permite también ampliar y reducir el zoom a una determinada zona en específico (figura 9).

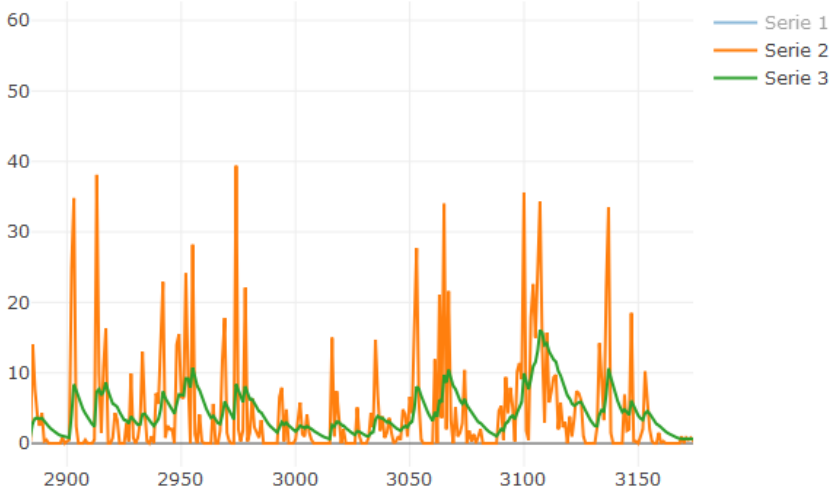

Figura 9: Figura 8 mostrando detalle de 2 series

Llegados a este punto, ya era posible visualizar varias series a la vez eligiendo cuáles se mostraban en cada momento y pudiendo ir al detalle de un determinado periodo. El siguiente paso fue al percatarnos de que al tener series muy largas, se puede querer visualizar una determinada cantidad de tiempo constante. Por ejemplo, para eventos como pueden ser las temperaturas, puede ser interesante observar doce meses consecutivos y que el resto de la serie no se muestre. Pero si se quiere avanzar o retroceder en el tiempo, no había una forma cómoda de hacerlo. Para esto la solución fue utilizar un *Slider*, que permitía delimitar cuánta cantidad de la serie se ve y desplazarse a lo largo de ella manteniendo fija dicha cantidad (figura 10).

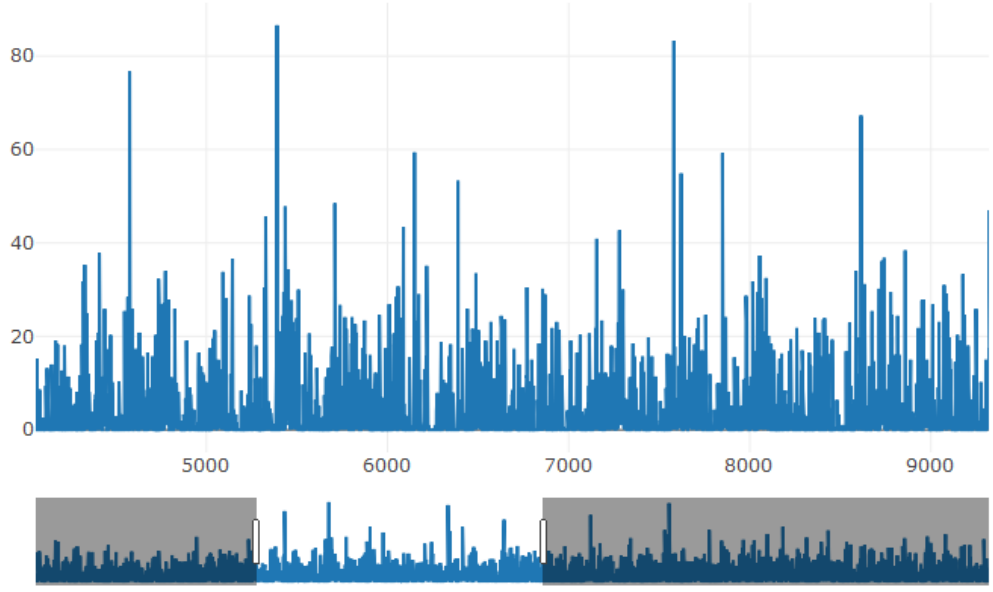

Figura 10: Serie muy larga usando Slider

Una vez conseguido esto, el siguiente paso fue intentar visualizar de forma sencilla qué zonas eran las que se predecian peor. Como ya se ha comentado anteriormente, uno de los principales objetivos del proyecto es poder distinguir si determinadas combinaciones de k y d predicen mejor ciertas etapas de las series y mal otras. Para ello, una alternativa que se presentaba era recorrer todo el gráfico viendo qué distancia había entre la serie y la predicción. Como se puede ver en la figura 11, esto resultaba muy laborioso e impráctico.

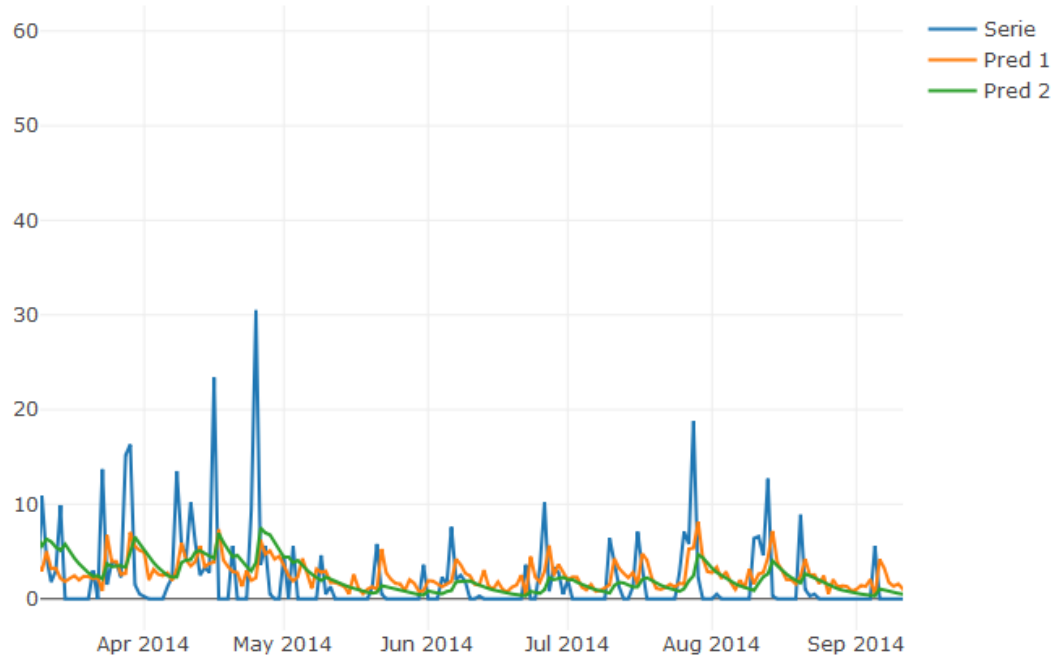

Figura 11: Gráfico con una serie y 2 predicciones

Para solucionar este problema la idea que surgió fue utilizar un diagrama de barras en el que se dibujase la diferencia entre valor de la serie y valor predicho, teniendo así en las zonas de peor predicción barras más altas y lo contrario en las de buena predicción. Sin embargo, esto ocasionaba otro problema a la hora de intentar ver la serie y los errores a la vez de manera que se sincronizasen, como se puede observar en la figura 12. La solución fue hacer una gráfica compuesta por las dos anteriores, y utilizar un slider compartido para no perder lo conseguido en la etapa anterior, consiguiendo así el resultado de la figura 13.

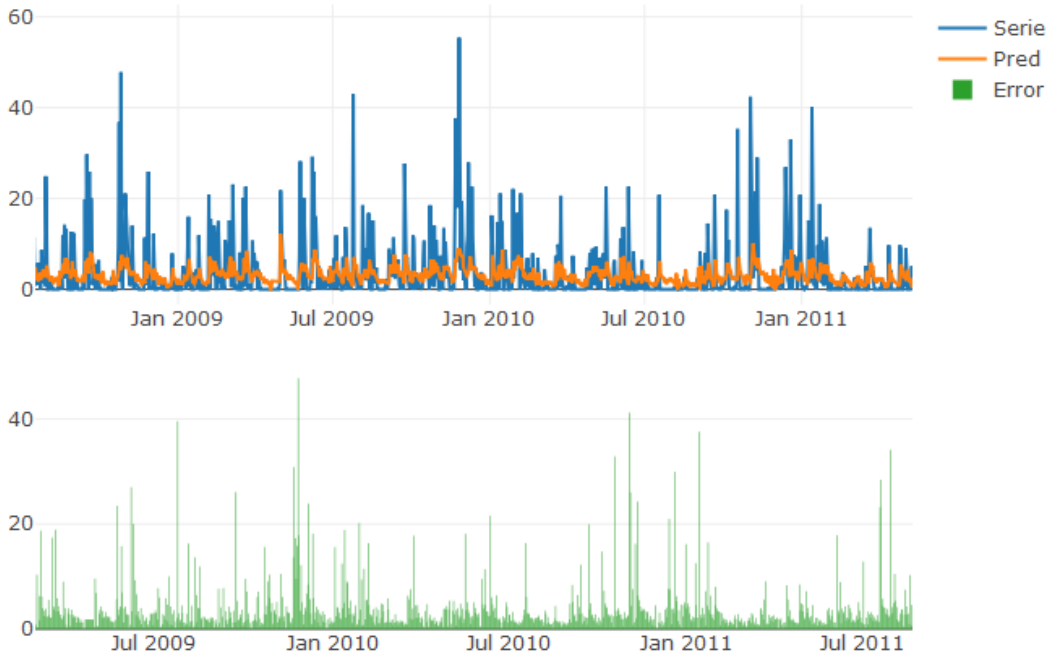

Figura 12: Gráficas sin sincronización

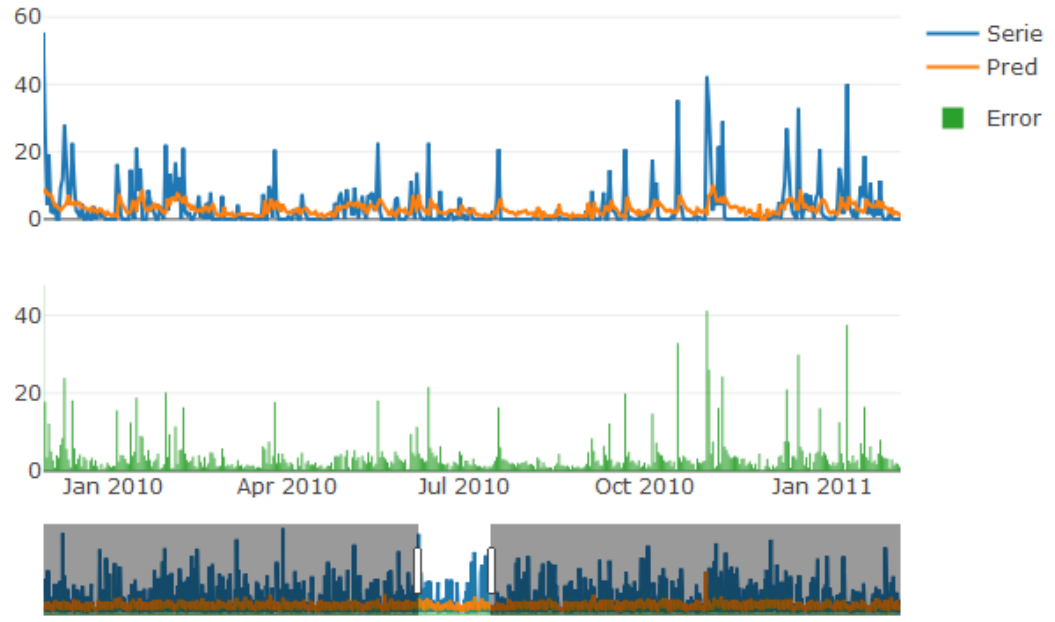

Figura 13: Gráficas ya sincronizadas con scroll y selección

Por último, una vez colocadas las barras de error y sincronizadas con las predicciones, se comprobó que al mostrar distintas series temporales de forma simultánea la lectura e interpretación de la gráfica se volvía algo compleja. Esto era debido a que se acorta la anchura de las barras para encajarlas en un instante temporal, como se puede ver en la figura 14.

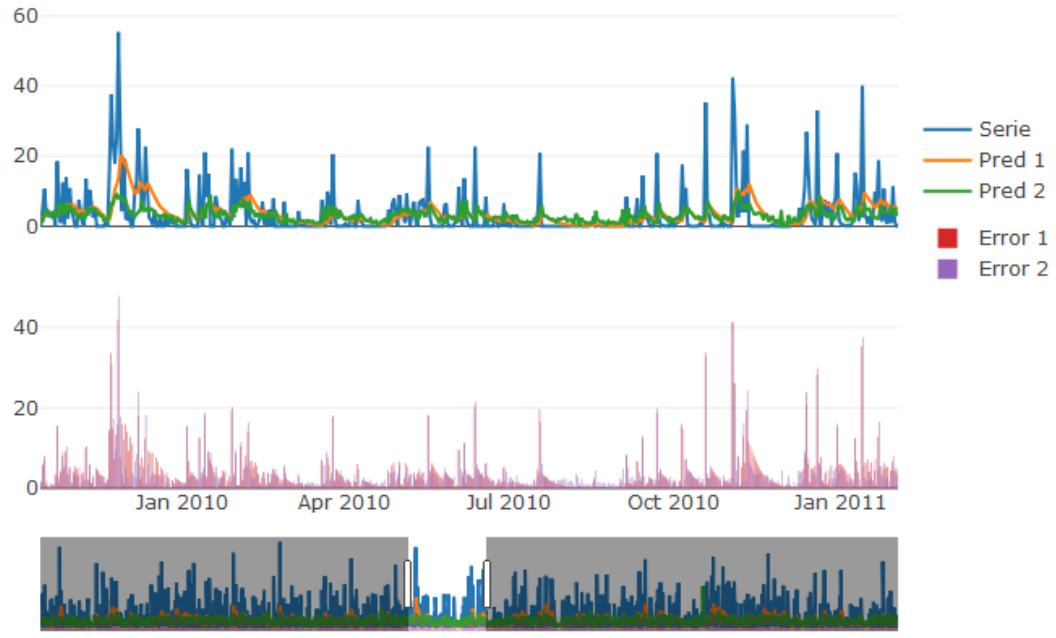

Figura 14: 2 predicciones con errores en barras

Este problema se agravaba especialmente en series largas, ya que había que reducir más el intervalo de tiempo que se visible si no se quería perder legibilidad. Para corregir este problema surgieron dos alternativas:

- Utilizar la funcionalidad de Plotly para mostrar y ocultar distintas gráficas, de manera que sea el usuario el que decida en cada momento mostrar sólo uno o dos de los diagramas de barras que estén presentes, perdiendo así la posibilidad de leer de forma cómoda varios errores a la vez.

- Cambiar el diagrama de barras por uno de líneas, de manera que la visualización de los errores de varias predicciones a la vez se volvía mucho más cómoda. Esto suponía renunciar ligeramente a la facilidad de interpretación que proporciona el diagrama de barras.

Dado que el problema sólo ocurre cuando se tienen varias predicciones graficadas al mismo tiempo, se optó por utilizar el gráfico de barras si hay una única predicción y el de líneas si hay más de una, dado lugar al gráfico que puede verse en una figura 15.

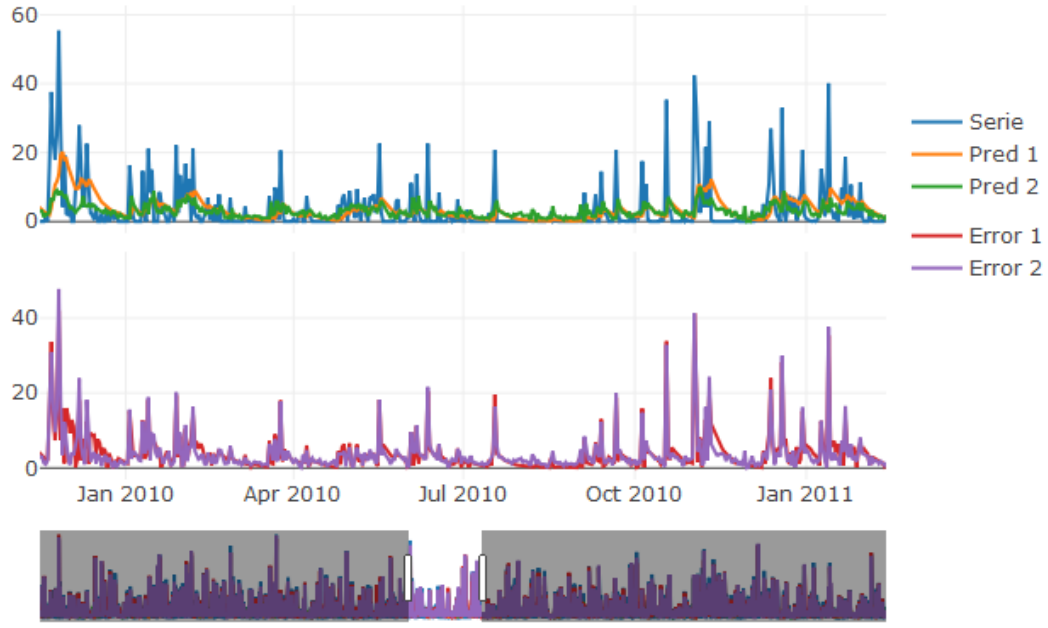

Figura 15: Dos predicciones con errores en lineas en la gráfica intermedia

6.1.2 Visualización de errores de k y d

En segundo lugar había que encontrar una forma de mostrar los errores generados con las distintas combinaciones de k y d que se habían probado durante el proceso de optimización. El principal escollo a superar a la hora de visualizar esto es el hecho de que para cada combinación tenemos un dato, lo que significa que tenemos 3 dimensiones en las variables. Esto planteaba dos alternativas:

- Realizar un gráfico tridimensional en el que se viesen las distintas combinaciones y para cada una el error fuese la altura.
- Realizar un gráfico bidimensional que fuese una proyección de un gráfico tridimensional, utilizando algún tipo de código de colores o similar, para representar la tercera dimensión que en este caso son los errores.

Dado que un gráfico en 3D en principio parece ser más intuitivo, decidimos trabajar en ello. Al tener una matriz de dos variables en la que cada posición tiene un determinado valor, la solución pensada consistió en hacer una gráfica de tipo superficie, como la que se puede ver en la figura 16. El problema que presentaba esta solución es que según el equipo en el que se visualizase y el motor de renderizado que se utilizase, se perdía mucha fluidez y hasta llegaba a notarse cierto nivel de ralentización. Por este motivo se pasó a probar la otra alternativa, de cara a comprobar si pedía menos recursos mejor sin perder la legibilidad del gráfico a cambio.

A la hora de abordar esta opción se presentaron dos posibilidades:

- Un diagrama de contorno, como los que se suelen utilizar para estudiar la orografía
- Un mapa de calor, práctico para encontrar máximos y/o mínimos locales

Al utilizar un mapa de calor se observó que la representación salía muy pixelada, ya que es una matriz con unas decenas de elementos o a lo sumo centenas. Esto es porque suelen funcionar mejor con mayores cantidades de datos. Algo similar ocurría con el diagrama de contorno, ya que funcionan mejor con transiciones más suaves entre elementos adyacentes, cosa que tiende a no ocurrir en este caso como se puede observar en la figura 16.

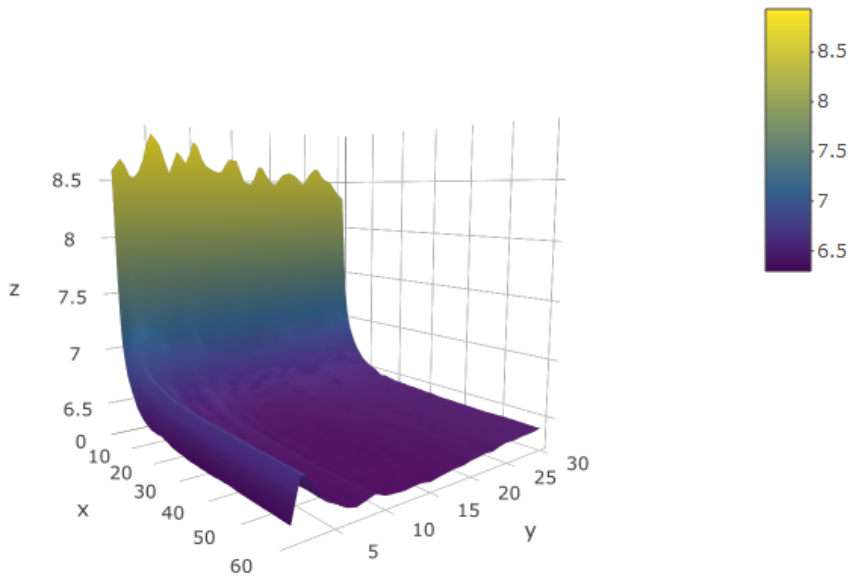

Figura 16: Gráfica tipo superficie en 3D. X es la k, Y es la d y el color es el error

La solución fue utilizar una combinación de ambos: un mapa de contorno en el que los colores asignados a cada región fuesen pintados con un degradado para representar la transición entre valores (ver figura 17).

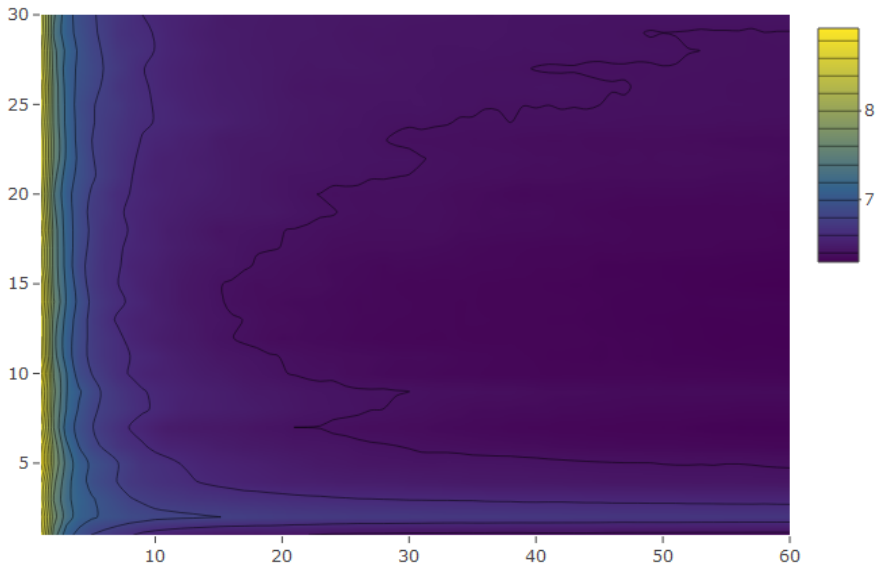

Figura 17: Gráfico de tipo contour-heatmap básico

Aunque esta solución ofrecía mucho mayor rendimiento se perdía cierta utilidad, ya que desaparecía la facilidad visual que aportaba la reprentación en 3D real pudiendo rotar para encontrar los mínimos. Para solucionar esto, se plantearon dos objetivos a mejorar:

- Intentar reducir el rango de valores para los que se utiliza todo el espectro de colores. Como se puede ver en las figuras 16 y 17, el rango entre uno y cinco tiene una gran transición de colores. Esto causa que en los niveles más bajos de error las diferencias sean comparativamente tan bajas que apenas queda gama cromática con la que poder apreciar diferencias.

- Intentar aportar una mejor legibilidad a las transiciones de valores por las que pasa el gráfico, pudiendo así observar a simple vista en qué rango está cada punto. Además había que intentar centrar las lineas de contorno en los valores más bajos, ya son los que minimizan el error y por tanto serán los importantes. Además los valores altos en general tendrán menos interés y por tanto tienen un papel secundario.

Para abordar el primer objetivo la primera idea que surgió fue truncar los valores altos de la matriz, que son el motivo por el que aumenta tanto el rango de valores presentes en la misma. El problema de esto era que se estaban modificando los valores obtenidos en la optimización, y aunque puedan no aportar demasiada información, es interesante que el usuario pueda ver qué error generan realmente esas combinaciones. La siguiente idea fue intentar que el gráfico utilizase el mismo color para las posiciones de la matriz que tengan valores mayores que uno determinado. Es decir, que a partir de un determinado valor de error satura el color. Esto hacía que se ganase gama cromática en el gráfico sin perder los valores reales de la matriz de optmización (figura 18).

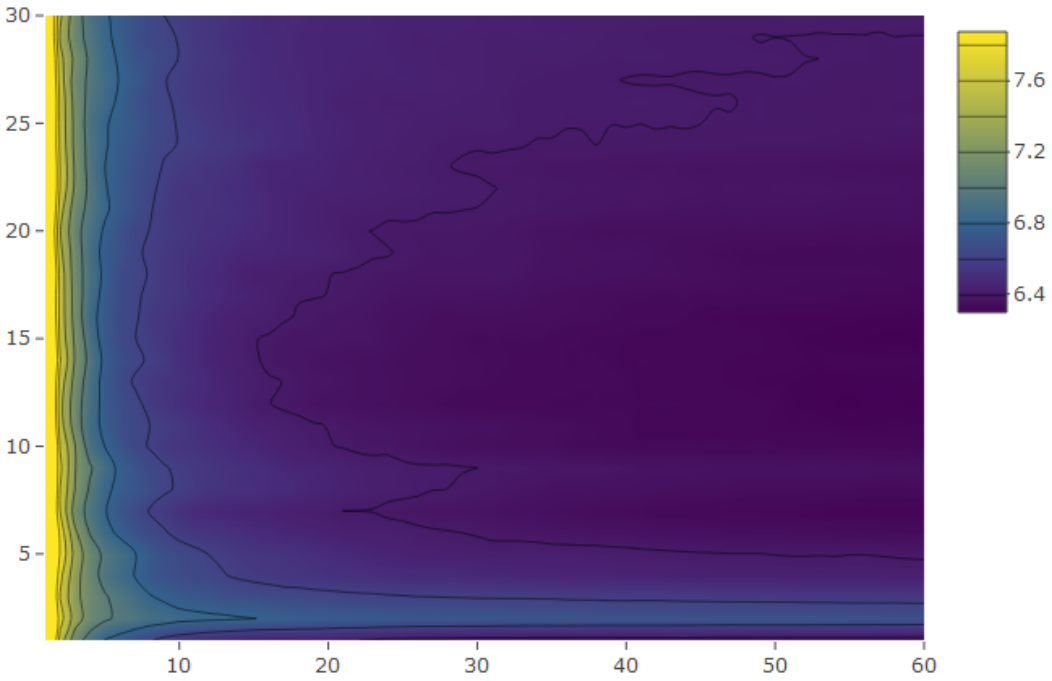

Figura 18: Gráfico contour con el máximo truncado

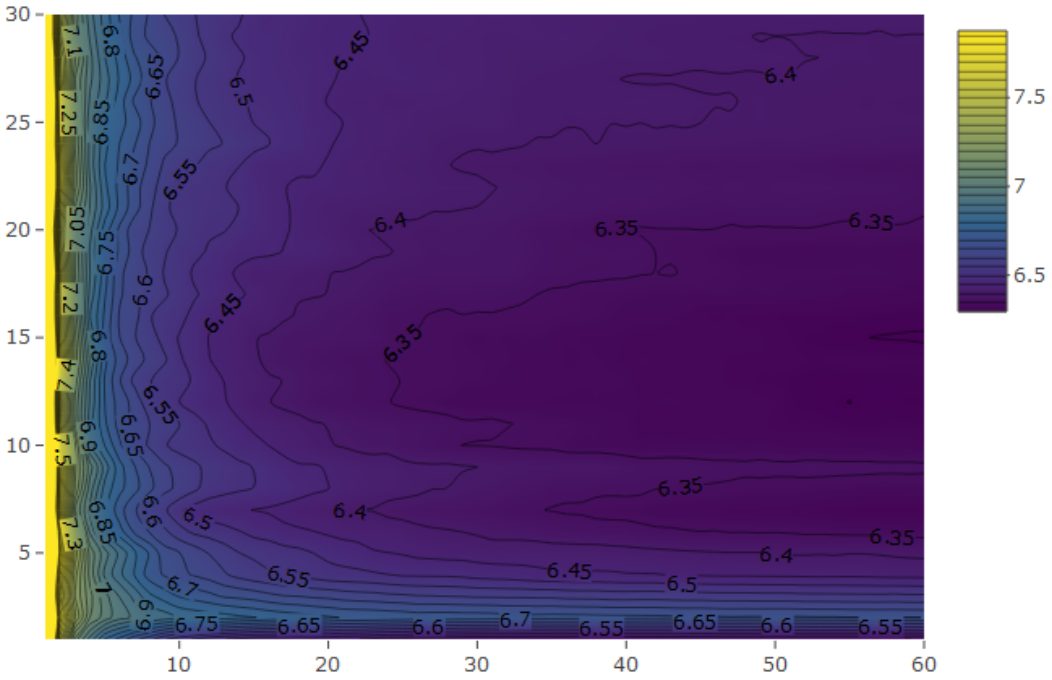

Figura 19: Gráfico con mayor cantidad de líneas de contorno

Para el segundo objetivo se optó por definir una mayor cantidad de líneas de contorno, de manera que en las zonas de menor transición hubiese más cantidad. Esto ocasionaba que hubiese zonas del gráfico que presentasen demasiadas líneas debido a grandes transiciones abruptas (figura 19). Por este motivo, se decidió que una alternativa mejor era definir sólo un rango de valores en el que se mostrasen líneas de contorno, solucionando así la baja cantidad en valores próximos sin saturar la gráfica con exceso de líneas. Además, se añadió una leyenda numérica a la gráfica, de manera que se pudiese observar de un vistazo qué valores tiene cada punto sin necesidad de pasar el cursor por encima de dicho punto (ver figura 20).

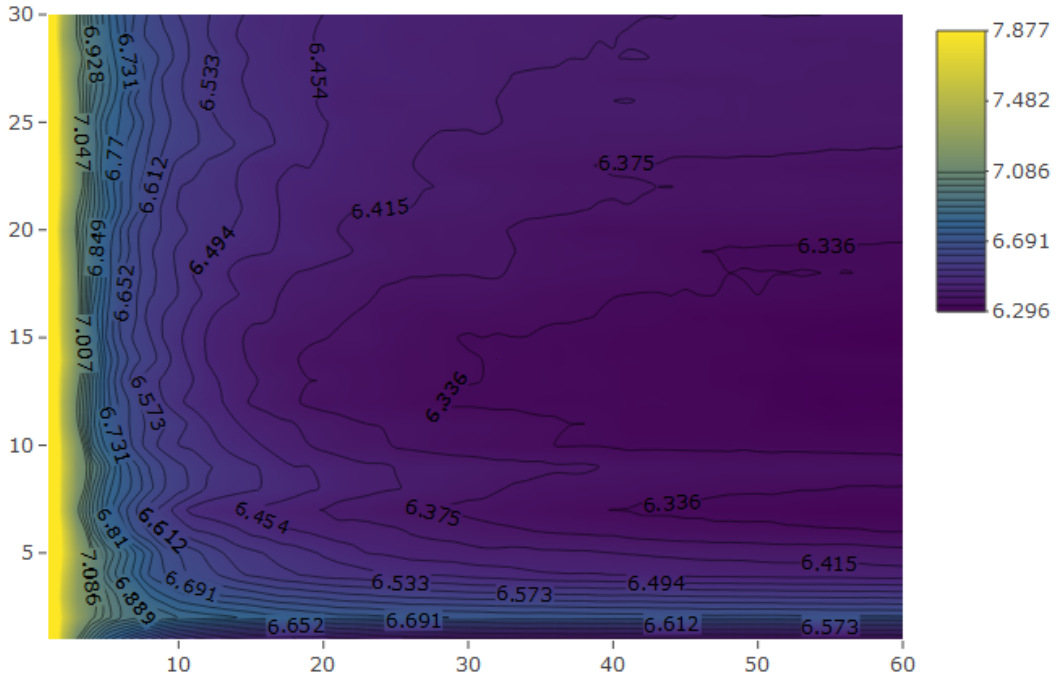

Figura 20: Gráfico con lineas de contorno concetradas

6.2 Integración de gráficas en interfaz *(plotly + shiny)*

Una vez creadas las dos gráficas principales, había que introducirlas en algún tipo de interfaz. Dada la naturaleza de plotly que utiliza tecnologías web, un paso lógico es utilizar algún tipo de interfaz orientada a navegador web. Para este propósito decidimos utilizar `Shiny`, un paquete que proporciona un framework para aplicaciones web. Nos decidimos por él ya que tiene una estructura que guarda ciertas similitudes con la de las páginas HTML y sobre todo porque `Plotly` en R está pensado para ser utilizado junto con `Shiny`.

Junto a las gráficas, `Shiny` permite una gran variedad de los elementos típicos de HTML como pueden ser pestañas, botones o tablas.

6.2.1 Pestaña de predicción

El objetivo de esta pestaña es mostrar la predicción hecha por la combinación óptima obtenida de cara a poder compararla con otras predicciones. Para esto se utiliza la gráfica combinada de la serie temporal y los errores en cada instante temporal. Como añadido a la gráfica base de la combinación óptima, se muestra por defecto la predicción *Ingenua* (*Naive*) como benchmark básico. Además, para que el usuario tenga una referencia de sobre qué parte de la serie se ha obtenido la optimización, se incluyen dos líneas verticales que indican los periodos de train y test (ver figura 21).

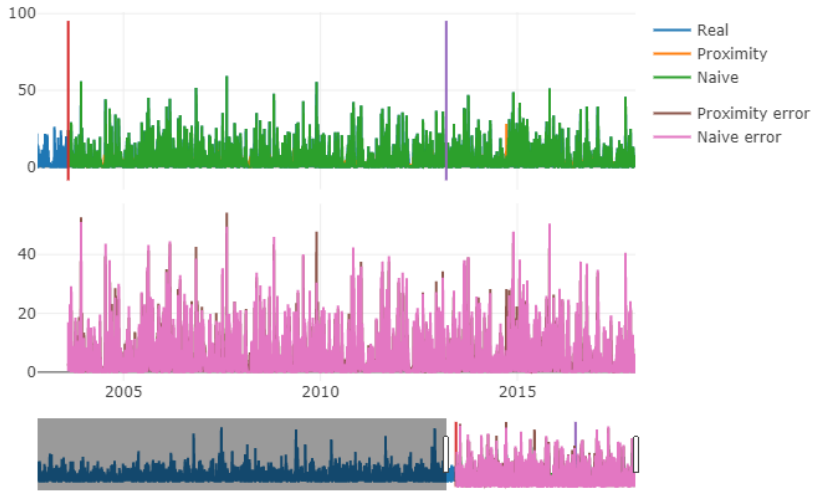

Figura 21: Gráfica base de la pestaña

Para poder comparar el método del k-NN con otros, se permite al usuario subir sus propias predicciones. El principal objetivo de esta funcionalidad es que un usuario nuevo en este tipo de predicción pueba comprobar fácilmente cómo de bien funcionan predicciones hechas con otros métodos en comparación a este (figura 22).

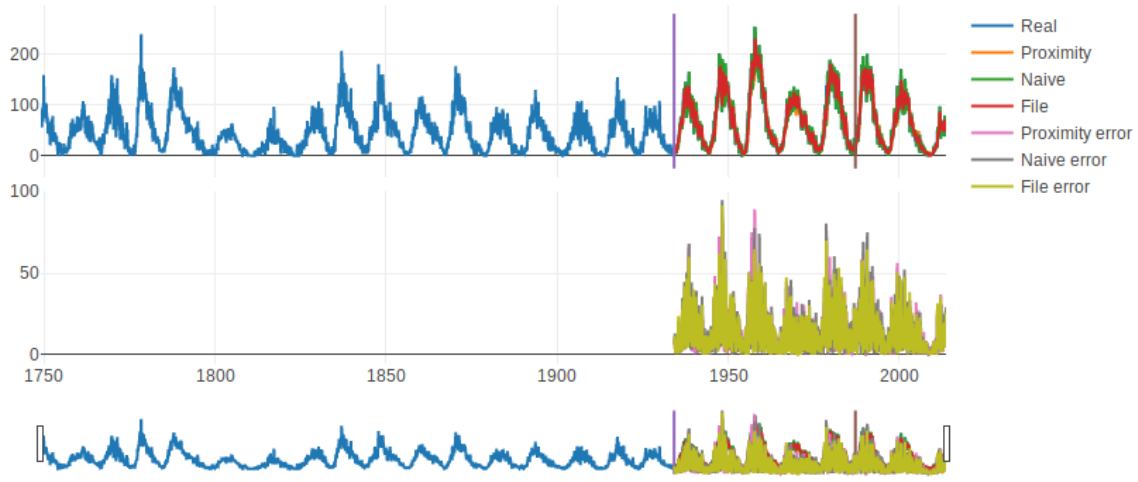

Figura 22: Gráfica con una serie subida por fichero

Además, de cara a poder tener una comparación más formal del grado de calidad que presentan las distintas predicciones, se decidió añadir una tabla como la que se puede ver en la figura 23. En ella se muestran algunos de las medidas de error más comunes en series temporales.

	Name	train ME	train RMSE	train MAE	test ME	test RMSE	test MAE
1	Proximity	2.57	19.72	14.49	1.06	17.21	12.56
2	Naive	0.02	19.41	14.3	0.27	22.01	16.07
3	File	0.02	18.43	13.55	0.4	19.68	14.21

Figura 23: Tabla de errores de distintas predicciones

6.2.2 Pestaña de optimización

En esta pestaña el objetivo es que el usuario tenga la información relativa al proceso de optimización realizado sobre una determinada serie. Como funcionalidad añadida, se busca que el usuario pueda probar las predicciones con combinaciones de k y d que no sean la óptima. Esto de cara a poder encontrar esas posibles soluciones que, como ya se ha comentado anteriormente, funcionen mejor para unas etapas concretas de la serie aunque no para toda ella en global.

Para lograr esta funcionalidad, lo que se hizo fue utilizar el sistema de captura de clicks de plotly. Esto permitió ser capaces de leer qué punto del mapa de calor había seleccionado el usuario y mostrar en una gráfica la predicción correspondiente a la combinación escogida (figura 24).

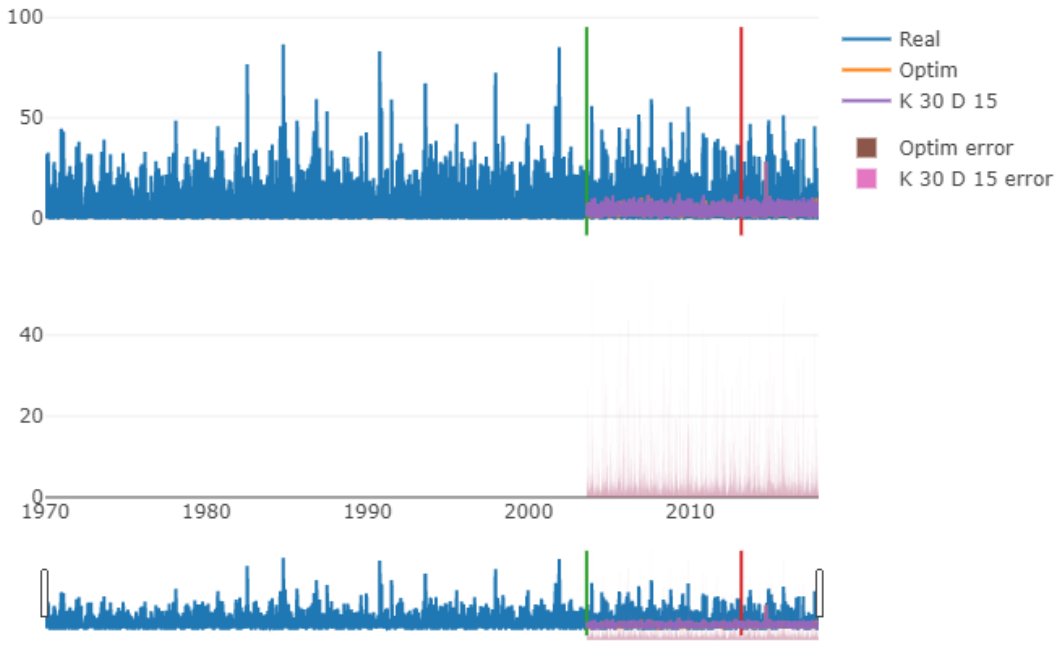

Figura 24: Predicción hecha al seleccionar en el mapa de contorno

De cara a no sobrecargar el gráfico con demasiadas series, aunque se pueda elegir cuáles muestra plotly y cuáles no, decidimos que era mejor poder ser capaces de eliminar de la misma manera que se habían añadido las predicciones.

Por último, con el objetivo de darle un mejor feedback visual al usuario, decidimos hacer unas marcas con puntos en los lugares de la gráfica seleccionados por el usuario. De la misma manera. Asimismo, por defecto se marcan los 5 puntos que indiquen las combinaciones que mejor minimicen el error, siendo la óptima de un color diferente. En la figura 25 podemos ver los cinco mejores valores a la derecha, siendo el mejor el de color verde, y a la izquierda un valor seleccionado a mano.

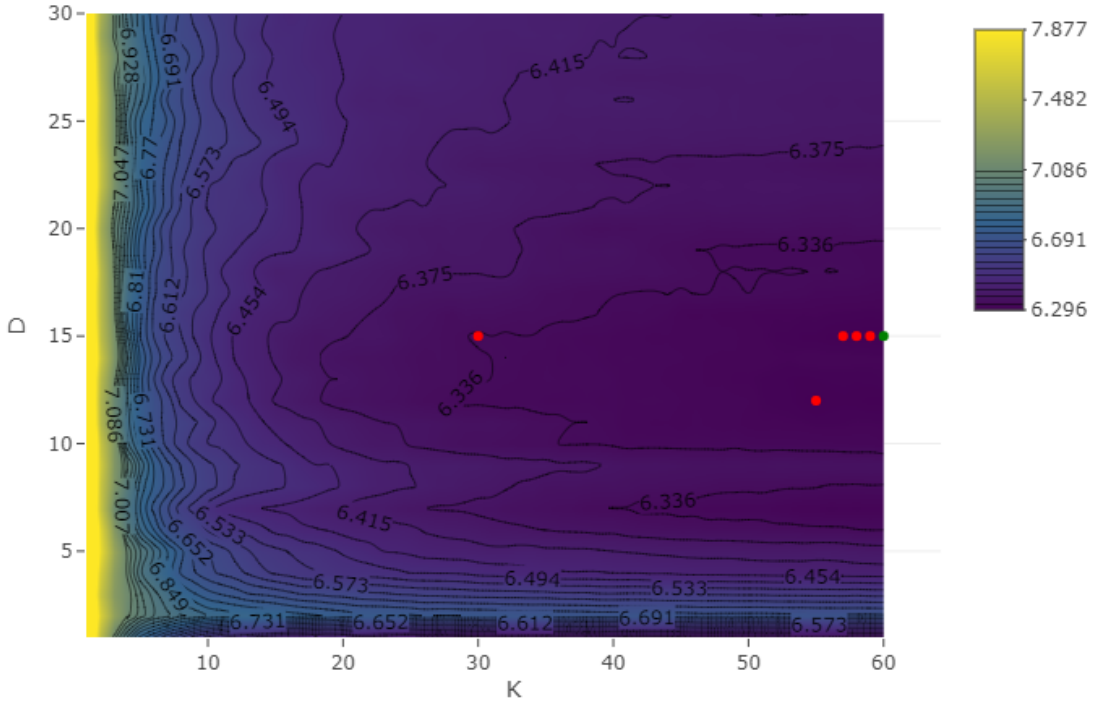

Figura 25: Gráfico con los 5 mejores (derecha) y 1 seleccionado a mano (izquierda)

7 Trabajo futuro

Acompañando la familiarización con los fundamentos teóricos de la predicción de series temporales y la interiorización del método del k-NN, fueron surgiendo ideas de diversas funcionalidades que se podrían incorporar al conjunto de herramientas elaboradas. A continuación se proponen y explican algunas de esas ideas:

- Aumentar el horizonte temporal a predecir: consiste en añadir la capacidad de predecir varios instantes futuros de una serie. Para ello habría que agregar un parámetro h que definiría los instantes futuros a predecir. Esto supondría adaptar todas las funciones a este cambio.

- Asignación de pesos según la cercanía temporal: a cada uno de los k vecinos se le asignaría un peso proporcional a su distancia temporal con el elemento identificado por el instante n de una serie temporal de tamaño n. Esto permitiría recoger mejor las series que tienen un comportamiento con tendencia, ya que los vecinos que estén lejanos en el pasado tendrán menor peso que aquellos muy recientes.

- Visualización de los k vecinos: a la interfaz gráfica desarrollada se agregaría la posibilidad de observar los k vecinos más cercanos a cada elemento de la serie temporal. Esto con el objetivo de poder interpretar de manera más clara los resultados y parámetros del método del k-NN. Lo expuesto se realizaría remarcando graficamente de distinto color cada uno de los k vecinos al instante temporal seleccionado.

- Implementación del algoritmo en un lenguaje de programación compilado: un lenguaje como C++ proporcionaría mayor control sobre el uso de memoria e hilos de procesamiento pudiendose lograr una implementación más eficiente del algoritmo.

- Extender la manera de conformar elementos: en la implementación cada elemento esta constituido por d instantes temporales, todos ellos consecutivos. Con ésta nueva funcionalidad se permitiría conformar cada elemento por d instantes no necesariamente consecutivos. Esto quiere decir que dado un vector de retrasos $\{l_1, l_2, \cdots, l_i, l_{i+1}, \cdots, l_d\}$, cada elemento determinado por un instante t quedaría constituido por los instantes temporales $\{t - l_1, t - l_2, \cdots, t - l_i, t - l_{i+1}, \cdots, t - l_d\}$.

- Implementación de una forma de completar series temporales incompletas: en ocasiones ciertas series temporales pueden tener instantes con valores que no están presentes. El objetivo sería encontrar distintas formas de completar dichos datos en función de la información que se posea sobre la serie.

- Ampliar las matrices de distancias en fichero: una vez que se ha guardado en disco la matriz de distancias para una determinada d, implementar una forma de que si se vuelve a llamar a la función de optimización, no recalcule las matrices que ya tiene guardadas. Esto permitiría que si se tienen calculadas las matrices para cuarenta *d's* diferentes, y se quieren añadir diez más, no haga falta calcular cincuenta sino únicamente las nuevas. De manera análoga, si en llamadas sucesivas de la función se usan *d's* ya calculadas, se omitiría su cálculo.

8 Contribuciones al proyecto

Dadas nuestras experiencias en trabajos en grupo a lo largo de la carrera, como tenemos una amistad que no queríamos perder por rencillas con el TFG, decidimos desde el principio trabajar juntos en todas las partes durante el proceso completo. Como consecuencia, en esta parte de la memoria intentaremos comentar simplemente algunas partes del trabajo o algunas ideas que son más atribuibles a alguno de los dos, pero no por ello signifca que el otro no trabajase también en ello. Si se quiere un registro más detallado de actividad, se puede ver el historial de commits de GitHub.

8.1 Daniel Bastarrica Lacalle

En un principio para el cálculo de las matrices de distancias, al invocar a `knn_elements`, se generaba la matriz de la máxima d que se quisiese probar. Como último valor se concatenaba el valor siguiente a cada elemento formado. Observé con el monitor de recursos del sistema la cantidad de memoria que se utilizaba, y llegamos a la conclusión de que R siempre hacía copias efectivas. Por este motivo descartamos esta única matriz y decidimos hacer tantas como $d's$ se estuviesen probando, así como eliminar la columna del valor futuro.

Cuando quisimos paralelizar, no sabíamos cómo se podían hacer ese tipo de cosas en R. Me puse a investigar distintos paquetes y formas de hacerlo. Busqué cuáles recomendaba la gente, cuál se adaptaba mejor a nuestro código y me decidí por el que finalmente acabamos utilizando.

Asimismo, hice la parte de dividir la información que necesitábamos procesar entre los distintos hilos. Busqué maneras de paralelizar por varias variables haciendo las combinaciones de todas ellas. Finalmente encontré la forma de concatenar los `foreach` y de ejecutarlos en paralelo.

Por otro lado busqué cómo se utilizan los *clusters* de *threads* en R, cómo afectan a la paralelización y cómo se tratan según el Sistema Operativo. Puse la sección de código paralelizada dentro del método de ejecución con el *cluster* creado y luego la vuelta al modelo de ejecución secuencial. Además descubrí que hay ciertos problemas con *Windows*, debido a la manera que tiene de crear hilos hijos respecto a sistemas basados en *kernel Linux*.

Este es el motivo por el cual tiene un rendimiento ligeramente inferior en *Windows*.

Una vez que teníamos la paralelización hecha por los distintos parámetros, pensé en qué fragmentos de los datos eran reutilizables dentro de un mismo hilo y cuáles no tenía sentido dividir. Al final decidimos que todas las posibles $k's$ siempre se procesasen juntas, y que lo que tenía más sentido era paralelizar por instantes temporales y $d's$.

Una vez estaban hechas las paralelizaciones de nuevo acudí al monitor de recursos para ver cuánta memoria consumía cada hilo de ejecución. Otra vez comprobé cómo los hilos hijos consumían casi tanta memoria como el padre, por lo que me puse a buscar maneras de evitar que esto pasase. Intenté buscar si existen punteros o referencias, pero dado el ámbito de trabajo en el que se suele utilizar R, como muchas veces no se usa por programadores sino por estadísticos, no tiene implementados ese tipo de elementos. En su lugar, se permite hacer código en lenguaje C al que se invoque desde funciones en R.

A la hora de empezar con la interfaz, llegamos al mismo problema de antes, no teníamos ningún conocimiento en la generación de gráficas en R. Basándome en mis ligeros conocimientos del lenguaje Octave, busqué similitudes para empezar a trabajar. Luego encontré el paquete *Plotly*.

Dada la gran cantidad de gráficos y parámetros de personalización que permite dicho paquete, hubo que investigar cuáles nos resultaban útiles. Probé varias gráficas distintas, busqué parámetros de personalización como los Sliders, el uso de gráficas 3D (aunque tuviésemos que descartarlas finalmente), el truncamiento de valores, las líneas de contorno, las leyendas o combinación de gráficas, por mencionar algunas.

Una vez estuvieron hechas las distintas gráficas, hubo que pasar de generar gráficos sueltos a integrarlos en una interfaz. Me puse a investigar cómo incluir gráficos de *Plotly* en una interfaz, y encontré para ello el paquete de *Shiny*. Busqué la teoría de cómo se organiza la información entre el lado de la interfaz y la del servidor, así como del paso de información de uno a otro.

8.2 Javier Berdecio Trigueros

Una vez las matrices de elementos ya se calculaban de manera independiente en lugar de generar una unificada, nos dimos cuenta de que a veces resultaba algo complicada de entender y trabajar con ella. Por este motivo, me puse a reorganizarla de una manera que la estructura fuese siempre más constante y sencilla de utilizar.

También respecto a las matrices, en un principio la matriz de distancias que devolvía la función `parDist` la convertíamos en una matriz normal en lugar de un vector indexado en forma de matriz triangular. De nuevo nos dimos cuenta de que R estaba utilizando el doble de memoria debido a esto, por lo que pasé el código de la versión convertida a acceder a los índices correspondientes del vector.

Durante el proceso de paralelización, hubo que encontrar una manera cómoda y entendible de guardar y combinar los datos generados por cada hilo. Para ello decidí combinar por columnas y generar la matriz de errores que devuelve la optimización.

Cuando estuvo hecha la primera versión de la parelización, con filas y $d's$, pensé en utilizar mejor los hilos de ejecución en vez de generar muchos. Mi planteamiento fue que en lugar de crear muchos hilos relativamente pequeños, era mejor crear una menor cantidad de ellos y hacer que realicen más cálculos. Esta solución ayudó especialmente a la eficiencia en *Windows*, ya que tiene que generar menos hilos, que es en lo que tiene un peor desempeño.

En cuanto a la paralelización mediante ficheros, me encargué de buscar las distintas formas de guardar y cargar datos en disco. Busqué distintos métodos y finalmente me decanté por el que acabamos utilizando. Además una vez hicimos la segunda versión en la que se guardan sólo fragmentos de matrices, me encargué de adaptar el código para que cada hilo supiese qué archivo ha de cargar.

A la hora de permitir utilizar sus propias predicciones al usuario en la pestaña correspondiente de la interfaz, había que buscar una manera cómoda de permitir al usuario añadir la serie temporal al gráfico. Para ello busqué cómo se hace esto en el paquete *Shiny* y en qué formato tienen que estar, así como de actualizar la tabla de errores.

Asimismo, a la hora de poder tener un gráfico interactivo en la pestaña de optimización de la interfaz, mi compañero se encargó de buscar algunas referencias sobre el manejo de la interacción con los gráficos en *Plotly*, y yo me encargué de aplicar el tratamiento de los clicks en la gráfica, así como de poder modificarlas dinámicamente en ejecución, en lugar de tener que volver a generar las gráficas.

Transversal a todo el desarrollo de las funciones, como uno de los objetivos de este trabajo era poder generar un paquete en R publicable, había que buscar qué tipo de requisitos y formatos se exigen para poder publicar un nuevo paquete. Busqué bibliografía al respecto y me encargué de explicarle a mi compañero qué tipo de cambios hay que realizar, así como adaptar mejor los comentarios que ya teníamos hechos en el código en ese momento.

Por último, también relacionado con la usabilidad de nuestro código, me encargué de pulir mejor los tipos de comentarios que hacíamos en las distintas, sobre todo explayándome más e intentando hacer más comprensible el código.

9 Conclusiones

Por último, una vez concluído el trabajo y la presente memoria, se analizarán los resultados obtenidos respecto a los objetivos iniciales.

En primer lugar, el objetivo principal era el desarrollo de una herramienta para la predicción mediante el algoritmo del k-NN. Este objetivo está cumplido, ya que se han desarrollado tanto los métodos de predicción como el de optimización. Todos presentan unos parámetros fáciles de entender y devuelven el resultado de manera sencilla de utilizar.

Por otro lado, el segundo gran objetivo de este trabajo era permitir que el k-NN deje de ser un método de predicción de tipo "caja negra". Consideramos que este objetivo también se ha cumplido, ya que la interfaz facilita la interpretación y el análisis de las predicciones del k-NN, así como del proceso de optimización.

Asimismo, se había decidido optimizar el código una vez realizada una primera versión. Creemos que este objetivo está cumplido de una manera original, ya que la idea de usar ficheros como forma de optimizar la paralelización se ha demostrado que sorprende a quien se lo hemos comentado. Tanto compañeros como profesores se han mostrado intrigados, dado que resulta una idea contraintuiva pensar que acudir a disco puede ser más rápido que usar memoria RAM. Esto entra en conflicto completamente con las ideas que cualquier persona con nociones en informática tiene, ya que se sabe que existen jerarquías de memoria con diferentes capacidades y velocidades.

También cabe comentar que como objetivo transversal se ha conseguido ver que el k-NN es un método más que capaz para algunas series, siendo un algoritmo completamente válido como buen predictor y no sólo como benchmark. Esto pone de manifiesto que no siempre hace falta acudir a sistemas de predicción más complejos, como las redes neuronales, para hacer buenas predicciones.

Por último, y más importante que lo anterior, este trabajo ha servido para desarrollarnos profesionalmente. Nos ha permitido afianzar conceptos aprendidos durante la carrera, ser capaces de demostrar que hemos adquirido cierta capacidad de análisis y enfrentarnos a problemas trabajando como equipo de manera independiente.

10 Conclusions

Once the work was concluded, we analyzed the achieved results in terms of our initial objectives.

First, the main objective was the development of a tool to predict values using the k-NN algorithm. This goal was accomplished as the methods for prediction and optimization have been developed. All of them are easily understandable parameters and their output value can be used in a simple way.

On the other hand, the second main objective of this work was to allow the k-NN to be interpreted more easily so it could be viewed more as a white box. We consider this objective accomplished as the graphical user interface facilitates the interpretation and analysis of the predictions made by the k-NN and the optimization process.

In this way, it had been decided to optimize the code once a first version of this algorithm was fully developed. We believe that this objective is accomplished in an original manner given that the idea of using files as a way to optimize the parallelization has been shown to surprise anyone to whom we have exposed it to. Both students and professors were intrigued, as it is contraintuitive to think that using secondary memory can be faster than to use RAM memory. This results in a conflict with the ideas of anyone in the computer engineer field. as it is known that there are memory hierarchies with different capacities and velocities.

Furthermore, it is worth to mention that as a transversal objective we have accomplished in viewing the k-NN is a method more than capable for predicting some time series, and therefore being a completely valid algorithm for forecasting and not only as a benchmark. This gives forwards the notion that it is not always needed to use more complex systems for forecasting (e.g.neural nets) to make good predictions.

Finally and more importantly, this work has helped as to develop ourselves professionally. It has allowed us to secure the concepts learned throughout the degree, and made us capable of demonstrating that we have acquired a certain capacity of analysis. Lastly, it has made us more capable of facing problems as an independent team.

References

[1] R. J. Hyndman and G. Athanasopoulos, "Forecasting data and methods," in *Forecasting: Principles and Practice*, 2013, ch. 1.4.

[2] ——, "Time series components," in *Forecasting: Principles and Practice*, 2013, ch. 6.1.

[3] ——, "Arima models," in *Forecasting: Principles and Practice*, 2013, ch. 8.

[4] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4(2):1883.

[5] J. Arroyo, "El método de k-nn," Ph.D. dissertation, Universidad Pontificia Comillas, 2008.

[6] ——, "El k-nn como método de predicción de series temporales," Ph.D. dissertation, Universidad Pontificia Comillas, 2008.