Predicción de series temporales mediante el método k-NN: *explicabilidad* y algoritmos de ensamblado
Time series forecasting by the k-NN method: explainability and ensemble algorithms

Trabajo de Fin de Máster
Curso 2019–2020

Autor
Daniel F. Bastarrica Lacalle

Director
Javier Arroyo Gallardo

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Predicción de series temporales mediante el método k-NN: *explicabilidad* y algoritmos de ensamblado

Time series forecasting by the k-NN method: explainability and ensemble algorithms

Trabajo de Fin de Máster en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Daniel F. Bastarrica Lacalle

Director
Javier Arroyo Gallardo

Convocatoria: *Junio/Julio 2020*
Calificación: *Sobresaliente 9*

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

23 de julio de 2020

Agradecimientos

En primer lugar al principal culpable de que este trabajo haya ocurrido. Javier: profesor, tutor, compañero, jefe, tutor, confidente... Está claro que nunca me hubiese visto haciendo "predicción de series temporales" y fue por tu buen hacer en Bases de Datos que hemos llegado aquí. Está claro que eso ha marcado mi futuro, de formas que quizás ahora ni llegue a imaginar. Sólo el tiempo lo dirá. Tal vez a esa lista de roles llegamos a añadir Director de Tesis. Nunca digas nunca...

En segundo lugar a los amigos y compañeros de clase. Sin vosotros tantas horas de clase y prácticas y tantas asignaturas, hubiesen sido demasiado arduas.

Es obligado también agradecer a mi familia. Es innegable que sin su apoyo, no sólo económico sino también moral, no habría llegado a cerrar esta etapa que aquí concluye (al menos de momento). Quién iba a decir que ese chico que suspendía para septiembre llegaría hasta aquí sin repetir ni 1 más.

Y por último, pero ni por asomo menos importante, a Eli. Gracias por tanto apoyo, ánimo, cariño, cuidados, risas y momentos de felicidad en general. Pero al igual que hay luces, también hay sombras. Agradezco también todos esos momentos amargos, ya que aunque lo pasemos mal en el momento considero que nos hacen más fuertes. Gracias por el apoyo emocional en los momentos más duros del TFM, así como ocurrió con el TFG. Este TFM tiene un poquito de tí.

Dedicatoria

En primer lugar, dada la situación en la que se han desarrollado los últimos compases de este trabajo, me resulta obligado dedicarle este trabajo a los que más duramente han trabajado. Gracias a todo el personal sanitario por jugaros la vida por desconocidos. Espero que aprendamos la moraleja del cuento: menos circo y más ciencia.

Y la dedicatoria realmente planeada. A todo el profesorado de la escuela pública que se toma en serio su trabajo y lo hace con esfuerzo y cariño. Y especialmente a todos los profesores por cuyas manos he pasado. Este trabajo marca el fin de un ciclo, y sois todos vosotros los culpables, en mayor o menor medida, de que haya llegado hasta aquí.

El futuro de un país está en los más jóvenes.

Abstract

This paper deals with time series forecasting using a simple machine learning technique, the k-NN. It is based on a previous development and it is proposed to take it further in two aspects, improving the interpretability of the predictions generated by k-NN and improving its predictive capacity using ensemble algorithms, that is, combining k-NN predictors.

Regarding the interpretability of the models, k-NN, unlike other methods, is very transparent and therefore lends itself to the explanation of its predictions, although this is something for which there are no libraries, nor is it usually seen in academic works. Therefore, a web application is proposed to deepen the forecasts made by means of k-NN and its ensemble counterpart. This work seeks to explain the entire process of forecasting, from what data is used to predict, where more error is made and why, to provide information on the predictability of a series.

Regarding the ensemble algorithms, four alternatives are proposed. In all of them, sets of k-NN predictors will be generated whose predictions will be combined. This approach has been named Random Neighborhoods, since it is inspired by bagging and random forest methods.

Keywords

k-NN , time series, forecasting, lazy learning, ensemble, explainability, random neighborhoods

Resumen

Este trabajo aborda la predicción de series temporales utilizando una técnica sencilla de aprendizaje automático, el k-NN. Para ello se parte de un desarrollo previo y se plantea llevarlo más allá en dos aspectos, mejorando la interpretabilidad de las predicciones generadas por el k-NN y mejorando su capacidad predictiva utilizando algoritmos de ensamblado, es decir, combinando predictores de k-NN.

Respecto a la interpretabilidad de los modelos, el k-NN a diferencia de otros métodos es muy transparente y por ello se presta a la explicabilidad de sus predicciones, aunque es algo para lo que no existen librerías, ni se suele ver en trabajos académicos. Por ello se plantea realizar una aplicación web que permita ahondar en las predicciones realizas mediante el k-NN y su homólogo de ensamblado. En este trabajo se busca explicar todo el proceso de la predicción, desde qué datos se emplean para predecir, dónde se comete más error y por qué, hasta ofrecer información sobre la predictibilidad de una serie.

Respecto a los algoritmos de ensamblado, se proponen cuatro alternativas. En todas ellas se generarán conjuntos de predictores de k-NN cuyas predicciones se combinarán. A este enfoque se le ha denominado Random Neighborhoods, ya que está inspirado en los métodos de bagging y random forest.

Palabras clave

k-NN , series temporales, predicción, aprendizaje perezoso, ensamblado, interpretabilidad, vecindarios aleatorios

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

En el mundo actual, dado el alto grado de digitalización que se está alcanzando, cada vez hay una mayor cantidad de datos disponibles sobre los que trabajar para obtener conclusiones. Uno de los enfoques del tratamiento de datos es la predicción de nuevas observaciones, como pueden ser la meteorología enfocada en las precipitaciones o contaminación del aire, el abastecimiento ya sea de productos de primera necesidad o de servicios básicos del hogar, o el grado de propagación que puede tener una enfermedad de cara a preparar los recursos necesarios.

Dentro de la predicción de estos datos existe un campo que es el de las series temporales, en el cual se enfoca el presente trabajo. Por este motivo el primer paso es el de definir una serie temporal. En términos generales una serie temporal es un conjunto de datos relativos a una o varias métricas que se miden a lo largo del tiempo. Podemos por tanto dividir una serie temporal en dos partes:

- Instantes temporales: representan el momento en el tiempo en el que se hizo cada una de las mediciones. Ejemplos de ellos pueden ser la hora del día o la fecha del calendario. Se denotan como $\{t_1, t_2, \ldots, t_{n-1}, t_n\}$, siendo n el número de observaciones.
- Valores observados: representan los datos asociados a las diferentes métricas medidas. Se denotan como $\{y_1, y_2, \ldots, y_{n-1}, y_n\}$, y existirán tantas como diferentes métricas posea la serie temporal.

Se dice que una serie es *estacionaria* (Peña y Sánchez, 2007) cuando es estable a lo largo del tiempo, es decir, cuando sus valores oscilan alrededor de un nivel fijo y además posee una variabilidad constante. Si esto no se cumple, entonces la serie es *no estacionaria*. En dicho caso, algunas de estas

series podrían presentar una tendencia clara. Por otra parte, podría darse un caso particular de series no estacionarias en las que los valores oscilan alrededor de un valor central, aunque tienen un comportamiento distinto dependiendo del período de tiempo (años, meses, días). A este tipo de series se les denomina series *estacionales*.

La predicción de series temporales consiste en usar los valores pasados para obtener el siguiente valor de la serie o un valor futuro de la serie a un *horizonte temporal* determinado. Es decir, la distancia temporal que existe entre el instante actual y el que se desea predecir. Para realizar dicha predicción, como en cualquier campo de predicción o clasificación, existen múltiples técnicas y enfoques que se comentarán a continuación.

Un ejemplo de ellos son las técnicas de alisado (Brown, 1963), en las cuales se usa una media móvil de los últimos valores de la serie, realizando una ponderación donde los valores más recientes tienen mayor peso. Otro tipo de técnicas son las autorregresivas (Box, Jenkins, Reinsel y Ljung, 2011), que generan la predicción como una regresión lineal de valores pasados de la serie.

También existen aproximaciones de técnicas de *machine learning* muy sofisticadas, como son los perceptrones multicapa (Haykin, 1998). En este caso se utiliza la metodología autorregresiva, pero se mejora con la flexibilidad del perceptrón para detectar la falta de linealidad en la regresión.

Sin embargo, en muchos casos las técnicas de aprendizaje automático (*machine learning*) tienden a presentar un comportamiento de tipo *caja negra*. Esto es, los métodos reciben una entrada de datos y generan una determinada salida, pero no es posible saber por qué se ha generado dicha salida ni cómo se ha obtenido. Por este motivo se investiga de forma muy activa cómo hacer que las técnicas de aprendizaje automático sean explicables. Se ahondará al respecto en el siguiente capítulo.

Dentro de las técnicas de aprendizaje automático se encuentra el método de los *k-vecinos más cercanos* (en adelante *k-NN*, por sus siglas en inglés). Esta técnica consiste en comparar el dato que se quiere predecir con todos los demás. De todos ellos se escogerán los k que guarden una mayor semejanza. Por último, la predicción se realizará en función del valor que tomó la variable a predecir en cada uno de dichos vecinos.

Concretamente esto fue realizado durante el Trabajo de Fin de Grado (Bastarrica y Berdecio, 2018). En él se lleva a cabo una implementación de la predicción de series temporales mediante *k-NN*. Además se realiza el proceso de entrenamiento para seleccionar el modelo óptimo. Por último se desarrolló una aplicación web básica para explotar la explicabilidad de las predicciones y el entrenamiento. Según la implementación desarrollada, dado un determinado instante temporal que se desea predecir, dicha predicción se compone de dos parámetros principales:

- ***d***: se agruparán d instantes temporales consecutivos, de forma que su último instante sea aquel con el que se desea predecir. Dicha agrupación formará un *elemento*. Sea y_t un instante temporal, su elemento correspondiente de longitud d sería: $\{y_{t-d+1}, \ldots, y_{t-1}, y_t\}$. Serán dichos elementos, y no los instantes temporales discretos, los empleados por el *k-NN* para realizar los cálculos de la búsqueda de vecinos.

- ***k***: se calcularán las distancias existentes entre el *elemento* conformado por el instante mediante el que se desea predecir, y el resto de elementos anteriores al mismo en la serie. Mediante estas distancias, se elegirán los k elementos con menor distancia, y se realizará la predicción con los mismos.

Estos dos parámetros serán los únicos que caracterizarán un determinado modelo de predicción del *k-NN*. Es debido a que, aunque resulte contraintuitivo, el *k-NN* es un método de aprendizaje que no aprende mediante un modelo de predicción. Es decir, el *k-NN* no genera un modelo mediante entrenamiento, como ocurre por ejemplo con las diferentes neuronas de una red neuronal, sino que en su lugar necesita cargar en memoria todos los *elementos* para luego seleccionar los k mejores.

El *k-NN* como técnica de aprendizaje es un método que a pesar de su sencillez, tiene un buen rendimiento, además de prestarse a la *explicabilidad* y la transparencia. Esto se debe principalmente a que siempre se pueden identificar cuáles son los k-vecinos seleccionados para realizar la predicción. Además se puede comprobar si todos ellos fueron realmente parecidos al actual y si el valor a predecir que tuvieron resulta útil para la predicción.

Respecto a su transparencia, el método del *k-NN* tanto como en problemas de regresión, como de clasificación, no suele desarrollarse con estas características presentes. Es decir, aunque la posibilidad de inspeccionar cómo se ha llevado a cabo la predicción existe, raramente se explota dicha característica. En general los resultados se muestran como una media de error agregada y no se aporta mayor información al respecto.

Por ello, en este trabajo se plantea abordar este apartado desde varios puntos de vista. Por un lado, conocer y estudiar los k-vecinos que generan una predicción y qué características poseen. Por otro lado, estudiar por qué ciertos valores de k y d funcionan mejor que otros, y en particular si esto se cumple para un determinado comportamiento de la serie.

Cabe destacar que gracias a esa información relativa a las predicciones, se planteó la posibilidad de que ciertas combinaciones del *k-NN* tuviesen un buen desempeño cuando la serie presenta un comportamiento u otro. Por ello se planteó la hipótesis de utilizar varias predicciones de forma simultánea. De esta forma el resultado sería una conjunción de diferentes predicciones, cada una especializada en unos determinados comportamientos.

Esto último, combinado con la sencillez del *k-NN*, es una cualidad que permite plantear la combinación de predicciones. Para llevar a cabo este método, se realizan un conjunto de predicciones de los mismos datos con diferentes métodos, y una vez se tienen todas ellas se combinan para dar lugar a una única predicción. Este enfoque busca capturar diferentes aspectos de la serie temporal con cada método, y lograr una predicción que contenga todos ellos.

Este enfoque de la combinación de predicciones, aplicado al *k-NN* para series temporales, es algo que no se ha explorado hasta ahora en la literatura encontrada que versa al respecto.

Como consecuencia de todo lo anteriormente expuesto, el presente Trabajo de Fin de Máster se plantea intentar modernizar la técnica de *k-NN* para predicción de series temporales. Como objetivos se postula en primer lugar mejorar la capacidad de predicción del método mediante la combinación de predicciones. Como ya se ha comentado anteriormente este enfoque no presenta comparaciones directas, ya que no hay trabajos existentes al respecto. En segundo lugar se abordará el aumentar las propiedades de transparencia y *explicabilidad*, incluidas las de la combinación de predicciones. Para esto último se realizará una mejora y ampliación de la aplicación web básica ya existente, que mejore la *explicabilidad* y transparencia buscadas.

1.2. Objetivos

Por todo lo anteriormente expuesto, se proponen los siguientes objetivos para el presente trabajo:

- Desarrollar una estrategia de *boosting* para *k-NN* en series temporales. Para ello se llevará a cabo la implementación de la propuesta de predicción mediante combinación de predictores.
- Mejorar la eficiencia del *boosting*. Una vez desarrollada la estrategia, se propone mejorar la eficiencia del método desarrollado.
- Aumentar la *explicabilidad* de las predicciones mediante *k-NN*. Añadir métricas relativas a la predicción y visualización de los k-vecinos escogidos.
- Ampliar la funcionalidad de la aplicación web para incluir la predicción mediante combinación. Se busca integrar visualmente las diferentes componentes explicables, así como permitir al usuario entender los diferentes métodos de predicción mediante su uso.
- Rediseño de la aplicación: mejorar la experiencia de usuario, tanto de la facilidad de uso como de sus tiempos de respuesta.

1.3. Plan de trabajo

Para la realización del proyecto se seguirán los procesos descritos a continuación. Respecto a la mejora de las predicciones mediante combinación:

- Planteamiento e implementación del algoritmo de predicción mediante combinación. Estudiar el reaprovechamiento del código ya existente.
- Implementación de la predicción mediante los predictores generados por los algoritmos anteriores.
- Análisis de rendimiento del algoritmo de predicción mediante combinaciones medido en series temporales reales.
- Estudio de nuevas alternativas de algoritmos de generación de predictores.

Respecto a la aplicación web:

- Mejora de la experiencia de usuario: plantear una alternativa que evite la actualización automática de la aplicación web al modificar cualquier valor de la misma.
- Mejora de la experiencia de usuario: estudiar e implementar alternativas para la reutilización de datos empleados por varios elementos de la aplicación web.
- Integración en la aplicación de nuevas métricas de las predicciones, así como una gráfica para la visualización los k-vecinos escogidos.
- Integración del algoritmo de predicción mediante combinación.

Capítulo 2

Estado del arte

2.1. Predicción mediante ensamblado

Dadas las limitaciones existentes en el método desarrollado, y a raíz de los comportamientos observados en la aplicación web, se plantea como hipótesis la mejora de la capacidad de predicción mediante el ensamblado (*ensemble learning*). Esta técnica de *machine learning* consiste en la utilización de diferentes métodos de predicción de manera que, si se combinan, produzcan un mejor resultado que cualquiera de ellos de forma individual (Dietterich et al., 2002; Zhou, 2009; Polikar, 2012; Stock y Watson, 2006). Este tipo de técnicas llevan en uso desde hace varios años, y se ha comprobado su mejora de rendimiento en numerosas ocasiones (Koren, 2009).

Dentro de estas técnicas, existen las que en lugar de combinar diferentes métodos, realizan el ensamblado sobre un mismo tipo de predictor. Algunas de las técnicas dentro de este ámbito se basan en la generación de estructuras arborescentes para los modelos, como pueden ser el *bagging* (*bootstrap aggregating*), *random forest* y *gradient boosting*.

El *bagging* (Breiman, 1996) es una técnica consistente en generar subconjuntos de datos a partir de los originales. Con cada uno de estos conjuntos se entrena un determinado modelo. El resultado final se obtiene como un promedio del resultado de todos los modelos.

Por otro lado, como una variación del *bagging* están los *random forest* (Breiman, 2001). La diferencia en este caso reside en la división no sólo del conjunto de datos, sino también de las variables a emplear. Es decir, al generar cada nodo del modelo, no sólo se emplea la parte correspondiente del conjunto total, sino que además se emplean únicamente un subconjunto de las variables observadas.

Otra variación del *bagging* es el *gradient boosting* (Friedman, 2001). En este caso en lugar de dividir los datos en subconjuntos y generar modelos con cada uno, se sigue un proceso iterativo de generación de subconjuntos.

Concretamente, cada vez que se genera uno de los modelos, se prueba la eficacia de la combinación de todos los modelos sobre el conjunto de datos. Cuando se genera el siguiente subconjunto de datos para el siguiente modelo, la probabilidad de escoger cada dato será mayor si se realizó una mala predicción del mismo. Una implementación popular actualmente es el *XGBoost* (Chen y Guestrin, 2016).

A continuación se citan algunos ejemplos de ensamblado con *k-NN*. Existen implementaciones de ensamblado de *k-NN* para regresión de datos con distribuciones *Tweedie* (Farrelly, 2017), donde una de las conclusiones es que variar la k proporciona buenos resultados.

Dentro de este conjunto de técnicas existen las que afectan al presente trabajo, las aplicadas a la predicción de series temporales (Timmermann, 2006). Para su realización se crea un conjunto de métodos entrenados individualmente, y la predicción se lleva a cabo mediante el cálculo de la media ponderada de la predicción realizada por cada método. A su vez, dentro de dichos conjuntos de predictores, se plantean discrepancias al respecto de cuáles de ellos emplear y cuánto afecta (peso en la media ponderada) cada predicción individual al resultado final (Kourentzes, Barrow y Petropoulos, 2019). Algunas implementaciones recientes proponen crear meta-modelos que sean los encargados de asignar el peso de cada método en la combinación resultante (Montero-Manso, Athanasopoulos, Hyndman y Talagala, 2020).

Para la implementación se plantea como nombre *Random Neighborhoods* como referencia al método de *Random Forest*, entendiendo que en lugar de agrupar *árboles* se agrupan *vecinos*.

Se plantea hacer la división de los datos de entrenamiento siguiendo la metodología de *bagging*. De esta forma se crearán conjuntos de predictores *k-NN*, con sus respectivos parámetros de k y d, mediante el reparto de observaciones. Es decir, generar un conjunto finito de predictores que hayan sido entrenados sobre diferentes observaciones de una serie temporal.

Sin embargo se plantea una hipótesis diferente a las comentadas hasta ahora. En esta ocasión se usará como referencia los algoritmos genéticos (Holland et al., 1992; Goldberg, 1988; Pérez y Rückauer, 2002). Consiste en la creación de un conjunto de modelos (llamados *individuos*) que ofrecen una posible solución al problema. A base de combinar (cruce) y seleccionar individuos de forma iterativa, se acaba llegando a una posible solución final. Para realizar estas modificaciones se emplea una función de coste o *fitness*. Esto es, una función que indique el desempeño de cada individuo respecto al resultado deseado. Se han empleado las metodologías de asignación de *fitness* y selección para los *Random Neighborhoods* de los que versa este trabajo. Como resultado del entrenamiento, se obtendrán nuevamente conjuntos de predictores *k-NN*, con sus respectivos parámetros de k y d, que de forma conjunta puedan generar mejores predicciones que una única combinación.

2.2. Predicción explicable

Muchas de las decisiones que afectan a la sociedad se toman mediante algoritmos, y algunas de ellas se realizan sin supervisión. Un ejemplo de ello son los coches autónomos, donde las decisiones suelen ser tomadas por sistemas de tipo *caja negra*, donde no es posible saber cómo o por qué ha llegado el sistema a las conclusiones. Ejemplos de estos son los denominados Perceptrones (Stephen, 1990; Freund y Schapire, 1999) y los sistemas de *Deep Learning* (LeCun, Bengio y Hinton, 2015; Goodfellow, Bengio y Courville, 2016; Schmidhuber, 2015). En general estos sistemas suelen tener en común un modelo para predicción con un alto grado de complejidad. Gracias a esto suelen tener un gran desempeño, sin embargo traen como consecuencia una imposibilidad a la hora de ser interpretados.

Por estos motivos, existen iniciativas de desarrollo y proyectos destinados a evitar este tipo de modelos no auditables. Esta corriente se puede agrupar en lo que se denomina *Inteligencia Artificial eXplicable* o *XAI*, por sus siglas en inglés (Gunning, 2017; Adadi y Berrada, 2018; Arrieta, Díaz-Rodríguez, Del Ser, Bennetot, Tabik, Barbado, García, Gil-López, Molina, Benjamins et al., 2020). Con esto se busca, entre otras cosas, aportar confianza en los sistemas en los que se va a delegar la toma de decisiones. Sobre todo en aquellas que afectan directamente a la vida humana, como el citado caso de los coches autónomos.

Existen dos enfoques para lograr esto. Por un lado, se plantean algoritmos y desarrollos que buscan añadir métricas respecto a modelos ya existentes. Dos ejemplos de ello son *LIME*(Ribeiro, Singh y Guestrin, 2016), que consiste en añadir nuevos métodos de aprendizaje para explicar los ya existentes, y el uso de *Shapley values* (Lundberg, Erion, Chen, DeGrave, Prutkin, Nair, Katz, Himmelfarb, Bansal y Lee, 2020; Bracke, Datta, Jung y Sen, 2019), un conjunto de métricas provenientes de la teoría de juegos, que se miden sobre los modelos y permiten entender el funcionamiento interno de los mismos sin necesidad de reentrenar o cambiar la estructura del mismo.

El otro enfoque es el del desarrollo de técnicas de predicción que en su propio diseño y desarrollo tienen en cuenta que las características de explicabilidad estén presentes. Un ejemplo de ello serían las *Explainable Deep Neural Networks* (Assaf y Schumann, 2019). El planteamiento realizado es el de generar redes que sirvan tanto para predecir valores como para explicar cómo se obtienen los mismos.

Dentro de estas técnicas se puede englobar al *k-NN*, ya que dada su sencillez permite explicar el proceso de realización de predicciones. Por este motivo el enfoque buscado en el desarrollo de este trabajo está más relacionado con el segundo. Esto se debe a que aunque la técnica en sí del *k-NN* no es nueva, se vuelve más transparente sin alterar su esencia. Para ello se desarrollarán la herramienta y los estadísticos ya comentados con anterioridad.

2.3. Trabajo previo

Como ya se ha comentado, el presente trabajo parte de base con un desarrollo previo realizado durante el Trabajo de Fin de Grado (Bastarrica y Berdecio, 2018). Posteriormente se trabajó sobre ello como parte de una beca de investigación en la Facultad de Informática de la Universidad Complutense de Madrid. El desarrollo está alojado en el repositorio abierto de paquetes de `R` *CRAN*[1], así como en *GitHub*[2] A continuación se resume el estado final de funcionalidades disponibles, correspondiente a la versión realizada durante la beca.

2.3.1. *k-NN* para series temporales

Se desarrollaron principalmente dos funcionalidades. Por un lado está la generación de la propia predicción en sí misma. Esta se realiza mediante dos métodos, `knn_forecast` y `knn_past`. La primera calculará la predicción correspondiente al siguiente valor de la serie temporal proporcionada. La segunda función tiene como objetivo comprobar el rendimiento de una determinada combinación de k y d, sobre un intervalo de la serie. Para ello recibirá además de la propia serie, un índice correspondiente a un instante temporal de la misma. Este será el que marque el intervalo de serie sobre el que se quiere probar el rendimiento, que llegará hasta el final de la misma.

Por otro lado se ha creado la función `knn_param_search`, que es la encargada de realizar el entrenamiento para el *k-NN*. De forma similar a la función `knn_past`, recibirá un índice además de la propia serie temporal. Su labor consistirá en probar una serie de valores para k y d proporcionados, realizando predicciones con cada una de ellas para todos los intervalos de la serie temporal definidos. De esta forma, al tener la serie temporal y una determinada predicción, se puede medir el error cometido por la misma. Aquella combinación de k y d que tenga el menor error será la que se considere ganadora. Además de devolver la combinación, devolverá una matriz de errores donde cada posición se corresponda con una combinación de k y d probada. Esto permite estudiar qué otras combinaciones, aparte de la que haya minimizado el error, pueden resultar interesantes.

Cabe destacar que todas las funciones permiten definir la métrica de distancia empleada al buscar a los k-vecinos, el sistema de ponderación aplicado al valor siguiente de dichos vecinos para realizar la predicción y el tipo de error medido en las funciones de `knn_param_search` y `knn_past`. Además de esto, todas las funciones tienen implementado un sistema de paralelización basado en hilos concurrentes. Para ello cuentan con un parámetro que indicará la cantidad de estos hilos que se desea emplear de forma simultánea.

[1] En el siguiente enlace: `https://cran.r-project.org/package=knnp`
[2] Mediante el enlace: `https://github.com/grasia/knnp`

2.3.2. Explicabilidad del *k-NN*

Para cubrir el área de la explicabilidad, se desarrolló una aplicación web básica durante el Trabajo de Fin de Grado, que luego fue ampliada durante el período de beca [3]. Se ha desarrollado mediante el paquete *Shiny* de *R*. Concretamente la aplicación se compone de dos pestañas diferentes. En la primera de ellas se pueden probar combinaciones de k y d de forma manual. Para ello se muestran su predicción junto al error cometido en cada instante temporal, así como la propia serie temporal.

En la segunda pestaña el principal elemento es un mapa de calor, correspondiente a la matriz de errores generada por la función `knn_param_search`. Al hacer click en el mapa, se seleccionan diferentes combinaciones de k y d. Todas las seleccionadas se mostrarán en una gráfica similar a la de la primera pestaña. Sin embargo en este caso además de la propia serie, siempre se mostrará la predicción realizada por la combinación de k y d que minimice el error en *entrenamiento*. La peculiaridad será que todas las combinaciones de k y d seleccionadas en el mapa de calor, verán su correpondiente predicción reflejada en el gráfico de la serie temporal. Por último esta segunda pestaña posee una pequeña tabla resumen donde aparecen los errores de los diferentes partidos.

[3] Disponible en: `https://github.com/Grasia/Explainable-kNN`

Capítulo 3

Predicción de series temporales mediante *k-NN*

El *k-NN* es un método de aprendizaje automático basado en instancias (Aha, Kibler y Albert, 1991). Esto significa que las predicciones que genera provienen directamente de las propias instancias de entrenamiento sin estimar ningún modelo. Cabe destacar que generalmente esta información se almacena directamente en memoria, y que el proceso de predicción supone analizar todas las instancias conocidas. Por este motivo, el proceso de predicción puede ser costoso tanto en términos de memoria como de ejecución, ya que ambos crecen de forma lineal con el tamaño de los datos a analizar.

Por otro lado, como ya se ha comentado anteriormente, no se genera ningún modelo durante el proceso de entrenamiento. Esto se debe a que en lugar de aproximar una función por completo, se realiza una aproximación parcial de forma local. De esta forma, el proceso final de realización de los cálculos se delega al momento último de asignación del resultado. Este tipo de métodos se conocen como de *Aprendizaje perezoso* (Wettschereck, Aha y Mohri, 1997; Arroyo, 2008, cap. A.3).

3.1. Descripción del *k-NN* para series temporales

Resumido brevemente, el método del *k-NN* consiste en la búsqueda dentro del conjunto de datos de elementos similares al que se desea predecir usando una determinada medida de distancia. De entre esos elementos se escogen aquellos k elementos que guarden una mayor similitud con el elemento a predecir. Por último se realiza la predicción en función del valor siguiente de dichos elementos.

El *k-NN* es una técnica muy versátil que puede utilizarse para problemas de clasificación y de regresión, entre estos últimos se puede utilizar para la

predicción de series temporales, que son los que ocupan esta memoria. La particularización del *k-NN* para series temporales puede describirse de la siguiente forma (Arroyo, 2008, cap. A.3.1):

1. La serie temporal y, definida como $\{y_1, \ldots, y_{n-1}, y_n\}$ siendo n la longitud la misma, se transforma en *elementos* de longitud d. De esta forma, se obtiene una serie definida como $y_t^d = \{y_{t-d+1}, \ldots, y_{t-1}, y_t\}$.
2. Se calculan las distancias entre el elemento que se quiere predecir $y_n^d = \{y_{n-d+1}, \ldots, y_{n-1}, y_n\}$ y todos los anteriores a él de la serie.
3. Se ordenan los elementos según la distancia y se seleccionan los k más cercanos. Dichos elementos se denotan como $y_{t_1}^d, y_{t_2}^d, \ldots, y_{t_k}^d$.
4. Se obtienen los valores siguientes a cada uno de los k seleccionados, y se calcula la predicción como la media ponderada de dichos valores
$$\hat{y}_{n+1} = \frac{\sum_{i=1}^{k} w_i \cdot y_{t_i}^d}{\sum_{i=1}^{k} w_i}$$
siendo $\hat{y}_j$ la predicción del *j-ésimo* instante temporal y w_i el peso asociado, en la media ponderada, al valor siguiente del *i-ésimo* vecino.

3.1.1. Parametrización del *k-NN*

A continuación se presentan los diferentes parámetros definibles para la predicción mediante el *k-NN*:

Función	**Fórmula**
Canberra	$\sum_{i=1}^{d} \frac{\lvert p_i - q_i \rvert}{\lvert p_i \rvert + \lvert q_i \rvert}$
Euclídea	$\sqrt{\sum_{i=1}^{d} (q_i - p_i)^2}$
Manhattan	$\sum_{i=1}^{d} \lvert p_i - q_i \rvert$
Whittaker	$\sum_{i=1}^{d} \lvert \frac{a_i}{\sum_{j=1}^{d} a_j} - \frac{b_i}{\sum_{j=1}^{d} b_j} \rvert \frac{1}{2}$

Tabla 3.1: Ejemplos de medidas de distancia entre elementos

- ***k***: indica la cantidad de *elementos* a considerar para realizar la predicción. Una vez ordenados todos los elementos según su distancia con el elemento a predecir, se cogerán los k más cercanos y se realizará la media ponderada de sus valores siguientes. Valores mayores reducirán el ruido, aunque a cambio las predicciones serán más homogéneas. Por tanto k es un parámetro de suavizado.
- ***d***: define la cantidad de observaciones consecutivas de una serie que conforman un *elemento*. Un valor adecuado de este parámetro será aquel que permita obtener *vecinos* significativos para la predicción.

- ***Distancia***: cuando se mide el grado de similitud entre dos elementos debe usarse una función para medir distancias. En la tabla 3.1 se presentan algunas funciones de medición de distancias válidas para este parámetro.
- ***Ponderación***: define la ponderación aplicada para calcular la predicción. Es decir, designa el método por el cual se calculan los pesos que presentan cada uno de los k valores siguientes de los vecinos, en el momento de realizar la media ponderada. Sea w el vector de longitud k que define el peso de los k-vecinos, siendo w_k el peso del elemento más cercano y w_1 el más lejano, y sea d_i la distancia al elemento i-ésimo más próximo. En la tabla 3.2 se presentan algunos métodos de asignación de pesos.

Función	**Método**
Lineal	$w_i = k - (i - 1)$
Medio	$w_i = 1$
Proporcional	$w_i = 1/d_i$

Tabla 3.2: Métodos de asignación de pesos

3.2. Ensamblado de *k-NN* para series temporales

Como ya se ha explicado anteriormente, uno de los objetivos del presente trabajo es el de mejorar la predicción del método *k-NN* para series temporales mediante ensamblado. Esta técnica consiste en la utilización de varios predictores que de forma conjunta ofrecen un mejor desempeño que de forma individual. De esta forma, la predicción realizada con cada método puede aportar al resultado final datos que el resto no hayan sabido modelizar. A mayor número de métodos presentes en el ensamblado, el resultado final presentará menor ruido. Sin embargo, aumentar la cantidad también supondrá unas predicciones más homogéneas. Por este motivo, la cantidad de métodos empleados durante el ensamblado supone un factor de suavizado de predicciones.

En el caso del *k-NN*, dada la simplicidad del método, se propone la agrupación de diferentes combinaciones de *k-d*. Para lograr esto, en primer lugar se deben generar dichas combinaciones, lo cual se llevará a cabo mediante diferentes estrategias planteadas, como se explicará más adelante. Una vez escogido el conjunto de combinaciones que formarán el grupo de predictores, se realizará la predicción mediante ensamblado. Para ello se generarán las predicciones correspondientes a cada modelo. Una vez calculadas todas ellas, se realizará una media ponderada de las mismas y esa será la predicción mediante ensamblado.

Este proceso es similar al del algoritmo base del *k-NN*, donde la predicción se realiza mediante una media ponderada de los valores siguientes de los *k*-vecinos. Es decir, realmente el *k-NN* para regresión ya funciona como una combinación ponderada. Por tanto, al igual que ocurre con el parámetro *k*, al hacer ensamblado de predicciones se busca que los diferentes modelos de predicción se complementen entre sí y disminuyan mutuamente el ruido que puedan tener presente. De nuevo cabe destacar que a medida que se empleen más predictores en el ensamblado la predicción resultante también será más homogénea.

La hipótesis en la que se basa la realización del presente trabajo es la misma que subyace al ensamblado de predicciones, que los diferentes modelos pueden ser buenos para responder a un determinado comportamiento de la serie temporal. De esta forma, por ejemplo, unos modelos pueden presentar un buen desempeño al predecir los valores que toma la serie en momentos de estabilidad. Otros modelos pueden responder bien a los cambios bruscos de tendencia al alza o a la baja. Podrían existir otros modelos que responden a periodos de alta variabilidad. Para conseguir esto, como ya se ha comentado anteriormente, se han planteado dos enfoques diferentes:

1. Entrenamiento parcial: este enfoque, basado en la técnica de *bagging*, se basa en entrenar modelos mediante un subconjunto de los datos de entrenamiento. De esta forma aunque cada uno de dichos modelos pueda llegar a presentar sobreajuste, la agrupación de todos ellos generará una mejor predicción. Dicho de forma simplificada, cada modelo se *especializará* en un aspecto de la serie temporal y en conjunto todos ellos formarán una mejor predicción.

2. Selección según error: este enfoque, basado en los algoritmos genéticos, se basa en la utilización de la matriz de errores generada en el proceso de entrenamiento. En este caso se asignará un *fitness* a cada combinación, de forma que se puedan comparar los desempeños de cada una. Concretamente al tratarse de los errores cometidos se busca minimizarlos, por tanto se utilizará como *fitness* el inverso del error.

Más adelante se explicará en detenimiento cada uno de los métodos implementados para la generación de predictores, así como la función de predicción.

3.3. Explicabilidad del *k-NN* en series temporales

El otro objetivo del trabajo es mejorar la explicabilidad de las predicciones de series temporales mediante *k-NN*. Para ello, durante la realización del proyecto se ha pretendido dotar de transparencia al método de tipo *caja negra*, buscando aportar la mayor transparencia a la realización de la predicción.

Como ya se ha expuesto previamente, el *k-NN* es un método de aprendizaje *vago* basado en instancias. Estas dos características juntas permiten una gran explicabilidad, ya que en el momento de calcular la predicción tenemos toda la información relativa a la misma cargada en memoria. Dentro de toda esta información relativa a las predicciones se puede encontrar la siguiente:

- Distancias a todos los elementos anteriores. Al calcular la distancia existente entre el instante que se desea predecir y todos los anteriores, se genera un vector de distancias. Estas se pueden ordenar y agrupar para entender la relación que guardan con el instante predicho. Por ejemplo, si todas las distancias son altas puede significar que el instante estudiado es atípico respecto al resto de la serie.

- Elementos seleccionados. Al ordenar las distancias anteriores se pueden obtener los k más cercanos, conociendo sus instantes de comienzo y fin. De esta forma se pueden estudiar cada uno de ellos y ver si el comportamiento es similar al elemento predicho. Un ejemplo de ello sería que el elemento actual pertenezca a un instante de cambio al alza pero ninguno de los k-vecinos lo sea. En ese caso podrá observarse que no existen observaciones de cambios de tendencia ascendentes similares a dicho elemento.

- Valores empleados para la predicción. Al obtener los k-vecinos más cercanos, se obtienen los índices de sus respectivos valores siguientes con los cuales se realiza la predicción. De esta forma se puede estudiar la relación entre ellos y respecto al valor siguiente de la serie, en caso de conocerlo. Por ejemplo, si se observase que existe una gran variabilidad entre los valores siguientes, podría concluirse que el comportamiento de la serie en esos casos es caótico y por tanto cabe esperar una mala predicción.

- Error cometido en la predicción. Si se realiza la predicción para un valor conocido, se puede estudiar la relación entre los datos anteriores y el error cometido. Un posible ejemplo de resultado obtenido sería que todos los valores siguientes sean parecidos pero el valor real tomado por la serie difiera mucho, y por tanto el error cometido sea alto. Esto podría significar que dicho valor real es atípico para el comportamiento de la serie hasta ese momento.

- Errores cometidos por cada combinación *k-d*. En caso de realizar predicciones de valores conocidos para realizar el entrenamiento del *k-NN*, se pueden agrupar los errores en una matriz según su *k* y *d*. Un posible comportamiento observable en dicha matriz sería que muchos modelos con una *d* en común presenten un error sensiblemente bajo en comparación al resto. Esto podría deberse a que dicho valor de *d* genera elementos que en general representan correctamente el comportamiento de la serie.

Más adelante se presentarán todas las gráficas interactivas de la aplicación web dedicada a la explicabilidad.

Capítulo 4

Algoritmos de predicción mediante ensamblado

En este capítulo se abordan los dos enfoques diferentes y cuatro implementaciones llevadas a cabo para realizar la predicción mediante ensamblado. Cabe recordar que este método consiste generar una predicción como la media de varias predicciones distintas. En el presente trabajo se empleará una metodología en la que se combinarán diferentes predicciones generadas todas ellas mediante el *k-NN*. Dicha metodología se ha bautizado con el nombre de *Random Neighborhoods*, como referencia a la técnica de ensamblado *Random Forest*.

El desarrollo ha sido llevado a cabo extendiendo la funcionalidad creada mediante el paquete `knnp` comentado anteriormente. Se han desarrollado una serie de funciones en el lenguaje de programación `R`. Cuatro de esas funciones (`rn_blocks`, `rn_samples`, `rn_genetic`, `rn_localMins`) corresponden a los métodos de entrenamiento. El resultado de estos métodos será una lista de predictores o *learners*. Cada uno contendrá los valores de k y d correspondientes al predictor *k-NN* y el peso que tendrá dicho *learner* en la predicción ensamblada. Estos métodos tienen algunos parámetros comunes:

- `ks` y `ds`: definen el conjunto de combinaciones de dichos parámetros que se probarán durante el entrenamiento. Concretamente se probarán tantas combinaciones como parejas de k y d.
- `learners`: indica la cantidad de predictores deseados para el ensamblado.
- `distance`: indica el tipo de métrica empleada para calcular la distancia de los k-vecinos.
- `weight`: indica el sistema de ponderación empleado en la predicción del *k-NN* base.

- `error_measure`: indica el tipo de métrica de error empleada para comparar las predicciones realizadas en cada *learner*.
- `n_threads`: hilos concurrentes a usar en la paralelización.

Salvo el parámetro `learners` todos los demás son comunes a la implementación del paquete `knnp`. Cabe destacar que, al igual que ocurre con el paquete `knnp`, en este desarrollo también se ha desarrollado el proceso de paralelización para acelerar los cálculos.

Por otro lado, se encuentra la función de ensamblado de predicciones (`rn_forecast`). Esta función recibirá la lista de *learners* generada por alguno de los métodos comentados. Devolverá como resultado la predicción mediante ensamblado de *k-NN*, así como una serie de métricas de la predicción. Esta función será explicada con más detenimiento al final de este capítulo.

Todo el código del desarrollo se encuentra alojado en *GitHub*[1]. A continuación, se comentarán brevemente las hipótesis manejadas detrás de cada algoritmo, y se explicará el funcionamiento y los parámetros de cada método.

4.1. Generación de predictores mediante entrenamiento parcial

Por un lado se proponen los métodos de generación de predictores mediante entrenamiento parcial. Como se ha comentado a lo largo de la presente memoria, este enfoque está basado en el método de *bagging*. La hipótesis planteada para la realización de estas implementaciones es la de obtener *learners* especializados en predecir bien determinados intervalos de una serie temporal. Con esto se pretende conseguir diferentes predictores que al combinarlos realicen una buena predicción de forma general en toda la serie.

Sea l la longitud del intervalo de la serie temporal empleado para el entrenamiento y p el número de *learners* a emplear en el ensamblado, se formarán p conjuntos de entrenamiento de longitud l', tal que $l' < l$. Cuanto mayor sea l' más se parecerán los resultados a los obtenidos mediante el *k-NN* normal entrenado en todo el intervalo de entrenamiento. Cuanto menor sea, más específicos serán los learners para predecir dicho intervalo.

4.1.1. Entrenamiento en intervalos

Corresponde al método `rn_blocks`. Se basa en la idea de elegir aleatoriamente intervalos dentro de la serie temporal y encontrar el predictor que mejor prediga cada uno de ellos. Si se generan predictores especializados en

[1] Enlace: `https://github.com/Dani-Basta/TFM`

diferentes intervalos, se espera que el resultado de combinarlos sea mejor que cualquiera de ellos de forma individual.

Esta implementación es la más similar a la técnica *bagging*. Esto es debido a que, de forma análoga al mismo, se generan p intervalos de longitud l' de forma aleatoria, definiendo los instantes de inicio y final. Para cada uno de ellos se probarán todas las combinaciones de k y d definidas mediante los parámetros `ks` y `ds` respectivamente. De todas esas combinaciones, se buscará la que minimice el error medio (`error_measure`), que será la seleccionada.

Además de los parámetros ya existentes del entrenamiento del *k-NN*, están presentes los siguientes parámetros:

- `learners`: indicará la cantidad p de combinaciones de k y d que se desea obtener como resultado. Por tanto también indicará la cantidad de intervalos a generar. Como se ha comentado con anterioridad, de forma análoga al parámetro k en el *k-NN*, se trata de un parámetro de suavizado.
- `train_length`: indicará la longitud l' que tendrá cada intervalo de entrenamiento. Si se aumenta mucho el resultado se asemejará más al del *k-NN* básico y si disminuye dará lugar a predictores más específicos.
- `initial`: indica el instante temporal donde acaba el intervalo de *warm-up* y se desea que puedan empezar los intervalos de entrenamiento. No tiene valor por defecto, sin embargo en caso de no aportarlo se mostrará un mensaje de advertencia y se hará una aproximación. Concretamente se asignará el siguiente valor: $max(\lfloor p * 0{,}3 \rfloor, max(k_i) + max(d_i) + 1)$, siendo p la longitud de la serie temporal, y k_i y d_i los valores definidos como parámetros de entrada (`ks` y `ds`) de combinaciones de k y d. De esta forma se garantizará tener al menos $k + d + 1$ observaciones disponibles como *warm-up*.
- `ensemble`: indica el sistema de pesaje a aplicar a las combinaciones. Si se emplea el valor `proportional`, el peso asociado será igual al inverso del error. Si se define como `average`, se realizará una media aritmética asignando el mismo peso a todas las combinaciones.
- `n_threads`: indica la cantidad de hilos de ejecución que se desean emplear de forma simultánea. Por defecto se establece en 2 para proporciona un paralelismo básico.

El proceso seguido para generar cada *learner* es el siguiente:

1. **Generar instante comienzo**: se define aleatoriamente un instante temporal que marcará dónde comienza el intervalo de entrenamiento. El menor instante será el indicado mediante el parámetro `initial`.

2. **Calcular instante final**: se calcula el instante final del intervalo, sumando la longitud indicada en el parámetro `train_length` al instante de comienzo. En caso de que este valor exceda la longitud de la serie, se limitará el índice al último valor de la serie.

3. **Entrenar el predictor sobre el intervalo**: se entrenará el *learner* sobre el intervalo definido mediante la función `knn_param_search`, empleando los parámetros `ks` y `ds` establecidos para los valores de k y d respectivamente.

4. **Obtener parámetros del *learner***: se obtendrán los valores de k y d de la combinación óptima. También se calculará el peso que tendrá ese predictor en el ensamblado, mediante el inverso del error cometido por la combinación seleccionada. Es análogo a la ponderación de cada uno de los k-vecinos del *k-NN* básico.

Cabe destacar el uso del parámetro `n_threads` de esta función. Por un lado se planteó la opción de utilizar el parámetro `threads` de la función `knn_param_search` (Bastarrica y Berdecio, 2018). De esta forma se realizaría un bucle iterativo de llamadas a dicho método, y sería éste el encagardo de paralelizar el entrenamiento.

Sin embargo se planteó otra solución, la de entrenar los diferentes modelos de forma paralela. Para ello se utilizó un bucle que permite la ejecución de sus iteraciones de forma concurrente. De esta forma, cada *learner* hace uso de la función `knn_param_search` indicando que no se paralelice la ejecución. Y son estos predictores los que se entrenan de forma concurrente.

Para decidir el enfoque se realizaron pruebas de comparación de tiempos. Finalmente se ha empleado el segundo enfoque, en el que se emplea un único hilo para cada llamada de entrenamiento, y se entrenan simultáneamente tantos predictores como se haya indicado en el parámetro `n_threads`.

4.1.2. Entrenamiento disperso

Corresponde al método `rn_samples`. Este método se basa nuevamente en *bagging*. Es muy similar al anterior, sólo que en lugar de definir intervalos de la serie sobre los que entrenar, se definen instantes temporales dispersos. En este caso se busca entrenar los diferentes predictores mediante muestreos aleatorios de la serie.

Nuevamente se generarán p conjuntos de tamaño l' de forma aleatoria, definiendo todos los instantes temporales que los componen. Se realizarán predicciones con todas las combinaciones de k y d definidas y se seleccionará la que minimice el error medio.

Algunos parámetros característicos de esta implementación, además de los provenientes del *k-NN* básico, son los siguientes:

- `learners`: indicará la cantidad p de combinaciones de k y d que se desea obtener como resultado. Por este motivo también determinará la cantidad de conjuntos de observaciones a generar. De nuevo comparte el efecto del parámetro k en el *k-NN*, a mayor cantidad más se suaviza la predicción final.
- `n_instants`: de forma análoga al parámetro `train_length` del método anterior, indicará la cantidad de observaciones a seleccionar en cada conjunto de entrenamiento. Al igual que en el método anterior, a mayor cantidad de observaciones más se asemejarán los resultados a los del *k-NN* base.
- `initial`: indica el instante temporal en el que acaba el periodo de *warm-up* y a partir del cual se entrenarán los diferentes learners.
- `ensemble`: define el sistema de pesaje aplicado a las combinaciones. El valor `proportional` usará un peso igual al inverso del error. En caso de usar `average` se realizará una media aritmética (mismo peso).
- `n_threads`: define la cantidad de hilos de ejecución paralelos que se desean emplear en el proceso de entrenamiento. Al igual que en el caso anterior, se entrenarán $n\_threads$ modelos de forma simultánea, cada uno empleando un único hilo.

A diferencia del método anterior, en esta aproximación no aporta valor calcular las distancias de observaciones que estén más allá de la última a estudiar (la más reciente en el tiempo). Es decir, como se ha comentado anteriormente, el *k-NN* busca elementos similares en el pasado de la serie. Por tanto, dado el conjunto de `n_instants` a predecir, todos los instantes temporales de la serie posteriores al último seleccionado en el conjunto, no aportan información relevante. Por este motivo, en esta implementación se ha evitado el cálculo de distancias de todas las observaciones del final de la serie que no se vayan a predecir en el entrenamiento.

El proceso seguido por este método para generar cada uno de los diferentes learners es el siguiente:

1. **Generar instantes temporales**: se genera un conjunto de `n_instants` instantes temporales escogidos de forma aleatoria. La selección se realiza con reemplazamiento, por lo que podrá haber instantes repetidos. En este caso se evaluará su error tantas veces como se haya escogido.
2. **Entrenar el *learner* sobre el conjunto**: se entrenará el *learner* sobre todos los instantes temporales definidos en el conjunto de entrenamiento. Como se verá más adelante, dicho entrenamiento se realizará mediante la función `knn_param_search_disp` empleando los parámetros establecidos.

3. **Obtener parámetros del *learner***: se obtendrá el valor de los parámetros k y d del predictor que hayan minimizado el error establecido. También se calculará el peso de este predictor empleando para ello el inverso del error de la combinación seleccionada.

Respecto a la implementación, en este caso se ha desarrollado una función auxiliar para llevar a cabo el método. A fin de reaprovechar el código ya existente, se buscó una forma de entrenar un *learner* sobre un conjunto de observaciones dispersa en lugar de un intervalo. Para ello se creó la función `knn_param_search_disp` como una modificación de `knn_param_search`, la función para entrenar predictores del *k-NN* base.

En esta función se sustituye el parámetro `initial` por el parámetro `indexes`. Mediante ese parámetro se proporcionarán los índices correspondientes de todos los instantes temporales que se desean emplear para el entrenamiento.

Además, se optimizó el proceso de generación de matrices de distancias. En el *k-NN* base se predicen todos los instantes desde `init` hasta el final de la serie. Por ese motivo es necesario calcular la matriz de distancias desde el último elemento (el más reciente) hasta el primero.

En este método sí se ha empleado el parámetro `n_threads` de la función de entrenamiento. A diferencia del método anterior aquí las pruebas arrojaron un mejor rendimiento al paralelizar el entrenamiento en cada *learner*, en lugar de entrenar de forma simultánea diferentes predictores.

4.2. Generación de predictores mediante selección

Por otro lado están los métodos de generación de *learners* mediante selección. Como ya se ha comentado con anterioridad, estos se inspiran en los algoritmos genéticos. Por un lado se emplea la asignación de coste o *fitness* a los individuos (combinaciones de k y d), que consiste en una medida del desempeño que posee cada combinación. De esta forma presentarán mayor *fitness* aquellas combinaciones que minimicen el error de predicción. Por tanto la probabilidad de selección que tendrá cada individuo será en función de dicho valor de *fitness*.

En este caso la hipótesis detrás de estos métodos es que dentro de las combinaciones de k y d que prueba el método de entrenamiento, no basta con observar el valor agregado de la métrica del error, ya que una combinación sub-óptima puede aportar información al ensamblado que la óptima no posee. Al igual que en el algoritmo genético, se considerará que los individuos (combinaciones de k y d) con un *fitness* inferior al óptimo no deben ser descartados. Estos pueden proporcionar información relativa a la predicción de la que las otras combinaciones con mayor *fitness* (menor error) carecen.

De esta forma el resultado final buscado es el mismo que con el enfoque del *bagging*, encontrar diferentes combinaciones de *learners* que al ensamblarse funcionen mejor que todas ellas por separado.

4.2.1. Selección basada en fitness

Corresponde al método `rn_fitness`. Este método utiliza varias de las ideas detrás de los algoritmos genéticos, como el elitismo o la selección mediante torneo. De forma general la probabilidad de escoger los diferentes individuos (combinaciones de k y d) será inversamente proporcional al error que cometen.

En este caso no existe una partición del intervalo de entrenamiento como ocurre en los enfoques anteriores inspirados en *bagging*. Al igual que en el *k-NN* base se entrenan todas las combinaciones en el intervalo completo, y una vez obtenidos los errores medios se realiza la selección.

A continuación se explica el proceso seguido por el algoritmo para generar los diferentes *learners*. Para facilitar su comprensión, se explican los diferentes parámetros de entrada a medida que son empleados:

1. **Obtención de los errores de cada combinación**: se realiza la predicción correspondiente a cada combinación de k y d establecida. Esto se hace mediante la función `knn_param_search` para todo el intervalo de entrenamiento definido por el usuario.

2. **Selección directa**: se realiza mediante el parámetro `elitism`. Los individuos que presenten un mayor *fitness* serán escogidos directamente. Podrá tomar un valor entre 0 y el total de combinaciones de k y d definidas. Por ejemplo, si ambos toman valores de 1 a 10, existirán 100 combinaciones, y por tanto como máximo el elitismo podrá llegar a ese valor. Si se define a 0, no se aplica este método.

3. **Obtención del *fitness***: para cada combinación de k y d se calculará su *fitness*, siendo éste el inverso del error cometido en el entrenamiento.

4. **Filtrado de peores combinaciones**: se realiza mediante el parámetro `threshold`. Podrá tomar un valor entre 0,0 y 1,0, siendo el valor por defecto este último. Indicará el tanto por uno de combinaciones de k y d a considerar, descartando las demás. Por ejemplo, un valor de 0,7 descartará el 30 % de las combinaciones que peor error (menor *fitness*) tengan. El *fitness* de estas combinaciones se establecerá a 0, por lo que no podrán ser elegidas. Este parámetro actúa de filtro, ya que se puede considerar que hay ciertas combinaciones que sólo aportan ruido.

5. **Filtrado de combinaciones con mejora local**: se define mediante el parámetro `localMins`. Toma un valor entre 0, desactivado, y cualquier número positivo. De forma similar al anterior, determinará qué combinaciones pueden ser escogidas y cuáles no. Concretamente esta selección busca aquellas combinaciones que funcionen mejor que otras combinaciones adyacentes de acuerdo a sus valores de k y d. Sea e_{kd} el error cometido por la combinación definida con esos valores para k y d, y r el valor del parámetro `localMins`: $\forall i \in [k-r, k+r], \forall j \in [d-r, d+r], e_{ij} \leqslant e_{kd}$. Las combinaciones que cumplan dicha propiedad se consideran *Mínimos locales*, ya que presentan el mínimo error de sus combinaciones adyacentes. Todas las combinaciones que no cumplan la propiedad serán descartadas asignándoles un *fitness* de 0. Dicho de forma sencilla, una combinación será mínimo local si presenta un error menor que sus combinaciones adyacentes en la matriz de errores. Cuanto mayor sea este parámetro, más combinaciones cercanas comprobará para ver si comete el menor error, y por tanto habrá una cantidad menor o igual resultados respecto a un valor menor del mismo.

6. **Selección de *learners***: la variable `tournament` será la responsable de definir el método de selección. Si se establece a `FALSE`, se hace un muestreo de tantas combinaciones como falten, de tal forma que la probabilidad de escoger cada predictor es el *fitness* del mismo. En cambio si el parámetro se define a `TRUE`, se generarán el doble de dichas predicciones y luego se harán comparaciones dos a dos, para obtener la mejor combinación de cada pareja y seleccionar esa. Este proceso de selección no se llevará a cabo cuando el parámetro número de *learners* solicitado por parámetro. Esto puede no ocurrir en configuraciones con *elitismo* demasiado elevado.

Por último se calcularán los pesos asociados a cada *learner*. Para ello se emplea el parámetro `ensemble`. Al igual que en los métodos anteriores, si se establece a `proportional` se usará el *fitness*, y si se establece a `average` se usará el mismo peso para todos (media aritmética).

Al emplear la función `knn_param_search`, el parámetro `n_threads` se pasará directamente a la misma (paso 1) y por tanto se realizará un entrenamiento concurrente en tantos hilos como se indique.

4.2.2. Selección de mínimos locales

Corresponde al método `rn_localMins`. Este algoritmo simplifica la hipótesis del método anterior anterior. De esta forma sólo se emplean las combinaciones que cumplan la propiedad de mínimo local. Cabe destacar que por este motivo esta es la única implementación que no tiene un componente aleatorio en su generación de *learners*.

Nuevamente, al igual que en el método anterior, se estudiará las combinaciones de k y d establecidas mediante los parámetros *ks* y *ds* respectivamente. Se realizarán predicciones de todo el intervalo de entrenamiento definido para probar el rendimiento de cada combinación.

A continuación se explica el proceso por este algoritmo para generar los *learners*, junto al uso de cada parámetro a medida que se emplean:

1. **Obtención de los errores de cada combinación**: se realiza la predicción con cada combinación de k y d definida. Para ello se emplea la función `knn_param_search` sobre el intervalo de entrenamiento definido por el usuario completo.

2. **Obtención del *fitness***: se calcula el *fitness* de cada combinación de k y d como el inverso del error cometido en entrenamiento.

3. **Filtrado de combinaciones**: se lleva a cabo según el parámetro `threshold`. Este puede tomar valores mayores que 0,0 y hasta 1,0. Indica el tanto por uno de combinaciones a considerar, descartando el resto. Por ejemplo, un valor de 0,6 descartará el 40 % de las combinaciones con mayor error cometido. Mediante este parámetro se pueden descartar aquellas combinaciones que aún siendo mínimos locales, se consideren excesivamente malas en términos de error.

4. **Cálculo de mínimos locales**: se realiza según el parámetro `radius` y es análogo al método anterior. Para cada combinación de k y d se estudia si las combinaciones adyacentes según su valor de k y d en la matriz de errores, a una distancia `radius`, presentan un mayor error. En caso afirmativo se considera un mínimo local.

5. **Truncar *learners***: se hace mediante el parámetro `learners`. En este caso su funcionamiento difiere con el resto de métodos. Si la cantidad de mínimos locales hallados es mayor que `learners`, se escogen los mejores. En caso de hallar menos, se devuelve todos ellos y se muestra un *warning* al usuario para indicar que no se han encontrado suficientes.

Por último se calcula el peso de cada predictor en el ensamblado. Si el parámetro `ensemble` se establece como `proportional` el peso será el inverso del error. En caso de usar `average` el peso de todas las combinaciones será el mismo.

Nuevamente el grado de paralelización se determinará mediante el parámetro `n_threads`, el cual se proporcionará a la función `knn_param_search` (paso 1).

4.3. Predicción mediante ensamblado

Hasta ahora se han explicado los métodos de generación de predictores o *learners* a partir del conjunto de entrenamiento de la serie temporal. A continuación se explica el proceso de ensamblado o combinación de todos ellos.

Como se ha comentado con anterioridad, el resultado de los métodos de entrenamiento es un conjunto de *learners* o predictores, cada uno de ellos caracterizados de la siguiente manera:

- **K**: número de elementos a buscar.
- **D**: longitud de cada *elemento*.
- **Peso**: peso que tiene el *learner* en la media ponderada de predicciones.

Este método generará tantas predicciones como *learners* se le proporcionen y se corresponde al método `rn_forecast`. Para cada uno de ellos generará la predicción desde el instante `init` hasta el final de la serie. Esto se realizará mediante el método `knn_past`, empleando en cada predicción las métricas de distancia y peso definidas.

Una vez realizadas todas las predicciones, se harán las medias ponderadas instante por instante. Es decir, para cada instante de la serie se cogen las predicciones de todos los learners y se hace la media asignando a cada *learner* su peso correspondiente.

Capítulo 5

Aplicación web para predicción explicable mediante *k-NN*

Como ya se ha comentado anteriormente, uno de los objetivos del presente trabajo es el de facilitar y profundizar en la comprensión de las predicciones realizadas mediante el *k-NN*. Concretamente se busca la *explicabilidad* y transparencia de las mismas, esto es, entender el proceso que se ha seguido para realizar cada predicción y los datos empleados para ello. Para ello se ha trabajado buscando resolver los siguientes aspectos:

- Determinar el grado de similitud de un intervalo de la serie con respecto a los anteriores: como ya se ha comentado anteriormente, el *k-NN* trabaja con *elementos* de longitud d que representan intervalos de la serie. Se busca estudiar, para un cierto elemento de la serie, si los elementos anteriores se parecen en mayor o menor medida. Determinar por tanto si el elemento presenta un patrón típico o no, y observar si ha ocurrido dicho patrón con anterioridad.
- Identificar los empleados para cada predicción individual: de forma análoga al caso anterior, entender para un intervalo determinado qué intervalos pasados son los más parecidos. De esta forma se consigue comprender el grado de similitud que guardan los vecinos más cercanos con el intervalo y si los valores empleados para realizar la predicción son similares al que se desea predecir.
- Estudiar relaciones de los vecinos y sus valores futuros: dados los k-vecinos de una predicción y sus valores futuros, estudiar su relación con el desempeño de la predicción mediante diferentes métricas.

Por otro lado también se busca aumentar la *explicabilidad* y transparencia de los métodos de *Random neighborhoods* desarrollados. Esto se ha resuelto estudiando los siguientes apartados:

- Efecto de las predicciones empleadas en la combinación: observar el efecto de añadir o eliminar combinaciones a la predicción y modificar el peso que posee cada una de ellas.
- Efectividad de combinaciones mejores que sus similares: estudiar qué información aportan las combinaciones de K y D cuyo error en la predicción es menor que aquellas con valores cercanos de dichos parámetros. Es decir, entender por qué ciertas combinaciones presentan un mínimo local en posición de la matriz de errores respecto a los valores adyacentes.

Dados los diferentes ámbitos que se pretenden resolver mediante una interfaz de usuario, se ha planteado la modificación de las dos interfaces ya existentes así como la creación de una nueva dedicada exclusivamente a la combinación de predicciones.

5.1. Pestaña *Distances*

En primer lugar se encuentra la pestaña *Distances* (figura 5.1). El funcionamiento del método del *k-NN* se basa la premisa de que dado un instante temporal, el comportamiento que tendrá a continuación la serie será similar a la que tuvo en el pasado en momentos similares. Por tanto, el objetivo de esta pestaña es ayudar a descomponer la predicción de una serie temporal en cada uno de los diferentes instantes de la misma que se han predicho, para así estudiar individualmente si la premisa se cumple. De esta forma, se puede ir más allá de la medida de error obtenida para el conjunto completo estudiado. Con esto se logra entender si una combinación de k y d acierta o falla, y dónde se produce cada situación. Esta pestaña se compone a su vez de tres grupos de gráficas distintos.

5.1.1. Predicción de una serie temporal y métricas asociadas

Esta primera gráfica está compuesta por 4 gráficas juntas combinadas (ver figura 5.2). Su objetivo es proporcionar distintas métricas para facilitar la comprensión de la predicción realizada en cada instante temporal. En concreto busca relacionar, para cada instante, el error cometido con la similitud a los k vecinos y la desviación típica de los valores siguientes de dichos vecinos. La configuración de estas gráficas queda de la siguiente manera:

1. Muestra la serie estudiada, la predicción realizada y los intervalos utilizados para entrenamiento y test sombreados. Esta gráfica ya estaba presente anteriormente en el desarrollo.

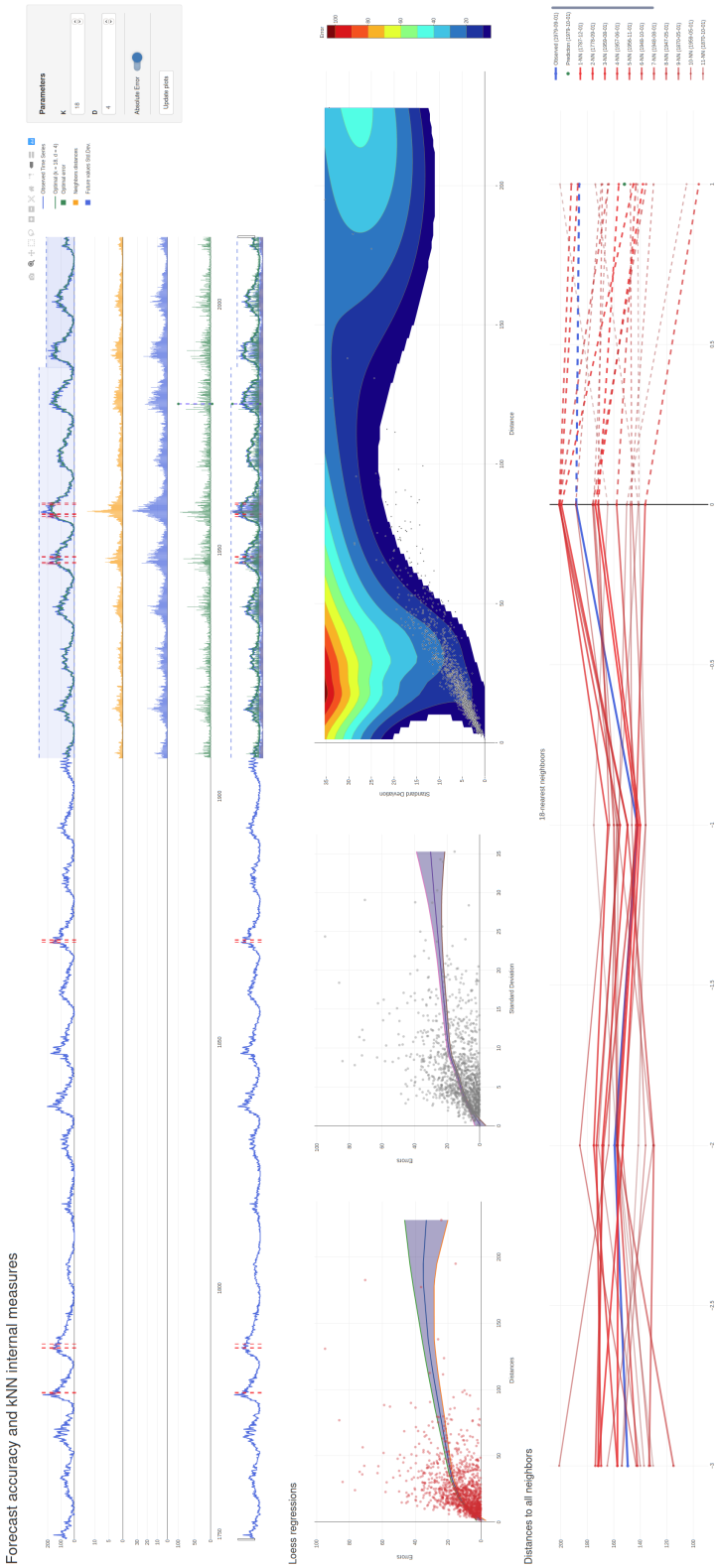


Figura 5.1: Vista general pestaña *Distances* completa

2. Muestra para cada instante la distancia media a sus k-vecinos más cercanos, y por tanto a aquellos con los que se ha realizado la predicción. Esto muestra el grado de similitud que tiene el elemento que queremos predecir con aquellos elementos que más se asemejan. Por tanto, se puede identificar aquellos elementos que no poseen vecinos parecidos.

3. Muestra para cada instante temporal la desviación estándar existente entre los valores siguientes de sus k-vecinos, es decir, los valores con los que se realiza la predicción. Esto permite conocer el grado de variabilidad existente entre ellos.

4. Muestra para cada instante temporal los errores cometidos al realizar la predicción mediante sus k-vecinos. Esta gráfica también se encontraba ya presente con anterioridad en el desarrollo.

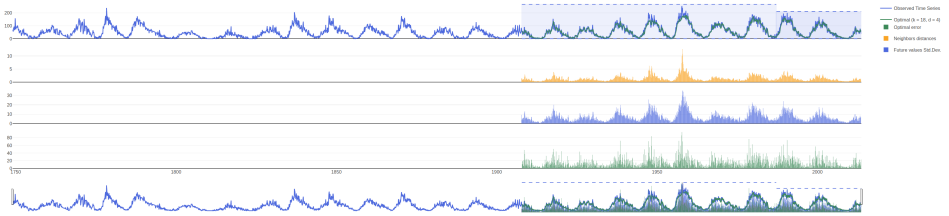


Figura 5.2: Serie temporal junto a la predicción del *k-NN* y sus métricas

El propósito de este conjunto de gráficas es estudiar la relación existente entre la similitud de los k-vecinos, la variabilidad de sus valores siguientes y el error cometido. De esta forma se observar si los valores están relacionados. Por ejemplo, si sus incrementos y decrementos se producen al unísono, o si esa relación no ocurre en algún momento del tiempo. De esta forma se pueden estimar los instantes temporales de la serie con mayor o menor probabilidad de ser correctamente predichos. O por el contrario encontrar aquellos instantes que a priori podrían haberse predicho con exactitud pero se ha cometido un gran error en la predicción.

Todos estos valores presentan un eje de abscisas común que permite agrandar o acortar el rango del mismo que se quiere observar (ver figura 5.3). Además al mantener sincronizadas automáticamente todas las gráficas, al modificar el rango del eje de abscisas permite observar cómodamente todos los valores correspondientes a las diferentes métricas para cada instante temporal.

Además de las propias gráficas, esta parte de la interfaz presenta al lado un pequeño panel (ver figura 5.4) para establecer qué configuración de parámetros k y d se quieren emplear, así como un selector de si se desea ver la gráfica del error en valor absoluto o no. Para confirmar la selección de la configuración actual se utilizará el botón de actualizar.

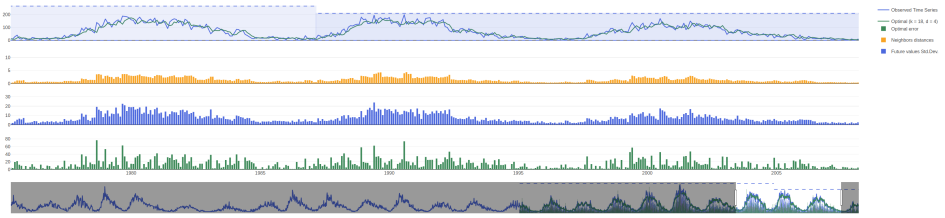


Figura 5.3: Intervalos de entrenamiento y *test* de la predicción del *k-NN*

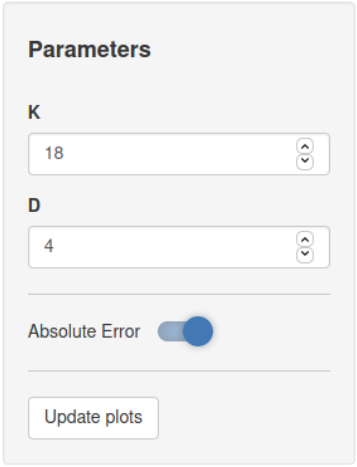


Figura 5.4: Panel de configuración de predicción mediante *k-NN*

5.1.2. Gráficas de dispersión

En este apartado de la aplicación se mostrará la relación entre la diferentes métricas del apartado anterior de nuevo. Sin embargo en este caso se hará una representación no dependiente de sus dimensión temporal. De esta forma se puede observar si existe una relación entre el error de predicción de cada instante temporal respecto a la distancia media a sus k-vecinos y/o la dispersión de los valores futuros de dichos vecinos.

Para ello este espacio está compuesto de 3 gráficas (ver figura 5.5) de relaciones entre diferentes métricas. Al igual que en el caso anterior, este gráfico se actualizará en caso de seleccionar una configuración diferente para los parámetros k y d y confirmar con el botón. Cada una de las gráficas representará lo siguiente:

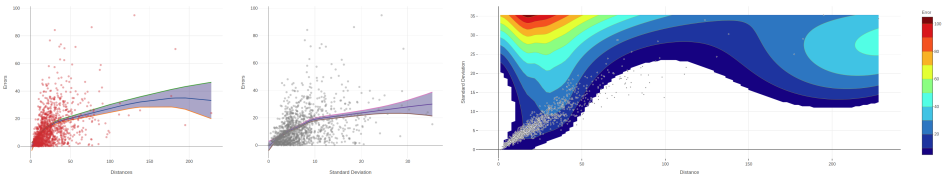
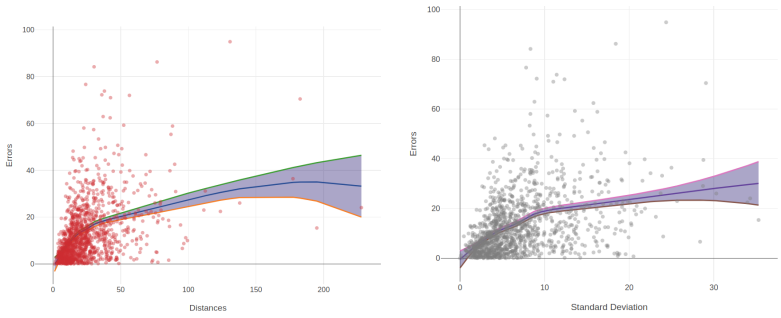


Figura 5.5: Gráficas de métricas asociadas a la predicción del *k-NN*

1. Relación entre distancias y errores. En esta primera gráfica (figura 5.6a) se mostrarán todas las distancias medias a los k-vecinos, comentadas anteriormente, con sus correspondientes errores en la predicción. Esto permite dibujar la relación existente entre cada error de predicción y la distancia que presentaban los elementos con los que se ha realizado dicha predicción. De esta forma se observará si dicha relación es lineal o del tipo que sea. Además se calculará la regresión local (*loess*) de dichos puntos, y se añadirán 3 líneas correspondientes al valor promedio de la regresión y al intervalo de confianza al 95 %. Es decir, el intervalo de confianza determinará el error medio con un 95 % de probabilidades.



(a) Distancia vs Error

(b) Desviación vs Error

Figura 5.6: Gráficas de relación de las métricas con el error

2. Relación entre desviaciones estándar y errores. Esta gráfica (figura 5.6b) es análoga anterior, pero representando la relación entre la desviación estándar de los valores futuros de cada elemento, comentada también anteriormente, y el error de predicción. Igualmente se calcula la regresión local y el intervalo de confianza al 95 %.

3. Relación entre distancias, desviaciones y errores. Esta gráfica (figura 5.7) es análoga a las anteriores, pero siendo el objetivo el de analizar la relación conjunta de la distancia media y la desviación tipica, respecto al error. Para ello el eje de abscisas se empleará para las distancias medias a los k-vecinos y el eje de ordenadas para las desviaciones estándar

de los valores siguientes. Mediante la combinación de ambas se creará un modelo de regresión local con dichas variables como entrada y el error como salida. El resultado de dicha combinación será un modelo realizado como un hiperplano estimado para la variable error. Dicho modelo se empleará para generar estimaciones de los errores cometidos a lo largo de los valores que toman tanto las distancias medias como las desviaciones estándar. Concretamente se dividirá el rango de valores que toma cada una de ellas en 100 valores equidistantes, de forma que se genere una matriz de 100x100 valores. Dicha matriz contendrá en cada posición la estimación de error del hiperplano creado, correspondiente a los valores de distancia y desviación de dicha posición. En caso de que la medida de error seleccionada no pueda presentar valores negativos (como el *MAE* o el *RMSE*), se eliminarán los posibles errores negativos presente el plano. Además, para poder conocer las zonas del mismo con mayor cantidad de combinaciones de distancia media y desviación, se mostrarán todos los puntos correspondientes a observaciones de las mismas. Esto permite conocer qué áreas del modelo se sustentan en una mayor o menor cantidad de observaciones.

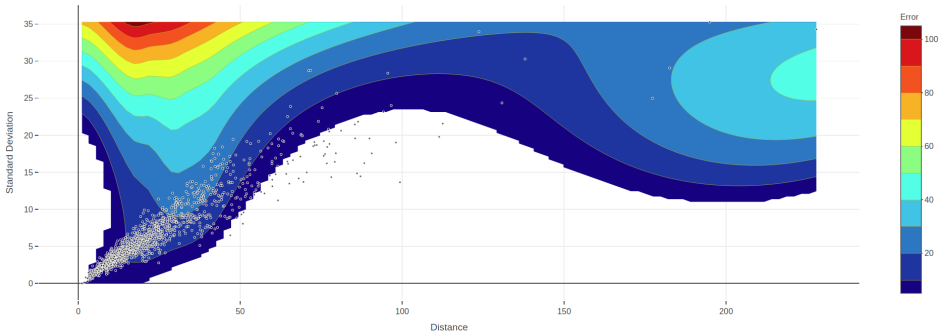


Figura 5.7: Relación de la distancia y la desviación con el error

5.1.3. k-vecinos más cercanos de un instante temporal

Esta gráfica (ver figura 5.8) es generada de forma dinámica al interactuar con la primera gráfica de la pestaña (de serie y predicción). Su función es proporcionar la transparencia y *explicabilidad* buscadas, pudiendo para ello observar en detalle la información relativa a la predicción realizada para cada instante temporal. Concretamente su comportamiento se activará al hacer click en un instante temporal de la primera gráfica de esta pestaña.

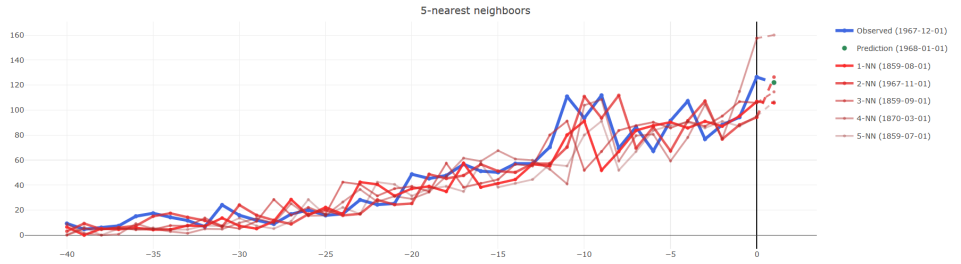


Figura 5.8: Gráfica de los k-vecinos seleccionados

Para ello, este gráfica mostrará los k-vecinos más cercanos al elemento seleccionado, es decir, los empleados para realizar la predicción seleccionada. Se mostrarán los D valores consecutivos que componen dichos k-vecinos así como el valor siguiente (ver figura 5.9) de cada uno. De forma análoga mostrará los D valores de la serie que caracterizan al elemento sobre el que se ha hecho click. Por último se mostrará el valor que se predijo con dicha combinación.

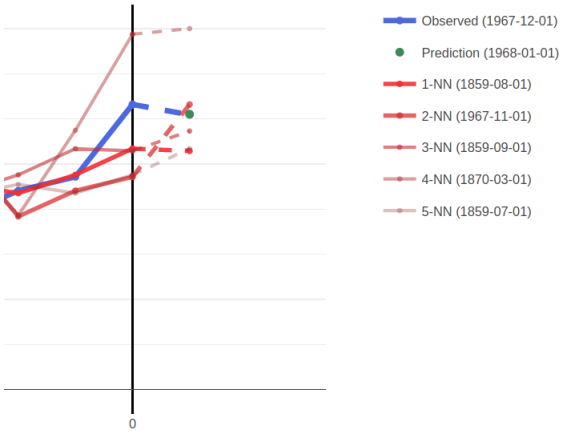


Figura 5.9: Detalle del valor siguiente de los k-vecinos

Esta información servirá para conocer en qué medida eran similares los k-vecinos más cercanos, así como la distancia existente entre sus valores siguientes y el valor real que tomó la serie a continuación. Con esta información es posible analizar caso a caso cada predicción, y estudiar por ejemplo si los valores menos útiles para realizar la predicción (los más lejanos al valor real tomado por la serie a continuación) pertenecen a los elementos menos similares de los k-vecinos.

5.2. Pestaña *Optimization*

El objetivo de esta pestaña es estudiar y comparar las predicciones de diferentes combinaciones de k y d. Para ello se muestra el rendimiento de cada una en el intervalo de entrenamiento definido, y permite añadir y eliminar de forma sencilla las combinaciones deseadas. También se puede añadir información adicional relativa al desempeño de cada combinación de k y d. Por último para cada combinación escogida, muestra la predicción, el error medio, y una comparativa del rendimiento respecto a la mejor.

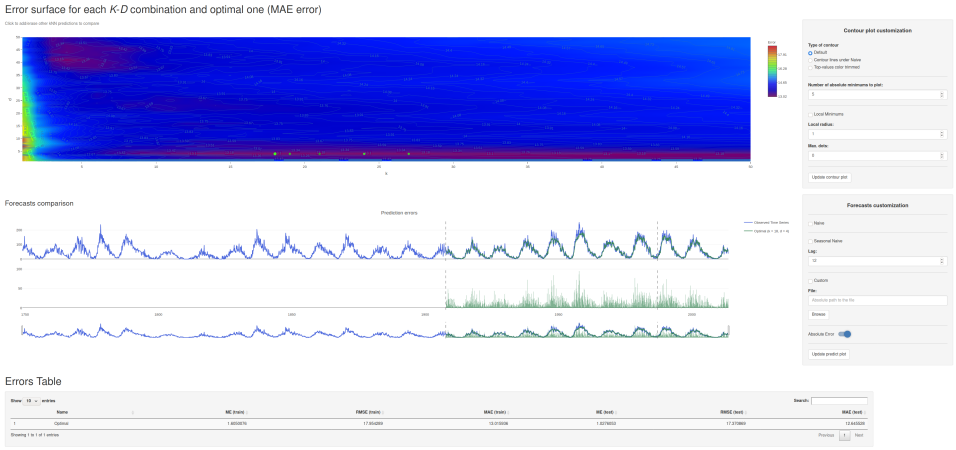


Figura 5.10: Vista general pestaña *Optimization*

Para realizar todo lo descrito, en esta pestaña (ver figura 5.10) se presentan un gráfico de tipo mapa de calor y contorno, que representa el error cometido en el intervalo de entrenamiento para cada combinación de k y d, un gráfico de series, donde aparecen la combinación óptima y las seleccionadas mediante la gráfica anterior, así como una tabla con los errores de cada predicción. Se han realizado dos modificaciones visuales durante el desarrollo del presente trabajo.

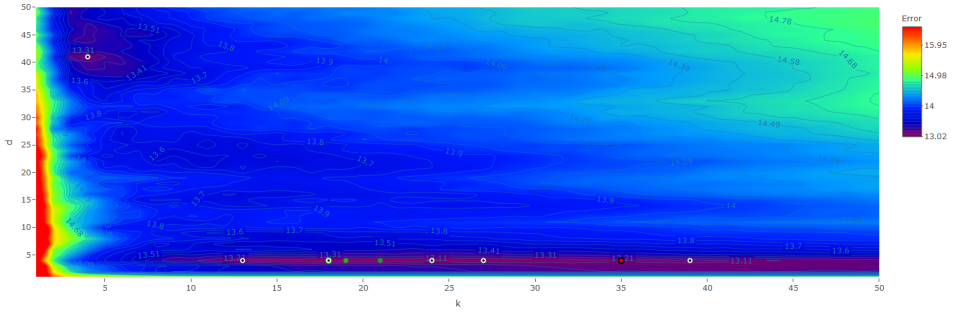
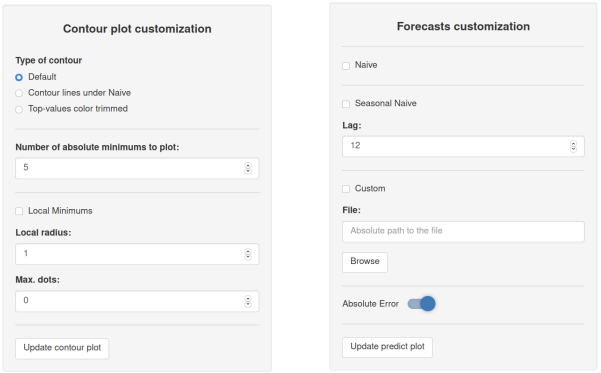


Figura 5.11: Errores de las combinaciones k-d de la pestaña *Optimization*

En primer lugar se ha modificado el gráfico de contorno para que pueda mostrar los mínimos locales (puntos negros con contorno blanco) presentes en la matriz de errores, generada por la función *_param_ search*. Es posible establecer el radio de elementos adyacentes a estudiar para determinar si es un mínimo local, así como limitar la cantidad máxima de ellos a mostrar en la gráfica (puntos negros figura 5.11).

Con estos mínimos locales se busca estudiar aquellas combinaciones que presentan un mejor rendimiento respecto a sus adyacentes. Éstas resultan interesantes de estudiar, ya que dicha mejora relativa puede deberse a que presenten dinámicas de predicción diferentes, como predecir mejor ciertos patrones de comportamiento. Son estos mínimos locales los empleados en los métodos de *Random Neighborhoods* de Fitness y de Mínimos. De esta forma se puede estudiar cada combinación de k y d mediante la pestaña *Distances*, y en caso de resultar interesante añadirla a la combinación de predicciones mediante la pestaña *Combination* para buscar mejores predicciones.

Por otro lado se ha modificado la parametrización de las dos gráficas. Para la primera gráfica, la matriz de errores, se han añadido los elementos necesarios para realizar la función comentada en el párrafo anterior respecto a los mínimos locales (figura 5.12a).



(a) Parametrización contour (b) Parametrización serie

Figura 5.12: Recuadros parametrización pestaña *Optimization*

En segundo lugar, respecto a la gráfica de comparación de predicciones, se ha agregado la carga de una predicción desde un fichero *RDS* como opción a comparar. Este tipo de ficheros son empleados por el entorno de *R* para almacenar objetos de cualquier tipo. Concretamente en este caso sería necesario guardar un serie temporal, ya sea en formato vector o `ts`. Esto permite al usuario añadir predicciones generadas mediante otros métodos, para compararlas con el *k-NN* y los *Naive*. Además, de forma análoga a la pestaña *Distances*, ahora los cambios en ambas gráficas no se vuelven efectivos hasta que se utilizan los botones correspondientes (figura 5.12b).

5.3. Pestaña *Combination*

Por último, se ha añadido una nueva pestaña enfocada en la predicción mediante combinación desarrollada en el presente trabajo (figura 5.13). Con esta pestaña se desea facilitar la comprensión de la predicción mediante *Random Neighborhoods*, así como proveer de una interfaz sencilla con la que poder probar manualmente dichas combinaciones, añadiendo o eliminando predictores a la combinación. Al igual que en las pestañas anteriores, se puede dividir en dos grupos.

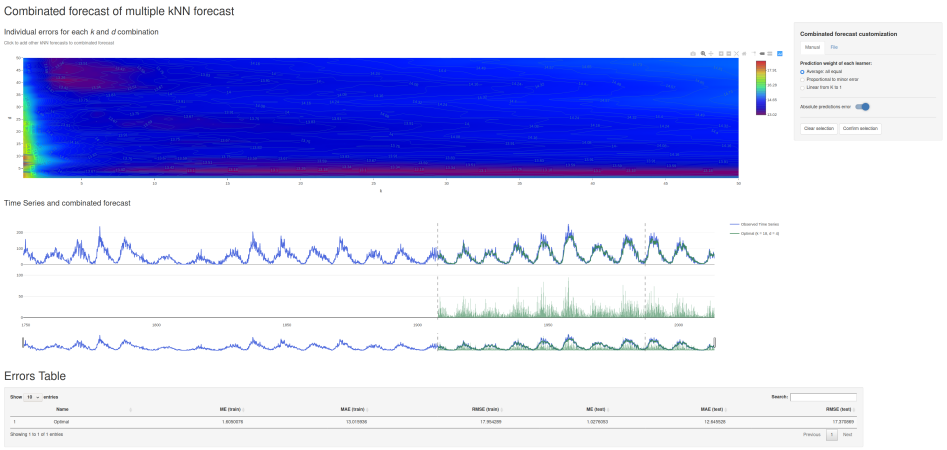


Figura 5.13: Vista general pestaña *Combination*

5.3.1. Parametrización de combinación

Este primer grupo está formado por la gráfica de contorno y el cuadro de personalización. La gráfica de contorno (figura 5.14) sería similar a la de la pestaña anterior, consistiendo en un gráfico de contorno combinado con un mapa de calor, en el que cada posición contiene el error medio cometido por dicha combinación en el intervalo de entrenamiento. Sin embargo en esta ocasión al seleccionar una determinada posición, permite añadir la predicción generada por ella a los *Random Neighborhoods*. De esta forma, se pueden añadir y eliminar predictores a la combinación de forma sencilla.

Por otro lado está el cuadro de personalización asociado a la predicción mediante combinación. Este a su vez contiene dos pestañas, para seleccionar de forma manual los pesos y predicciones o realizarlo mediante un fichero. En caso de seleccionar la primera (figura 5.15a), podremos elegir entre los tres métodos de ponderación de las predicciones: media aritmética (mismo peso para todos), proporcional al error individual o lineal (pesos de K a 1 a medida que aumenta el error).

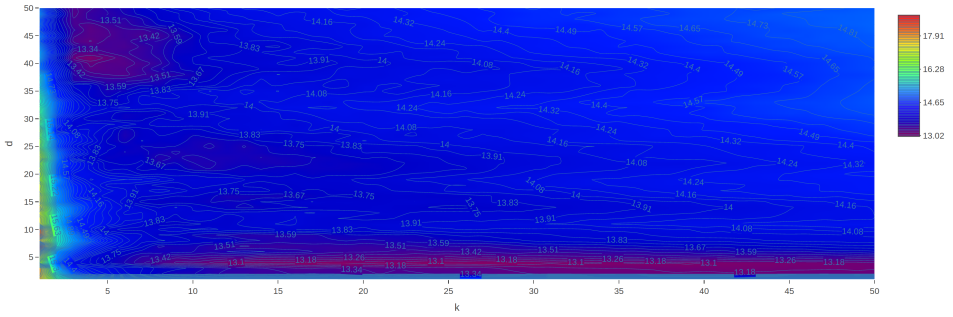
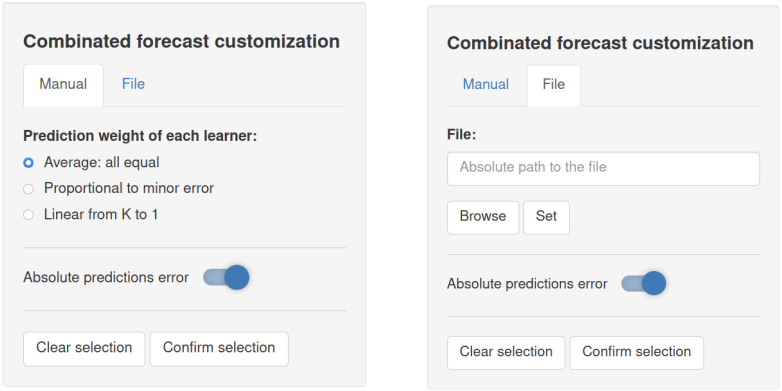


Figura 5.14: Mapa de errores para selección de combinación



(a) Parametrización manual (b) Parametrización desde fichero

Figura 5.15: Recuadros parametrización pestaña *Combination*

Sin embargo en la opción de cargarlo mediante fichero (figura 5.15b) se utilizará un fichero *RDS*. En este caso deberá contener el objeto creado por alguno de los cuatro métodos de generación de combinaciones, de forma que también se cargarán los pesos calculados por el método. Cabe destacar que dichos pesos serán los empleados para la ponderación siempre y cuando el usuario no modifique manualmente los predictores que conforman la combinación.

Además el cuadro presenta un selector en ambas pestañas que permite elegir si se desea que el error mostrado para la combinación sea en valor absoluto o no, así como dos botones para eliminar toda la selección actual y para confirmar la selección, respectivamente.

5.3.2. Visualización de predicción y su error

Por último este grupo está conformado por una gráfica y una tabla de errores, ambas análogas a las de la pestaña de *Optimization*. La gráfica (figura

5.16) mostrará la serie temporal, la predicción realizada por la combinación óptima de K y D obtenida mediante el método *knn_param_search*, y la predicción generada mediante combinación de predicciones, en caso de que se haya seleccionado una. Se empleará la combinación óptima como referencia ya que es el mejor predictor individual encontrado. Además mostrará los errores cometidos por cada predicción en cada instante temporal de los intervalos de *train* y *test*, así como un gráfico de barras comparativo del error cometido por ambas predicciones. Este último gráfico presentará un valor positivo si el error de la combinación de predicciones seleccionada es mejor que la óptima y un valor negativo en caso contrario.

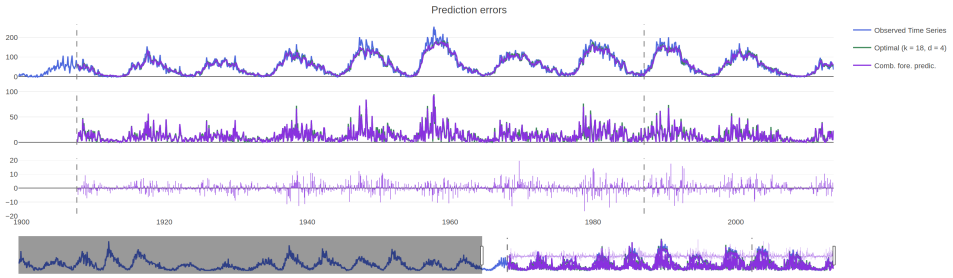


Figura 5.16: Serie temporal junto a la predicción mediante combinación

En la tabla de errores (figura 5.17) se mostrarán los errores *ME*, *MAE* y *RMSE* cometido por ambas prediciones tanto para el intervalo de *train* como para el de *test*. Cabe destacar que como se comentó en el apartado anterior, no se hará efectiva la predicción por combinación hasta que no se pulse el botón de la interfaz.

	Name	ME (train)	MAE (train)	RMSE (train)	ME (test)	MAE (test)	RMSE (test)
1	Optimal	1.6050076	13.015936	17.954289	1.0276053	12.645528	17.370869
2	Combinated kNNs	1.9180146	12.64983	17.573638	0.2494914	0.68870407	0

Figura 5.17: Errores de las predicciones seleccionadas

5.4. Detalles de implementación

Además de todas las modificaciones y novedades visuales comentadas en los apartados anteriores, también se han realizado cambios a nivel *interno* de la aplicación web. El principal, mencionado anteriormente en varias ocasiones a lo largo de los apartados relativos a la interfaz, es el cambio a la hora de actualizar todas las gráficas y, por ende, cómo es la interacción con ellas. El comportamiento que presentaba anteriormente la interfaz, en el que los gráficos y tablas cambiaban en cuanto se modificaba cualquier parámetro, se debía a que el servidor *Shiny* trata cualquier valor de entrada de la interfaz

como un *Valor Reactivo* [1]. Esto se traducía en una penalización de los fallos del usuario, ya que cualquier modificación se procesaba inmediatamente, con los consiguientes cálculos de predicción y cambios en la interfaz que ello conlleva. Además, en caso de querer modificar varios aspectos de la interfaz de forma simultánea, suponía dicho tiempo de espera después de cada cambio hasta llegar al resultado final buscado. Todo esto suponía un gran perjuicio a la *Experiencia de usuario*.

Para llevar a cabo el cambio en la aplicación se han añadido los correspondientes botones para confirmar los parámetros respectivos a cada grupo de gráficas. Gracias a estos, el usuario posee una mayor capacidad para configurar las diferentes opciones disponibles en cada método. Es decir, dada la gran cantidad de opciones disponibles, facilita al usuario poder confirmar un conjunto de cambios. Sin embargo, para que estos botones cumplan su función, ha sido necesario añadir *Reactive values*, de forma que se obtiene un flujo de datos correspondiente al paradigma de *Programación Reactiva*. Este paradigma consiste en un flujo de datos similar a un *pipeline*, en el que los datos *"avanzan"* por las diferentes etapas al cumplirse determinadas condiciones.

Este cambio a *Valores Reactivos* significa que los métodos que lean valores de este tipo se registrarán en el servidor como *observadores* de los mismos, y por tanto serán avisados y actualizados en cuanto se modifique dicho valor reactivo. Para evitar este comportamiento, se han creado una serie de valores reactivos agrupados en listas, que hacen de intermediarios entre los campos de parametrización presentes en la interfaz y los métodos que actualizan dicha interfaz. Concretamente, dichos valores intermedios se actualizan cuando se pulsa el correspondiente botón de confirmación, y son sólo estos valores los que son leídos por los diferentes métodos que actualizan las gráficas y tablas. De esta manera, se consigue que las funciones encargadas de generar los resultados mostrados por la interfaz sólo se actualicen al pulsar los respectivos botones.

Como añadido adicional, cuando se tienen diferentes elementos que utilizan la misma información para actualizarse (como por ejemplo las tablas de errores y los gráficos de predicción), es posible guardar la información relativa a la predicción en un valor reactivo utilizado por varios elementos de la interfaz. De esta forma se consigue poder realizar el cálculo una única vez sin perder la sincronización entre dichos elementos.

[1] Información disponible en el enlace: `https://shiny.rstudio.com/articles/reactivity-overview.html`

Capítulo 6

Resultados

Para hablar respecto a los resultados del presente trabajo se dividirá esta sección en dos partes diferenciadas aunque relacionadas. Por un lado, se tratarán los resultados de predicción obtenidos para varias series temporales mediante los diferentes métodos de combinación de predicciones ya comentados con anterioridad. Por otro lado, se comentarán los resultados observados respecto al estudio de las series temporales y la predicción mediante el *k-NN* con la aplicación web. Para realizar todo este proceso, se hará uso de un conjunto de series temporales de diferente naturaleza y magnitud que posteriormente se comentarán.

6.1. Metodología empleada

En primer lugar, respecto al tratamiento de cada serie temporal, se han dividido todas ellas de forma proporcional a su tamaño. Concretamente se ha usado el 60 % inicial de cada serie como *warm-up*, el 30 % siguiente como conjunto de entrenamiento, sobre el que buscar las mejores combinaciones de k, y d, y el 10 % restante como *test*. Este último conjunto proveerá datos no observados previamente, que permiten ver cómo se desenvuelve la combinación seleccionada mediante el intervalo de entrenamiento sobre nueva información.

Respecto a la parametrización de las funciones para generar los parámetros del método de *k-NN*, se ha empleado la misma configuración para todas ellas. Concretamente, los valores de 1 a 50 para la búsqueda de la combinación de k y d óptima, la distancia *manhattan* para la similitud de elementos, la ponderación de pesos *proporcional* a la distancia para el cálculo de la predicción y el error *RMSE* para medir sobre la predicción. Además por motivos de replicabilidad de los resultados, se han empleado *semillas* definidas manualmente para la generación de valores aleatorios, de forma que una misma configuración de un método sobre una serie, siempre da el mismo resultado.

Es importante destacar este último aspecto, ya que los resultados de predicción obtenidos para cada serie están claramente marcados por este apartado de la configuración. Cabe esperar que en una serie de pruebas verdaderamente aleatorias, realizadas de manera exhaustiva, se encontrasen resultados mejores que los aquí obtenidos. Esta suposición parte de la premisa de que las *semillas* empleadas se han definido manualmente de forma arbitraria. Por este motivo, se considera altamente improbable que éstas resulten ser las que generan las combinaciones de predicción óptimas para cada método. Por tanto, los resultados expuestos en la sección 6.2 deben ser tomados como una aproximación de las capacidades de los métodos desarrollados.

Por último, se han empleado distintos métodos para comparar el desempeño de los *Random Neighborhoods*. Además de los errores medios, se ha empleado el método estadístico *Diebold-Mariano* (Diebold y Mariano, 2002; Harvey, Leybourne y Newbold, 1997).

Este método busca comparar predicciones de series temporales de forma adimensional, por lo que los resultados de series de diferentes magnitudes pueden ser comparados. Concretamente se usarán los valores del estadístico *dm* y del *p-valor*. El primero indicará en qué medida una predicción es mejor que otra, mientras que el *p-valor* indicará si el valor anterior se puede considerar significativo. En este caso se empleará un umbral para este valor de 0,05, de forma que si las comparaciones presentan un *p-valor* superior, se considerará que ninguna de ellas realiza una predicción significativamente mejor.

Ambas métricas de error se han medido empleando la función `accuracy` para el error y la función `dm.test` para el estadístico, ambas del paquete `forecast` (Hyndman y Khandakar, 2008).

Las métodos de referencia utilizados para comparar el resultado de cada algoritmo son los siguientes:

- **Naive**: la predicción consiste en usar el último valor conocido.
- **Seasonal Naive**: la predicción consiste en utilizar el valor correspondiente al intervalo anterior. Por ejemplo, si una serie horaria tiene frecuencia semanal, se empleará el dato correspondiente a la misma hora y día de la semana anterior.
- **k-NN**: la predicción corresponde al resultado de la función `knn_past`, empleando el entrenamiento obtenido mediante la función `knn_param_search` aplicada al intervalo de entrenamiento.

6.2. Resultados de predicción

6.2.1. Serie manchas solares

La serie `sunspot.month` pertenece al paquete básico `datasets` proporcionado de forma nativa por el entorno de `R`. Esta serie corresponde a la estimación mensual de manchas solares observadas, desde enero de 1749 hasta septiembre de 2013, dando lugar a 3177 observaciones (fuente original: SILSO World Data Center (1749-2013)). Esta serie se considera estacional dado el comportamiento del Sol, que presenta un periodo aproximado de 11 años en el patrón de repetición de este fenómeno.

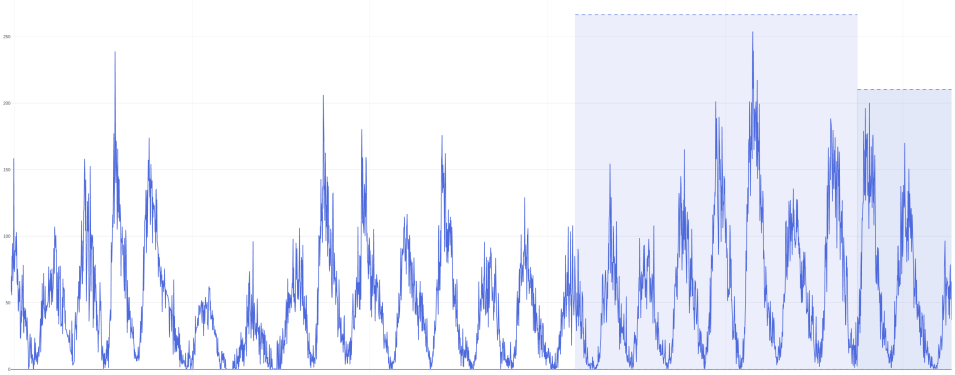


Figura 6.1: Serie `Sunspot.month` con intervalos de *train* y *test*

Para esta serie se han empleado 1906 observaciones como *warm-up*, 953 para el intervalo de entrenamiento y las 318 restantes para *test*. Las configuraciones empleadas para los diferentes métodos son las siguientes:

6.2.1.1. Ajuste de los métodos

En primer lugar se comentarán los parámetros establecidos para los métodos *Random Neighborhoods*. Se han entrenado con valores para k de 1 a 80 y d de 1 a 200. La configuración específica de cada método es la siguiente:

- **Bloques**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, y bloques de 50 a 200 observaciones consecutivas en incrementos de 50, conformando un total de 20 combinaciones.
- **Muestras**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, y una muestra de 50 a 200 observaciones en incrementos de 50, dando lugar a un total de 20 pruebas.
- **Fitness**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, filtrando o no únicamente al 80 % de las mejores

combinaciones, con un elitismo (mejores combinaciones) de 5 o 10 o sin él, filtrando o no por las combinaciones que son mínimos locales en un radio de 1 o 2, y usando o no selección por torneo para elegir las diferentes combinaciones de k y d. No se han realizado todas las posibles combinaciones de estos elementos ya que serían casi dos centenares, sino un subconjunto de ellas. En total se han probado 28 conjuntos de *learners*.

- **Mínimos**: se han empleado radios de 1 o 2 combinaciones de k y d adyacentes en la matriz de errores. En el caso del radio 1 se han limitado los modelos de salida con los valores 5, 10, 15, 20, 25, 30, 35, 40, 45, 47. Respecto al radio 2 se han probado los valores $2, 4, 5, 6, 8, 10$. Ambos límites superiores se corresponden al número de mínimos locales presentes en cada caso. En este caso se han probado un total de 16 conjuntos de *learners*.

Por otro lado, los ajustes empleados para los métodos de referencia son los siguientes:

- **Seasonal Naive**: dado el periodo de estacionalidad de aproximadamente 11 años comentado anteriormente, se han probado los valores de *lags* desde 12 (1 año) hasta 156 (13 años). El que ha minimizado el error en entrenamiento es 124, correspondiente a 10 años y 4 meses, cercano al periodo aproximado que se estima para este fenómeno.
- **k-NN**: mediante la función `knn_param_search`, probando los valores de k y d de 1 a 50, se ha obtenido que la combinación óptima en el intervalo de entrenamiento corresponde a **18 vecinos** conformados por **4 observaciones** consecutivas.

6.2.1.2. Resultados de *Random Neighborhoods*

Método	Train		Test		
	Peor	Mejor	Peor	Seleccionado	Mejor
Bloques	17,55	17,11	16,76	16,38	15,98
Muestras	17,08	16,61	16,59	16,25	16,22
Fitness	18,99	16,68	17,50	16,65	16,42
Mínimos	17,72	16,63	18,69	16,58	16,31

Tabla 6.1: Errores de *Random Neighborhoods*, serie *Sunspot*

En la tabla 6.1 se encuentra el resumen de los errores *RMSE* para cada método de *Random Neighborhoods*, medidos tanto en el intervalo de entrenamiento como el de *test*. La columna **Seleccionado** corresponde al error, en

el intervalo de *test*, de la mejor combinación en el intervalo de entrenamiento. A modo ilustrativo se muestra el mínimo error observado en el intervalo de *test*, a fin de analizar la diferencia entre el desempeño de la combinación seleccionada y la mejor encontrada en las pruebas.

En todos los casos se puede observar que el error en *test* de las combinaciones seleccionadas de cada *Random Neighborhoods* está en el promedio de errores de dicho intervalo, aunque más próximos al mínimo observado en *test*. También cabe destacar cómo los errores mínimos alcanzados en entrenamiento por tres de los métodos son muy similares, aunque luego no se repite dicha tendencia en *test*. Ya que *Muestras* consigue el mejor resultado pero los otros dos métodos presentan un error muy similar al del intervalo de entrenamiento. Por último, cabe destacar que en el intervalo de *test* el mínimo observado (*Bloques*), es sensiblemente mejor que los otros tres.

6.2.1.3. Comparación de resultados

Método	***Train***	***Test***
Bloques	17,11	16,38
Muestras	16,61	16,25
Fitness	16,68	16,65
Mínimos	16,63	16,58
Naive	18,02	18,69
S. Naive	35,81	43,25
k-NN	17,75	18,91

Tabla 6.2: Comparación de errores *RMSE* en la serie `Sunspot`

En primer lugar en la tabla 6.2 se resumen los errores en los intervalos de entrenamiento y *test* de los diferentes métodos. En todos los casos se han seleccionado los predictores que den mejor resultado en el intervalo de entrenamiento. Como se puede ver todos los métodos de combinación de predicciones mejoran todos los métodos de referencia en el intervalo de *test*, siendo en el caso del método *k-NN* básico concretamente hasta un 15 % de mejora. Cabe destacar que el *k-NN* ha obtenido un resultado sensiblemente peor en el intervalo de *test* que en el entrenamiento, justo al opuesto que en los *Random Neighborhoods*. Esto hace pensar que el *k-NN* base ha sufrido un sobreajuste mientras que los métodos de ensamblado no.

Por otro lado, como se comentó anteriormente, para comparar las diferentes predicciones se empleará el estadístico del método *Diebold-Mariano*. Los resultados están presentes en la Tabla 6.3. Las posiciones marcadas con * indican que según el *p-valor* no se aprecian diferencias suficientemente significativas entre predicciones.

Como se puede observar los 4 métodos de predicción superan a los de referencia sencillos y al *k-NN* básico. Sin embargo en dos casos de entrenamiento, respecto al *k-NN* base, las diferencias de predicción no son significativas, aunque la del método de *Bloques* está en el límite de serlo. De estos resultados podemos concluir que el *k-NN* básico ha sufrido de un sobreajuste en entrenamiento al amplicar los valores de exploración de k y d. Sin embargo esto no les ocurre a los *Random Neighborhoods*.

Método	**Entrenamiento**			**Test**		
	Naive	**S. Naive**	**k-NN**	**Naive**	**S. Naive**	**k-NN**
Bloques	2,1518	13,255	1,9583*	2,8808	9,0232	4,2881
Muestras	3,9351	13,375	4,1642	3,512	9,0364	4,4852
Fitness	3,9513	13,322	4,7302	3,044	8,9695	4,3322
Mínimos	1,8339	13,22	1,4169*	2,2786	8,9682	3,0957

Tabla 6.3: Estadístico comparativo *DM* para la serie *Sunspot*

6.2.2. Serie demanda eléctrica

La serie `taylor` pertenece al paquete `forecast`. Esta serie consiste en el muestreo con una frecuencia de media hora de la demanda de energía eléctrica medida en megavatios, en Inglaterra y Gales, desde el 5 de junio al 27 de agosto del año 2000, dando lugar a 4032 observaciones. La serie presenta dos periodos estacionales de 48 y 336, correspondientes a un día y una semana respectivamente.

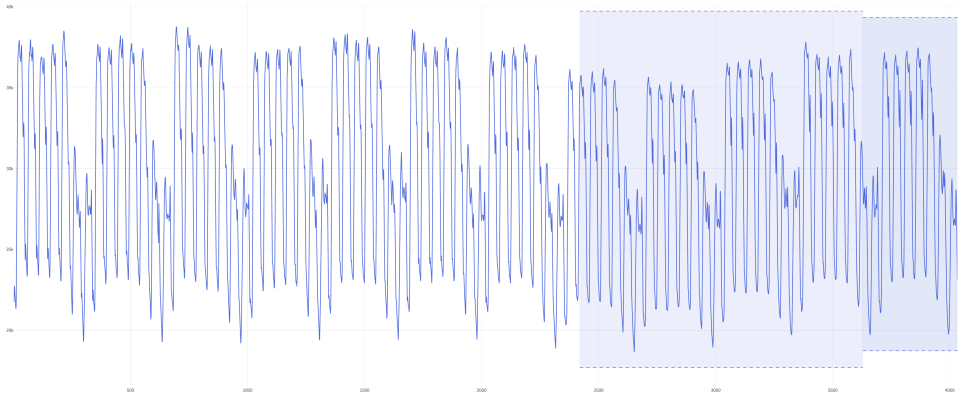


Figura 6.2: Serie `taylor` con intervalos de *train* y *test*

Para esta serie se han empleado 3024 observaciones como *warm-up*, 604 para el intervalo de entrenamiento y las 404 restantes para *test*. Las configuraciones empleadas para los diferentes métodos son las siguientes:

6.2.2.1. Ajuste de métodos

En primer lugar se comentarán los parámetros establecidos para los métodos *Random Neighborhoods*. En todos los casos se han definido unos valores para k y d de 1 a 50. La configuración para cada método es la siguiente:

- **Bloques**: se han entrenado combinaciones de 10 a 50 *learners* en incrementos de 10, y observaciones consecutivas en bloques de 50 a 200 con incrementos de 50, conformando un total de 20 combinaciones.
- **Muestras**: se han empleado un total de 10 a 50 *learners* en incrementos de 10, y una muestra de 50 a 200 observaciones en incrementos de 50, formando en total 20 pruebas.
- **Fitness**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, filtrando o no únicamente al 80 % de las mejores combinaciones, con un elitismo (mejores combinaciones) de 5 o 10 o sin él, filtrando o no por las combinaciones que son mínimos locales en un radio de 1 o 2, y usando o no selección por torneo para elegir las diferentes combinaciones de k y d. En total se han probado 28 conjuntos de *learners*.
- **Mínimos**: se han empleado radios de 1 o 2 combinaciones de k y d adyacentes en la matriz de errores. En el caso del radio 1 se han limitado los modelos de salida con los valores 2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18, 20, 22. Respecto al radio 2 se han probado los valores 2, 4, 5, 6, 8, 9. Ambos límites superiores se corresponden al número de mínimos locales presentes en cada caso. Se han probado un total de 19 conjuntos de *learners*.

Respecto a los métodos de referencia, su configuración es la siguiente:

- **Seasonal Naive**: se han probado los valores de *lags* desde 12 (6 horas) hasta 2300 (algo más de mes y medio). El que ha minimizado el error en entrenamiento es 2016, correspondiente a 42 días.
- **k-NN**: mediante la función `knn_param_search` se ha obtenido que el mejor modelo es de **3 vecinos** formados por **6 observaciones** consecutivas.

6.2.2.2. Resultados de *Random Neighborhoods*

Método	Train		Test		
	Peor	Mejor	Peor	Seleccionado	Mejor
Bloques	409,61	337,977	415,76	343,42	333,77
Muestras	366,571	337,848	344,36	336,13	332,598
Fitness	474,426	334,007	494,73	334,18	330,878
Mínimos	387,08	337,68	393,45	344,72	331,66

Tabla 6.4: Errores de *Random Neighborhoods*, serie *Taylor*

En la tabla 6.4 se encuentra el resumen de los errores *RMSE* tanto en intervalo de entrenamiento como el de *test* para cada método de *Random Neighborhoods*. La columna **Seleccionado** corresponde al error de la mejor combinación en el intervalo de entrenamiento, pero medido en el intervalo de *test*. También se muestra el mínimo error observado, para cada método, en el intervalo de *test* a modo ilustrativo. Esto se ha realizado con el fin de analizar la diferencia entre el desempeño de la combinación seleccionada y la mejor encontrada en las pruebas.

En todos los casos se puede observar que el error en *test* de las combinaciones seleccionadas de cada *Random Neighborhoods* están muy cercanas al mínimo observado en dicho intervalo. Por otra parte, se puede observar cómo los errores mínimos alcanzados en entrenamiento por todos los métodos es muy similar, destacando ligeramente el método *Fitness* de selección basado en el algoritmo genético (334,18) sobre el de *Bloques* (343,42). Para el intervalo de *test* los errores nuevamente son muy similares dos a dos. Sin embargo, esta vez las mejores combinaciones pertenecen al algoritmo de *bagging* de muestreo disperso (336,13) y al de *Fitness* (334,18).

6.2.2.3. Comparación de resultados

Método	*Train*	*Test*
Bloques	337,977	343,421
Muestras	337,848	336,139
Fitness	334,007	334,186
Mínimos	342,009	354,319
Naive	943,93	888,12
S. Naive	850,21	872,39
k-NN	440,62	415,99

Tabla 6.5: Comparación de errores *RMSE* en la serie `Taylor`

En la tabla 6.5 se resumen los errores de todos los métodos, medidos en los intervalos de entrenamiento y *test*. En todos los casos se han seleccionado los predictores que han minimizado el error en el intervalo de entrenamiento. Como se puede observar todos los métodos de combinación de predicciones mejoran todos los métodos de referencia en el intervalo de *test*, siendo en el caso del método *k-NN* básico una mejora de hasta un 20 % menor error.

Por otro lado, como se comentó anteriormente, para comparar las diferentes predicciones se empleará el estadístico del método *Diebold-Mariano*. Los resultados están presentes en la Tabla 6.6.

Como se puede observar los 4 métodos de predicción superan a los de referencia sencillos y al *k-NN* básico. Respecto al *k-NN* base destacan el método de *Muestras* y *Fitness*, siendo el primero mejor en el intervalo de *test* y el segundo en entrenamiento.

Método	**Entrenamiento**			**Test**		
	Naive	**S. Naive**	**k-NN**	**Naive**	**S. Naive**	**k-NN**
Bloques	12,24	15,27	5,18	9,83	8,74	4,62
Muestras	12,35	14,84	6,16	9,97	8,69	5,95
Fitness	12,38	14,92	6,25	9,99	8,67	5,13
Mínimos	12,34	15,11	5,22	9,85	8,72	3,83

Tabla 6.6: Estadístico comparativo *DM* para la serie *Taylor*

6.2.3. Serie precipitaciones

La serie `rain` pertenece al paquete `ismev`. Esta serie consiste en 17531 observaciones que representan las precipitaciones acumuladas diarias en una localidad al suroeste de Inglaterra, medida durante el periodo de 1914 a 1962. Presenta una estacionalidad anual.

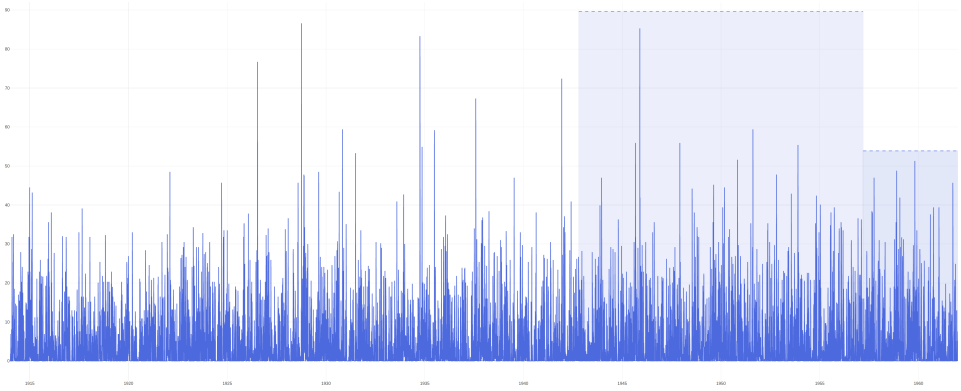


Figura 6.3: Serie `rain` con intervalos de *train* y *test*

Para esta serie se han empleado 10518 observaciones como *warm-up*, 5259 para el intervalo de entrenamiento y las 1754 restantes para *test*. Las configuraciones empleadas para los diferentes métodos son las siguientes:

6.2.3.1. Ajuste de métodos

Los métodos de *Random Neighborhoods* se han entrenado con valores para k y d de 1 a 50. La configuración específica de los métodos es la siguiente:

- **Muestras**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, y una muestra de 50 a 200 observaciones en incrementos de 50, dando lugar a un total de 25 pruebas.
- **Bloques**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, y bloques de 50 a 200 observaciones consecutivas en incrementos de 50, conformando un total de 20 combinaciones.
- **Fitness**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, filtrando o no únicamente al 80 % de las mejores combinaciones, con un elitismo (mejores combinaciones) de 5 o 10 o sin él, filtrando o no por las combinaciones que son mínimos locales en un radio de 1 o 2, y usando o no selección por torneo para elegir las diferentes combinaciones de k y d. En total se han probado 28 conjuntos de *learners*.
- **Mínimos**: se han empleado radios de 1 o 2 combinaciones de k y d adyacentes en la matriz de errores. En el caso del radio 1 se han limitado los modelos de salida con los valores 5, 10, 15, 20, 25, 30, 40, 50, 55, 60, 65, 70, 75, 80, 83. Respecto al radio 2 se han probado los valores 2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18, 20, 21. Ambos límites superiores se corresponden al número de mínimos locales presentes en cada caso. En este caso dada la elevada cantidad de mínimos locales se han probado un total de 28 conjuntos de *learners*.

Para los métodos de referencia se han utilizado los siguientes ajustes:

- **Seasonal Naive**: se han probado los valores de *lags* desde 7 (1 semana) hasta 10500 (aproximadamente 28 años y 9 meses). El que ha minimizado el error en entrenamiento es 10490, correspondiente a 28 años y aproximadamente 8 meses.
- **k-NN**: mediante la función `knn_param_search` en el intervalo se ha encontrado que la combinación óptima será de **50 vecinos** conformados por **42 observaciones** consecutivas.

Método	Train		Test		
	Peor	Mejor	Peor	Seleccionado	Mejor
Bloques	6,359	6,199	6,26	6,131	6,120
Muestras	6,467	6,212	6,46	6,143	6,142
Fitness	6,335	6,213	6,25	6,155	6,148
Mínimos	6,413	6,198	6,34	6,137	6,129

Tabla 6.7: Errores de *Random Neighborhoods*, serie *Rain*

6.2.3.2. Resultados de *Random Neighborhoods*

En la tabla 6.7 se encuentra el resumen de los errores *RMSE* medidos en el intervalo de entrenamiento y de *test*, todos correspondientes a los métodos de *Random Neighborhoods*. La columna **Seleccionado** corresponde al error de la combinación óptima observada en el intervalo de entrenamiento, pero en el intervalo de *test*. A fin de analizar el desempeño en el intervalo de *test* de dicha combinación seleccionada, se muestra a modo ilustrativo el mínimo error observado en el intervalo de *test* en cada método.

En todos los casos se puede observar que el error en *test* de las combinaciones seleccionadas de cada *Random Neighborhoods* está muy cerca del mínimo observado de dicho intervalo. De hecho en ningún caso hay más de tres combinaciones mejores que con errores comprendidos entre el seleccionado y el mínimo. En este caso los métodos que ofrecen mejores resultados, tanto en entrenamiento como en *test*, han sido el basado en *bagging* con entrenamiento mediante intervalos de la serie (6,131) y el de selección de los mínimos locales (6,137).

6.2.3.3. Comparación de resultados

Método	*Train*	*Test*
Bloques	6,199	6,131
Muestras	6,212	6,143
Fitness	6,213	6,155
Mínimos	6,265	6,197
Naive	7,886	7,663
S. Naive	8,594	9,23
k-NN	6,419	6,328

Tabla 6.8: Comparación de errores *RMSE* en la serie `Rain`

En primer lugar en la tabla 6.8 se resumen los errores en los intervalos de entrenamiento y *test* de los diferentes métodos. En todos los casos se han seleccionado los predictores que den mejor resultado en el intervalo de

entrenamiento. Como se puede ver todos los métodos de combinación de predicciones mejoran todos los métodos de referencia en el intervalo de *test*, aunque la mejora respecto al *k-NN* básico es sólo de un 3 % como máximo.

Por otro lado, como se comentó anteriormente, para comparar las diferentes predicciones se empleará el estadístico del método *Diebold-Mariano*. Los resultados están presentes en la Tabla 6.9. En esta ocasión todos los resultados son significativo de acuerdo al *p-valor*.

Como se puede observar los 4 métodos de predicción superan a los de referencia sencillos y al *k-NN* básico. En esta ocasión el método que maximiza el estadístico es *Fitness* en ambos intervalos.

Método	**Entrenamiento**			**Test**		
	Naive	**S. Naive**	**k-NN**	**Naive**	**S. Naive**	**k-NN**
Bloques	10,32	16,92	8,50	6,05	8,15	4,45
Muestras	10,18	16,86	9,03	5,95	8,13	4,78
Fitness	10,13	16,91	10,05	5,88	8,12	5,06
Mínimos	10,33	16,92	8,37	6,04	8,13	4,01

Tabla 6.9: Estadístico comparativo *DM* para la serie *Rain*

6.2.4. Serie calidad del aire

Se ha empleado la serie de mediciones horarias de la cantidad de Dióxido de Nitrógeno presente en el aire en la zona del Paseo de la Castellana. Esta serie consiste en 21912 observaciones, correspondientes al periodo de tiempo comprendido desde Octubre del 2017 a Marzo del 2020, medidas por la estación de la Comunidad de Madrid presente en dicha zona. Dado que esta estación ha presentado algunos fallos, y por tanto había observaciones no disponibles, se han estimado dichos valores perdidos. El cálculo realizado ha sido hallar la media del último valor válido, el valor de la misma hora del día anterior y el valor de la misma hora de la semana anterior. Concretamente había 38 valores no disponibles que han sido estimados mediante este método. Esta serie presenta una estacionalidad de 24 observaciones, correspondiente a un día, ya que las tasas de coches que circulan responden a las entradas y salidas laborales.

Para esta serie se han empleado 13147 observaciones como *warm-up*, 6573 para el intervalo de entrenamiento y las 2192 restantes para *test*. Las configuraciones empleadas para los diferentes métodos son las siguientes:

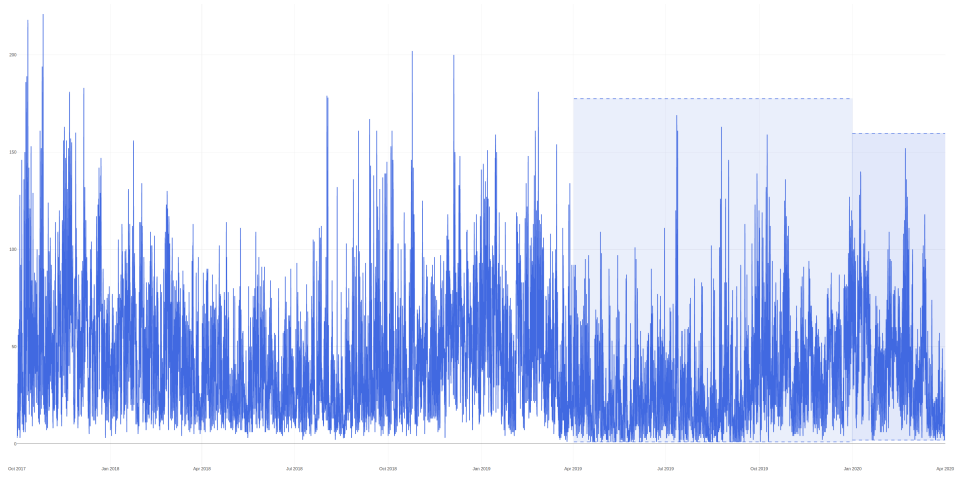


Figura 6.4: Serie de calidad del aire con intervalos de *train* y *test*

6.2.4.1. Ajuste de métodos

En primer lugar se comentarán los parámetros establecidos para los métodos *Random Neighborhoods*. Se han probado unos valores para k y d de 1 a 50. La configuración de los parámetros específicos de cada método es la siguiente:

- **Muestras**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, y una muestra de 50, 100, 150, 200, 300 y 400 observaciones en incrementos de 50, dando lugar a un total de 30 pruebas.
- **Bloques**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, y bloques de 50, 100, 150, 200, 300 y 400 observaciones consecutivas, conformando un total de 30 combinaciones.
- **Fitness**: se han empleado combinaciones de 10 a 50 *learners* en incrementos de 10, filtrando o no únicamente al 80 % de las mejores combinaciones, con un elitismo (mejores combinaciones) de 5 o 10 o sin él, filtrando o no por las combinaciones que son mínimos locales en un radio de 1 o 2, y usando o no selección por torneo para elegir las diferentes combinaciones de K y D. En total se han probado 28 conjuntos de *learners*.
- **Mínimos**: se han empleado radios de 1 o 2 combinaciones de k y d adyacentes en la matriz de errores. En el caso del radio 1 se han limitado los modelos de salida con los valores 2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18, 20, 22, 24. Respecto al radio 2 se han probado los valores $2, 4, 5, 6, 8$. Ambos límites superiores se corresponden al número de mínimos locales

presentes en cada caso. En este caso se han probado un total de 19 conjuntos de *learners*.

- **Seasonal Naive**: se han probado los valores de *lags* desde 6 (6 horas) hasta 9000 (1 año y 10 días). El que ha minimizado el error en entrenamiento es 24 horas. Este valor se corresponde con una estacionalidad diaria.
- **k-NN**: mediante la función `knn_param_search` se ha obtenido que el modelo óptimo es el de **30 vecinos** conformados por **4 observaciones** consecutivas.

6.2.4.2. Resultados de *Random Neighborhoods*

Método	Train		Test		
	Peor	**Mejor**	**Peor**	**Seleccionado**	**Mejor**
Bloques	10,261	10,049	9,42	9,18	9,17
Muestras	10,21	10,049	9,38	9,20	9,20
Fitness	10,39	10,024	9,49	9,21	9,21
Mínimos	10,43	10,032	9,43	9,15	9,15

Tabla 6.10: Errores de *Random Neighborhoods*, serie *Aire*

En la tabla 6.10 se encuentra el resumen de los errores *RMSE* en los diferentes intervalos para cada método de *Random Neighborhoods*. La columna **Seleccionado** corresponde al error, medido en el intervalo de *test*, de la combinación con mejor resultado en el intervalo de entrenamiento. A modo ilustrativo se muestra el mínimo error observado en el intervalo de *test*, como medida de referencia del mejor resultado encontrado en las pruebas.

En esta ocasión se puede observar que en todos los casos, salvo para el método *Bloques*, la combinación que ha obtenido el mejor resultado en la fase de entrenamiento coincide con la combinación que lo obtiene en la fase de *test*. Cabe destacar que en el caso del método de *Bloques* dicha combinación (9,18) es sólo la segunda mejor de las observadas en dicho método.

6.2.4.3. Comparación de resultados

En primer lugar en la tabla 6.11 se resumen los errores en los intervalos de entrenamiento y *test* de los diferentes métodos. En todos los casos se han seleccionado los predictores que den mejor resultado en el intervalo de entrenamiento. Nuevamente todos los métodos de combinación de predicciones mejoran métodos de referencia tanto en el intervalo de entrenamiento como el de *test*, siendo en el caso del método *k-NN* básico concretamente de un 3 % de mejora como máximo.

Método	***Train***	***Test***
Bloques	10,04	9,18
Muestras	10,04	9,20
Fitness	10,02	9,21
Mínimos	10,03	9,15
Naive	11,12	10,15
S. Naive	23,0	23,33
k-NN	10,44	9,44

Tabla 6.11: Comparación de errores *RMSE* en la serie `aire`

Por otro lado, como ya se ha comentado con anterioridad, para comparar las diferentes predicciones se empleará el estadístico del método *Diebold-Mariano*. Los resultados están presentes en la Tabla 6.12. En esta ocasión de nuevo todos los resultados son significativos según el *p-valor* asociado.

Como se puede observar los cuatro métodos de predicción superan a los de referencia sencillos y al *k-NN* básico. Dentro de ellos el que ha presentado un mayor valor en el estadístico, tanto en entrenamiento como en *test*, es el de *Bloques*.

Método	**Entrenamiento**			**Test**		
	Naive	**S. Naive**	**k-NN**	**Naive**	**S. Naive**	**k-NN**
Bloques	9,9928	25,438	7,6305	6,7004	21,172	3,6816
Muestras	9,5668	25,435	6,3429	6,275	21,194	3,05155
Fitness	10,13	25,587	7,2129	6,4016	21,317	2,9733
Mínimos	9,2673	25,633	6,5939	6,4745	21,41	3,5167

Tabla 6.12: Estadístico comparativo *DM* para la serie *Aire*

6.3. Tiempos de ejecución y consumo de memoria

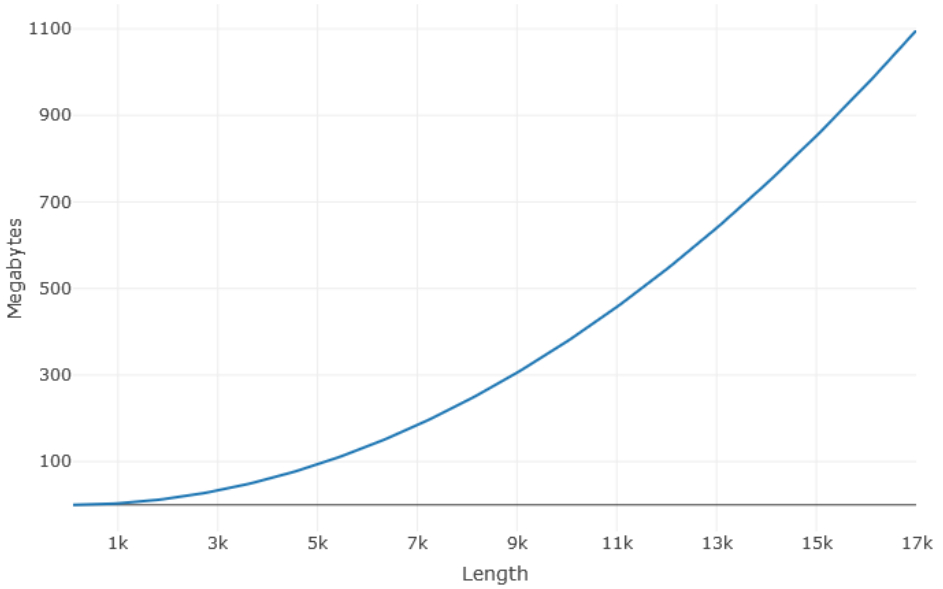


Figura 6.5: Consumo de memoria en función de la longitud de la serie

Respecto al consumo de memoria, el aspecto clave que afecta a este aspecto son las distancias entre *elementos* de la serie temporal. Concretamente el consumo crece de forma cuadrática. Esto se debe a que por cada observación de la serie temporal, hay que calcular tantas distancias como observaciones le precedan. En la figura 6.5 se muestran las mediciones realizadas del consumo de memoria a medida que se incrementa la longitud de la serie.

Por otro lado, también se han realizado pruebas de medición de los tiempos necesarios para el entrenamiento de cada método en cada serie. Para ello se ha hecho uso de un ordenador portátil con las siguientes características:

- Procesador: *Intel Core i7-8750H* (6 *cores* físicos, 12 lógicos)
- Memoria *RAM*: 12 GB
- Sistema operativo: *Windows 10*

Para estandarizar todas las pruebas y hacerlas comparables, se estableció el parámetro de paralelización `n_threads` a 6 hilos concurrentes. Sin embargo la última serie, la de calidad del aire, no ha sido imposible incluirla en las pruebas. Esto es debido a que esta serie se incluyó posteriormente con las demás pruebas ya realizadas, y la configuración establecida para medir todas las demás era incompatible.

El principal problema resultó ser memoria RAM, ya que cada hilo de ejecución requería más de 1GB y al restar la memoria necesaria por el sistema operativo, se producía *swap* a disco y el rendimiento se degradaba. En caso de ocurrir esto, es recomendable reducir la cantidad de hilos hasta el máximo que no genere *swappìng*.

Para la configuración de los parámetros de las funciones entrenamiento de los *Random Neighborhoods* se establecieron los siguientes valores:

- `ks` y `ds`: valores a estudiar de 1 a 50.
- `learners`: se probaron de 10 a 50 predictores, aumentando de 10 en 10.
- Longitud de entrenamiento: correspondiente a los parámetros `train_length` y `n_instants`. Se han probado los valores 50, 100, 150 y 200.

Cabe destacar que los tiempos de ejecución se ven afectados por la longitud de entrenamiento pero sobre todo por la cantidad de *learners*. Por este motivo, se simplificarán los resultados expuestos y sólo se comentarán los resultados correspondientes a longitudes de entrenamiento de 200 y generación de 50 *learners*.

Serie	**Bloques**	**Muestras**	**Fitness**	**Mínimos**
Sunspot	750	670	25	25
Taylor	650	860	35	35
Rain	10.000	15.000	530	560

Tabla 6.13: Tiempos de ejecución en segundos de *Random Neighborhoods*

Como se puede ver en la tabla 6.13 la diferencia entre métodos de *Random Neighborhoods* es sustancial. Los métodos que requieren más tiempo son los basados en *bagging*. Esto es debido a que en el proceso de entrenamiento no se comparte información entre cada *learner*, sino que los datos comunes entre ellos se re-calculan por cada uno. Sin embargo, dado que el proceso de entrenamiento de estos métodos para generar 50 *learners* requiere 50 entrenamientos, estos resultados son positivos. Si se multiplican los tiempos de entrenamiento de *Fitness* o *Mínimos*, que sólo realizan un entrenamiento, por 50 *learners* generados, daría un resultado de más de 26000 segundos. Por tanto aunque los tiempos sean sensiblemente superiores, se consideran bastante optimizados.

6.4. Discusión de resultados

Para terminar se realizará un discusión acerca de los resultados obtenidos. Al comparar los errores de predicción, no cabe lugar a duda que los cuatro métodos de *Random Neighborhoods* superan los métodos de referencia en todas las series, en especial a *Naive* y *S. Naive*, dado que son métodos muy sencillos. Respecto al *k-NN* base las diferencias de mejora varían según la serie, siendo la de consumo eléctrico donde se presentan las mayores diferencias.

Posición	**Sunspot**	**Taylor**	**Rain**	**Aire**
1º	Muestras	Fitness	Bloques	Mínimos
2º	Bloques	Muestras	Mínimos	Bloques
3º	Mínimos	Bloques	Muestras	Muestras
4º	Fitness	Mínimos	Fitness	Fitness

Tabla 6.14: Métodos ordenados según *RMSE* en *test*

En la tabla 6.14 se han ordenado los métodos de *Random Neighborhoods* en función de el *RMSE* cometido en el intervalo de *test*. Como se puede observar, todos los métodos han quedado en primera posición en una ocasión. Sin embargo el método de *Fitness* ha quedado en último lugar en tres de las cuatro ocasiones.

Dados los resultados, se considera que los cuatro métodos tienen el potencial de destacar en función de las características de la serie. Por este motivo, no hay razones concluyentes para pensar que alguno es mejor que el resto. Sin embargo, sí cabe apreciar que el método de *Fitness*, basado en el algoritmo genético, parece tener alguna limitación para superar al resto, no así para superar al *k-NN* base.

Capítulo 7

Conclusiones

En primer lugar comentar los objetivos respecto a la predicción de series temporales mediante ensamblado, los *Random Neighborhoods*. Este objetivo se considera plenamente alcanzado, ya que se han realizado cuatro implementaciones de entrenamiento partiendo de los dos enfoques diferentes, el *bagging* y la selección inspirada en los algoritmos genéticos. Además como se ha visto en el capítulo 6 de resultados, todas las implementaciones han logrado superar la predicción mediante el *k-NN* en todas las series planteadas.

Por otro lado se ha logrado emplear la paralelización en hilos concurrentes en todos los algoritmos, de forma que se puedan aprovechar los diferentes *cores* que, cada vez más, van teniendo los procesadores actuales. Como se ha comentado en la sección 6.3, la paralelización empleada en los métodos de *Bloques* y *Muestras*, aunque lejos de ser la óptima, es bastante buena.

Asimismo se ha mejorado y ampliado la aplicación web ya existente. Para ello se han añadido nuevas gráficas y ampliado las ya existentes, así como nuevos elementos de parametrización para ofrecer más opciones al usuario. Ahora se presentan los k-vecinos de una determinada predicción así como sus métricas asociadas. También se permite estudiar el desempeño de cada combinación de k y d, incluyendo el cálculo de los *mínimos locales*. Además se ha creado una nueva pestaña dedicada a los *Random Neighborhoods* y el ensamblado en general, pudiendo generara combinaciones manualmente en función de la información extraída mediante la herramienta.

Por otro lado se ha mejorado la experiencia de usuario, permitiendo un flujo de trabajo más cómodo mediante los botones de confirmación. También se ha optimizado el reaprovechamiento de información entre elementos de la interfaz que la compartan.

Dado lo novedoso del trabajo, se pretende llevar el presente trabajo más allá publicando de dos artículos. Uno versaría sobre el ensamblado de predicciones mediante *Random Neighborhoods* y el otro sobre la explicabilidad del *k-NN* como método de predicción de series temporales.

Trabajo futuro

Para dichos artículos, es necesario ampliar las pruebas realizadas a más series con una mayor componente no lineal. De esta forma se podría comprobar el potencial de los *Random Neighborhoods*. Respecto a la *explicabilidad*, sería interesarte buscar más series donde explotar las métricas y funcionalidades de la aplicación web. De esta forma se podrían encontrar fenómenos que puedan presentarse en las series.

Además se plantean varios objetivos previos de mejora de las implementaciones. Por un lado se plantea la optimización de los *Random Neighborhoods* basados en *bagging*, *Bloques* y *Muestras*. Para ello sería necesario reformular los métodos para reaprovechar al máximo el cálculo de todas las distancias así como de las predicciones. Esto es así porque actualmente durante el proceso de entrenamiento de cada modelo, éstos no comparten información entre sí.

También respecto a dichos métodos de *Random Neighborhoods*, se plantea la posibilidad de crear un sistema de ponderación de cada modelo por similitud a lo entrenado. Es decir, al realizar una predicción mediante ensamblado, asignar un peso a cada modelo en función de lo parecido que sea el dato a predecir con los datos empleados para generar cada modelo.

Chapter 8

Introduction

8.1. Motivation

In today's world, given the high degree of digitalization that is being achieved, there is an increasing amount of data available to work on in order to draw conclusions. One approach to data processing is the prediction of new observations, such as meteorology focused on rainfall or air pollution, the supply of either basic household goods or services, or the extent to which a disease may spread in order to prepare the necessary resources.

Within the prediction of these data there is a field that is the time series, on which this work is focused. For this reason the first step is to define a time series. In general terms, a time series is a set of data related to one or several metrics that are measured over time. We can therefore divide a time series into two parts:

- Time instants: they represent the moment in time when each of the measurements was made. Examples include the time of day or the date on the calendar. They are denoted as $\{t_1, t_2, \ldots, t_{n-1}, t_n\}$, where n is the number of observations.
- Observed values: represent the data associated with the different measured metrics. They are denoted as $\{y_1, y_2, \ldots, y_{n-1}, y_n\}$, and there will be as many as different metrics in the time series.

It is said that a series is *stationary* (Peña y Sánchez, 2007) when it is stable over time, that is, when its values oscillate around a fixed level and also has a constant variability. If this is not the case, then the series is *non-stationary*. In such a case, some of these series could present a clear trend. On the other hand, there could be a particular case of non-stationary series in which the values oscillate around a central value, although they have

a different behaviour depending on the time period (years, months, days). This type of series is called *seasonal* series.

Time series forecasting consists of using past values to obtain the next value of the series or a future value of the series at a given *time horizon*. That is, the time distance that exists between the current instant and the one to be predicted. To make such a forecast, as in any field of prediction or classification, there are multiple techniques and approaches which will be discussed below.

An example of these are the smoothing techniques (Brown, 1963), in which a moving average of the last values of the series is used, performing a weighting where the most recent values have more weight. Another type of technique is the autoregressive (Box, Jenkins, Reinsel y Ljung, 2011), which generates the forecast as a linear regression of past values of the series.

There are also very sophisticated approaches using *machine learning* techniques, such as multi-layer perceptrons (Haykin, 1998). In this case the autoregressive methodology is used, but it is improved with the flexibility of the perceptron to detect the lack of linearity in the regression.

However, in many cases machine learning techniques tend to present a behavior of *black box* type. That is, the methods receive an input and generate a certain output, but it is not possible to know why that output has been generated or how it has been obtained. For this reason, research is being done very actively on how to make the machine learning techniques explainable. This will be discussed in detail in the next chapter.

Among the machine learning techniques is the *k-Nearest Neighbor* method (henceforth referred to as *k-NN*). This technique consists of comparing the data you want to predict with all the rest. From all of them, the k that has the most similarity will be chosen. Finally, the prediction will be made according to the value that took the variable to predict in each one of these neighbors.

Specifically, this was done during the End of Grade Project. In it, an implementation of time series forecasting is carried out by *k-NN*. In addition, the training process is carried out to select the optimal model. Finally, a basic web application was developed to exploit the forecast and training explainability. Depending on the developed implementation, given a certain moment in time to be forecasted, this forecast is composed of two main parameters:

- ***d***: d consecutive time instants will be grouped together, so that their last instant is the one to be predicted. This grouping will form an *element*. Let y_t be a temporary instant, its corresponding element of length d would be: $\{y_{t-d+1}, \ldots, y_{t-1}, y_t\}$. These elements, and not the discrete time instants, will be used by the *k-NN* to carry out the computations that give rise to the forecast.

- ***k***: the distances between the *element* formed by the instant by which it is desired to predict, and the rest of the elements previous to it in the series, will be calculated. By means of these distances, the elements with the k shortest distances will be chosen and the forecast will be carried out with them.

These two parameters will be the only ones that will characterize a certain forecasting model of the *k-NN*. This is because, although it is counter-intuitive, *k-NN* is a learning method that does not learn through a predictive model. In other words, the *k-NN* method does not generate a model through training, as is the case for example with the different neurons in a neural network, but instead needs to load all the *elements* in memory in order to select the k best.

The *k-NN* as a learning technique is a method that, despite its simplicity, performs well and lends itself to explainability and transparency. This is mainly due to the fact that it is always possible to identify which are the k selected neighbors to make the forecast. In addition, it is possible to check whether all of them were really similar to the current one and whether the value they had to predict was useful for the prediction.

On one hand, with regard to its transparency, the *k-NN* method both as a predictor and a classifier, is not usually developed with these characteristics in mind. That is to say, although the possibility of inspecting how the prediction has been carried out exists, this characteristic is rarely exploited. In general, results are shown as an aggregated mean of error and no further information is provided in this respect.

Therefore, it is proposed to address this section from several points of view. Firstly, to know and study the k neighbours that generate a forecast and what characteristics they possess. In addition, to study why certain values of k and d are more effective than others, and in particular whether this is true for a certain behaviour in the series.

It should be noted that thanks to this information regarding the forecasts, the possibility was raised that certain models of the *k-NN* would perform well whenever the series presents one behaviour or another. Therefore, the hypothesis of using several forecasts simultaneously was raised. Consequently, the result would be a conjunction of different forecasts, each one specialized in certain behaviors.

The latter, combined with the simplicity of the *k-NN*, is a quality that allows the combination of forecasts. To carry out this method, a set of forecasts are made from the same data with different methods, and once they are all available they are combined to give a single forecast. This approach seeks to capture different aspects of the time series with each method, and to achieve a forecast that contains all those aspects.

This approach to combining forecasts, applied to the time series *k-NN*, is

something that has not been explored so far in the literature on the subject.

As a consequence of all the above, the present Master's thesis aims to try to modernize the technique of *k-NN* for time series prediction. The first objective is to improve the predictive capacity of the method by combining predictions. As mentioned above, this approach does not present direct comparisons, since there are no existing works on this subject. Secondly, it will be addressed to increase the transparency and explainability properties, including those of the combination of forecasts.For the latter, the existing basic web application will be improved and extended to enhance the sought-after explainability and transparency.

8.2. Objectives

For all the above reasons, the following objectives are proposed for this work:

- Develop a strategy for *k-NN boosting* in time series. To do this, the implementation of the forecast proposal will be carried out by means of a combination of predictors.
- Improve the efficiency of the *boosting* method. Once the strategy has been developed, it is proposed to improve the efficiency of the developed method.
- Increase the *k-NN* forecasts' explainability. Add metrics related to the forecasting and display of the chosen textitk-neighbours.
- Extend the functionality of the web application to include forecasting by combination. The aim is to visually integrate the different explainable components, as well as to allow the user to understand the different forecasting methods by using them.
- Application redesign: improve the user experience, both in terms of ease of use and response times-

8.3. Work plan

The processes described below will be followed to carry out the project. Regarding the improvement of the forecasts by means of combination:

- Approach and implementation of the forecasting algorithm by means of combination. Study the reuse of existing code.

- Implementation of the forecasting using the predictors generated by the above algorithms.
- Performance analysis of the forecasting algorithm by means of combinations measured in real time series.
- Study of new alternative predictor generation algorithms.

Regarding the web application:

- Improving the user experience: propose an alternative that avoids the automatic update of the web application whenever any value is modified.
- Improving the user experience: study and implement alternatives for the reuse of data used by various elements of the web application.
- Integration in the application of new metrics of the forecasts, as well as a graph for the visualization of the k chosen neighbors.
- Integration of the forecasting algorithm through combination.

Chapter 9

Conclusions

First, discuss the objectives regarding time series forecasting by assembly, the *Random Neighborhoods*. This objective is considered to be fully achieved, as there have been four training implementations based on the two different approaches, *bagging* and selection inspired by genetic algorithms. Furthermore, as seen in the chapter 6 results, all the implementations have managed to overcome the prediction by means of *k-NN* in all the proposed series.

On the other hand, it has been possible to use parallelization in concurrent threads in all the algorithms, so that it is possible to take advantage of the different cores that, increasingly, current processors have. As it has been commented on in the section 6.3, the parallelization used in the methods of *Blocks* and *samples*, although far from being the optimal one, is considered quite good.

The existing web application has also been improved and expanded. To this end, new graphics have been added and existing ones expanded, as well as new parameterization elements to offer more options to the user. Now, the k neighbors of a certain prediction are presented as well as their associated metrics. It is also possible to study the performance of each combination of k and d, including the calculation of local minimums. In addition, a new tab has been created for *Random Neighborhoods* and ensembles in general, and combinations can be generated manually based on the information extracted from the tool.

On the other hand, the user experience has been improved, allowing a more comfortable workflow through the confirmation buttons. The reuse of information between interface elements that share it has also been optimized.

As a consequence of the aforementioned, and given the novelty of the work, the intention is to take this work further by publishing two articles. The first of them would be about the ensemble of predictions by means of *Random Neighborhoods*. The second would deal with the explanability of

k-NN as a time series forecasting method.

Future work

For such articles it is necessary to extend the tests carried out to more series with a larger non-linear component. In this way the potential of *Random Neighborhoods* could be tested. Regarding the *explicability*, it would be interesting to look for more series where to exploit the metrics and functionalities of the web application. This way you could find phenomena that could be presented in the series.

In addition, several previous objectives of improvement of the implementations are proposed. On the one hand, the optimization of *Random Neighborhoods* based on textitbagging, *Blocks* and *Samples* is proposed. This would require a reformulation of the methods to make the best use of the calculation of all distances as well as of the predictions. This is because currently during the training process of each model, they do not share information with each other.

Also with regard to these methods of Random Neighborhoods, the possibility of creating a weighting system for each model based on similarity to what was trained is being considered. That is, when making a prediction by means of ensemble, assigning a weight to each model according to how similar the data to be predicted is to the data used to generate each model.

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

Adadi, A. y Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, vol. 6, páginas 52138–52160, 2018. ISSN 2169-3536.

Aha, D. W., Kibler, D. y Albert, M. K. Instance-based learning algorithms. *Machine learning*, vol. 6(1), páginas 37–66, 1991.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R. et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, vol. 58, páginas 82–115, 2020. ISSN 1566-2535.

Arroyo, J. *Métodos de Predicción para Series Temporales de Intervalos e Histogramas*. Tesis Doctoral, Universidad Pontificia Comillas, 2008.

Assaf, R. y Schumann, A. Explainable Deep Neural Networks for Multivariate Time Series Predictions. *International Joint Conference on Artificial Intelligence*, 2019.

Bastarrica, D. y Berdecio, J. Librería para la predicción usando el k-NN: paralelización y visualización de resultados, 2018. Universidad Complutense, Facultad de Informática, curso 2017/2018.

Box, G. E., Jenkins, G. M., Reinsel, G. C. y Ljung, G. M. *Time Series Analysis: Forecasting and Control*, vol. 734. John Wiley & Sons, 2011.

Bracke, P., Datta, A., Jung, C. y Sen, S. Machine learning explainability in finance: An application to default risk analysis. *SSRN Electronic Journal*, 2019.

Breiman, L. Bagging predictors. *Machine learning*, vol. 24(2), páginas 123–140, 1996.

Breiman, L. Random forests. *Machine learning*, vol. 45(1), páginas 5–32, 2001.

Brown, R. *Smoothing, Forecasting and Prediction of Discrete Time Series*. International series in management. Prentice-Hall, 1963.

Chen, T. y Guestrin, C. Xgboost: A scalable tree boosting system. En *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, páginas 785–794. 2016.

Diebold, F. X. y Mariano, R. S. Comparing predictive accuracy. *Journal of Business & Economic Statistics*, vol. 20(1), páginas 134–144, 2002.

Dietterich, T. G. et al. Ensemble learning. *The handbook of brain theory and neural networks*, vol. 2, páginas 110–125, 2002.

Farrelly, C. M. Knn ensembles for tweedie regression: The power of multiscale neighborhoods. *arXiv preprint arXiv:1708.02122*, 2017.

Freund, Y. y Schapire, R. E. Large margin classification using the perceptron algorithm. *Machine learning*, vol. 37(3), páginas 277–296, 1999.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, páginas 1189–1232, 2001.

Goldberg, D. E. Genetic algorithms in search optimization and machine learning. Addison Wesley, Reading: MA, 1988.

Goodfellow, I., Bengio, Y. y Courville, A. *Deep learning*. MIT press, 2016.

Gunning, D. Explainable Artificial Intelligence (XAI). *Defense Advanced Research Projects Agency (DARPA), Information Innovation Office (I2O)*, 2017.

Harvey, D., Leybourne, S. y Newbold, P. Testing the equality of prediction mean squared errors. *International Journal of forecasting*, vol. 13(2), páginas 281–291, 1997. ISSN 0169-2070.

Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, USA, 2nd edición, 1998. ISBN 0132733501.

Holland, J. H. et al. *Adaptation in Natural and Artificial Systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

Hyndman, R. J. y Khandakar, Y. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, vol. 26(3), páginas 1–22, 2008.

Koren, Y. The Bellkor solution to the Netflix Grand Prize. *Netflix prize documentation*, vol. 81(2009), páginas 1–10, 2009.

Kourentzes, N., Barrow, D. y Petropoulos, F. Another look at forecast selection and combination: Evidence from forecast pooling. *International Journal of Production Economics*, vol. 209, páginas 226 – 235, 2019. ISSN 0925-5273. The Proceedings of the 19th International Symposium on Inventories.

LeCun, Y., Bengio, Y. y Hinton, G. Deep learning. *Nature*, vol. 521(7553), páginas 436–444, 2015.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N. y Lee, S.-I. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, vol. 2(1), páginas 2522–5839, 2020.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J. y Talagala, T. S. Fforma: Feature-based forecast model averaging. *International Journal of Forecasting*, vol. 36(1), páginas 86–92, 2020.

Peña, D. y Sánchez, I. El análisis de series temporales: situación y perspectivas. *Boletín de Estadística e Investigación Operativa. BEIO*, vol. 23(4), páginas 4–8, 2007.

Pérez, J. H. y Rückauer, C. C. Una revisión de los algoritmos evolutivos y sus aplicaciones. 2002.

Polikar, R. Ensemble learning. En *Ensemble machine learning*, páginas 1–34. Springer, 2012.

Ribeiro, M. T., Singh, S. y Guestrin, C. "why should i trust you?". *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks*, vol. 61, páginas 85–117, 2015.

SILSO World Data Center. The international sunspot number. *International Sunspot Number Monthly Bulletin and online catalogue*, 1749-2013.

Stephen, I. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, vol. 50(2), página 179, 1990.

Stock, J. H. y Watson, M. W. Chapter 10 forecasting with many predictors. vol. 1 de *Handbook of Economic Forecasting*, páginas 515 – 554. Elsevier, 2006.

Timmermann, A. Chapter 4 forecast combinations. vol. 1 de *Handbook of Economic Forecasting*, páginas 135 – 196. Elsevier, 2006.

Wettschereck, D., Aha, D. W. y Mohri, T. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, vol. 11(1-5), páginas 273–314, 1997.

Zhou, Z.-H. Ensemble learning. *Encyclopedia of biometrics*, vol. 1, páginas 270–273, 2009.