Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Funciones de verosimilitud en el entorno del aprendizaje automático

Autor: Víctor Mirasierra Calleja

Tutor: Teodoro Álamo Cantarero

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Funciones de verosimilitud en el entorno del aprendizaje automático

Autor:
Víctor Mirasierra Calleja

Tutor:
Teodoro Álamo Cantarero
Catedrático de Universidad

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Grado: Funciones de verosimilitud en el entorno del aprendizaje automático

Autor: Víctor Mirasierra Calleja
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

Este trabajo no habría sido posible sin el apoyo incondicional de mi madre Dory, quien ha conseguido guiarme durante todos estos años de estudio. También agradecer el apoyo a mi tutor Teo, sin el cual no habría descubierto el gran campo del machine learning, centro de este trabajo y en el que me gustaría trabajar en un futuro.

Resumen

Este documento recoge el estudio y desarrollo de funciones de disimilitud en el entorno del aprendizaje automático, así como su comparación con otras metodologías destacadas del sector, como los mínimos cuadrados o las máquinas de vectores soporte (SVM), desarrollando librerías especializadas para ello, que contemplan los principales problemas en el campo del aprendizaje automático o machine learning.

Durante el estudio se tratará la importancia del campo del aprendizaje automático en la actualidad y los diferentes problemas que existen ligados a él, entre los cuales destacan los problemas de regresión y clasificación, pero no son los únicos, ya que a medida que avanza la tecnología se necesitan soluciones a otro tipo de problemas como el tratamiento de imágenes o el desarrollo de sistemas de recomendación.

Puesto que el aprendizaje automático es una rama fuertemente ligada a los datos, se estudiarán en profundidad, incluyendo el formato, el proceso de tratamiento o la forma de introducirlos en los diferentes algoritmos. Además, se propondrán diversas formas de representar los datos, lo que permitirá medir el desempeño de los algoritmos y comprobar que éstos realmente *aprenden* según lo esperado.

Como se verá, las funciones de verosimilitud propuestas ofrecen una familia entera de soluciones a algunos de estos problemas. En especial, cabe destacar su relevancia en el problema de predicción intervalar, ya que se adaptan mejor a las funciones probabilísticas de los datos del problema, lo que desemboca en unos mejores resultados.

Por último, se comprobará que a veces metodos simples como un mantenedor de los datos ofrece mejores resultados que métodos más complejos y requeridores de una mayor potencia de cálculo. Esto es debido típicamente a la naturaleza ruidosa y díficil de predecir de algunas muestras.

Abstract

The following report includes the study and development of dissimilarity functions in the context of machine leaning, as well as a comparison with other featured methodologies like the least-squares methods or the support vector machine (SVM) ones. This is accomplished by developing various specialized libraries which take into account the main difficulties in the field of machine learning.

Throughout this report, I will try to address the significance of the machine learning field these days and the problems linked to it. Among which, we usually stand out the regression and classification problems, but as technology develops, others problems such as image processing or recommender systems have appeared and machine leaning has found some interesting ways to solve them.

Because maching learning is highly linked to data, they will be essential in this report and their format, treatment and way to use in algorithms will be addressed. Furthermore, multiple ways of data representation will be proposed, which will allow us to measure the performance of the algorithms and check whether they are really working as planned.

As we will see, the proposed dissimilarity functions offer a family of solutions to some of the main machine learning problems. In particular, it's remarkable their relevance in the interval prediction problem, as they adapt better to data probabilistic functions, which leads to better results.

To conclude, it will be checked that easier methods such as a zero order maintainer can sometimes offer a better performance than more sophisticated ones. This is usually because of the noisy and hardly predictable data.

Índice

1 Introducción

Desde el pasado siglo y gracias al avance tecnológico, el control clásico ha evolucionado y han surgido nuevas ramas de éste que aprovechan la potencia de los ordenadores para resolver los diferentes problemas de control, abriendo camino a resultados más óptimos a cambio normalmente de un mayor requerimiento de potencia de cálculo.

En especial, ha surgido una rama moderna del control automático conocida como Machine Learning, gracias a la cual podemos conseguir que un computador *aprenda*, mediante una serie de datos, a resolver un determinado problema. En el mundo de datos en el que vivimos, esto se ha convertido en una herramienta muy potente y muy amplia, aplicable a una gran variedad de problemas. En este documento, se informará acerca del aprendizaje automático, su historia, sus problemas y los algoritmos que se suelen utilizar para resolverlos. Asimismo, se propondrá una metodología de resolución basada en funciones de verosimilitud que complementan algunos de los algoritmos ya existentes.

Todo esto vendrá acompañado de una comparación de los resultados de los diversos algoritmos en diferentes sets de datos pertenecientes a distintos problemas. Para este trabajo, se han desarrollado librerías capaces de resolver los problemas que se citan. Todo el código desarrollado es propio (a excepción de funciones incluidas en matlab y el método Fista de optimización), y estará recogido al final del documento.

1.1 Objetivos

A continuación se detalla el programa de trabajo seguido y el orden en el que se ha seguido para conseguir desarrollar las tareas descritas anteriormente. Este orden supondrá a su vez una directriz de este trabajo.

1. Toma de contacto con el entorno de trabajo del *Machine Learning*. Estudio de los principales problemas y metodologías típicas para su resolución.
2. Definición de los problemas a tratar que permitan comprobar la eficacia de los métodos estudiados.
3. Desarrollo del algoritmo de disimilitud.
4. Solución de los problemas mediante algoritmos tradicionales.
5. Aplicación del nuevo algoritmo a los problemas planteados.
6. Representación y análisis de los resultados obtenidos.

1.2 Organización de la memoria

En el capítulo 2 se realiza una introducción al campo del aprendizaje automático. El capítulo comienza explicando los orígenes del campo y comentando la importancia de éste. Posteriormente, describe los principales problemas que se tratan, los tipos de aprendizaje que se realizan, las limitaciones del machine learning y por último se valoran algunos de los métodos más conocidos de resolución de problemas de aprendizaje automático.

El capítulo 3 se centrará en los datos y su tratamiento para el correcto funcionamiento de los algoritmos. Se presentarán los problemas que se tratarán en este trabajo y se propondrán formas de representación de los datos para poder extraer información de ellos.

El capítulo 4 está orientado al desarrollo de una familia de funciones de verosimilitud. Se procederá a un desarrollo matemático de estas funciones, en el que se comentará la importancia de éstas en la resolución de algunos problemas de aprendizaje automático. Finalmente, se estudiará la forma de resolver estos problemas gracias a estas funciones.

En el capítulo 5 se muestran los resultados obtenidos por los diferentes algoritmos en los sets de datos considerados. Se ofrecen datos como el error cuadrático medio cometido por cada algoritmo (y sus parámetros asociados) o tablas y gráficas que comparan los valores reales con los valores predichos por los algoritmos.

El capítulo 6 supone una análisis de los resultados obtenidos en el apartado 6, comentando el desempeño de las diferentes metodologías en los problemas propuestos. El capítulo termina con una reflexión sobre los pasos que se deberían realizar para intentar mejorar los resultados obtenidos.

Por último, el capítulo 7 recopila todo el código desarrollado para este trabajo.

2 Aprendizaje automático

Hay muchas definiciones de aprendizaje automático. En general todas ellas coinciden en lo mismo: el aprendizaje automático es una rama de la inteligencia artificial que desarrolla técnicas que permiten a los ordenadores aprender. La metodología de aprendizaje es la de alimentar al ordenador o máquina con un conjunto de datos representativos del problema que tendrá que resolver, tal que la máquina sea capaz de obtener información de ellos y resolver el problema.

Es difícil saber cuándo se creó el campo, ya que bebe de muchos otros como las matemáticas o la ciencia computacional. Aun así, sabemos que fue Arthur Samuel en 1952 quien escribió el primer programa por computador capaz de aprender con el tiempo. Aunque difiere mucho del aprendizaje automático moderno, este programa era capaz de jugar al juego de las damas y mejorar en el juego a medida que jugaba más partidas. Posteriormente llegaron metodologías que se asemejan más al aprendizaje automático tal como lo conocemos hoy. En 1957, Frank Rosenblatt diseñó la primera red neuronal para ordenadores, el perceptrón, el cual utilizaba conocimientos de neurociencia para conseguir el aprendizaje. En 1967 se escribió el algoritmo del "vecino cercano", el cual permitía ya a los ordenadores un reconocimiento básico de patrones.

El aprendizaje automático continuó levantando el interés de los investigadores, y hoy ya es visible en múltiples aplicaciones tal como sistemas de recomendación de películas o música, o la conducción autónoma, en la cual se han hecho grandes avances.

2.1 Problemas típicos

El campo del aprendizaje automático es muy amplio y abarca numerosos tipos de problemas, entre los cuales siempre se suelen destacar los de clasificación y regresión. Pero a medida que se han ido desarrollando los conocimientos sobre este campo, han surgido otro tipo de problemas que han sido posibles de resolver gracias al aprendizaje automático. Entre ellos destacan los problemas de detección de anomalías y el desarrollo de sistemas de recomendación, entre otros. Aunque pueda parecer que estos problemas son muy diferentes entre sí, en realidad se han conseguido formular de forma similar, lo que hace posible que su resolución sea posible con algoritmos similares.

Mencionar también que hay clasificaciones secundarias en cada tipo de problema, de las cuales se destacarán la clasificación supervisada y la clasificación insupervisada.

2.1.1 Problema de regresión

El objetivo de este tipo de problema es el de encontrar una función matemática cuya curva se adapte lo mejor posible a los datos. Esto nos permitirá predecir el valor de una variable salida conocidos los valores de otras variables entradas, que suelen incluir tanto parámetros del sistema, como valores pasados de la variable salida.

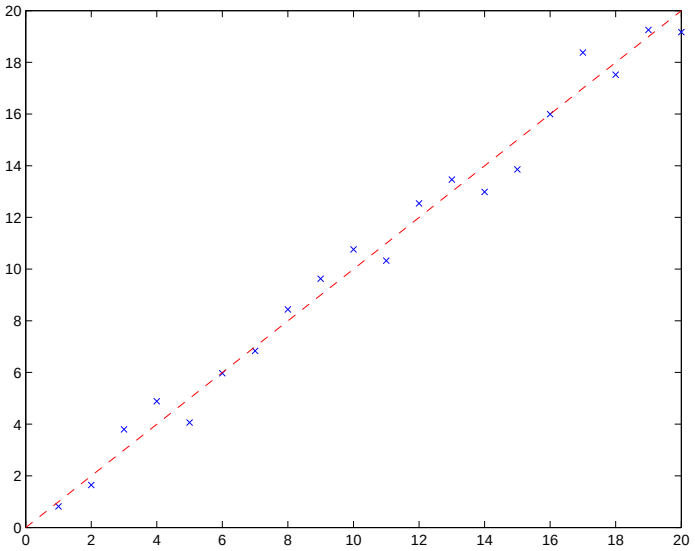

Figura 2.1 Ejemplo problema de regresión.

Predicción temporal

La predicción temporal es una particularización del problema de regresión. En este caso, tenemos que predecir el valor que tendrá cierta curva partiendo del conocimiento de sus valores pasados. Así, se puede comprobar que es un ejemplo del problema de regresión colocando los valores pasados de las salidas como variables entradas. Es conveniente aclarar que, aunque se pueda ampliar el horizonte de predicción, esto aumenta la incertidumbre y por lo tanto el error del algoritmo.

Predicción intervalar

La predicción intervalar se puede entender como un caso particular de regresión. Mientras que la regresión ordinaria nos proporciona el punto de mayor confianza, la esperanza matemática de que el valor real se corresponda con el punto predicho es 0. Para evitar eso, podemos proporcionar una solución menos ambiciosa. Dado un grado de confianza en la predicción, el problema consiste en establecer un margen tal que la esperanza de que el valor real entre en dicho margen sea la establecida por el usuario.

2.1.2 Problema de clasificación

A diferencia del problema de regresión, en el problema de regresión los datos se encuentran agrupados en clases. El objetivo será el de calcular la clase a la que pertenece un determinado dato, conocidos los valores de las variables entradas. Es destacable que este tipo de problema, a diferencia del problema de regresión, tiene una salida discreta y limitada al número de clases existente.

En función del número de clases existente, podemos distinguir entre clasificación binaria y multiclase.

Clasificación binaria

La clasificación binaria recibe su nombre porque separa dos clases o regiones. Es la clasificación más básica y se emplea también para separar una única clase, entendiendo por la segunda clase aquella que engloba todos los datos que no pertenecen a la clase que queremos separar. La clasificación en género (hombre o mujer) basada en datos como la altura o el peso es un ejemplo de este problema.

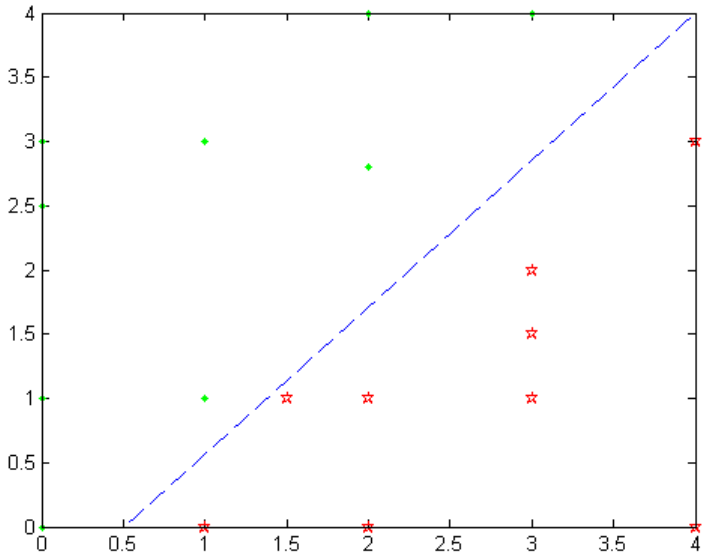

Figura 2.2 Ejemplo problema de clasificación binaria.

Clasificación multiclase

La clasificación multiclase separa varias clases. Normalmente se resuelve como un conjunto de clasificaciones binarias en lo que se conoce como la técnica "1 vs all", cuya idea es la de crear clasificadores independientes para cada clase, y clasificar los datos en aquella clase que obtenga un resultado más sólido. Un ejemplo puede ser la clasificación de flores en función de datos como la longitud del pétalo o del sépalo.

2.1.3 Detección de anomalías

La idea es la de, dado un set de datos, buscar si existe alguno que sobresalga de la media de su clase. Se puede entender como una extensión del problema de clasificación. Este tipo de problema se puede entender como una clasificación binaria sesgada (el número de anomalías es muy reducido respecto al de datos normales) si se tuviera un histórico de anomalías. En el caso de no poseer un histórico, se trataría de un problema de aprendizaje no supervisado en el que se consideran anomalías aquellos datos que se separen del resto. Un ejemplo de este tipo de problema es el de detectar piezas defectuosas en el proceso de fabricación.

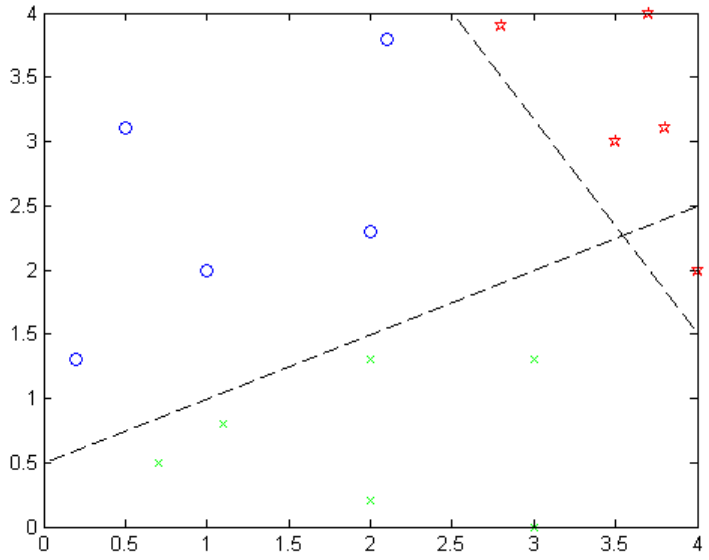

Figura 2.3 Ejemplo problema de clasificación multiclase.

2.1.4 Sistemas de recomendación

Este problema ha surgido debido al desarrollo de internet y la creación de compañías como *Amazon* o *Netflix*. La idea es la de sugerir al usuario determinados productos, basándose en sus gustos (recogidos típicamente en un historial). Aunque este tema no se tratará en este trabajo, es notable conocerlo ya que está adquiriendo importancia y supone una gran fuente de beneficios en numerosas compañías.

2.1.5 Aprendizaje supervisado-Aprendizaje no supervisado

Estos tipos no aprendizajes no son problemas, sino dos herramientas para resolver los problemas anteriores. La diferencia entre ambos reside en que, mientras el aprendizaje supervisado alimenta al algoritmo con valores de salida, indicando por ejemplo la clase a la que pertenece un dato, el aprendizaje no supervisado no lo hace, y deja que el algoritmo divida los datos sin una alimentación previa de la salida. Cada uno se usará en determinados problemas. Así, el aprendizaje no supervisado puede ser útil en situaciones en las que desconozcamos a priori el número de clases, como por ejemplo en la agrupación de alumnos según su rendimiento académico, mientras que el aprendizaje supervisado será más beneficioso en problemas como la detección de cáncer, en los que un aprendizaje no supervisado podría resultar en una clasificación diferente, y nunca indicaría cuál de las clases sería la afectada por el cáncer. No hay que olvidar que, mientras el aprendizaje supervisado proporciona una etiqueta a la salida, el aprendizaje no supervisado simplemente separa las clases, sin ofrecer ninguna información sobre éstas más que tienen características -desconocidas por el usuario- en común.

2.1.6 Ejemplo: Tratamiento de imágenes

Los problemas antes mencionados son solo algunos de los problemas en los que se sustenta el aprendizaje automático. Hay multitud de variantes a los problemas comentados. Un

ejemplo de estos y que se tratará en este estudio es el de tratamiento de imágenes. Veremos como, por su naturaleza, se trata a priori de un problema de aprendizaje no supervisado, ya que los datos no suelen tener etiquetas o marcas que nos permitan tratarlo de forma supervisada (aunque se pueden dar situaciones en las que sí, no es lo normal). Así, veremos como podemos emplear algoritmos de aprendizaje no supervisado para resolver problemas en este ámbito de la ingeniería.

2.2 Limitaciones del aprendizaje automático

A pesar de ser un campo prometedor, el machine learning tiene algunas limitaciones a considerar cuando se trata de resolver un problema. A continuación trataré dos de las limitaciones más relevantes.

2.2.1 Maldición de la dimensión

La primera se trata de un problema conocido en inglés por el nombre de *Curse of dimensionality*. Este no es un problema a resolver, sino uno que se presenta a medida que las dimensiones de nuestro problema principal crecen. La maldición de la dimensión afirma que, a medida que aumentamos el número de características de un problema (o dimensiones), la cantidad de datos requerida para generalizar adecuadamente aumenta exponencialmente. Esto es debido a la reducción exponencial del volumen de la hiperesfera a medida que aumenta el número de dimensiones de la misma, lo que complica la tarea de encontrar puntos alrededor de ésta. Para lidiar con esto, se trata de elegir cuidadosamente los datos con los que se alimentan los algoritmos, intentando evitar datos irrelevantes para el problema.

2.2.2 Sobre ajuste y falta de ajuste

Otro de los problemas a los que se enfrentan los algoritmos de aprendizaje automático es el de conseguir que el algoritmo generalice correctamente a partir de los datos que le entregamos. Al resolver un problema, debemos tener en cuenta que el algoritmo va a trabajar con datos diferentes a aquellos con los que lo estamos alimentando. De no ser así, podríamos conseguir ajustar una curva que predijera los datos sin error, tal que pasara por todos los puntos definidos. Aunque matemáticamente es posible, esta solución es indeseada ya que el modelo solo sería perfecto para los datos de entrenamiento, mientras que el error cometido en el resto de datos sería elevado. Así, normalmente se introduce un término de regulación o se detiene el proceso de aprendizaje cuando se considera que el resultado generaliza lo suficientemente bien los datos.

2.3 Estado del arte

En la sección anterior se han mencionado diferentes ejemplos de problemas de aprendizaje automático. Aunque puedan parecer problemas muy diversos, en la práctica, nos encontramos generalmente algoritmos transversales, que, con escasas modificaciones aplican a muchos de los tipos de problemas. El objetivo de la clasificación anterior es el de definir a qué conjunto de problemas es aplicable cada algoritmo.

Los métodos se basan típicamente en la resolución de un problema de optimización con restricciones que se puede formular de la forma:

$$\mathscr{L} = \frac{1}{2}\sum_{i=1}^{N}(e^2(\theta)) + \lambda\frac{\theta^T\theta}{2} \tag{2.1}$$

Donde $\mathscr{L}$ sería la denominada función de coste, cuyo valor queremos minimizar.

A continuación se tratarán algunos de los más conocidos y empleados tradicionalmente en el campo de aprendizaje automático.

2.3.1 Gradient descent

El Gradient Descent es un método de optimización que consigue emplear la definición del gradiente de una función para seguir a ésta hasta su mínimo. Recordemos que el objetivo del aprendizaje es el de minimizar una cierta función denominada *función de coste* por lo que ésta es una solución sencilla y bastante intuitiva que será muy útil para empezar a resolver problemas. Entre los inconvenientes del este algoritmo, cabe destacar el incremento de tiempo de cálculo a medida que el número de datos disponibles aumenta, y el hecho de que seguir la dirección del gradiente puede provocar que el algoritmo acabe en un mínimo local si el problema no fuera convexo.

La hipótesis que usaremos para predecir la salida será $h(\theta) = \theta^T x$, siendo θ el vector de parámetros y x las entradas del algoritmo. Así, tendremos que minimizar el error cuadrático medio, osea,

$$J(\theta) = \frac{1}{2m}(\theta^T x - y)^T(\theta^T x - y) \tag{2.2}$$

siendo m el número de datos de entrenamiento. $J(\theta)$ es la función de coste. En el método del gradient descent nos interesa su derivada. La calculamos algebraicamente, obteniendo la regla de actualización de los parámetros θ:

$$\theta = \theta - \alpha x^T(\theta^T x - y) \tag{2.3}$$

Esta actualización se repite hasta la convergencia de los parámetros o hasta un cierto límite de iteraciones. El parámetro α recibe el nombre de ratio de aprendizaje, y modifica la velocidad de aprendizaje del algoritmo. Un valor bajo nos asegura que el algoritmo converge, aunque lentamente, mientras que un valor alto supone que los parámetros se actualicen más rápido pero puede resultar en la divergencia de estos.

Para solucionar el problema del tiempo de cálculo creciente con el aumento del número de datos, se recurre a modificaciones del algoritmo como *Stochastic Gradient Descent* o *Mini Batch Gradient Descent*. La principal característica de estos dos métodos es que no necesitan del conjunto completo de datos en cada iteración, por lo que el algoritmo avanza más rápido hacia el mínimo. La diferencia de tiempos es especialmente notable si el set de datos es muy grande. La idea es la de realizar divisiones en el set de datos de entrenamiento y alimentar al algoritmo con estas, en el caso del *Stochastic Gradient Descent*, las divisiones serán de un único dato, mientras que en el *Mini Batch Gradient Descent* serán de varios.

Por otra parte, seguimos teniendo el problema de que los algoritmos acaban en un mínimo local, en vez de en el mínimo global, que es el que realmente nos interesa. Solucionar esto

es más complejo, y la solución más empleada es la de ejecutar varias veces el algoritmo desde diferentes puntos iniciales y escoger el mejor resultado obtenido.

2.3.2 Minimos cuadrados ordinarios (OLS)

El método de los mínimos cuadrados nos permite identificar modelos lineales en los parámetros de la forma

$$y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) = b_1 u(k-1) + \cdots + b_m u(k-m) \tag{2.4}$$

Dicha formulación se puede simplificar de la forma $Y = \theta^T X$, siendo X el regresor y θ el vector de parámetros de nuestro sistema. Nuestro objetivo será el de, dado un regresor, calcular el vector de parámetros que nos permita minimizar el error cuadrático medio de la salida. Así, definimos el error del algoritmo como la diferencia entre la salida real y la salida predicha: $E = Y - \hat{\theta}^T X$. El error cuadrático medio será por tanto:

$$J(\hat{\theta}) = \frac{1}{N} E^T E \tag{2.5}$$

siendo N el número de datos con el que alimentamos el algoritmo y $J(\hat{\theta})$ la función de coste de este algoritmo. Podemos reescribir $J(\hat{\theta})$ como:

$$J(\hat{\theta}) = \frac{1}{N} (Y - \hat{\theta}^T X)^T (Y - \hat{\theta}^T X) \tag{2.6}$$

Puesto que nuestro objetivo es el de reducir el error cuadrático medio, calculamos la derivada de la función de coste y la igualamos a 0 para hallar el valor del regresor que minimiza nuestra función.

$$\frac{dJ(\hat{\theta})}{d\hat{\theta}} = 0 \tag{2.7}$$

$$(\hat{\theta}^T X - Y) X = 0 \tag{2.8}$$

De ahí obtenemos que el estimador de mínimos cuadrados será:

$$\theta^* = (X^T X)^{-1} X^T Y \tag{2.9}$$

Como vemos, este segundo método emplea una solución algebraica del problema, por lo que evitamos la iteración que necesitábamos en el Gradient Descent. Sin embargo, sí necesita calcular la inversa de $X^T X$, algo muy caro computacionalmente que conlleva que el algoritmo sea muy lento cuando el número de datos es grande. También se puede dar el caso de que $X^T X$ sea no invertible. Aunque este caso es poco común, se puede solucionar eliminando características redundantes a nivel de datos, o bien calculando la pseudoinversa de la matriz.

Dentro de los mínimos cuadrados, existen variantes que permiten adaptarnos mejor al problema en cuestión, entre las cuales caben destacar los mínimos cuadrados recursivos, una reformulación del problema que permite utilizar el algoritmo en línea, reduciendo el coste computacional y siendo más interesante para sistemas con dinámica cambiante y capaces de obtener información de sí mismos a través de sensores.

Mencionar también los mínimos cuadrados ponderados, otra variante del método de los mínimos cuadrados que nos permite, con escasas variaciones respecto a la formulación original, dar más peso a unos datos que a otros. Normalmente son más relevantes para el problema los datos más recientes, aunque también pueden ser importantes ciertos datos pasados debido a una posible periodicidad de los mismos. El algoritmo de los mínimos cuadrados ponderados nos permite tener en cuenta esto y ponderar aquellos datos que sean más relevantes.

2.3.3 Máquinas de vectores soporte (SVM)

Los métodos explicados anteriormente funcionan bien, pero tienen una limitación importante: están limitados a clasificadores lineales. Si, como ocurre en el caso de la función lógica XOR, los datos son imposibles de separar mediante un clasificador lineal, estos métodos fallarán. Los algoritmos de SVM solucionan estos problemas en un tiempo de cálculo aceptable, gracias al empleo de funciones kernel, funciones muy estudiadas que nos permiten convertir un problema de una magnitud considerable (en término de dimensiones) en un problema relativamente simple.

El algoritmo de SVM fue presentado por Vapnik en 1992 y se ha convertido en uno de los algoritmos más populares y empleados en el aprendizaje automático moderno. Esto es debido en gran parte a su versatilidad, ya que engloba el método de mínimos cuadrados ordinarios, entre otros. Aunque cuando el número de datos incrementa mucho, el algoritmo pierde interés, su desempeño en sets de datos razonablemente grandes es tan bueno que es interesante su estudio.

Imaginemos dos conjuntos de datos perfectamente separables con un separador lineal como los de la figura 2.4. Hay infinitos separadores que permiten la clasificación perfecta entre dos conjuntos de datos, pero se busca el mejor de ellos. Es necesario entonces establecer un criterio por el que un separador es mejor que otro. El criterio que se escoge es simple e intuitivo: un separador será mejor cuanta más distancia deje a los puntos más cercanos de cada clase. Esta distancia recibirá el nombre de margen, y nuestro objetivo es maximizarlo a la vez que obtenemos un buen clasificador en términos de rendimiento (que clasifique correctamente las clases). Los puntos que están más cercanos al clasificador también reciben un nombre: vectores soporte. Estos puntos son los más útiles, ya que son los más propensos a ser mal clasificados. Esto nos lleva a una primera propiedad de estos algoritmos: podemos guardar únicamente los vectores soporte, ya que son los que más información aportan, y prescindir del resto de datos. Esto nos permitirá el ahorro de espacio de almacenamiento.

A continuación, se explicará el método SVM-LS, el cual hace uso de los métodos de SVM y OLS para obtener un mejor resultado.

Buscamos un clasificador de la forma $y = \theta^T x + b$, siendo θ el vector de parámetros como en el caso de OLS, $\mathbf{x}$ el vector o matriz de datos, y $\mathbf{b}$ la contribución del sesgo. Cualquier punto que cumpla $\theta^T x + b \geq M$ será clasificado como clase 1, mientras cualquier punto que cumpla $\theta^T x + b \leq -M$ será clasificado en la clase 2. Los puntos que cumplan la igualdad serán los vectores soporte.

Se puede demostrar que nuestro objetivo de aumentar el margen se traduce a minimizar el producto $\theta^T \theta$. Por lo que, sumado al problema de la correcta clasificación podemos

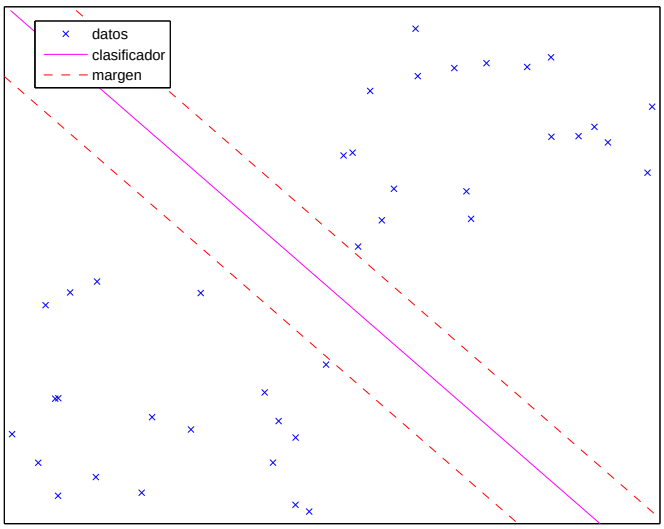

Figura 2.4 Clasificador de SVM.

reformular el problema de la forma:

$$\begin{aligned} min \quad & \frac{1}{2}\left(\theta^T\theta + \gamma E^T E\right) \\ s.a. \quad & Y - w^T\varphi(x) - b - E \end{aligned} \tag{2.10}$$

Donde Y es la salida objetivo, E es el error cometido y γ es un parámetro de balance, que se ajusta para darle mayor o menor peso a la normalización. Resolviendo el problema dual llegamos a:

$$\begin{bmatrix} 0 & 1 \\ 1 & MM^T + \frac{1}{\gamma}I \end{bmatrix} \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ Y \end{bmatrix} \tag{2.11}$$

$$\theta = M^T\vec{\alpha} \tag{2.12}$$

Donde $M = \begin{bmatrix} \varphi^T(x_1) \\ \varphi^T(x_2) \\ \vdots \\ \varphi^T(x_N) \end{bmatrix}$ y φ es una función de las variables entradas que nos permitirá expandir las dimensiones de nuestro problema. Además:

$$\hat{y} = \theta^T\varphi(x) + b \tag{2.13}$$

Ahora es cuando viene la magia de este método, y es que el producto MM^T está definido para determinadas funciones φ sin necesidad de moverse del espacio original, es decir, sin aumentar las dimensiones del problema. Estos productos reciben el nombre de kernels. Así, conociendo la forma de los kernels, somos capaces de crear separadores no lineales en el espacio original (aunque lineales en el espacio modificado), lo que da lugar a más libertad y usualmente a mejores resultados que métodos como el de los mínimos cuadrados ordinarios. Al igual que en el algoritmo de los mínimos cuadrados, es posible modificar el algoritmo de SVM para ponderar ciertos datos sobre otros, lo que daría lugar a un método de SVM ponderado.

2.3.4 Algoritmo k-means

Los métodos anteriores son ejemplos de aprendizaje supervisado, ya que utilizan el valor de la salida del problema como dato para aprender. El algoritmo k-means sin embargo no necesita esta información. Es un algoritmo simple y por ello y por sus buenos resultados es uno de los más empleados en problemas de aprendizaje no supervisado.

En su formulación más básica (ya que, como la mayoría de algoritmos que aquí se explican, se han desarrollado multitud de variantes del algoritmo original), se define al comienzo el número de clases que queremos que divida. Este número se puede estimar dependiendo del problema, aunque a veces es fijo como por ejemplo en el problema de dividir un conjunto de camisetas en tallas pequeña, mediana o grande. Una vez definido el número de clases, el algoritmo coloca arbitrariamente por el espacio de estados tantos centros como clases, y empieza a iterar hasta que estos centros representan realmente el centro de la clase a la que están asociados y no varían con las iteraciones.

El funcionamiento del algoritmo puede visualizarse en la figura 2.5. En cada iteración, el algoritmo clasifica los datos según el centro que esté más cercano a ellos, y posteriormente mueve los centros tal que se sitúen en el centro de los datos de cada clase. Las distancias pueden definirse de diversas formas, pero por su sencillez elegiremos la norma euclídea como medida de distancia entre dos puntos. El cálculo del centro de los datos de una misma clase es por tanto bastante intuitivo, ya que se sitúa en la media de las características de los datos de la clase. Algunas variantes del algoritmo sustituyen el valor de la media para el cálculo de los centros por el de la mediana, considerado un estadístico robusto, y elimina el posible malfuncionamiento del algoritmo debido a valores atípicos.

El algoritmo k-means no sirve únicamente para la clasificación, sino que también tiene otras funcionalidades como la reducción de ruido en medidas. Así, si se desea medir los valores que una variable adquiere en varios puntos, es típico repetir las medidas para mejorarlas. El algoritmo k-means es capaz de, dado el número de medidas diferentes que se han realizado, encontrar los centros de esas medidas, que equivaldrían a medidas más limpias.

2.3.5 Árboles de decisión

Los árboles de decisión suponen una rama del aprendizaje automático muy distinta a las anteriores que emplea conocimiento sobre árboles binarios y teoría de la información para resolver los problemas. El concepto de árbol binario es muy simple, pero a la vez muy potente. Todos estamos familiarizados con él y es la base de juegos como "¿Quién es quién?", el famoso juego infantil en el que la meta es encontrar un personaje particular haciendo preguntas de sí y no.

¿Pero qué tiene en común un juego de niños con una de las metodologías más empleadas del aprendizaje automático? Bueno, esta metodología aprovecha que las decisiones binarias son muy baratas computacionalmente. Además, al entrar en contacto con la teoría de la información, nos permite conocer factores como la cantidad de información que obtendremos conociendo el valor de una característica. Así, siempre eligiremos las características que más información nos aporten sobre el problema.

En la figura 2.6 se muestra un ejemplo simple de un árbol de decisión que aplica dos clasificaciones binarias para decidir el tipo de camiseta más adecuado en cada día.

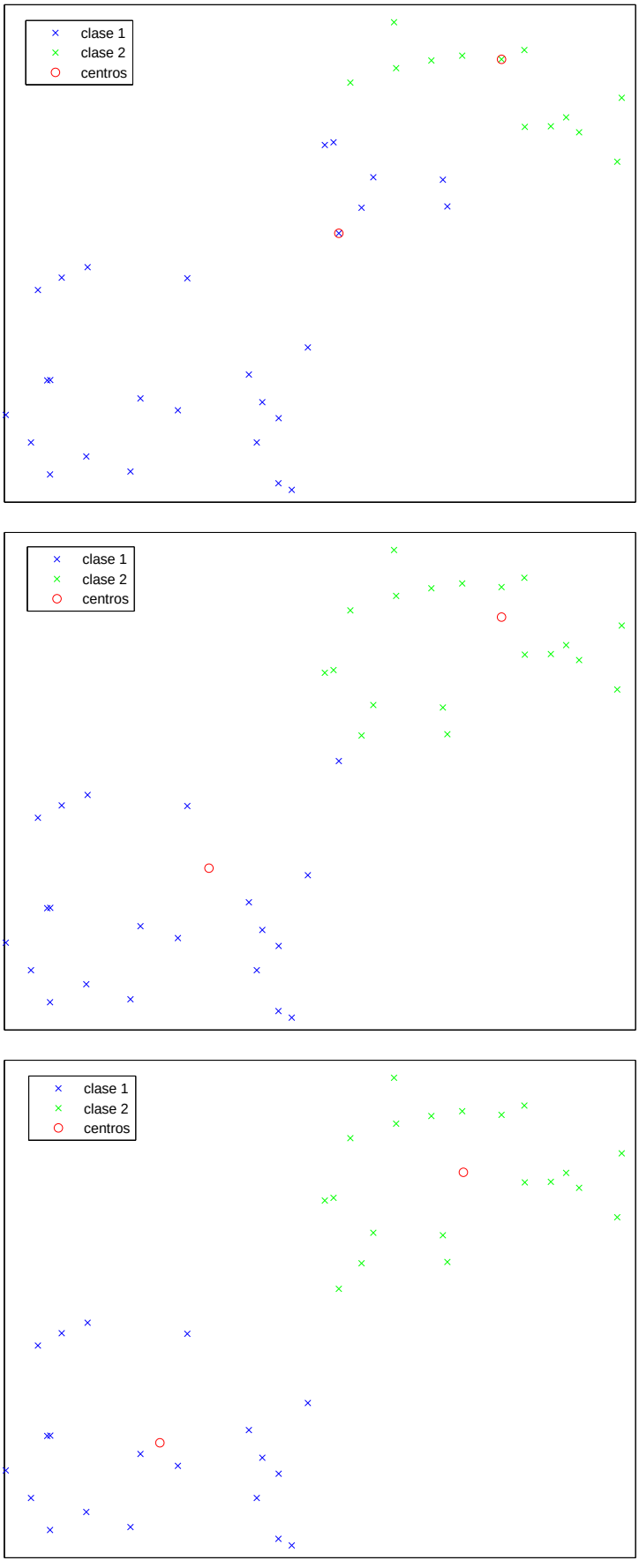

Figura 2.5 Iteraciones del algoritmo k-means.

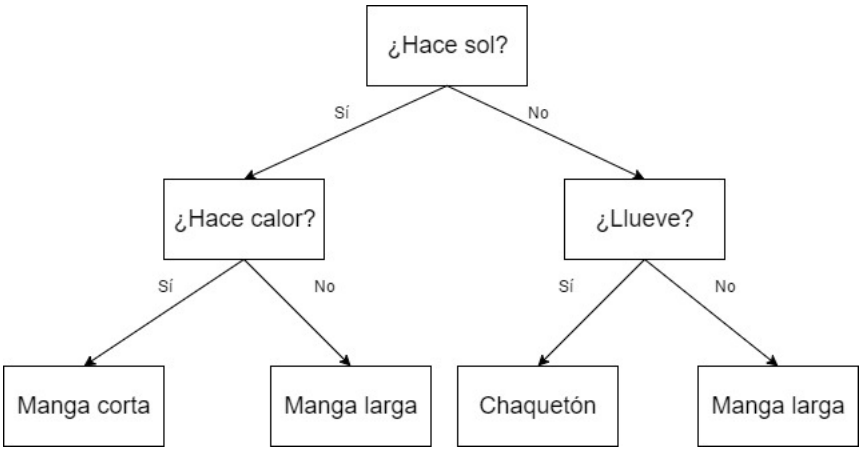

Figura 2.6 Ejemplo de un árbol de decisión clasificador.

2.3.6 Redes neuronales

Esta solución es más compleja y menos intuitiva que las anteriores. Parte del modelo de aprendizaje del cerebro humano y lo simplifica para conseguir ajustar gran variedad de funciones a nuestros datos. La idea es la de crear una red de neuronas consistente en varias capas como se muestra en la figura 2.7. La primera capa correspondería a las entradas de nuestro problema, mientras que la última lo haría a las salidas. La idea es la de conectar las neuronas (simbolizadas por circunferencias) de una capa con las de la siguiente, mediante una serie de nodos (pesos). Las neuronas de la segunda capa se activarán cuando la suma de todas las neuronas conectadas a ella multiplicadas por sus coeficientes supere un cierto umbral.

El proceso de aprendizaje aparece cuando se comparan los resultados obtenidos en la última capa con aquellos objetivos. Los algoritmos son capaces de realizar el proceso inverso al realizado por los datos y reajustar los coeficientes de las neuronas.

Entre las limitaciones de este tipo de metodologías encontramos la imposibilidad de crear separadores no lineales y la posible dificultad a la hora de diseñar la arquitectura de las redes, es decir, elegir cantidad de capas intermedias y de neuronas en cada capa.

2.4 Conclusiones

En este capítulo se ha estudiado la historia del aprendizaje automático, se han planteado los problemas típicos que se plantean, las limitaciones del campo y algunas de las metodologías tradicionales de resolución. Sin embargo, el machine learning es una ciencia de datos, todos los algoritmos necesitan de datos para aprender del problema. Es por ello que se va a emplear el capítulo siguiente a hablar de éstos, de cómo se tratan y de la forma de incluirlos en los algoritmos.

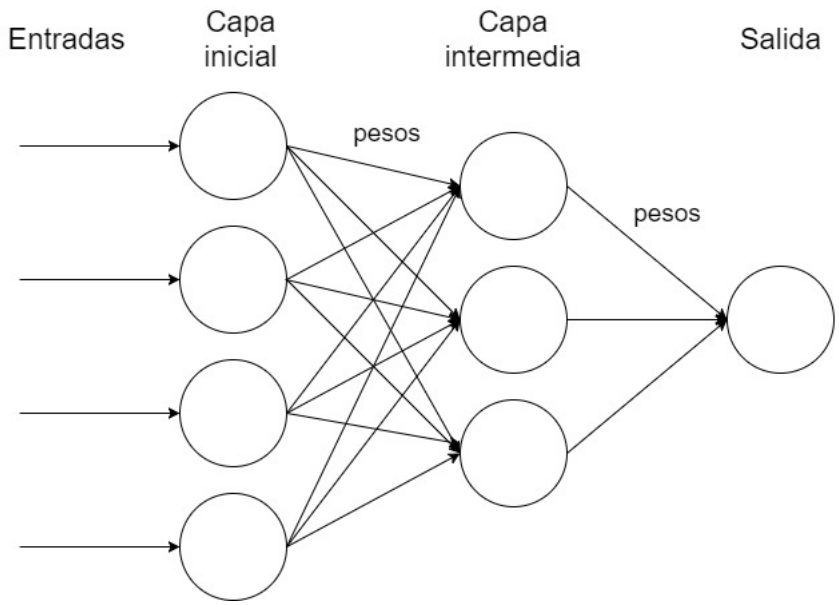

Figura 2.7 Esquema de una red neuronal.

3 Tratamiento previo de datos

Ya hemos estudiado los problemas y las soluciones tradicionales del aprendizaje automático, pero no hemos tratado uno de los aspectos fundamentales de éste: los datos. Como sabemos, el aprendizaje automático requiere de datos del problema para conseguir su resolución, es por ello que tenemos que ser capaces de conseguirlos y manipularlos de forma que consigamos extraer de ellos toda la información del problema y de la solución que nos ofrece el algoritmo. Para ello se explicarán los sets de datos seleccionados para evaluar los algoritmos, se propondrá un esquema de tratamiento de datos, se detallará un método de división de datos en sets de entrenamiento y finalmente se detallarán métodos de representación de los mismos que nos permitan comprobar que nuestros algoritmos realmente funcionan según lo previsto.

3.1 Problemas a tratar

En esta sección se describirán los cuatro sets de datos elegidos para la evaluación de los algoritmos, explicando los datos que contienen y detallando los problemas aplicables a cada uno de ellos.

3.1.1 Flor de iris

El set de datos de Iris es un problema de clasificación multiclase que fue empleado por primera vez por Fisher en 1936 en el el artículo *The Use of Multiple Measurements in Taxonomic Problems*. Incluye más de 50 muestras de cada flor en la que incluye las siguientes características:

1. Identificación de la muestra.
2. Longitud del sépalo en cm.
3. Amplitud del sépalo en cm.
4. Longitud del pétalo en cm.
5. Amplitud del pétalo en cm.
6. Especie correspondiente.

En función de estos parámetros, se distingue entre tres especies: Iris-setosa, Iris-versicolor e Iris-virginica. La primera clase es linealmente separable de las otras dos, mientras que las restantes no son linealmente separables entre ellas.

Es uno de los sets de datos más empleados en la literatura y en este estudio se empleará para mostrar la eficacia de diferentes algoritmos en la clasificación multiclase.

3.1.2 Concentraciones de proceso químico

Esta muestra fue ofrecida por George Box y Gwilym Jenkins en 1976 y ofrece un conjunto de 197 medidas de concentración de un proceso químico con un intervalo de separación de dos horas entre cada muestra. Se estudiará ya que es una muestra sencilla de un problema de predicción temporal que nos permitirá evaluar los algoritmos de SVM y de verosimilitud desarrollados.

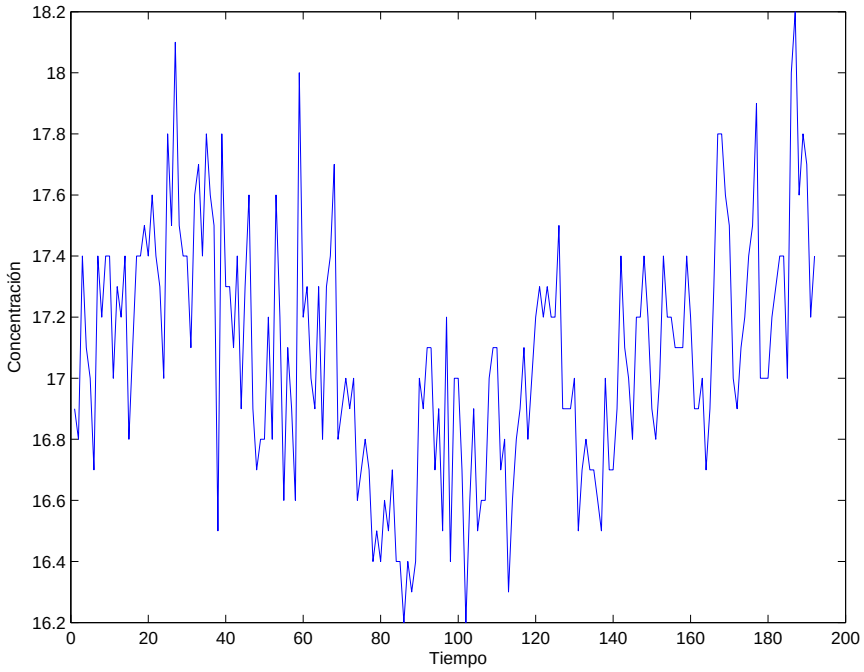

Figura 3.1 Concentraciones de proceso químico.

3.1.3 Plan de pensiones

A veces las empresas proporcionan datos relevantes en sus páginas web. Un ejemplo interesante y que se ha recogido en este trabajo es el caso de Bankia, que ofrece públicamente en su web la evolución del valor liquidativo de su plan de pensiones a través de los años. Exactamente, los datos recogen los valores desde el 19 de mayo del 2007 al mismo día de 2017 con una separación diaria entre cada dato. Con esto, se realizará una predicción temporal con un horizonte de predicción variable de la muestra de la figura 3.2. Esta muestra permitirá estudiar el desempeño de los algoritmos en muestras más complejas, ya que cuenta con un total de 3653 datos.

3.1.4 Imágenes varias

Para el caso de los algoritmos de aprendizaje no supervisado, ya que por su naturaleza es difícil cuantificar su éxito, se ha estimado más conveniente emplearlos para el tratamiento de imágenes. El objetivo es el de, alimentando al algoritmo con los valores de los tres campos de una imagen RGB, conseguir que el algoritmo separe los objetos del fondo de

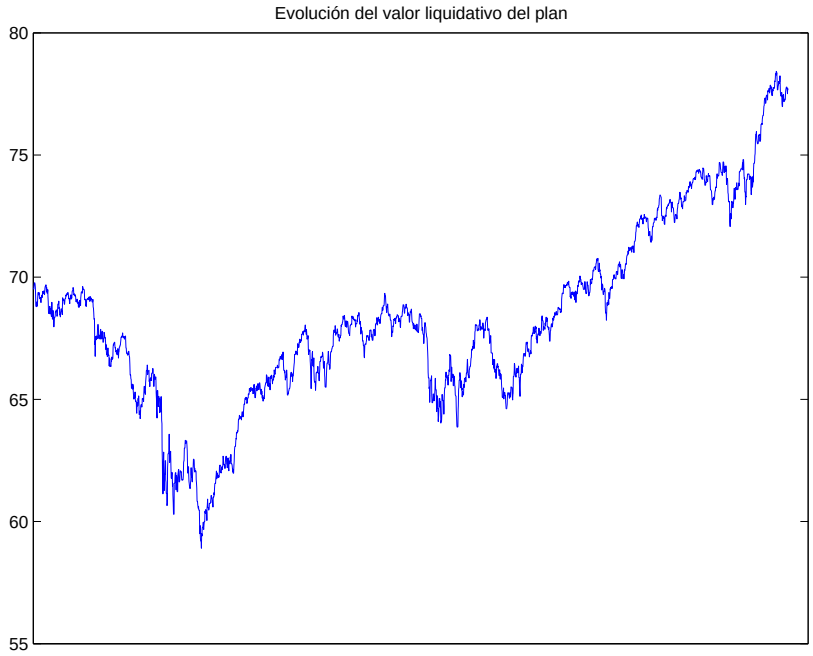

Figura 3.2 Evolución del valor liquidativo del plan.

las imágenes. Para ello se ha empleado una imagen de un triángulo enmarcado sobre un fondo blanco junto con otras imágenes incluídas en Matlab como 'moon.tif'y 'saturn.png', tal como se muestra en la figura 3.3.

El objetivo no es el de la separación perfecta de regiones, es por ello que no se terminará el tratamiento con otros métodos como el suavizado, ya que se distancian del objetivo de este trabajo. El fin es el de comprobar que los métodos de aprendizaje no supervisado son útiles en este campo.

Figura 3.3 Imágenes a tratar.

3.2 Tratamiento de datos

Para empezar a tratar los problemas, es necesario proveerse de un set de datos sobre el mismo, y tratarlo adecuadamente para que éstos se encuentren organizados de tal forma que nos sea sencillo trabajar con ellos. Para ello, llevamos a cabo las siguientes operaciones:

1. Adquisición de datos.
2. Cambio de formato.
3. Lectura de datos.
4. Ordenamiento de datos.
5. División en sets.
6. Normalización de datos.

3.2.1 Adquisición de datos

Lo primero que debemos hacer es adquirir los datos, cualquiera que sea el formato en el que estos se encuentren. Para ello, se puede recurrir a páginas web como *www.kaggle.com* donde encontramos normalmente datos para probar los algoritmos, o podemos solicitar los datos al organismo poseedor de estos. Estos datos se encontrarán usualmente en formatos no adecuados para empezar a trabajar con ellos, de ahí que sea necesario el siguiente punto.

3.2.2 Cambio de formato

Tras adquirir los datos, los pasaremos a un formato ordenado y adecuado antes de abrirlos desde el programa de tratamiento, que en nuestro caso será Matlab. Normalmente los datos originales están incompletos y tienen varias características que pueden no interesarnos, por lo que es conveniente un filtrado previo en este paso para evitar malos resultados por parte del algoritmo.

En el estudio, el formato empleado ha sido *.xlsx* por su limpieza y las facilidades que proporciona para ordenar los datos en tablas, que posteriormente serán interpretadas como matrices. El medio para trabajar con este formato ha sido Microsoft Excel 2016.

3.2.3 Lectura de datos

Una vez tenemos los datos en el formato adecuado, pasamos a leerlos en nuestro entorno de trabajo, que como ya se ha comentado, será Matlab. Asignamos los datos del fichero *.xlsx* a una variable mediante la orden: *xlsread(nombredelarchivo.xlsx)*.

Cuando tenemos el set de datos cargado en Matlab, será conveniente ordenarlo con las herramientas que nos proporciona este programa, como pueden ser vectores, matrices, estructuras o celdas. Esto puede incluir la creación de variables auxiliares que permitan una interpretación más adecuada de los datos y es necesario para simplificar todo el trabajo posterior.

Un ejemplo de esto es la organización de los datos en matrices, la creación de variables como índices que permitan identificar cada medida o el cálculo de distancias en función de las coordenadas iniciales y finales de un móvil.

3.2.4 Normalización de datos

Conocido en inglés como *feature scaling*. Esta técnica permite mejorar la eficacia de los algoritmos escalando todas las características en el mismo rango y evitando los valores atípicos o outliers. Es necesario para que el algoritmo reaccione de igual forma a todos los datos, pues si una característica está en una escala diferente al resto, típicamente aprenderá a distinta velocidad, lo que no suele ser conveniente.

Entre los métodos de normalización encontramos muchos dependiendo del problema. En este estudio hemos considerado principalmente dos métodos:

1. Eliminación de outliers. Los outliers son datos atípicos que entorpecen el algoritmo sesgando los datos. Hay diversas formas de evitarlos, pero en este trabajo se ha optado por saturar los datos por debajo del percentil 5 y por encima del percentil 95. Esto es computacionalmente muy barato y descarta los datos mayores y menores del algoritmo, cualquiera que sea su valor, asignándole un valor máximo y un valor mínimo definido en estos percentiles.

   El problema es que nuestro algoritmo no nos proporcionará buenos resultados cuando los datos se alejen mucho de los valores promedio, pero a cambio conseguimos un método rápido de evitar los outliers que mejorará la eficacia general del método.

2. Normalización 0-1. El segundo de los métodos empleados ha sido una normalización afín de los datos resultantes en el intervalo [0,1]. Así, aunque hay algoritmos como el de verosimilitud que estudiaremos que no son sensibles a transformaciones afines, hay otros que sí, y se comportan mejor en un intervalo [0,1] de lo que lo harían en intervalos diferentes.

3.3 División en sets de entrenamiento

Una vez tenemos los datos ordenados, podemos dividirlos en los diferentes sets de trabajo: entrenamiento, validación y ensayo. El objetivo es el de establecer los datos que le mostramos al algoritmo para que generalice, tal que elija los valores de parámetros más adecuados y finalmente verifique si el algoritmo funciona según lo deseado. El uso de estos sets se refleja en la figura 3.4. La idea es utilizar el set de entrenamiento para crear diferentes modelos del sistema variando los valores de los parámetros. Posteriormente, emplear el set de validación para elegir cual de ellos obtiene mejores resultados. El set de entrenamiento no es representativo en este aspecto ya que no permite comprobar la generalización del algoritmo a datos nuevos (que es con los que trabajará en la realidad). Por último, empleamos el set de datos de ensayo o test para calcular la eficacia del algoritmo.

Estos sets contienen usualmente datos diferentes, pues la idea es no sesgar los resultados utilizando para medir la eficacia los mismos datos con los que hemos entrenado el algoritmo. Los datos incluidos en los sets se suelen escoger aleatoriamente, pues es probable que inicialmente los datos estuvieran ordenados de alguna forma, y cada uno de los sets de entrenamiento no reflejase la totalidad del sistema, que es lo que nos interesa.

La cantidad de datos que incluiremos en cada conjunto no está definida, pero se suele adoptar un 60 % de datos para el set de entrenamiento, un 20 % para el set de validación y un 20 % para el de ensayo. En este estudio, se ha definido la función *splitdata()* para realizar esta división.

3.3.1 División en entrada-salida

La clasificación de los datos en entrada y salida es aún más importante que la clasificación anterior, ya que el algoritmo trabaja de forma diferente los datos de entrada que los de salida. Esta clasificación requiere conocer el problema y analizar qué datos pueden interesar en nuestro problema. A menudo se proporcionan muchos más datos de los necesarios para

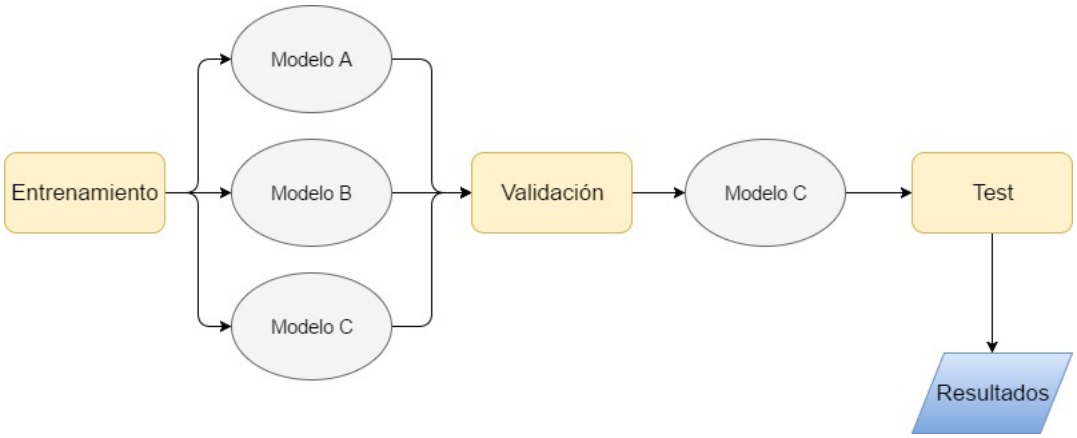

Figura 3.4 Empleo de los sets de entrenamiento.

resolver un problema. Estos datos, además de ser innecesarios, suponen una pérdida de eficacia del algoritmo, llegando incluso a hacerlo inútil, por lo que el previo análisis de los datos de entrada y de salida será esencial.

3.4 Representación de datos

Tan importante es conseguir que el algoritmo de aprendizaje aprenda de los datos como visualizar que realmente lo está haciendo de la forma indicada. Es por ello que se presentan a continuación dos técnicas que nos permiten comprobar el correcto funcionamiento de los algoritmos para los dos problemas principales: la regresión y la clasificación. Para el problema de regresión haremos uso de las curvas de aprendizaje, una herramienta potente que nos ayuda a visualizar el proceso de aprendizaje de nuestro algoritmo, mientras que para el problema de regresión emplearemos las matrices de confusión, que nos permitirá visualizar dónde se han cometido los fallos.

3.4.1 Curvas de aprendizaje

El objetivo de esta técnica será el de definir una forma de comprobar si nuestros algoritmos funcionan correctamente. Ya que el objetivo del trabajo es el desarrollo de diferentes algoritmos de aprendizaje automático, es muy útil analizar si los algoritmos que implementamos están funcionando bien o si, por el contrario, hay ciertos bugs o errores que impiden el correcto funcionamiento de estos, por lo que sería necesario pulirlos. Con este fin, se estudian las curvas de aprendizaje, conocidas como *Learning curves*. Estas curvas no son sino una representación de cómo evoluciona nuestro algoritmo conforme lo entrenamos con un número creciente de datos.

La figura 3.5 nos muestra la curva ideal, en la que podemos apreciar como, a medida que introducimos más datos, el algoritmo empieza a no ajustarse a todos los datos de entrenamiento y el error en este set aumenta, mientras que para el set de validación o prueba, el error disminuye, pues el algoritmo evoluciona para procesar datos nuevos. Este sería el funcionamiento ideal, pero también observamos otros tipos de comportamientos.

La figura 3.6 muestra una curva con sobreajuste, en la que el error en el set de entrenamiento se mantiene muy bajo. Esto se produce por la presencia de un numero excesivo de parámetros y/o por la ausencia de un término de regulación, o por la presencia de uno

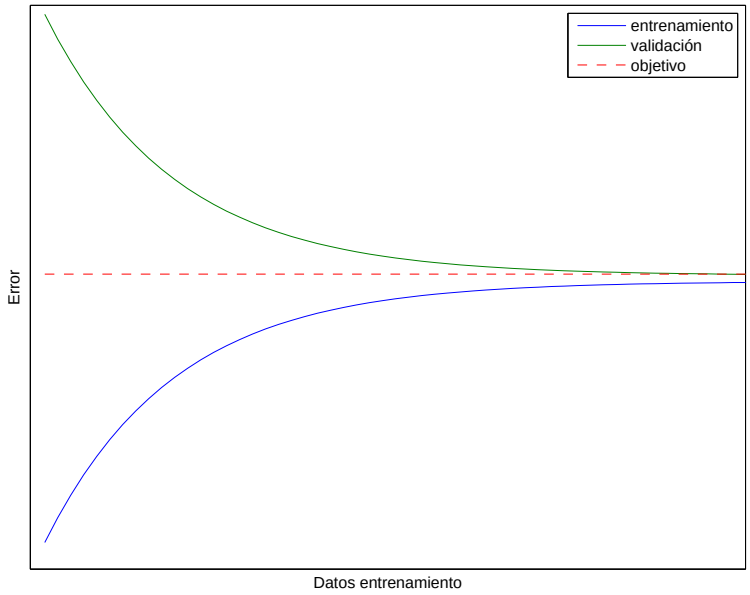

Figura 3.5 Curva ideal.

despreciable. En este caso, el error en el set de validación o prueba se mantiene alto, como consecuencia de que el algoritmo no ha generalizado, sino que ha sobreajustado. Este tipo de gráficas se arreglan o bien reduciendo el número de parámetros o bien aumentando el término de regulación. También se puede comprobar que aumentar el número de datos de entrenamiento ayuda a mejorar el rendimiento del algoritmo, pero muchas veces el número de datos es limitado o costoso y no supone una opción viable.

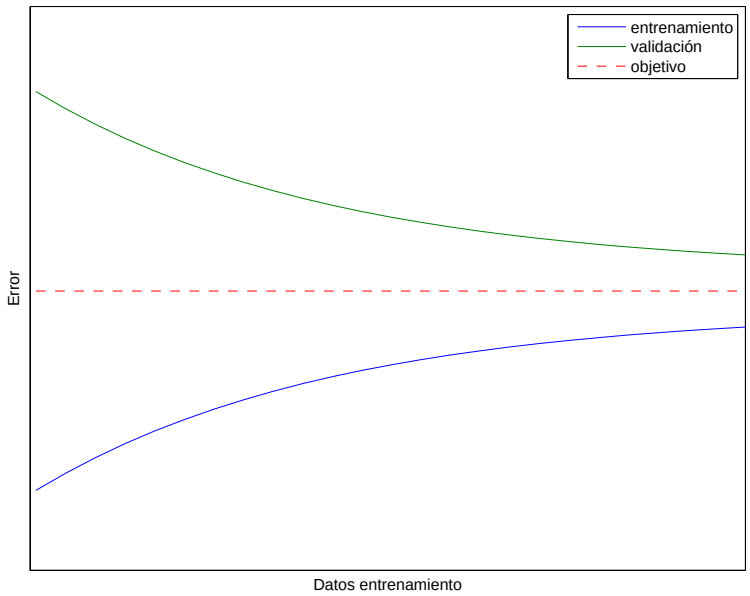

Figura 3.6 Curva con sobreajuste.

La imagen 3.7 muestra una curva con falta de ajuste, en la que el error en el set de entrenamiento se mantiene por encima de lo esperado, mientras que el error de ensayo se mantiene muy próximo a éste, sin estar tampoco cerca del rendimiento esperado. Esto

se produce por la generalización excesiva debido a un término de regulación elevado o bien a un número de parámetros bajo e inadecuado. Este tipo de gráficas se arreglan o bien aumentando el número de parámetros o bien disminuyendo el término de regulación. Recordamos en este punto que una forma de aumentar el número de parámetros cuando no se dispone de otras características del problema, es la de crear combinaciones de las ya existentes, siguiendo la misma filosofía que en las máquinas de vectores soporte.

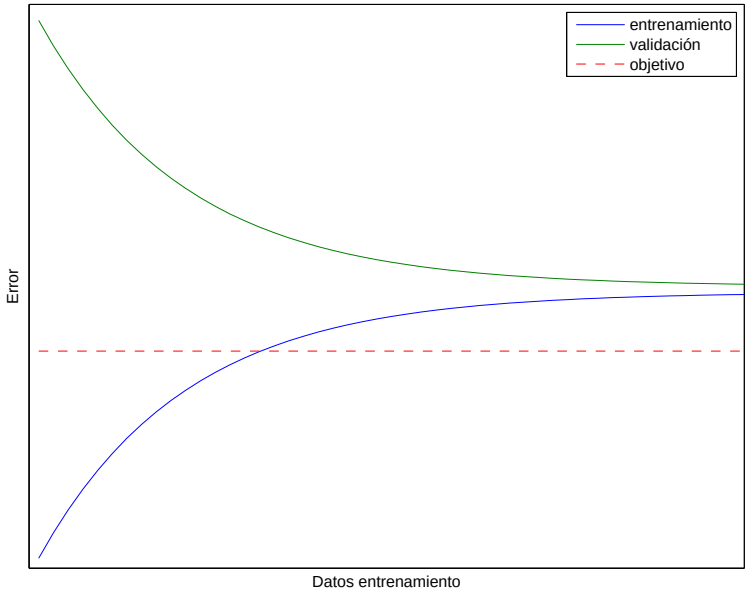

Figura 3.7 Curva con falta de ajuste.

Como hemos visto, el empleo de estas gráficas es muy beneficioso para el desarrollo y ajuste de algoritmos de aprendizaje automático, permitiendo visualizar la evolución del aprendizaje y permitiendo el ajuste de diversos parámetros relativos a él. Se ha implementado la función *learningcurves().m* que permite visualizar estas curvas para cualquier método de los diseñados.

3.4.2 Matrices de confusión

La matriz de confusión o confusion matrix, es un método que nos permite visualizar en una tabla dónde se clasifican los datos y dónde se deberían clasificar. La idea es simple: se crea una matriz como la mostrada en la tabla 3.1 en la que a lo largo de una dimensión se colocan las clases objetivo (donde deberíamos clasificar los datos), mientras que en la otra dirección se colocan las clases predichas por el algoritmo de clasificación. Así, si un dato de clase 1 lo predecimos correctamente en la clase 1, el contador de la matriz de confusión en la posición (1,1) se incrementaría en una unidad. La suma resultante de todos los valores de la matriz se corresponde por tanto al número total de clasificaciones realizadas. Es fácil comprobar que la matriz de confusión será siempre una matriz cuadrada, y que los elementos de su diagonal se corresponderán a clasificaciones correctas.

Pero esta técnica va más allá de la visualización en tabla de los resultados, ya que nos permite definir unas medidas de acierto del algoritmo. Para ello, definimos tablas biclase para cada clase. Así definimos #TP (del inglés True Positives) aquellos datos que se hayan clasificado correctamente en la clase 1, #TN (del inglés True Negatives)

Tabla 3.1 Esquema de matriz de confusión.

		Clasificaciones	
		C1	C2
Realidad	C1	#TP	#FP
	C2	#FN	#TN

aquellos que se hayan clasificado correctamente en la clase 0. Por último, los datos que no se hayan clasificado correctamente los denominaremos #FN (False Negatives) si se han clasificado en la clase 0 y #FP (False Positives) si se han clasificado en la clase 1. Con estos cuatro parámetros se definen medidas auxiliares y representativas del problema como la sensibilidad, especificidad, precisión y exhaustividad de la forma

$$\text{Sensibilidad} = \frac{\#TP}{\#TP + \#FN} \tag{3.1}$$

$$\text{Especificidad} = \frac{\#TN}{\#TN + \#FP} \tag{3.2}$$

$$\text{Precisión} = \frac{\#TP}{\#TP + \#FP} \tag{3.3}$$

$$\text{Exhaustividad} = \frac{\#TP}{\#TP + \#FN} \tag{3.4}$$

También podemos calcular el acierto del método, comparando el número de aciertos con el número de fallos en la clasificación. En los términos recién definidos la podemos expresar de la forma

$$\text{Acierto} = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \tag{3.5}$$

Pero con las medidas auxiliares definidas somos capaces de conseguir más información que con la medida del acierto, que únicamente separa los datos correctamente clasificados de los que no. De ahí surge la medida F1, que se puede expresar de la siguiente forma

$$F_1 = 2\frac{\text{precisión} \times \text{exhaustividad}}{\text{precisión} + \text{exhaustividad}} \tag{3.6}$$

Las medidas anteriores consideran que el set de datos está equilibrado, algo que no ocurre en muchos de los casos, especialmente en problemas como el de detección de anomalías. En estos casos, predecir siempre la salida más probable podría darnos buenos resultados de exactitud, sin embargo fallaría siempre que se intenta predecir la más improbable, por lo que el clasificador sería nefasto. Esto se puede solucionar calculando el Coeficiente de Correlación de Matthew, conocido por sus siglas inglesas MCC

$$MCC = \frac{\#TP \times \#TN - \#FP \times \#FN}{\sqrt{(\#TP + \#FP)(\#TP + \#FN)(\#TN + \#FP)(\#TN + FN)}} \tag{3.7}$$

Si el denominador fuese nulo, entonces se sustituye por un 1.

3.5 El coeficiente de correlación

Ya se han estudiado diferentes sets de datos, cómo obtenerlos y modificarlos de la forma adecuada para poder introducirlos en los diferentes algoritmos. Pero hay un factor importante el cual se tratará en este apartado, y es la información que nos proporciona un determinado dato para la predicción que llevamos a cabo. Una forma de medir esto es el coeficiente de correlación. Dadas las medidas i y j como ω_i y ω_j respectivamente, y sus medias como μ_i y μ_j. Se define el coeficiente de correlación $\rho_{i,j}$ entre i y j se define de la forma:

$$\rho_{i,j} = \frac{E\{(\omega_i - \mu_i)(\omega_j - \mu_j)\}}{\sqrt{E\{(\omega_i - \mu_i)^2\}E\{(\omega_j - \mu_j)^2\}}} \tag{3.8}$$

Cuando se trata un problema de predicción temporal y_k, es interesante estudiar el comportamiento del coeficiente de correlación entre muestras separadas por una constante de lag τ. Teniendo en cuenta que la muestra i y la muestra j es la misma pero retrasada τ muestras, la media de ambas será igual. Así, el coeficiente de correlación se define de la siguiente forma:

$$\rho_y(\tau) = \frac{E\{(y_k - \mu_y)(y_{k-\tau} - \mu_y)\}}{\sqrt{E\{(y_k - \mu_y)^2\}E\{(y_{k-\tau} - \mu_y)^2\}}} \tag{3.9}$$

Si $\rho_y(\tau)$ fuera cercano a 1, podríamos predecir con un error pequeño el valor de la variable y_k dado el valor de la variable $y_{k-\tau}$. En el caso de $\tau = 1$ y que el coeficiente $\rho_y(1)$ fuera muy próximo a 1, el valor de la última muestra sería suficiente para dar una buena aproximación de la siguiente. En estos casos, aproximadores como el mantenedor de orden cero (MOC) son muy interesantes, ya que suponen la solución más simple al problema. Soluciones como el MOC o el MOC multiplicado por una cierta constante α pueden llegar en casos como este a proporcionar mejores resultados que métodos más complejos como los mínimos cuadrados ordinarios, o métodos basados en máquinas de vectores soporte.

Por último, también podemos representar el factor de correlación de las variaciones $\Delta\omega_k$ para estudiar cómo afectan estas variaciones al dato a predecir.

3.6 Conclusiones

Ya se ha puesto en contexto el aprendizaje automático, se ha hablado de los problemas principales y de los algoritmos principales para su resolución. Se ha explicado asimismo cómo se tratan los datos para alimentar correctamente a los distintos algoritmos y cómo, a veces, un algoritmo tan simple como el MOC puede obtener buenos resultados, llegando incluso a mejorar aquellos de algoritmos más complejos. Una vez explicado todo esto, ya es posible definir (en el siguiente capítulo) la familia de funciones de verosimilitud objetivo de este estudio.

4 Función de verosimilitud

En este capítulo se definirá una familia de funciones que nos permita calcular la similitud (o disimilitud en nuestro caso) de un dato respecto a un conjunto previamente definido (que serán los datos de entrenamiento de los algoritmos). Posteriormente se estudiará la utilidad de esta función para resolver muchos de los principales problemas del aprendizaje automático.

La metodología fue propuesta por Teodoro Álamo et al. en 2016 y este trabajo supone un estudio y resumen de sus publicaciones, unido a la implementación en el software Matlab de los diversos algoritmos necesarios para la resolución de los algunos de los problemas del aprendizaje automático actual.

4.1 Definición matemática

Dado un set de datos D y un vector x, estamos interesados en encontrar una función J_d indicativa de la disimilitud entre x y el set de datos D. Funciones de similitud entre ambos conjuntos serían fácilmente deducibles de la función de disimilitud mediante una relación como $J_s = exp(-cJ_d)$, que es capaz de transformar una la función de disimilitud $J_d \to [0,\infty)$ en la función de similitud $J_s \to (0,1]$ mediante el parámetro $c \geq 0$.

Hay muchas funciones que pueden servir como funciones de similitud, entre ellas encontramos por ejemplo la suma de las normas del vector x respecto a cada uno de los elementos de D. Aunque muchas de estas funciones sean útiles y empleadas con éxito en muchos casos, los problemas más avanzados requieren otro tipo de funciones. En este capítulo se propondrá una familia de funciones de similitud definidas por la optimización de un problema convexo.

Dado un set de entrenamiento $D = \{z_1, \cdots, z_N\}$ y un escalar $\gamma \geq 0$ que servirá como parámetro de compromiso, podemos definir la función de disimilitud J_γ de la forma:

$$\begin{aligned} J_\gamma(z) \quad &= \quad \min_{\lambda_1,\cdots,\lambda_N} \sum_{i=1}^{N} \lambda_i^2 + \gamma \sum_{i=1}^{N} |\lambda_i| \\ s.t. \quad & z = \sum_{i=1}^{N} \lambda_i z_i \\ & 1 = \sum_{i=1}^{N} \lambda_i \end{aligned} \tag{4.1}$$

Al igual que en las ecuaciones de OLS o SVM, es posible introducir un término que permita ponderar ciertas medidas sobre otras, de la forma:

$$\sum_{i=1}^{N} \tau_i \lambda_i^2 + \gamma \sum_{i=1}^{N} |\lambda_i| \tag{4.2}$$

τ_i podría tomar el valor de una función de similitud entre z y z_i, de forma que se incorporaría información local a la función, lo que puede resultar de gran interés cuando el set de datos no es lineal. Este caso no se ha tratado ya que distrae de la tarea principal, que es la demostración de que la familia de funciones propuesta está bien diseñada y cumple su objetivo correctamente.

Asimismo, se puede demostrar que la función propuesta es invariante respecto a transformaciones afines, lo que evita la necesidad de normalizar los datos con la que se alimenta, aunque sigue siendo sensible a datos atípicos o outliers.

Por último, se demostrará cómo, para $\gamma = 0$, la función de verosimilitud resultante es la curva normal.

Denotamos $\lambda = [\lambda_1, \cdots, \lambda_N]^T$, $u = [1, \cdots, 1]^T$ y $Z = [z_!, \cdots, z_N]^T$. Por tanto, las restricciones al problema original quedan de la forma:

$$\begin{aligned} z &= Z\lambda \\ 1 &= u^T \lambda \end{aligned} \tag{4.3}$$

El problema dual para el caso de $\gamma = 0$ queda por lo tanto de la forma:

$$\mathscr{L}(\lambda, \mu, \nu) = \lambda^T \lambda + \mu^T (Z\lambda - z) + \nu(u^T \lambda - 1) \tag{4.4}$$

Para obtener los valores óptimos de la variable primal y las duales, igualamos la función lagrangiana anterior respecto a la variable que nos interese. En este caso, nos interesa el valor óptimo de la variable primal λ, ya que nos permitirá calcular el valor de la función de disimilitud J_0.

$$\frac{\partial \mathscr{L}(\lambda^*, \mu^*, \nu^*)}{\partial \lambda} = 0 \Leftrightarrow \lambda^* = -\frac{1}{2}(Z^T \mu^* + u^* \nu) \tag{4.5}$$

Para eliminar de la ecuación las variables duales, premultiplicamos ambos términos por u^T, y sabiendo que $u^T \lambda = 1$, que $Zu = N\bar{z}$ y que $u^T u = N$ llegamos a:

$$\begin{aligned} 1 &= -\frac{N}{2}(\bar{z}^T \mu^* + \nu^*) \\ \nu^* &= -\frac{2}{N} - \bar{z}^T \nu^* \end{aligned}$$

Sustituyendo la expresión de ν^* en 4.5 obtenemos:

$$\begin{aligned} \lambda^* &= \frac{1}{2}\Big(Z^T \mu^* - u\big(\frac{2}{N} + \bar{z}^T \mu^*\big)\Big) \\ &= \frac{u}{N} - \frac{1}{2}(Z^T - u\bar{z}^T)\mu^* \end{aligned} \tag{4.6}$$

Premultiplicando por Z y teniendo en cuenta del problema original que $Z\lambda^* = z$

$$\begin{aligned} Z\lambda^* &= \bar{z} - \frac{1}{2}(ZZ^T - N\bar{z}\bar{z}^T)^{-1}(z-\bar{z}) \\ \mu^* &= -2(ZZ^T - N\bar{z}\bar{z}^T)^{-1}(z-\bar{z}) \end{aligned} \tag{4.7}$$

De nuevo, sustituimos la expresión de μ^* en 4.5 y obtenemos el valor de λ^* independiente de las variables duales.

$$\lambda^* = \frac{u}{N} + (Z^T - u\bar{z})(ZZ^T - N\bar{z}\bar{z}^T)^{-1}(z-\bar{z}) \tag{4.8}$$

Teniendo en cuenta que

$$u^T(Z^T - u\bar{z}^T)^T(Z^T - u\bar{z}^T) = ZZ^T - N\bar{z}\bar{z}^T$$

obtenemos que

$$(Z^T - u\bar{z}^T)^T(Z^T - u\bar{z}^T) = ZZ^T - N\bar{z}\bar{z}^T$$

Por último, de la última igualdad y de 4.5 obtenemos el valor de la función de disimilitud para $\gamma = 0$

$$\begin{aligned} J_0(z,D) &= (\lambda^*)^T\lambda^* \\ &= \frac{1}{N} + (z-\bar{z})^T(ZZ^T - N\bar{z}\bar{z}^T)^{-1}(z-\bar{z}) \end{aligned} \tag{4.9}$$

La ecuación 4.9 representa el exponente de la curva normal con signo cambiado (algo que, mediante la transformación $J_s = exp(-c * J_d)$ previamente comentada quedaría igual a la curva normal). Por lo que se ha demostrado que la función de disimilitud es equivalente a dicha curva para el caso $\gamma = 0$. Pero además de conseguir ajustar la función normal a los datos, podemos variar γ para ampliar el número de curvas que podemos aproximar a la función, por lo que se induce que la familia propuesta es más rica que la familia de curvas normales.

4.2 Detección de anomalías

La formulacion anterior permite definir una familia de funciones de disimilitud, pero ahora hay que estudiar cómo se aplican a los problemas principales de aprendizaje automático. Por su simplicidad, empezaremos con el problema de detección de anomalías.

El objetivo de este problema es el de destacar los datos que tengan una disimilitud muy grande respecto a los demás. Para ello, empezamos por calcular la disimilitud de un dato con respecto al conjunto total de los datos. Respecto al umbral que separa un dato anómalo de uno normal, suele ser difuso. Una solución es la de ordenar los datos en función de su disimilitud respecto al conjunto de datos, esto permite poder estudiar los casos con mayor disimilitud primero (pues son los más probables de presentar anomalías). Podemos además de ordenar los datos, estimar que aquellos que entren en un percentil determinado de disimilitud pueden ser anómalos, o si el problema pide más exactitud, calcular un umbral de anomalía, tal que si la anomalía de un dato supera dicho umbral, el dato se considera anómalo.

Para todas estas alternativas, la familia de funciones de verosimilitud presenta diversas soluciones en función del valor de γ. Así, conocida la distribución del set de datos que estemos analizando, o conocido un histórico de anomalías, es posible ajustar γ para que la curva se adapte mejor a la distribución real de los datos y la solución resultante mejore.

4.3 Regresión con funciones de disimilitud

Dadas unas entradas al algoritmo x_k, las estimaciones de las salidas y_k se pueden obtener resolviendo el problema original con las entradas

$$\begin{aligned} \min_{\lambda_1,\cdots,\lambda_N} \quad & \sum_{i=1}^{N} \lambda_i^2 + \gamma \sum_{i=1}^{N} |\lambda_i| \\ s.t. \quad & x = \sum_{i=1}^{N} \lambda_i x_i \\ & 1 = \sum_{i=1}^{N} \lambda_i \end{aligned} \tag{4.10}$$

para posteriormente emplear los valores óptimos de λ y predecir las salidas de la forma

$$\hat{y}_k = \sum_{i=1}^{N} \lambda_i^* y_i \tag{4.11}$$

El resultado de este método sería una combinación lineal de las salidas, algo consistente con la literatura.

Se puede demostrar que, para $\gamma = 0$, la solución aportada por el regresor es la misma que la de mínimos cuadrados (OLS), por lo que, de nuevo, el método demuestra su eficacia ya que ofrece una familia entera de soluciones diferentes (a una que ya funciona), variando el parámetro γ.

4.4 Predicción intervalar con funciones de disimilitud

Uno de los puntos más importantes del estudio de estas funciones de disimilitud es la posibilidad de mejorar el problema de la predicción intervalar.

Como se indicó, es posible transformar las funciones de disimilitud estudiadas en funciones de similitud mediante una transformación como: $J_s = exp(-c * J_d)$. Expresión que emplearemos en este apartado para definir la predicción intervalar.

A partir de las funciones definidas previamente, podemos definir una función de densidad de probabilidad condicional empírica de la forma

$$ecp(y,x_k) = \frac{exp\Big(-cJ_\gamma(\begin{bmatrix} y \\ x_k \end{bmatrix}, D)\Big)}{\int_y exp\Big(-cJ_\gamma(\begin{bmatrix} \hat{y} \\ x_k \end{bmatrix}, D)\Big) d\hat{y}} \tag{4.12}$$

Observamos, que por definición,

$$\int_y ecp(\hat{y},x_k)d\hat{y} = 1 \tag{4.13}$$

Aunque generalmente no podamos calcular de forma teórica el resultado de la integral $\int_y exp\Big(-cJ_\gamma(\begin{bmatrix}\hat{y}\\ x_k\end{bmatrix},D)\Big)d\hat{y}$, dados los parámetros $c \geq 0$ y $\gamma \geq 0$, sí podemos obtener una aproximación a ella mediante métodos númericos, como por ejemplo mediante la regla de Simpson 3/8, ya que $y \subset \mathbb{R}$.

Por tanto, dados x_k, γ y c, los valores de el cálculo del percentil α de la función $ecp(\cdot,\cdot)$ se puede realizar de la forma

$$\int_{y_{min}}^{\bar{y}\alpha} ecp(\hat{y},x_k)d\hat{y} = \alpha \tag{4.14}$$

Siendo y_{min} idealmente $-\infty$. Para resolver la integral por métodos como la regla de Simpson 3/8, este valor de y_{min} es inviable, es por ello que se recurre al hecho de que, por su naturaleza, podemos despreciar el valor de la integral de la función de $-\infty$ a y_{min}. Así, si conseguimos ajustar los parámetros c y γ tal que se ajusten a la distribución real, podemos calcular la probabilidad de que una predicción entre en nuestro intervalo con el β % de probabilidad de fallo mediante el cálculo de los percentiles $\beta/2$ y $1-\beta/2$, de la forma

$$\begin{aligned}\int_{y_{min}}^{\underline{y}_i} ecp(\hat{y},x_k)d\hat{y} &= \beta/2\\ \int_{y_{min}}^{\bar{y}_i} ecp(\hat{y},x_k)d\hat{y} &= 1-\beta/2\end{aligned} \tag{4.15}$$

El intervalo de confianza será por tanto $[\underline{y}_i,\bar{y}_i]$.

5 Resultados

En el capítulo se presentarán los resultados obtenidos al probar los algoritmos desarrollados en diferentes problemas. Los resultados están dispuestos de tal forma que no solo es posible comparar los algoritmos entre sí, sino que se puede visualizar el efecto del cambio de parámetros dentro del mismo algoritmo. Así, veremos en este capítulo cómo aumentar el término de regulación conlleva a curvas más suaves o cómo el aumento de la ganancia en el método del gradient descent puede llevar a desestabilizar el algoritmo y dar resultados erróneos. Junto a los resultados se incluirá asimismo un ejemplo de curvas de aprendizaje que nos permitirá comprobar que el algoritmo ha aprendido realmente según lo deseado. Los comentarios sobre los resultados se proporcionarán en el siguiente capítulo, mientras que este irá destinado a su ilustración.

Dentro de cada problema, se han elegido los algoritmos que lo resuelven que ofrecen mejores resultados. Así, se han ignorado algoritmos no pensados para la clasificación como el regresor de disimilitud o el algoritmo de mínimos cuadrados ordinarios en el problema de flor de iris.

El capítulo está ordenado por sets de datos, así, primero se estudiarán los resultados en el set de imágenes, posteriormente se verán los resultados sobre la muestra de concentraciones de un proceso químico, donde se estudiará el problema de predicción intervalar, entre otros. A continuación se seguirá con el problema de clasificación de flores y se terminará con la estimación de precios de casas. El valor de error proporcionado siempre será el del error cuadrático medio.

5.1 Imágenes

Se mostrarán a continuación las imágenes de partida a la izquierda, acompañadas por las imágenes tratadas por el algoritmo k-means a su derecha, siendo la entrada de este los tres canales (R,G,B) de la imagen original y buscando por salida una distinción de los objetos de las diferentes imágenes. El tratamiento proporciona únicamente una distinción entre objetos dentro de una imagen, para visualizarlo mejor, se ha optado por representar cada uno de los objetos en un tono de gris, así, ya que en todas las imágenes hay solo dos objetos, los tonos son blanco puro y negro puro.

Figura 5.1 Luna frente a fondo. K-means.

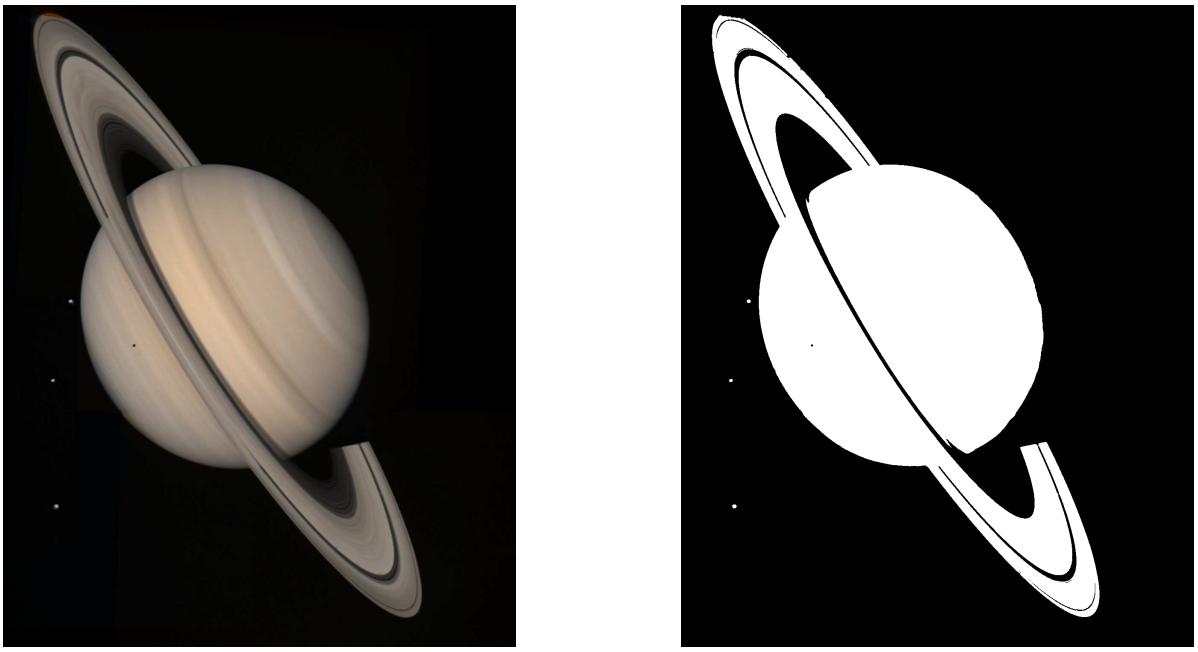

Figura 5.2 Saturno frente a fondo. K-means.

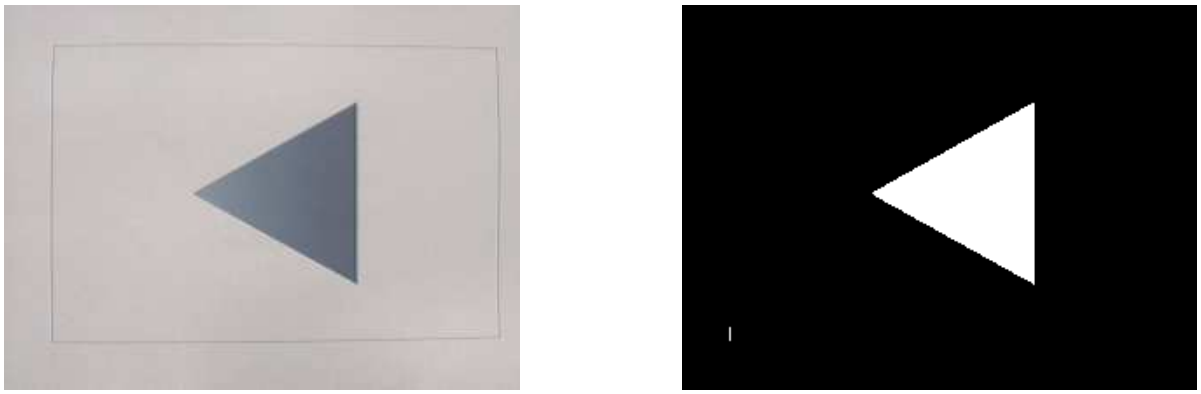

Figura 5.3 Triángulo frente a fondo. K-means.

5.2 Concentraciones de proceso químico

En esta muestra probaremos algoritmos de regresión, en particular las máquinas de vectores soporte (SVM), el algoritmo del gradient descent, los mínimos cuadrados ordinarios (OLS), el regresor de disimilitud y por último emplearemos el algoritmo de predicción de intervalos mediante funciones de verosimilitud para hallar un intervalo de confianza.

Puesto que el problema se reduce en hallar una salida a partir de sus valores pasados, los resultados serán muy visuales, ya que podremos representar en un plano la evolución de las salidas reales y predichas respecto al tiempo. Además de representar las predicciones de los diferentes algoritmos en la gráfica de concentraciones original, se proporcionarán los valores del error, tanto en el set de validación como en el de entrenamiento.

Todos los ensayos han sido alimentando a los diferentes algoritmos con las cinco salidas previas a la que debe predecir.

5.2.1 SVM

Para cada resultado se indicará tanto el kernel empleado como sus parámetros. Así, también se indicará el valor escogido para el parámetro de equilibrio γ.

Tabla 5.1 Resultados SVM para concentraciones químicas.

Kernel	Parámetros	Figuras	Error cuadrático medio Validación	Ensayo
Lineal	$\lambda = 1$ $\gamma = 1$	5.4 5.5	0.0478	0.1329
Polinomial	$\lambda = 1$ $\gamma = 1$ $c = 1$ $d = 2$	5.6 5.7	0.0499	0.1360
Polinomial	$\lambda = 1$ $\gamma = 1$ $c = 1$ $d = 4$	5.8 5.9	0.0460	0.2491
RBF	$\lambda = 1$ $\gamma = 100$ $\sigma = 0.5$	5.10 5.11	0.0767	0.1742
RBF	$\lambda = 1$ $\gamma = 100$ $\sigma = 1.5$	5.12 5.13	0.0678	0.1455
RBF	$\lambda = 1$ $\gamma = 100$ $\sigma = 5$	5.14 5.15	0.0502	0.1451

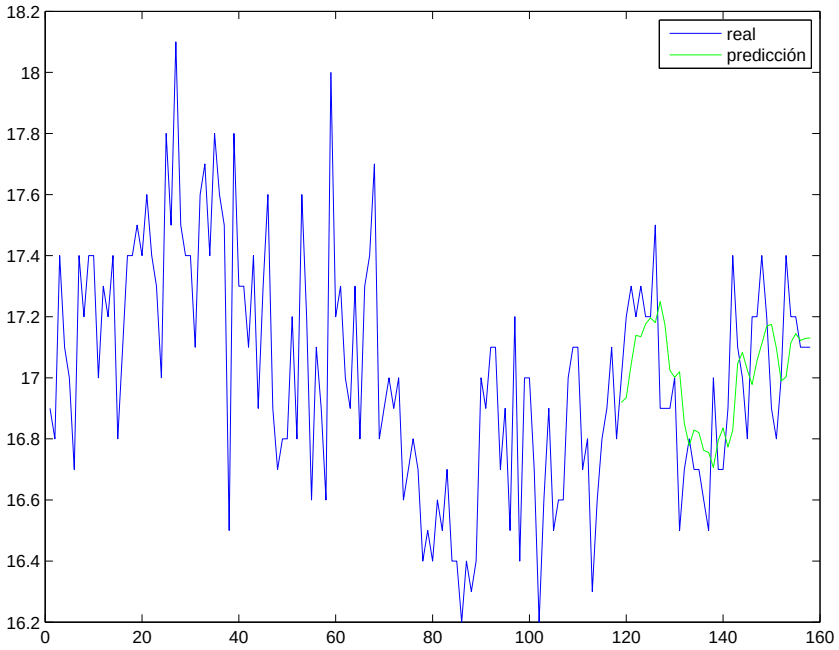

Figura 5.4 Kernel lineal. $\lambda = 1$. $\gamma = 1$. Set de validación.

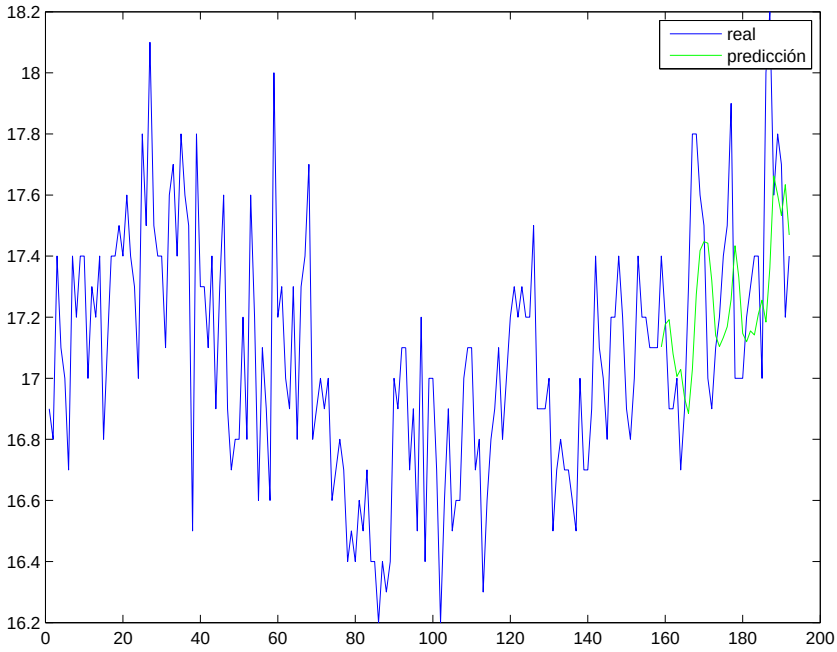

Figura 5.5 Kernel lineal. $\lambda = 1$. $\gamma = 1$. Set de ensayo.

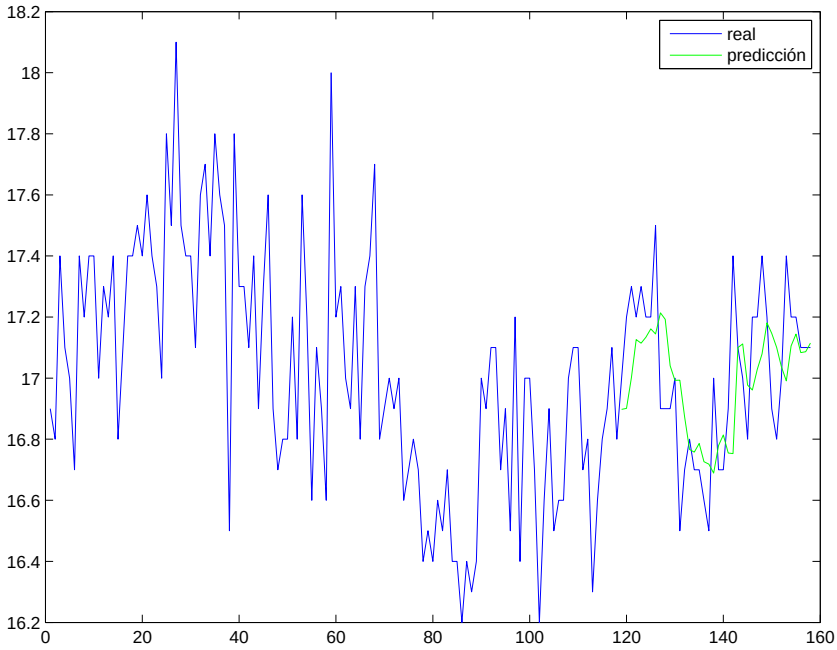

Figura 5.6 Kernel polinomial. $\lambda = 1$. c=1. d=2. $\gamma = 1$. Set de validación.

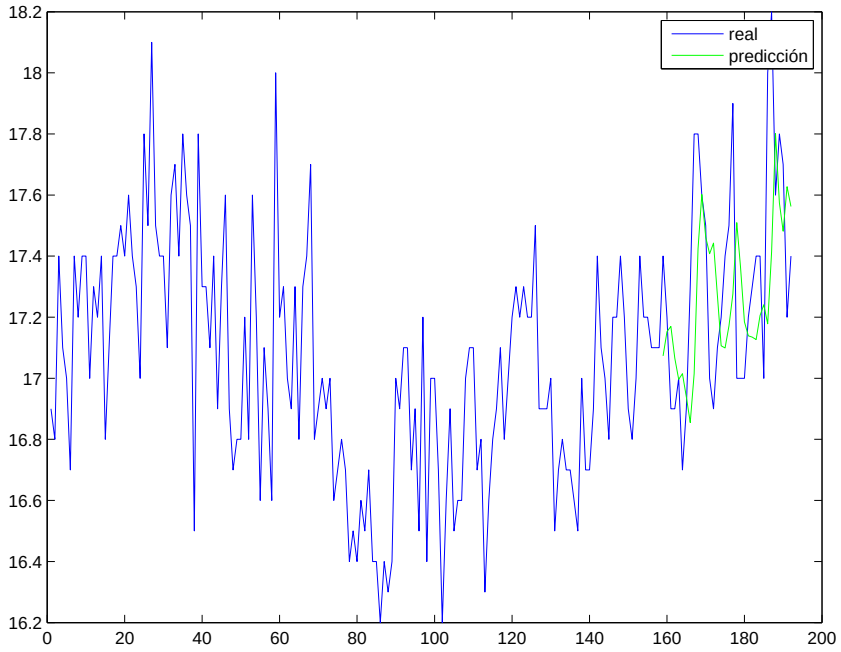

Figura 5.7 Kernel polinomial. $\lambda = 1$. c=1. d=2. $\gamma = 1$. Set de ensayo.

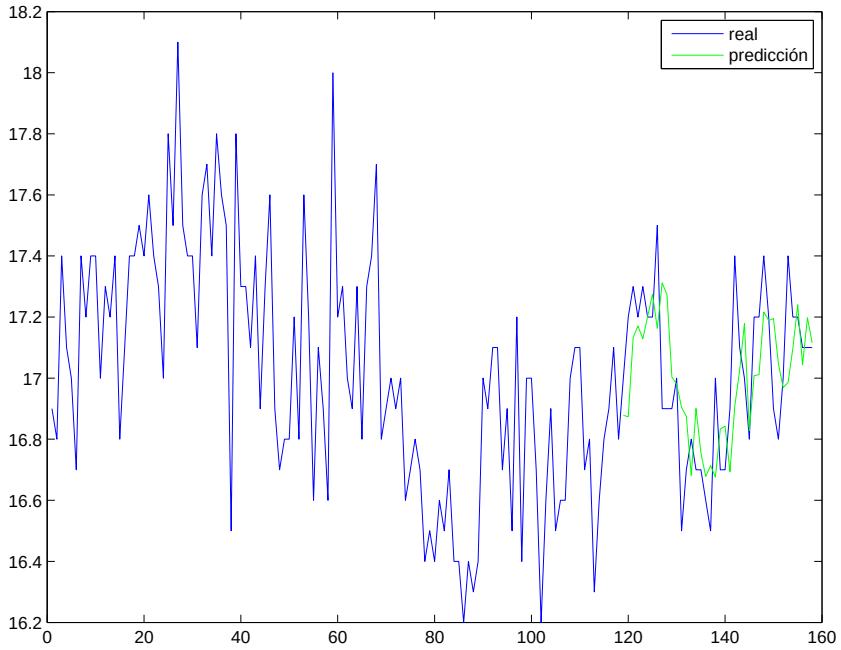

Figura 5.8 Kernel polinomial. $\lambda = 1$. c=1. d=4. $\gamma = 1$. Set de validación.

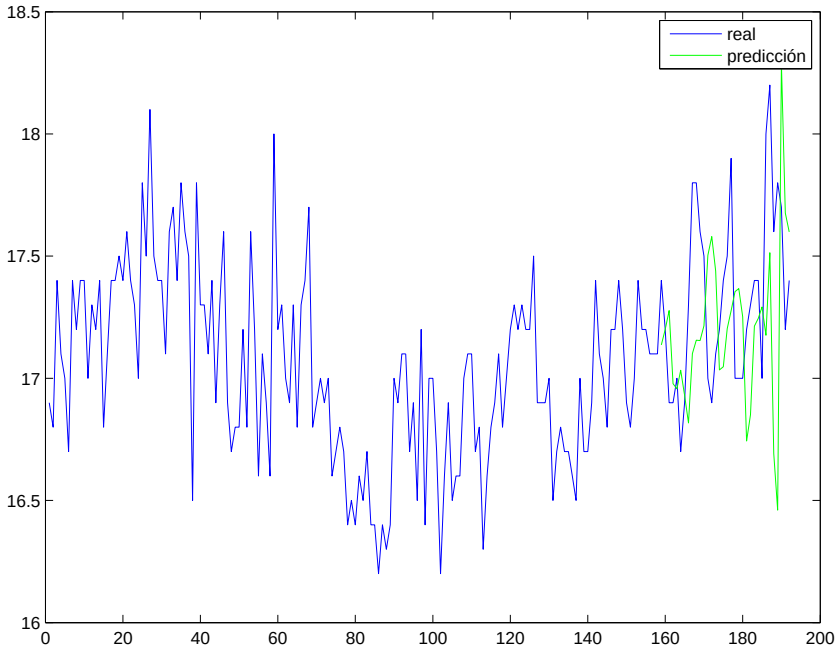

Figura 5.9 Kernel polinomial. $\lambda = 1$. c=1. d=4. $\gamma = 1$. Set de ensayo.

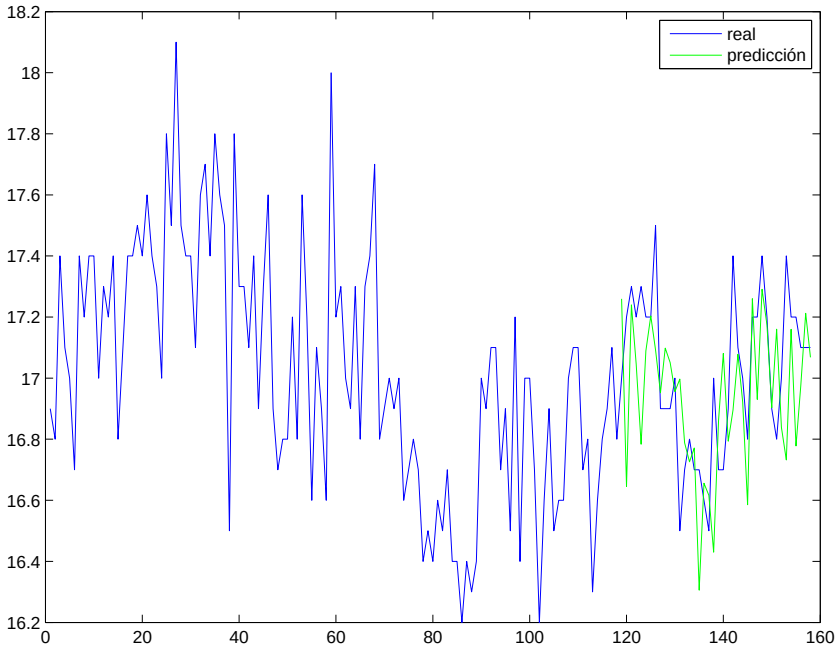

Figura 5.10 Kernel RBF. $\lambda = 1$. $\sigma = 0.5$. $\gamma = 100$. Set de validación.

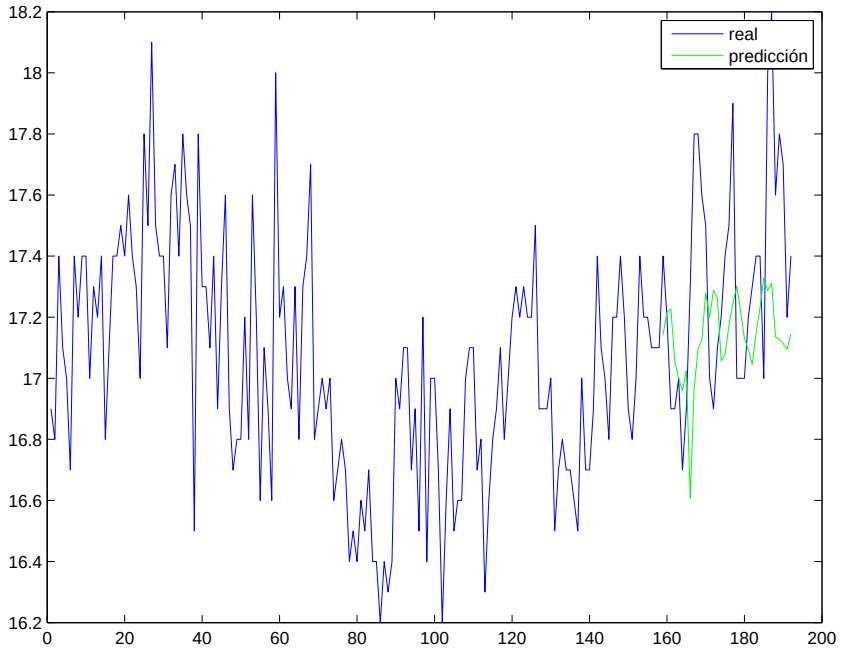

Figura 5.11 Kernel RBF. $\lambda = 1$. $\sigma = 0.5$. $\gamma = 100$. Set de ensayo.

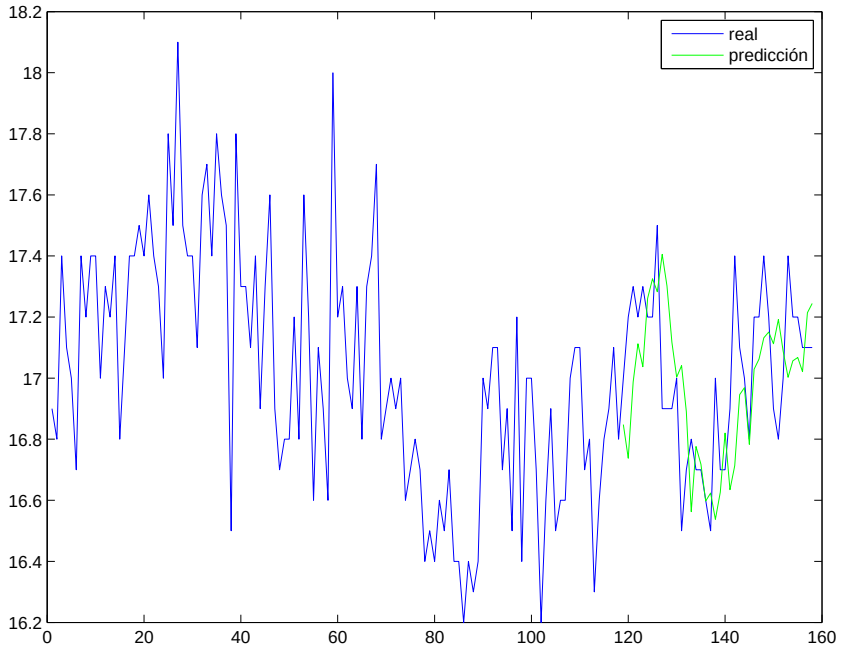

Figura 5.12 Kernel RBF. $\lambda = 1$. $\sigma = 1.5$. $\gamma = 100$. Set de validación.

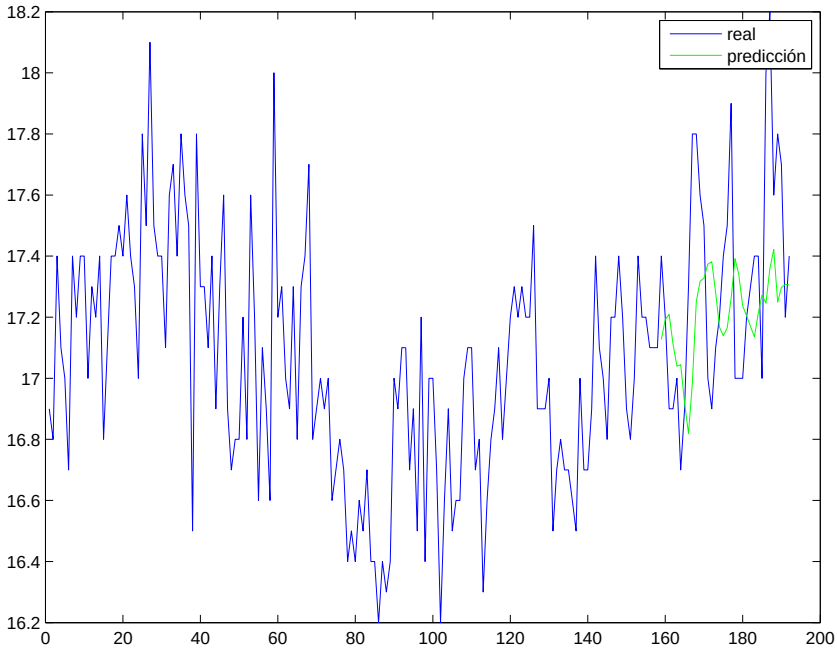

Figura 5.13 Kernel RBF. $\lambda = 1$. $\sigma = 1.5$. $\gamma = 100$. Set de ensayo.

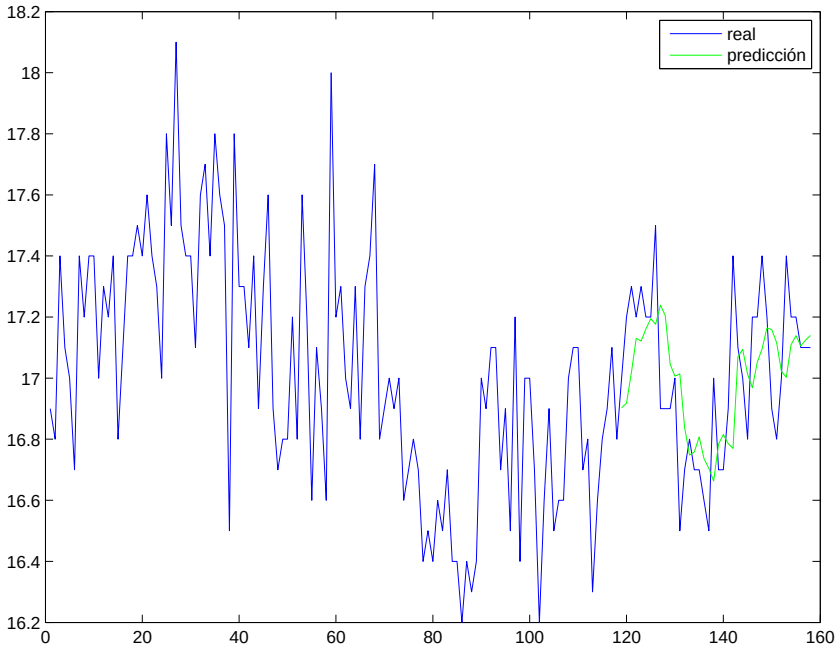

Figura 5.14 Kernel RBF. $\lambda = 1$. $\sigma = 5$. $\gamma = 100$. Set de validación.

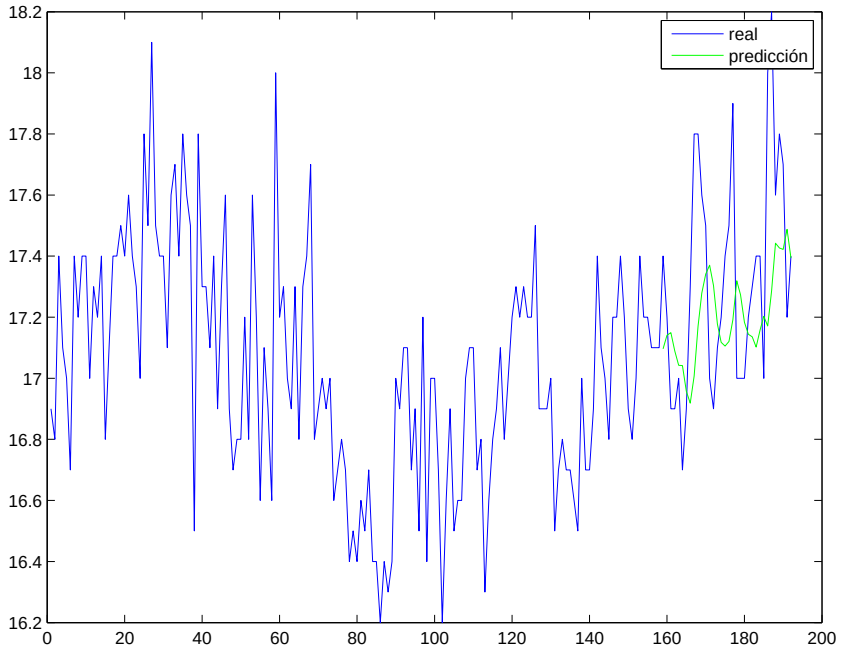

Figura 5.15 Kernel RBF. $\lambda = 1$. $\sigma = 5$. $\gamma = 100$. Set de ensayo.

5.2.2 OLS & Gradient descent

Se han aplicado dos regresiones de mínimos cuadrados ordinarios, variando el parámetro de regulación λ. En cuanto al gradient descent, se han aplicado unos valores de parámetros con un valor de alfa lo suficientemente bajo y un número de iteraciones lo suficientemente alto, para que converja al mínimo de la función de coste. Los valores son: $\alpha = 10^{-3}$ y 200 iteraciones.

Tabla 5.2 Resultados OLS para concentraciones químicas.

	Parámetros	Figuras	Error cuadrático medio Validación	Ensayo
OLS	$\lambda = 0$	5.38 5.39	0.0478	0.1329
OLS	$\lambda = 100$	5.40 5.41	0.0541	0.1784
Gradient descent	$\alpha = 10^{-3}$ Iteraciones $= 200$ $\lambda = 1$	5.20 5.21	0.0478	0.1329

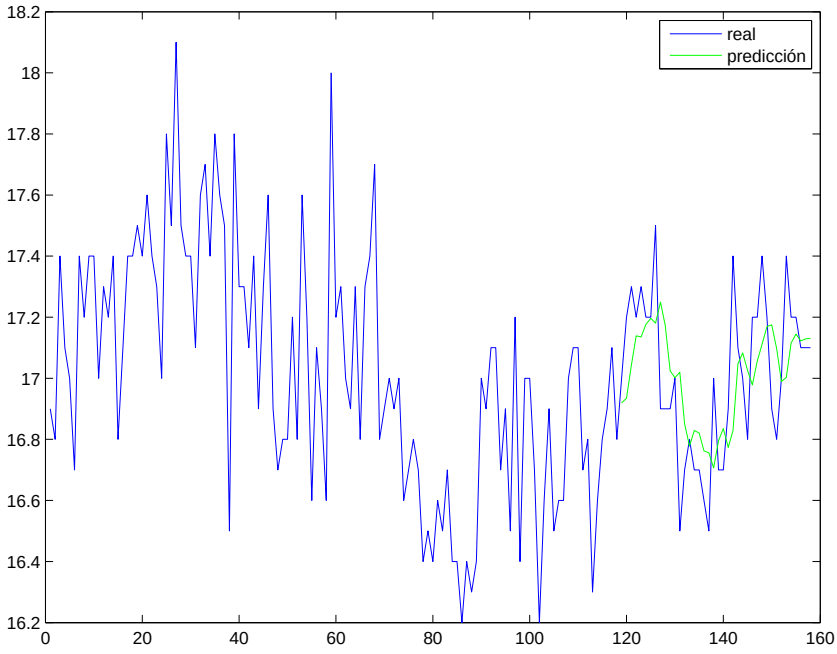

Figura 5.16 OLS. $\lambda = 0$. Set de validación.

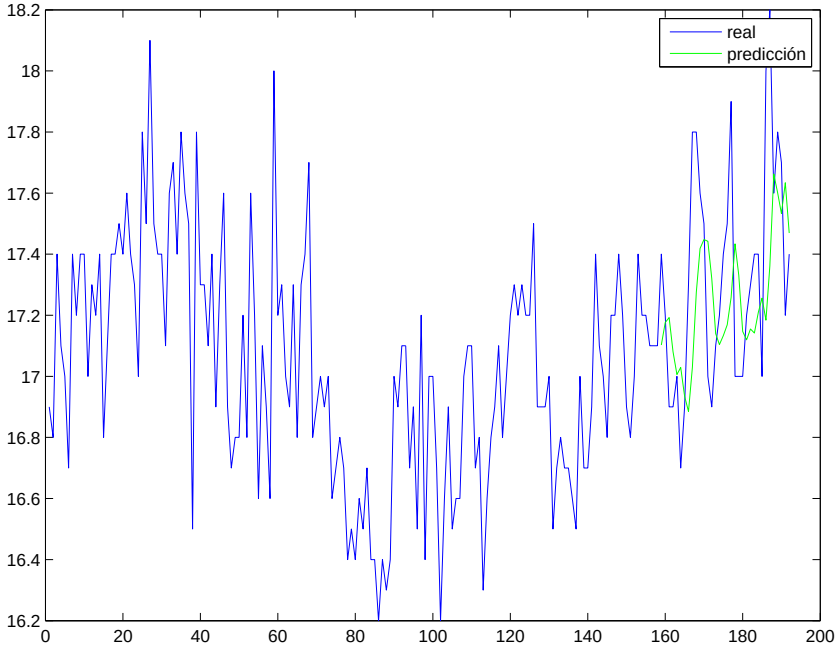

Figura 5.17 OLS. $\lambda = 0$. Set de ensayo.

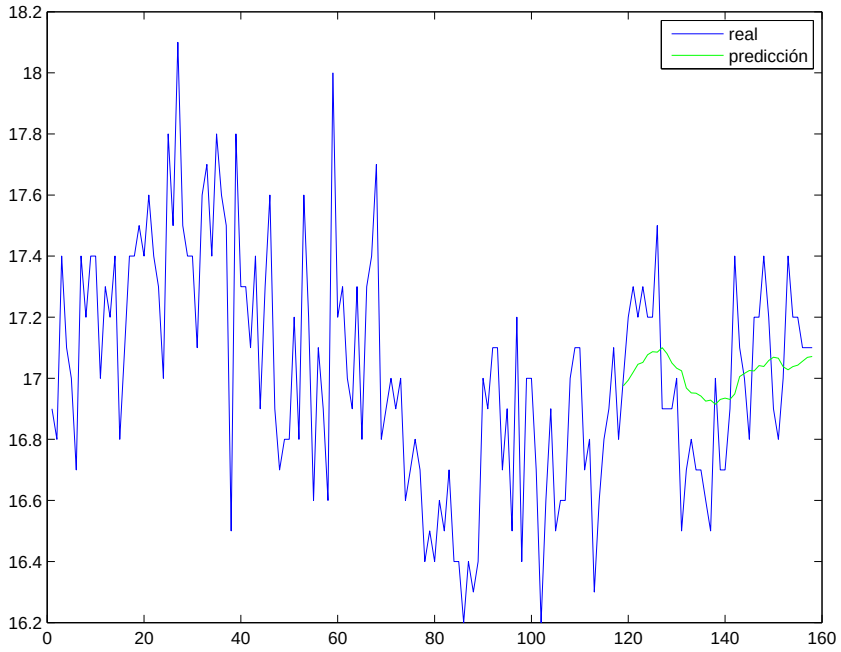

Figura 5.18 OLS. $\lambda = 100$. Set de validación.

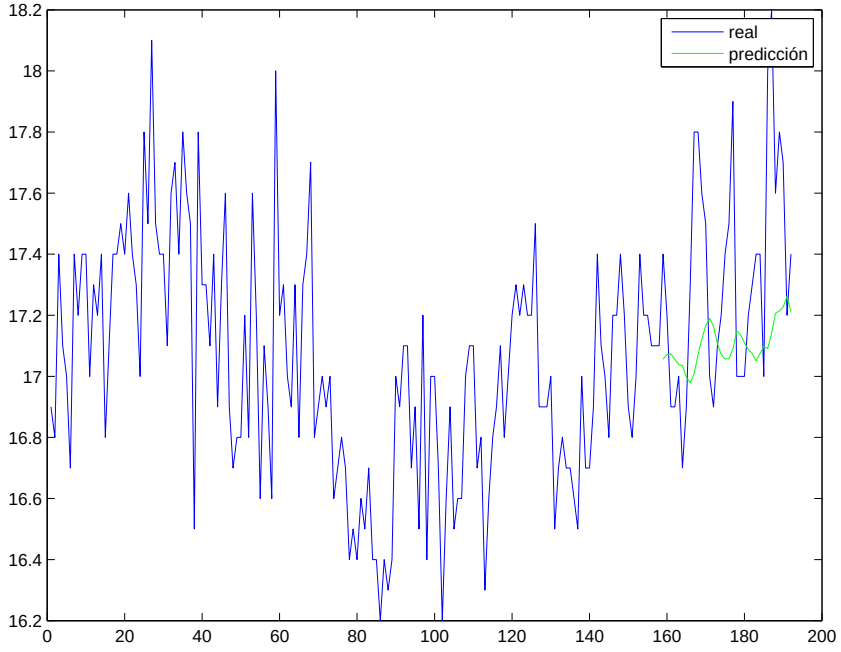

Figura 5.19 OLS. $\lambda = 100$. Set de ensayo.

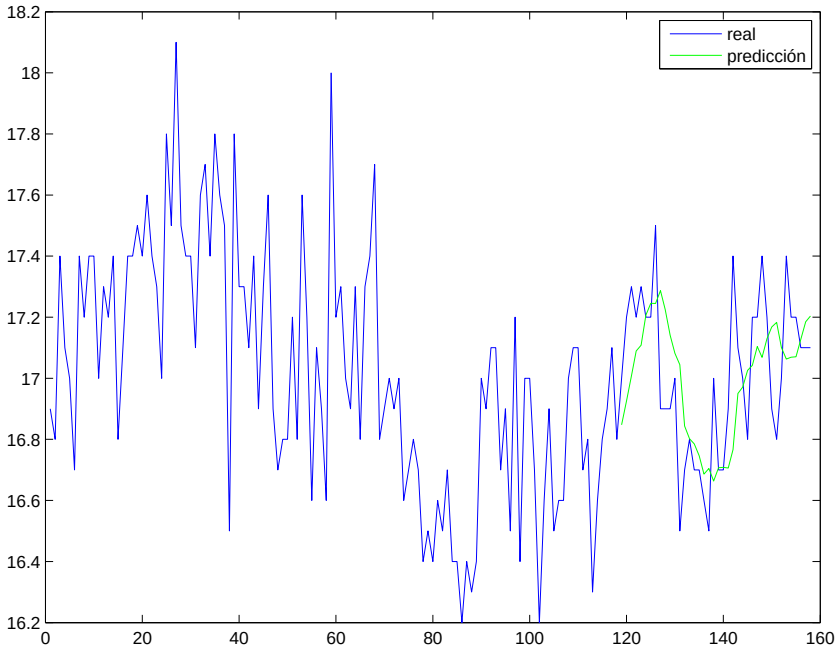

Figura 5.20 Gradient descent. Set de validación.

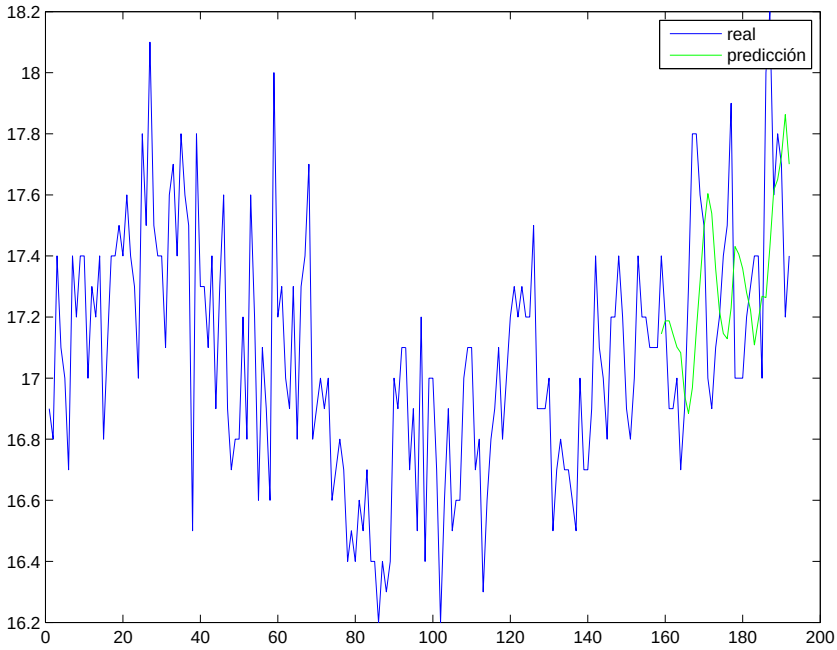

Figura 5.21 Gradient descent. Set de ensayo.

5.2.3 Regresor de verosimilitud

En este caso se ha elegido solo un valor del parámetro γ, siendo $\gamma = 0.05$ el elegido debido a que se ajusta a la distribución probabilística de los datos tal y como veremos en los resultados de predicción intervalar.

Tabla 5.3 Resultados regresor disimilitud para concentraciones químicas.

	Parámetros	Figuras	Error cuadrático medio Validación	Ensayo
Regresor disimilitud	$\gamma = 0.05$	5.22 5.23	0.0489	0.1285

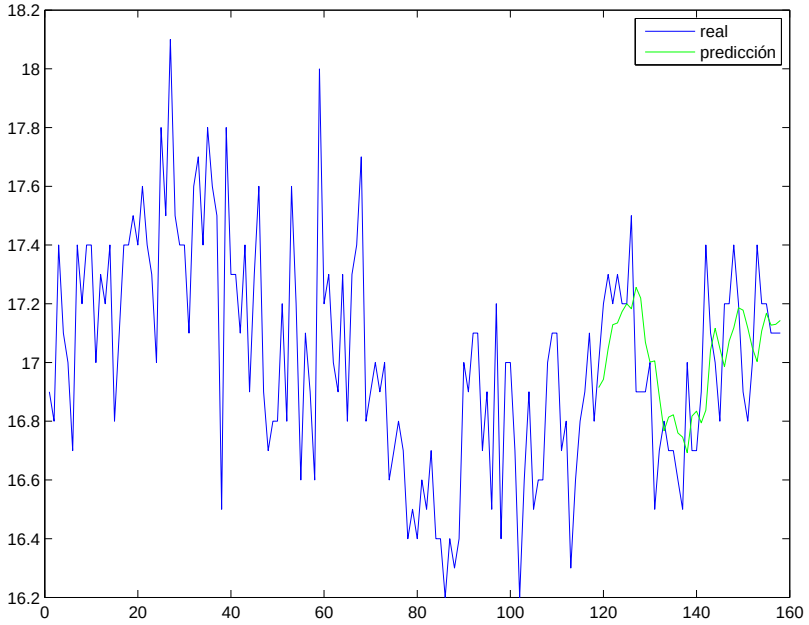

Figura 5.22 Regresor verosimilitud. $\gamma = 0.05$. Set de validación.

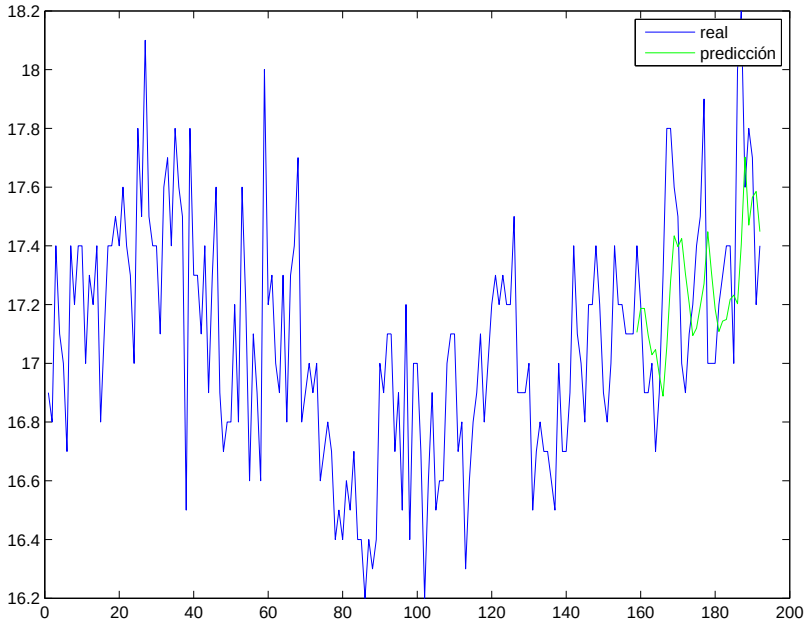

Figura 5.23 Regresor verosimilitud. $\gamma = 0.05$. Set de ensayo.

5.2.4 Predicción intervalar. Algoritmo de verosimilitud

Para terminar con la muestra, se enseñarán los resultados obtenidos mediante la predicción intervalar con funciones de verosimilitud. Se mostrarán varios ejemplos de sets de validación hasta conseguir los parámetros que ajustan a la muestra en este set. Se ha considerado que los parámetros ajustan a la muestra cuando el error cometido en el set de validación es igual (o muy cercano) al valor de α, es decir, al grado de verosimilitud pedido. A continuación, se emplearán estos valores para analizar los resultados del algoritmo en el set de ensayo.

Se han considerado como resultados relevantes la longitud media del intervalo, el tiempo medio de cálculo (por muestra) y el acierto, entendiendo como acierto el número de datos que entran en el intervalo predicho entre el número de datos totales calculados.

Tabla 5.4 Resultados predictor intervalar para concentraciones químicas.

			Acierto (%)		Tiempo (s)		Longitud	
	Parámetros	Figuras	Val.	Test	Val.	Test	Val.	Test
Predicción intervalar	$c = 10$ $\gamma = 0.1$ $\alpha = 0.9$	5.24	1		10.89		2.47	
Predicción intervalar	$c = 100$ $\gamma = 0$ $\alpha = 0.9$	5.25	0.95		3.15		0.89	
Predicción intervalar	$c = 100$ $\gamma = 0.15$ $\alpha = 0.9$	5.26	0.88		17.94		0.63	
Predicción intervalar	$c = 500$ $\gamma = 0.1$ $\alpha = 0.9$	5.27	0.5		16.02		0.34	
Predicción intervalar	$c = 1000$ $\gamma = 0$ $\alpha = 0.9$	5.28	0.67		3.22		0.43	
Predicción intervalar	$c = 100$ $\gamma = 0.05$ $\alpha = 0.9$	5.29	0.9		10.21		0.69	
Predicción intervalar	$c = 100$ $\gamma = 0.05$ $\alpha = 0.85$	5.30 5.31	0.85	0.65	10.88	10.94	0.60	0.64
Predicción intervalar	$c = 100$ $\gamma = 0.05$ $\alpha = 0.95$	5.32 5.33	0.95	0.82	9.83	10.15	0.82	0.86

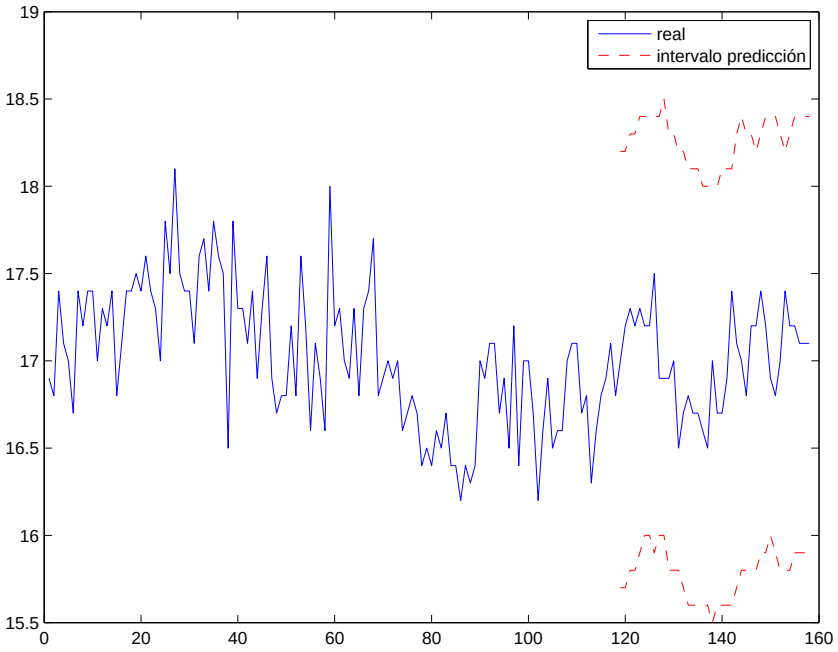

Figura 5.24 Predicción intervalar. c=10. $\gamma = 0.1$. $\alpha = 0.9$. Set de validación.

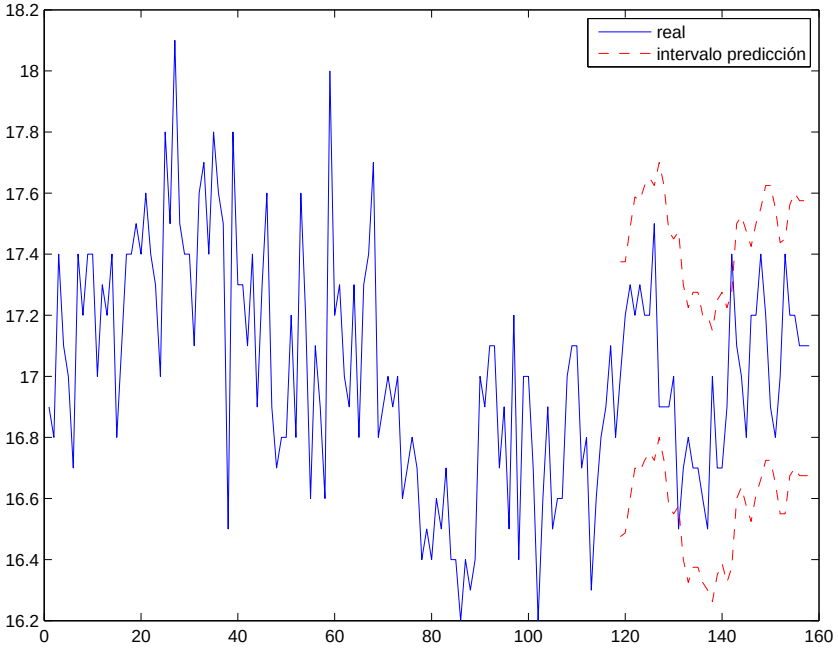

Figura 5.25 Predicción intervalar. c=100. $\gamma = 0$. $\alpha = 0.9$. Set de validación.

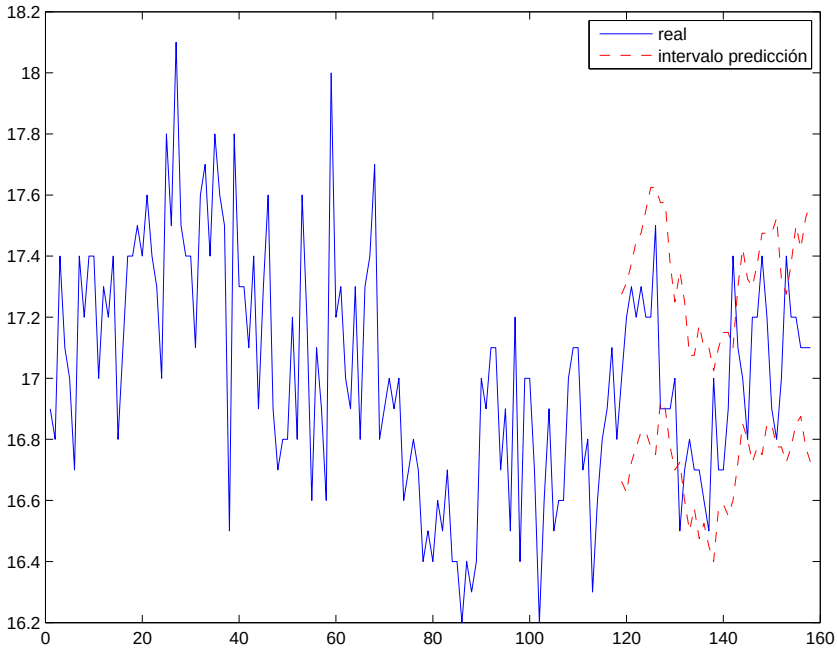

Figura 5.26 Predicción intervalar. c=100. $\gamma = 0.15$. $\alpha = 0.9$. Set de validación.

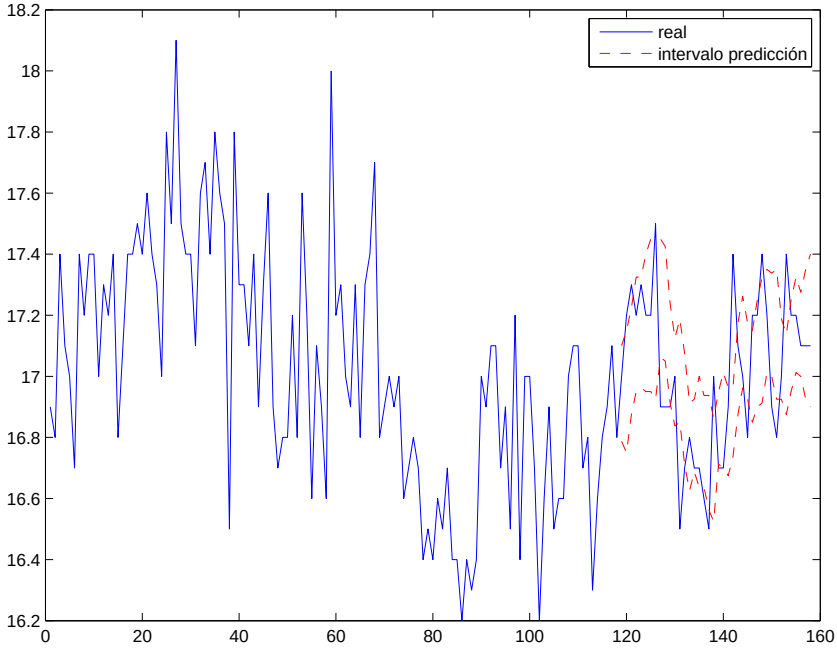

Figura 5.27 Predicción intervalar. c=500. $\gamma = 0.1$. $\alpha = 0.9$. Set de validación.

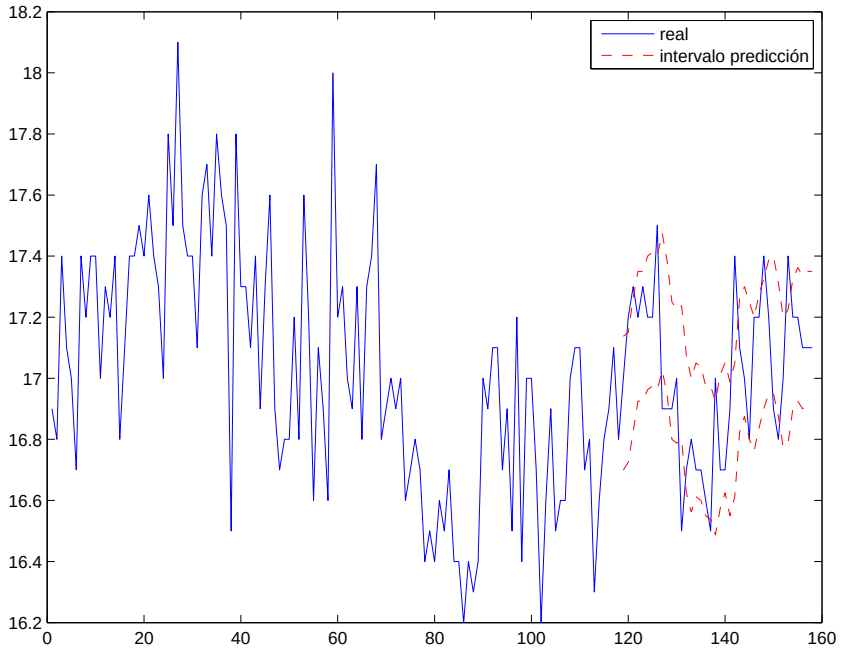

Figura 5.28 Predicción intervalar. c=1000. $\gamma = 0$. $\alpha = 0.9$. Set de validación.

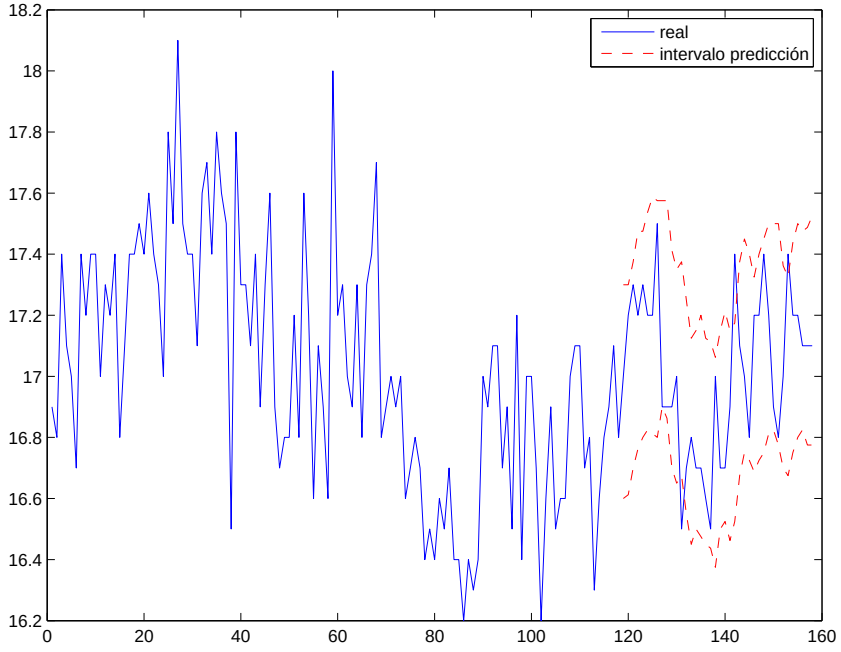

Figura 5.29 Predicción intervalar. c=100. $\gamma = 0.05$. $\alpha = 0.9$. Set de validación.

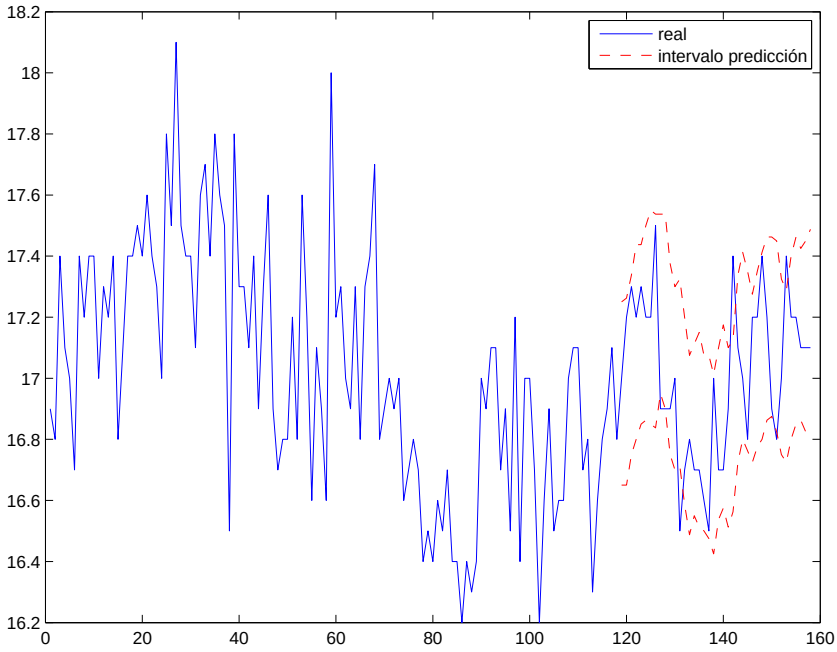

Figura 5.30 Predicción intervalar. c=100. $\gamma = 0.05$. $\alpha = 0.85$. Set de validación.

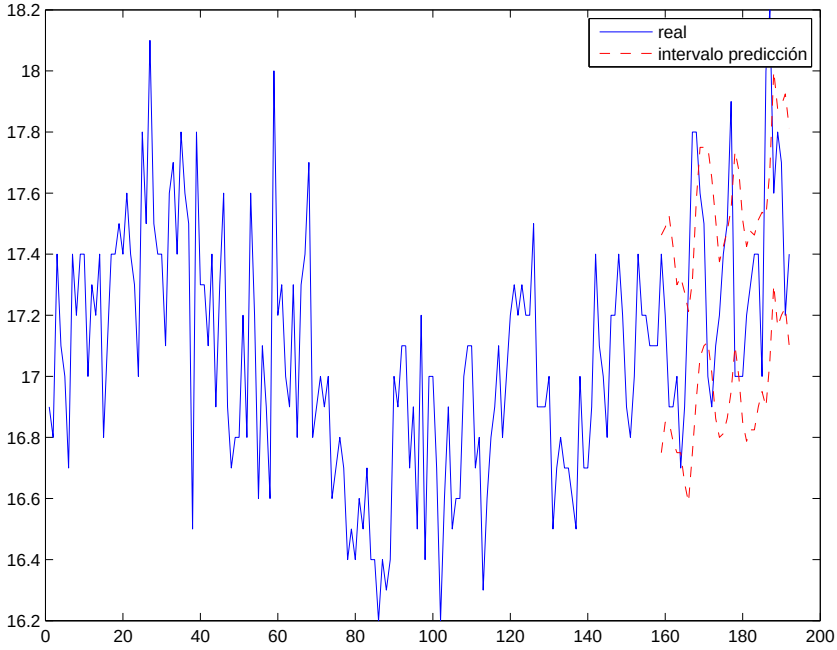

Figura 5.31 Predicción intervalar. c=100. $\gamma = 0.05$. $\alpha = 0.85$. Set de ensayo.

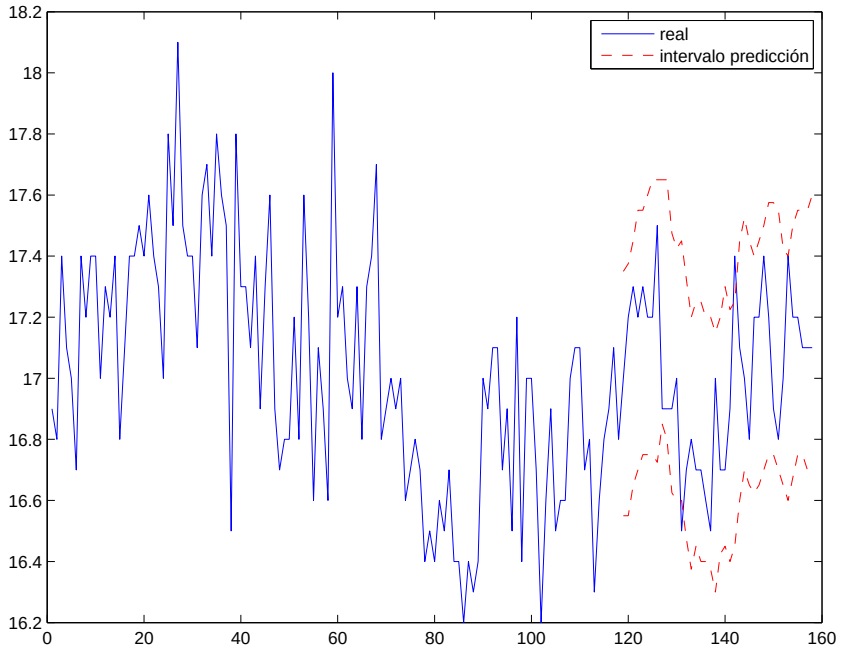

Figura 5.32 Predicción intervalar. c=100. $\gamma = 0.05$. $\alpha = 0.95$. Set de validación.

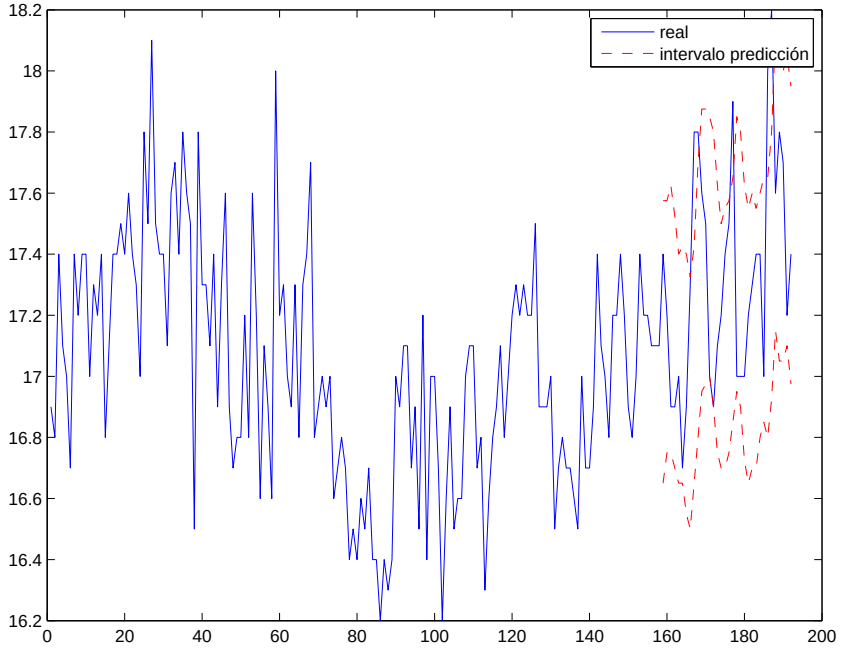

Figura 5.33 Predicción intervalar. c=100. $\gamma = 0.05$. $\alpha = 0.95$. Set de ensayo.

5.3 Flor de iris

Esta muestra nos ayudará a resolver el problema de la clasificación multiclase en un set de datos reducido y bien estudiado en la literatura. Se utilizarán los algoritmos de Gradient descent y de SVM. Los resultados se mostrarán mediante la matriz de confusión y mediante estadísticos como el porcentaje de aciertos (acierto), la medida F1 y el coeficiente de correlación de Matthew (MCC).

Se ha resuelto la clasificación mediante 3 clasificadores binarios. Los estadísticos se han calculado mediante la media de estos estadísticos para las tres clases.

Tabla 5.5 Resultados Gradient descent para flor de iris.

	Parámetros	Matriz confusión	Acierto Val.	Acierto Test	F1 Val.	F1 Test	MCC Val.	MCC Test
Gradient descent	$\lambda = 0$ $\alpha = 0.2$ Iteraciones $= 100$	5.6 5.7	0.64	0.67	0.64	0.70	0.61	0.65
Gradient descent	$\lambda = 0$ $\alpha = 2$ Iteraciones $= 100$	5.8 5.9	0.96	0.83	0.96	0.85	0.94	0.80
Kernel lineal	$\lambda = 1$ $\gamma = 1$	5.10 5.11	0.84	0.67	0.84	0.70	0.79	0.65
Kernel polinómico	$\lambda = 1$ $\gamma = 1$ $c = 1$ $d = 2$	5.12 5.13	1	0.96	1	0.96	1	0.94
Kernel RBF	$\lambda = 1$ $\gamma = 100$ $\sigma = 1$	5.14 5.15	1	1	1	1	1	1

Tabla 5.6 Gradient descent. $\lambda = 0$. $\alpha = 0.2$. Iteraciones $= 100$. Set de validación.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	8	0	0
	Versicolor	0	3	9
	Virginica	0	0	5

Tabla 5.7 Gradient descent. $\lambda = 0$. $\alpha = 0.2$. Iteraciones $= 100$. Set de ensayo.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	6	0	0
	Versicolor	0	4	8
	Virginica	0	0	6

Tabla 5.8 Gradient descent. $\lambda = 0$. $\alpha = 2$. Iteraciones $= 100$. Set de validación.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	8	0	0
	Versicolor	0	11	1
	Virginica	0	0	5

Tabla 5.9 Gradient descent. $\lambda = 0$. $\alpha = 2$. Iteraciones $= 100$. Set de ensayo.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	6	0	0
	Versicolor	0	8	4
	Virginica	0	0	6

Tabla 5.10 SVM. Kernel lineal. $\lambda = 1$. $\gamma = 1$. Set de validación.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	8	0	0
	Versicolor	0	8	4
	Virginica	0	0	5

Tabla 5.11 SVM. Kernel lineal. $\lambda = 1$. $\gamma = 1$. Set de ensayo.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	6	0	0
	Versicolor	0	4	8
	Virginica	0	0	6

Tabla 5.12 SVM. Kernel polinómico. $\lambda = 1$. $\gamma = 1$. $c = 1$. $d = 2$. Set de validación.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	8	0	0
	Versicolor	0	12	0
	Virginica	0	0	5

Tabla 5.13 SVM. Kernel polinómico. $\lambda = 1$. $\gamma = 1$. $c = 1$. $d = 2$. Set de ensayo.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	6	0	0
	Versicolor	0	11	1
	Virginica	0	0	6

Tabla 5.14 SVM. Kernel RBF. $\lambda = 1$. $\gamma = 100$. $\sigma = 1$. Set de validación.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	8	0	0
	Versicolor	0	12	0
	Virginica	0	0	5

Tabla 5.15 SVM. Kernel RBF. $\lambda = 1$. $\gamma = 100$. $\sigma = 1$. Set de ensayo.

		Clasificaciones		
		Setosa	Versicolor	Virginica
Realidad	Setosa	6	0	0
	Versicolor	0	12	0
	Virginica	0	0	6

5.4 Plan de pensiones

El número de muestras de este set es de 3,653, lo que nos ayudará en el proceso de aprendizaje ya que, a mayor número de datos, mayor es la facilidad de generalizar a partir de ellos. El problema a tratar es el de predecir el valor que tendrá la salida en "r" muestras. Esto es muy útil, sobretodo en problemas como este ya que permite predecir momentos óptimos en los que realizar cierta inversión. Cabe mencionar que a mayor horizonte de predicción, mayor es la incertidumbre, lo que conlleva a una peor predicción.

A este problema aplican todos los algoritmos que resuelven problemas de regresión, sin embargo, puesto que ya se han estudiado y comparado varios de ellos anteriormente, en este caso se va a emplear el algoritmo de mínimos cuadrados ordinarios (OLS), pues proporciona una solución rápida y buena. Se comparará con el mantenedor de orden cero (MOC), la solución más simple a cualquier problema de predicción temporal. Al contrario de lo que pueda parecer, el MOC es una solución que suele obtener buenos resultados en la práctica y es la empleada en multitud de aplicaciones, por lo que su comparación resulta de interés. Se trabajará con una dimensión de regresor 10 en el caso del algoritmo OLS, equivalente a tomar 10 salidas pasadas para predecir una futura. A diferencia de los resultados en el set de concentraciones químicas, se ha cambiado la visualización de los resultados en la gráfica temporal por una comparación de las salidas predichas con las reales, donde, en una regresión perfecta, todos los puntos se situarían en la recta y=x, pero,

en una regresión real, los puntos se situarán lo más cerca posible a la recta mencionada. La tabla donde se muestran los errores cuadráticos medios de los algoritmos en ambos sets (de validación y de ensayo) se ha mantenido, pues ofrece un resultado numérico muy claro que nos permite valorar rápidamente el desempeño de los algoritmos.

Tras esto, se hará uso del set de validación para visualizar una curva de aprendizaje (learning curve) que nos permita comprobar si el algoritmo realmente aprende a medida que aumentamos los datos de entrenamiento.

Por último, se representarán los coeficientes de correlación tanto de la muestra original, como de las variaciones de ésta. Estos coeficientes permitirán explicar la importancia que tienen regresores como el MOC en esta muestra.

Tabla 5.16 Resultados OLS para concentraciones químicas.

	Parámetros	Figuras	Error cuadrático medio Validación	Ensayo
MOC	$r = 1$	5.34 5.35	0.0238	0.0380
MOC	$r = 10$	5.36 5.37	0.2726	0.3730
OLS	$r = 1$ $d = 10$ $\lambda = 0$	5.38 5.39	0.0251	0.0402
OLS	$r = 10$ $d = 10$ $\lambda = 0$	5.40 5.41	0.3495	0.4305

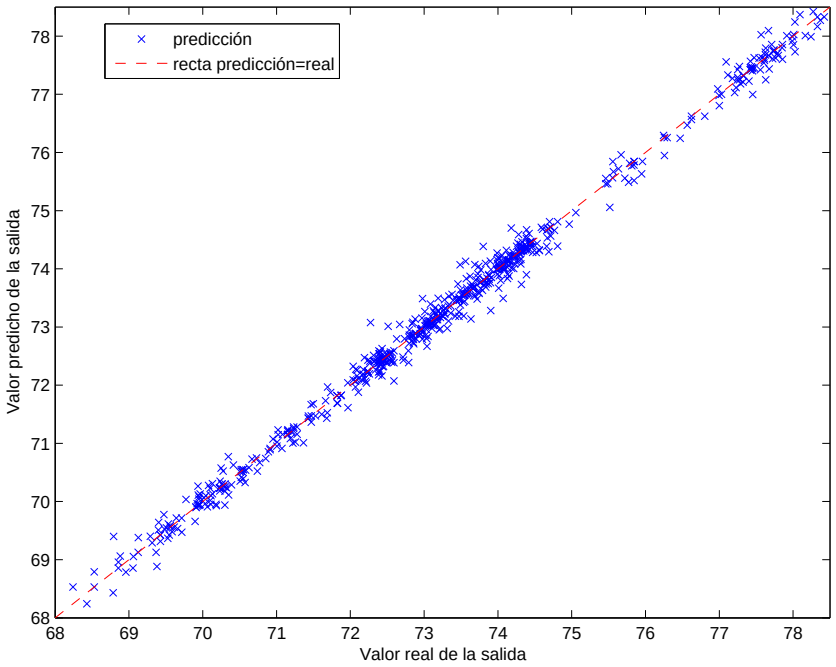

Figura 5.34 MOC. $r = 1$. Set de validación.

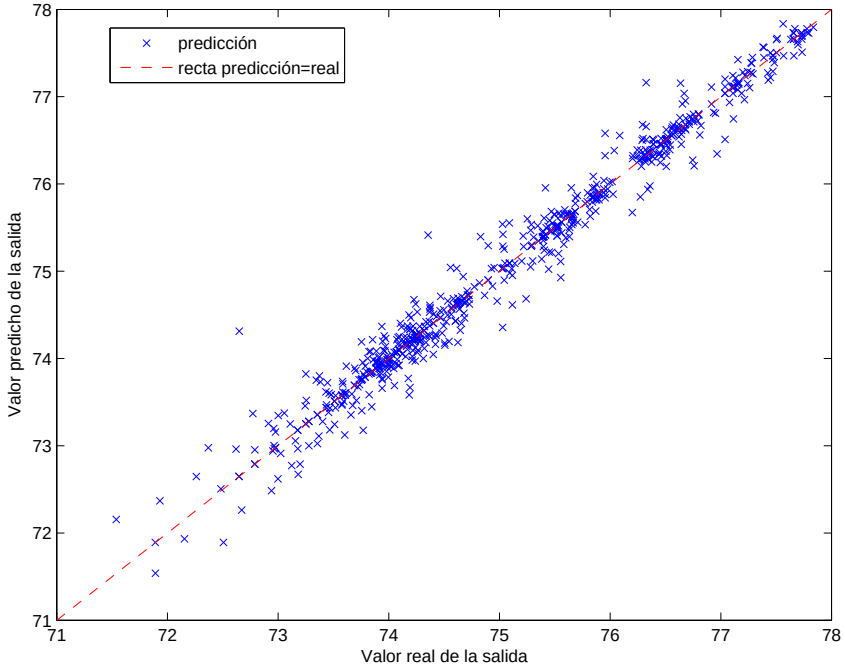

Figura 5.35 MOC. $r = 1$. Set de ensayo.

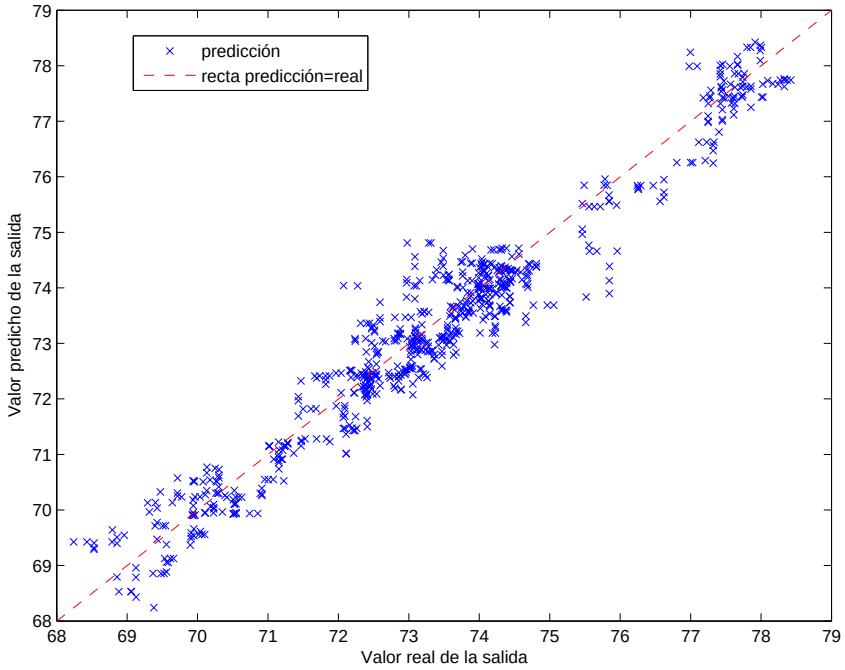

Figura 5.36 MOC. $r = 10$. Set de validación.

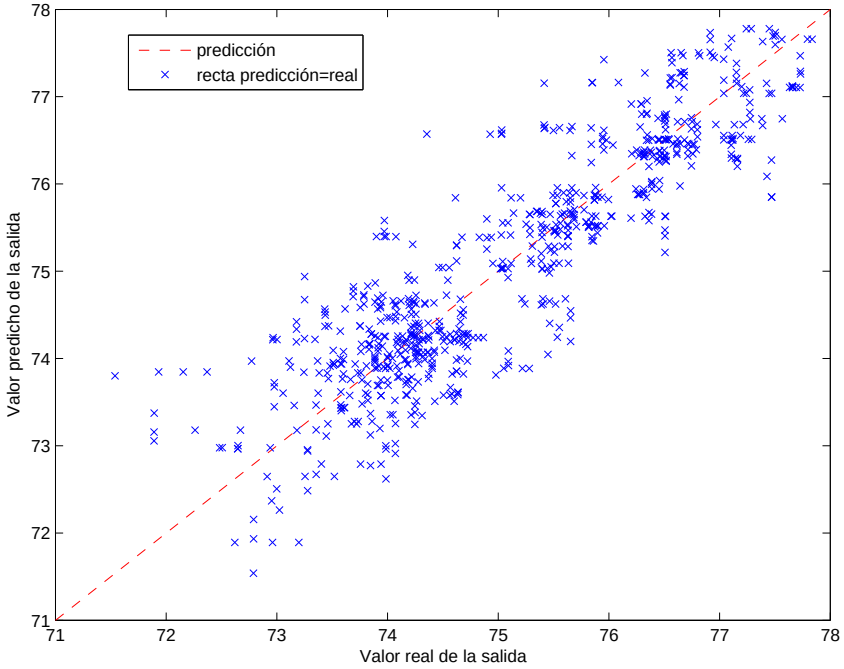

Figura 5.37 MOC. $r = 10$. Set de ensayo.

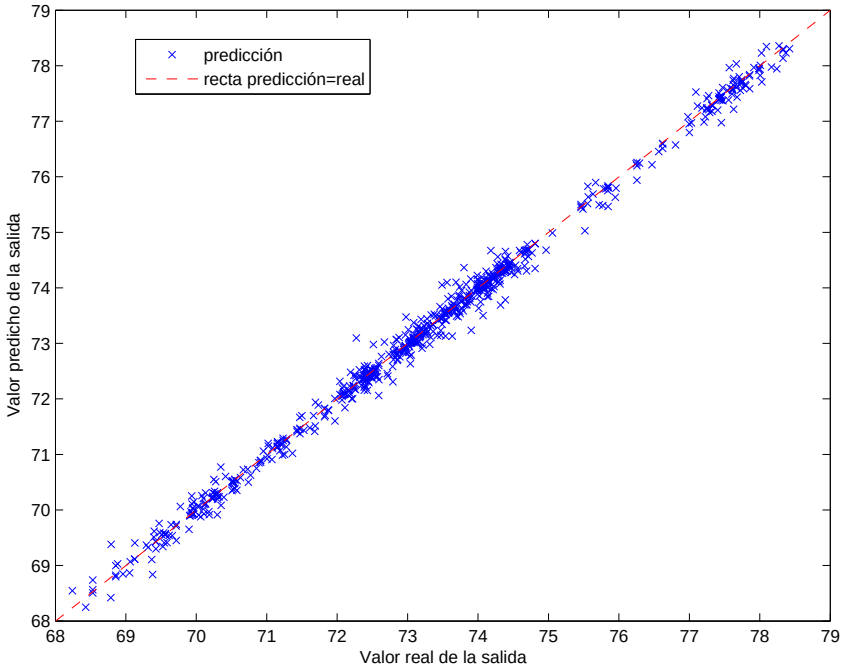

Figura 5.38 OLS. $\lambda = 0$. d=10. r=1. Set de validación.

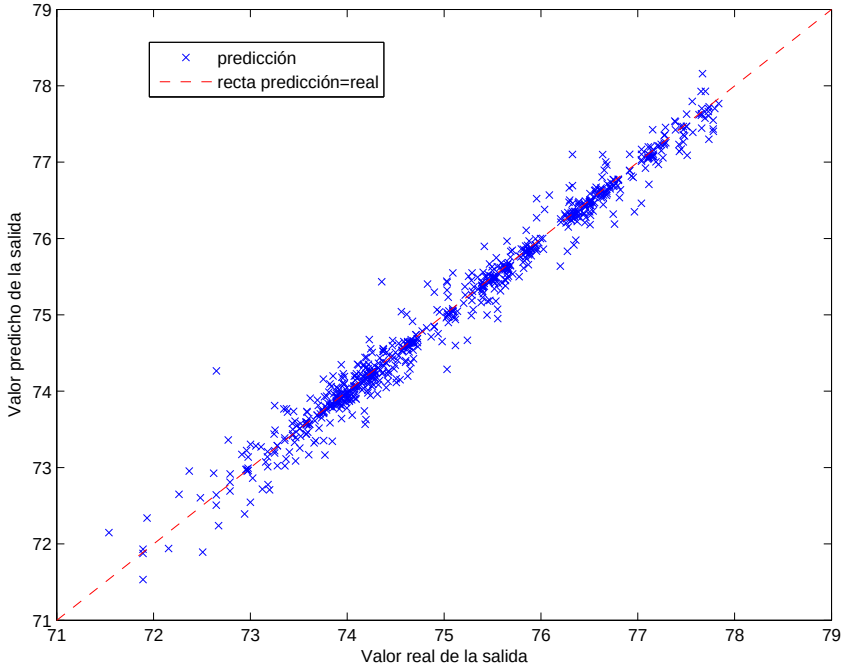

Figura 5.39 OLS. $\lambda = 0$. d=10. r=1. Set de ensayo.

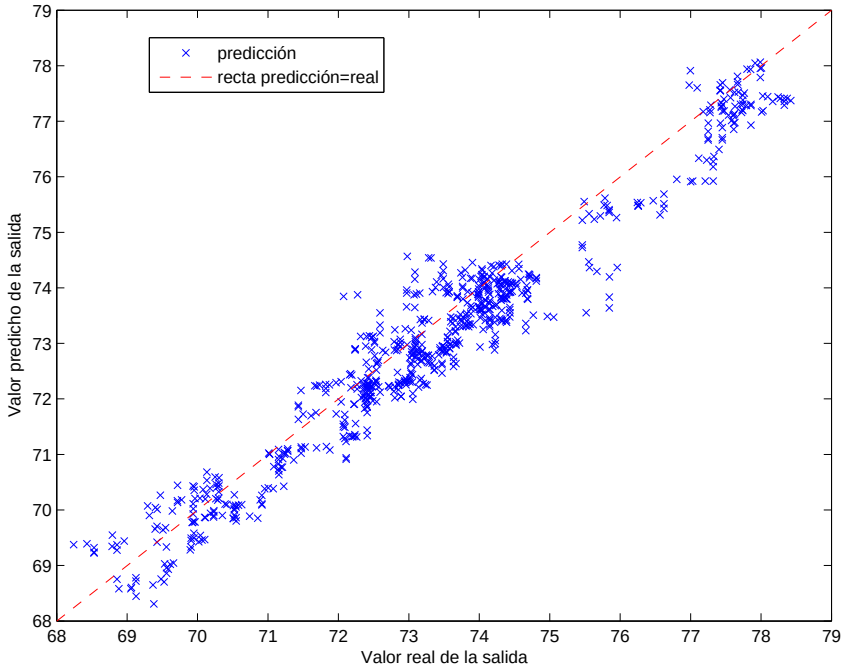

Figura 5.40 OLS. $\lambda = 0$. d=10. r=10. Set de validación.

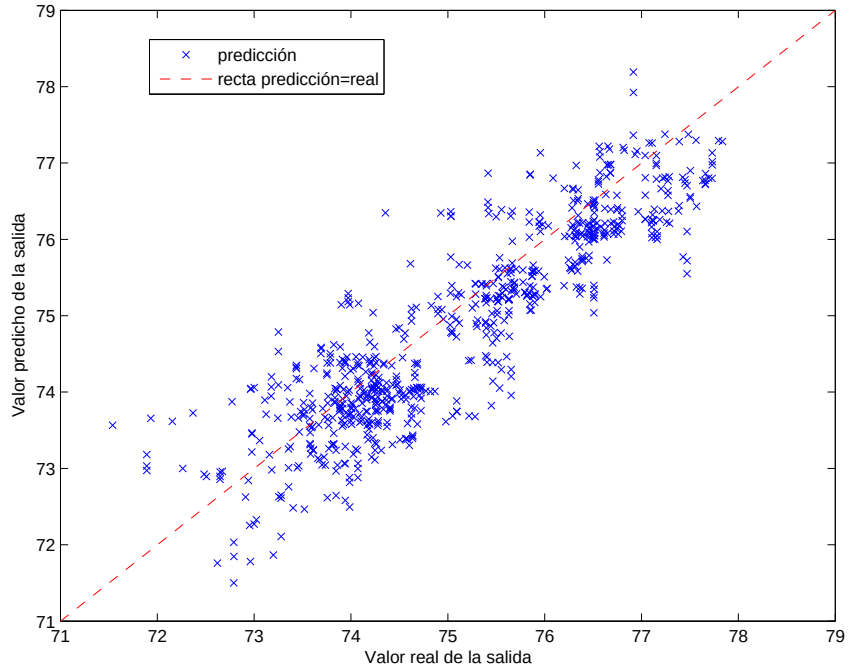

Figura 5.41 OLS. $\lambda = 0$. d=10. r=10. Set de ensayo.

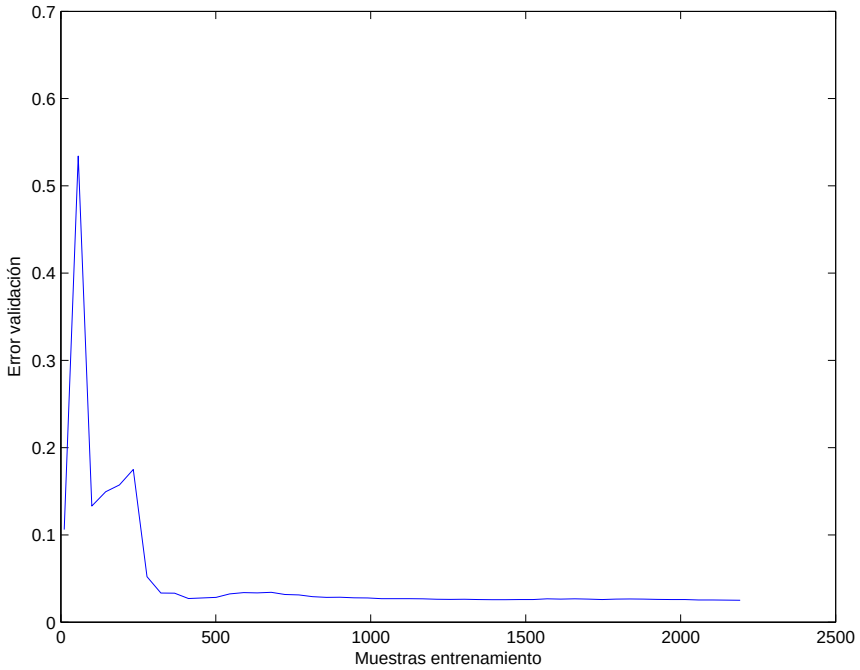

Figura 5.42 Curva aprendizaje OLS. $\lambda = 0$. d=10. r=1. Set de validación.

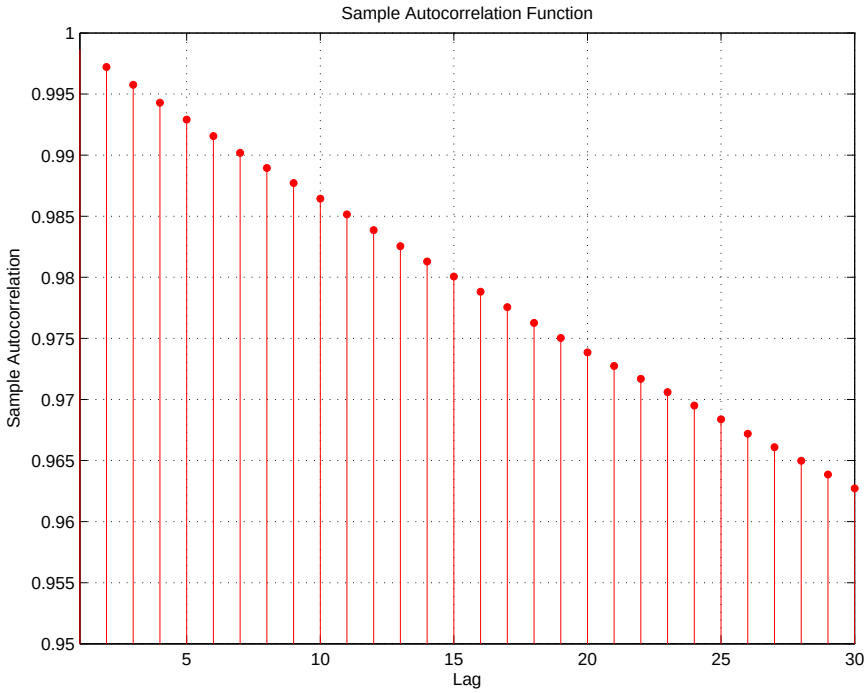

Figura 5.43 Coeficiente de correlación de los datos de bankia.

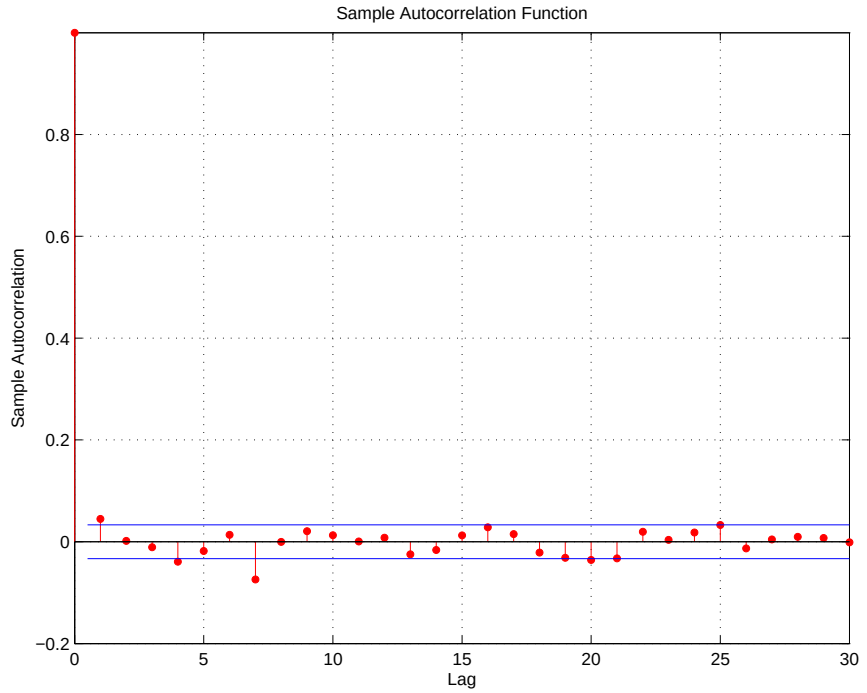

Figura 5.44 Coeficiente de correlación de las variaciones de los datos de bankia.

6 Conclusiones

En el capítulo se analizarán los resultados presentados en el apartado anterior. Se estudiará tanto el desempeño de los diversos algoritmos, como el efecto que tienen los parámetros sobre estos.

6.1 Imágenes

Como se ha visto en el apartado anterior, el algoritmo k-means nos proporciona un buen tratamiento inicial de las imágenes a la hora de dividirlas en regiones. El problema que se ha tratado ha sido únicamente la separación simple de regiones, pero se podría continuar el estudio añadiendo campos en el regresor que nos permitan una separación más fina, como puede ser información local de los puntos del entorno. Observamos asimismo que la separación realizada es sensible al ruido, como se puede apreciar en el ruido sal y pimienta de la imagen 5.2. También podemos observar cómo la separación puede no ser perfecta por otros motivos, como los de la imagen 5.3, en la que se aprecia como el marco, al estar en un estado intermedio entre el triángulo y el fondo, no termina de separarse bien y, mientras que la mayoría de él se clasifica como fondo, un pequeño trozo abajo a la izquierda logra clasificarse como triángulo.

6.2 Concentraciones proceso químico

6.2.1 SVM

Los resultados obtenidos mediante los métodos de SVM se ajustan generalmente bastante bien a la realidad. Observamos que el kernel lineal ofrece los mejores resultados, algo predecible ya que los modelos simples tienden a dar buenos resultados. A diferencia de modelos polinomiales, el kernel lineal no varía en exceso frente a los cambios en las entradas, por lo que proporciona una aproximación poco ambiciosa (un modelo complejo podría intentar aproximar mejor la curva), pero eficaz. Esto se comprueba en los dos resultados siguientes del kernel polinomial, en los que se observa que, a medida que se complica el modelo aumentando el grado del polinomio, el algoritmo puede llegar a ajustar mejor a la muestra, pero cuando la muestra varía y probamos dichos parámetros en el test set, los resultados empeoran muy rápido. Por último, el kernel RBF (Radial Basis Function) ofrece también buenos resultados siempre que se tunée adecuadamente el parámetro σ.

6.2.2 OLS & Gradient descent

Estos dos métodos proporcionan los mismos resultados, al igual que el kernel lineal. Estos resultados se ajustan bastante al problema y, la diferencia reside en las posibilidades de cada algoritmo. Mientras que el kernel lineal ofrece la sencillez de comparar los resultados con otros kernel sin usar más que un algoritmo, el algoritmo de Gradient Descent nos permite trabajar con sets de datos grandes, en especial con variantes como el *Stochastic Gradient Descent* o el *Mini Batch Gradient Descent*. Por último el algoritmo OLS nos permite resolver el problema algebraicamente sin necesidad de iterar, por lo que, en sets de datos pequeños es muy aconsejable.

6.2.3 Regresor de verosimilitud

Los resultados proporcionados por el regresor basado en las funciones de verosimilitud estudiadas son bastante prometedores, ya que proporcionan el mejor resultado en el test de ensayo de entre los algoritmos estudiados. La grandeza del método es que, a diferencia del método de los mínimos cuadrados, nos permite ajustar la predicción a la función de probabilidad. Con un valor de γ adecuado, se pueden conseguir funciones de probabilidad diferentes a la normal, que es la que utilizan los métodos anteriores, por lo que, si cualquiera de estas funciones se adapta más a los datos que la normal, el regresor de verosimilitud ofrecerá mejores resultados. En este caso, observamos que $\gamma = 0.05$ adapta muy bien y los resultados obtenidos son muy buenos.

6.2.4 Predictor intervalar. Algoritmo de verosimilitud

Para terminar con la muestra de concentraciones químicas, analizaremos los resultados del predictor intervalar. Las primeras muestras que se aprecian en la tabla 5.4 se corresponden al proceso de ajuste de los parámetros c y γ al problema. Unos parámetros que no se ajusten bien llevarán a unos malos resultados. Puesto que se está prediciendo con un intervalo de confianza de $\alpha = 0.9$, se espera que el acierto sea asimismo 0.9, algo que ocurre cuando los parámetros $c = 100, \gamma = 0.05$ se ajustan a la muestra. Posteriormente comprobamos que si cambiamos el valor de confianza a $\alpha = 0.85$ y a $\alpha = 0.95$, el acierto sigue siendo igual al valor de confianza α, por lo que observamos que la función se ajusta bien a la muestra. También observamos que en el set de ensayo, los resultados obtenidos empeoran. Esto era esperable ya que ha ocurrido en los métodos anteriores y se puede deber a un cambio de la dinámica de la muestra, que se traduce en que la función de probabilidad ha cambiado del set de validación al de ensayo.

Por otra parte, es interesante ver cómo al cambiar los parámetros, también cambian el tiempo de cálculo del intervalo y la longitud de este. Observamos que, a medida que los valores de α y c aumentan, también lo hace el tiempo de cálculo, mientras que la longitud media del intervalo disminuye.

6.3 Flor de Iris

En este set se ha optado por estudiar el comportamiento del Gradient descent con dos valores diferentes distintos del coeficiente α de aprendizaje. Observamos como con $\alpha = 0.2$ y 100 iteraciones, el algoritmo no ha terminado de converjer, por lo tanto no ha aprendido lo suficiente de los datos si lo comparamos con el segundo ensayo, con $\alpha = 2$, donde

se aprecian resultados mejores, dado que el algoritmo sí ha convergido. Si siguiésemos aumentando este coeficiente, llegaríamos a un valor en el que dejaría de aprender más rápido (necesitando menos iteraciones) y pasaría a no converger, tal como se explicó en apartados anteriores.

Por otro lado tenemos que los métodos de SVM ofrecen esta vez muy buenos resultados, llegando incluso a separar perfectamente los tres sets. Observamos como en esta muestra sí somos capaces de mejorar los resultados mediante el uso de métodos más sofisticados que los mínimos cuadrados o el Gradient Descent.

6.4 Plan de pensiones

Este problema nos ofrece unos resultados un poco inesperados a priori. El mantenedor de orden 0, método que predice que el valor futuro será igual al actual, proporciona siempre mejores resultados que el algoritmo de mínimos cuadrados en esta muestra. Esto se produce tanto con horizontes de predicción bajos (horizonte de predicción un día) como a horizontes medios. Como ya se comentó, el mantenedor de orden cero, conocido como MOC, es una solución típicamente muy buena y hay problemas, entre los cuales incluimos este del plan de pensiones, en los que es muy difícil conseguir mejores resultados que el MOC. Esto se puede explicar con la ayuda de las figuras 5.43 y 5.44, donde se observa que la muestra $k-1$, tiene una correlación muy alta con la muestra k, que es la que predecimos. Asimismo, observamos que los coeficientes de correlación de las variaciones de los datos son muy bajos, por lo que podríamos despreciar el resto de muestras, ya que no aportan mucha información más que la muestra $k-1$. Esto explica por qué métodos como el MOC funcionan tan bien, ya que solo necesitan de la muestra $k-1$ para predecir, y ésta está suficientemente correlacionada con la muestra k como para que la predicción sea fiable.

Por último, vemos en la curva de aprendizaje cómo el algoritmo realmente aprende a medida que crece el número de muestras con las que se alimenta. En este caso, por sencillez, solo se ha mostrado la evolución del error en el set de validación a medida que se entrena el algoritmo con más muestras, pero es suficiente para comprobar que el error baja, por lo que el algoritmo aprende correctamente de los datos.

6.5 Futuras líneas

Este trabajo tenía como objetivo el estudio de las funciones de verosimilitud propuestas en el capítulo 4, así, se ha centrado en la comprobación de que estas funciones son realmente útiles sin llegar a profundizar del todo en ellas. Para mejorar los resultados, se podría incluir en estas funciones información local del problema tal y como se comenta en el capítulo 4. Esto abriría la puerta a otra familia de soluciones variando los distintos parámetros de ponderación, por lo que sería más fácil obtener mejores resultados. Asimismo, con las funciones desarrolladas y recogiendo los datos adecuados, sería posible utilizar lo estudiado para resolver otros problemas como la detección de anomalías o sistemas de recomendación.

7 Código

Nota: los símbolos ¬ del código corresponden al símbolo de negación ∼.

7.1 Gradient descent

```
function [val_error, train_error, yp_val, yp_train, CM_train, ...
    CM_val] = clas_gradient_descent(xtrain, ytrain, xval, yval, ...
    alfa,niter,lambda,tipo_problema)
if ¬(strcmp(tipo_problema, 'clas') || strcmp(tipo_problema, ...
    'regr'))
      return %Fallo
end
M=size(xtrain,1);
xtrain=[ones(M,1),xtrain];
xval=[ones(length(xval),1),xval];
[M,N]=size(xtrain);
Mv=size(xval,1);
n=size(ytrain,2);
%Inicializacion de variables
theta=NaN(N,n);
yp_train=NaN(M,n);
yp_val=NaN(Mv,n);
for jj=1:n
    theta(:,jj)=rand(N,1);
    for ii=1:niter
        if strcmp(tipo_problema,'clas')
            h=(1./(1+exp(-xtrain*theta(:,jj))));
        elseif strcmp(tipo_problema,'regr')
            h=xtrain*theta(:,jj);
        end
        theta(:,jj)=theta(:,jj)*(1-alfa*lambda/M)- ...
            (alfa/M)*xtrain'* (h-ytrain(:,jj));
        aux(:,ii)=theta(:,jj);
    end
    yp_train(:,jj)=xtrain*theta(:,jj);
    yp_val(:,jj)=xval*theta(:,jj);
end
%Calculo de clases en base a las salidas previamente calculadas
if strcmp(tipo_problema,'clas')
```

```
31      [yp_train, yp_val]= clases(yp_train, yp_val);
32  end
33  %Analizo los resultados de ambos sets
34  [val_error, train_error, yp_train, yp_val, CM_train, CM_val]= ...
        analiza_resultados(ytrain, yp_train, yval, yp_val, ...
        tipo_problema);
```

7.2 Mínimos cuadrados ordinarios (OLS)

```
1
2  function [val_error, train_error, yp_val, yp_train]= ...
       normal_equation( xtrain, ytrain, xval, yval, lambda)
3  if nargin<5
4      lambda=1;
5      if nargin<4
6          return
7      end
8  end
9  M=size(xtrain,1);
10 Mv=size(xval,1);
11 xtrain=[ones(M,1),xtrain];
12 xval=[ones(Mv,1),xval];
13 N=size(xtrain,2);
14 L=eye(N);
15 L(1,1)=0;
16 Ny=size(ytrain,2);
17 %Inicializacion de variables
18 theta=NaN(N,Ny);
19 yp_train=NaN(M,Ny);
20 yp_val=NaN(Mv,Ny);
21 %Calculo de salidas
22 for jj=1:Ny
23     theta(:,jj)=pinv(xtrain'*xtrain+lambda*L)*xtrain'*ytrain(:,jj);
24     yp_train(:,jj)=xtrain*theta(:,jj);
25     yp_val(:,jj)=xval*theta(:,jj);
26 end
27 %Analizo los resultados de ambos sets
28 [val_error, train_error,yp_train,yp_val]= analiza_resultados( ...
       ytrain, yp_train, yval, yp_val, 'regr');
```

7.3 SVM

```
1
2  function [val_error, train_error, yp_train, yp_val, CM_train, ...
       CM_val]= svm( xtrain, ytrain, xval, yval, id, lambda, par1, ...
       par2, tipo_problema, gamma)
3  %Calculo y analizo resultados en el set de entrenamiento
4  yp_train= svm_raw(xtrain, ytrain, xtrain, id, lambda, par1, ...
       par2, gamma);
```

```
%Calculo y analizo resultados en el set de validacion
yp_val=svm_raw(xtrain, ytrain, xval, id, lambda, par1, par2, ...
    gamma);
%Calculo de clases en base a las salidas previamente calculadas
if strcmp(tipo_problema,'clas')
    [yp_train,yp_val]=clases(yp_train,yp_val);
end
%Analizo los resultados de ambos sets
[val_error, train_error,yp_train,yp_val,CM_train,CM_val]= ...
    analiza_resultados(ytrain, yp_train, yval, yp_val, ...
    tipo_problema);
```

```

function y=svm_raw(xi, yd, x, id, lambda, par1, par2, gamma) ...
    %xi entradas, yd salida deseada, id identificador kernell y ...
    x entrada actual. y es la salida actual que se calcula.
switch id
    case 1
    case 2
        c=par1;
        d=par2;
    case 3
        sigma=par1;
    case 4
        ca=par1;
        cb=par2;
    otherwise
        return
end
N=size(xi,1);
n=size(x,1);
tau=NaN(N,1); %tau es el vector de pesos
for ii=1:N
    tau(ii)=lambda^(N-ii);
end
s=sqrt(tau);
D=diag(s);
K=NaN(N);
for ii=1:N
    for jj=1:N
        if id==1 %Lineal
            K(ii,jj)=xi(ii,:)*xi(jj,:)';
        end
        if id==2 %Polinomica
            K(ii,jj)=(c+xi(ii,:)*xi(jj,:)')^d;
        end
        if id==3 %RBF
            K(ii,jj)=exp(-(norm(xi(ii,:)-xi(jj,:))^2)'/sigma^2);
        end
        if id==4 %Sigmoidal
            K(ii,jj)=tanh(ca*xi(ii,:)*xi(jj,:)'+cb);
        end
    end
end
nn=size(D*yd,2);
```

```
Af=[0,s';s,D*K*D+(1/gamma)*eye(length(s))];
bf=[zeros(1,nn);D*yd];
xo=Af\bf;
b=xo(1,:);
alpha=xo(2:end,:);
K2=NaN(n,N);
for ii=1:n
    for jj=1:N
        if id==1
            K2(ii,jj)=x(ii,:)*xi(jj,:)';
        end
        if id==2
            K2(ii,jj)=(1+x(ii,:)*xi(jj,:)')^d;
        end
        if id==3
            K2(ii,jj)=exp(-(norm(xi(jj,:)-x(ii,:))^2)/(sigma^2));
        end
        if id==4
            K2(ii,jj)=tanh(ca*x(ii,:)*xi(jj,:)'+cb);
        end
    end
end
b=repmat(b,n,1);
y=double(b+K2*(D*alpha));
```

7.4 Regresor de disimilitud

```
function [yp_val,val_error,yp_train,train_error]= ...
    dissimilarity_regressor_fista_reg(xtrain, ytrain, xval, ...
    yval, gamma)
if nargin<5
    gamma=0;
    if nargin <4
        return
    end
end

%Adecua las dimensiones
xtrain=xtrain';
ytrain=ytrain';
xval=xval';
yval=yval';

m=size(ytrain,2);
yp_train=NaN(m,1);
M=size(yval,2);
yp_val=NaN(M,1);
%Calculo resultados en el set de entrenamiento
for ii=1:m
    yp_train(ii,1)= dissimilarity_regressor_fista(xtrain, ...
        ytrain, xtrain(:,ii), gamma);
end
%Calculo resultados en el set de test
```

```
for ii=1:M
    yp_val(ii,1)= dissimilarity_regressor_fista(xtrain, ytrain, ...
       xval(:,ii), gamma);
end

%Analizo los resultados de ambos sets
train_error= (1/(length(ytrain))) * (yp_train-ytrain')' * ...
    (yp_train-ytrain');
val_error= (1/(length(yval))) * (yp_val-yval')' * (yp_val-yval');
```

```
function ...
    y_predict=dissimilarity_regressor_fista(xtrain,ytrain,x,gamma)
if nargin<4
    gamma=0;
end
N=size(xtrain,2);
H=ones(N,1);
% Esta parte sirve para incorporar informacion local al algoritmo
% for ii=1:N
%     H(ii,1)=(1/norm(x-xtrain(:,ii)))^2;
%     if H(ii,1)==Inf
%         H(ii,1)=1000;
%     end
% end
A=[xtrain;ones(1,N)];
b=[x;1];
[¬,lambda_opt]=Fista_Method(H,gamma,A,b);
y_predict=ytrain*lambda_opt;
end
```

7.5 Método Fista

```
% Esta funcion obtiene u que minimiza
% J=(1/2)*u'*(u./T) + alpha*sum(abs(u));
% Sujeto a A*u=b;
%
% Notese que T es un vector !!

function [x,u,J,iter]=Fista_Method(T,alpha,A,b,x_inic)


n=size(A,1);

if (nargin==4)
    x=zeros(n,1);
else
    x=x_inic;
end

B_T_sqrt=repmat(sqrt(T)',n,1);
ATsqrt=A.*B_T_sqrt;
```

```
20 H=ATsqrt*ATsqrt'; % H es igual a A*diag(T)*A' pero se calcula ...
       sin tener
21                   % que montar la matriz diag(T). De esta ...
                        forma, el numero
22                   % de operaciones es del orden de N*nx, ...
                        hacerlo directamente
23                   % daria algo que crece con N^2.
24
25 R=chol(H)'; % H=R*R';
26 S=R\eye(n);
27
28 As=S*A;
29 bs=S*b;
30
31 iter=0;
32
33 GoOn=1;
34
35 t0=1;
36 x_old=x;
37 y=x;
38 while (GoOn)
39
40     t1=0.5*(1+sqrt(1+4*t0^2));
41
42     u=c_u_Dual(As'*y,T,alpha);
43     x_new=y-(As*u-bs);
44     y=x_new+((t0-1)/t1)*(x_new-x_old);
45     t0=t1;
46     x_old=x_new;
47     iter=iter+1;
48     if (norm(A*u-b)<1e-4)
49         GoOn=0;
50     end
51 end
52 x=x_new;
53
54 J=(1/2)*u'*(u./T) + alpha*sum(abs(u));
```

```
1  function u=c_u_Dual(c,T,alpha)
2
3  N=length(T);
4  u=zeros(N,1);
5
6  % case c>alpha;
7  c_dif=c-alpha;
8  Ip=c_dif>0;
9  u(Ip)=T(Ip).*c_dif(Ip);
10
11 % case c<-alpha;
12 c_sum=c+alpha;
13 In=c_sum<0;
14 u(In)=T(In).*c_sum(In);
```

7.6 Predicción intervalar

```
function interval_alfa_table(xtrain,ytrain,xval,yval,alfa,cc,gama)
m=size(xval,1);
lm=NaN(m,length(cc)); %longitud media (del intervalo)
for gamma=gama
for c=cc
    pc=0;
    for ii=1:m
        tic
        i1(ii,1)= interval_alfa_quantile2(xtrain, ytrain, ...
            xval(ii,:), (1-alfa)/2, c, gamma);
        i2(ii,1)= interval_alfa_quantile2(xtrain, ytrain, ...
            xval(ii,:), 1-((1-alfa)/2), c, gamma);
        time(ii,1)=toc;
        if yval(ii)≥i1(ii,1) && yval(ii)≤i2(ii,1)
            pc=pc+1;
        end
        lm(ii,1)=i1(ii,1)-i2(ii,1);
    end
    pc=pc/m; %Porcentaje de predicciones correctas
    savename=['c' num2str(c) 'gamma' num2str(gamma) 'pred' ...
        num2str(alfa)];
    savename=savename(savename≠'.')
    savetime=time(:,1);
    savelm=lm(:,1);
    savepc=pc(1);
    savei1=i1(:,1);
    savei2=i2(:,1);
    save(savename, 'savetime', 'savelm', 'savepc', 'savei1', ...
        'savei2', 'alfa')
end
end
```

```
function res=interval_alfa_quantile2(xtrain,ytrain,xk,alfa,c,gamma)
prec=0.005; %Precision en el calculo de alfa
paso=0.1;
if nargin<6
    gamma=0;
end
naux=1;
niter=2;
integralz= int_simpson_fista(@Jgamma_fista, 0, 30, xtrain', ...
    ytrain', xk', gamma, c);
if integralz==0
    'ERROR. INTEGRAL=0'
end
for cc=c
    a=0;
    integral0= int_simpson_fista(@Jgamma_fista, a, a+paso, ...
        xtrain', ytrain', xk', gamma, cc);
    while (integral0/integralz<alfa-prec || ...
        integral0/integralz>alfa+prec)
```

```
        if integral0/integralz<alfa-prec
            paso(niter)=2*paso(niter-1);
            a(niter)=a(niter-1)+paso(niter);
            integral0= int_simpson_fista(@Jgamma_fista, a(1), ...
                a(niter), xtrain', ytrain', xk', gamma, cc);
            naux=niter-1;
            niter=niter+1;
        end
        if integral0/integralz>alfa+prec
            paso(niter)=0.5*paso(niter-1);
            a(niter)=a(naux)+paso(niter);
            integral0= int_simpson_fista(@Jgamma_fista, a(1), ...
                a(niter), xtrain', ytrain', xk', gamma, cc);
            niter=niter+1;
        end
        if integral0==Inf
            'Exceso de precision'
            return
        end
    end
    res=a(end);
end
end
```

```
function ...
    integral=int_simpson_fista(f,a,b,xtrain,ytrain,xk,gamma,cc,n)
%Si no se introduce n, se elige por defecto
if nargin==8
    n=(b-a)*50;
    if n<3
        n=3;
    end
end
n=n-rem(n,3);
h=(b-a)/n;
s=f(xtrain,ytrain,xk,a,gamma,cc)+f(xtrain,ytrain,xk,b,gamma,cc);
for ii=1:n-1
    xi=a+h*ii;
    if rem(ii,3)==0
        s=s+2*f(xtrain,ytrain,xk,xi,gamma,cc);
    else
        s=s+3*f(xtrain,ytrain,xk,xi,gamma,cc);
    end
end
integral=3*s*h/8;
end
```

7.7 Tratamiento de imágenes

```
function separa_imagen(im_old,nclases,niter)
[im_new,M,N]=normaliza_imagen(im_old);
```

```
clase=kmeans_reg(im_new,nclases,niter)-1;
clase=clase/max(clase);
im_new=uint8(255*reshape(clase,M,N));
imshow(im_new),shg;
end
```

```
function [im_new,M,N]=normaliza_imagen(im)
M=size(im,1);
N=size(im,2);
O=size(im,3);
im_new=double(reshape(im,M*N,O));
end
```

```
function clase=kmeans_reg(xtrain,nclases,niter)
rng('shuffle');
centro=NaN(nclases,size(xtrain,2));
aux=[];
%Elige puntos al azar de xtrain como centros
for ii=1:nclases
    aux(ii)=round(rand*(size(xtrain,1)-1))+1;
    while ismember(aux(ii),aux(1:ii-1))
        aux(ii)=round(rand*(size(xtrain,1)-1))+1;
    end
    centro(ii,:,1)=xtrain(aux(ii),:);
end


for nn=1:niter
    for ii=1:size(xtrain,1)
        for jj=1:nclases
            dist(jj)=norm(xtrain(ii,:)-centro(jj,:,nn),2);
        end
        [¬,b]=min(dist);
        clase(ii)=b;
    end
    if nn==1
        figure;
        bclase=clase>1;
        oclase=clase==1;
        plot(xtrain(bclase,1), xtrain(bclase,2), 'bx', ...
            xtrain(oclase,1), xtrain(oclase,2), 'gx', ...
            centro(1,1), centro(1,2), 'ro', centro(2,1), ...
            centro(2,2), 'ro'),shg
    end
    for jj=1:nclases
        centro(jj,:,nn+1)=mean(xtrain(clase==jj,:));
    end
    bclase=clase>1;
    oclase=clase==1;
    figure;
    plot(xtrain(bclase,1), xtrain(bclase,2), 'bx', ...
        xtrain(oclase,1), xtrain(oclase,2), 'gx', ...
```

```
        centro(1,1,nn), centro(1,2,nn), 'ro', centro(2,1,nn), ...
        centro(2,2,nn), 'ro'),shg
end

end
```

7.8 Genérico

```
function [val_error, train_error, yp_train, yp_val, CM_train, ...
    CM_val, MCC_train, MCC_val]= analiza_resultados(ytrain, ...
    yp_train, yval, yp_val, tipo_problema)
if strcmp(tipo_problema,'clas')
    ytrain=union_output(ytrain);
    yval=union_output(yval);
    [CM_train,F1_train,MCC_train]=conf_matrix_multiclas(ytrain, ...
        yp_train);
    [CM_val,F1_val,MCC_val]=conf_matrix_multiclas(yval, yp_val);
    train_error=mean(F1_train);
    val_error=mean(F1_val);
end
if strcmp(tipo_problema,'regr')
    CM_train=-1;
    CM_val=-1;
    MCC_train=-1;
    MCC_val=-1;
    train_error=(1/(length(ytrain))) * (yp_train-ytrain)' * ...
        (yp_train-ytrain);
    val_error=(1/(length(yval)))*(yp_val-yval)'*(yp_val-yval);
end
end
```

```
%Dadas las predicciones para cada clase,
%distingue la clase mas probable
function [clase_train,clase_val]=clases(yp_train,yp_val)
[M,N]=size(yp_train);
Mv=size(yp_val,1);
clase_train=NaN(N,1);
clase_val=NaN(N,1);
for ii=1:M
    yp_max=-Inf;
    for jj=1:N
        if yp_train(ii,jj)>yp_max
            yp_max=yp_train(ii,jj);
            clase_train(ii,1)=jj;
        end
    end
end
for ii=1:Mv
    yp_max2=-Inf;
    for jj=1:N
        if yp_val(ii,jj)>yp_max2
```

```
21              yp_max2=yp_val(ii,jj);
22              clase_val(ii,1)=jj;
23          end
24      end
25  end
26  end
```

```
1   function ...
        wname=descarga_datos2(file,tipo_problema,output_pos,caracs)
2   %Por defecto, coge todas las caracteristicas y la salida esta ...
        al principio
3   if nargin<4
4       caracs=':';
5       if nargin<3
6           output_pos=1;
7           if nargin<2
8               tipo_problema='ts';
9           end
10      end
11  end
12  %Si file es un archivo, lo abre. Si es una matriz, la lee
13  if ischar(file)
14      DATA=xlsread(file);
15  else
16      DATA=file;
17  end
18  if strcmp(tipo_problema,'ts')
19      N=length(DATA);
20      u1=int64(N*6/10);
21      u2=int64(N*8/10);
22      ytrain=DATA(1:u1,:);
23      yval=DATA(u1+1:u2,:);
24      ytest=DATA(u2+1:end,:);
25      wname=datestr(now);
26      wname=wname([1,2,13,14]);
27      save(wname,'DATA','ytrain','yval','ytest','tipo_problema');
28      return;
29  end
30
31  %Inicializacion de variables
32  min_x=[];min_y=[];
33  max_x=[];max_y=[];
34  %Descarta las caracteristicas que no se han seleccionado
35  DATA=DATA(:,caracs);
36  %Coloco columna salida la ultima para que la reconozca split_data
37  y=DATA(:,output_pos);
38  DATA(:,output_pos)=DATA(:,end);
39  DATA(:,end)=y;
40  %Divido datos en sets de entrenamiento
41  [xtrain,ytrain,xval,yval,xtest,ytest]=split_data(DATA);
42  %Normalizo las salidas en funcion del tipo de problema
43  if strcmp(tipo_problema,'clas')
44      min_ytrain=-1;
45      min_yval=-1;
46      min_ytest=-1;
```

```
47      max_ytrain=-1;
48      max_yval=-1;
49      max_ytest=-1;
50      ytrain=split_output(ytrain);
51      yval=split_output(yval);
52      ytest=split_output(ytest);
53      V_DATA=logical(ones(length(ytrain),1));
54      V_DATA2=logical(ones(length(yval),1));
55      V_DATA3=logical(ones(length(ytest),1));
56  else
57      [ytrain,min_ytrain,max_ytrain,V_DATA]=normaliza2(ytrain,1);
58      [yval,min_yval,max_yval,V_DATA2]=normaliza2(yval,1);
59      [ytest,min_ytest,max_ytest,V_DATA3]=normaliza2(ytest,1);
60  end
61  %Ahora normalizo las entradas (independiente del problema)
62  [xtrain,min_x,max_x,V_DATA]=normaliza2(xtrain(V_DATA,:),1);
63  ytrain=ytrain(V_DATA,:);
64  [xval,¬,¬,V_DATA2]=normaliza2(xval(V_DATA2,:),1);
65  yval=yval(V_DATA2,:);
66  [xtest,¬,¬,V_DATA3]=normaliza2(xtest(V_DATA3,:),1);
67  ytest=ytest(V_DATA3,:);
68  %Por ultimo, guardo las variables en un archivo que recibe el ...
        nombre del
69  %dia actual seguido de la hora actual
70  wname=datestr(now);
71  wname=wname([1,2,13,14]);
72  save(wname,'xtrain','ytrain','xval','yval','xtest','ytest',...
73      'min_x','min_ytrain','min_yval','min_ytest','max_x', ...
            'max_ytrain','max_yval','max_ytest','tipo_problema');
```

```
1  %Se introducen los datos como [entradas,salida] y devuelve los ...
       datos
2  %ordenados aleatoriamente y divididos en sets de entrenamiento.
3  function [xtrain,ytrain,xval,yval,xtest,ytest]=split_data(datos)
4  rng('shuffle');
5  N=length(datos);
6  u1=int64(N*6/10);
7  u2=int64(N*8/10);
8  v=datos(randperm(N),:);
9  train=v(1:u1,:);
10 val=v(u1+1:u2,:);
11 test=v(u2+1:end,:);
12
13 xtrain=train(:,1:end-1);
14 xval=val(:,1:end-1);
15 xtest=test(:,1:end-1);
16 ytrain=train(:,end);
17 yval=val(:,end);
18 ytest=test(:,end);
19 end
```

```
1  %Esta funcion divide un problema de clasificacion multiclase en ...
       uno 1 vs
```

```
%all.
%"y" se corresponde con el vector salida inicial mientras que ... "Y" sera una
%matriz con tantas columnas como clases.

%Las clases originales de "y" pueden estar numeradas (1,2,3...) ... o bien ser cadenas de
%caracteres, en cuyo caso se numeraran automaticamente, ... guardando en n el orden de las clases
function [Y,n]=split_output(y)
n=[];
M=length(y);
z=ischar(y);
if ¬z
    Y=zeros(M,max(y));
    %Si la salida es binaria (0,1), transforma dichas clases en ... (1,2).
    if min(min(y))==0
        y=y+1;
    end
end
for ii=1:M
    if z
        a=ismember(y(ii),n); %a es true si y(ii) es una clase ... ya vista (y por tanto metida en n). En ese caso b ... seria el indice de n en el que esta dicha clase.
        if ¬a
            n=[n y(ii)];
        end
        [¬,b]=ismember(y(ii),n);
        Y(ii,b) = 1;
    else
        Y(ii,y(ii))=1;
    end
end
end
```

```
%Pasa varias salidas de clasificacion binaria
%en una de multiclase.
function y=union_output(Y)
[M,N]=size(Y);
for ii=1:M
    for jj=1:N
        if Y(ii,jj)==1
            y(ii,1)=jj;
        end
    end
end
```

```
function [xnew,min_x,max_x,V_DATA]=normaliza2(x,percentiles)
if nargin<2 %Si no se le indica lo contrario, no descarta outliers.
    percentiles=0;
end
```

```
[M,N]=size(x);
%Inicializacion de variables
xnew=[];
V_DATA=ones(M,1);
xprctile03=NaN(N,1);
xprctile97=NaN(N,1);
min_x=NaN(N,1);
max_x=NaN(N,1);
for jj=1:N
    xprctile03(jj)=percentil(x(:,jj),0.05);
    xprctile97(jj)=percentil(x(:,jj),0.95);
    if percentiles==1
        for ii=1:M
            if x(ii,jj)>xprctile97(jj) || x(ii,jj)<xprctile03(jj)
                x(ii,jj)=NaN;
                V_DATA(ii)=0;
                continue %No alimento el algoritmo con outliers
            end
        end
    end
    min_x(jj)=min(x(:,jj));
    aux=x(:,jj)-min_x(jj);
    max_x(jj)=max(aux);
    xnew=[xnew,aux/max_x(jj)];
end
V_DATA=logical(V_DATA);
xnew=xnew(V_DATA,:);

end
```

```
function x=desnormaliza(xnew,max_x,min_x)

[M,N]=size(xnew);
x=NaN(M,N);
for jj=1:N
    x(:,jj)=xnew(:,jj)*max_x(jj)+min_x(jj);
end

end
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] Fernando Sancho Caparrini, *Introducción al aprendizaje automático*, *http://www.cs.us.es/~fsancho/?e=75*, Consultado 03-05-2017.

[2] R.A. Fisher, *Iris species*, *https://www.kaggle.com/uciml/iris*, Recuperado 20-03-2017.

[3] Teodoro Álamo, *Modeling uncertainity. the one-dimensional case.*, Tech. report, Universidad de Sevilla.

[4] _____, *Optimization strategies in systems engineering*, Tech. report, Universidad de Sevilla, 2014.

[5] _____, *Ingeniería de control*, Clase de universidad, 2016.

[6] Teodoro Álamo y J.M. Bravo, *Weighted least squares support vector machines*, Tech. report, Universidad de Sevilla, 2015.

[7] _____, *Interval predictor based on dissimilarity functions*, Tech. report, Universidad de Sevilla, 2016.

[8] Bernard Marr, *A short history of machine learning – every manager should read*, *https://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read/#538507b415e7*, Consultado 11-03-2017.

[9] Stephen Marsland, *Machine learning : an algorithmic perspective*, 2ª ed., Chapman and Hall/CRC, 2014.

[10] Andrew Ng, *Aprendizaje automático*, *https://www.coursera.org/learn/machine-learning/*, Consultado 01-05-2017.

[11] George Box y Gwilym Jenkins, *Chemical concentration readings*, *https://datamarket.com/data/set/232f/chemical-concentration-readings#!ds=232f&display=line*, Recuperado 11-03-2017.

[12] Daniel Rodriguez y Teodoro Álamo, *Identificación mediante el método de los mínimos cuadrados*, *http://control-class.com/Tema_2/Slides/Tema_2_IdentificacionMinimosCuadrados.pdf*, Consultado 03-05-2017.