Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Métodos numéricos para el aprendizaje automatizado utilizando funciones de disimilitud

Autor: Pablo Carranza Jiménez

Tutor: Teodoro Álamo Cantarero

Dpto. Ingenieria de Sistemas y Automatica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Métodos numéricos para el aprendizaje automatizado utilizando funciones de disimilitud

Autor:
Pablo Carranza Jiménez

Tutor:
Teodoro Álamo Cantarero
Catedrático

Dpto. Ingenieria de Sistemas y Automatica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado: Métodos numéricos para el aprendizaje automatizado utilizando funciones de disimilitud

Autor: Pablo Carranza Jiménez
Tutor: Teodoro Álamo Cantarero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Quiero expresar mi profunda gratitud a todas las personas que me brindaron su apoyo y orientación durante la realización de este trabajo de fin de grado.

En primer lugar, quiero agradecer a mis padres por su amor incondicional, su constante aliento y su sacrificio inquebrantable. Su apoyo emocional y su confianza en mí fueron fundamentales para superar los desafíos que encontré en este camino académico.

También quiero agradecer a mi tutor, Teodoro, por su dedicación, paciencia y sabiduría. Sus valiosas sugerencias, comentarios y dirección fueron cruciales para la realización exitosa de este proyecto. Su guía experta me ayudó a crecer tanto académica como personalmente, y estoy sinceramente agradecido por su mentoría.

Por último, pero no menos importante, agradezco a todos aquellos que de alguna manera contribuyeron a este trabajo, ya sea brindando recursos, compartiendo conocimientos o simplemente brindando palabras de aliento.

Este logro no habría sido posible sin el apoyo inquebrantable de todos ustedes. Gracias de todo corazón.

Pablo Carranza Jiménez
Sevilla, 2024

Resumen

En este documento se estudian distintos métodos para afrontar la resolución de problemas de optimización con funciones de disimilitud como función objetivo, en el contexto del aprendizaje automático, en concreto, su uso como modelo predictivo.

Primero es necesario introducir los conceptos más importantes del aprendizaje automático, su historia y la variedad de problemas que existen en este campo. También se destaca su creciente importancia en la actualidad y los distintos campos de aplicación posibles.

El uso del aprendizaje automático como predictor requiere el tratamiento de cantidades enormes de datos, esto puede ralentizar enormemente el tiempo de ejecución de los algoritmos empleados, lo cual para ciertas aplicaciones es un factor determinante.

Es por esto por lo que surge la necesidad de estudiar diferentes algoritmos que puedan reducir los tiempos de ejecución, en concreto se estudiará el algoritmo FISTA, y una serie de variaciones de este, que pretenden ser versiones más rápidas.

Para ello es necesario tener en cuenta las características específicas del problema a resolver, y del set de datos empleado, por lo que es posible que el algoritmo deba ser modificado para diferentes aplicaciones.

Por último, se pondrán en práctica los algoritmos estudiados para un problema real concreto de gran relevancia en la actualidad, en este caso se tratará de predecir la demanda de energía eléctrica en España y se realizará un análisis de los resultados obtenidos.

Abstract

In this paper we study different methods for solving optimization problems with dissimilarity functions as the cost function, in the context of machine learning, in particular, its use for making predictions.

First it is necessary to introduce the most important concepts of machine learning, its history and the variety of problems that exist in this field. Its growing importance today and the various possible fields of application are also highlighted.

The use of machine learning as a predictor requires the processing of huge amounts of data, which can greatly slow down the execution time of the algorithms used, which for certain applications is a determining factor.

This is why the need arises to study different algorithms that can reduce execution times, in particular the FISTA algorithm will be studied, and a series of variations of this, which are intended to be faster versions.

For this it is necessary to take into account the specific characteristics of the problem to be solved, and of the data set used, so it is possible that the algorithm must be modified for different applications.

Finally, the algorithms studied will be put into practice for a specific real problem of great relevance today, in this case it will try to predict the demand for electricity in Spain and an analysis of the results obtained will be carried out.

Índice Abreviado

Índice

1 Introducción

Este documento, como su título indica, trata sobre los diferentes métodos numéricos aplicables a problemas de aprendizaje automático, utilizando funciones de disimilitud como función objetivo de un problema de optimización.

El aprendizaje automático esta ligado estrechamente a la optimización de funciones objetivo, hay una enorme cantidad de funciones objetivo diferentes que se pueden emplear según los objetivos del modelo, algunas de estas funciones son conocidas como funciones de disimilitud las cuales miden la falta de parecido entre dos elementos.

Dentro de la variedad de problemas que se tratan en el ámbito del aprendizaje automático se centrara en problemas de regresión, en concreto los métodos Kriging y similares, el papel de la optimización en ellos y los algoritmos de optimización que se emplean para resolverlos.

Estos métodos sirven para realizar predicciones sobre las características de una localización basadas en otras localizaciones ya muestreadas, o en lo que se centra más este documento realizar predicciones de un valor, basadas en los valores obtenidos en muestras pasadas.

Además, se revisarán las aplicaciones originales de estos métodos, sus aplicaciones actuales y se pondrá en practica con un tema de actualidad, analizando los resultados y rendimiento de los diferentes algoritmos optimización y sus posibles modificaciones.

1.1 Aprendizaje Automático

El aprendizaje automático ha transformado por completo la manera en la que las máquinas adquieren conocimiento y mejoran su rendimiento en diferentes tareas sin una programación explicita. Se ha extendido su uso a una gran variedad de ámbitos, como el análisis de grandes cantidades de datos, la medicina , la industria o la economía.

En este trabajo, se introducen los conceptos básicos del aprendizaje automático y se dará una breve explicación de la evolución de este, desde su concepción hasta la actualidad.

Se explorarán en más profundidad sus aplicaciones en el campo de los modelos predictivos, es decir, su uso para predecir un valor en base a una serie de datos previos.

En general el aprendizaje automático incluye tres elementos:

- Datos de entrenamiento: Son usados por el algoritmo como ejemplos de entrada y salida esperada, los datos de entrenamiento pueden ser entradas con salidas asociadas en el aprendizaje supervisado o solo datos de entrada si se trata de aprendizaje no supervisado.
- Algoritmo: El algoritmo toma los datos de entrada, los analiza y en base a ellos realiza predicciones o toma decisiones.
- Función objetivo: La función objetivo se utiliza para determinar el rendimiento del modelo, compara las predicciones del modelo con las respuestas reales en el caso del aprendizaje supervisado.

El caso que se estudia en este documento es un modelo predictivo, se tienen muestras del pasado, cada una con una serie de datos de entrada y una salida correspondiente, el objetivo es predecir la próxima salida proporcionando los datos de entrada.

Esto se hace mediante una serie de algoritmos que resuelven problemas de optimización, en los que se maximiza la similitud, o se minimiza la disimilitud entre el punto a estudiar y los puntos o muestras del pasado, es decir se busca como de parecida es la muestra a estudiar, a las muestras conocidas, para predecir la salida.

De esta forma tenemos los tres elementos típicos del aprendizaje automático. Se verán en mas profundidad a lo largo de este documento, con un especial enfoque en los diferentes algoritmos empleados y sus mejoras.

1.2 Kriging

Los métodos de regresión Kriging surgen en el ámbito de la Ingeniería de Minas, ideado por el ingeniero de minas Daniel Gerhardus Krige en 1951 y formulado por el matemático e Ingeniero Civil de minas francés Georges François Paul Marie Matheron en 1963.

Consiste en la estimación o predicción de fenómenos espaciales en localizaciones no muestreadas. Aunque su uso puede extenderse a multitud de áreas de estudio, este método surge para predecir las condiciones de diferentes localizaciones sin acceder a ellas, por ejemplo, la cantidad de C02 subterráneo suele mostrarse en mapas que describen las variaciones de C02 basadas en algunas mediciones locales, para construir estos mapas se requieren técnicas de interpolación que tengan en cuenta la variabilidad espacial. La más usada es el método Kriging. [35]

El enfoque se fundamenta en el empleo de un campo aleatorio y una colección de supuestos, como la estacionariedad y la ergodicidad espacial, es decir, las propiedades estadísticas y estructurales de un proceso aleatorio son invariantes con respecto a la posición

en el espacio. Para condensar la información requerida en un variograma que puede ser estimado a partir de las mediciones disponibles.

La incertidumbre presente en el proceso puede provocar dudas en la validez de la interpolación resultante, dado que las mediciones pueden ser inexactas, y la elección del variograma depende de la habilidad de la persona que realiza el estudio. Además, la validez estadística del variograma esta en conflicto con la relevancia local de las muestras obtenidas: En cuanto mas grande sea el área considerada más datos y más válido será el variograma, y menos relevante sera la localización, y al contrario, un análisis muy local tendrá una menor validez estadística. Sin embargo, muy pocos han discutido la naturaleza de la incertidumbre del proceso, excepto el propio G. Matheron y aun menos han discutido teorías alternativas. [34]

Hay numerosos ejemplos del uso de Kriging en diferentes ámbitos, por ejemplo en [37] se usa para hacer un estudio de la evolución de la pandemia Covid-19 en España y en la Comunidad Valenciana, incluyendo predicciones del número de contagios en zonas donde no se de dispone de datos. En [6] se evalúa la potencial contaminación de arsénico en una mina abandonada en Portugal, la cual provoca niveles elevados de arsénico en el suministro de agua de la zona.

1.3 Optimización y Kriging en el aprendizaje automático

Como se ha mencionado anteriormente la optimización de funciones objetivo juega un papel fundamental en el aprendizaje automático. Para que un modelo pueda aprender de los datos y mejorar su rendimiento, se hace uso de algoritmos de optimización, se busca minimizar o maximizar una función objetivo que mida el ajuste del modelo a los datos de entrenamiento.

La elección de la función objetivo depende del tipo de problema que se este abordando. Por ejemplo, en problemas de clasificación, se busca maximizar la tasa de acierto o la precisión del modelo, mientras que en los problemas de regresión se busca minimizar el error entre las predicciones del modelo y las salidas reales.

Aunque el enfoque del Kriging es especifico para la interpolación espacial y se basa en la teoría de campos aleatorios, puede considerarse un problema de regresión en el contexto mas amplio del aprendizaje automático, donde el objetivo es estimar valores continuos a partir de datos muestreados.

Es por esto que la optimización, el Kriging y el aprendizaje automático están muy relacionados entre si, y lo que se desarrolla en este documento, son precisamente los diferentes algoritmos a emplear para resolver los problemas de optimización de funciones de disimilitud que permiten realizar predicciones usando el kriging como problema de regresión.

1.4 Objetivos

Los objetivos de este trabajo son varios y muy amplios, pero realmente hay un solo objetivo final, y los objetivos intermedios son los hitos que se han considerado necesarios para llegar a este. Antes de implementar los métodos que se tratan, es necesario presentar los conceptos e ideas previas para comprender el trabajo, un contexto histórico y actual que demuestre la importancia de los temas que se trabajan y un análisis de las diferentes opciones a las que se pueden aplicar dichos métodos.

Primero se da una breve explicación de los que es el aprendizaje automático en general, se detalla su historia y su creciente importancia en la actualidad, se describen las divisiones que se encuentran dentro del aprendizaje automático y los problemas típicos que resuelve.

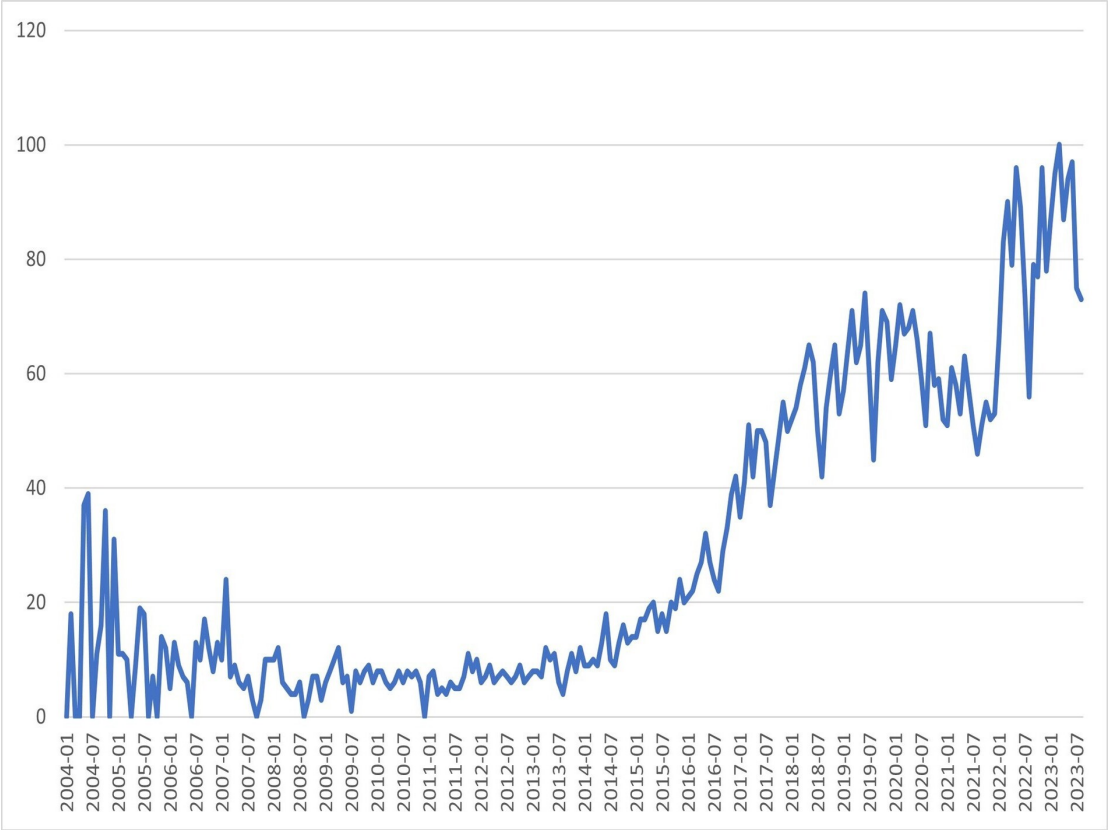

Figura 1.1 Búsquedas de Machine learning en Google desde 2004.

Después se introducen las funciones de similitud y su uso en los modelos predictivos y la regresión, se plantean diferentes alternativas de funciones que pueden emplearse o que han sido empleadas anteriormente, y se propone una función para este caso.

Por último analizar los distintos algoritmos de optimización y proponer uno de ellos en base al rendimiento obtenido en cada uno, teniendo en cuenta la velocidad de ejecución y las iteraciones necesarias, para ello se aplicaran todos los métodos a un mismo problema que servirá de ejemplo y para comprobar la eficacia de los métodos, en este caso se tratará

de predecir la demanda energética en España peninsular en un día concreto.

2 Aprendizaje Automático

A lo largo de este capítulo se introducirá el concepto de aprendizaje automático o machine learning en inglés, se hará un recorrido por su historia y se describirán los usos más habituales del aprendizaje automático, así como sus limitaciones, haciendo especial hincapié en los problemas de regresión, objeto de este documento.

El aprendizaje automático es un subcampo de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos estadísticos que permiten a los computadores *aprender*, mediante una serie de datos, sin ser programados explícitamente. En la última década el machine learning ha experimentado un rápido crecimiento debido al aumento de la capacidad de los procesadores y el almacenamiento de datos, lo que ha permitido el análisis de grandes cantidades de información.

2.1 Historia

En la actualidad los algoritmos de aprendizaje automático permiten a los ordenadores comunicarse con los humanos, conducen coches de forma autónoma, hacen sugerencias de contenido personalizadas para cada usuario en redes sociales, como Tiktok o Instagram y plataformas de streaming de contenido, como Netflix o Amazon prime video, incluso suplen las limitaciones de hardware fotográfico de los dispositivos móviles actuales mediante procesado de imágenes, donde cabe destacar los dispositivos Google, que mediante aprendizaje automático son capaces de de sacar fotos nítidas en condiciones de baja luminosidad o enfocar fotografías borrosas.

En esta sección se repasará el progreso del aprendizaje automático hasta llegar a su estado actual, destacando los hitos más significativos de su historia.

El aprendizaje automático es a menudo llamado Inteligencia artificial, aunque en realidad es una rama de esta. Hasta el final de los años 70, era una parte de la evolución de la Inteligencia artificial, después se separó y evolucionó por su cuenta.

El aprendizaje automático esta en parte basado en un modelo de interacción neuronal.

El modelo fue creado en 1949 por Donald Hebb en un libro llamado *The Organization of Behavior*. El libro presenta las teorías de Hebb sobre excitación neuronal y comunicación entre neuronas. Su modelo puede ser descrito como una forma de alterar la relación entre neuronas artificiales (también llamadas nodos) y los cambios en neuronas individuales. La relación entre dos nodos se fortalece si se activan de forma simultánea y se debilita si se activan de forma individual. El término *peso* se usa para describir estas relaciones, y los nodos que tienden a ser ambos positivos o ambos negativos se dice que tienen pesos positivos fuertes, sin embargo los nodos que tiendan a tener pesos opuestos desarrollan pesos negativos fuertes.

2.1.1 Prueba de Turing

En 1950 Alan Turing crea la *Prueba de Turing* para determinar si un ordenador tiene inteligencia real. Para pasar la prueba, un ordenador debe ser capaz de convencer a un humano de que el ordenador es humano también. La prueba consiste en que un humano y una maquina diseñada para generar respuestas similares a las de un humano, tuvieran una conversación limitada a un medio únicamente textual, como un teclado y un monitor, de forma que sea irrelevante la capacidad de la máquina de transformar texto en habla. Turing propuso que un evaluador externo con conocimiento de que uno de los participantes es una maquina, tratase de distinguir que participante es la maquina. Esta prueba no evalúa la capacidad de la máquina de responder preguntas correctamente, solo se tiene en cuenta su capacidad de generar respuestas similares a las que daría un humano.

2.1.2 Juego de damas

Arthur Samuel de IBM desarrollo un programa para jugar a las damas en 1952, como el programa tenia muy poca memoria disponible, Samuel inició lo que se llama la Poda alfa-beta. En vez de buscar cada camino hasta llegar al final del juego, su diseño incluía una función de puntuaje usando las posiciones de las piezas en el tablero. La función de puntuaje pretendía medir las probabilidades de cada jugador de ganar. El programa elegía sus movimientos basándose en una estrategia minimax, es decir, se examina cada posible estrategia y se ejecuta aquella que minimiza la perdida máxima esperada considerando las posibles respuestas del adversario.

Samuel también diseño una serie de mecanismos que permitían al programa mejorar mediante la memorización de las posiciones que ya habia visto, y los movimientos de las jugadas ganadoras en esas situaciones. Arthur Samuel fue el primero en usar el término machine learning en 1952.

2.1.3 El perceptrón

La primera red neuronal para ordenadores fue diseñada por Frank Rosenblatt en 1957, combinó el modelo de Donald Hebb de interacción neuronal con los avances en machine learning de Arthur Samuel creando el perceptrón, el cual simulaba el proceso de pensamien-

to humano. El perceptrón fue planeado inicialmente como una máquina, no un programa. El software en origen diseñado para el IBM 704, fue instalado en una maquina hecha a medida llamada Mark 1 perceptron, la cual había sido construida para reconocimiento de imágenes. Esto hizo que el software y los algoritmos fueran transferibles a otros ordenadores.

2.1.4 Vecino más cercano

El algoritmo *Nearest neighbor* o vecino más cercano en español, fue escrito por primera vez en 1967, fue el comienzo del reconocimiento de patrones básicos. El algoritmo podía usarse para buscar rutas eficientes, siendo de los primeros algoritmos en usarse para resolver el famoso problema del *Traveling salesman* o vendedor viajero en español, de forma que empezando en una ciudad aleatoria buscaba una ruta eficiente para que el vendedor visitara todas las ciudades programadas.

2.1.5 Multicapas

En la década de 1960, el descubrimiento y uso de multicapas abrió un nuevo camino en la investigación de redes neuronales. Se descubrió que usando dos o más capas en el perceptrón se conseguía un aumento considerable de capacidad de procesamiento. Otras versiones de redes neuronales fueron creadas después de que el perceptrón abriera las puertas de las multiples capas en redes neuronales, esto llevó a las redes neuronales prealimentadas las cuales no forman un ciclo, y a backpropagation o propagación hacia atrás.

La propagación hacia atrás fue desarrollada en los años 70, permite a una red neuronal ajustar su capas ocultas para adaptarse a nuevas situaciones. Describe 'la propagación hacia atrás de errores', un error se procesa a la salida y se distribuye hacia atrás a través de las capas con el propósito de aprender.

Una red neuronal artificial tiene capas ocultas las cuales se usan para responder a tareas más complicadas de las que los primeros perceptrones podían responder. Las redes neuronales artificiales son una herramienta principal en aprendizaje automático. Las redes usan capas de entrada y salida, y normalmente capas ocultas diseñadas para transformar las entradas en datos que puedan ser usados por la capa de salida. Estas capas intermedias ocultas son capaces de encontrar patrones que un humano no podría detectar, es decir, un humano no podría encontrar el patrón y luego enseñar a la máquina a reconocerlo.

2.1.6 Separación de la Inteligencia Artificial y el Aprendizaje Automático

A finales de los años 70 y principio de los 80, la inteligencia artificial se centro más en la lógica y el conocimiento que en los algoritmos. Adicionalmente, la investigación de redes neuronales perdió importancia. Esto provocó una división entre la inteligencia artificial y el aprendizaje automático. Hasta entonces el aprendizaje automático había sido usado como un programa de entrenamiento para la inteligencia artificial.

La industria del aprendizaje automático, fue reconocida como un campo separado de

la inteligencia artificial. El objetivo de la industria cambio de ser un entrenamiento para la inteligencia artificial a resolver problemas prácticos que proporcionasen un servicio. Se mantuvo centrado en la idea de las redes neuronales y floreció en la década de 1990, resultado del crecimiento de internet y el aumento de información disponible.

2.1.7 Boosting

El boosting es un método utilizado en aprendizaje automático para reducir los errores en el análisis predictivo de datos. Los científicos de datos entrenan al modelo de machine learning, con datos etiquetados para efectuar predicciones sobre datos no etiquetados. Un único modelo de machine learning puede cometer errores de predicción según la precisión del conjunto de datos de entrenamiento, por ejemplo, si un modelo de identificación de gatos es entrenado para identificar a los gatos como animales con las orejas puntiagudas, es posible que en ocasiones no identifique un gato que tenga las orejas dobladas. El boosting trata de solucionar este problema mediante el entrenamiento secuencial de varios modelos para mejorar la precisión del sistema en general.

Los estudiantes débiles tienen una precisión de predicción baja, similar a predicciones aleatorias. Son propensos al sobreajuste, es decir, no pueden clasificar datos que varían demasiado de su conjunto de datos original.

Los estudiantes fuertes tienen una precisión de predicción más alta. El boosting convierte un sistema de estudiantes débiles en un único sistema de aprendizaje fuerte. Por ejemplo para identificar la imagen del gato, se combina un estudiante débil que identifica las orejas puntiagudas y otro que identifica la forma de los ojos. Después de analizar la imagen del animal por las orejas puntiagudas, el sistema la analiza una vez más por la forma de sus ojos, esto mejora la precisión general del sistema. El concepto de boosting fue presentado por primera vez en un documento de 1990 titulado *The strength of weak learnability* de Robert Schapire.

2.2 Problemas típicos del aprendizaje automático

La versatilidad del aprendizaje automático permite su aplicación en una amplia gama de áreas y problemas, lo que la convierte en una herramienta poderosa para abordar desafíos complejos y tomar decisiones basadas en datos. Los diferentes tipos de problemas que resuelve el aprendizaje automático pueden agruparse en categorías, según la técnica con la que se acometa su resolución.

El objetivo de esta sección es ofrecer una visión general de los tipos de problemas para los que se usa habitualmente el aprendizaje automático y su clasificación.

2.2.1 Aprendizaje supervisado

El aprendizaje supervisado es una rama de Machine Learning , un método de análisis de datos que utiliza algoritmos que aprenden iterativamente de los datos para permitir que los ordenadores encuentren información escondida sin tener que programar de manera explícita dónde buscar. La acción más corriente es que el algoritmo efectúe ajustes en el modelo que está generando cada vez que se le indica que se ha equivocado, con el objetivo de mejorar sus predicciones.

Se utiliza un conjunto de datos etiquetados conocidos para entrenar un algoritmo para realizar tareas específicas. Utiliza modelos para predecir resultados conocidos. Por ejemplo, un proceso de aprendizaje supervisado podría consistir en clasificar vehículos de dos y cuatro ruedas a partir de sus imágenes. Los datos de entrenamiento tendrían que estar correctamente etiquetados para identificar si un vehículo es de dos o cuatro ruedas. El aprendizaje supervisado permite que los algoritmos aprendan de datos históricos/de entrenamiento y los apliquen a entradas desconocidas para obtener la salida correcta. Existen dos tipos principales de aprendizaje supervisado; clasificación y regresión.

Regresión

El objetivo de este tipo de problemas es encontrar una función matemática cuya curva se adapte lo mejor posible a los datos. Esto nos permitirá predecir el valor de una variable de salida Y conocidos los valores de otras variables de entrada X.

Los problemas de regresión se caracterizan en que la variable de respuesta Y es cuantitativa. Esto significa que la solución a nuestro problema es representada por una variable continua que puede ser flexiblemente determinada por las entradas X a nuestro modelo, en lugar de estar restringida a un grupo posible de valores.

Las predicciones dentro de el rango de valores del dataset que se usa para ajustar el modelo reciben el nombre de interpolaciones. Por el contrario, aquellas que están fuera del rango del dataset usado para ajustar o entrenar el modelo reciben el nombre de extrapolaciones. La realización de extrapolaciones se basan fuertemente en supuestos. Cuanto más lejos esté la extrapolación de los datos, más espacio hay para fallos debido a las diferencias entre las suposiciones y la realidad.

Para el ajuste o entrenamiento del modelo se usan las salidas reales conocidas para corregir sus parámetros y mejorar la predicción.

Existen diferentes modelos que pueden ser usados para predecir una variable cuantitativa, siendo la regresión lineal de los más simples. En el caso de la regresión lineal, el algoritmo intenta ajustarse a los datos de la mejor forma obteniendo el hiperplano más óptimo (por ejemplo, el que minimice la distancia cuadrática a los puntos).

Los problemas de regresión que tienen entradas con dependencia temporal son también llamados problemas de predicción de series temporales o forecasting.

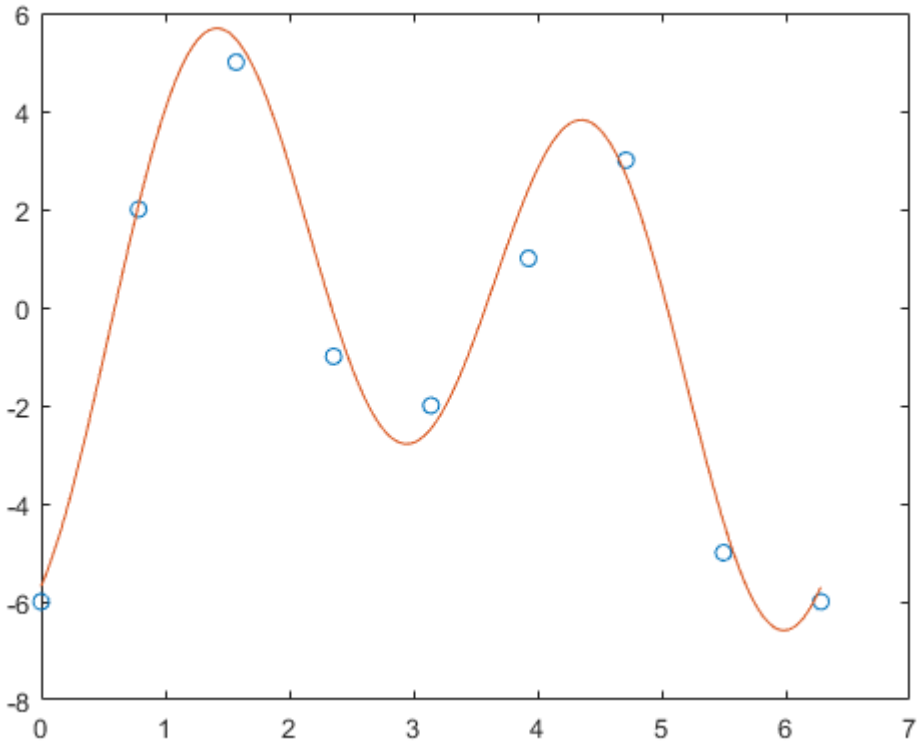

Figura 2.1 Ejemplo regresión.

Clasificación

Los problemas de clasificación se caracterizan por tener una variable cualitativa Y como respuesta. Muchas veces las variables cualitativas también reciben el nombre de variables categóricas.

Al hecho de predecir una respuesta cualitativa de una observación se le llama clasificar dicha observación, es decir, predecir la categoría o clase de dicha observación. Para clasificar de forma automática se crea un modelo predictivo, al que entregando las variables de entrada genere como salida las etiquetas de clase correspondientes.

Muchas veces los métodos encargados de clasificar lo que hacen es predecir la probabilidad de una observación de pertenecer a cada una de las categorías. En cierto modo se comportan también como algoritmos de regresión.

Entre los métodos de clasificación se encuentran:

- Regresión logística
- K-NN (K-nearest neighbors)
- SVMs

Normalmente, en el caso que usemos probabilidades, se tiende a elegir la clase con probabilidad más alta como resultado del proceso de clasificación. Otra posibilidad que

está a nuestro alcance es fijar un mínimo de probabilidad antes de estar dispuestos a dar un resultado. Por ejemplo, si no estamos seguros con una probabilidad 80 % o mayor, diremos que no estamos seguros. Esta estrategia puede ser útil cuando el coste de equivocarnos es alto, comparado con el beneficio de obtener la respuesta correcta.

También podemos diferenciar entre la clasificación binaria y clasificación multiclase. La clasificación binaria separa dos clases o incluso a una sola clase considerando la segunda clase aquella que engloba todos los datos que no pertenecen a la clase que queremos separar. La clasificación multiclase como su nombre indica separa varias clases. Normalmente se resuelve como un conjunto de clasificaciones binarias con la técnica conocida como 1 *vs all*, la cual crea clasificadores independientes para cada clase, y clasifica los datos en aquella clase que obtenga los resultados más solidos.

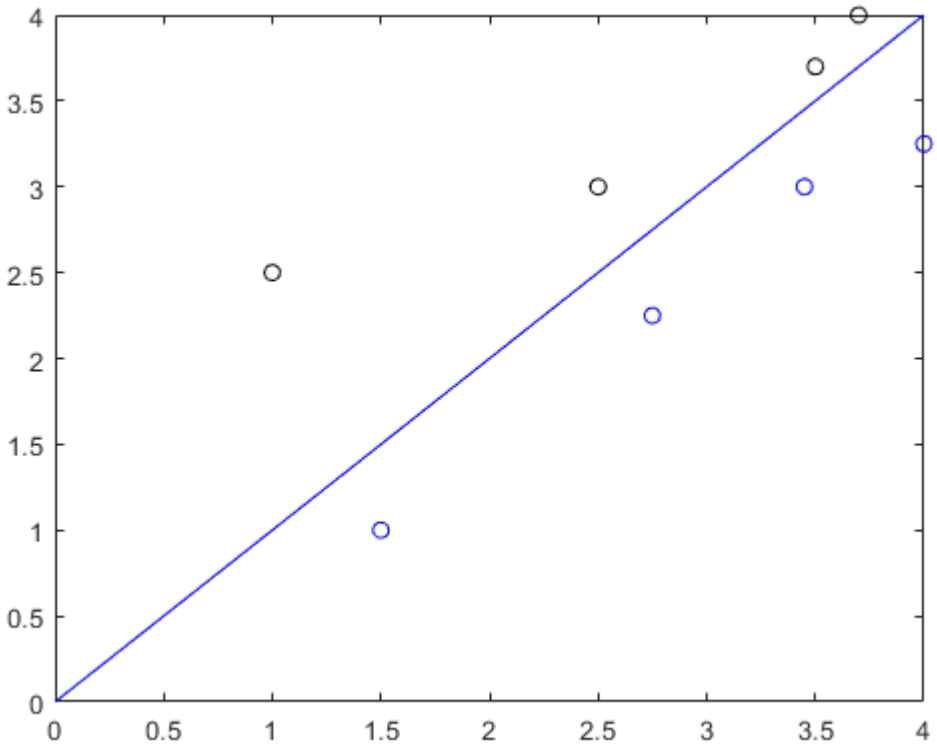

Figura 2.2 Ejemplo clasificación binaria.

2.2.2 Aprendizaje no supervisado

El aprendizaje no supervisado es una de las formas en que Machine Learning *aprende* los datos. El aprendizaje no supervisado tiene datos sin etiquetar que el algoritmo tiene que intentar entender por sí mismo.

El aprendizaje no supervisado se utiliza para explorar datos desconocidos. Puede revelar

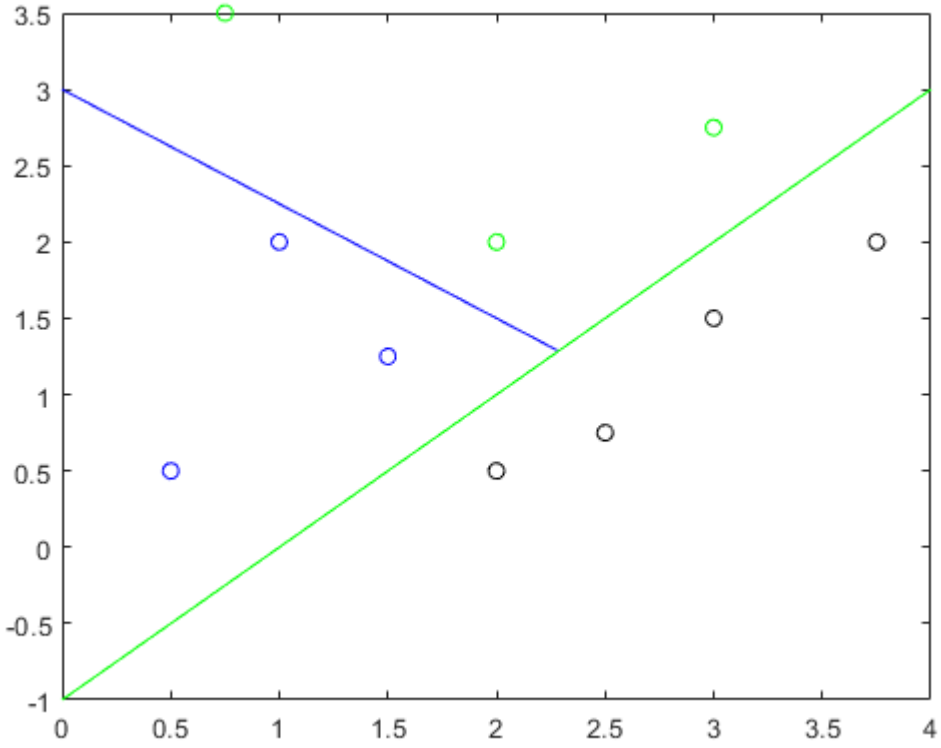

Figura 2.3 Ejemplo clasificación multiclase.

patrones que podrían haberse pasado por alto o examinar grandes conjuntos de datos que serían demasiado para que los abordara una sola persona. Se verán en más detalle 3 variedades de aprendizaje no supervisado; clustering, asociación y reducción de variables

Clustering

Los algoritmos de clustering o agrupamiento son un amplio conjunto de técnicas para encontrar subgrupos o clústers en los datos. Cuando agrupamos los datos queremos encontrar particiones que dividan los datos en grupos distintos pero homogéneos, es decir, queremos que los grupos sean lo más distintos posibles entre ellos pero que las muestras dentro de cada grupo sean similares. Para alcanzar este objetivo hay que definir qué entendemos con que unas muestras sean similares o distintas. Esta definición de similaridad depende del conocimiento de los datos a estudiar, por ejemplo, calculando la distancia a la que se encuentran unas de otras en el espacio generado por los valores de sus variables.

Supongamos que tenemos p variables de n observaciones de enfermedades. Podríamos usar las técnicas de clustering para agrupar las n observaciones en grupos de enfermedades. Este es un problema no supervisado ya que intentamos obtener conocimiento de la estructura de los datos (en este caso grupos de datos).

Asociación

La búsqueda de asociaciones entre ciertos valores de las variables que componen las muestras se efectúa buscando la concurrencia entre ellos, es decir, contando las veces que aparecen simultáneamente. Este tipo de problema puede generar como resultado un conjunto de reglas de asociación.

Este tipo de problema está muy presente en la actualidad, con el desarrollo de internet han surgido compañías que usan los modelos de asociación como sistemas de recomendación. En plataformas de streaming de contenido como *Netflix*, *Amazon video*, *Disney+* o *YouTube*, se basan en las series, películas y vídeos que ves para recomendarte más contenido, y en páginas de comercio electrónico como *Amazon* antes de realizar una compra, se hacen sugerencias de productos que han sido comprados de forma conjunta por otros usuarios previamente.

Reducción de dimensionalidad

Los métodos de reducción de dimensionalidad son los procesos de reducir el número de variables obteniendo un subgrupo de variables principales. Mediante el análisis de la distribución de los valores de las variables en el conjunto de muestras, es posible determinar cuáles de ellas aportan más información, cuáles están correlacionadas con otras y por tanto son redundantes o si es posible encontrar una distribución estadística subyacente que genera esos datos, lo cual permitiría simplificar su representación original. En este tipo de tarea existen multitud de técnicas posibles, desde la selección y extracción de variables hasta lo que se denomina *manifold learning*, consistente en encontrar la citada distribución subyacente.

Estos métodos sirven para problemas donde se tienen una gran cantidad de variables y se pretende resumir o recopilarlas en otro grupo de variables más fácil de tratar.

Los métodos de reducción de dimensionalidad convierten la información de un dataset de gran dimensión en otro con menos dimensión asegurándose de que contiene información similar.

Entre las ventajas de la reducción de dimensionalidad se encuentran:

- Comprime y reduce el espacio requerido para almacenar los datos.
- Mejora el tiempo de procesamiento para la realización de los mismos cálculos.
- Tiene en cuenta el efecto de multicolinearidad, borra o minimiza variables redundantes incrementando el rendimiento de los modelos.
- Puede reducir los datos a dimensiones representables como la 2D o 3D.

Estas ventajas son generales, siendo cada método de reducción de dimensionalidad distinto y siendo necesario de un estudio previo de cómo funcionan en particular.

3 Modelos predictivos usando funciones de disimilitud

Como se ha mencionado previamente la optimización de funciones objetivo es fundamental en el aprendizaje automático, y el objetivo de este trabajo es estudiar los diferentes algoritmos que se pueden emplear para ello, que mejor se adapten a las características de nuestro problema.

Para ello, primero es necesario determinar que función objetivo se desea optimizar.

En este capítulo, se estudiaran las funciones de disimilitud y su uso en modelos predictivos. Esta familia de funciones permitirá calcular la similitud (o disimilitud) de una muestra respecto a un conjunto previamente definido.

Como resultado obtenemos una técnica para pronosticar cualquier serie temporal, dicha técnica tiene en cuenta la localización para mejorar la predicción de procesos con comportamientos no lineales.

3.1 Funciones de disimilitud

Dada un cierto set de datos

$$D = [\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_N] \in \mathbb{R}^{n \times N} \tag{3.1}$$

donde cada $\vec{d}_i \in \mathbb{R}^n$ es una muestra pasada y N es el número de muestras, nos interesa saber si el vector dado $d \in \mathbb{R}^n$ puede considerarse similar a los vectores de la matriz de datos D. Es decir buscamos una función que mida la disimilitud entre el punto dado d y la matriz de datos D.

$$J_d(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^{nxN} \Rightarrow [0,\infty) \tag{3.2}$$

Grandes valores de $J_d(d,D)$ indican una gran disimilitud, mientras que valores pequeños indican una gran similitud. Es fácil obtener una función de similitud $J_s(d,D)$ de una de disimilitud $J_d(d,D)$, por ejemplo, dado $c \geq 0$ si $J_d(d,D) \Rightarrow [0,\infty)$, podemos deducir $J_s(d,D) = \exp(-cJ_d)$ con $J_s(d,D) \Rightarrow (0,1]$ [39].

Hay una gran variedad de funciones que pueden servir como funciones de disimilitud, una opción son las normas vectoriales.

Existen distintas normas vectoriales, para el caso particular de que D sea un conjunto unitario $(D = \vec{d})$, puede emplearse

$$J_d(d,\vec{d}) = \left\| d - \vec{d} \right\|, \tag{3.3}$$

donde $\|\cdot\|$ es una norma vectorial.

Una opción similar es usar la mínima distancia de d respecto a cada miembro de D, para el caso en el que D no sea un conjunto unitario. [13]

$$J_d(d,\vec{d}) = \min_{i=1,\ldots,N} \left\| d - \vec{d}_i \right\|, \tag{3.4}$$

Otras variaciones del uso de normas vectoriales como funciones de disimilitud son, la distancia Manhattan o L_1 que nos dice que la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas, en nuestro caso haríamos la suma de las diferencias de las coordenadas de d con cada punto de D.

$$L_1 = \sum_{i=1}^{n} |d_i - d_{ji}|, \text{ para j=1}, \ldots, N \tag{3.5}$$

También existe la variable L_2 con la diferencia absoluta al cuadrado de forma que se enfatizan más las grandes diferencias absolutas entre las coordenadas de los puntos. ver capítulo 2 de [24] en los que se describen una gran variedad de funciones de similitud y disimilitud en el contexto de tratamiento de imágenes.

3.2 Las funciones de disimilitud y la regresión

Las funciones de similitud y disimilitud que se discuten en este capítulo, son aplicables a los problemas de regresión descritos en el capítulo 2, ya que en estos se trata de encontrar una función matemática, cuya curva se adapte lo mejor posible a los datos y permita predecir el valor futuro de una variable, y lo que se pretende con el kriging es obtener también dicha predicción pero con otra metodología.

En el kriging no se busca una función de ajuste estableciendo una relación funcional entre las variables de entrada y salida, se busca un valor desconocido en función de los datos vecinos basándose en la teoría de campos aleatorios, es decir en vez de usar una curva para encontrar los siguientes puntos, se buscan directamente los puntos.

Si tenemos una serie de muestras x_i, de tamaño tan grande como puntos de información de tengan de las muestras, y sus correspondientes salidas y_i con $i = 1,\ldots,N$ siendo N el numero de muestras, si dada una muestra x cuya salida es desconocida, queremos estimar su correspondiente salida y, dada la función de similitud $J_s(x,x_i)$, una opción para la estimación $\hat{y}$ de y es

$$\hat{y} = \sum_{i=1}^{N} \lambda_i y_i, \tag{3.6}$$

en la que los escalares λ_i, son elegidos de forma que si la función de similitud $J_s(x,x_i)$ es pequeña, es decir, si x es fácil de representar como combinación de las muestras , λ_i es pequeño.

Es interesante normalizar unitariamente la suma de los escalares λ_i, es decir

$$\sum_{i=1}^{N} \lambda_i = 1. \tag{3.7}$$

3.3 Función de disimilitud propuesta

Las funciones de disimilitud descritas anteriormente son validas para algunas aplicaciones, sin embargo para los problemas más complejos son necesarias funciones de disimilitud más sofisticadas. La familia de funciones propuesta, la cual ha sido previamente aplicada en [12] y [4], está definida por un problema de optimización convexo.

Dado el set de datos de muestras $D = [\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_N] \in \mathbb{R}^{n\times N}$ y el vector $d \in \mathbb{R}^n$ con los escalares positivos $\{\gamma_i\}_{i=1}^N$, que se usarán como parámetro de compromiso, la función de disimilitud $J_\gamma(d,D)$ queda definida como:

$$\begin{aligned} \min_{\lambda_1,\ldots\lambda_N} \quad & \sum_{i=1}^{N} \lambda_i^2 + \gamma_i|\lambda_i| \\ \text{s.t.} \quad & 1 = \sum_{i=1}^{N} \lambda_i \\ & x = \sum_{i=1}^{N} d_i\lambda_i. \end{aligned}$$

Cabe destacar la posibilidad de añadir pesos constantes no negativos a la función de costes, quedando la siguiente función de costes:

$$\sum_{i=1}^{N} w_i\lambda_i^2 + \gamma_i|\lambda_i| \tag{3.8}$$

Esta será la función de costes que se use finalmente en este documento, El valor de los pesos w_i puede elegirse en base a una función de similitud entre d y d_i de forma que se usa

información local en la función, por ejemplo, si se pretende predecir el precio del Kw/h en España, usando los precios históricos como salidas y las características de cada día pasado como muestra d_i, se podría asignar el peso w_i en función a la proximidad temporal entre d_i y d, la época del año o el día de la semana.

Es posible demostrar que la función no se ve afectada por transformaciones afines, por lo que no es necesaria la normalización del set de datos de muestras, sin embargo, puede verse influenciada por valores atípicos.

Mediante el uso de la formulación dual del problema de optimización propuesto, es posible demostrar que para $\gamma = 0$ la función resultante es la curva normal, en [39] se desarrolla esta demostración llegando a la conclusión de que el valor de la función de disimilitud para $\gamma = 0$ representa el exponente de la curva normal con signo cambiado, y mediante la transformación mencionada anteriormente $J_s(d,D) = \exp(-cJ_d)$ quedaría igual a la curva normal.

Además al variar γ se amplia el número de curvas que se pueden aproximar a la función, luego la familia propuesta es más rica que la familia de curvas normales.

Por ultimo, es importante distinguir que el método propuesto no hace uso de la base de datos como entrenamiento, como hacen las redes neuronales, por lo que no se considera un set de entrenamiento, en cambio la base de datos se usa cada vez que se requiere una predicción, esto significa que el método propuesto no requiere una fase de entrenamiento, por lo que nuevos datos pueden ser introducidos en la base de datos cuando estén disponibles, sin tener que re-entrenar el predictor.

Esto permite que el uso del predictor solo este limitado por la velocidad de actualización de los datos, para muchas aplicaciones es importante poder hacer predicciones actualizadas a diario o varias veces en un mismo día, para ello es esencial que se pueda hacer uso del predictor según se actualice la fuente de datos empleada.

También es importante destacar la relevancia del problema de optimización a resolver, es un problema que ha sido resuelto en multitud de ocasiones y con diferentes aplicaciones. A continuación se describirán sus aplicaciones más relevantes en este contexto.

Una de los usos que se le ha dado a este método y en concreto usando un problema de optimización muy similar al propuesto, es la predicción de los índices bursátiles [4].

Este es uno de los casos en los que como se ha mencionado anteriormente, es importante poder actualizar el modelo predictivo tan pronto como se actualicen los datos, en el documento citado se emplean los precios de las acciones como salida y el estado del mercado como muestra, de forma que se obtiene una predicción combinando los valores de las acciones asociados a los estados pasados del mercado.

Un tema que ha tomado importancia en los últimos años, es el estudio estadístico de las pandemias, en concreto, la pandemia de COVID-19, por ello se han implementado técnicas similares a la que se propone para su control estadístico y predicción. En [37] se emplea el Kriging para realizar predicciones sobre el avance de la pandemia en España, y en concreto en la Comunidad Valenciana.

Por otro lado, en [3] se estudia un aspecto muy importante de este tipo de problemas, el acceso a bases de datos abiertas. Se analiza que variables son requeridas para el análisis de la pandemia y que organizaciones hacen públicos los datos sobre estas variables.

4 Métodos matemáticos

En este capítulo se describirán los diferentes métodos numéricos para la resolución del problema visto en el capítulo 3, todos se Implementarán en el software Matlab.

Con los diferentes métodos numéricos se tratará de reducir el tiempo de ejecución del algoritmo, ya que con grandes cantidades de datos el tiempo de ejecución puede ser de considerable.

Como se vio en el capítulo 3, la función de disimilitud propuesta y problema a resolver en este capítulo es:

Dado $d \in \mathbb{R}^n$, $\{d_i\}_{i=1}^N \in \mathbb{R}^n$, y los escalares positivos $\{\gamma_i\}_{i=1}^N$, la función de disimilitud $J_\gamma(d,D)$ queda definida como

$$\begin{aligned} J = \quad & \min_{\lambda_1,\ldots\lambda_N} \quad \sum_{i=1}^{N} w_i \lambda_i^2 + \gamma_i |\lambda_i| \\ & \text{s.a.} \quad 1 = \sum_{i=1}^{N} \lambda_i \\ & \qquad d = \sum_{i=1}^{N} d_i \lambda_i. \end{aligned}$$

4.1 Función quadprog de Matlab

Esta función propia de Matlab es un solver para funciones objetivo cuadráticas, Quadprog encuentra un mínimo para un problema especificado por:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^T Hx + f^T x \\ \text{s.a.} \quad & A_{eq}x = b_{eq} \\ & Ax \leq b, \end{aligned}$$

Donde H, A y A_{eq} son matrices, y f, b, b_{eq} y x son vectores. Luego lo primero que tenemos que hacer es reescribir nuestro problema para que tenga esta estructura.

Podríamos reescribir el problema como

$$\begin{aligned} \min_{\{\lambda_i\}_{i=1}^N, \{\tau_i\}_{i=1}^N} \quad & \sum_{i=1}^{N} w_i \lambda_i^2 + \gamma_i \tau_i \\ \text{s.a.} \quad & 1 = \sum_{i=1}^{N} \lambda_i \\ & d = \sum_{i=1}^{N} d_i \lambda_i \\ & -\tau_i \leq \lambda_i \leq \tau_i \end{aligned}$$

Donde hemos introducido la variable τ para eliminar el valor absoluto de la función de costes, introduciendo para ello una nueva restricción. Si definimos $x = [\lambda_1, \ldots, \lambda_N, \tau_1, \ldots, \tau_N] \in \mathbb{R}^{2N}$, el problema de optimización puede ser reescrito con la estructura que sigue la función quadprog de Matlab. Donde:

$$\begin{aligned} H_i &= 2 \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_N \end{bmatrix} \\ H &= \begin{bmatrix} H_i & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{2N \times 2N}. \\ f &= \begin{bmatrix} 0 & 0 & \ldots & 0 & \gamma_1 & \ldots & \gamma_N \end{bmatrix}^\top \in \mathbb{R}^{2N}. \\ A_{eq} &= \begin{bmatrix} d_1 & d_2 & \ldots & d_N & 0 & 0 \ldots & 0 \\ 1 & 1 & \ldots & 1 & 0 & 0 \ldots & 0 \end{bmatrix} \in \mathbb{R}^{(n+1) \times 2N} \\ b_{eq} &= \begin{bmatrix} d \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1} \\ A &= \begin{bmatrix} I & -I \\ -I & -I \end{bmatrix} \in \mathbb{R}^{2N \times 2N}. \\ b &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \in \mathbb{R}^{2N}. \end{aligned}$$

Una vez tenemos el problema de optimización convexo original reescrito con esta estructura, es fácil resolverlo usando quadprog.

4.2 Formulación dual del problema

Partiendo del problema de optimización convexo propuesto, buscamos su formulación dual que nos permitirá aplicar ciertos algoritmos para su resolución. Para ello trataremos de "objetivizar las restricciones", es decir eliminar las restricciones añadiéndolas a la función objetivo o función de costes con un parámetro de regularización (Lagrangiano). Para ello, primero es interesante juntar ambas restricciones en una, usando la siguiente notación

$$\begin{aligned} r &= \begin{bmatrix} d \\ 1 \end{bmatrix}, \\ r_i &= \begin{bmatrix} d_i \\ 1 \end{bmatrix}, \; i = 1, \ldots, N. \end{aligned}$$

Podemos reescribir el problema original como

$$\begin{aligned} \min_{\lambda_1, \ldots \lambda_N} \quad & \sum_{i=1}^{N} w_i \lambda_i^2 + \gamma_i |\lambda_i| \\ \text{s.t.} \quad & r = \sum_{i=1}^{N} r_i \lambda_i. \end{aligned}$$

Asumimos que el problema de optimización es siempre viable, esto significa que la matriz $[r_1 r_2 \ldots r_N]$ tiene todas las columnas independientes, es decir tiene rango completo $R = N$.

Ahora ya podemos objetivizar la restricción resultante, quedando la siguiente función dual

$$\varphi(\mu) = \min_{\lambda_1, \ldots \lambda_N} \sum_{i=1}^{N} w_i \lambda_i^2 + \gamma_i |\lambda_i| + \mu^T (\sum_{i=1}^{N} r_i \lambda_i - r). \tag{4.1}$$

Podemos observar que el problema es separable. Esto significa que podemos calcular el valor óptimo de cada λ_i, $i = 1, \ldots, N$, lo haremos con la siguiente expresión

$$\lambda_i^*(\mu) = \arg\min_{\lambda_i \in \mathbb{R}} w_i \lambda_i^2 + \gamma_i |\lambda_i| + (\mu^T r_i) \lambda_i. \tag{4.2}$$

Luego, cada problema podemos generalizarlo ya que pertenece a la siguiente familia de problemas

$$z^* = \arg\min_{z\in\mathbb{R}} \; wz^2 + \gamma|z| + cz, \tag{4.3}$$

Donde $w > 0$ y $\gamma \geq 0$, $c \in \mathbb{R}$.

Entonces el mínimo de la función $f(z) = wz^2 + \gamma|z| + cz$, y solución general de la familia de problemas se obtiene

$$z^* = \begin{cases} \dfrac{\gamma - c}{2w} & \text{si } c > \gamma, \\ 0 & \text{si } |c| \leq \gamma, \\ \dfrac{-\gamma - c}{2w} & \text{si } c < -\gamma. \end{cases} \tag{4.4}$$

Podemos observar que claramente $f(0) = 0$, por lo que el valor mínimo de la función z^* debe satisfacer $f(z^*) \leq 0$. Consideramos tres casos:

1. $c > \gamma$: Como $\gamma \geq 0$ inferimos que también debe cumplirse $c > 0$. luego, observando la función $f(z)$ y sabiendo que $f(z^*) \leq 0$ implica que $z^* < 0$. Por lo tanto,

$$\begin{aligned} z^* &= \arg\min_{z\leq 0} \; wz^2 + \gamma|z| + cz \\ &= \arg\min_{z\leq 0} \; wz^2 - \gamma z + cz = \frac{\gamma - c}{2w} < 0. \end{aligned}$$

2. $|c| \leq \gamma$: De esta inecuación inferimos que, para cada $z \in \mathbb{R}$:

$$\begin{aligned} f(z) &= wz^2 + \gamma|z| + cz \\ &\geq wz^2 + \gamma|z| - |cz| \\ &= wz^2 + (\gamma - |c|)|z| \\ &\geq wz^2 \geq 0. \end{aligned}$$

Luego, en este caso, $f(z) \geq 0$, $\forall z \in \mathbb{R}$. Sabiendo esto y que debe cumplirse $f(z^*) \leq 0$ ya que $f(0) = 0$, solo se cumplen estas condiciones para $z^* = 0$.

3. $c < -\gamma$: Como $\gamma \geq 0$ inferimos que también debe cumplirse $c < 0$. luego, observando la función $f(z)$ y sabiendo que $f(z^*) \leq 0$ implica que $z^* > 0$. Por lo tanto,

$$\begin{aligned} z^* &= \arg\min_{z>0} \; wz^2 + \gamma|z| + cz \\ &= \arg\min_{z>0} \; wz^2 + \gamma z + cz = -\frac{(c+\gamma)}{2w} > 0. \end{aligned}$$

De esta forma quedan descritas todas las posibles soluciones de la familia de problemas

4.3 Análisis de la función dual

Como $\varphi(\tau)$ es una cota inferior de J por ser su formulación dual, el objetivo es calcular el máximo de $\varphi(\tau)$ respecto a τ.

De los resultados estándar de análisis convexo, obtenemos que la cota inferior de la función dual $\varphi(\mu+\Delta\mu)$ es

$$\varphi(\mu+\Delta\mu) \geq \varphi(\mu)+\Delta\mu^{\top} g(\mu)-\frac{1}{2}\Delta\mu^{\top} R H^{-1} R^{\top}\Delta\mu, \tag{4.5}$$

donde

$$\begin{aligned} g(\mu) &= \left(\sum_{i=1}^{N} r_i \lambda_i^*(\mu) - r\right), \\ R &= \begin{bmatrix} r_1 & r_2 & \dots & r_N \end{bmatrix}, \\ H &= 2\begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_N \end{bmatrix} \end{aligned}$$

Luego, como buscamos maximizar la función dual, debemos maximizar su cota mínima, es decir buscamos el valor de $\Delta\mu$ que satisfaga

$$g(\mu) - R H^{-1} R^{\top}\Delta\mu = 0. \tag{4.6}$$

Despejando $\Delta\mu$, queda

$$\Delta\mu^* = (R H^{-1} R^{\top})^{-1} g(\mu). \tag{4.7}$$

Una vez hecho este análisis y descrito el valor óptimo de $\Delta\mu$ podemos discutir diferentes algoritmos para la obtención de los valores de λ

4.4 Algoritmo ISTA

El algoritmo ISTA (Iterative Shrinkage Thresholding Algorithm) es una técnica iterativa para la resolución de problemas de optimización, es habitual su uso en el ámbito del procesamiento de imágenes, como puede verse en [7], y en el procesamiento de señales como en [51].

Hoy en día, el algoritmo ISTA no es la mejor solución para resolver este tipo de problemas, sin embargo, es muy simple, fácil de implementar y un buen punto de inicio para empezar a tratar con este tipo de algoritmos.

El algoritmo para el caso que se trata en este documento, consiste en calcular de forma iterativa los valores de λ_i, mediante la resolución de la familia de funciones descrita en la sección anterior, con μ siendo actualizado en cada iteración, con el valor optimo de $\Delta\mu$, descrito también en la sección anterior.

Luego, el algoritmo ISTA seria

1. Elige el parámetro de precisión $0 < \varepsilon \ll 1$.

2. $k = 0$, $\mu_0 = 0 \in \mathbb{R}^{n+1}$.

3. Para $i = 1, \ldots, N$ calcular:

$$\lambda_i^*(\mu_k) = \arg\min_{\lambda_i \in \mathbb{R}} \; w_i\lambda_i^2 + \gamma_i|\lambda_i| + (\mu_k^\top r_i)\lambda_i$$

4. calcular

$$g_k = \left(\sum_{i=1}^{N} r_i\lambda_i^*(\mu_k) - r \right).$$

5. $\mu_{k+1} = \mu_k + (RH^{-1}R^\top)^{-1} g_k$.

6. If $\|g_k\|_2 \leq \varepsilon$ entonces SALIR.

7. $k = k+1$ e IR A (3).

El algoritmo es sencillo de programar en Matlab, creando una función que lo ejecute con los datos de entrada D, w, ε, γ y d, obteniendo como datos de salida los valores de λ_i que aproximen $\sum_{i=1}^{N} r_i\lambda_i^*(\mu_k)$ lo suficiente a r para que $g_k = \left(\sum_{i=1}^{N} r_i\lambda_i^*(\mu_k) - r \right)$, cumpla $\|g_k\|_2 \leq \varepsilon$.

4.5 Algoritmo FISTA

El algoritmo FISTA (Fast Iterative Shrinkage Thresholding Algorithm), es una variante del algoritmo ISTA, pero como su propio nombre indica es más rápido.

La convergencia del ISTA, esta muy documentada en diferentes contextos, como en [21], [18] y es un método ampliamente utilizado por su gran ventaja, que es su simplicidad, sin embargo, es un método lento.

Es por esto que surge el algoritmo FISTA, la diferencia entre estos dos algoritmos es el cálculo del parámetro que se actualiza en cada iteración.

En ISTA tenemos un valor de μ inicial que se actualiza en cada iteración, por el contrario en FISTA se tienen unos valores iniciales β_1 y β_0, con los que se calcula el valor de μ que

se usara para el cálculo de los valores λ_i, y es la variable β la que se actualiza en cada iteración, para volver a calcular μ.

Luego el algoritmo FISTA seria

1. Elige el parámetro de precisión $0 < \varepsilon \ll 1$.

2. $\beta_1 := \beta_0 := 0 \in \mathbb{R}^{n+1}$.

3. $k := 1, t_0 := 1$.

4. $t_k := \frac{1}{2}\left(1 + \sqrt{1+4t_{k-1}^2}\right)$.

5. $\mu_k := \beta_k + \frac{t_{k-1}-1}{t_k}(\beta_k - \beta_{k-1})$.

6. Para $i = 1, \ldots, N$ calcular:

$$\lambda_i^*(\mu_k) = \arg\min_{\lambda_i \in \mathbb{R}} w_i\lambda_i^2 + \gamma_i|\lambda_i| + (\mu_k^\top r_i)\lambda_i$$

7. Calcular

$$g_k = \left(\sum_{i=1}^{N} r_i\lambda_i^*(\mu_k) - r\right).$$

8. $\beta_{k+1} = \mu_k + (RH^{-1}R^\top)^{-1}g_k$.

9. Si $\|g_k\|_2 \leq \varepsilon$ entonces SALIR.

10. $k = k+1$ y IR A (4).

Teniendo el algoritmo ISTA programado en Matlab previamente como base, esta variante no es difícil de programar.

Aunque el algoritmo FISTA es una mejora frente a ISTA, sigue siendo lento, en especial cuando se trabaja con matrices de datos muy grandes. Se puede tratar de rebajar el tiempo de convergencia en estos casos mediante equivalentes matemáticos que simplifiquen los cálculos en los pasos más demandantes del algoritmo.

Uno de los pasos más demandantes del algoritmo, es el calculo de β_{k+1} en FISTA y μ_{k+1} en ISTA:

$$\beta_{k+1} = \mu_k + (RH^{-1}R^\top)^{-1}g_k \tag{4.8}$$

$$\mu_{k+1} = \mu_k + (RH^{-1}R^\top)^{-1}g_k \tag{4.9}$$

Estos son equivalentes a

$$(RH^{-1}R^{\top})\beta_{k+1} = g_k + (RH^{-1}R^{\top})\mu_k \tag{4.10}$$

Luego, el calculo de β_{k+1} es equivalente a la solución de un sistema de ecuaciones. Hay diferentes opciones a la hora de resolver un sistemas de ecuaciones. La primera es calcular la descomposición de la matriz $Q = RH^{-1}R^{\top}$, por ejemplo, la descomposición cholesky, o la descomposición QR. La descomposición cholesky puede obtenerse con el comando propio de Matlab (chol).

Si $Q \geq 0$, la descomposición de cholesky es $Q = S^T S$, donde S es triangular. La idea es resolver el problema $Qx = y$, como $S^T Sx = y$ en dos pasos:

$$S^{\top} z = y \tag{4.11}$$

$$Sx = z \tag{4.12}$$

Donde $y = g_k + (RH^{-1}R^{\top})\mu_k$ y $x = \beta_{k+1}$

La descomposición debe ejecutarse fuera del bucle, para que no se recalcule en cada iteración, ya que no depende de μ o β, y por lo tanto no varia.

Algo a tener en cuenta sobre este algoritmo, es que muestra un comportamiento indeseado, tiende a producir oscilaciones periódicas que realentizan su convergencia. Frente a este comportamiento es habitual utilizar RESTART SCHEMES, para mejorar la convergencia. Podemos ver ejemplos en los que se aplican estas metodologías con éxito en [2].

4.6 Identificar las λ_i que no cambian de signo

Es habitual en estos problemas tratar con un número muy elevado de muestras, esto aumenta el tiempo de convergencia de los algoritmos, por ello es interesante reducir el impacto del número de muestras en el tiempo de convergencia. Para ello Teodoro Álamo sugiere la siguiente solución para reducir el tiempo de calculo de g_k Supongamos que sabemos los signos de un subconjunto de $\lambda_i = 1, \ldots, N$

Llamaremos a este subconjunto S, supongamos entonces que

$$signo(\lambda_i) = s_i, \forall i \in S$$

entonces para cada $i \in S$, se cumple $|\lambda_i| = s_i \lambda_i$. Luego

$$\begin{aligned}
\lambda_i^*(\mu_k) &= \arg\min_{\lambda_i\in\mathbb{R}} w_i\lambda_i^2 + \gamma_i|\lambda_i| + (\mu_k^\top r_i)\lambda_i \\
&= \arg\min_{\lambda_i\in\mathbb{R}} w_i\lambda_i^2 + \gamma_i s_i\lambda_i + (\mu_k^\top r_i)\lambda_i \\
&= \arg\min_{\lambda_i\in\mathbb{R}} w_i\lambda_i^2 + (\gamma_i s_i + \mu_k^\top r_i)\lambda_i \\
&= -\left(\frac{\gamma_i s_i + \mu_k^\top r_i}{2w_i}\right).
\end{aligned}$$

Esta nueva forma de calcular λ_i para $i \in S$, nos permite ahora escribir

$$\begin{aligned}
g_k &= \left(\sum_{i=1}^{N} r_i\lambda_i^*(\mu_k) - r\right) \\
&= -r + \sum_{i\notin S} r_i\lambda_i^*(\mu_k) + \sum_{i\in S} r_i\lambda_i^*(\mu_k) \\
&= -r + \sum_{i\notin S} r_i\lambda_i^*(\mu_k) + \sum_{i\in S} -r_i\left(\frac{\gamma_i s_i + \mu_k^\top r_i}{2w_i}\right) \\
&= -r + \sum_{i\notin S} r_i\lambda_i^*(\mu_k) - \sum_{i\in S}\left(\frac{\gamma_i s_i}{2w_i}\right) - \left(\frac{r_i r_i^\top}{2w_i}\right)\mu_k
\end{aligned}$$

Con esto se consigue sacar μ_k del sumatorio, para los valores λ_i que pertenecen al conjunto para el que sabemos los signos, esto permite calcular $\sum_{i\in S}\left(\frac{\gamma_i s_i}{2w_i}\right)$ y $\left(\frac{r_i r_i^\top}{2w_i}\right)$ fuera del bucle ya que solo aparecen valores constantes, acelerando la convergencia.

4.6.1 Criterio para fijar los signos

Para poder aplicar este método es necesario establecer un criterio que determine que signos debemos fijar, hay multitud de opciones que se pueden implementar, por ejemplo,

1. Obtener los λ_i óptimos para $\gamma = 0$.
2. Realizar 10 iteraciones con el γ definitivo.
3. Comparar los signos de λ_i obtenidos en los dos primeros pasos.
4. Fijar los signos que se mantengan.
5. Si $\|g_k\|_2 \leq \varepsilon$ entonces SALIR.

Esta es una heurística sencilla y que no tiene en cuenta las características del problema a resolver. Los diferentes criterios para fijar los signos pueden funcionar mejor o peor

dependiendo del problema, es por ello que es interesante explorar las distintas opciones.

A continuación se exploraran diferentes heurísticas, cada vez más complejas y más personalizadas al problema en cuestión.

En primer lugar, se añaden actualizaciones de los signos fijados, cada cierto número de iteraciones del algoritmo.

1. Obtener los λ_i óptimos para $\gamma = 0$.

2. Realizar 10 iteraciones con el γ definitivo.

3. Comparar los signos de λ_i obtenidos en los dos primeros pasos.

4. Fijar los signos que se mantengan.

5. Aplicar algoritmo.

6. Cada cierto número de iteraciones comparar signos.

7. Fijar signos que se mantengan.

8. Si $\|g_k\|_2 \leq \varepsilon$ entonces SALIR.

Con esta opción se aumenta el número λ_i con el signo fijo, de forma que se reduce aun más el cálculo según avanza el algoritmo.

Otra opción similar es comparar los λ_i, cuando se produzca una reducción establecida del gradiente, en vez de un número establecido de iteraciones.

1. Obtener los λ_i óptimos para $\gamma = 0$.

2. Realizar 10 iteraciones con el γ definitivo.

3. Comparar los signos de λ_i obtenidos en los dos primeros pasos.

4. Fijar los signos que se mantengan.

5. $\|g_k\|_c = \|g_k\|_2$.

6. Aplicar algoritmo.

7. Si $\|g_k\|_2 \leq \|g_k\|_c \cdot 0.1$ entonces comparar signos.

8. Fijar signos que se mantengan y IR a (5)

9. Si $\|g_k\|_2 \leq \varepsilon$ entonces SALIR.

En el paso 7, se han probado valores distintos al 0.1, y se han obtenido valores muy similares, por lo que en los resultados solo se analiza un caso.

5 Posibles aplicaciones

En este capítulo se proponen algunas de las posibles aplicaciones en el mundo real del problema planteado en el Capítulo 3. La función del problema planteado es la de realizar predicciones, para ello usa muestras pasadas y establece la similitud o disimilitud de estas con la muestra a pronosticar. Por ello para que sea viable su aplicación, es importante la accesibilidad a bases de datos y que haya una relación entre entradas y salidas.

A continuación se analizarán algunas de estas posibles aplicaciones y su viabilidad, teniendo en cuenta la cantidad de datos disponible y la relación causa-efecto.

5.1 Depósitos geológicos

El Kriging se desarrolló en el ámbito de la ingeniería de minas, precisamente para predecir las características de los depósitos geológicos, sin necesidad de realizar costosas exploraciones, por ello una de las posibles aplicaciones de la metodología que se plantea en este documento es realizar estas predicciones. Se ha mencionado anteriormente el uso de metodologías kriging para el estudio de reservas de C02 subterráneo, pero hay muchos más ejemplos de yacimientos geológicos que se pueden estudiar con estas técnicas.

Puede utilizarse para evaluar la existencia de depósitos de gas natural o petróleo, estos son muy caros de explorar y puede ser muy beneficioso usar la información de la que ya se dispone para ayudar a estimar la probabilidad de existencia de uno de estos depósitos subterráneos.

Los datos a usar en las muestras pasadas $\vec{d}_i \in \mathbb{R}^n$ dependen de la predicción especifica que se quiera realizar, pero algunos ejemplos de posibles elementos a introducir pueden ser:

- Datos de perforación: Los datos de perforación, como los núcleos de perforación o los registros de pozos, proporcionan información directa sobre las características del depósito en puntos específicos. Estos datos son fundamentales para comprender la distribución vertical de las características del depósito.

- Datos de muestreo de superficie: Las muestras de superficie recolectadas a lo largo de la superficie del terreno pueden proporcionar información sobre la variabilidad horizontal de las características del depósito. Estos datos son especialmente útiles para caracterizar la distribución lateral del depósito.

- Datos geofísicos: Los datos geofísicos, como las mediciones de resistividad eléctrica, la gravedad o el magnetismo, pueden proporcionar información complementaria sobre la estructura y la composición del depósito. Estos datos pueden ayudar a mejorar la precisión del modelo de kriging al proporcionar información adicional sobre la variabilidad del depósito.

- Datos históricos: Los datos históricos de exploración, como los informes de perforación anteriores o los mapas geológicos, pueden proporcionar información contextual sobre el depósito y ayudar a guiar la selección de sitios de muestreo.

Sin embargo es importante entender sus limitaciones y usar técnicas complementarias, es posible que los datos de muestra sean limitados o de baja calidad y que exista mucha variabilidad y complejidad geológica, lo que puede perjudicar a la precisión de las predicciones.

La exploración de hidrocarburos suele incluir una variedad de técnicas que pueden complementar a la que se propone, como estudios sísmicos, análisis geoquímicos y perforaciones exploratorias o de muestreo.

Figura 5.1 Histórico precio petróleo.

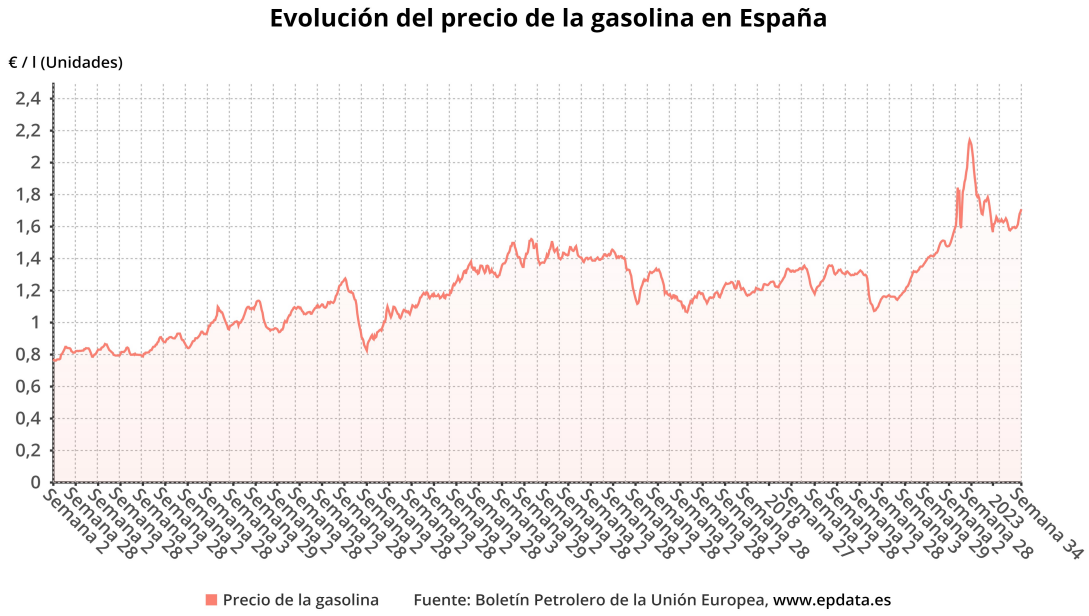

Figura 5.2 Histórico precio gasolina España.

Aunque el precio de la gasolina Figura 5.2 no guarda una relación directa con el precio del

barril de petróleo Figura 5.1, ya que también influyen factores como el coste de producción o los impuestos, sí es un factor a tener en cuenta y en la Figura 5.2 se puede ver la importancia y escasez que tiene este recurso en la actualidad. Es por esto que es muy necesaria la eficacia en la explotación de los yacimientos de petróleo.

Figura 5.3 Histórico precio gas natural.

La reciente Guerra entre Rusia y Ucrania, ha creado un ambiente de tensión política entre Rusia y los países occidentales. Siendo Rusia el principal proveedor de gas natural de Europa, se ha elevado el precio de este bien en momentos puntuales, como consecuencia de las restricciones impuestas por Rusia. Esto hace que sea cada vez más importante encontrar proveedores de gas natural en países aliados, de forma que los países europeos no se vean afectados en gran medida por las posibles restricciones impuestas, por lo que crecen en importancia este tipo de técnicas.

5.2 Medio ambiente

De una forma muy similar a la que se utiliza para realizar predicciones sobre depósitos geológicos, se podría plantear esta metodología para predecir diferentes valores en relación con el medio ambiente.

En primer lugar, hay una extensa red de estaciones meteorológicas en España que

proporcionan datos en tiempo real, sin embargo, estas son fuentes de información muy local, por lo que es necesario realizar predicciones basadas en estas muestras para dar un dato aproximado de la calidad del aire, temperatura, velocidad del viento, índice uv o dispersión de contaminantes en las zonas no muestreadas.

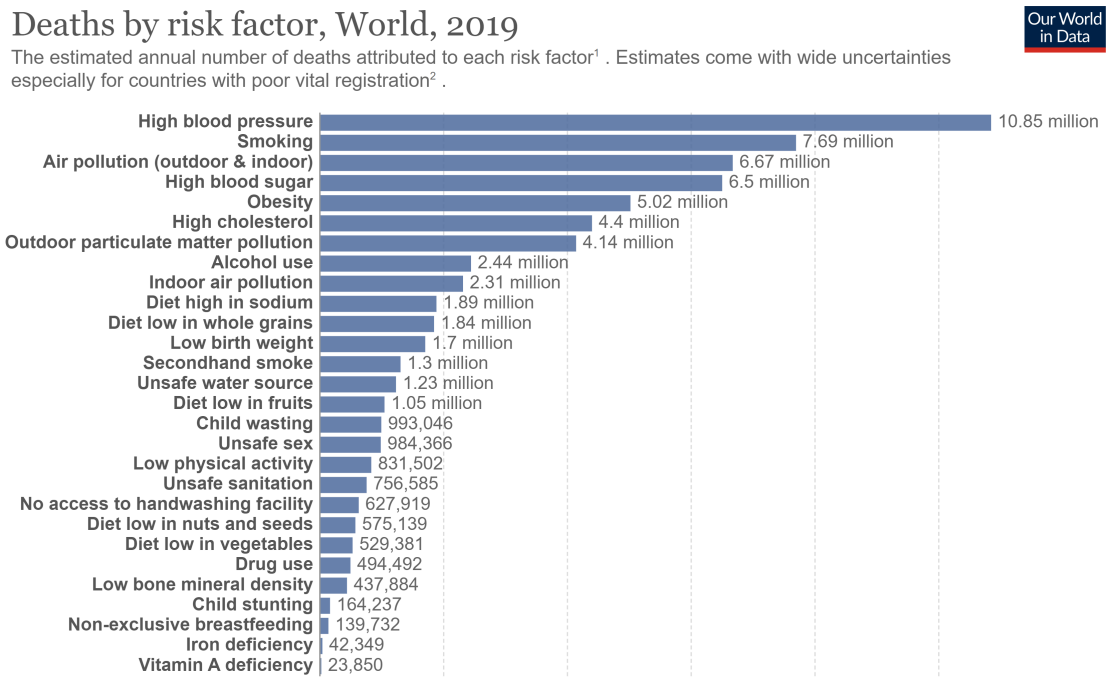

Figura 5.4 Número de muertes por factor de riesgo.

En Figura 5.4 se muestra que en el año 2019 la polución del aire llegó a ser el tercer factor de riesgo que más muertes provoca.

Por otra parte, puede ayudar a la gestión sostenible de recursos como el agua, el suelo y la biodiversidad. Permite estimar la cantidad y calidad de estos recursos, complementando a otras técnicas en la toma de decisiones para su uso y conservación.

En las zonas urbanas, los modelos pueden predecir el crecimiento de las ciudades, la expansión urbana y sus efectos en el medio ambiente, un ejemplo puede ser la contaminación acústica, ya que permite la elaboración de mapas de niveles de ruido en toda la ciudad, de forma que se identifiquen las zonas de alto y bajo ruido. Los resultados de las predicciones pueden emplearse para tomar decisiones informadas en la planificación de la ciudad y la eliminación de la contaminación acústica. Ayudando a diseñar medidas para reducir el ruido en zonas críticas.

En zonas del planeta donde los desastres naturales son frecuentes, se puede aprovechar la información obtenida en pasados desastres para predecir los desastres naturales futuros, como inundaciones, terremotos o tsunamis.

En cuanto al cambio climático, puede usarse para predecir sus efectos a corto, medio y

largo plazo, lo que es esencial para entender el cambio climático y tomar medidas para mitigar sus efectos.

El estudio de la biodiversidad también puede complementarse con estas técnicas, los modelos pueden predecir la distribución de especies animales y hábitats, de forma que se identifiquen las áreas con mayor prioridad para la conservación y ayudando en la toma de decisiones sobre la protección de las especies.

La protección de la biodiversidad es un tema muy relevante en la actualidad, durante años no se llevó a cabo ningún tipo de control sobre la caza y venta de animales y eso ha provocado la extinción de una gran cantidad de especies animales.

Aunque el cambio climático fue un factor importante, hay pruebas que indican que la sobre explotación y cambios drásticos en los hábitat naturales llevados a cabo por los seres humanos también influyeron en gran medida.

Desde la industrialización a principios del siglo XIX, se ha incrementado el ritmo de extinción animal Figura 5.4, es por ello que se está tratando de revertir esta situación y cobran gran importancia en la actualidad las medidas para la protección de las especies.

Los modelos predictivos como el que se propone, pueden ser útiles en concreto para la identificación de áreas prioritarias de conservación, para ello se pueden usar como muestras $\vec{d}_i \in \mathbb{R}^n$, datos sobre la distribución de las especies, la abundancia de especies, la diversidad genética y la estructura del hábitat.

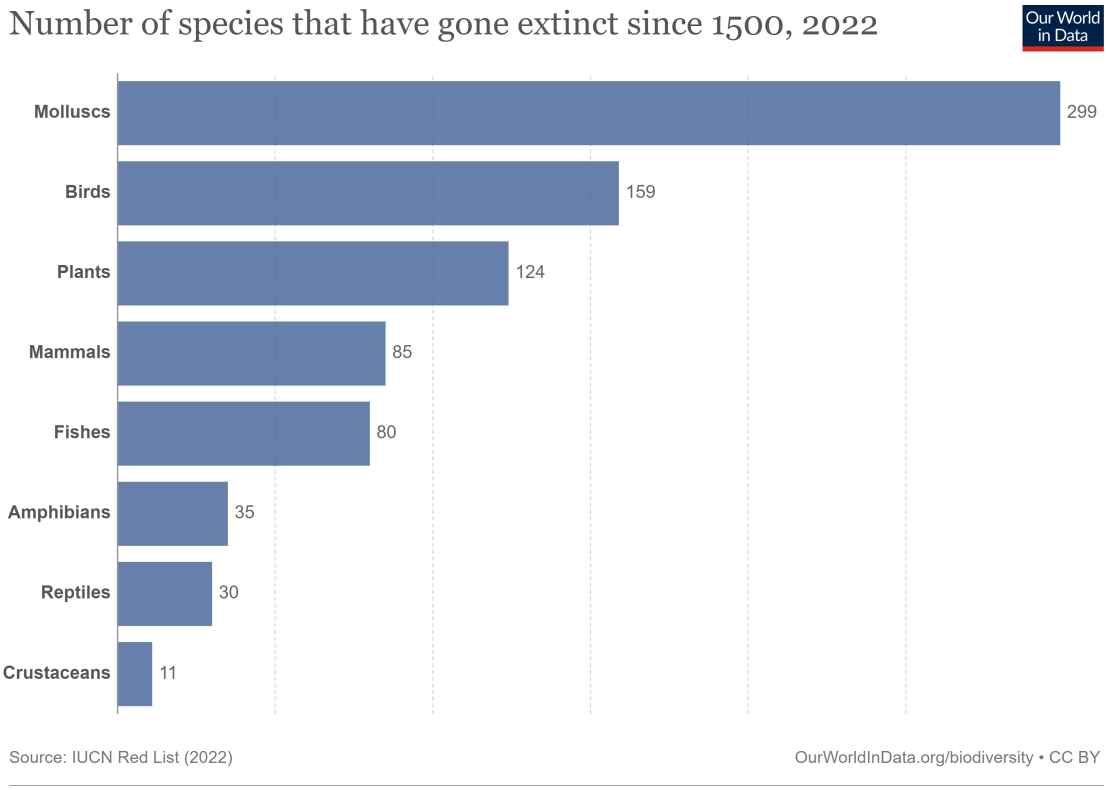

Figura 5.5 Especies extintas desde el 1500.

5.3 Industria

La industria manufacturera, está en constante búsqueda de herramientas y técnicas para mejorar sus procesos, abaratar costes y garantizar calidad, impulsados por el aumento de la demanda de productos de alta calidad y eficiencia. En este ambiente de búsqueda de mejora continua y eliminación de despilfarros de recursos, los métodos Kriging o similares, como el que se propone, han surgido como una poderosa herramienta para afrontar los desafíos de la industria moderna.

Estos métodos que tienen la capacidad de hacer predicciones de manera precisa, se han convertido en una importante estrategia en la búsqueda de la optimización de procesos de fabricación, la identificación y corrección de defectos, y la mejora continua de la calidad.

5.3.1 Mejora continua-Lean

Desde el origen de los modelos de fabricación de Lean, en Japón en 1950 a manos de la empresa automovilística Toyota, La gran mayoría de empresas han acabado adaptando este modelo o alguna variación de este, logrando un aumento muy significativo en la producción, como puede observarse en Figura 5.6, donde se muestra la producción de Toyota a lo largo de los años, y se aprecia como a partir de 1950 empieza a crecer rápidamente. Se trata de

un modelo de producción de mejora continua basado en la eliminación de despilfarros.

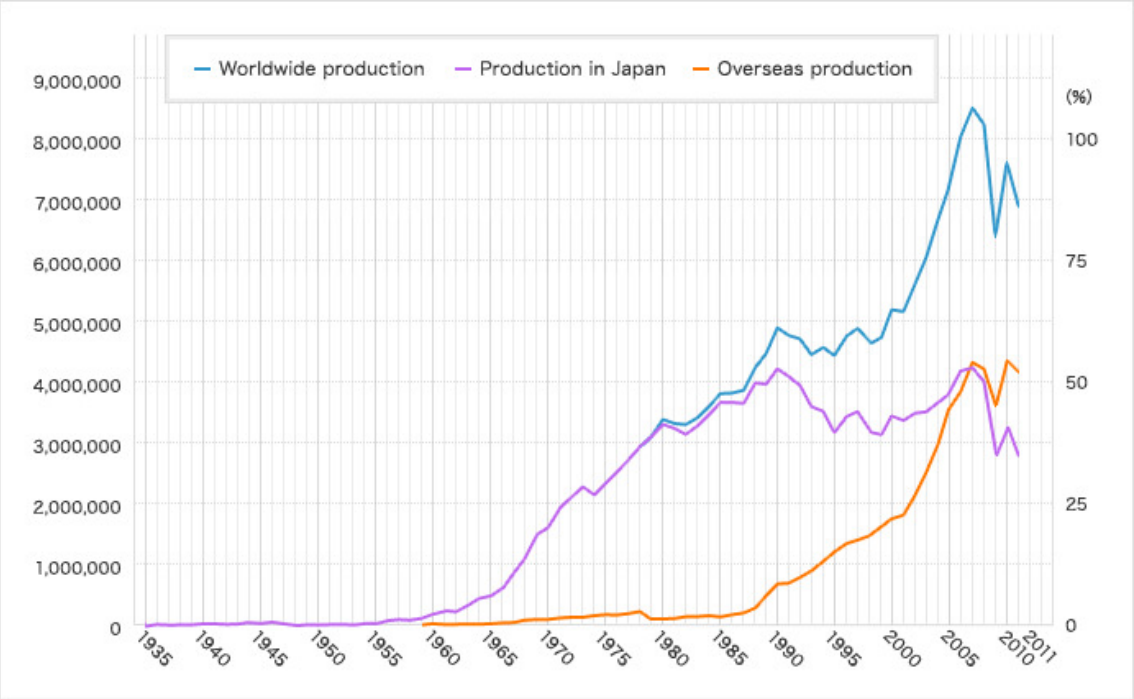

Figura 5.6 Producción de Toyota desde 1935.

Para entender como puede ser útil la metodología propuesta a este caso es importante conocer cuales son los 8 despilfarros de la metodología Lean:

- Despilfarros por sobreproducción: Se trata de los desperdicios ocasionados por producir más de lo que se necesita, esto puede concurrir en una acumulación de inventario excesiva, aumento de los costos de almacenamiento, deterioro del producto terminado debido a un almacenamiento prolongado o incluso a deshacerse del stock.

- Despilfarros por tiempo de espera o tiempo vacío: Tiempo en el que las piezas, productos o empleados están inactivos, estas esperas están provocadas por la falta de un flujo de trabajo continuo, para que haya un flujo de trabajo continuo se busca un flujo pull, es decir, que se fabrique en función a la demanda real y no a la previsión de demanda.

  Para que este flujo sea posible cada estación del proceso productivo debe 'Tirar'de la estación previa, por lo que las plantas de producción deben estar diseñadas de forma que cada estación tenga una duración similar, los materiales necesarios esten en sus ubicaciones en el momento preciso y el numero de operarios y máquinas deber ser el planificado en el diseño de las estaciones.

- Despilfarros por transporte: Movimientos innecesarios o más largos de lo necesario, provocados por un mal diseño de planta, o una distancia excesiva entre proveedores

y clientes.

- Despilfarros por movimientos innecesarios: Similar a los despilfarros de transporte, pero más enfocado a los movimientos dentro de cada puesto de trabajo, la ergonomía y el diseño de cada estación son fundamentales para realizar cada proceso con los mínimos movimientos posibles.

- Despilfarros por sobreproceso: Los despilfarros por sobreproceso se producen al realizar cualquier trabajo que no influya en la valoración del producto por parte del cliente, es decir, cualquier característica o tratamiento que no haga que el cliente pague más, es un despilfarro.

  Es importante distinguir este despilfarro de los trabajos que no aportan valor físico al producto, pero que pueden hacer que la percepción de la marca o producto mejore y sea mejor valorada, por ejemplo, cumplir normativas para obtener certificaciones medioambientales, no mejoran el producto en si, pero si la imagen.

- Despilfarros por exceso de inventario: Mantener más inventario del necesario, no solo aumenta el coste de almacenamiento y gestión del inventario, además tener un inventario muy elevado puede ocultar problemas de producción, por ejemplo, si hay piezas que se rompen constantemente pero siempre hay stock suficiente para reemplazarlas o si se mantiene mucho inventario del producto entre fases de forma que nunca falte para la siguiente fase, realmente lo que se esta haciendo es suplir la falta de calidad y diseño de planta con stock, de forma que la producción no para pero no se produce de forma eficiente.

- Despilfarros por defectos: La producción con defectos, requiere retrabajos o reparaciones. De esta forma aumenta el gasto en materiales y mano de obra, a la vez que se aumenta el tiempo de producción.

- Despilfarros por talento subutilizado y exceso de información: Una mala gestión del talento de los empleados, también puede considerarse un desperdicio, una persona que está realizando una tarea para la que esta sobrecualificada y que podría estar haciendo trabajos que aportaran más valor al producto es un despilfarro de personal.

  Por otro lado la obtención excesiva de información que no aporta valor al producto, es también un desperdicio de personal, se esta perdiendo tiempo en obtener unos datos que no se utilizan y que pueden llevar a la confusión.

Para evitar estos despilfarros, la metodología Lean hace uso de una serie de técnicas que mediante la acumulación de pequeñas mejoras repetidas de forma continuada puede generar una gran mejora en el cómputo global de la producción.

No en todas las técnicas de mejora continua es útil el uso de modelos predictivos, ya que algunas son herramientas muy simples que no hacen uso de la información futura de la planta o empresa. A continuación se detallaran algunas de estas técnicas y los motivos por

los que puede beneficiarse, de los modelos predictivos.

En primer lugar es esencial antes de establecer un plan de mejora continua conocer la situación actual de la fábrica, para ello se cuenta con tres herramientas que sirven para mostrar cuanto se esta desperdiciando y en que áreas es más relevante este desperdicio, esta información puede usarse para muestras que se usan de entrada en el modelo predictivo para el cual la salida podría ser, por ejemplo, la producción diaria.

- Indicadores: Los indicadores evalúan la situación actual de una empresa, es habitual que esta situación se represente en forma de porcentaje de rendimiento, y es la forma de medir el rendimiento la que varia de un indicador a otro, para calcular el rendimiento es fundamental la toma de datos, en este caso mediante un estudio de los tiempos de proceso, este estudio debe hacerse de forma que los datos no estén falseados, ya que es habitual que los operarios traten de modificar los tiempos estándar.

  Hay una gran variedad de indicadores que se usan en la actualidad pero se puede destacar el más usado, *OEE* (Overall Equipment Performance), el *OEE* es el producto de unos valores calculados de Disponibilidad, Rendimiento y Calidad, estos valores se calculan comparando los tiempos y niveles de producción esperados con los reales.

- 20 Claves: Esta es una técnica que puede ser más complicada de implementar en la metodología planteada, ya que no es algo que se suela medir diariamente o incluso varias veces al día como los indicadores, pero podría adaptarse para este caso.

  Las 20 claves para mejorar la fábrica, son 20 categorías que se describen en [28] y se considera que el estado actual de la fábrica en esos aspectos, sirve para indicar el estado de la fábrica a nivel general. La idea de esta técnica es que se realice antes de comenzar un proyecto de mejora continua, pero para la aplicación que se desarrolla en este documento se podría adaptar de forma que se establecieran unos valores diarios para estas 20 categorías o a alguna variante de estas si no tuviera sentido medirlas a diario.

  Una opción seria usar la media de todas las valoraciones como una única variable, ya que muchas de estas categorías no tienen sentido medidas por si solas, si no como un conjunto que indica el estado global de la fábrica.

  Habitualmente se da una nota de 0 a 5 a estas categorías y se representa en un gráfico radar como se ve en Figura 5.7.

  - Limpieza y organización
  - Racionalización del sistema
  - Actividades de mejoras de equipos
  - Reducción de inventarios en curso

  - Tecnologías de cambio rápido de útiles y de herramientas
  - Mejoras de métodos
  - Fabricación con supervisión cero
  - Fabricación acoplada
  - Mantenimiento de equipos
  - Control del tiempo y compromiso
  - Aseguramiento de la calidad
  - Participación de proveedores
  - Eliminación de despilfarro
  - Habilitar a los trabajadores para hacer mejoras
  - Polivalencia de los empleados
  - Programación de la producción
  - Control de la eficiencia
  - Utilización de sistemas de información
  - Ahorro de energía y materiales
  - Tecnología

- VSM: El VSM (Value Stream Mapping) consiste en representar esquemáticamente cualquier proceso productivo, logístico o administrativo, con el objetivo de identificar las operaciones que aportan valor a la fábrica y las que se consideran mudas, es decir las que no aportan valor. De esta forma se identifican las variables que son más relevantes para la producción o facturación, este tipo de esquemas puede ayudarnos a descartar la información que no es relevante para la variable que se desee predecir.

Una vez conocida la situación actual de la planta, la metodología lean propone una serie de técnicas con las que lograr la mejora continua, entre ellas se destaca TPM (Total

Figura 5.7 Ejemplo 20 claves Heineken Sevilla.

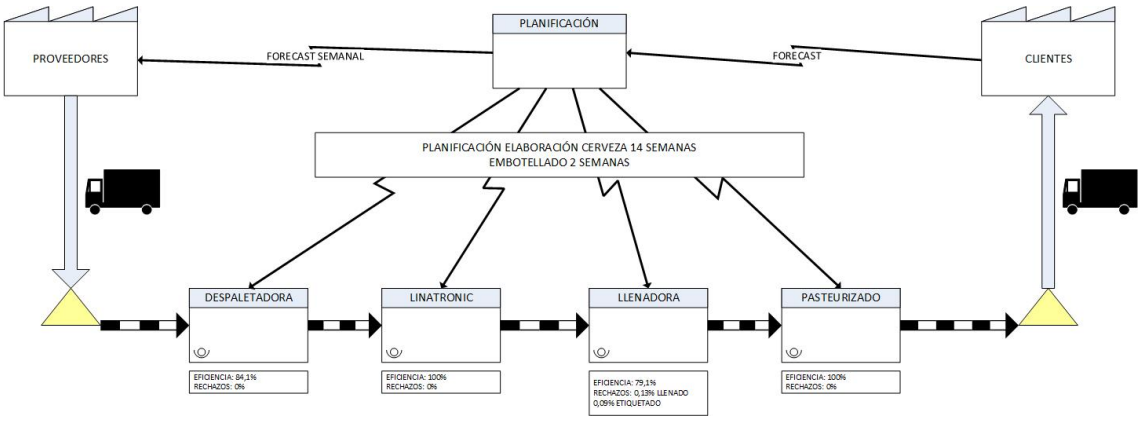

Figura 5.8 Ejemplo VSM Heineken Sevilla.

Productive Maintenance), ya que es la que tiene un uso de los modelos predictivos más directo.

El TPM fomenta involucrar a los operarios en el mantenimiento de sus equipos, con un énfasis en el mantenimiento preventivo, para mejorar la producción hasta alcanzar la perfección.

- Sin averías

- Sin pequeñas paradas ni funcionamiento lento

- Sin defectos

El mantenimiento preventivo consiste en la realización de una serie de operaciones, que pretenden evitar futuros fallos en la maquinaria, para ello se estima cuanto tiempo debe pasar entre un mantenimiento y el siguiente. Una predicción precisa de los fallos, podría reducir los costes al evitar paradas por mantenimiento que no son necesarias y paradas por fallos que se podrían haber evitado.

Puede parecer escasa la información que es posible obtener de la maquinaria para predecir sus fallos, paradas y bajadas de rendimiento, pero es algo que ya se ha estudiado con anterioridad, en [29] se emplean algoritmos de aprendizaje automático para realizar dichas predicciones, y asegura que la información recogida en SAP Enterprise Resource Planing (ERP), es suficiente para obtener una precisión de hasta un 95 %, por otro lado en [20] se hace uso de algoritmos y los datos que proporcionan los sensores de la maquinaria para predecir más del 85 % de los fallos.

5.4 Energía

Por último, en el ámbito de la energía, y en concreto de la energía eléctrica, hay una enorme cantidad de aplicaciones posibles para el método de predicción propuesto y una gran cantidad de datos que facilitan su uso. A continuación se describen brevemente algunas de estas aplicaciones.

- Producción diaria: Usando muestras de los días pasados con datos relacionados con la demanda de energía, la disponibilidad de las fuentes de generación, las políticas energéticas y su regulación, los precios de los combustibles, las condiciones climáticas, y las conexiones internacionales, pueden hacerse predicciones de la energía total producida en España, o la energía producida por cada fuente de energía de forma individual.

- Demanda diaria: En este documento es esta la aplicación elegida para poner a prueba con datos reales el predictor, de la misma forma que se calcula la producción, puede calcularse la demanda, con la diferencia de que los parámetros muestreados deben ser los que sean más significativos para el consumo de energía eléctrica, como pueden ser, la época del año, la temperatura, el día de la semana, y la políticas energéticas. Otra opción es usar la demanda de los días anteriores como muestra, esta es la metodología que se usará en este documento a la hora de aplicar a datos reales el predictor.

- Precio: De forma muy similar a la producción y consumo de energía, se puede tratar de predecir el precio de la luz, para ello habría que añadir importancia a los datos relevantes en el ámbito económico y político, por ello el precio de la luz puede ser más complicado de predecir que las opciones anteriores.

6 Demanda eléctrica

En este capítulo se detallará la aplicación de los algoritmos descritos en el capítulo 4 a un ejemplo real, en este caso se trata de predecir la demanda de energía eléctrica en España (península) en un día concreto, para ello, es necesario ajustar los parámetros de los algoritmos a las características del problema y del set de datos.

Se explicarán los motivos de la elección del set de datos empleado, así como la fuente de origen de este, por otra parte, es necesario también determinar la metodología para establecer los pesos de las muestras, y el parámetro γ.

6.1 Datos

Para realizar la predicción de la demanda de energía en España peninsular, se usaran como muestra los datos de la demanda de los 50 días previos, es decir, para cada muestra usaremos de entrada los valores de la demanda de los 50 días previos al día correspondiente a dicha muestra, y como salida la demanda del día de la muestra.

Se usará un set de datos que incluye 731 muestras, estos son los datos de demanda eléctrica diaria durante 2016 y 2017, con este set de datos se tratará de predecir la demanda de 730 días, los cuales corresponden con los años 2018 y 2019, se han elegido estas fechas, con la intención de evitar los posibles efectos de la pandemia de Covid-19.

El set de datos ha sido obtenido de la empresa Redeia [41], la cual opera en España con el nombre de Red Eléctrica de España, Red Eléctrica de España es propietaria de toda la red española de electricidad en alta tensión, como operador del sistema eléctrico español, establece las previsiones de la demanda de energía eléctrica.

Los datos de Redeia se exportan en forma de hoja de cálculo, en la que aparece cada fecha con su demanda de energía eléctrica correspondiente, para poder trabajar con estos datos en Matlab, hay que hacer un tratamiento previo para que se pueda exportar a Matlab y el formato corresponda con el que se requiere.

Con los datos de la demanda hay que elaborar una matriz en la que aparezcan para cada

muestra la demanda de los 50 días previos, esta se usará como set de datos. Por otra parte también son necesarios un vector con los datos de salida de cada muestra, para obtener las predicciones, y otro vector con los datos reales que se han tratado de predecir para poder evaluar la exactitud del método.

Los datos de las fechas, se usarán además, para la determinación de los pesos de las muestras, ya que como se verá a continuación, estos van a depender del mes y dia de la semana de cada muestra.

6.2 Pesos w_i y parámetro γ

Los pesos son una forma de introducir información local en el modelo, de esta forma no tienen el mismo peso todas las muestras, las muestras que correspondan a días más parecidos al día del que se busca hacer la predicción, tendrán un peso menor y por tanto menor penalización.

Los pesos deben establecerse de forma específica para las características de cada problema, es una de las limitaciones de estos problemas, ya que la eficacia del modelo se ve condicionada por el ajuste de los pesos que se realice.

En este caso se deben tener en cuenta las características que afectan a la demanda de energía eléctrica, y establecer una asignación de pesos en base a ellas, en este caso se han tenido en cuenta tres categorías y en cada categoría cada muestra obtendrá una puntuación, siendo el peso asignado a cada muestra la suma de las tres puntuaciones.

En base a las temperaturas medias de cada mes, se establece una puntuación entre 0 y 4 para cada mes, teniendo en cuenta la diferencia de temperatura media entre el mes de la muestra a la que se asigna el peso, y el mes del día que se pretende predecir. Para ello se usa el siguiente criterio.

- $0 \leq \triangle$ T $^{\circ}$C $< 3 \rightarrow 0$
- $3 \leq \triangle$ T $^{\circ}$C $< 6 \rightarrow 1$
- $6 \leq \triangle$ T $^{\circ}$C $< 9 \rightarrow 2$
- $9 \leq \triangle$ T $^{\circ}$C $< 13 \rightarrow 3$
- $13 \leq \triangle$ T $^{\circ}$C $\leq 17 \rightarrow 4$

Se dividen los días de la semana en dos categorías, la primera corresponde a los días laborables de lunes a jueves, los cuales se ha considerado que tienen unas similares en cuanto a la demanda energética, la segunda categoría de días incluye viernes, sábado y domingo, de nuevo porque se consideran parecidos en cuanto a demanda energética.

El día de la semana se ha considerado el criterio más determinante a la hora de establecer el peso de cada muestra, es por ello que si la muestra y el día a predecir coinciden en categoría de día se otorga una puntuación de 0, y si no coinciden se obtiene una puntuación de 5, de forma que se vean penalizadas de gran manera.

Por último se valora de 0 a 3, la muestra más próxima al día que se trata de predecir, tendrá una puntuación de 0 y se reducirá la puntuación a razón de 3/731 cada día que se aleje.

El parámetro de compromiso γ_i, se establece de forma que se obtenga la mejor predicción posible, para ello una opción es probar con diferentes valores y se observan los resultados, para simplificar la aplicación del algoritmo se tendrá el mismo valor para todos los γ_i, y se ha seleccionado los valores $\gamma_i = 1$, ya que se obtiene una buena aproximación que será suficiente para el propósito de este documento.

7 Resultados y conclusiones

En este capítulo se analizan los resultados obtenidos en la aplicación de los diferentes algoritmos planteados al caso real de predicción de demanda de energía eléctrica en España peninsular, para ello se hace uso de diferentes gráficas y datos concretos que permiten visualizar de forma sencilla el rendimiento de los algoritmos y los patrones que presentan.

Para el análisis del comportamiento de los algoritmos aplicados, los parámetros más relevantes son la veracidad de los resultados, la precisión entre los distintos algoritmos y en especial los datos relativos a la velocidad de ejecución, como pueden ser el número de iteraciones, el tiempo de ejecución y el número de operaciones realizadas.

7.1 Veracidad y precisión

Este documento no se centra en la veracidad y precisión de los algoritmos empleados, ya que estos son métodos que ya se conoce que son efectivos para la predicción, sin embargo es importante tenerlo en cuenta ya que si nos pueden indicar errores en los algoritmos, y en el ajuste de parámetros.

Con la intención de poder calcular la veracidad o exactitud del predictor, se predicen valores de fechas pasadas y cuyos valores reales son conocidos. Se ha calculado la diferencia entre el valor real y el valor del predictor para cada predicción realizada, tanto en valor absoluto como en porcentaje.

En Figura 7.1, Figura 7.2, Figura 7.3, Figura 7.4 se pueden observar el valor real y el de la predicción para cada una de las predicciones, se ha dividido la gráfica en 4 tramos para mejorar la legibilidad de las gráficas, se puede apreciar que en general la predicción es bastante cercana al valor real, aunque si se observan un cierto número de predicciones con un mayor déficit de exactitud.

Esto resulta en una media de error en la predicción del 2.32 %, por lo que se considera que para los objetivos de este documento es suficiente, aunque como se verá más adelante, hay indicios de que los parámetros del modelo podrían estar mejor ajustados.

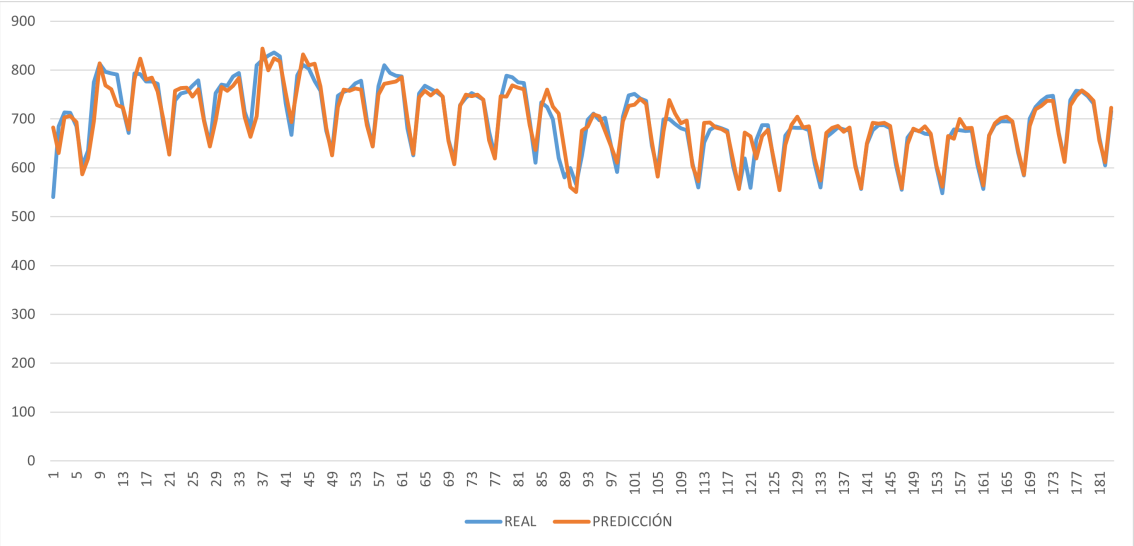

Figura 7.1 Diferencia absoluta entre real y predicción tramo 1.

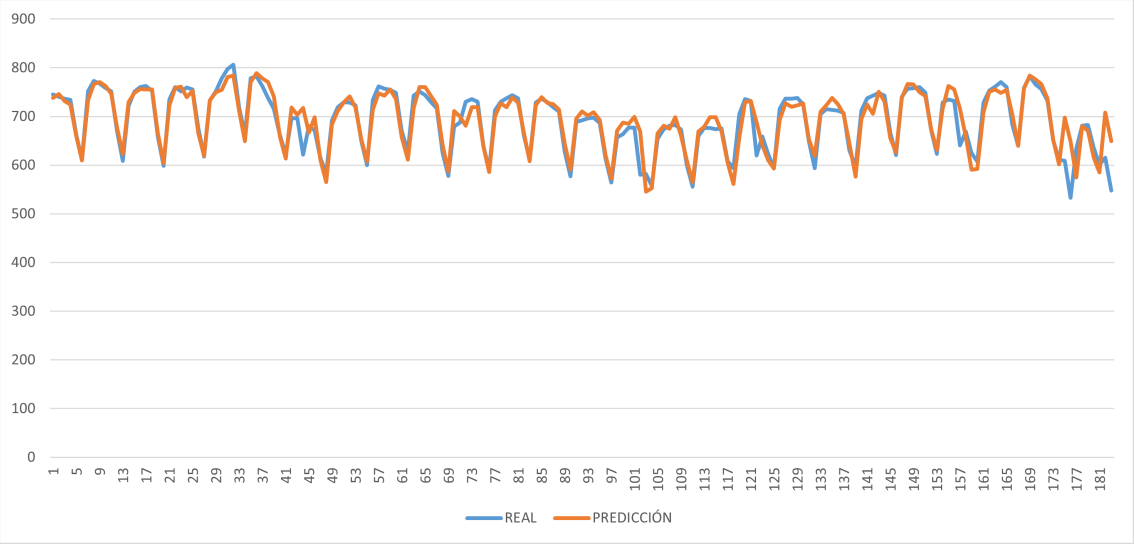

Figura 7.2 Diferencia absoluta entre real y predicción tramo 2.

En Figura 7.5 se muestra el error de cada predicción como porcentaje del valor real, ya en esta gráfica parece haber una mayor concentración de malas predicciones en los meses iniciales y finales del año, aunque no se de forma muy marcada.

Por último en Figura 7.6 se muestra un histograma del error de las predicciones en el que se puede observar que la gran mayoría de predicciones tienen un error porcentual de entre un 0 y un 2.41 %, por lo que pese a haber casos con grandes errores, estos son muy puntuales y de ahi que la exactitud media resulte en un 97.68 %.

En cuanto a la precisión de los diferentes algoritmos, al ser todos variaciones de los algoritmos conocidos ISTA (Iterative Shrinkage-Thresholding Algorithm) y FISTA (Fast Iterative Shrinkage-Thresholding Algorithm), todos obtienen de forma consistente el mismo resultado, ya que las variaciones implementadas buscan acelerar la ejecución y reducir el número de operaciones a realizar pero no cambian los resultados.

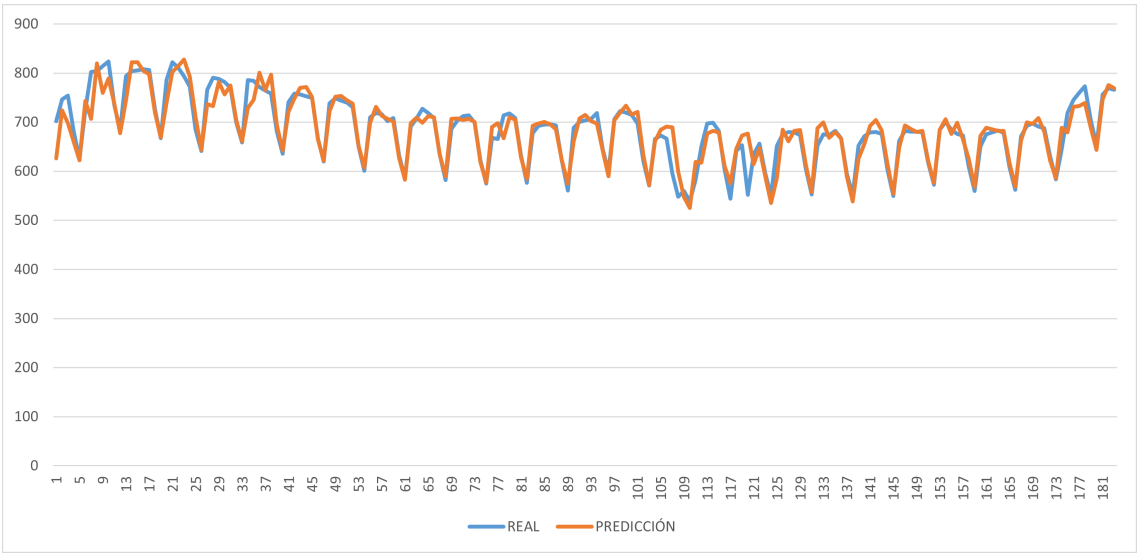

Figura 7.3 Diferencia absoluta entre real y predicción tramo 3.

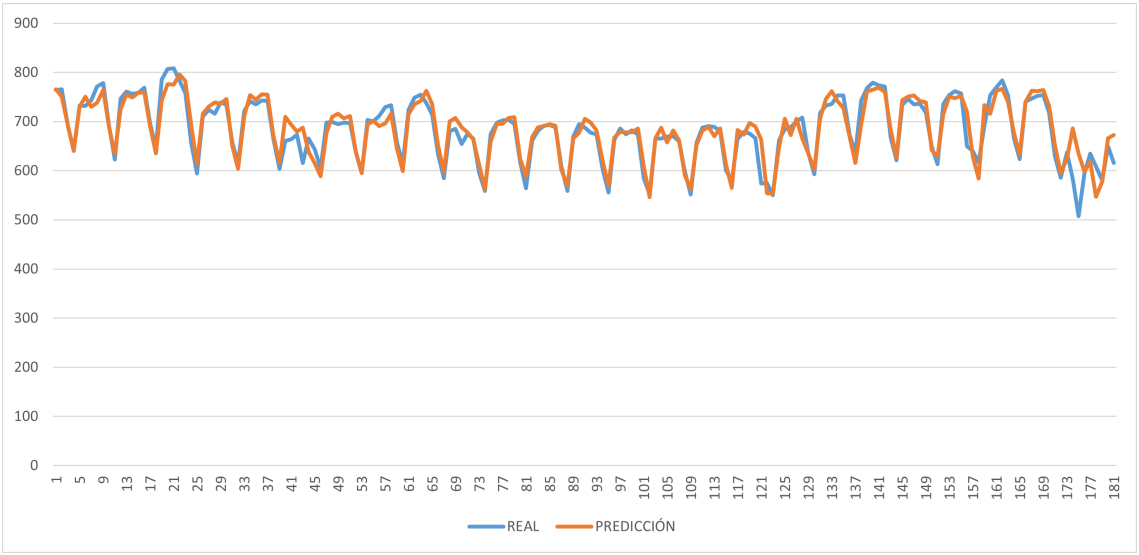

Figura 7.4 Diferencia absoluta entre real y predicción tramo 4.

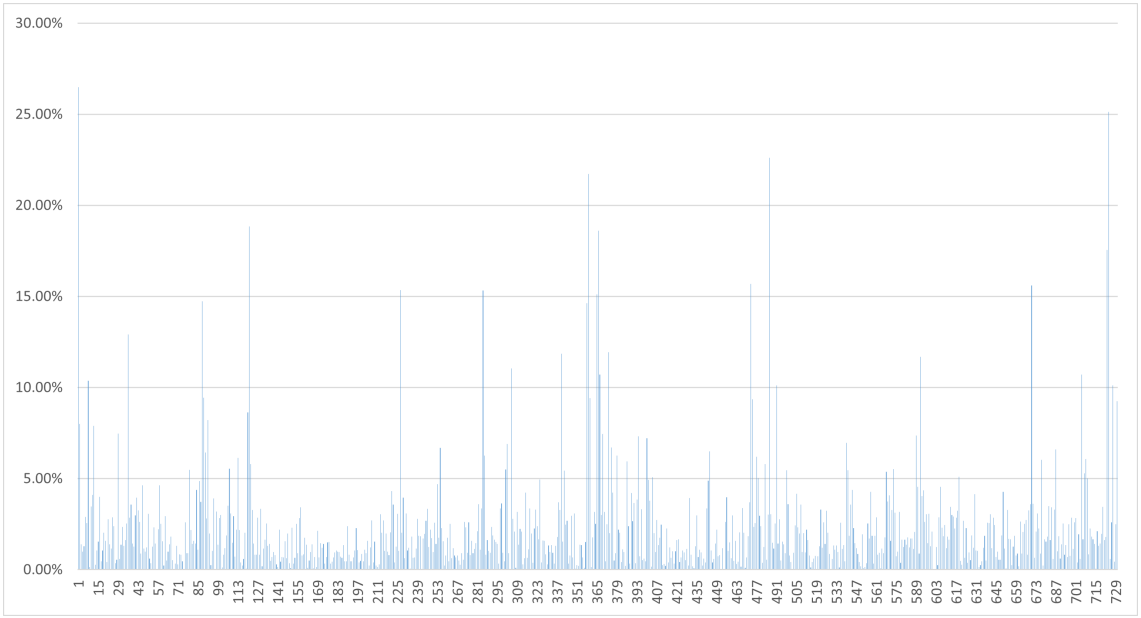

Figura 7.5 Porcentaje de error en las predicciones.

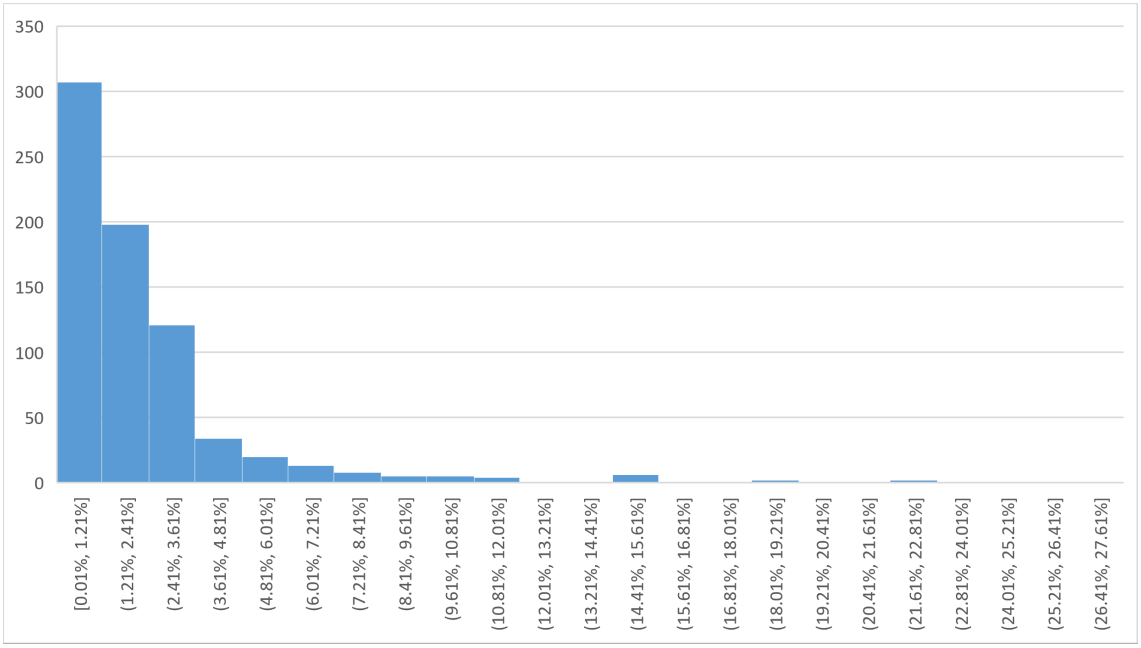

Figura 7.6 Histograma de error en las predicciones.

7.2 Iteraciones, tiempo de ejecución y número de operaciones

Esta es la sección más importante de los resultados, ya que se trata del tema principal del documento y donde se verá si se han conseguido los objetivos establecidos.

Para el correcto análisis de los resultados de los diferentes algoritmos se realizan una serie de gráficas que muestran la misma información para cada variante del algoritmo, de forma que se pueda hacer una comparación directa entre las variantes.

Como ya se ha mencionado previamente, el resultado de la predicción se mantiene en todos los algoritmos por lo que en esta sección las gráficas se centran exclusivamente en datos relativos a la velocidad de ejecución.

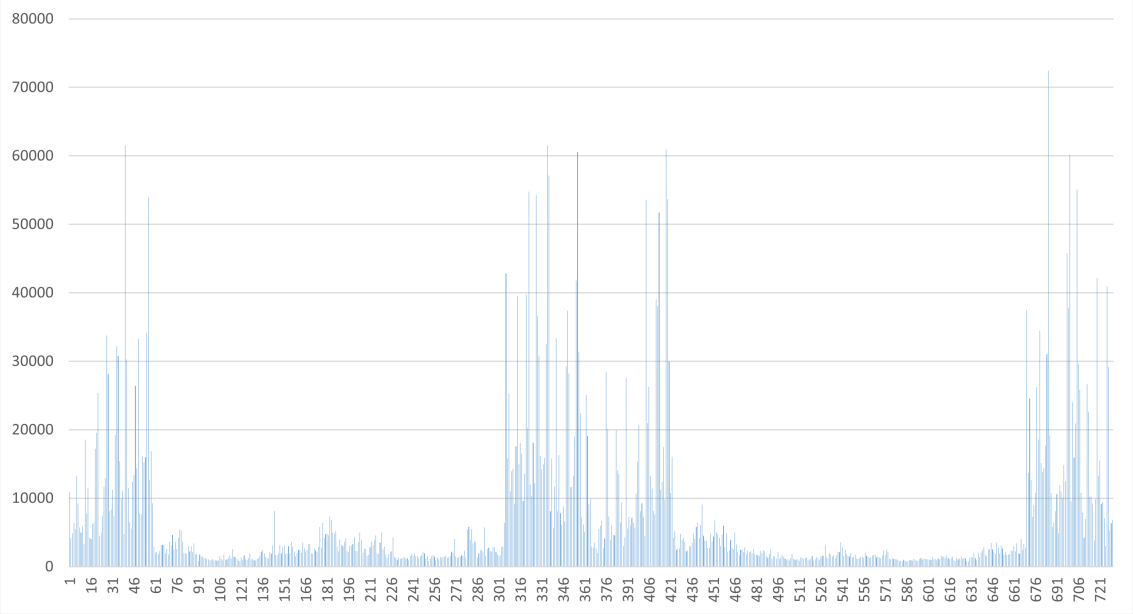

Figura 7.7 Número de iteraciones ISTA.

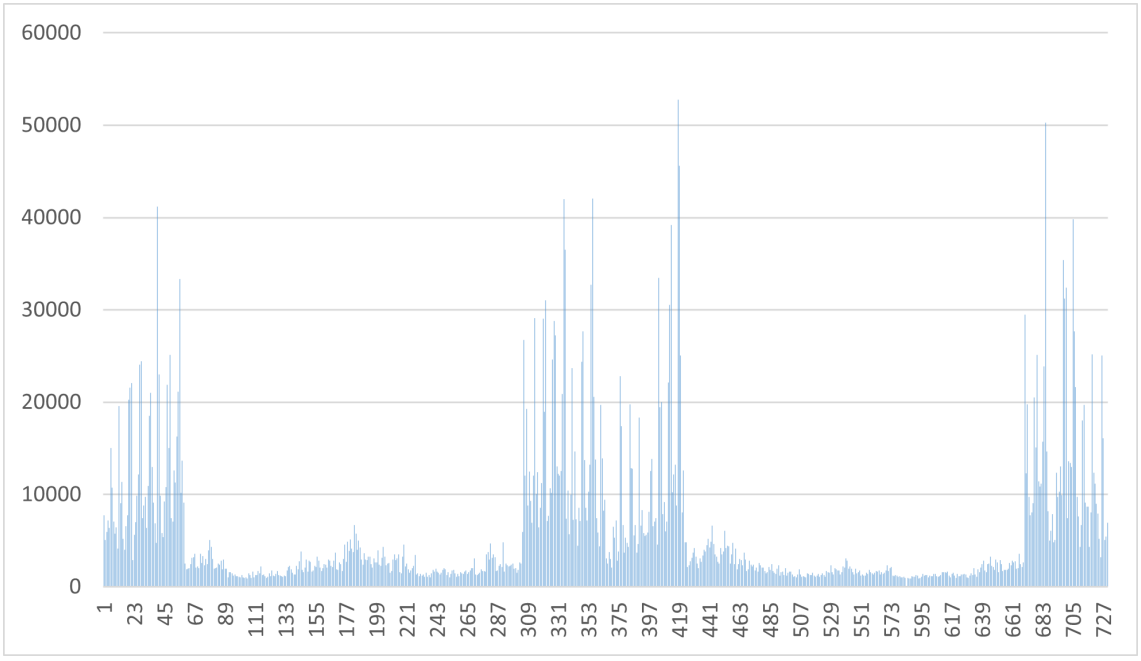

Figura 7.8 Número de iteraciones FISTA.

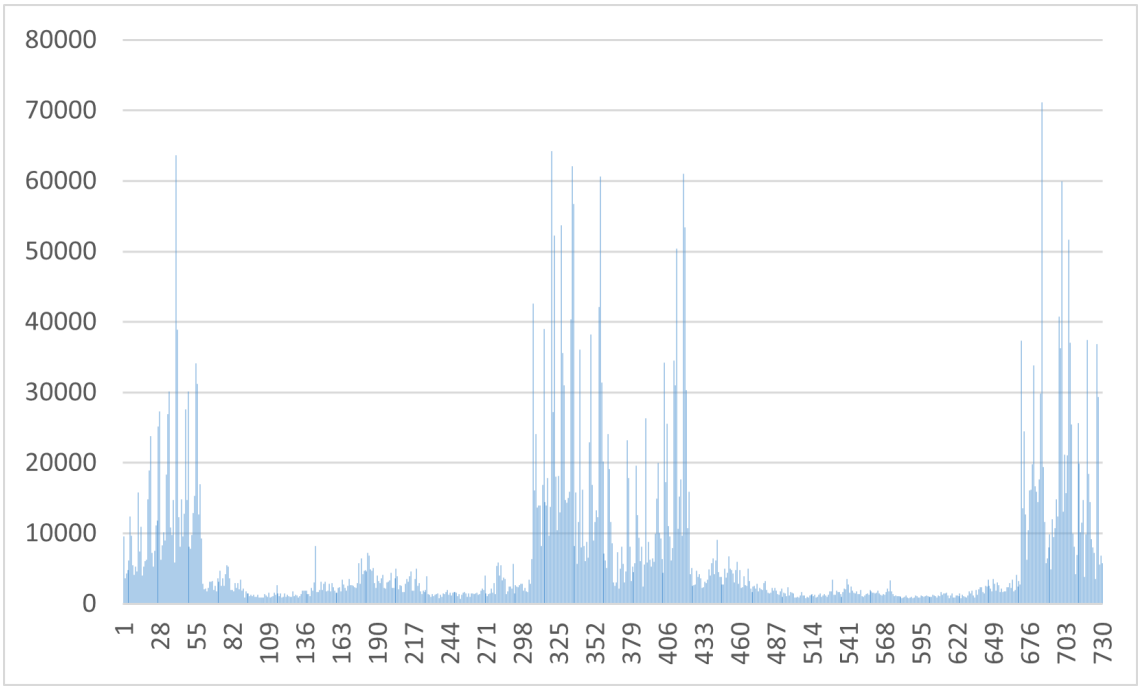

Figura 7.9 Número de iteraciones ISTA con reducción cálculo de signos simple.

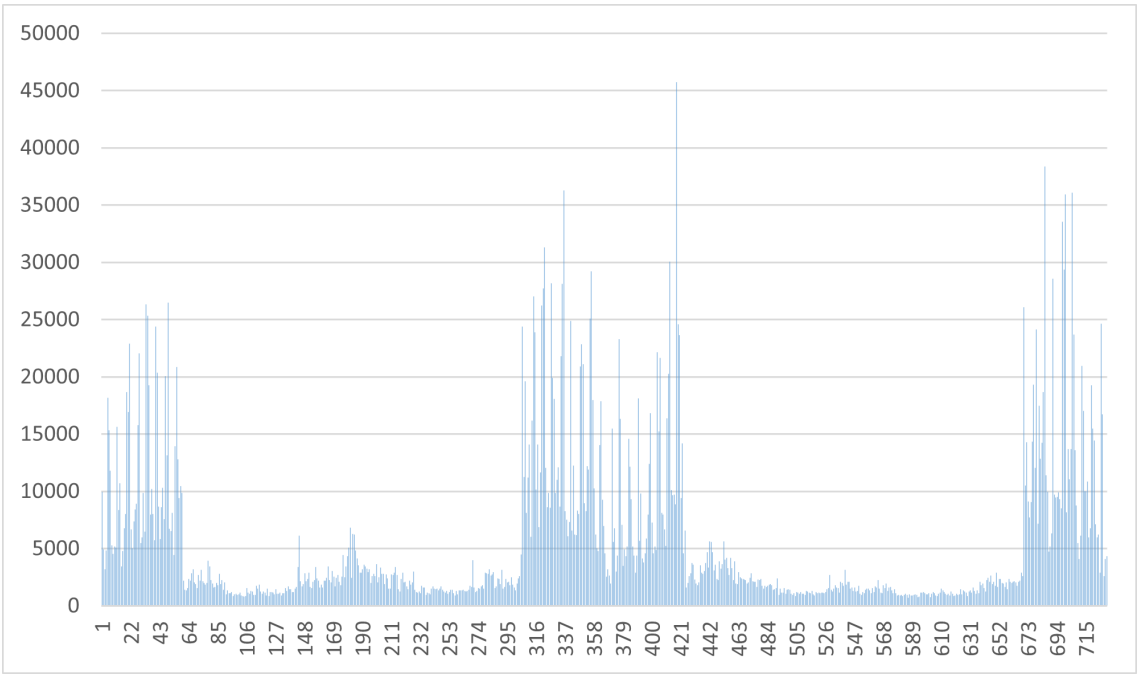

Figura 7.10 Número de iteraciones FISTA con reducción cálculo de signos simple.

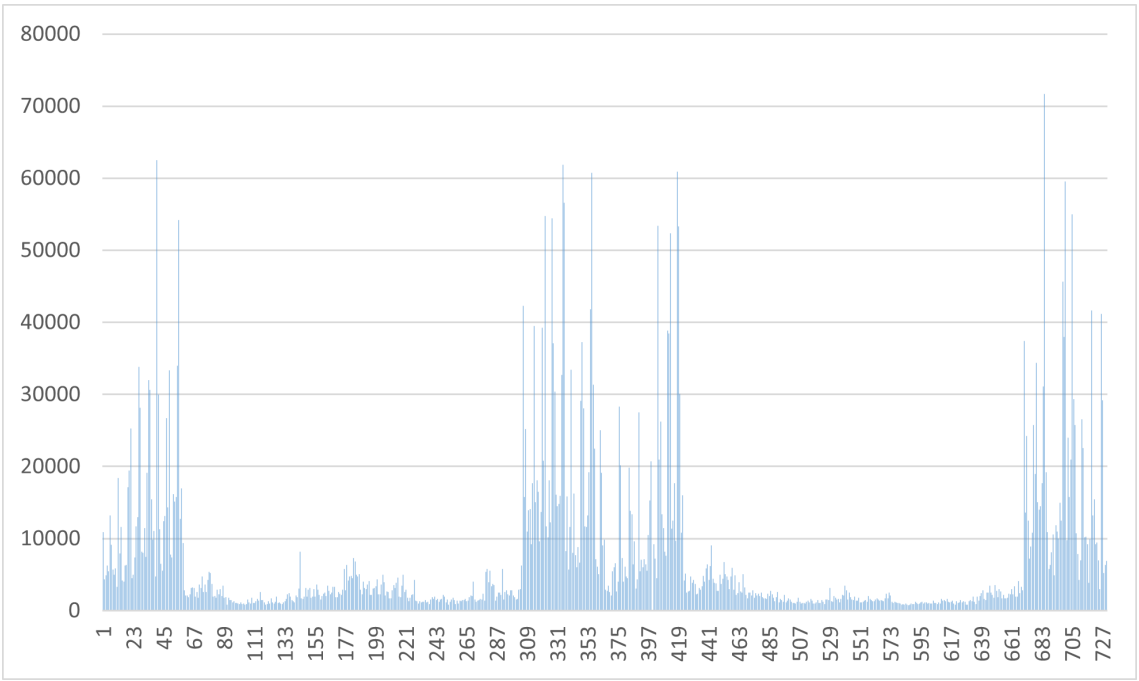

Figura 7.11 Número de iteraciones ISTA con reducción cálculo de signos en función de las iteraciones.

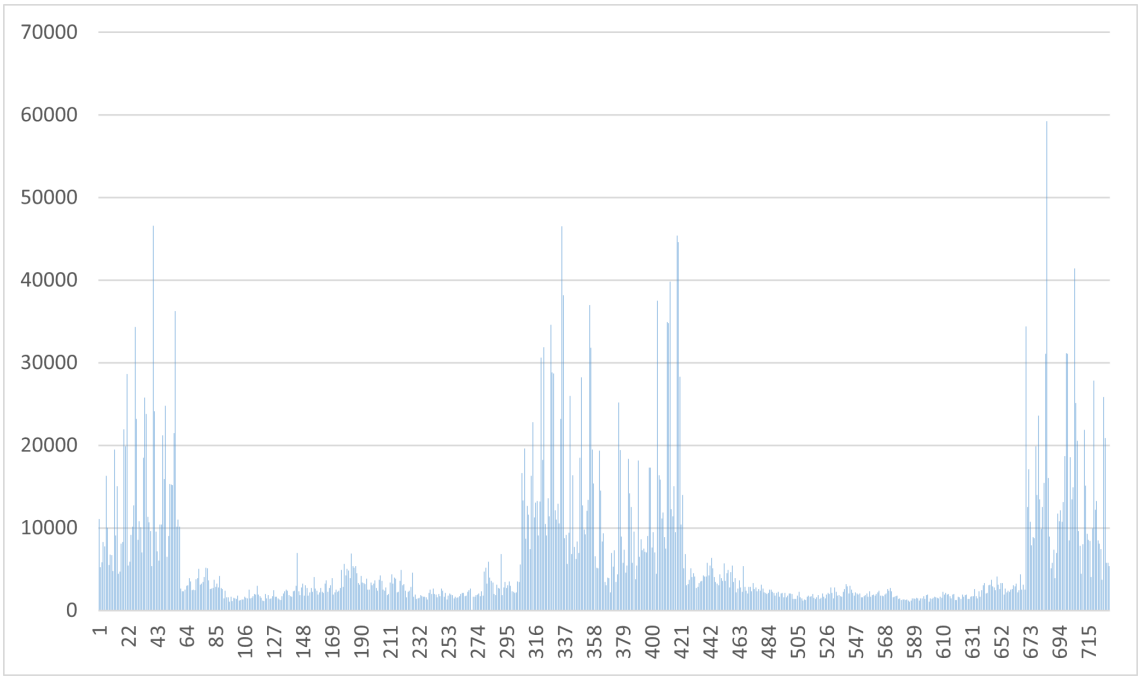

Figura 7.12 Número de iteraciones FISTA con reducción cálculo de signos en función de las iteraciones.

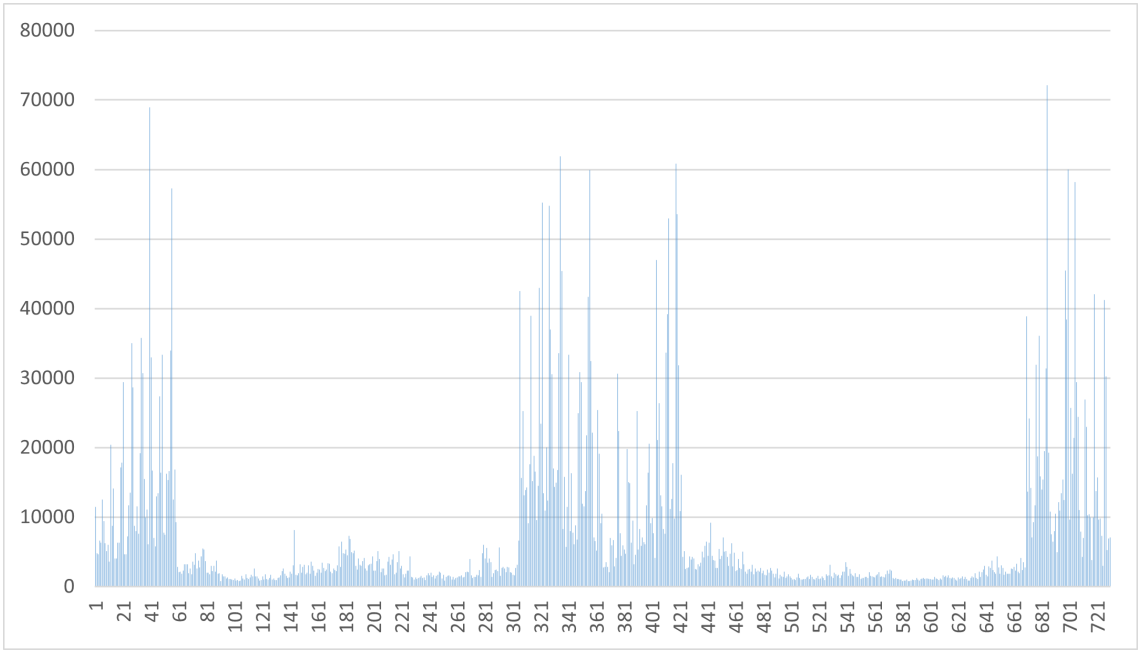

Figura 7.13 Número de iteraciones ISTA con reducción cálculo de signos en función del gradiente.

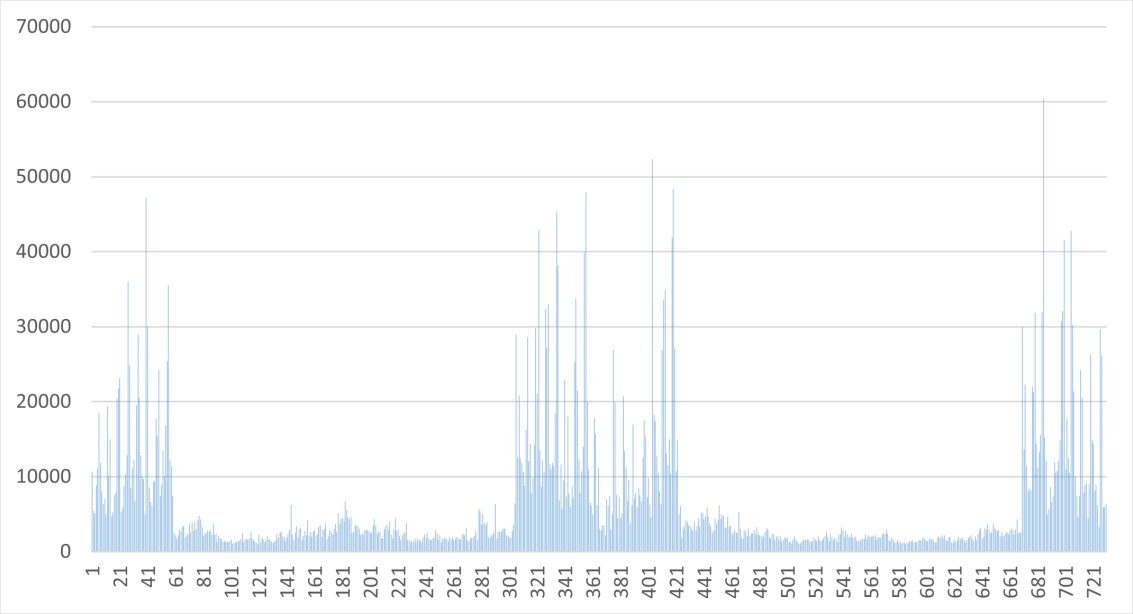

Figura 7.14 Número de iteraciones FISTA con reducción cálculo de signos en función del gradiente.

En Figura 7.7, Figura 7.8, Figura 7.9, Figura 7.10, Figura 7.11, Figura 7.12, Figura 7.13, Figura 7.14, se muestra el número de iteraciones que se ha realizado de cada algoritmo para cada predicción. Se aprecia un claro patrón común en el que son las predicciones centrales y de los extremos de las gráficas las que tienen mayor número de iteraciones, esto corresponde aproximadamente con los meses de enero, febrero, noviembre y diciembre, los meses más fríos.

Esto puede indicar que el ajuste de los parámetros no es adecuado para estos meses y que por tanto hay bastante margen de mejora en cuanto al ajuste del modelo, y una dirección clara para alcanzar dicha mejora.

Con esta información se pueden comparar los algoritmos, y determinar con cual de ellos se requieren menos iteraciones, para ello se calcula el número de iteraciones que requiere de media cada algoritmo para resolver el problema, en este documento se evalúan 8 algoritmos, que corresponden con los algoritmos ISTA y FISTA, y 3 variantes de estos.

Los resultados obtenidos son los siguientes:

- Algoritmo ISTA: 7084.127397
- Algoritmo FISTA: 5773.279452
- Algoritmo ISTA reducción signos simple: 6955.758904
- Algoritmo FISTA reducción signos simple: 5332.236986
- Algoritmo ISTA reducción signos iteraciones: 7082.553425
- Algoritmo FISTA reducción signos iteraciones: 6311.023288
- Algoritmo ISTA reducción signos gradiente: 7220.131507
- Algoritmo FISTA reducción signos gradiente: 6438.09863

Se observa que para cada variante hay una clara reducción en el número de iteraciones en los algoritmos FISTA, respecto a los ISTA, esto concuerda con documentos previos como [7], sin embargo, no se observa de forma clara una reducción del número de iteraciones entre las distintas variantes, esto es de esperar ya que no buscan reducir el número de iteraciones, si no reducir el número de operaciones por iteración.

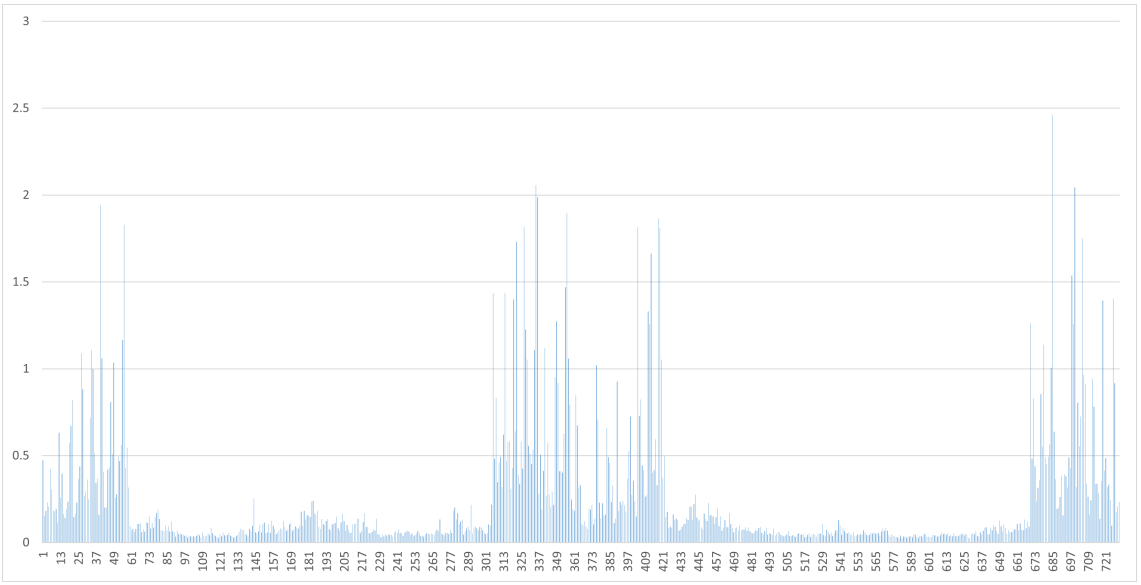

Figura 7.15 Tiempos de ejecución ISTA.

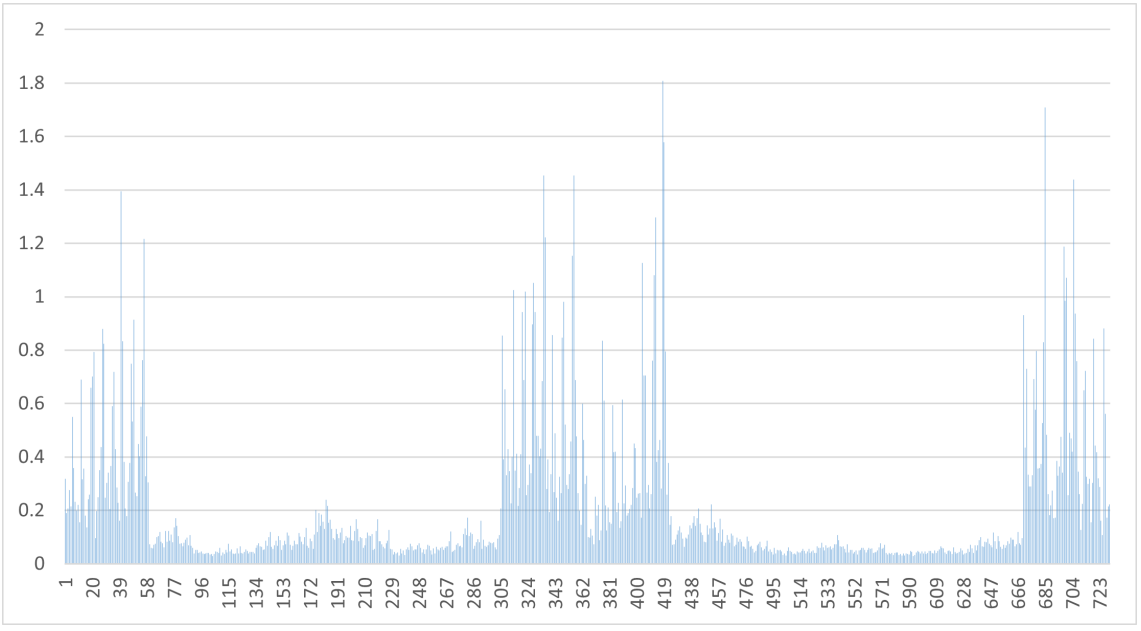

Figura 7.16 Tiempos de ejecución FISTA.

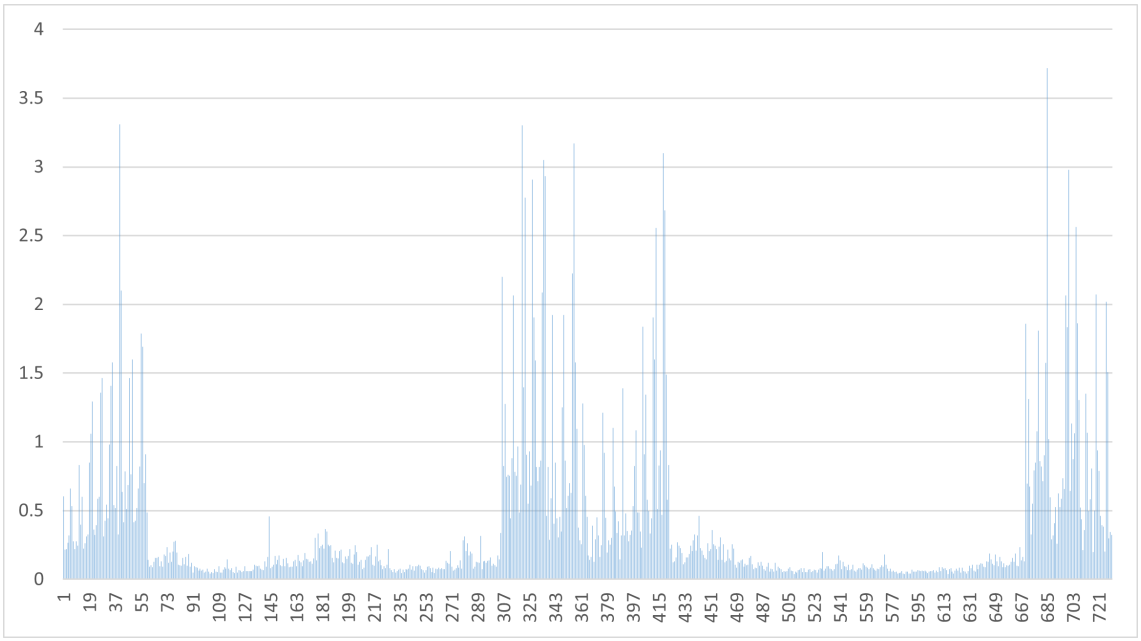

Figura 7.17 Tiempos de ejecución ISTA con reducción cálculo de signos simple.

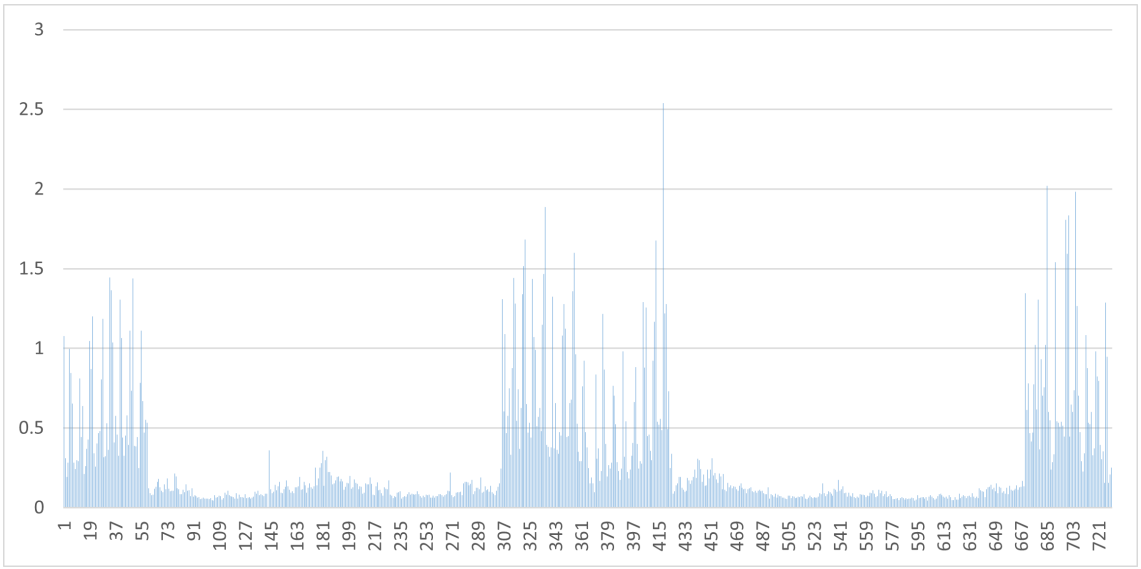

Figura 7.18 Tiempos de ejecución FISTA con reducción cálculo de signos simple.

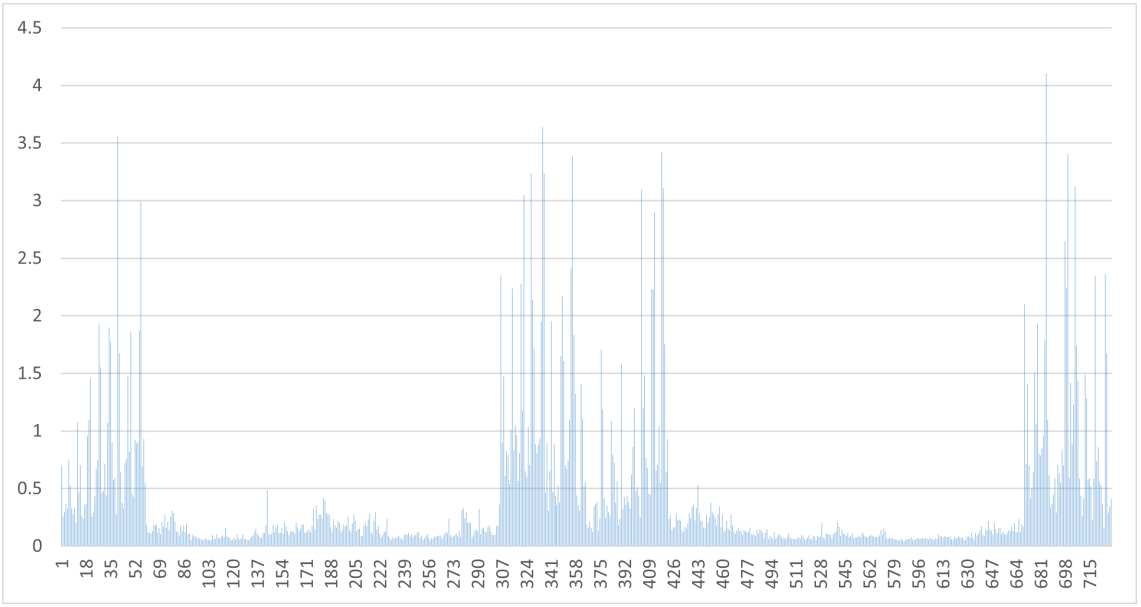

Figura 7.19 Tiempos de ejecución ISTA con reducción cálculo de signos en función de las iteraciones.

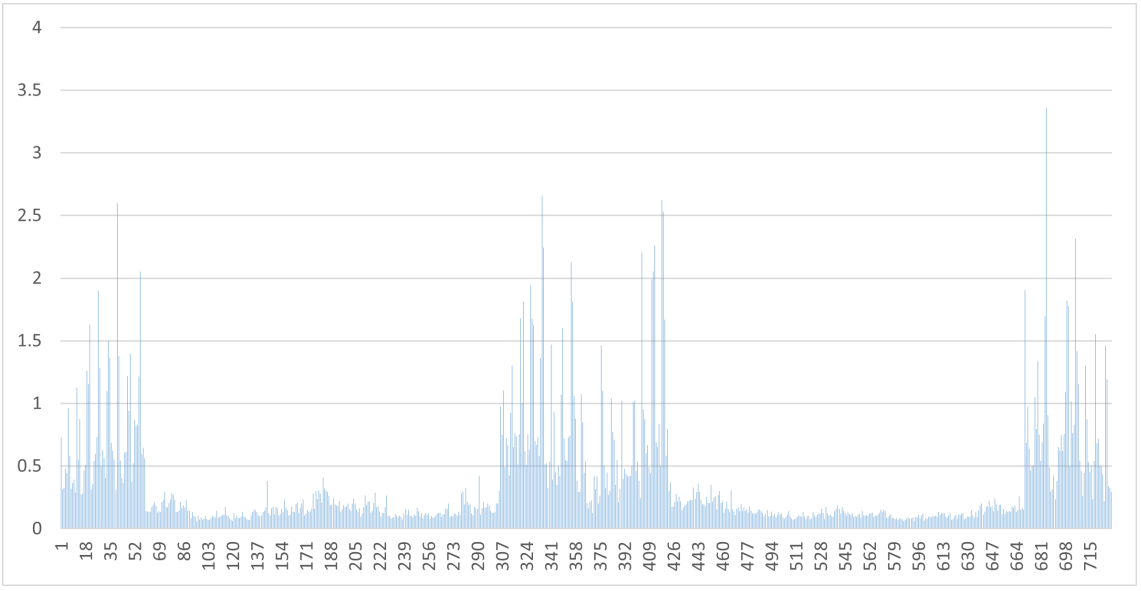

Figura 7.20 Tiempos de ejecución FISTA con reducción cálculo de signos en función de las iteraciones.

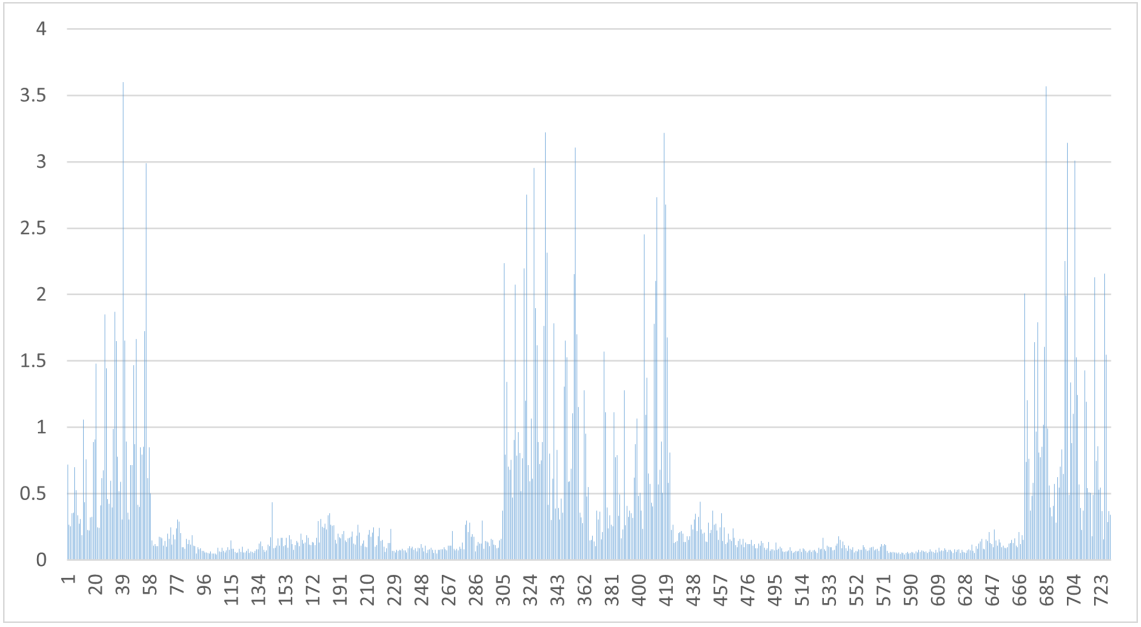

Figura 7.21 Tiempos de ejecución ISTA con reducción cálculo de signos en función del gradiente.

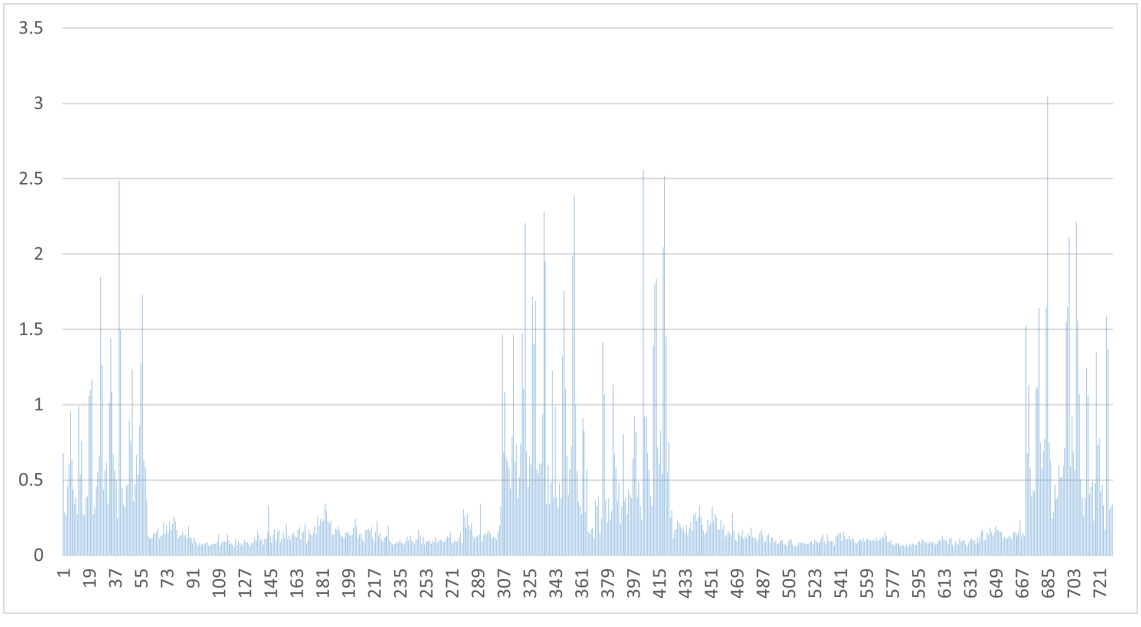

Figura 7.22 Tiempos de ejecución FISTA con reducción cálculo de signos en función del gradiente.

En Figura 7.15, Figura 7.16, Figura 7.17, Figura 7.18, Figura 7.19, Figura 7.20, Figura 7.21, Figura 7.22, se muestran los tiempos de ejecución de cada predicción de los distintos algoritmos, de nuevo se observa un patrón de peor ajuste en los meses de enero, febrero, noviembre y diciembre.

De estos datos podemos sacar conclusiones sobre la velocidad de ejecución de los algoritmos, para ello se calcula la media de los tiempos de ejecución de las distintas variables, Es importante hacer la media de todas las predicciones y no sacar conclusiones de una sola muestra ya que hay variables externas al propio algoritmo que pueden afectar al rendimiento.

Los resultados obtenidos son los siguientes:

- Algoritmo ISTA: 0.237573875 s
- Algoritmo FISTA: 0.199921404 s
- Algoritmo ISTA reducción signos simple: 0.365944914 s
- Algoritmo FISTA reducción signos simple: 0.291423956 s
- Algoritmo ISTA reducción signos iteraciones: 0.407693772 s
- Algoritmo FISTA reducción signos iteraciones: 0.361766923 s
- Algoritmo ISTA reducción signos gradiente: 0.377960893 s
- Algoritmo FISTA reducción signos gradiente: 0.335548475 s

Como se puede observar, las variantes FISTA tienen una clara ventaja respecto a ISTA en cuanto a tiempo de ejecución, esto es lo esperado ya que se ha documentado en multitud de ocasiones que FISTA es una variante de ISTA más rápida tanto en la teoría como en la práctica.

Sin embargo, no se obtienen mejores resultados con las distintas variaciones de ISTA y FISTA, pese a que en la teoría se puede llegar a reducir de forma significativa el número de operaciones totales del algoritmo.

Debido a que las variantes estudiadas en la teoría son más efectivas en cuanto mayor sea la dimensión del problema, se han realizado una serie de pruebas con datos aleatorios, por una parte se fija el tamaño de las muestras a 50 elementos y se varia el número de muestras N, además para un número fijo de 1000 muestras, se varia el tamaño de las muestras.

Tabla 7.1 Número de iteraciones en función de N parte 1.

N	ISTA	FISTA	ISTA SIMPLE	FISTA SIMPLE
100	23061.585	4094.495	28348.745	4531.915
500	61054.435	14603.99	61055.46	15032.595
1000	241498.82	30494.28	241497.04	30865.75
1500	314598.16	37703.03	314596.78	37635.62
2500	249251.9	51070.7	249252.5	51877

Tabla 7.2 Número de iteraciones en función de N parte 2.

N	ISTA ITER	FISTA ITER	ISTA GRAD	FISTA GRAD
100	28345.54	4444.355	23186.91	4260.555
500	61054.055	14977.53	61381.12	15049.115
1000	241495.76	30327.67	242936.1	30710.78
1500	314598.65	35515.94	316504.21	37305.68
2500	249252.72	50705.96	251926.96	52110.18

Tabla 7.3 Tiempos de ejecución en función de N parte 1.

N	ISTA	FISTA	ISTA SIMPLE	FISTA SIMPLE
100	0.332668304	0.076755081	0.704490202	0.140441747
500	1.932385841	0.422385511	2.721119246	0.674535195
1000	9.039216375	1.144041991	14.00835295	1.796852122
1500	14.19179337	1.709338761	21.74874736	2.623211643
2500	15.83870899	3.279940808	23.69127847	4.979878642

Tabla 7.4 Tiempos de ejecución en función de N parte 2.

N	ISTA ITER	FISTA ITER	ISTA GRADE	FISTA GRAD
100	0.897066767	0.176209344	0.709457329	0.153487515
500	2.966782363	0.738790127	2.678169937	0.662266605
1000	15.42550464	1.955028386	13.91248464	1.760037189
1500	24.16990565	2.755925446	21.46986657	2.566162717
2500	26.76307045	5.528397054	24.03411773	4.998566516

Tabla 7.5 Número de iteraciones en función del tamaño de d parte 1.

d	ISTA	FISTA	ISTA SIMPLE	FISTA SIMPLE
50	153492.3265	28390.95918	153493.4694	28390.53061
100	812112.74	50814.1	812123.76	51673.26
200	488191.1	49256.08	488187.66	49050.96
400	259195.42	50139.38	259196.16	51724.56

Tabla 7.6 Número de iteraciones en función del tamaño de d parte 2.

d	ISTA ITER	FISTA ITER	ISTA GRAD	FISTA GRAD
50	153493.0204	28192.55102	152139.4082	28878.2449
100	812117.3	50776.06	823289.74	42080.76
200	488192.24	49670.58	494332.18	52622.24
400	259200.14	50868.1	263222.72	50872.4

Tabla 7.7 Tiempos de ejecución en función del tamaño de d parte 1.

d	ISTA	FISTA	ISTA SIMPLE	FISTA SIMPLE
50	5.68168842	1.034845863	8.645089841	1.709179416
100	37.14192291	2.323342388	44.785	3.719065572
200	33.75933225	3.425045292	46.9346733	4.592391714
400	29.55275371	5.73433905	48.499941	9.400488522

Tabla 7.8 Tiempos de ejecución en función del tamaño de d parte 2.

d	ISTA ITER	FISTA ITER	ISTA GRAD	FISTA GRAD
50	10.04829695	1.513970663	8.457846145	1.626927043
100	64.40843837	4.03508584	58.47443525	3.036183134
200	65.83761212	6.723956992	45.60762621	4.840163582
400	77.1378413	14.99309979	46.28012292	8.771577546

Con la aplicación de los algoritmos a problemas de dimensiones diferentes, se observa que siguen sin ser una mejora respecto a los algoritmos tradicionales ISTA y FISTA, en cuanto a número de iteraciones o tiempo de ejecución, pero se observa un aumento notable de la ventaja de FISTA frente a ISTA.

Sin embargo, si se aprecia que al aumentar el valor de N, el número de iteraciones que realiza el algoritmo que fija los signos en función del gradiente, sobrepasa al número de iteraciones que realiza el algoritmo que fija los signos en función del número de iteraciones. A pesar de esto los valores de tiempos de ejecución se mantienen por debajo, lo cual indica que realiza iteraciones más rápidas.

Por el contrario, al aumentar el tamaño de las muestras, no se observa ningún patrón claro en cuanto al número de iteraciones, únicamente se aprecia que para un tamaño de 50 elementos el algoritmo basado en iteraciones es más rápido y realiza menos iteraciones que el simple y el basado en gradiente , sin , para tamaños mayores se ve superado por ambos, con una ligera ventaja para el algoritmo basado en el gradiente.

7.3 Conclusiones y posibles mejoras

Por último se recopilaran las conclusiones obtenidas durante el desarrollo de este documento.

En primer lugar, el Kriging es un método de predicción que permite obtener muy buenos resultados, con una aplicación sencilla y que permite incluso a individuos no expertos realizar predicciones con una gran utilidad real.

Es aplicable a una gran variedad de casos, entre los que se puede destacar la demanda de energía eléctrica para la cual se ha demostrado que se obtiene una gran exactitud, con una aplicación sencilla.

A la hora de resolver el problema dado, se demuestra la efectividad de los algoritmos ISTA y FISTA, y de sus variantes recopiladas en este documento, además se confirma la mayor velocidad de convergencia de FISTA frente a ISTA.

En cuanto a las variantes de ISTA y FISTA propuestas, no se puede afirmar que en la práctica aceleren la resolución del problema que se ha tratado en este documento.

Como posibles mejoras se plantean numerosas opciones, principalmente se debe probar con mayores dimensiones del problema ya que como se explica en el Capitulo 4, la mejora de las variantes propuestas es en teoría mayor cuanto mayor sea la dimensión del problema.

Por otro lado, es necesario seguir desarrollando propuestas de mejora basadas en la reducción de operaciones mediante el método propuesto, ya que hay infinidad de heurísticas diferentes que es posible que sean más adecuadas.

En definitiva, el Kriging es una herramienta muy potente, que puede verse beneficiada enormemente de mejoras en los algoritmos empleados para implementarla, sin embargo,

tras explorar la opción de fijar signos para acelerar los algoritmos tradicionales, no se han obtenido resultados positivos. A pesar de esto, es una opción que tiene potencial y que se debe seguir explorando.

8 Codigo

Código 8.1 Aplicación de Quadprog en Matlab.

```
%Queremos optimizar una funcion de disimilitud que nos
   indique como de
%diferente es un vector d a los vectores d_i de un set de
    datos D
%ponderados con el vector positivo de w
%Resolvemos el Problema de optimizacion dado mediante
   quadprog
%para ello necesito H,f,A,l,Aeq,beq
%usare N=400 como ejemplo del numero de samples que
   tenemos y cada sample
%tendra n=50 elementos.
d = randn(50,1); % vector d que buscamos comparar con los
    vectores del set de datos D

D = randn(50,400);

w = abs(randn(1,400))+1;% vector de pesos positivo, he añ
   adido el +1 para evitar un peso 0

gamma = abs(randn(1,400));% vector no negativo

H = 2*(w.*eye(400));

O=zeros(400);

Hqp = [H O;O O]; %H

q = [zeros(1,400),gamma]';% f

A = [D zeros(50,400); ones(1,400) zeros(1,400)];% Aeq
```

```
b = [d;1];% beq

C = [eye(400) -eye(400); -eye(400) -eye(400)]; %A

l = zeros(800,1);%b

sol = quadprog(Hqp,q,C,l,A,b);
```

Código 8.2 Aplicación de ISTA en Matlab.

```
function [mu,lambda,k] = ISTABUENO(H,R,b,eps,w,gamma)

[m,n]=size(R);

mu = zeros(m,1);

k=0;

error=1;

ter = inv(R*inv(H)*R');

Z= zeros(1,n);

verr=[];

vk=[];

while error > eps

   k=k+1;

   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;

   Z(I)=0;

   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));
```

```
   lambda= Z;

   t = R*lambda';

   g = t-b;

   mu = mu+ter*g;

   error = norm(g);

   verr(k)=error;

   vk(k)=k;
end
semilogy(vk,verr)
end
```

Código 8.3 Aplicación de FISTA en Matlab.

```
function [mu,lambda,k] = FISTA(H,R,b,eps,w,gamma)
[m,n]=size(R);

Abeta= zeros(m,1);

beta= zeros(m,1);

k=0;

Ati=1;

error=1;

ter = inv(R*inv(H)*R');

verr=[];

vk=[];

while error > eps

   k=k+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;
```

```
   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;

   Z(I)=0;

   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda= Z;

   t = R*lambda';

   g = t-b;

   Ati=ti;

   Abeta=beta;

   beta = mu+ter*g;

   error = norm(g);

   verr(k)=error;

   vk(k)=k;
end
%plot(vk,verr)
semilogy(vk,verr)
end
```

Código 8.4 Descomposición en ISTA.

```
function [mu,lambda,k] = ISTADECO(H,R,b,eps,w,gamma)
[m,n]=size(R);

mu = zeros(m,1);

k=0;
```

```
error=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

verr=[];

vk=[];

while error > eps

   k=k+1;

   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;

   Z(I)=0;

   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda= Z;

   t = R*lambda';

   g = t-b;

   y=g+Q*mu;

   mu = dQ\y;

   error = norm(g);

   verr(k)=error;

   vk(k)=k;

end
```

```
semilogy(vk,verr)
end
```

Código 8.5 Descomposición en FISTA.

```
function [mu,lambda,k] = FISTADECO(H,R,b,eps,w,gamma)
[m,n]=size(R);

Abeta= zeros(m,1);

beta= zeros(m,1);

k=0;

Ati=1;

error=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

verr=[];

vk=[];

while error > eps

   k=k+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;

   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;

   Z(I)=0;
```

```
   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda= Z;

   t = R*lambda';

   g = t-b;

   Ati=ti;

   Abeta=beta;

   y=g+Q*mu;

   beta = dQ\y;

   error = norm(g);

   verr(k)=error;

   vk(k)=k;
end
semilogy(vk,verr)
end
```

Código 8.6 ISTA ACTUALIZACION simple.

```
function [mu,lambda,k,error] = ISTASIGNOS1(H,R,b,eps,w,
   gamma)

[m,n]=size(R);

mu = zeros(m,1);

k=0;

error=1;

errori=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');
```

```
Z= zeros(1,n);

gammai=zeros(1,n);

while errori > eps

   k=k+1;

   c = mu'*R;

   I=c>gammai;

   Z(I)= (gammai(I)-c(I))./(2*w(I));

   I=abs(c) <= gammai;

   Z(I)=0;

   I=c < -gammai;

   Z(I) = (-gammai(I)-c(I))./(2*w(I));

   lambda1= Z;

   t = R*lambda1';

   g = t-b;

   y=g+Q*mu;

   mu = dQ\y;

   errori = norm(g);

end

while k<10

   k=k+1;

   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;
```

```
   Z(I)=0;

   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda2= Z;

   t = R*lambda2';

   g = t-b;

   y=g+Q*mu;

   mu = dQ\y;

   error = norm(g);

end

vector1pos=lambda1>0; % cojo los positivos de la primera
   lambda

vector2pos=lambda2>0; % cojo los positivos de la segunda
   lambda

sumapos=vector1pos+vector2pos; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido positivas
    en ambas lambdas

posmanten=sumapos>1; %saco vector con los que signos
   positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la primera
   lambda

vector2neg=lambda2<0; % cojo los negativos de la segunda
   lambda

sumaneg=vector1neg+vector2neg; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido negativos
    en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que signos
   negativos que se han mantenido
```

```
cambionulinv=negmanten+posmanten; % vector donde los 0
   indican los signos que han cambiado o valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);

s3(TF3)=0;

t2=R1*s3;

while error > eps
```

```
k=k+1;

c = mu'*R;

I=c>gamma;

I=I.*cambionul;

L=logical(I);

Z(L)= (gamma(L)-c(L))./(2*w(L));

I=abs(c) <= gamma;

I=I.*cambionul;

L=logical(I);

Z(L)=0;

I=c < -gamma;

I=I.*cambionul;

L=logical(I);

Z(L) = (-gamma(L)-c(L))./(2*w(L));

Z(posmanten) = -((gamma(posmanten)+c(posmanten))./(2*w
   (posmanten)));

Z(negmanten) = -((-gamma(negmanten)+c(negmanten))./(2*
   w(negmanten)));

lambda= Z;

m=lambda.*cambionul;

t=R2*m';

te2=t2*mu;

g = -b+t-t1-te2;

y=g+Q*mu;

mu = dQ\y;
```

```
   error = norm(g);

end
% semilogy(vk,verr)
end
```

Código 8.7 FISTA ACTUALIZACION simple.

```
function [mu,lambda,k,error] = FISTASIGNOS1(H,R,b,eps,w,
   gamma)

[m,n]=size(R);

Abeta= zeros(m,1);

beta= zeros(m,1);

k=0;

Ati=1;

error=1;

errori=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

gammai=zeros(1,n);

while errori > eps

   k=k+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;

   c = mu'*R;

   I=c>gammai;
```

```
    Z(I)= (gammai(I)-c(I))./(2*w(I));

    I=abs(c) <= gammai;

    Z(I)=0;

    I=c < -gammai;

    Z(I) = (-gammai(I)-c(I))./(2*w(I));

    lambda1= Z;

    t = R*lambda1';

    g = t-b;

    Ati=ti;

    Abeta=beta;

    y=g+Q*mu;

    beta = dQ\y;

    errori = norm(g);

end

while k<10

    k=k+1;

    ti=0.5*(1+sqrt(1+4*(Ati)^2));

    mu=beta+(Ati-1)*(beta-Abeta)/ti;

    c = mu'*R;

    I=c>gamma;

    Z(I)= (gamma(I)-c(I))./(2*w(I));

    I=abs(c) <= gamma;

    Z(I)=0;

    I=c < -gamma;
```

```
   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda2= Z;

   t = R*lambda2';

   g = t-b;

   Ati=ti;

   Abeta=beta;

   y=g+Q*mu;

   beta = dQ\y;

   error = norm(g);

end

vector1pos=lambda1>0; % cojo los positivos de la primera
   lambda

vector2pos=lambda2>0; % cojo los positivos de la segunda
   lambda

sumapos=vector1pos+vector2pos; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido positivas
    en ambas lambdas

posmanten=sumapos>1; %saco vector con los que signos
   positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la primera
   lambda

vector2neg=lambda2<0; % cojo los negativos de la segunda
   lambda

sumaneg=vector1neg+vector2neg; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido negativos
    en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que signos
   negativos que se han mantenido
```

```
cambionulinv=negmanten+posmanten; % vector donde los 0
   indican los signos que han cambiado o valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);

s3(TF3)=0;

t2=R1*s3;

while error > eps
```

```
k=k+1;

ti=0.5*(1+sqrt(1+4*(Ati)^2));

mu=beta+(Ati-1)*(beta-Abeta)/ti;

c = mu'*R;

I=c>gamma;

I=I.*cambionul;

L=logical(I);

Z(L)= (gamma(L)-c(L))./(2*w(L));

I=abs(c) <= gamma;

I=I.*cambionul;

L=logical(I);

Z(L)=0;

I=c < -gamma;

I=I.*cambionul;

L=logical(I);

Z(L) = (-gamma(L)-c(L))./(2*w(L));

Z(posmanten) = -((gamma(posmanten)+c(posmanten))./(2*w
   (posmanten)));

Z(negmanten) = -((-gamma(negmanten)+c(negmanten))./(2*
   w(negmanten)));

lambda= Z;

m=lambda.*cambionul;

t=R2*m';

te2=t2*mu;

g = -b+t-t1-te2;
```

```
   Ati=ti;

   Abeta=beta;

   y=g+Q*mu;

   beta = dQ\y;

   error = norm(g);

end
% semilogy(vk,verr)
end
```

Código 8.8 ISTA ACTUALIZACION ITERACIONES.

```
function [mu,lambda,k,error] = ISTAITER(H,R,b,eps,w,gamma
   )

[m,n]=size(R);

mu = zeros(m,1);

k=0;

error=1;

errori=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

verr=[];

vk=[];

gammai=zeros(1,n);

while errori > eps

   k=k+1;
```

```
   c = mu'*R;

   I=c>gammai;

   Z(I)= (gammai(I)-c(I))./(2*w(I));

   I=abs(c) <= gammai;

   Z(I)=0;

   I=c < -gammai;

   Z(I) = (-gammai(I)-c(I))./(2*w(I));

   lambda1= Z;

   t = R*lambda1';

   g = t-b;

   y=g+Q*mu;

   mu = dQ\y;

   errori = norm(g);

   verr(k)=errori;

   vk(k)=k;
end

while k<10

   k=k+1;

   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;

   Z(I)=0;

   I=c < -gamma;
```

```
   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda2= Z;

   t = R*lambda2';

   gi = t-b;

   y=gi+Q*mu;

   mu = dQ\y;

   error = norm(gi);

   verr(k)=error;

   vk(k)=k;
end

vector1pos=lambda1>0; % cojo los positivos de la primera
   lambda

vector2pos=lambda2>0; % cojo los positivos de la segunda
   lambda

sumapos=vector1pos+vector2pos; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido positivas
    en ambas lambdas

posmanten=sumapos>1; %saco vector con los que signos
   positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la primera
   lambda

vector2neg=lambda2<0; % cojo los negativos de la segunda
   lambda

sumaneg=vector1neg+vector2neg; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido negativos
    en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que signos
   negativos que se han mantenido

cambionulinv=negmanten+posmanten; % vector donde los 0
   indican los signos que han cambiado o valores nulos
```

```
cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);

s3(TF3)=0;

t2=R1*s3;

k2=0;

while error > eps
```

```
k=k+1;

k2=k2+1;

c = mu'*R;

I=c>gamma;

I=I.*cambionul;

L=logical(I);

Z(L)= (gamma(L)-c(L))./(2*w(L));

I=abs(c) <= gamma;

I=I.*cambionul;

L=logical(I);

Z(L)=0;

I=c < -gamma;

I=I.*cambionul;

L=logical(I);

Z(L) = (-gamma(L)-c(L))./(2*w(L));

Z(posmanten) = -((gamma(posmanten)+c(posmanten))./(2*w
   (posmanten)));

Z(negmanten) = -((-gamma(negmanten)+c(negmanten))./(2*
   w(negmanten)));

lambda= Z;

m=lambda.*cambionul;

t=R2*m';

te2=t2*mu;

g = -b+t-t1-te2;

y=g+Q*mu;
```

```
mu = dQ\y;

error = norm(g);

verr(k)=error;

vk(k)=k;

ni=k2/50;

if k2 == 1

   lambda1=lambda;

end

if isreal(ni) && (ni > 0) && (rem(ni,1) == 0)

   lambda2=lambda;

   vector1pos=lambda1>0; % cojo los positivos de la
      primera lambda

   vector2pos=lambda2>0; % cojo los positivos de la
      segunda lambda

   sumapos=vector1pos+vector2pos; % los sumo de forma
      que solo sumaran 2 las posiciones que hayan sido
       positivas en ambas lambdas

   posmanten=sumapos>1; %saco vector con los que
      signos positivos que se han mantenido

   vector1neg=lambda1<0; % cojo los negativos de la
      primera lambda

   vector2neg=lambda2<0; % cojo los negativos de la
      segunda lambda

   sumaneg=vector1neg+vector2neg; % los sumo de forma
      que solo sumaran 2 las posiciones que hayan sido
       negativos en ambas lambdas

   negmanten=sumaneg>1; %saco vector con los que
      signos negativos que se han mantenido
```

```
cambionulinv=negmanten+posmanten; % vector donde
   los 0 indican los signos que han cambiado o
   valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);

s3(TF3)=0;

t2=R1*s3;

lambda1=lambda2;
```

```
   end


end
semilogy(vk,verr)
end
```

Código 8.9 ISTA ACTUALIZACION ERROR.

```
function [mu,lambda,k,error] = ISTAERROR(H,R,b,eps,w,
   gamma)

[m,n]=size(R);

mu = zeros(m,1);

k=0;

error=1;

errori=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

verr=[];

vk=[];

gammai=zeros(1,n);

while errori > eps

   k=k+1;

   c = mu'*R;

   I=c>gammai;

   Z(I)= (gammai(I)-c(I))./(2*w(I));

   I=abs(c) <= gammai;
```

```
   Z(I)=0;

   I=c < -gammai;

   Z(I) = (-gammai(I)-c(I))./(2*w(I));

   lambda1= Z;

   t = R*lambda1';

   g = t-b;

   y=g+Q*mu;

   mu = dQ\y;

   errori = norm(g);

   verr(k)=errori;

   vk(k)=k;
end

while k<10

   k=k+1;

   c = mu'*R;

   I=c>gamma;

   Z(I)= (gamma(I)-c(I))./(2*w(I));

   I=abs(c) <= gamma;

   Z(I)=0;

   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda2= Z;

   t = R*lambda2';

   gi = t-b;
```

```
   y=gi+Q*mu;

   mu = dQ\y;

   error = norm(gi);

   verr(k)=error;

   vk(k)=k;
end

vector1pos=lambda1>0; % cojo los positivos de la primera
   lambda

vector2pos=lambda2>0; % cojo los positivos de la segunda
   lambda

sumapos=vector1pos+vector2pos; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido positivas
    en ambas lambdas

posmanten=sumapos>1; %saco vector con los que signos
   positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la primera
   lambda

vector2neg=lambda2<0; % cojo los negativos de la segunda
   lambda

sumaneg=vector1neg+vector2neg; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido negativos
    en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que signos
   negativos que se han mantenido

cambionulinv=negmanten+posmanten; % vector donde los 0
   indican los signos que han cambiado o valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);
```

```
d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);

s3(TF3)=0;

t2=R1*s3;

k2=0;

while error > eps

   k=k+1;

   k2=k2+1;

   c = mu'*R;

   I=c>gamma;
```

```
I=I.*cambionul;

L=logical(I);

Z(L)= (gamma(L)-c(L))./(2*w(L));

I=abs(c) <= gamma;

I=I.*cambionul;

L=logical(I);

Z(L)=0;

I=c < -gamma;

I=I.*cambionul;

L=logical(I);

Z(L) = (-gamma(L)-c(L))./(2*w(L));

Z(posmanten) = -((gamma(posmanten)+c(posmanten))./(2*w
   (posmanten)));

Z(negmanten) = -((-gamma(negmanten)+c(negmanten))./(2*
   w(negmanten)));

lambda= Z;

m=lambda.*cambionul;

t=R2*m';

te2=t2*mu;

g = -b+t-t1-te2;

y=g+Q*mu;

mu = dQ\y;

error = norm(g);

verr(k)=error;

vk(k)=k;
```

```
if k2 == 1

   lambda1=lambda;

   errorcomp=error;

end

if error < errorcomp*0.1

   lambda2=lambda;

   vector1pos=lambda1>0; % cojo los positivos de la
      primera lambda

   vector2pos=lambda2>0; % cojo los positivos de la
      segunda lambda

   sumapos=vector1pos+vector2pos; % los sumo de forma
      que solo sumaran 2 las posiciones que hayan sido
       positivas en ambas lambdas

   posmanten=sumapos>1; %saco vector con los que
      signos positivos que se han mantenido

   vector1neg=lambda1<0; % cojo los negativos de la
      primera lambda

   vector2neg=lambda2<0; % cojo los negativos de la
      segunda lambda

   sumaneg=vector1neg+vector2neg; % los sumo de forma
      que solo sumaran 2 las posiciones que hayan sido
       negativos en ambas lambdas

   negmanten=sumaneg>1; %saco vector con los que
      signos negativos que se han mantenido

   cambionulinv=negmanten+posmanten; % vector donde
      los 0 indican los signos que han cambiado o
      valores nulos

   cambionul=abs(cambionulinv-ones(1,n));

   n1=gamma.*posmanten;
```

```
      n2=-(gamma.*negmanten);

      d1=2*(w.*posmanten);

      d2=2*(w.*negmanten);

      s1=n1./d1;

      s2=n2./d2;

      TF1=isnan(s1);

      TF2=isnan(s2);

      s1(TF1)=0;

      s2(TF2)=0;

      suma=s1+s2;

      R1=R.*cambionulinv;

      R2=R.*cambionul;

      t1=R1*suma';

      n3=R1';

      d3=2*(w.*cambionulinv);

      s3=n3./d3';

      TF3=isnan(s3);

      s3(TF3)=0;

      t2=R1*s3;

      lambda1=lambda2;

      errorcomp=error;
   end


end
semilogy(vk,verr)
end
```

Código 8.10 FISTA ACTUALIZACION ITERACIONES.

```
function [mu,lambda,k,error] = FISTAITER(H,R,b,eps,w,
   gamma)

[m,n]=size(R);

Abeta= zeros(m,1);

beta= zeros(m,1);

k=0;

Ati=1;

error=1;

errori=1;

Q = R*inv(H)*R';

dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

verr=[];

vk=[];

gammai=zeros(1,n);

while errori > eps

   k=k+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;

   c = mu'*R;

   I=c>gammai;

   Z(I)= (gammai(I)-c(I))./(2*w(I));
```

```
    I=abs(c) <= gammai;

    Z(I)=0;

    I=c < -gammai;

    Z(I) = (-gammai(I)-c(I))./(2*w(I));

    lambda1= Z;

    t = R*lambda1';

    g = t-b;

    Ati=ti;

    Abeta=beta;

    y=g+Q*mu;

    beta = dQ\y;

    errori = norm(g);

    verr(k)=errori;

    vk(k)=k;

end

while k<10

    k=k+1;

    ti=0.5*(1+sqrt(1+4*(Ati)^2));

    mu=beta+(Ati-1)*(beta-Abeta)/ti;

    c = mu'*R;

    I=c>gamma;

    Z(I)= (gamma(I)-c(I))./(2*w(I));

    I=abs(c) <= gamma;

    Z(I)=0;
```

```
   I=c < -gamma;

   Z(I) = (-gamma(I)-c(I))./(2*w(I));

   lambda2= Z;

   t = R*lambda2';

   g = t-b;

   Ati=ti;

   Abeta=beta;

   y=g+Q*mu;

   beta = dQ\y;

   error = norm(g);

   verr(k)=error;

   vk(k)=k;
end

vector1pos=lambda1>0; % cojo los positivos de la primera
   lambda

vector2pos=lambda2>0; % cojo los positivos de la segunda
   lambda

sumapos=vector1pos+vector2pos; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido positivas
    en ambas lambdas

posmanten=sumapos>1; %saco vector con los que signos
   positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la primera
   lambda

vector2neg=lambda2<0; % cojo los negativos de la segunda
   lambda
```

```
sumaneg=vector1neg+vector2neg; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido negativos
    en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que signos
   negativos que se han mantenido

cambionulinv=negmanten+posmanten; % vector donde los 0
   indican los signos que han cambiado o valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);
```

```
s3(TF3)=0;

t2=R1*s3;

k2=0;

while error > eps

   k=k+1;

   k2=k2+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;

   c = mu'*R;

   I=c>gamma;

   I=I.*cambionul;

   L=logical(I);

   Z(L)= (gamma(L)-c(L))./(2*w(L));

   I=abs(c) <= gamma;

   I=I.*cambionul;

   L=logical(I);

   Z(L)=0;

   I=c < -gamma;

   I=I.*cambionul;

   L=logical(I);

   Z(L) = (-gamma(L)-c(L))./(2*w(L));

   Z(posmanten) = -((gamma(posmanten)+c(posmanten))./(2*w
      (posmanten)));
```

```
Z(negmanten) = -((-gamma(negmanten)+c(negmanten))./(2*
   w(negmanten)));

lambda= Z;

m=lambda.*cambionul;

t=R2*m';

te2=t2*mu;

g = -b+t-t1-te2;

Ati=ti;

Abeta=beta;

y=g+Q*mu;

beta = dQ\y;

error = norm(g);

verr(k)=error;

vk(k)=k;

ni=k2/50;

if k2 == 1

   lambda1=lambda;

end

if isreal(ni) && (ni > 0) && (rem(ni,1) == 0)

   lambda2=lambda;

   vector1pos=lambda1>0; % cojo los positivos de la
      primera lambda

   vector2pos=lambda2>0; % cojo los positivos de la
      segunda lambda
```

```
sumapos=vector1pos+vector2pos; % los sumo de forma
   que solo sumaran 2 las posiciones que hayan sido
    positivas en ambas lambdas

posmanten=sumapos>1; %saco vector con los que
   signos positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la
   primera lambda

vector2neg=lambda2<0; % cojo los negativos de la
   segunda lambda

sumaneg=vector1neg+vector2neg; % los sumo de forma
   que solo sumaran 2 las posiciones que hayan sido
    negativos en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que
   signos negativos que se han mantenido

cambionulinv=negmanten+posmanten; % vector donde
   los 0 indican los signos que han cambiado o
   valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;

TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;
```

```
      R1=R.*cambionulinv;

      R2=R.*cambionul;

      t1=R1*suma';

      n3=R1';

      d3=2*(w.*cambionulinv);

      s3=n3./d3';

      TF3=isnan(s3);

      s3(TF3)=0;

      t2=R1*s3;

      lambda1=lambda2;
   end

end
semilogy(vk,verr)
end
```

Código 8.11 FISTA ACTUALIZACIÓN ERROR.

```
function [mu,lambda,k,error] = FISTAERROR(H,R,b,eps,w,
   gamma)

[m,n]=size(R);

Abeta= zeros(m,1);

beta= zeros(m,1);

k=0;

Ati=1;

error=1;

errori=1;

Q = R*inv(H)*R';
```

```
dQ= decomposition(Q,'chol','upper');

Z= zeros(1,n);

verr=[];

vk=[];

gammai=zeros(1,n);

while errori > eps

   k=k+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;

   c = mu'*R;

   I=c>gammai;

   Z(I)= (gammai(I)-c(I))./(2*w(I));

   I=abs(c) <= gammai;

   Z(I)=0;

   I=c < -gammai;

   Z(I) = (-gammai(I)-c(I))./(2*w(I));

   lambda1= Z;

   t = R*lambda1';

   g = t-b;

   Ati=ti;

   Abeta=beta;

   y=g+Q*mu;

   beta = dQ\y;
```

```
    errori = norm(g);

    verr(k)=errori;

    vk(k)=k;

end

while k<10

    k=k+1;

    ti=0.5*(1+sqrt(1+4*(Ati)^2));

    mu=beta+(Ati-1)*(beta-Abeta)/ti;

    c = mu'*R;

    I=c>gamma;

    Z(I)= (gamma(I)-c(I))./(2*w(I));

    I=abs(c) <= gamma;

    Z(I)=0;

    I=c < -gamma;

    Z(I) = (-gamma(I)-c(I))./(2*w(I));

    lambda2= Z;

    t = R*lambda2';

    g = t-b;

    Ati=ti;

    Abeta=beta;

    y=g+Q*mu;

    beta = dQ\y;

    error = norm(g);

    verr(k)=error;
```

```
   vk(k)=k;
end

vector1pos=lambda1>0; % cojo los positivos de la primera
   lambda

vector2pos=lambda2>0; % cojo los positivos de la segunda
   lambda

sumapos=vector1pos+vector2pos; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido positivas
    en ambas lambdas

posmanten=sumapos>1; %saco vector con los que signos
   positivos que se han mantenido

vector1neg=lambda1<0; % cojo los negativos de la primera
   lambda

vector2neg=lambda2<0; % cojo los negativos de la segunda
   lambda

sumaneg=vector1neg+vector2neg; % los sumo de forma que
   solo sumaran 2 las posiciones que hayan sido negativos
    en ambas lambdas

negmanten=sumaneg>1; %saco vector con los que signos
   negativos que se han mantenido

cambionulinv=negmanten+posmanten; % vector donde los 0
   indican los signos que han cambiado o valores nulos

cambionul=abs(cambionulinv-ones(1,n));

n1=gamma.*posmanten;

n2=-(gamma.*negmanten);

d1=2*(w.*posmanten);

d2=2*(w.*negmanten);

s1=n1./d1;

s2=n2./d2;
```

```
TF1=isnan(s1);

TF2=isnan(s2);

s1(TF1)=0;

s2(TF2)=0;

suma=s1+s2;

R1=R.*cambionulinv;

R2=R.*cambionul;

t1=R1*suma';

n3=R1';

d3=2*(w.*cambionulinv);

s3=n3./d3';

TF3=isnan(s3);

s3(TF3)=0;

t2=R1*s3;

k2=0;

while error > eps

   k=k+1;

   k2=k2+1;

   ti=0.5*(1+sqrt(1+4*(Ati)^2));

   mu=beta+(Ati-1)*(beta-Abeta)/ti;

   c = mu'*R;

   I=c>gamma;

   I=I.*cambionul;

   L=logical(I);
```

```
Z(L)= (gamma(L)-c(L))./(2*w(L));

I=abs(c) <= gamma;

I=I.*cambionul;

L=logical(I);

Z(L)=0;

I=c < -gamma;

I=I.*cambionul;

L=logical(I);

Z(L) = (-gamma(L)-c(L))./(2*w(L));

Z(posmanten) = -((gamma(posmanten)+c(posmanten))./(2*w
   (posmanten)));

Z(negmanten) = -((-gamma(negmanten)+c(negmanten))./(2*
   w(negmanten)));

lambda= Z;

m=lambda.*cambionul;

t=R2*m';

te2=t2*mu;

g = -b+t-t1-te2;

Ati=ti;

Abeta=beta;

y=g+Q*mu;

beta = dQ\y;

error = norm(g);

verr(k)=error;
```

```
vk(k)=k;

ni=k2/50;

if k2 == 1

   lambda1=lambda;

   errorcomp=error;
end

if error < errorcomp*0.1

   lambda2=lambda;

   vector1pos=lambda1>0; % cojo los positivos de la
      primera lambda

   vector2pos=lambda2>0; % cojo los positivos de la
      segunda lambda

   sumapos=vector1pos+vector2pos; % los sumo de forma
      que solo sumaran 2 las posiciones que hayan sido
       positivas en ambas lambdas

   posmanten=sumapos>1; %saco vector con los que
      signos positivos que se han mantenido

   vector1neg=lambda1<0; % cojo los negativos de la
      primera lambda

   vector2neg=lambda2<0; % cojo los negativos de la
      segunda lambda

   sumaneg=vector1neg+vector2neg; % los sumo de forma
      que solo sumaran 2 las posiciones que hayan sido
       negativos en ambas lambdas

   negmanten=sumaneg>1; %saco vector con los que
      signos negativos que se han mantenido

   cambionulinv=negmanten+posmanten; % vector donde
      los 0 indican los signos que han cambiado o
      valores nulos

   cambionul=abs(cambionulinv-ones(1,n));
```

```
      n1=gamma.*posmanten;

      n2=-(gamma.*negmanten);

      d1=2*(w.*posmanten);

      d2=2*(w.*negmanten);

      s1=n1./d1;

      s2=n2./d2;

      TF1=isnan(s1);

      TF2=isnan(s2);

      s1(TF1)=0;

      s2(TF2)=0;

      suma=s1+s2;

      R1=R.*cambionulinv;

      R2=R.*cambionul;

      t1=R1*suma';

      n3=R1';

      d3=2*(w.*cambionulinv);

      s3=n3./d3';

      TF3=isnan(s3);

      s3(TF3)=0;

      t2=R1*s3;

      lambda1=lambda2;

      errorcomp=error;
   end

end
semilogy(vk,verr)
```

```
end
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Teodoro Alamo, Pablo Krupa, and Daniel Limon, Gradient based restart fista, 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, 2019, pp. 3936–3941.

[2] Teodoro Alamo, Daniel Limon, and Pablo Krupa, Restart fista with global linear convergence, 2019 18th European Control Conference (ECC), IEEE, 2019, pp. 1969–1974.

[3] Teodoro Alamo, Daniel G Reina, Martina Mammarella, and Alberto Abella, Covid-19: Open-data resources for monitoring, modeling, and forecasting the epidemic, Electronics **9** (2020), no. 5, 827.

[4] Gerardo Alfonso, A Daniel Carnerero, Daniel R Ramirez, and Teodoro Alamo, Stock forecasting using local data, IEEE Access **9** (2020), 9334–9344.

[5] Alvaro, `machinelearningparatodos.com`, 9 de Agosto de 2023.

[6] IMHR Antunes and MTD Albuquerque, Using indicator kriging for the evaluation of arsenic potential contamination in an abandoned mining area (portugal), Science of the Total Environment **442** (2013), 545–552.

[7] Amir Beck and Marc Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM journal on imaging sciences **2** (2009), no. 1, 183–202.

[8] Bernard Marr, `www.forbes.com`, 11 de Agosto de 2023.

[9] José M Bravo, Teodoro Álamo, and Daniel Limón, Aplicación del análisis intervalar al control predictivo basado en modelo, Aplicación del análisis intervalar al control predictivo basado en modelo (2005), 1000–1010.

[10] José Manuel Bravo, Teodoro Alamo, Manuel Vasallo, and ME Gegundez, A general framework for predictors based on bounding techniques and local approximation, IEEE Transactions on Automatic Control **62** (2016), no. 7, 3430–3435.

[11] Andy Canion, Lori McCloud, and Dean Dobberfuhl, Predictive modeling of elevated groundwater nitrate in a karstic spring-contributing area using random forests and

regression-kriging, Environmental Earth Sciences **78** (2019), no. 9, 271.

[12] A Daniel Carnerero, Daniel R Ramirez, and Teodoro Alamo, Probabilistic interval predictor based on dissimilarity functions, IEEE Transactions on Automatic Control **67** (2021), no. 12, 6842–6849.

[13] Alfonso Daniel Carnerero Panduro, Probabilistic data-driven methods for forecasting, identification and control, (2022).

[14] Jean-Paul Chiles and Pierre Delfiner, Geostatistics: modeling spatial uncertainty, vol. 713, John Wiley & Sons, 2012.

[15] Noel Cressie, Kriging nonstationary data, Journal of the American Statistical Association **81** (1986), no. 395, 625–634.

[16] ———, The origins of kriging, Mathematical geology **22** (1990), 239–252.

[17] dataversity, `www.dataversity.net`, 12 de Agosto de 2023.

[18] Ingrid Daubechies, Michel Defrise, and Christine De Mol, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences **57** (2004), no. 11, 1413–1457.

[19] Gamze Erdogan Erten, Mahmut Yavuz, and Clayton V Deutsch, Combination of machine learning and kriging for spatial estimation of geological attributes, Natural Resources Research **31** (2022), no. 1, 191–213.

[20] Sven F Falkenberg and Stefan Spinler, The role of novel data in maintenance planning: Breakdown predictions for material handling equipment, Computers & Industrial Engineering **169** (2022), 108230.

[21] Mário AT Figueiredo and Robert D Nowak, An em algorithm for wavelet-based image restoration, IEEE Transactions on Image Processing **12** (2003), no. 8, 906–916.

[22] Francisco Charte, `www.campusmvp.es`, 9 de Agosto de 2023.

[23] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro, Kriging is well-suited to parallelize optimization, Computational intelligence in expensive optimization problems (2010), 131–162.

[24] A Ardeshir Goshtasby, Image registration: Principles, tools and methods, Springer Science & Business Media, 2012.

[25] Deng Huang, Theodore T Allen, William I Notz, and R Allen Miller, Sequential kriging optimization using multiple-fidelity evaluations, Structural and Multidisciplinary Optimization **32** (2006), 369–382.

[26] Zhenya Jia, Eddie Davis, Fernando J Muzzio, and Marianthi G Ierapetritou, Predictive modeling for pharmaceutical processes using kriging and response surface, Journal

of Pharmaceutical Innovation **4** (2009), 174–186.

[27] Hongsheng Jiang, Sujun Dong, Zheng Liu, Yue He, and Fengming Ai, Performance prediction of the centrifugal compressor based on a limited number of sample data, Mathematical Problems in Engineering **2019** (2019), 1–13.

[28] Iwao Kobayashi, 20 claves para mejorar la fica, Routledge, 2020.

[29] Manu Kohli, Predicting equipment failure on sap erp application using machine learning algorithms, Int. J. Eng. Technol. **7** (2018), no. 2.28, 306.

[30] Pablo Krupa, Daniel Limon, and Teodoro Alamo, Harmonic based model predictive control for set-point tracking, IEEE Transactions on Automatic Control **67** (2020), no. 1, 48–62.

[31] Pablo Krupa, Mario Pereira, Daniel Limon, and Teodoro Alamo, Single harmonic based model predictive control for tracking, 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, 2019, pp. 151–156.

[32] Geoffrey M Laslett, Kriging and splines: an empirical comparison of their predictive performance in some applications, Journal of the American Statistical Association **89** (1994), no. 426, 391–400.

[33] Jean Lefebvre, Helene Roussel, Eric Walter, Dominique Lecointe, and Walid Tabbara, Prediction from wrong models: the kriging approach, IEEE Antennas and propagation Magazine **38** (1996), no. 4, 35–45.

[34] Kevin Loquin and Didier Dubois, Kriging and epistemic uncertainty: a critical discussion, Methods for handling imperfect spatial information (2010), 269–305.

[35] ———, Kriging with ill-known variogram and data, Scalable Uncertainty Management: 4th International Conference, SUM 2010, Toulouse, France, September 27-29, 2010. Proceedings 4, Springer, 2010, pp. 219–235.

[36] Gicela Lupera, Ahmed Shokry, Sergio Medina-González, Eduardo Vyhmeister, and Antonio Espuña, Ordinary kriging: A machine learning tool applied to mixed-integer multiparametric approach, Computer Aided Chemical Engineering, vol. 43, Elsevier, 2018, pp. 531–536.

[37] Lucía Masero Bravo, Una aplicación de interpolación kriging para el estudio de la evolución de la pandemia covid-19 en españa y en la comunidad valenciana, (2020).

[38] Dory Merhy, Teodoro Alamo, Cristina Stoica Maniu, and Eduardo F Camacho, Zonotopic constrained kalman filter based on a dual formulation, 2018 IEEE Conference on Decision and Control (CDC), IEEE, 2018, pp. 6396–6401.

[39] Victor Mirasierra Calleja, Funciones de verosimilitud en el entorno del aprendizaje automático, (2017).

[40] ourworldindata, `ourworldindata.org`, 10 de Diciembre de 2023.

[41] ree, www.ree.es, 20 de Diciembre de 2023.

[42] Mirko Reguzzoni, Fernando Sansó, and Giovanna Venuti, The theory of general kriging, with applications to the determination of a local geoid, Geophysical journal international **162** (2005), no. 2, 303–314.

[43] Jacob Roll, Alexander Nazin, and Lennart Ljung, Nonlinear system identification via direct weight optimization, Automatica **41** (2005), no. 3, 475–490.

[44] Scott Roy, Algorithms for convex optimization with applications to data science, Ph.D. thesis, 2017.

[45] Jose R Salvador, D Muñoz de la Peña, T Alamo, and A Bemporad, Data-based predictive control via direct weight optimization, IFAC-PapersOnLine **51** (2018), no. 20, 356–361.

[46] Jose R Salvador, D Munoz de la Pena, DR Ramirez, and T Alamo, Predictive control of a water distribution system based on process historian data, Optimal Control Applications and Methods **41** (2020), no. 2, 571–586.

[47] Jose R Salvador, Daniel Rodriguez Ramirez, Teodoro Alamo, and David Munoz de la Pena, Offset free data driven control: application to a process control trainer, IET Control Theory & Applications **13** (2019), no. 18, 3096–3106.

[48] Statista, es.statista.com, 22 de Diciembre de 2023.

[49] A Stein and LCA Corsten, Universal kriging and cokriging as a regression procedure, Biometrics (1991), 575–587.

[50] tradingeconomics, tradingeconomics.com, 10 de Diciembre de 2023.

[51] Stefan Vujović, Isidora Stanković, Miloš Daković, and Ljubiša Stanković, Comparison of a gradient-based and lasso (ista) algorithm for sparse signal reconstruction, 2016 5th Mediterranean Conference on Embedded Computing (MECO), IEEE, 2016, pp. 377–380.

[52] SJ Yakowitz and F Szidarovszky, A comparison of kriging with nonparametric regression methods, Journal of Multivariate Analysis **16** (1985), no. 1, 21–53.