Aplicación de técnicas de aprendizaje automático para la predicción de la turbidez en plantas de desalinización

UNIVERSIDAD
COMPLUTENSE
MADRID

Samuel Lozano Iglesias

Supervisado por:

Dr. Ángel González Prieto & Dr. Miguel Ruiz García

Facultad de Ciencias Matemáticas

Universidad Complutense de Madrid

Trabajo de Fin de Grado presentado para optar al

Doble Grado en Matemáticas y Física

Junio de 2023

Resumen

El aprendizaje automático es una técnica de inteligencia artificial ampliamente utilizada para la predicción de eventos con dependencia temporal, que en este trabajo sirve para abordar la predicción de la turbidez del agua en plantas de desalinización. Con este fin se estudia uno de los tipos de aprendizaje automático, el aprendizaje supervisado, y se destacan los diferentes objetivos de los modelos para los que es aplicable: los de regresión y los de clasificación. Además, se exponen las redes neuronales artificiales y los mecanismos que permiten su aprendizaje, como la regularización o los algoritmos de optimización. Se detallan también las características de este tipo de redes, diferenciando entre prealimentadas y recurrentes, y se discuten teoremas de aproximación universal que fundamentan su uso. Todos estos elementos se aplican al problema de la turbidez, que se trata de resolver mediante dos modelos, uno de regresión y otro de clasificación. Para ambos se emplea un mismo tipo de red neuronal recurrente: las de memoria a corto-largo plazo. Finalmente, la comparación de los modelos revela que el de clasificación muestra una mayor precisión, a pesar de proporcionar menos información, y resulta suficiente para que se implemente en la planta desaladora.

Palabras clave: *aprendizaje automático*, *regresión*, *clasificación*, *red neuronal*, *turbidez*.

Abstract

Machine learning is an artificial intelligence technique widely used for the prediction of time-dependent events, which we use to address the prediction of water turbidity in desalination plants. To this end, we study one of the types of machine learning, supervised learning, highlighting the different objectives of the models for which it is applicable: regression and classification models. In addition, we present artificial neural networks and the mechanisms that allow them to learn, such as regularisation or optimisation algorithms. The characteristics of this type of networks are also detailed, differentiating between feedforward and recurrent networks, and we discuss universal approximation theorems that underpin their use. All these elements are applied to the turbidity problem, which is solved by means of two models, a regression model and a classification model. Both use the same type of recurrent neural network: long short-term memory. Finally, the comparison of the models reveals that the classification model shows a higher accuracy, despite providing less information, and is sufficient to be implemented in the desalination plant.

Key words: *machine learning*, *regression*, *classification*, *neural network*, *turbidity*.

Índice general

Capítulo 1

Introducción

La predicción meteorológica ha sido un tema de estudio desde comienzos de la civilización humana. En las épocas más primitivas era necesario predecir si iba a llover para, en función de ello, modificar los cultivos o adaptar los periodos de caza. Así, se establecieron reglas sencillas que relacionaban las tonalidades del cielo o las nubes con los cambios en el tiempo. Con la evolución de las sociedades y el método científico, a mediados del siglo XIX se desarrollaron modelos empíricos capaces de anticiparse a eventos temporales adversos, pero no sería hasta principios del siglo XX cuando surgieron los primeros modelos predictivos numéricos [32]. A día de hoy, la predicción del tiempo sigue siendo ampliamente estudiada y mejorada con técnicas modernas.

El problema que se estudia en este trabajo está muy relacionado con la predicción meteorológica, y es de gran importancia para empresas del sector de la potabilización del agua, como es el caso de ACCIONA. La cuestión es la siguiente: *¿puede predecirse con antelación cuál será el grado de turbidez que presente el agua recibida en una planta de desalinización?*

Una de las tareas en las que participé durante mi periodo de prácticas en el departamento de *Knowledge Applied to Business* en ACCIONA fue abordar este problema. Esto es, desarrollar un modelo capaz de estimar con varias horas de antelación el nivel de turbidez que presentará el agua recogida en una de las plantas de la compañía. Un modelo con estas características resulta muy útil para la operatividad de la planta, pues permite evitar problemas derivados de picos extremos de turbidez y realizar tareas de mantenimiento en momentos en los que la producción no es posible.

En un primer momento, el pensamiento científico lleva a asumir que el modelo óptimo se conseguirá identificando las variables externas implicadas y desarrollando una serie de reglas que las relacionen con lo que se quiere predecir. Esta técnica no tiene por qué ser fructífera, y de hecho no lo es en el caso de la turbidez, ya sea por falta de conocimiento, por imposibilidades técnicas o por una cuestión de incertidumbre intrínseca.

Una vez se ha comprobado que la modelización analítica no es posible, surge la intención de aplicar métodos generales de predicción. Esto consiste en deducir cuál será el valor de la turbidez sin llegar a la relación con las variables externas implicadas, o incluso sin conocer dichas variables. Para ello se emplean los modelos de predicción, o modelos predictivos.

Tradicionalmente, los modelos predictivos se han centrado en estudiar series temporales: conjuntos de datos dispuestos cronológicamente e interrelacionados. Este estudio se realiza en función de sus registros históricos (modelos autorregresivos) o de otras variables independientes que podrían afectar a la serie (modelos causales). Sin embargo, estos modelos no son óptimos para analizar grandes volúmenes de datos o con muchos efectos causales entre las variables. Por ello, a medida que la inteligencia artificial ha avanzado, los algoritmos de predicción han adaptado sus técnicas y desarrollado modelos que tienen la capacidad de "aprender": el denominado **aprendizaje automático**.

1.1. Aprendizaje automático

El aprendizaje automático (*Machine Learning*, en inglés) es una rama de la inteligencia artificial basada en el concepto de "aprender": transformar la experiencia en conocimiento. Para ello, los modelos de *Machine Learning* entrenan con una serie de datos iniciales para, posteriormente, predecir un resultado o realizar una tarea. Como el aprendizaje puede desarrollarse de muchas maneras, las características de los modelos son muy distintas en función de la tarea que se quiera resolver, dando paso a áreas diferenciadas del aprendizaje automático.

Dentro de los tipos de aprendizajes que forman parte de la teoría del aprendizaje automático, el foco del Capítulo *2* se centra en uno de ellos: el **aprendizaje supervisado**. En él se describen los principales modelos lineales que componen este tipo de aprendizaje, en función de su objetivo y su complejidad, destacando los modelos de regresión y los de clasificación.

Por otro lado, para introducir la diferencia entre los dos tipos principales de aprendizaje, el supervisado y el no supervisado, puede pensarse en la tarea de detectar valores anómalos en una serie de datos. Un aprendizaje supervisado consistiría en entrenar un modelo a través del estudio de dichos datos, pero añadiendo un indicador si el valor es anómalo, para que el modelo tenga el resultado correcto. En contraposición, un aprendizaje no supervisado se entrenaría recibiendo el volumen de datos sin dicho indicador, teniendo que agrupar los datos en base a ciertas características comunes que infiera y, luego, detectar los valores anómalos.

En el aprendizaje supervisado el conjunto de datos con el que se entrena tiene información suficiente y significativa para realizar la tarea que se requiere. Así, la idea del modelo es que sea capaz de deducir esa información significativa cuando no esté presente, en lo que se denomina una

fase de validación. Por el contrario, el aprendizaje no supervisado tiene información suficiente pero no significativa, teniendo que determinar correlaciones entre los datos para detectar diferencias entre ellos. En este último caso, una fase de entrenamiento es indistinta de una posterior fase que valide si el modelo está funcionando correctamente, pues no se modifican los datos de entrada.

Por último, hay un tipo de aprendizaje intermedio, denominado aprendizaje por refuerzo (del inglés *reinforcement learning*), cuya principal característica es que sí tiene información significativa en la fase de entrenamiento, pero esta no siempre se le proporciona inmediatamente después de realizarse la predicción. De esta manera, el modelo tiene que explorar los datos como el aprendizaje no supervisado pero recibiendo después el *feedback* del aprendizaje supervisado.

En definitiva, cuando se busca una mejora en base a la experiencia, i.e. un aprendizaje, tiene sentido utilizar algoritmos de *Machine Learning* y no modelos de predicción clásicos para realizar las tareas. Estos algoritmos se pueden ejecutar mediante **redes neuronales artificiales**: redes de nodos que simulan el efecto de las neuronas en el cerebro. Dentro de los tipos de redes neuronales destacan las redes prealimentadas y las recurrentes, que se desarrollan en el Capítulo *3*.

Las redes neuronales artificiales presentan ciertas características que se deben definir adecuadamente y estudiar sus posibilidades, como la distribución utilizada para la inicialización, la función de activación o la función de pérdida. Asimismo, en el tercer capítulo se describen algunos algoritmos de optimización y se presentan teoremas de aproximación universal, que aportan la base teórica para la utilización de las redes e indican bajo qué condiciones una función puede aproximarse hasta una cierta precisión con una red neuronal artificial.

Cabe destacar que la elección del algoritmo de aprendizaje y la red neuronal es fundamental, pues no hay un algoritmo óptimo para todos los problemas de un conjunto dado, pero sí hay un algoritmo mejor que los demás para cada problema específico. Esto es lo que demuestran los teoremas de *No Free Lunch*, presentados por David H. Wolpert y William G. Macready en [42].

Por todo ello, desarrollar un modelo predictivo acorde con el paradigma actual implica comprender el problema a tratar, seleccionar el tipo de red neuronal más adecuada para resolverlo y adaptar el aprendizaje al modelo que se desea implementar. Esto es precisamente lo que se ha hecho para predecir la turbidez del agua, entrenando dos modelos: uno de regresión y otro de clasificación.

El desarrollo y los resultados de ambos modelos se presentan en el Capítulo *4* final, incluyendo una comparativa con sus ventajas e inconvenientes. Cabe resaltar que uno de ellos ha demostrado ser lo suficientemente preciso para que ACCIONA lo implemente en su planta de desalinización. Esto ha mejorado la capacidad de planificación de la planta, impactando directamente en su productividad y, por ende, en el suministro de agua a las poblaciones cercanas.

Capítulo 2

Aprendizaje supervisado

El aprendizaje supervisado es un tipo de aprendizaje automático que se basa en aprender comparando las predicciones con los resultados que se deberían haber obtenido. De esta manera, se emplea un conjunto finito de datos de entrada que comúnmente se llama *dataset*, o dominio, X, y que contiene diferentes características relevantes para resolver la tarea. Con él, se intenta estimar el valor de un conjunto finito Y, que incluye una etiqueta para cada elemento de X. Finalmente, el modelo aprende minimizando la diferencia entre las estimaciones y las etiquetas.

Dentro del aprendizaje supervisado hay dos tipos de modelos: los de regresión y los de clasificación. Su estructura y desarrollo es común, pero difieren en algunas características y, principalmente, en su objetivo. Por ello, es necesario identificar el tipo de modelo que se debe utilizar para un problema específico.

Los modelos de regresión pretenden aproximar el valor de una etiqueta dada en el entrenamiento, aunque su valor puede no ser el de ninguna etiqueta correcta. Este podría ser el caso de un modelo que busca predecir la altura de una persona dados ciertos parámetros de entrada, pues el modelo podría dar como resultado un valor predicho que no se corresponde con ninguna de las alturas reales con las que se ha entrenado.

Por otro lado, los modelos de clasificación pretenden deducir cuál de las etiquetas posibles es la correcta, luego la predicción siempre es una de las etiquetas con las que se ha entrenado. Este podría ser el caso de un modelo que busca predecir si un semáforo está en rojo, amarillo o verde, pues el resultado será una de las tres clases.

Además, en los algoritmos que utilizan el aprendizaje supervisado se distinguen dos fases: el entrenamiento y la validación. El entrenamiento es la fase que ya se ha descrito: el modelo recibe el dominio X y el conjunto de etiquetas Y, con el objetivo de realizar estimaciones y compararlas con las etiquetas reales para que aprenda mediante un *feedback* recibido. Sin embargo, en la validación

el modelo recibe un dominio X_V con el que nunca ha entrenado, y no recibe sus etiquetas asociadas Y_V. El objetivo en este caso es predecir el valor de las Y_V para, posteriormente, comparando las estimaciones se comprueba que el modelo es capaz de generalizar y la estimación sigue siendo adecuada cuando se usan datos con los que nunca se ha entrenado.

Formalmente, se considera el conjunto de entrenamiento T como una secuencia finita de pares $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$ en el espacio $X \times Y$. Además, se supone que existe una función desconocida $f : X \to Y$ y se obtiene un estimador de ella, o función de predicción, $\hat{f}$, que se pretende que sea tan similar a f como sea posible. Se busca también que $\hat{f}$ generalice, es decir, que mantenga la similitud con f en la validación. La función de predicción es de la forma $\hat{f} : X \to \mathbb{R}$ si el modelo es de regresión y $\hat{f} : X \to Y$ si es de clasificación, aunque en ocasiones para clasificaciones con K clases se utilizan funciones de predicción $\hat{f} : X \to [0,1]^K$ indicando probabilidades, o incluso funciones generales $\hat{f} : X \to \mathbb{R}$. Esta función de predicción se pretende que sea consistente [27].

Definición 2.0.1. *Una función de predicción se dice que es consistente con la muestra, o conjunto de entrenamiento, $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$, si para cada $i \in \{1, \ldots, N\}$ se tiene que:*

$$\hat{f}(\boldsymbol{x}^i) = y^i =: f(\boldsymbol{x}^i) \, .$$

En definitiva, se debe definir un error que indique la precisión de la predicción y la adecuación del entrenamiento, para intentar minimizarlo a medida que se entrene y, con ello, aprender:

Definición 2.0.2. *Dado un dominio X de cardinal N y el conjunto de etiquetas correctas $Y \subseteq \mathbb{R}$, se define el error de la función de predicción $\hat{f} : X \to \mathbb{R}$ como*

$$\mathcal{E}_{X,Y}(\hat{f}) \coloneqq \frac{1}{N} \sum_{i=1}^{N} d\left(\hat{f}(\boldsymbol{x}^i), y^i\right) \, , \tag{2.1}$$

donde $d(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ es una distancia definida en el espacio métrico $\mathbb{R}$.

Como se puede observar de la definición, el error depende de la distancia escogida en Y, luego hay distintos errores posibles que son más o menos adecuados en función del contexto y el algoritmo a utilizar. Un error destacado por su sencillez es el error empírico, dado por la distancia discreta:

$$d(y^1, y^2) = \begin{cases} 0 & \text{si} \quad y^1 = y^2 \\ 1 & \text{si} \quad y^1 \neq y^2 \end{cases} \quad \Longrightarrow \quad \mathcal{E}_{X,Y}(\hat{f}) \coloneqq \frac{|\{i \in \{1, \ldots, N\} : \hat{f}(\boldsymbol{x}^i) \neq y^i\}|}{N} \, .$$

Este error empírico no siempre es apropiado, pues evalúa por igual todas las predicciones erróneas. Debido a esto, en la práctica se definen errores adaptados a la estructura de las etiquetas Y consideradas. Algunos ejemplos de estos errores se detallan en lo sucesivo y son fundamentales para el desarrollo de los algoritmos de aprendizaje automático, pues estos tratan de minimizarlos y aprenden a partir del *feedback* que aportan los errores.

Cabe destacar que para que un modelo de *Machine Learning* sea útil no solo debe minimizarse su error en el entrenamiento, también debe minimizarse su error en la validación. De hecho, los factores que determinan lo bueno que es un algoritmo de aprendizaje automático son su capacidad para disminuir el error de entrenamiento ("capacidad para aprender") y su capacidad para disminuir la diferencia entre el error de entrenamiento y el de validación ("capacidad para generalizar") [12].

2.1. Regresión lineal múltiple

Los modelos de regresión lineal son los modelos más simples de *Machine Learning* que pueden presentarse para inferir un resultado que se aproxime a las etiquetas correctas. En ellos, se consideran datos de entrada $\boldsymbol{x}^i \in \mathbb{R}^M$ con $i \in \{1, \dots, N\}$, siendo $X = \{\boldsymbol{x}^1, \dots, \boldsymbol{x}^N\}$ el dominio con N observaciones que dependen de M variables de predicción. Este conjunto se combina linealmente mediante una serie de pesos $\tilde{\boldsymbol{w}} = (w_0, \dots, w_M)^T$ para obtener la función de predicción $\hat{f} : X \to \mathbb{R}$. Sea $\boldsymbol{x}^i = (x_1^i, \dots, x_M^i)^T$ el vector de datos asociados a una de las observaciones, entonces:

$$\hat{f}(\boldsymbol{x}^i, \tilde{\boldsymbol{w}}) = w_0 + \sum_{j=1}^{M} w_j x_j^i \, . \tag{2.2}$$

Cabe destacar que en este tipo de modelos, un signo positivo de w_j representa que la variable de predicción j-ésima y las etiquetas correctas tienen correlación directa, mientras que un signo negativo de w_j indica que la correlación es inversa [12].

Para extender este modelo a un problema con una complejidad mayor, se pueden utilizar M funciones $\phi_1(\boldsymbol{x}), \dots, \phi_M(\boldsymbol{x})$ no necesariamente lineales que verifican la relación:

$$\hat{f}(\boldsymbol{x}^i, \tilde{\boldsymbol{w}}) = w_0 + \sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}^i) = \sum_{j=0}^{M} w_j \phi_j(\boldsymbol{x}^i) \, ,$$

donde se ha considerado que $\phi_0(\boldsymbol{x}^i) \coloneqq 1$, para todo $i \in \{1, \dots, N\}$. En realidad, se podrían haber empleado $D \neq M$ funciones $\phi_1(\boldsymbol{x}), \dots, \phi_D(\boldsymbol{x})$, no necesariamente las mismas que el número de variables M, implicando eso que el número de pesos sería $D + 1$.

En cualquier caso, la función de predicción $\hat{f}(\boldsymbol{x}^i, \tilde{\boldsymbol{w}})$ puede no ser lineal si las funciones $\phi_j(\boldsymbol{x}^i)$ no lo son, aunque el modelo sigue siendo de regresión lineal, pues la función de predicción sí es lineal en los pesos $\tilde{\boldsymbol{w}}$. Para $\phi_j(\boldsymbol{x}^i)$ se pueden utilizar diferentes funciones, desde las polinómicas hasta otras más sofisticadas como las gaussianas [4]:

- Polinómicas: $\phi_j(\boldsymbol{x}^i) = \prod_{k=1}^{M} \left(x_k^i\right)^{k_j}$, para todo $j \in \{1, \dots, M\}$.

- Gaussianas: $\phi_j(\boldsymbol{x}^i) = \prod_{k=1}^{M} \exp\left(-\frac{(x_k^i - \mu_k)^2}{2s^2}\right)$, para todo $j \in \{1, \dots, M\}$.

2.1.1. Parámetros de máxima verosimilitud

Tal y como se ha expuesto en la sección anterior, fijando unos pesos $\tilde{\boldsymbol{w}} = (w_0, \ldots, w_M)$, es posible obtener un estimador lineal de la forma (2.2). En esta sección se estudia qué valores deben tomar los parámetros $\tilde{\boldsymbol{w}}$ para que minimicen la diferencia entre la regresión y la etiqueta correcta en cada observación del entrenamiento.

Supóngase entonces que el proceso aleatorio subyacente al problema estudiado es una cierta variable aleatoria $\mathcal{Y}$ que se distribuye de forma normal, con desviación típica σ fija pero media desconocida. Esto es, para todo $\boldsymbol{x}^i \in \mathbb{R}^M$ se tiene que $\mathcal{Y}(\boldsymbol{x}^i) \sim \mathcal{N}\left(f(\boldsymbol{x}^i), \sigma\right)$, con $f : \mathbb{R}^M \to \mathbb{R}$ una función a determinar que condiciona la media de $\mathcal{Y}$. De este modo, las etiquetas $Y = \{y^1, \ldots, y^N\}$ son muestras de la variable aleatoria $\mathcal{Y}$, i.e. $\mathcal{Y}(\boldsymbol{x}^1), \ldots, \mathcal{Y}(\boldsymbol{x}^N)$ con $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$ el dominio.

Minimizar la diferencia entre la función f y la función de predicción $\hat{f}_{\tilde{\boldsymbol{w}}}$ es equivalente a obtener los pesos que maximicen la verosimilitud, donde se ha añadido el subíndice $\tilde{\boldsymbol{w}}$ para enfatizar la dependencia de $\hat{f}$ con los pesos. La función de verosimilitud $\mathcal{L}(\tilde{\boldsymbol{w}})$ se define como el producto de las densidades de probabilidad:

$$\mathcal{L}(\tilde{\boldsymbol{w}}) = \prod_{i=1}^{N} p\left(\hat{f}_{\tilde{\boldsymbol{w}}}(\boldsymbol{x}^i) = y^i\right) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{\left(y^i - \hat{f}_{\tilde{\boldsymbol{w}}}(\boldsymbol{x}^i)\right)^2}{2\sigma^2}\right) . \tag{2.3}$$

Con esto, se demuestra el siguiente resultado:

Teorema 2.1.1. *Sea un conjunto de entrenamiento $T = \{(x_j^i, y^i) : i = 1, \ldots, N; j = 1, \ldots, M\}$, i.e. con N observaciones y M variables de predicción, y $x_0^i = 1$, para todo $i \in \{1, \ldots, N\}$. Tomando $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$ y $Y = \{y^1, \ldots, y^N\}$, de manera que las etiquetas y^i son muestras $\mathcal{Y}(\boldsymbol{x}^i)$ de una variable aleatoria que sigue una normal con desviación típica σ y media desconocida $f(\boldsymbol{x}^i)$. Si la media se estima por una función de predicción lineal:*

$$\hat{f}_{\tilde{\boldsymbol{w}}}(\boldsymbol{x}^i) = \sum_{j=0}^{M} w_j x_j^i = (\boldsymbol{x}^i)^T \cdot \tilde{\boldsymbol{w}} ,$$

entonces existen parámetros $\tilde{\boldsymbol{w}}^*$ de máxima verosimilitud dados por la solución del sistema lineal $(X^T X) \cdot \tilde{\boldsymbol{w}}^* = X^T Y$.

Demostración. (Adaptada de [30]). Considerando la función de verosimilitud $\mathcal{L}(\tilde{\boldsymbol{w}})$, definida en (2.3), los parámetros de máxima verosimilitud $\tilde{\boldsymbol{w}}^*$ buscados se obtienen maximizándola. De esta forma, mediante una transformación a logaritmos neperianos que no modifica los extremos de la función:

$$\log\left(\mathcal{L}(\tilde{\boldsymbol{w}})\right) = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{\left(y^i - (\boldsymbol{x}^i)^T \cdot \tilde{\boldsymbol{w}}\right)^2}{2\sigma^2}\right)\right) =$$
$$= N \cdot \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \sum_{i=1}^{N} \frac{\left(y^i - (\boldsymbol{x}^i)^T \cdot \tilde{\boldsymbol{w}}\right)^2}{2\sigma^2} =$$
$$= N \cdot \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2\sigma^2} \cdot \left(Y - X^T \cdot \tilde{\boldsymbol{w}}\right)^T \cdot \left(Y - X^T \cdot \tilde{\boldsymbol{w}}\right) .$$

Considerando esta relación se tiene que:

$$\mathcal{E}(\tilde{\boldsymbol{w}}) \coloneqq \log\left(\mathcal{L}(\tilde{\boldsymbol{w}})\right) = N \cdot \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2\sigma^2} \cdot \left(Y - X^T \cdot \tilde{\boldsymbol{w}}\right)^T \cdot \left(Y - X^T \cdot \tilde{\boldsymbol{w}}\right) . \tag{2.4}$$

Dado que la expresión obtenida en (2.4) es cuadrática en $\tilde{\boldsymbol{w}}$, su optimización es inmediata a través del gradiente de dicha expresión y se tendrá un máximo global, pues la matriz Hessiana $\nabla^2_{\tilde{\boldsymbol{w}}}\mathcal{E}(\tilde{\boldsymbol{w}}) = -X^T X/\sigma^2$ es semidefinida negativa por ser un producto escalar cambiado de signo:

$$\frac{d\mathcal{E}}{d\tilde{\boldsymbol{w}}} = \frac{-1}{2\sigma^2} \cdot \frac{d}{d\tilde{\boldsymbol{w}}}\left(\left(Y - X^T \cdot \tilde{\boldsymbol{w}}\right)^T \cdot \left(Y - X^T \cdot \tilde{\boldsymbol{w}}\right)\right) =$$
$$= \frac{-1}{2\sigma^2} \cdot \left(Y^T Y - 2Y^T X \cdot \tilde{\boldsymbol{w}} + \tilde{\boldsymbol{w}}^T X^T X \cdot \tilde{\boldsymbol{w}}\right) = \frac{-1}{\sigma^2}\left(-Y^T X + \tilde{\boldsymbol{w}}^T X^T X\right) .$$

Imponiendo que el gradiente anterior sea nulo se alcanza el sistema lineal buscado:

$$\frac{d\mathcal{E}}{d\tilde{\boldsymbol{w}}} = \frac{-1}{\sigma^2}\left(-Y^T X + \tilde{\boldsymbol{w}}^T X^T X\right) = \boldsymbol{0}^T \iff (\tilde{\boldsymbol{w}}^*)^T X^T X = Y^T X \iff (X^T X) \cdot \tilde{\boldsymbol{w}}^* = X^T Y .$$

■

Observación 2.1.2. *La generalización al caso en que se consideran funciones $\phi_j(\boldsymbol{x}^i)$ en lugar de $\boldsymbol{x}^i$ es análogo y puede encontrarse en [9].*

En ciertas situaciones resulta interesante no asumir que la varianza es conocida y estimar la de máxima verosimilitud $(\sigma^*)^2$, dada una cierta serie de pesos $\tilde{\boldsymbol{w}}$. Para ello, siguiendo el desarrollo realizado en la demostración anterior:

$$\frac{\partial\mathcal{E}}{\partial\sigma^2} = \frac{\partial}{\partial\sigma^2}\left(N \cdot \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \sum_{i=1}^{N} \frac{\left(y^i - \boldsymbol{x}^T \cdot \tilde{\boldsymbol{w}}\right)^2}{2\sigma^2}\right) = -\frac{N}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{N}\left(y^i - (\boldsymbol{x}^i)^T \cdot \tilde{\boldsymbol{w}}\right)^2$$

De nuevo, igualando a 0 se obtiene la estimación de máxima verosimilitud de la varianza, que no es más que el promedio de las distancias al cuadrado entre los valores de la función de predicción $\hat{f}_{\tilde{\boldsymbol{w}}}(\boldsymbol{x}^i)$ y las etiquetas y^i:

$$\frac{\partial\mathcal{E}}{\partial\sigma^2} = 0 \implies (\sigma^*)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y^i - (\boldsymbol{x}^i)^T \cdot \tilde{\boldsymbol{w}}\right)^2 \tag{2.5}$$

Observación 2.1.3. *En general, la distribución de probabilidad de la variable aleatoria $\mathcal{Y}$ asociada al proceso puede no ser una normal. En tal caso, se sustituye por la distribución correcta y se obtienen otros parámetros de máxima verosimilitud, como se hace en [29] para la distribución de Bernouilli.*

2.1.2. Regularización

Como se ha visto, se puede determinar la calidad de un modelo de *Machine Learning* en función de su error, que se ha definido de manera general en (2.1). Para dicho error puede considerarse la función (2.4), dada por el logaritmo neperiano negativo de la verosimilitud (2.3) pero ignorando la dependencia en σ^2 por no ser un parámetro libre y despreciando el término constante, i.e.

$$\mathcal{E}_{X,Y}(\tilde{\boldsymbol{w}}) = \frac{1}{2}||Y - X^T \cdot \tilde{\boldsymbol{w}}||^2 \,. \tag{2.6}$$

En un algoritmo de aprendizaje automático, este error tiende a disminuirse modificando los pesos $\tilde{\boldsymbol{w}} = (w_0, \ldots, w_M)^T$ en la fase de entrenamiento. Como el objetivo de un modelo es que sea capaz de predecir también con datos con los que nunca ha entrenado, i.e. generalizar, se puede extender la definición de error del entrenamiento a la fase de validación y buscar que sea bajo. A las técnicas con este propósito se las denomina de "regularización".

Una vez se han obtenido unos pesos $\tilde{\boldsymbol{w}}$ óptimos para la fase de entrenamiento, es probable que se produzca un efecto de sobreajuste (*overfitting*, en inglés). Esto es, el algoritmo tiene un error de entrenamiento bajo pero no es capaz de disminuir el error de validación: *el modelo aprende pero no generaliza*. Un método para evitarlo es penalizar la amplitud de los pesos en un proceso denominado regularización de pesos [27].

Así, se introduce en el error del modelo un parámetro regularizador $\lambda \geq 0$, denominado decaimiento de peso, que va acompañado de la amplitud de los pesos. Para el error cuadrático (2.6) se obtiene la siguiente expresión:

$$\mathcal{E}_{X,Y}(\tilde{\boldsymbol{w}}) = \frac{1}{2}||Y - X^T \cdot \tilde{\boldsymbol{w}}||^2 + \frac{\lambda}{2} \cdot ||\tilde{\boldsymbol{w}}||^2 \,. \tag{2.7}$$

Observación 2.1.4. *La norma utilizada puede ser distinta de la euclídea, pudiendo ser $||\cdot||_p$ para cualquier valor de $p \geq 1$. De hecho, valores menores de p proporcionan soluciones con mayor número de parámetros $w_j = 0$, pero pueden hacer a los modelos menos estables [9].*

En este caso, el valor óptimo $\tilde{\boldsymbol{w}}^*$ de los pesos puede obtenerse aplicando el gradiente en la ecuación (2.7), resultando ser tan solo una variación del óptimo para el caso sin regularizar:

$$\frac{\partial \mathcal{E}}{\partial \tilde{\boldsymbol{w}}} = \frac{\partial}{\partial \tilde{\boldsymbol{w}}} \left(\frac{1}{2} \cdot \left[(Y - X^T \cdot \tilde{\boldsymbol{w}})^T \cdot (Y - X^T \cdot \tilde{\boldsymbol{w}}) + \lambda \tilde{\boldsymbol{w}}^T \cdot \tilde{\boldsymbol{w}} \right] \right) = -Y^T X + \tilde{\boldsymbol{w}}^T X^T X + \lambda \tilde{\boldsymbol{w}}^T$$

$$\Longrightarrow \quad \frac{\partial \mathcal{E}}{\partial \tilde{\boldsymbol{w}}} = \mathbf{0}^T \quad \Longleftrightarrow \quad (\tilde{\boldsymbol{w}}^*)^T \cdot \left(X^T X + \lambda I \right) = Y^T X \quad \Longleftrightarrow \quad \left(\lambda I + X^T X \right) \cdot \tilde{\boldsymbol{w}}^* = X^T Y \,.$$

2.2. Clasificación lineal múltiple

Los modelos de clasificación son, junto con los de regresión, los modelos más comunes dentro del aprendizaje supervisado. Estos se aplican a problemas en los que las etiquetas pertenecen a un conjunto discreto de clases posibles. Así, la idea de estos modelos es que asignan a cada elemento del entrenamiento una probabilidad de pertenecer a cada una de las clases. En la práctica, la mayoría de algoritmos de clasificación se basan en hacer particiones del espacio $\mathbb{R}^M$ tal que cada etiqueta recaiga en una única partición [35].

Considerando un dominio $X \subseteq \mathbb{R}^M$ y un conjunto de etiquetas correctas $Y \subseteq \mathbb{R}$, existe un conjunto discreto de K clases $\mathcal{C} = \{C_1, \ldots, C_K\}$ de manera que, para todo $i \in \{1, \ldots, N\}$, $y^i \in C_k$ para algún $k \in \{1, \ldots, K\}$. Asumiendo el caso habitual en el que las clases son disjuntas, entonces el espacio dado por los datos de entrada puede dividirse en regiones disjuntas $R_1, \ldots, R_K$ denominadas regiones de decisión. Los bordes de estas regiones se denominan superficies de decisión, y en el caso de clasificación lineal son hiperplanos con dimensión $(M-1)$, donde M es la dimensión de X [12].

En general se deben distinguir dos tipos de modelos de clasificación:

- **Modelos bayesianos**: infieren, mediante modelización, la densidad de probabilidad de los datos condicionados a una clase, $p(\boldsymbol{x}^i|C_k)$, así como la de las distintas clases, $p(C_k)$. Con ello y el Teorema de Bayes (véase A.1 del capítulo de apéndices), se calculan las densidades de probabilidad de las clases condicionadas a los datos de entrada, la llamada probabilidad *a posteriori*, $p(C_k|\boldsymbol{x}^i)$:

$$p(C_k|\boldsymbol{x}^i) = \frac{p(\boldsymbol{x}^i|C_k)p(C_k)}{p(\boldsymbol{x}^i)} = \frac{p(\boldsymbol{x}^i|C_k)p(C_k)}{\sum_{l=1}^{K} p(\boldsymbol{x}^i|C_l)p(C_l)} \,. \tag{2.8}$$

  Con dicha probabilidad se determina la clase a la que corresponden los datos de entrada $\boldsymbol{x}^i$. En el caso más simple con dos clases C_1 y C_2, la probabilidad *a posteriori* es de la forma:

$$p(C_1|\boldsymbol{x}^i) = \frac{p(\boldsymbol{x}^i|C_1)p(C_1)}{\sum_{l=1}^{2} p(\boldsymbol{x}^i|C_l)p(C_l)} = \frac{1}{1 + \frac{p(\boldsymbol{x}^i|C_2)p(C_2)}{p(\boldsymbol{x}^i|C_1)p(C_1)}} = \frac{1}{1 + \exp\left(-\log\left(\frac{p(\boldsymbol{x}^i|C_1)p(C_1)}{p(\boldsymbol{x}^i|C_2)p(C_2)}\right)\right)} \,. \tag{2.9}$$

Observación 2.2.1. *Como se describe en la* Sección *3.3.2, esta probabilidad a posteriori no es más que la función logística sigmoide* (3.7) *con* $\alpha = 1$ *y* $a = \log\left(\frac{p(\boldsymbol{x}^i|C_1)p(C_1)}{p(\boldsymbol{x}^i|C_2)p(C_2)}\right)$.

Para el caso en el que hay $K > 2$ clases se puede describir la densidad de probabilidad *a posteriori* mediante una generalización multiclase determinada por la ecuación (2.8):

$$p(C_k|\boldsymbol{x}^i) = \frac{p(\boldsymbol{x}^i|C_k)p(C_k)}{\sum_{l=1}^{K} p(\boldsymbol{x}^i|C_l)p(C_l)} = \frac{\exp\left(\log\left(p(\boldsymbol{x}^i|C_k)p(C_k)\right)\right)}{\sum_{l=1}^{K} \exp\left(\log\left(p(\boldsymbol{x}^i|C_l)p(C_l)\right)\right)} \,. \tag{2.10}$$

Observación 2.2.2. *De nuevo, como se describe en la* SECCIÓN *3.3.2, esta probabilidad a posteriori es la función softmax* (3.8) *considerando* $a_k = \log\left(p(\boldsymbol{x}|C_k)p(C_k)\right)$.

- **Modelos discriminativos**: modelan de manera directa la probabilidad *a posteriori*, $p(C_k|\boldsymbol{x}^i)$, asignando directamente una etiqueta a cada dato de entrada $\boldsymbol{x}^i$. Cabe destacar que lo habitual en un modelo discriminativo con K clases posibles es que se tengan como etiquetas correctas los vectores $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^N$, cada uno de la forma $\boldsymbol{y}^i = (y_1^i, \ldots, y_K^i)^T$ con $y_k^i \in \{0, 1\}$, para todo $k \in \{1, \ldots, K\}$. Esto es, la etiqueta correcta de cada observación es un vector con un 1 en la posición de la clase a la que pertenece y un 0 en el resto, representando la probabilidad de que pertenezca a cada una de las clases.

Modelizar directamente la probabilidad *a posteriori* resulta más útil para los casos prácticos sencillos que modelizar la probabilidad *a priori* para después obtener la probabilidad *a posteriori* mediante la ecuación (2.8). Por este motivo, en este trabajo se analizan únicamente modelos discriminativos. Aun así, para consultar un resultado que describe los parámetros de máxima verosimilitud en modelos bayesianos, puede verse A.2 del capítulo de apéndices.

2.2.1. Máquinas de vector soporte

Comenzando con un análisis similar al propuesto en la SECCIÓN *2.1*, el modelo lineal de clasificación es el modelo discriminativo más simple que puede plantearse. En él, la función de predicción, ahora denominada discriminante, es lineal y se generaliza a $\hat{f} : X \to \mathbb{R}$, siendo $\boldsymbol{w} = (w_1, \ldots, w_M)^T$ el vector de pesos y $\tilde{\boldsymbol{w}} = (w_0, \ldots, w_M)^T$ su ampliación:

$$\hat{f}(\boldsymbol{x}^i, \tilde{\boldsymbol{w}}) = w_0 + \sum_{j=1}^{M} w_j x_j^i = w_0 + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w}\,, \tag{2.11}$$

donde se ha considerado una de las N observaciones $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$ que conforman el dominio.

Considerando también la clasificación lineal más sencilla, en dos clases con valores $+1$ y -1, entonces se puede definir la clase C_1 como las observaciones que cumplen que $\hat{f}(\boldsymbol{x}^i) \geq 0$, y la clase C_2 como las que no lo cumplen. Por ende, la superficie de decisión es una generalización M-dimensional del hiperplano bidimensional dado en la figura 2.1 [4]:

$$H = \{\boldsymbol{x} \in \mathbb{R}^M : \hat{f}(\boldsymbol{x}) = 0\}\,.$$

De hecho, en problemas de clasificación lineal resulta necesario emplear los hiperplanos de decisión para definir el concepto de conjunto de datos linealmente separable:

Definición 2.2.3. *Sea* $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$ *un conjunto de datos de entrada con* $\boldsymbol{x}^i \in \mathbb{R}^M$*, cuyos datos se dividen en* K *clases disjuntas. Entonces se dice que* X *es linealmente separable si sus regiones de decisión pueden separarse por superficies lineales, i.e. hiperplanos con dimensión* $(M-1)$.

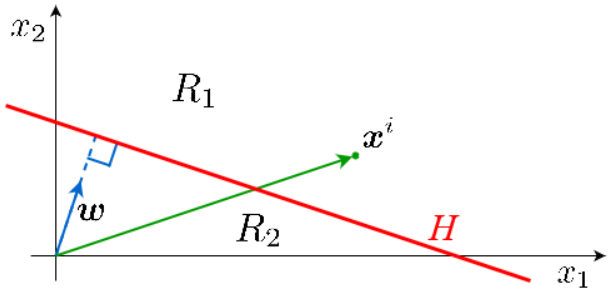


Figura 2.1: *Superficie de decisión en un espacio bidimensional con dos regiones de decisión, R_1 y R_2, separadas por una superficie de decisión, el plano H, y con $\boldsymbol{w}$ el vector de pesos.*

Observación 2.2.4. *En esta sección se trata exclusivamente la clasificación binaria, y la generalización a un número finito K de clases no es trivial. En primer lugar, un planteamiento del clasificador formado por $K-1$ clasificadores de tipo dos clases (la clase considerada frente al resto) lleva a la creación de regiones de ambigüedad [30]. Lo mismo ocurre si se considera el clasificador formado por $K(K-1)/2$ clasificadores de dos clases (pares de posibles clases). Por ello, lo más sencillo es generalizar la función de predicción: $\hat{f}_k(\boldsymbol{x}^i) = w_{k0} + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w}_k$.*

Ahora, en la proposición siguiente se caracteriza geométricamente la superficie de decisión lineal, i.e. su posición y vector normal, así como la distancia de un punto arbitrario a dicho hiperplano.

Proposición 2.2.5. *Dado un modelo lineal de clasificación en dos clases, el vector de pesos $\boldsymbol{w}$ es ortogonal a la superficie de decisión, el término w_0 determina la posición de dicha superficie, y la distancia perpendicular de un punto arbitrario $\boldsymbol{x} \in \mathbb{R}^M$ a la superficie es $|\hat{f}(\boldsymbol{x})|/||\boldsymbol{w}||$.*

Demostración. (Adaptada de [4]). Considerando dos puntos $\boldsymbol{x}^a$ y $\boldsymbol{x}^b$ en la superficie de decisión $\hat{f}(\boldsymbol{x}) = 0$, entonces se tiene que el vector $\boldsymbol{w}$ es ortogonal a ambos, luego es ortogonal a la superficie:

$$w_0 + (\boldsymbol{x}^a)^T \cdot \boldsymbol{w} = w_0 + (\boldsymbol{x}^b)^T \cdot \boldsymbol{w} = 0 \quad \Longrightarrow \quad \left((\boldsymbol{x}^a)^T - (\boldsymbol{x}^b)^T\right) \cdot \boldsymbol{w} = 0\,.$$

Asimismo, considerando un punto $\boldsymbol{x}^c$ en la superficie, la distancia perpendicular del origen a dicha superficie viene dada por:

$$d(O,H) = \frac{\left|(\boldsymbol{x}^c)^T \cdot \boldsymbol{w}\right|}{||\boldsymbol{w}||} = \frac{|-w_0|}{||\boldsymbol{w}||}\,, \quad \text{pues } \hat{f}(\boldsymbol{x}) = w_0 + (\boldsymbol{x}^c)^T \cdot \boldsymbol{w} = 0\,.$$

Por último, la distancia perpendicular entre el hiperplano y el punto $\boldsymbol{x}^d$, tal que $\boldsymbol{x}^d \notin H$, puede deducirse mediante la proyección ortogonal $\boldsymbol{x}^d_\perp$ de $\boldsymbol{x}^d$ sobre la superficie H. De hecho, la proyección ortogonal $\boldsymbol{x}^d_\perp$ se obtiene desplazando el punto $\boldsymbol{x}^d$ una distancia (con signo) $\varrho = \pm d(\boldsymbol{x}^d, H)$ en la dirección normal a H. Esta dirección es la unitaria $\boldsymbol{w}/||\boldsymbol{w}||$, y el signo de ϱ se toma positivo si $\boldsymbol{x}^d$ está por encima del hiperplano de decisión y negativo si $\boldsymbol{x}^d$ está por debajo. De esta manera:

$$\boldsymbol{x}^d_\perp = \boldsymbol{x}^d - \frac{\boldsymbol{w}}{||\boldsymbol{w}||} \cdot \varrho \quad \Longrightarrow \quad \varrho \cdot \frac{\boldsymbol{w}}{||\boldsymbol{w}||} = \boldsymbol{x}^d - \boldsymbol{x}^d_\perp\,.$$

Ahora, se multiplica a ambos lados por $\boldsymbol{w}^T$ y se suma y resta w_0. Además, se considera la relación (2.11) y que $||\boldsymbol{w}||^2 = \boldsymbol{w}^T\boldsymbol{w}$, con lo que se obtiene:

$$\varrho \cdot \frac{\boldsymbol{w}^T\boldsymbol{w}}{||\boldsymbol{w}||} = w_0 + \boldsymbol{w}^T \cdot \boldsymbol{x}^d - w_0 - \boldsymbol{w}^T \cdot \boldsymbol{x}_\perp^d \quad \Longrightarrow \quad \varrho = \frac{\hat{f}(\boldsymbol{x}^d) - \hat{f}(\boldsymbol{x}_\perp^d)}{||\boldsymbol{w}||} .$$

Con esta expresión, como $\boldsymbol{x}_\perp^d \in H$, entonces $\hat{f}(\boldsymbol{x}_\perp^d) = 0$ y se consigue el resultado buscado:

$$\pm d(\boldsymbol{x}^d, H) = \varrho = \frac{\hat{f}(\boldsymbol{x}^d)}{||\boldsymbol{w}||} \quad \Longrightarrow \quad d(\boldsymbol{x}^d, H) = \frac{|\hat{f}(\boldsymbol{x}^d)|}{||\boldsymbol{w}||} .$$

■

Con este concepto de distancia entre el hiperplano y un punto $\boldsymbol{x}$ surge entonces la posibilidad de buscar el hiperplano que "mejor" separa las clases. Esto es, dado un conjunto de entrenamiento $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$ linealmente separable se define la "mejor" separación como aquella que maximiza la distancia entre la superficie de decisión y todos los puntos del dominio X, a lo que se denominará margen. Esta es la base de las máquinas de vector soporte (SVM, del inglés *Support Vector Machines*).

Definición 2.2.6. *Dado un conjunto de datos $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$, el margen r en una máquina de vector soporte con dos clases y estimador $\hat{f}$ se define como:*

$$r = \min_{\boldsymbol{x} \in X} \{d(\boldsymbol{x}, H)\} \qquad con \qquad H = \{\boldsymbol{x} \in \mathbb{R}^M : \hat{f}(\boldsymbol{x}) = 0\} .$$

Observación 2.2.7. *El concepto de margen fue utilizado por Vladimir Vapnik y Alexey Chervonenkis para probar que el aprendizaje es posible cuando el margen es "grande" [39].*

En primer lugar, dado que el vector de pesos $\boldsymbol{w}$ solo es relevante por su dirección, se puede asumir que $||\boldsymbol{w}|| = 1$, por lo que la condición de las máquinas de vector soporte se puede plantear como un problema de optimización, donde los $y^i \in \{+1, -1\}$ son las etiquetas correctas:

$$\begin{aligned} &\text{máx} && \{r\} \\ &\text{s.a} && y^i \cdot \left(w_0 + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w}\right) \geq r , \quad \text{para todo } i \in \{1, \ldots, N\} \\ & && ||\boldsymbol{w}|| = 1 \\ & && r > 0 \end{aligned} \tag{2.12}$$

Alternativamente, se puede definir un factor de escala que asegure que $\hat{f}(\boldsymbol{x}^a) = 1$ si $\boldsymbol{x}^a$ es el punto más cercano al hiperplano. De esta forma, por la proposición 2.2.5, el margen es $r = ||\boldsymbol{w}||^{-1}$ y la condición de las máquinas de vector soporte es la de minimizar el tamaño de los pesos, pues maximizar $||\boldsymbol{w}||^{-1}$ es equivalente a minimizar $||\boldsymbol{w}||$. En definitiva, queda el problema de optimización:

$$\begin{aligned} &\text{mín} \qquad \{||\boldsymbol{w}||\} \\ &\text{s.a} \qquad y^i \cdot \left(w_0 + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w}\right) \geq 1\,, \quad \text{para todo } i \in \{1, \ldots, N\} \end{aligned} \tag{2.13}$$

Ambos problemas (2.12) y (2.13) determinan la utilidad de las máquinas de vector soporte, encargadas de obtener el hiperplano de decisión que maximiza el margen r. Por ello, los dos problemas deben ser equivalentes, como se demuestra en el teorema siguiente:

Teorema 2.2.8. *Maximizar el margen r considerando pesos normalizados es equivalente a minimizar el tamaño de los pesos si se escalan los datos para que el margen sea unitario, i.e. los problemas de optimización* (2.12) *y* (2.13) *son equivalentes.*

Demostración. (Extraída de [9]). Asumiendo (2.12), se puede reparametrizar la ecuación por un vector $\boldsymbol{w'}$ no normalizado de forma que se considera $\boldsymbol{w} \coloneqq \frac{\boldsymbol{w'}}{||\boldsymbol{w'}||}$. Así, como $r > 0$, se obtiene:

$$y^i \cdot \left(w_0 + (\boldsymbol{x}^i)^T \cdot \frac{\boldsymbol{w'}}{||\boldsymbol{w'}||}\right) \geq r \quad \Longrightarrow \quad y^i \cdot \left(\frac{w_0}{r} + (\boldsymbol{x}^i)^T \cdot \frac{\boldsymbol{w'}}{r||\boldsymbol{w'}||}\right) \geq 1\,.$$

Renombrando los parámetros, $w_0'' \coloneqq \frac{w_0}{r}$ y $w'' \coloneqq \frac{\boldsymbol{w'}}{r||\boldsymbol{w'}||}$, con lo que queda:

$$||\boldsymbol{w''}|| = \left|\left|\frac{\boldsymbol{w'}}{r||\boldsymbol{w'}||}\right|\right| = \frac{1}{r} \cdot \left|\left|\frac{\boldsymbol{w'}}{||\boldsymbol{w'}||}\right|\right| = \frac{1}{r} \quad \Longrightarrow \quad r = \frac{1}{||\boldsymbol{w''}||}\,.$$

Sustituyendo esto en (2.12) y teniendo en cuenta que maximizar $||\boldsymbol{w''}||^{-1}$ es equivalente a minimizar $||\boldsymbol{w}||$, se obtiene el problema (2.13) buscado:

$$\min_{\boldsymbol{w''}} \{||\boldsymbol{w''}||\} \quad \text{cumpliendo que} \quad y^i \cdot (w_0'' + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w''}) \geq 1 \quad \text{para todo } i \in \{1, \ldots, N\}\,.$$

Por el contrario, asumiendo (2.13), se considera $r \coloneqq ||\boldsymbol{w}||^{-1}$, que es mayor que 0. Por ende, se puede multiplicar por r la desigualdad impuesta:

$$y^i \cdot (w_0 + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) \geq 1 \quad \Longrightarrow \quad y^i \cdot \left(w_0 + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w}\right) \cdot r \geq r \quad \Longrightarrow \quad y^i \cdot \left(\frac{w_0}{||\boldsymbol{w}||} + (\boldsymbol{x}^i)^T \cdot \frac{\boldsymbol{w}}{||\boldsymbol{w}||}\right) \geq r\,.$$

Definiendo $w_0' \coloneqq \frac{w_0}{||\boldsymbol{w}||}$ y $\boldsymbol{w'} \coloneqq \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$, entonces $||\boldsymbol{w'}|| = 1$ y se obtiene la relación:

$$y^i \cdot \left(\frac{w_0}{||\boldsymbol{w}||} + (\boldsymbol{x}^i)^T \cdot \frac{\boldsymbol{w}}{||\boldsymbol{w}||}\right) \geq r \quad \Longrightarrow \quad y^i \cdot \left(w_0' + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w'}\right) \geq r\,.$$

Finalmente, minimizar $||\boldsymbol{w}||$ es equivalente a maximizar $||\boldsymbol{w}||^{-1}$, luego con las condiciones anteriores se obtiene el problema (2.12) buscado:

$$\max_{r}\{r\} \quad \text{cumpliendo que} \quad y^i \cdot (w_0' + (\boldsymbol{x}^i)^T \cdot \boldsymbol{w'}) \geq r \quad \text{con} \quad ||\boldsymbol{w'}|| = 1 \quad \text{y} \quad r > 0\,.$$

■

2.2.2. Métodos *kernel*

El uso de modelos lineales para la clasificación de un conjunto de datos de entrada no es, en absoluto, óptimo. Tanto es así que los modelos de este tipo se suelen transformar mediante la introducción de una función *kernel*, o de núcleo, que permite mejorar el rendimiento del algoritmo de clasificación. A su vez, utilizar dichas funciones permite extender conjuntos de datos no linealmente separables a espacios en los que sí lo son (véase el ejemplo 2.2.10).

Dado un conjunto de entrenamiento $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$, el problema de clasificación binaria se puede analizar mediante máquinas de vector soporte si los N puntos son linealmente separables. Considerando un espacio M-dimensional y dos clases dadas, lo que se busca es comprobar que una de las posibles particiones de los N puntos de T en dos clases es linealmente separable. Así, la introducción de funciones *kernel* está basada en que una partición de N puntos en un espacio M-dimensional tiene mayor probabilidad de ser linealmente separable cuanto mayor sea M.

Para comprobar esta afirmación se define $C(N, M)$ como el número de particiones de N puntos que son linealmente separables en un espacio M-dimensional, y se busca que $C(N, H) > C(N, M)$ si $H > M$, es decir, el número de particiones linealmente separables aumenta con la dimensión. Esto es precisamente lo que demuestra en el Teorema de Cover siguiente [7]:

Teorema 2.2.9. *Un problema de clasificación de N puntos en dos clases en un espacio de dimensión M, con $N > M$, que se proyecta por una transformación no lineal en otro espacio de dimensión H, con $H > M$, es más probable que sea linealmente separable que si se proyecta en un espacio de dimensión H con $H < M$.*

Demostración. (Adaptada de [28]). Por inducción, supóngase que se tienen las particiones linealmente separables de N puntos y se añade uno más:

1. Si existe un hiperplano que separaba a los N puntos y pasa por el punto nuevo, entonces el hiperplano puede desplazarse infinitesimalmente hacia cualesquiera de sus dos direcciones perpendiculares para incluir al nuevo punto en alguna de las clases. Así, por cada partición separable de los N puntos que pase por el nuevo existen dos particiones linealmente separables posibles, con el nuevo punto en cada una de las dos regiones. Se denota el número de particiones de este tipo para N puntos como $C_1(N, M)$.

2. Si un hiperplano que separaba a los N puntos no pasa por el punto nuevo, entonces el hiperplano sigue separando linealmente a los $N + 1$ puntos y se ha obtenido una partición para los $N + 1$ puntos a partir de la de los N puntos, simplemente incluyendo el nuevo punto en la región correspondiente. Se denota el número de particiones de este tipo para N puntos como $C_2(N, M)$.

De esta forma, para obtener $C(N+1, M)$ se deben contar las particiones linealmente separables del primer tipo dos veces, y las del segundo tipo una vez. Esto no es más que considerar que

$$C(N+1, M) = 2 \cdot C_1(N, M) + C_2(N, M) = C(N, M) + C_1(N, M) \, .$$

De hecho, $C_1(N, M) = C(N, M-1)$, pues restringir que el hiperplano pase por un punto equivale a quitar un grado de libertad, i.e. proyectar los N puntos sobre un espacio $M-1$ dimensional. Así:

$$C(N+1, M) = C(N, M) + C(N, M-1) \, .$$

Ahora, se aplica esta relación recursivamente y se tiene en cuenta que $C(1, k) = 0$ si $k < 1$ y $C(1, k) = 2$ si $k \geq 1$.

- Aplicando la relación una vez:

$$C(N+1, M) = C(N, M) + C(N, M-1) \, .$$

- Aplicando la relación dos veces:

$$\begin{aligned} C(N+1, M) &= C(N-1, M) + C(N-1, M-1) + C(N-1, M-1) + C(N-1, M-2) = \\ &= C(N-1, M) + 2C(N-1, M-1) + C(N-1, M-2) \, . \end{aligned}$$

- Aplicando la relación tres veces:

$$\begin{aligned} C(N+1, M) = C(N-2, M) &+ C(N-2, M-1) + 2C(N-2, M-1) + 2C(N-2, M-2) + \\ &+ C(N-2, M-2) + C(N-2, M-3) = \\ &= C(N-2, M) + 3C(N-2, M-1) + 3C(N-2, M-2) + C(N-2, M-3) \, . \end{aligned}$$

- Aplicando la relación N veces:

$$C(N+1, M) = C(1, M) + N \cdot C(1, M-1) + \binom{N}{2} C(1, M-2) + \ldots + C(1, M-N) \, .$$

En definitiva, se obtiene que, como se considera $N > M$:

$$C(N+1, M) = 2 \cdot \sum_{j=0}^{M-1} \binom{N}{j} \, .$$

Por ende, para un número fijo de puntos $N+1$ se tendrá un mayor número de particiones linealmente separables cuanto mayor sea el valor de M, i.e. para $H > M$ se tiene mayor probabilidad que para $H < M$ de encontrar el hiperplano buscado. ■

Con este resultado, el aprendizaje de los modelos lineales descritos en la SECCIÓN *2.2* puede mejorarse si se extiende a espacios con mayor dimensión a través de funciones no lineales. Así, dado un dominio X de dimensión M, se escoge una función $\phi : X \to \mathcal{H}$, donde $\mathcal{H}$ se denomina espacio de características (*feature space*, en inglés) y es un espacio de Hilbert, i.e. un espacio vectorial con un producto escalar y una norma asociada, que además es completo con respecto a dicha normal. Esta función permite transformar el conjunto de entrenamiento inicial $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$ en $T_\phi = \{(\phi(\boldsymbol{x}^1), y^1), \ldots, (\phi(\boldsymbol{x}^N), y^N)\}$, con el que se entrena la función de predicción, $\hat{f}$, y se obtiene la predicción de los datos de entrada como $\hat{f}(\phi(\boldsymbol{x}^i))$. Además, la función de pérdida se transforma trivialmente como $\mathcal{E}_{X,\phi}(\hat{f}) = \mathcal{E}_X(\hat{f} \circ \phi)$.

La función ϕ debe ser escogida adecuadamente, pues no todas las extensiones a espacios de dimensión superior son útiles, como ilustra el siguiente ejemplo, representado en la figura 2.2.

Ejemplo 2.2.10. *El dominio $X = \{-2, -1, 0, 1, 2\}$, donde $x_3 = 0$ tiene la etiqueta 0 y el resto tienen la etiqueta 1, no es linealmente separable por un hiperplano en $\mathbb{R}$, i.e. por un punto. Si se extiende a $\mathbb{R}^2$ mediante la función $\phi_1 : \mathbb{R} \to \mathbb{R}^2$ dada por $\phi_1(x) = (x, x^2)$, entonces $X_{\phi_1} = \{\phi_1(-2), \phi_1(-1), \phi_1(0), \phi_1(1), \phi_1(2)\}$ sí es linealmente separable, pues un hiperplano separador en $\mathbb{R}^2$ sería la recta horizontal que pasa por el punto $(0, 1/2)$. Sin embargo, la extensión $\phi_2 : \mathbb{R} \to \mathbb{R}^2$ dada por $\phi_1(x) = (x, x^3)$ no sería adecuada por no transformar al conjunto en linealmente separable.*

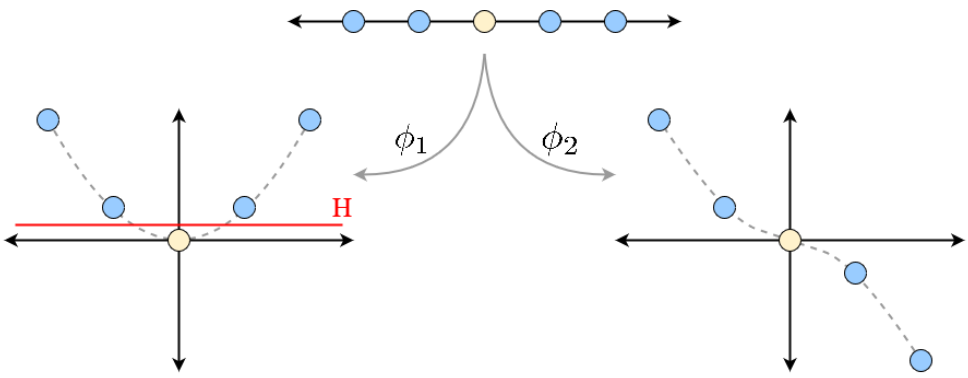


Figura 2.2: *Dominio en $\mathbb{R}$ no separable linealmente que al transformarse a $\mathbb{R}^2$ por $\phi_1(x) = (x, x^2)$ sí es linealmente separable, pero por $\phi_2(x) = (x, x^3)$ no lo es.*

Computacionalmente es muy costoso definir explícitamente la extensión ϕ y realizar operaciones con ella en espacios con dimensión mayor. Por ello, en la práctica se definen funciones *kernel* mediante el producto escalar en el espacio de Hilbert considerado:

$$\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) = \langle \phi(\boldsymbol{x}^i), \phi(\boldsymbol{x}^j) \rangle_{\mathcal{H}} \,. \tag{2.14}$$

Ejemplo 2.2.11. *Si $\mathcal{H} = \mathbb{R}^H$, entonces: $\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) = \langle \phi(\boldsymbol{x}^i), \phi(\boldsymbol{x}^j) \rangle_{\mathbb{R}^H} = \phi(\boldsymbol{x}^i)^T \cdot \phi(\boldsymbol{x}^j)$.*

Estas funciones *kernel* pueden interpretarse como funciones de similaridad, entendida la similaridad por la que definen de manera implícita las extensiones, pero que son simplemente funciones $\mathcal{K} : X \times X \to \mathbb{R}$. Un ejemplo de funciones con estas características son los *kernels* polinomiales de orden k, presentados en 2.2.12.

Ejemplo 2.2.12. *Los kernels polinomiales de orden k son funciones $\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) = ((\boldsymbol{x}^i)^T \cdot \boldsymbol{x}^j)^k$. Se puede probar de forma sencilla que, efectivamente, existen extensiones $\phi : \mathbb{R}^M \to \mathbb{R}^H$ que cumplen la relación* (2.14)*, con $H = M^k$:*

$$\begin{aligned}\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) = ((\boldsymbol{x}^i)^T \cdot \boldsymbol{x}^j)^k = ((\boldsymbol{x}^i)^T \cdot \boldsymbol{x}^j) \cdot \ldots \cdot ((\boldsymbol{x}^i)^T \cdot \boldsymbol{x}^j) = \\ = \left(\sum_{l=1}^{M} x_l^i x_l^j\right) \cdot \ldots \cdot \left(\sum_{l=1}^{M} x_l^i x_l^j\right) = \sum_{J \in \{1,\ldots,M\}^k} \prod_{m=1}^{k} x_{J_m}^i x_{J_m}^j = \\ = \sum_{J \in \{1,\ldots,M\}^k} \prod_{m=1}^{k} x_{J_m}^i \prod_{m=1}^{k} x_{J_m}^j \quad \Longrightarrow \quad \phi(\boldsymbol{x}) = \prod_{m=1}^{k} x_{J_m} \quad con \quad \phi : \mathbb{R}^M \to \mathbb{R}^{M^k}\,.\end{aligned}$$

2.2.2.1. Métodos *kernel* en máquinas de vector soporte

Como se ha comentado, la aplicación de los métodos *kernel* a las máquinas de vector soporte no solo implica una mejora en el algoritmo de clasificación, sino que permite clasificar conjuntos que previamente no eran clasificables. Una condición necesaria para que una máquina de vector soporte lineal pudiese clasificar un conjunto de entrenamiento $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$ era que fuese linealmente separable, como se indicó en la Sección *2.2.1*. Con la introducción de funciones $\phi : X \to \mathcal{H}$, esta condición deja de ser necesaria, para pasar a que la condición que se imponga sea que el conjunto de entrenamiento $T_\phi = \{(\phi(\boldsymbol{x}^1), y^1), \ldots, (\phi(\boldsymbol{x}^N), y^N)\}$ sea linealmente separable.

Por otro lado, las funciones *kernel* $\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) = \langle \phi(\boldsymbol{x}^i), \phi(\boldsymbol{x}^j) \rangle_{\mathcal{H}}$ no surgen artificialmente, sino de considerar los problemas de optimización (2.12) y (2.13) que determinan las máquinas de vector soporte. En particular, se utiliza una adaptación del problema (2.13) pero considerando la minimización del cuadrado en lugar de la maximización del inverso, i.e. un problema de optimización equivalente dado por[1]:

$$\begin{aligned} &\text{mín} && \left\{\frac{1}{2}||\boldsymbol{w}||^2\right\} \\ &\text{s.a} && y^i \cdot (w_0 + \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) \geq 1 \quad \text{para todo } i \in \{1, \ldots, N\} \end{aligned} \tag{2.15}$$

Proposición 2.2.13. *Dado un problema de clasificación binaria con clases $+1$ y -1, y un conjunto de entrenamiento $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$ con $\boldsymbol{x}^i \in \mathbb{R}^M$ para todo $i \in \{1, \ldots, N\}$. Suponiendo una extensión a un espacio real de dimensión finita $H > M$, $\phi : \mathbb{R}^M \to \mathbb{R}^H$, entonces la función de predicción $\hat{f} : \mathbb{R}^M \to \mathbb{R}$ obtenida por la máquina de vector soporte es:*

[1]Se multiplica por un factor de escala $1/2$ que no afecta a la optimización [26].

$$\hat{f}(\boldsymbol{x}) = \frac{1}{|\mathcal{G}|} \cdot \left[\sum_{i \in \mathcal{G}} y^i - \sum_{i,j \in \mathcal{G}} u_j y^j \mathcal{K}(\boldsymbol{x}^j, \boldsymbol{x}^i) \right] + \sum_{i \in \mathcal{G}} u_i y^i \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^i) \,, \tag{2.16}$$

donde los $u_i \geq 0$ *para todo* $i \in \{1, \ldots, N\}$ *y* $\mathcal{G}$ *es el conjunto de las restricciones activas, i.e.* $j \in \mathcal{G}$ *si* $u_j \neq 0$. *Además, se ha considerado el kernel* $\mathcal{K} : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ *con* $\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) = \langle \phi(\boldsymbol{x}^i), \phi(\boldsymbol{x}^j) \rangle_{\mathbb{R}^H}$.

Demostración. Se considera el producto escalar usual en $\mathbb{R}^H$: $\langle \phi(\boldsymbol{x}^i), \phi(\boldsymbol{x}^j) \rangle_{\mathbb{R}^H} = \phi(\boldsymbol{x}^i)^T \cdot \phi(\boldsymbol{x}^j)$. Sea el problema de optimización (2.15), considerando ahora las condiciones KKT, de Karush-Kuhn-Tucker (véase A.3 del capítulo de apéndices), con $\boldsymbol{u} = (u_1, \ldots, u_N) \in \mathbb{R}$, la función lagrangiana es de la forma:

$$L(w_0, \boldsymbol{w}, \boldsymbol{u}) = \frac{1}{2} ||\boldsymbol{w}||^2 + \sum_{i=1}^{N} u_i \cdot \left[1 - y^i \cdot (w_0 + \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) \right] \,.$$

Derivando dicha relación con respecto a $\boldsymbol{w}$ y w_0 e igualando ambas a cero, se obtienen dos de las condiciones a cumplir para la minimización:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{N} u_i y^i \phi(\boldsymbol{x}^i) = 0 \quad \Longrightarrow \quad \boldsymbol{w} = \sum_{i=1}^{N} u_i y^i \phi(\boldsymbol{x}^i) \,.$$

$$\frac{\partial L}{\partial w_0} = - \sum_{i=1}^{N} u_i y^i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{N} u_i y^i = 0 \,.$$

Sustituyendo estos resultados en la función de predicción (2.11) modificada por $\phi(\boldsymbol{x}^i)$:

$$\hat{f}(\boldsymbol{x}) = w_0 + \phi(\boldsymbol{x})^T \cdot \boldsymbol{w} = w_0 + \phi(\boldsymbol{x})^T \cdot \sum_{i=1}^{N} u_i y^i \phi(\boldsymbol{x}^i) = w_0 + \sum_{i=1}^{N} u_i y^i \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}^i) \,. \tag{2.17}$$

Además, como se tiene que cumplir la relación $u_i \cdot \left[1 - y^i \cdot (w_0 + \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) \right] = 0$ para todo $i \in \{1, \ldots, N\}$, entonces las condiciones activas de $\mathcal{G}$, i.e. los j tales que $u_j \neq 0$, serán las que satisfagan que $y^i \cdot (w_0 + \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) = 1$, permitiendo obtener el valor de w_0:

$$y^i \cdot (w_0 + \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) = 1 \quad \Longrightarrow \quad (y^i)^2 \cdot (w_0 + \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w}) = y^i \quad \Longrightarrow \quad w_0 = y^i - \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w} \quad \Longrightarrow$$
$$\Longrightarrow \quad |\mathcal{G}| \cdot w_0 = \sum_{i \in \mathcal{G}} y^i - \phi(\boldsymbol{x}^i)^T \cdot \boldsymbol{w} \,.$$

De esta forma, empleando la relación para $\boldsymbol{w}$ en función de los u_i se tiene:

$$w_0 = \frac{1}{|\mathcal{G}|} \cdot \left[\sum_{i \in \mathcal{G}} y^i - \sum_{i \in \mathcal{G}} \phi(\boldsymbol{x}^i)^T \boldsymbol{w} \right] = \frac{1}{|\mathcal{G}|} \cdot \left[\sum_{i \in \mathcal{G}} y^i - \sum_{i,j \in \mathcal{G}} u_j y^j \mathcal{K}(\boldsymbol{x}^j, \boldsymbol{x}^i) \right] \,.$$

Finalmente, el otro sumatorio de la ecuación (2.17) se pueden transformar al conjunto de índices $\mathcal{G}$, pues $u_j = 0$ si $j \notin \mathcal{G}$, quedando la relación buscada. ■

Observación 2.2.14. *La proposición 2.2.13 puede generalizarse al caso $\phi : \mathbb{R}^M \to \mathcal{H}$ con $\mathcal{H}$ un espacio de Hilbert, sustituyendo también el producto escalar por el que corresponda en $\mathcal{H}$. La demostración de la proposición generalizada puede consultarse en [26].*

La modificación de los máquinas de vector soporte mediante funciones *kernel* resulta de especial utilidad porque la función de predicción $\hat{f}$ tiene un coste computacional menor. Esto se debe a que en su cálculo solo se consideran las restricciones activas $u_j \neq 0$, o, equivalentemente, los datos de entrenamiento que cumplan $y^j \cdot \left(w_0 + \phi(\boldsymbol{x}^j)^T \cdot \boldsymbol{w}\right) = 1$.

En definitiva, el problema de optimización (2.15) se transforma entonces en el problema de optimización siguiente, que depende de los parámetros $u_i \geq 0$ con $i \in \{1, \ldots, N\}$ y cuya resolución se puede llevar a cabo mediante el algoritmo SMO [26]:

$$\begin{aligned} \text{máx} \quad & \left\{ \sum_{i=1}^{N} u_i - \frac{1}{2} \sum_{i,j=1}^{N} u_i u_j y^i y^j \mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j) \right\} \\ \text{s.a} \quad & \sum_{i=1}^{N} u_i y^i = 0 \\ & u_i \geq 0\,, \quad \text{para todo } i \in \{1, \ldots, N\} \end{aligned} \tag{2.18}$$

Por último, cabe destacar que no todos los *kernel* son válidos porque no todos llevarán a problemas de optimización bien definidos. De hecho, los *kernel* adecuados son los que satisfacen el Teorema de Mercer siguiente, cuya demostración puede encontrarse en [25].

Teorema 2.2.15. *Una función $\mathcal{K} : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ es un kernel válido si y solo si, para cualquier $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\} \subseteq \mathbb{R}^M$, la matriz $K = \left(\mathcal{K}(\boldsymbol{x}^i, \boldsymbol{x}^j)\right)_{i,j=1}^{N}$ es simétrica y semidefinida positiva.*

Capítulo 3

Redes neuronales artificiales

Con el fin de emular el funcionamiento del cerebro humano, a mediados del siglo XX se comenzaron a desarrollar computacionalmente las redes neuronales artificiales (ANN, del inglés *Artifical Neural Network*), destacando el trabajo de Frank Rosenblatt, que implementó por primera vez una red neuronal artificial en [31].

Una red neuronal artificial es un modelo diseñado para simular el procesamiento cerebral de una tarea determinada, a través de un proceso de "aprendizaje" que emplea células computacionales interconectadas, denominadas neuronas artificiales. El "conocimiento" se adquiere a través de un algoritmo de aprendizaje, como es el caso del aprendizaje supervisado descrito en el Capítulo *2*, y se almacena en las neuronas mediante pesos sinápticos, que determinan las conexiones interneuronales.

Las redes neuronales artificiales tienen propiedades fundamentales que aumentan su utilidad frente a programas de computación o neuronas individuales [13]:

- **Generalización**: las redes neuronales pueden producir resultados similares a los de la fase de aprendizaje, pero con datos de entrada distintos, lo que permite que realicen predicciones.
- **No linealidad**: dada su estructura y construcción, las redes permiten desarrollar algoritmos complejos con propiedades no lineales que describan sistemas físicos y reales con gran precisión.
- **Adaptatividad**: los pesos sinápticos de la red pueden modificarse a medida que el modelo se reentrena, pudiendo adaptarse a sistemas no estacionarios.

Una neurona artificial, o nodo, es la unidad fundamental que compone una red neuronal artificial y que está formada por un conjunto de conexiones con pesos sinápticos, un combinador lineal y una función de activación no necesariamente lineal que modifica el valor de salida y limita su rango. Puede incluirse también un sesgo que altere el valor de salida del combinador lineal, antes de utilizarse la función de activación.

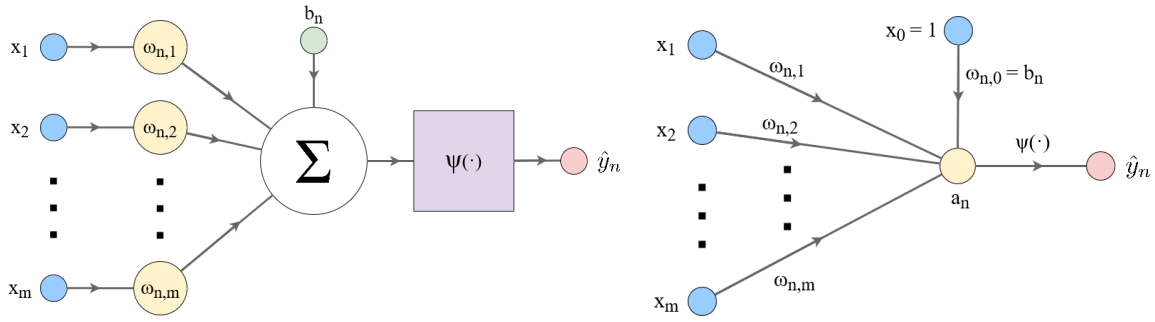
De manera formal, para una neurona n se tienen valores de entrada $x_j \in \mathbb{R}$ con $j \in \{1, \ldots, m\}$ y m el número de conexiones incidentes a la neurona. Además, los pesos sinápticos $w_{n,j} \in \mathbb{R}$ están asociados a la neurona n y a la conexión de entrada j, mientras que el combinador lineal produce la activación $a_n \in \mathbb{R}$, i.e. el valor de salida anterior a la función de activación:

$$a_n = \sum_{j=1}^{m} w_{n,j} x_j \ .$$

A este se le añade un posible sesgo $b_n \in \mathbb{R}$ y se le aplica una función de activación $\psi : \mathbb{R} \to \mathbb{R}$, luego la salida de la neurona $\hat{y}_n$ es (véase la figura 3.1a):

$$\hat{y}_n = \psi(a_n + b_n) = \psi\left(\sum_{j=1}^{m} w_{n,j} x_j + b_n\right) \ . \tag{3.1}$$

En realidad el sesgo puede interpretarse como una entrada más, con valor $x_0 = 1$ y peso sináptico $w_{n,0} = b_n$. Asimismo, esta representación de la neurona no es más que un grafo (véase A.4 del capítulo de apéndices, que incluye los conceptos empleados de teoría de grafos) orientado simple con $m + 3$ vértices, los correspondientes a las $m + 1$ variables de entrada (incluyendo el sesgo), el vértice operador y el vértice de salida (véase la figura 3.1b). Presenta $m + 2$ arcos: $m + 1$ arcos sinápticos con pesos $w_{n,j}$ y el arco de activación restante que aplica la función de activación $\psi(\cdot)$.



(a) *Modelo no lineal.* (b) *Grafo.*

Figura 3.1: *Representaciones de la neurona n.*

Con esta interpretación de las neuronas como grafos se llega al concepto de red neuronal completa, que puede simplificarse a una red neuronal parcialmente completa si cada neurona se considera como un único vértice y el arco de activación se sobreentiende (véase la figura 3.2). Además, en estas redes los sesgos se incluyen en los nodos fuente como valores de entrada de cada neurona.

Definición 3.0.1. *Una red neuronal parcialmente completa es un grafo orientado $G = \{V, A\}$ con $|V|$ el número de vértices y $|A|$ el número de arcos, donde $V = V_s \cup V_n$ con $|V_s|$ el número de nodos fuente y $|V_n|$ el número de neuronas. Además, los vértices V_s cumplen que no tienen predecesores pero tienen al menos un sucesor y los vértices V_n deben tener al menos un predecesor.*

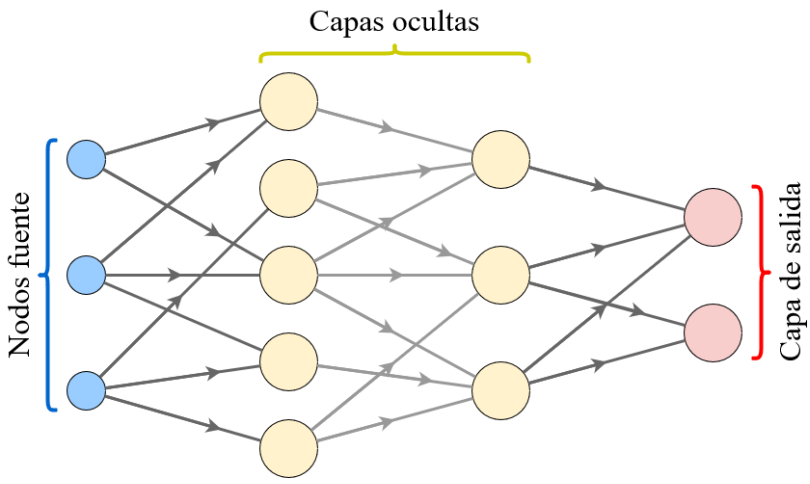


Figura 3.2: *Representación como grafo parcialmente completo de una red neuronal.*

Observación 3.0.2. *El concepto de red neuronal completa no tiene nada que ver con que el grafo que la representa sea, o no, completo. De hecho, una red neuronal completa puede definirse también como un grafo orientado* $G = \{V, A\}$ *que cumple que* $V = V_s \cup V_{n,w} \cup V_{n,\psi}$ *y* $A = A_w \cup A_\psi$*, con* $|V_{n,w}| = |V_{n,\psi}|$*,* $|A_w|$ *el número de arcos sinápticos y* $|A_\psi|$ *el número de arcos de activación. Además, los vértices* $V_{n,w}$ *tienen un único sucesor que pertenece a* $V_{n,2}$ *y los vértices* $V_{n,\psi}$ *tienen un único predecesor que pertenece a* $V_{n,w}$*. Los arcos incidentes a* $V_{n,w}$ *son arcos sinápticos y los arcos incidentes a* $V_{n,\psi}$ *son arcos de activación.*

Por lo general, los V_s que no son sesgos pertenecen a la capa de entrada, los V_n que no tienen sucesores pertenecen a la capa de salida y el resto de V_n pertenecen a las capas ocultas. Se define como Z la profundidad de la red, i.e. el número de capas excluyendo la capa con los nodos fuente, mientras que el ancho de la red es $\max_{z\in\{1,\dots,Z\}} |V_z|$. Por último, el tamaño de la red no es más que $|V|$ el número de neuronas y nodos fuente. Estos parámetros se denominan la arquitectura de la red neuronal, y, como ejemplo, la red de la figura 3.2 tiene tamaño 13, profundidad 3 y anchura 5.

Una capa se dice completamente conectada (*fully connected layer*, en inglés) si cada una de sus neuronas está conectada con todas la neuronas de su capa anterior y todas las de su capa posterior. Si esto ocurre para todas las capas de la red, se tiene una red neuronal completamente conectada, i.e. *fully connected neuronal network*.

El uso de este tipo de redes es muy habitual porque no asume una cierta disposición o propiedades de la red, todos los valores de entrada afectan a todas las neuronas y a todos los valores de salida, siendo la propia red la que decide qué conexiones son relevantes y cuáles no. En su defecto, estas redes tienen una convergencia más difícil que otras redes en las que algunas conexiones no son posibles, pues dependen de un mayor número de parámetros [12].

Las redes neuronales descritas como grafos pueden clasificarse en dos tipos: las que no presentan ciclos y las que sí los presentan. Una red como la de la figura 3.2 se denomina prealimentada, pues sus arcos inciden únicamente a neuronas en capas posteriores, mientras que si existiesen ciclos, i.e. hay arcos incidentes a neuronas en capas anteriores, la red es recurrente.

Una vez la red ha obtenido unos valores de salida, resulta necesario definir un método para validar que la red ha mejorado y ha "adquirido conocimiento". Para ello, se define un error total de la red para un entrenamiento que se pretende minimizar: se tiene una función de pérdida dependiente de los pesos sinápticos que debe minimizarse y que está muy relacionada con las funciones de activación de la capa de salida [30]. Esto no es más que lo que se describió en el CAPÍTULO *2* como error del aprendizaje supervisado, pero aplicado al conjunto de la red neuronal.

Volviendo a los conceptos desarrollados para el aprendizaje supervisado, se considera un entrenamiento que pretende minimizar la función de pérdida $\mathcal{E}(\boldsymbol{w})$ dada, donde $\boldsymbol{w}$ es un vector con los pesos de la red neuronal. Un mínimo local de la función de pérdida es un valor $\boldsymbol{w}^*$ tal que $\nabla\mathcal{E}(\boldsymbol{w}^*) = 0$, pero en general no se pueden encontrar soluciones analíticas de dicha relación. Por ello, es útil recurrir a métodos numéricos de la forma:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{\Delta}^{(t)}\,, \quad \text{con } t = 1, 2, \ldots \tag{3.2}$$

donde t indica el paso iterativo, $\boldsymbol{w}^{(0)}$ es un vector inicial de pesos convenientemente escogido (véase la SECCIÓN *3.3.1*) y $\boldsymbol{\Delta}^{(t)}$ es la modificación de $\boldsymbol{w}^{(t)}$ en cada iteración t.

Observación 3.0.3. *Pese a que lo que se busca al entrenar una red neuronal es encontrar el mínimo global de la función de pérdida, en la práctica valdrá con encontrar varios mínimos locales y compararlos, pues ni siquiera la existencia de un mínimo global está asegurada [4].*

3.1. Optimización por descenso de gradiente

La forma más sencilla de iterar la relación (3.2) es considerando $\boldsymbol{\Delta}^{(t)} = -\eta \cdot \nabla\mathcal{E}\left(\boldsymbol{w}^{(t)}\right)$, con $\eta > 0$ la "razón de aprendizaje". Esto es simplemente considerar un pequeño desplazamiento en la dirección en la que el error disminuye, i.e un paso en la dirección negativa del gradiente. Esta técnica es conocida como descenso de gradiente, y no es computacionalmente efectiva porque, dado que la función de pérdida se define para un conjunto de entrenamiento $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$, el método implica evaluar el gradiente de la función de pérdida en el conjunto completo.

Así, para redes neuronales que se entrenan con conjuntos T grandes, i.e. $N \gg 1$, se considera un método de descenso de gradiente estocástico que itera sobre el peso en cada elemento de T:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \cdot \nabla\mathcal{E}_i\left(\boldsymbol{w}^{(t)}\right) \quad \text{con} \quad \mathcal{E}(\boldsymbol{w}) = \sum_{i=1}^{N} \mathcal{E}_i(\boldsymbol{w})\,, \tag{3.3}$$

donde $\mathcal{E}_i(\boldsymbol{w})$ es la función de pérdida que se obtendría si se considerase el conjunto de entrenamiento $T_i = \{(\boldsymbol{x}^i, y^i)\}$ para un i dado. Equivalentemente, $\mathcal{E}_i(\boldsymbol{w})$ es la aportación del dato de entrada $\boldsymbol{x}^i$ a la función de pérdida $\mathcal{E}(\boldsymbol{w})$.

Esta modificación permite simplificar computacionalmente el algoritmo de optimización al requerir un menor número de operaciones por iteración. Además, el método estocástico es más efectivo para conjuntos de entrenamiento que tienen elementos repetidos o muy similares, y evita también que la optimización se quede atrapada en un mínimo local.

Una modificación al método de descenso de gradiente estocástico consiste en introducir "lotes" (*batches*, en inglés), que son subconjuntos del entrenamiento T sobre los que se realiza una iteración. Esto es, en el descenso de gradiente por lotes se consideran subconjuntos $T_1, \ldots, T_B \subseteq T$ y se modifica la ecuación (3.3) para que en cada iteración t se utilice la función de pérdida obtenida para uno de esos lotes, $\mathcal{E}_{T_b}(\boldsymbol{w})$, con $b \in \{1, \ldots, B\}$.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \cdot \nabla \mathcal{E}_{T_b}\left(\boldsymbol{w}^{(t)}\right) \quad \text{con} \quad T_b \subseteq T\,. \tag{3.4}$$

Observación 3.1.1. *Puede entenderse el descenso de gradiente estocástico como un tipo de descenso de gradiente por lotes en el que hay N lotes con un solo dato.*

El concepto de lote no debe confundirse con el de "época". Una época es un paso completo por el conjunto de entrenamiento T, i.e. se considera que se ha completado una época cuando se ha utilizado cada $(\boldsymbol{x}^i, y^i) \subseteq T$ en al menos una iteración del algoritmo de optimización.

Frecuentemente, al entrenar un modelo con un conjunto de entrenamiento T se consideran B lotes $T_1, \ldots, T_B$, disjuntos y tales que $T_1 \cup \ldots \cup T_B = T$. Cada uno de ellos se utiliza en una iteración de los pesos $\boldsymbol{w}$, no necesariamente en orden, obteniéndose pesos $\boldsymbol{w}^{(t_1)}, \ldots, \boldsymbol{w}^{(t_B)}$. Cuando se han realizado las B iteraciones se dice que se ha completado una época ϵ. El proceso reiterativo de entrenar va completando épocas $\epsilon_1, \ldots, \tilde{\epsilon}$ a medida que entrena con el conjunto T completo. Además, en cada época se pueden utilizar lotes distintos y en un orden arbitrario.

3.1.1. Optimizador *Adam*

El método de descenso de gradiente estocástico (3.3) presenta un problema principal relacionado con la convergencia: si se considera una razón de aprendizaje baja, la variación de los pesos sinápticos es menor y estable pero la convergencia es lenta; si se considera una razón de aprendizaje alta, la variación de los pesos sinápticos es mayor e inestable pero la convergencia es rápida.

Observación 3.1.2. *Recientemente han surgido resultados que respaldan los posibles beneficios a largo plazo de las inestabilidades locales, como el de Cohen et al. [6]. Para aprovecharlas, puede introducirse una razón de aprendizaje dinámica, como se presenta en [33].*

Clásicamente se ha buscado modificar dicho método para conseguir una convergencia rápida pero evitando a su vez la inestabilidad. Para ello, se incluye una "constante de momento" $\alpha \in (0,1)$ que modifica el término iterativo añadiendo una ponderación con el obtenido en la iteración anterior:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{\Delta}^{(t)} \quad \text{con} \quad \boldsymbol{\Delta}^{(t)} = \alpha \boldsymbol{\Delta}^{(t-1)} - \eta \cdot \nabla \mathcal{E}_i \left(\boldsymbol{w}^{(t)} \right) . \tag{3.5}$$

De esta forma, la relación se puede escribir recursivamente como:

$$\boldsymbol{\Delta}^{(t)} = -\eta \sum_{s=1}^{t} \alpha^{t-s} \cdot \nabla \mathcal{E}_i \left(\boldsymbol{w}^{(t-s)} \right) .$$

Esto implica que, si el gradiente toma un determinado signo en varias iteraciones consecutivas, el momento hace que el término $\boldsymbol{\Delta}^{(t)}$ aumente y acelera el descenso. Por otro lado, si el gradiente oscila mucho en signo para iteraciones cercanas, el término de momento induce una estabilización, disminuyendo el valor de $\boldsymbol{\Delta}^{(t)}$.

El método *Adam* es una modificación del método del momento (3.5) que introduce unos parámetros β_1 y β_2, cercanos a 1, para controlar el decaimiento de $\boldsymbol{\Delta}^{(t)}$ y del cuadrado $\left(\boldsymbol{\Delta}^{(t)}\right)^2$. A estos términos se les denomina "momentos" de primer y segundo orden, respectivamente.

Este método de optimización, presentado por Diederick P. Kingma y Jimmy Lei Ba en [17], se utiliza ampliamente en el desarrollo de modelos predictivos por su rápida convergencia y su adaptación, estando afectado también por un parámetro pequeño $\varepsilon \sim 10^{-8}$:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \cdot \frac{\dfrac{\boldsymbol{\Delta}^{(t)}}{1-\beta_1^t}}{\sqrt{\dfrac{\left(\boldsymbol{\Delta}^{(t)}\right)^2}{1-\beta_2^t}} + \varepsilon} \quad \text{con} \quad \begin{cases} \boldsymbol{\Delta}^{(t)} = \beta_1 \boldsymbol{\Delta}^{(t-1)} - (1-\beta_1) \nabla \mathcal{E}_i \left(\boldsymbol{w}^{(t)} \right) \\ \left(\boldsymbol{\Delta}^{(t)}\right)^2 = \beta_2 \left(\boldsymbol{\Delta}^{(t-1)}\right)^2 - (1-\beta_2) \left(\nabla \mathcal{E}_i \left(\boldsymbol{w}^{(t)} \right) \right)^2 \end{cases} .$$

3.2. Regularización por *dropout*

Una vez se ha optimizado el aprendizaje de un modelo construido con una red neuronal, surge el problema que se ha comentado en la Sección *2.1.2*: la generalización. En dicha sección se ha descrito una técnica general de regularización para modelos de aprendizaje supervisado. Esta incluía un parámetro regularizador sobre el error del modelo, lo que para una red neuronal equivale a la función de pérdida, modificando dicho error para que los pesos tiendan a ser inferiores.

Otra técnica habitual de regularización es el denominado *dropout*, o dilución, que evita el sobreajuste eliminando de manera aleatoria una cantidad determinada de conexiones. Las conexiones eliminadas son distintas en cada iteración, por lo que este método reduce los posibles parámetros que estén aprendiendo únicamente ruido o detalles concretos sobre los datos de entrenamiento. Sin

embargo, ha de tenerse en cuenta que la aplicación de estas técnicas introduce un error añadido a la red que no es despreciable.

En la práctica, para aplicar técnicas de dilución se puede considerar una capa de *dropout* que establece como 0 una cierta proporción c de los valores de entrada para dicha capa, variando los valores tomados como nulos en cada entrenamiento. De esta manera, para proporciones bajas ($c \ll 1$), todas las conexiones entrenan en alguna época, pero no a la vez, y los valores de entrada que sí se utilizan se reescalan multiplicándolos por $(1-c)^{-1}$ para que la suma de todos los términos de entrada a la capa permanezca invariante.

Cabe destacar que estas técnicas de *dropout* por capas son más efectivas y menos costosas que otras técnicas de regularización, como el decaimiento de peso, como comprobaron Srivastava *et al.* en [37]. Esto implica que tienen numerosas aplicaciones y que permiten desarrollar modelos más complejos pero manteniendo la convergencia.

3.3. Características de una red neuronal

Como se ha expresado en la descripción general de una red neuronal, existen ciertas características que determinan de manera efectiva el funcionamiento y la convergencia de la red. Por ende, estas características deben determinarse de manera certera para que la red sea adecuada, y para ello es necesario comprender las posibilidades que existen y cuáles son óptimas en función del tipo de aprendizaje y red empleados. Entre estas características destacan: la distribución de inicialización, la función de activación y la función de pérdida.

3.3.1. Distribución de inicialización

Para comenzar el proceso iterativo de entrenamiento, resulta necesario definir unos pesos sinápticos iniciales, $\boldsymbol{w}^{(t)}$, de forma que su optimización sea lo más eficiente posible. Para ello, existen distintas estrategias heurísticas que indican distribuciones de inicialización que mejoran el rendimiento de los entrenamientos.

Cabe destacar que esta tarea es muy relevante, pues un mismo modelo puede o no converger en función de los pesos iniciales que se consideren. Por ello, se evita que los pesos iniciales sean grandes, dado que en las redes recurrentes se tiende a inestabilidades, y se evita también que dos neuronas con la misma función de activación y conectadas a una misma variable presenten los mismos valores iniciales, ya que entonces tenderían a actualizarse de la misma forma constantemente.

En general, la inicialización de los pesos suele ser aleatoria, estando determinada por una distribución gaussiana o una uniforme. La distribución uniforme es tal que su función de densidad

toma el valor $(b-a)^{-1}$ en el intervalo $[a,b]$ y se anula en el resto. Una distribución de inicialización habitual para una red completamente conectada con M valores de entrada y S de salida es la uniforme: $U(-M^{-1/2}, M^{-1/2})$. La utilidad de esta distribución fue descrita por Xavier Glorot y Joshua Bengio en [11].

Asimismo, una versión normalizada de esta distribución de inicialización es aun más útil para redes muy interconectadas, denominándose inicialización normalizada de Glorot (*Glorot normal*, en inglés) y estando determinada por la distribución uniforme:

$$w_{n,j} \sim U\left(-\sqrt{\frac{6}{M+S}}\,,\,\sqrt{\frac{6}{M+S}}\right)\,. \tag{3.6}$$

3.3.2. Función de activación

La función de activación principal en la capa de salida para modelos de regresión es la función identidad $\psi(a) = a$, pero hay algunas funciones de activación que mejoran el rendimiento en los distintos modelos, dado que introducen no linealidades [13]:

- **Función logística sigmoide**: esta función toma valores en el rango $(0,1)$ y, con $\alpha = 1$, es la función habitual en la capa de salida para modelos de clasificación binaria. Se trata de una aproximación suave de la función de Heaviside $\psi(a)$, que toma el valor 1 si $a \geq 0$ y 0 si no. De hecho, la función de Heaviside se recupera al tender el parámetro $\alpha \to \infty$:

$$\sigma(a) = \frac{1}{1+\exp(-\alpha a)}\,. \tag{3.7}$$

  Puede observarse que esta expresión se deduce directamente de los modelos bayesianos para clasificación binaria con aprendizaje supervisado, descritos en la Sección *2.2*.

- **Función *softmax***: es una generalización de la función logística sigmoide que, además, representa una versión suave de la función máximo [16]. Como se ha deducido en la Sección *2.2*, tiene sentido utilizarla en la capa de salida para clasificaciones multiclase con K clases, donde se recupera la función logística sigmoide si $K = 2$. La diferencia principal con la logística sigmoide es que, en este caso, la función considera las activaciones $\boldsymbol{a} = (a_1, \ldots, a_K)$ de las K neuronas de dicha capa y produce un vector con las probabilidades asociadas a cada una de las clases:

$$\text{Softmax}(\boldsymbol{a}) = \left(\frac{\exp(a_1)}{\sum_{n=1}^{K}\exp(a_n)}\,,\,\cdots\,,\,\frac{\exp(a_K)}{\sum_{n=1}^{K}\exp(a_n)}\right)^T\,. \tag{3.8}$$

- **Función lineal rectificadora**, ReLU (del inglés *Rectified linear unit*): es una versión modificada y continua de la función de Heaviside que minimiza errores en redes neuronales de gran

tamaño y no modifica las activaciones nulas, por lo que se utiliza en capas ocultas. También destaca su uso en la capa de salida para modelos de regresión con valores no negativos [26]:

$$\mathrm{ReLU}(a) = \begin{cases} 0 & \text{si} \quad a < 0 \\ a & \text{si} \quad a \geq 0 \end{cases} . \tag{3.9}$$

3.3.3. Función de pérdida

La función de pérdida de una red neuronal en un conjunto de entrenamiento con N observaciones independientes, $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$, se puede definir de manera general como se hizo en (2.1), es decir, como la suma de las distancias entre el resultado predicho y el real para cada dato $\boldsymbol{x}^i \in X$. No obstante, para un estudio en el que la capa de salida de la red neuronal presenta una función de activación ψ concreta, la función de pérdida queda definida por ella [4]:

- **Error cuadrático medio**: para un problema de regresión, se generaliza lo que se ha expresado en la Sección *2.1.1* para una función de predicción arbitraria $\hat{f} : X \to \mathbb{R}$. Entonces, la función de pérdida se define, salvo constante, como el logaritmo neperiano negativo de la función de máxima verosimilitud (2.3), es decir, es la dada por la generalización de la expresión (2.6):

$$\mathcal{E}_{X,Y}(\hat{f}) = \frac{1}{2} \sum_{i=1}^{N} \left(y^i - \hat{f}(\boldsymbol{x}^i) \right)^2 . \tag{3.10}$$

  Esta función de pérdida se conoce comúnmente como error cuadrático, pero resulta más conveniente utilizar el promedio sobre el número de valores de entrenamiento, el error cuadrático medio (MSE, del inglés *mean-squared-error*):

$$\mathcal{E}_{X,Y}(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^i - \hat{f}(\boldsymbol{x}^i) \right)^2 . \tag{3.11}$$

- **Entropía cruzada**: para un problema de clasificación binaria con dos clases C_0 y C_1, para $y^i = 0$ o $y^i = 1$, respectivamente, se utiliza que la probabilidad *a posteriori* de una de las clases, $p\left(C_1|\boldsymbol{x}^i\right)$, es la dada por la ecuación (2.8). Entonces, se puede deducir que la probabilidad de la otra clase es $p\left(C_0|\boldsymbol{x}^i\right) = 1 - p\left(C_1|\boldsymbol{x}^i\right)$. Además, como $\hat{f}(\boldsymbol{x}^i) \in [0, 1]$, pues se obtiene de la función de activación logística sigmoide (3.7) de la capa de salida, se puede interpretar $\hat{f}(\boldsymbol{x}^i)$ como la probabilidad condicionada $p\left(C_1|\boldsymbol{x}^i\right)$.

  A partir de la ecuación (2.3), la función de verosimilitud se define en este caso como el producto de distribuciones Bernouilli independientes. Esto se debe a que la distribución de una variable aleatoria $\mathcal{Y}$ con dos posibilidades discretas 0 y 1, en función de un parámetro $q \in (0, 1)$, es una Bernouilli con $p\left(\mathcal{Y} = y\right) = q^y (1-q)^{1-y}$. En definitiva, la función de verosimilitud obtenida es:

$$\mathcal{L}(\hat{f}) = \prod_{i=1}^{N} p\left(\hat{f}(\boldsymbol{x}^i) = y^i\right) = \prod_{i=1}^{N} \hat{f}(\boldsymbol{x}^i)^{y^i} \cdot \left(1 - \hat{f}(\boldsymbol{x}^i)\right)^{1-y^i} . \tag{3.12}$$

La función de pérdida de entropía cruzada (*cross-entropy*, en inglés) se define, salvo constante, como el logaritmo neperiano negativo de función de verosimilitud anterior:

$$\mathcal{E}_{X,Y}(\hat{f}) = -\sum_{i=1}^{N} \left[y^i \cdot \log\left(\hat{f}(\boldsymbol{x}^i)\right) + (1-y^i) \cdot \log\left(1-\hat{f}(\boldsymbol{x}^i)\right)\right] . \tag{3.13}$$

- **Entropía cruzada categórica**: para un problema de clasificación multiclase con K clases disjuntas, i.e. excluyentes, la etiqueta correcta $\boldsymbol{y}^i$ será un vector de dimensión K con todos los elementos nulos salvo el $y_k^i = 1$ si la etiqueta pertenece a la clase C_k. Ahora, dado que el vector $\hat{\boldsymbol{f}}(\boldsymbol{x}^i) = \left(\hat{f}_1(\boldsymbol{x}^i), \ldots, \hat{f}_K(\boldsymbol{x}^i)\right) \in [0,1]^K$ lo determina la función *softmax*, es un vector de dimensión K cuyos términos suman 1 y se encuentran entre 0 y 1, por lo que puede interpretarse como un vector con las probabilidades de las distintas clases. En esta situación, la función de verosimilitud se define como:

$$\mathcal{L}(\hat{f}) = \prod_{i=1}^{N} p\left(\hat{\boldsymbol{f}}(\boldsymbol{x}^i) = \boldsymbol{y}^i\right) = \prod_{i=1}^{N}\prod_{k=1}^{K} p\left(\hat{f}_k(\boldsymbol{x}^i) = y_k^i\right) = \prod_{i=1}^{N}\prod_{k=1}^{K} \left[\hat{f}_k(\boldsymbol{x}^i)\right]^{y_k^i} . \tag{3.14}$$

Considerando logaritmos neperianos negativos a ambos lados se obtiene la función de pérdida de entropía cruzada categórica (*categorical cross-entropy*, en inglés):

$$\mathcal{E}_{X,Y}(\hat{f}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^i \cdot \log\left(\hat{f}_k(\boldsymbol{x}^i)\right) . \tag{3.15}$$

En caso de que se tengan clases desbalanceadas, es decir, que algunas clases presentan más elementos que las otras, resulta interesante modificar la función de pérdida (3.15) por una función balanceada. Para ello, se añaden términos constantes $W_1, \ldots, W_K$ que ponderan de distinta forma la aportación de cada clase a la función de pérdida general:

$$\mathcal{E}_{X,Y}(\hat{f}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} W_k \cdot y_k^i \cdot \log\left(\hat{f}_k(\boldsymbol{x}^i)\right) . \tag{3.16}$$

Observación 3.3.1. *En la práctica, los pesos $W_1, \ldots, W_K$ deben ser mayores cuanto menor sea el número de elementos en el entrenamiento de una clase dada, pues así se sobrepondera un error en su predicción y se contrarrestra el desbalance.*

3.4. Redes prealimentadas

Como se ha descrito al comienzo de este capítulo, las redes neuronales prealimentadas (*feedforward*, en inglés) son grafos cuyos arcos solo conectan capas consecutivas, es decir, conectan la capa de entrada con la primera oculta, la z-ésima oculta con la $(z+1)$-ésima, y la última capa oculta con la capa de salida. Esto implica que estas redes no pueden presentar ciclos, y que solo transmiten información "hacia delante" [26].

Este tipo de redes pueden interpretarse como una composición de aplicaciones distintas para cada capa z de tamaño $|V_z|$ de la red, $\xi^{(z)} : \mathbb{R}^{|V_{z-1}|} \to \mathbb{R}^{|V_z|}$ con $z \in \{1, \ldots, Z\}$. De hecho, las aplicaciones $\xi^{(z)}$ vienen descritas por una composición de una aplicación afín $\boldsymbol{a}^{(z)} : \mathbb{R}^{|V_{z-1}|} \to \mathbb{R}^{|V_z|}$ y una aplicación de activación $\Psi^{(z)} : \mathbb{R}^{|V_z|} \to \mathbb{R}^{|V_z|}$.

La aplicación afín es tal que $\boldsymbol{a}^{(z)}\left(\hat{\boldsymbol{y}}^{(z-1)}\right) = W^{(z)} \cdot \hat{\boldsymbol{y}}^{(z-1)} + \boldsymbol{b}^{(z)}$, donde $\boldsymbol{b}^{(z)} \in \mathbb{R}^{|V_z|}$ son los sesgos de la capa z-ésima y $W^{(z)} = \left(w_{n,j}^{(z)}\right) \in \mathbb{R}^{|V_z| \times |V_{z-1}|}$ los pesos sinápticos, mientras que $\hat{\boldsymbol{y}}^{(z-1)}$ es la salida de la capa $z-1$, o lo que es lo mismo, la entrada de la capa z. Por otro lado, la aplicación de activación está formada por funciones de activación $\psi_n^{(z)}$, con $n \in \{1, \ldots, |V_z|\}$, reales y no necesariamente iguales:

$$\Psi^{(z)}\left(\boldsymbol{a}^{(z)}\right) = \left(\psi_1^{(z)}\left(a_1^{(z)}\right), \ldots, \psi_{|V_z|}^{(z)}\left(a_{|V_z|}^{(z)}\right)\right)^T .$$

En definitiva, para una red neuronal se puede definir su función de predicción $\hat{f} : \mathbb{R}^M \to \mathbb{R}^S$ en base a un vector de datos de entrada $\boldsymbol{x} \in \mathbb{R}^M$:

$$\hat{f}(\boldsymbol{x}) = \xi^{(Z)} \circ \ldots \circ \xi^{(1)}(\boldsymbol{x}) = \left(\Psi^{(Z)} \circ \boldsymbol{a}^{(Z)}\right) \circ \ldots \circ \left(\Psi^{(1)} \circ \boldsymbol{a}^{(1)}\right)(\boldsymbol{x}) .$$

Es preciso darse cuenta de que, de hecho, $\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{y}}^{(Z)}$, el vector de la salida de la capa final, y, denotando $\hat{\boldsymbol{y}}^{(0)} \coloneqq \boldsymbol{x}$, entonces para cualquier capa z se tiene que:

$$\hat{\boldsymbol{y}}^{(z)}(\boldsymbol{x}) = \xi^{(z)} \circ \ldots \circ \xi^{(1)}(\boldsymbol{x}) = \Psi^{(z)}\left(\boldsymbol{a}^{(z)}\left(\hat{\boldsymbol{y}}^{(z-1)}\right)\right) . \tag{3.17}$$

Con una descripción de la red prealimentada como una composición de funciones, su utilidad para aproximar otras funciones más complejas debe probarse. Esto es lo que hacen los teoremas de aproximación universal, aportar fundamento a este tipo de redes demostrando que la aproximación de funciones mediante redes *feedforward* con los parámetros adecuados es posible. Cabe destacar que los teoremas no son constructivos, muestran la existencia de dichos parámetros pero no la obtención de ellos, por lo que resulta necesario estudiar en detalle un modelo predictivo dado para intentar llegar a sus parámetros óptimos.

3.4.1. Teoremas de aproximación universal

En 1989, George V. Cybenko probó un primer teorema de aproximación universal en [8]. En él se afirma que una red neuronal prealimentada con una capa de salida lineal y al menos una capa oculta con función de activación sigmoidea y suficientes neuronas ocultas puede aproximar cualquier función continua, donde una función sigmoidea es una función continua $\sigma : \mathbb{R} \to \mathbb{R}$ tal que $\lim_{a \to \infty} \sigma(a) = 1$ y $\lim_{a \to -\infty} \sigma(a) = 0$. Por ello, se suele decir que las redes neuronales son "aproximadores universales".

El teorema de Cybenko es el único de los teoremas de aproximación universal que se demuestra en este trabajo, y su enunciado se basa en el concepto de conjunto denso:

Definición 3.4.1. *Se dice que un conjunto $A \subseteq X$ es denso en X si y solo si la adherencia de A es igual a X, i.e. $\bar{A} = X$.*

Con esto, el teorema de aproximación universal de Cybenko se enuncia como sigue [8]:

Teorema 3.4.2. *Considérese el conjunto de las redes neuronales prealimentadas con una sola capa oculta con función de activación continua sigmoidea, valores de entrada pertenecientes a $[0,1]^M$ y de salida en $\mathbb{R}$. Este conjunto es denso en el espacio vectorial de las funciones continuas $f : [0,1]^M \to \mathbb{R}$ con respecto a la norma infinito, $||f||_\infty = \sup\{|f(x)| : \boldsymbol{x} \in [0,1]^M\}$.*

Asimismo, el teorema de aproximación universal presentado es equivalente a afirmar que, para toda función $f : [0,1]^M \to \mathbb{R}$ continua y todo $\varepsilon > 0$, existe un $n \in \mathbb{N}$, $\boldsymbol{w}_l \in \mathbb{R}^M$ y $a_l, b_l \in \mathbb{R}$ para todo $l \in \{1, \ldots, n\}$, tales que:

$$\left|\left|f - \hat{f}\right|\right|_\infty < \varepsilon \quad \text{para} \quad \hat{f}(\boldsymbol{x}) = \sum_{l=1}^{n} a_l \sigma\left(\boldsymbol{x}^T \cdot \boldsymbol{w}_l + b_l\right) \quad \text{con} \quad \boldsymbol{x} \in [0,1]^M . \tag{3.18}$$

Observación 3.4.3. *En esta situación se aprecia que, de hecho, se está describiendo la salida de una red neuronal con una capa oculta de anchura arbitraria n, cuya función de activación de la capa de salida es la identidad. Además, los vectores $\boldsymbol{w}_i$ son los pesos de la capa oculta, los términos b_i son los sesgos y los a_i son los pesos de la capa de salida.*

Para la demostración de este teorema se necesita considerar los teoremas de Hanh-Banach y de representación de Riesz, así como un lema sobre las funciones continuas sigmoideas. Los tres resultados pueden encontrarse en la Sección *A.5* del capítulo de apéndices. Asimismo, debe recordarse lo que es una medida de Borel: μ es una medida sobre la σ-álgebra de Borel. Esto es, una aplicación no negativa que asigna 0 al conjunto vacío y cuyo valor para una unión numerable de conjuntos disjuntos es igual a la suma de los valores para los conjuntos. De hecho, la σ-álgebra de Borel es la σ-álgebra más pequeña que contiene los conjuntos abiertos del espacio, donde una σ-álgebra es una familia no vacía de subconjuntos que es cerrada bajo complementarios y uniones numerables.

Demostración. (Extraída de [8]). Sea $\mathcal{S}$ el conjunto de las funciones $\sum_{l=1}^{n} a_l \sigma\left(\boldsymbol{x}^T \cdot \boldsymbol{w}_l + b_l\right)$, y sea $\mathcal{C}\left([0,1]^M\right)$ el espacio vectorial de las funciones continuas en $[0,1]^M$, entonces $\mathcal{S} \subset \mathcal{C}\left([0,1]^M\right)$ y es un subespacio vectorial. Así, para probar que es denso basta ver que su adherencia $\bar{\mathcal{S}} = \mathcal{C}\left([0,1]^M\right)$.

Asumiendo que $\bar{\mathcal{S}} \neq \mathcal{C}\left([0,1]^M\right)$ y empleando el teorema de Hanh-Banach A.5.1, existe un funcional lineal acotado $L : \mathcal{C}\left([0,1]^M\right) \to \mathbb{R}$, tal que $L \neq 0$ y $L(f) = 0$ si $f \in \bar{\mathcal{S}}$.

Ahora, por el teorema de representación de Riesz A.5.2, existe una medida de Borel μ sobre $[0,1]^M$, de forma que, para toda función $f \in \mathcal{C}\left([0,1]^M\right)$, se tiene que:

$$L(f) = \int_{[0,1]^M} f(\boldsymbol{x}) d\mu(\boldsymbol{x}) \, .$$

En particular, como la función $\hat{f}(\boldsymbol{x}) = \sigma\left(b + \boldsymbol{x}^T \cdot \boldsymbol{w}\right)$ pertenece a $\mathcal{S} \subset \mathcal{C}\left([0,1]^M\right)$, entonces se tiene que, para cualesquiera $\boldsymbol{w} \in \mathbb{R}^M$ y $b \in \mathbb{R}$:

$$L(\hat{f}) = \int_{[0,1]^M} \hat{f}(\boldsymbol{x}) d\mu(\boldsymbol{x}) = \int_{[0,1]^M} \sigma\left(b + \boldsymbol{x}^T \cdot \boldsymbol{w}\right) d\mu(\boldsymbol{x}) = 0 \, .$$

En este caso, dado que $L \neq 0$, entonces $\mu \neq 0$, y se comprueba que esto es una contradicción. De hecho, como σ es una función continua sigmoidea, entonces, por el lema A.5.3, se tiene que $\mu = 0$.

■

Posteriormente, en 1993, Moshe Leshno *et al.* probaron un teorema de aproximación universal más fuerte en [19], dando una condición suficiente y necesaria sobre las funciones de activación de las redes prealimentadas para poder ser aproximadores universales. En su publicación afirmaron que una red prealimentada con una capa oculta arbitrariamente ancha, M nodos fuente en su capa de entrada y una sola neurona en la capa de salida puede aproximar cualquier función continua con cualquier grado de precisión si y solo si las funciones de activación de la red no son polinómicas.

Estos resultados se refieren a la categoría de redes neuronales arbitrariamente anchas y con profundidad acotada, mientras que, en 1999, Vitaly Maiorov y Allan Pinkus demostraron en [22] que existe una función de activación sigmoidea tal que las redes neuronales de dos capas ocultas y anchura acotada son aproximadores universales.

Teorema 3.4.4. *Existe una función de activación sigmoidea σ que es real, analítica y estrictamente creciente y que satisface que: para cualesquiera $\varepsilon > 0$ y $f \in [0,1]^M$, existen constantes reales $\tilde{a}_l, a_{lm}, \tilde{b}_l, b_{lm}$ y vectores $\boldsymbol{w}_{lm} \in \mathbb{R}^M$ tales que, para todo $\boldsymbol{x} \in [0,1]^M$ se cumple que:*

$$\left| f(\boldsymbol{x}) - \sum_{l=1}^{6M+3} \tilde{a}_l \sigma \left(\sum_{m=1}^{3M} a_{lm} \sigma\left(\boldsymbol{x}^T \cdot \boldsymbol{w}_{lm} + b_{lm}\right) + \tilde{b}_l \right) \right| < \varepsilon \, .$$

Observación 3.4.5. *Es claro apreciar que en el teorema 3.4.4 se describe una red neuronal con dos capas ocultas y anchura acotada, siendo la anchura de cada capa $3M$ y $6M+3$, respectivamente, donde M es la dimensión de los datos de entrada $\boldsymbol{x} \in \mathbb{R}^M$. De hecho, se han introducido pesos de la primera capa oculta $\boldsymbol{w}_{lm}$, con sesgos b_{lm}, pesos de la segunda capa oculta a_{lm}, con sesgos $\tilde{b}_l$, y pesos de la capa de salida $\tilde{a}_l$.*

Por otro lado, en 2017, Zhou Lu *et al.* probaron en [21] que, para redes neuronales con funciones de activación ReLU (3.9) y anchura acotada se tiene también un resultado de aproximación universal. Relacionado también con las funciones de activación ReLU, en 2022, Zuowei Shen *et al.* caracterizaron en [36] la profundidad y la anchura necesarias para aproximar una función mediante redes neuronales con este tipo de funciones de activación.

3.4.2. Retropropagación

Los resultados de aproximación universal presentados en la sección anterior afirman la existencia de redes neuronales prealimentadas capaces de ser aproximadores universales, pero no dan un método constructivo para su obtención. Por ello, para desarrollar un modelo predictivo utilizando una red neuronal *feedforward*, esta debe entrenarse mediante un algoritmo de optimización iterativo, como los descritos en la Sección *3.1.*

En la primera parte de este capítulo se ha llegado a que el vector de pesos $\boldsymbol{w}$ se debe actualizar iterativamente en base a la relación (3.2). El descenso de gradiente estocástico es un algoritmo de optimización que permite realizar esta actualización considerando el gradiente de la parte de la función de pérdida asociada al dato $\boldsymbol{x}^i$ en cada iteración t, i.e. $\nabla \mathcal{E}_i \left(\boldsymbol{w}^{(t)}\right)$. De esta forma, resulta necesario desarrollar un método que optimice el cálculo de dicho gradiente: en redes prealimentadas es la retropropagación.

Observación 3.4.6. *La retropropagación no es un algoritmo de optimización de la red neuronal que permite el aprendizaje, como puede ser el descenso de gradiente, sino una manera de calcular el gradiente necesario para aplicar dichos algoritmos [12].*

Considerando entonces la función de pérdida para un dato $\boldsymbol{x}$, $\mathcal{E}(\boldsymbol{w})$, su dependencia con los pesos $\boldsymbol{w}$ es a través de las funciones de activación. En particular, la dependencia de $\mathcal{E}$ con el peso $w_{n,j}^{(z)}$ que conecta la neurona j de la capa $z-1$ con la neurona n de la capa z, es tal que, utilizando la regla de la cadena:

$$\frac{\partial \mathcal{E}}{\partial w_{n,j}^{(z)}} = \frac{\partial \mathcal{E}}{\partial a_n^{(z)}} \cdot \frac{\partial a_n^{(z)}}{\partial w_{n,j}^{(z)}} = \frac{\partial \mathcal{E}}{\partial a_n^{(z)}} \cdot \hat{y}_j^{(z-1)} =: \delta_n^{(z)} \cdot \hat{y}_j^{(z-1)} . \tag{3.19}$$

Así, se definen las "señales de error" $\delta_n^{(z)}$ de la neurona n de la capa z, y se tiene en cuenta que $\hat{y}_j^{(z-1)} = 1$ en el caso en el que $w_{n,j}^{(z)} = b_n^{(z)}$ sea el sesgo de la neurona n de la capa z. Además, se considera que $\hat{\boldsymbol{y}}^{(0)} = \boldsymbol{x}$, los valores de entrada.

Observación 3.4.7. *El cálculo de estas señales de error es la parte interesante del método, pues es donde ocurre la "propagación hacia atrás" [30].*

Se puede utilizar la regla de la cadena para ver que, en realidad, la señal de error de la neurona n de la capa z queda totalmente determinada por las señales de error de las neuronas de la capa $z+1$, junto con la función de activación de la neurona n y los pesos sinápticos $w_{rn}^{(z+1)}$, donde $r \in \{1, \ldots, |V_{z+1}|\}$, que unen dicha neurona n con las de la capa siguiente:

$$\delta_n^{(z)} = \frac{\partial \mathcal{E}}{\partial a_n^{(z)}} = \sum_{r=1}^{|V_{z+1}|} \frac{\partial \mathcal{E}}{\partial a_r^{(z+1)}} \cdot \frac{\partial a_r^{(z+1)}}{\partial a_n^{(z)}} = \sum_{r=1}^{|V_{z+1}|} \delta_r^{(z+1)} \cdot w_{r,n}^{(z+1)} \cdot \left[\psi_n^{(z)}\left(a_n^{(z)}\right)\right]' . \tag{3.20}$$

En la última igualdad se ha usado que, por la ecuación (3.17):

$$\begin{aligned} \frac{\partial a_r^{(z+1)}}{\partial a_n^{(z)}} &= \frac{\partial}{\partial a_n^{(z)}}\left[\boldsymbol{w}_r^{(z+1)} \cdot \hat{\boldsymbol{y}}^{(z)} + b_r^{(z+1)}\right] = \frac{\partial}{\partial a_n^{(z)}}\left[\boldsymbol{w}_r^{(z+1)} \cdot \Psi^{(z)}\left(\boldsymbol{a}^{(z)}\right) + b_r^{(z+1)}\right] = \\ &= w_{r,n}^{(z+1)} \cdot \left[\psi_n^{(z)}\left(a_n^{(z)}\right)\right]' . \end{aligned}$$

Esta relación es válida para cualquier neurona n en una capa z oculta, pero no se puede aplicar si $z = Z$ la capa de salida, pues $z+1$ no es una capa de la red. Por ello, para la capa de salida se debe tener en cuenta que $\delta_n^{(Z)}$ se puede calcular a partir de la función de pérdida que se defina en la red y de la correspondiente función de activación de la capa de salida. La deducción de las expresiones explícitas para las funciones estudiadas en este trabajo puede consultarse en A.6 del capítulo de apéndices.

En definitiva, el cálculo del gradiente de la función de pérdida de una red neuronal prealimentada por el método de retropropagación se realiza de la siguiente manera [4]:

1. Se introduce un vector $\boldsymbol{x}$ de datos de entrada que se propagan por las distintas neuronas utilizando las funciones de activación y la relación (3.1). Así, se llega a unos valores de salida finales de la red $\hat{f}(\boldsymbol{x})$.

2. Se obtienen las señales de error $\delta_k^{(Z)}$ de las neuronas de la capa de salida, dados por la relación correspondiente para su función de pérdida. Estas señales de error se retropropagan por todas las neuronas en capas ocultas de la red mediante la relación (3.20).

3. El gradiente de la función de pérdida para una entrada $\boldsymbol{x}$ en una iteración se define como el vector de las derivadas parciales (3.19), en las que se incluyen las señales de error retropropagadas del segundo paso, y las salidas de las neuronas obtenidas en el primer paso.

3.5. Redes recurrentes

Las redes neuronales prealimentadas estudiadas en la Sección *3.4* presentan un problema principal: están limitadas a tareas estáticas, i.e. consiguen aproximar una función estática entre los

datos de entrada y los de salida. Sin embargo, en modelos predictivos con dependencia temporal, las tareas a predecir pueden presentar variaciones, por lo que es necesario emplear redes neuronales que permitan desarrollar tareas dinámicas.

Para conseguir esta propiedad se deben volver a introducir en la red las señales de entrenamientos anteriores, mediante conexiones recurrentes, luego las redes que incluyen dichas conexiones se denominan redes neuronales recurrentes (RNN, del inglés *recurrent neuronal networks*). Además, se introduce el concepto de "etapa", E, como el periodo de entrenamiento, es decir, la red comienza a entrenar en un instante t_1 y finaliza en un instante $t_2 > t_1$, siendo la etapa el periodo $[t_1, t_2]$.

La parte crucial de las redes recurrentes es el *feedback*, que permite crear arquitecturas muy distintas para casos particulares diferentes. En este tipo de redes, las neuronas de capas posteriores se conectan con neuronas de capas anteriores formando ciclos, o incluso con ellas mismas (*feedback* propio). Así, se puede entender que la red neuronal tiene un "estado" en cada etapa y que, en la etapa siguiente, las neuronas reciben tanto los valores de entrada $\boldsymbol{x}^i$ como los valores generados por las distintas neuronas en la etapa anterior. De esta manera, en las RRN se emplearán "retardadores", denotados por τ^{-1}, que permitirán modificar en qué etapa quieren introducirse los distintos valores.

Suponiendo una red neuronal prealimentada simple, con dos capas de M y S neuronas, una de entrada y una de salida, respectivamente. Esto es, tiene valores de entrada $\boldsymbol{x} \in \mathbb{R}^M$ y de salida $\hat{\boldsymbol{y}}^{(1)} \in \mathbb{R}^S$. Sea la etapa actual E, si se quiere convertir la red anterior en una red recurrente se pueden considerar como entrantes a la red los valores de entrada $\boldsymbol{x}_{E-q}, \ldots, \boldsymbol{x}_E$ y los valores de salida $\hat{\boldsymbol{y}}^{(1)}_{E-p}, \ldots, \hat{\boldsymbol{y}}^{(1)}_{E-1}$. Para ello, simplemente se introducen $p+q$ retardadores τ^{-1}, como en la figura 3.3, consiguiendo una red neuronal autorregresiva no lineal con entradas exógenas (NARX, del inglés *nonlinear autoregressive with exogenous inputs*). Esta red recurrente consta de $(p+q+1)\cdot M$ nodos fuente, asociados con el vector de entrada de la etapa actual y los vectores de entrada de las q etapas anteriores, así como los vectores de salida de las p etapas anteriores.

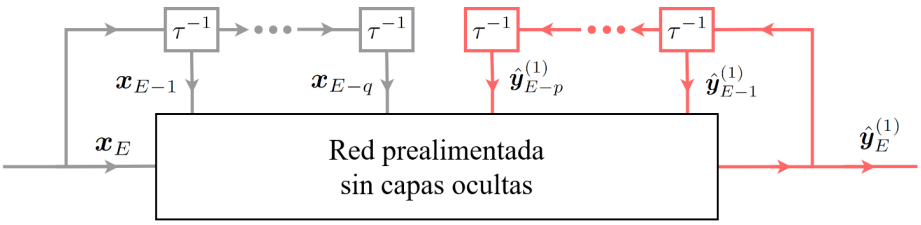


Figura 3.3: *Red neuronal NARX, donde las conexiones recurrentes del feedback aparecen en rojo.*

Como se puede observar, cuantas más capas y neuronas se introduzcan en la red neuronal prealimentada de soporte, más posibles redes neuronales recurrentes se podrán conseguir, simplemente mediante combinaciones de los *feedback* entre las distintas capas [13].

Un ejemplo habitual es el de los perceptrones multicapa recurrentes (RMLP, del inglés *recurrent multilayer perceptron*), en el que cada capa z de la red neuronal tiene como valores de entrada en una etapa E los habituales de dicha etapa $\hat{\boldsymbol{y}}_E^{(z-1)}$ y los valores producidos por dicha capa en la etapa anterior $\hat{\boldsymbol{y}}_{E-1}^{(z)}$, como se indica en la figura 3.4. Con esto, considerando funciones de activación $\Psi^{(z)}$ y sesgos $\boldsymbol{b}^{(z)}$ para las distintas capas de tamaño $|V_z|$, así como pesos sinápcticos habituales $w_{n,j}$ de la neurona j de la capa $z-1$ a la n de la capa z y pesos recurrentes $\tilde{w}_{n,j}$ de la neurona j de la capa z a la n de la capa z, entonces el valor de salida para la neurona n de la capa z es:

$$\left(\hat{y}_n^{(z)}\right)_E = \psi_n^{(z)}\left(b_n^{(z)} + \sum_{j=1}^{|V_{z-1}|} w_{n,j}^{(z)}\left(\hat{y}_j^{(z-1)}\right)_E + \sum_{k=1}^{|V_z|} \tilde{w}_{n,k}^{(z)}\left(\hat{y}_j^{(z)}\right)_{E-1}\right). \tag{3.21}$$

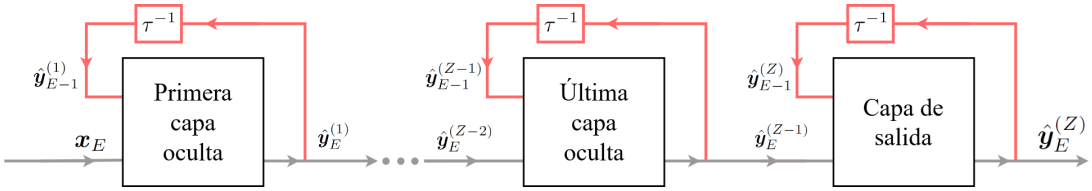


Figura 3.4: *Red RMLP, las conexiones recurrentes del feedback aparecen en rojo.*

Observación 3.5.1. *Las neuronas cuyas salidas se calculan con la ecuación* (3.21) *se denominan de primer orden, pudiéndose definir neuronas de segundo orden si incluyen en la entrada de su función de activación un término multiplicativo de los* $\left(\hat{y}_j^{(z-1)}\right)_E$ *con los* $\left(\hat{y}_k^{(Z)}\right)_{E-1}$.

Por otro lado, de manera análoga a los teoremas de aproximación universales para redes neuronales prealimentadas, en el caso de redes recurrentes se tienen teoremas que afirman que las RNN son aproximadores universales.

En 2002, Xiaoou Li y Wen Yu probaron en [20] que cualquier sistema dinámico no lineal puede ser aproximado por una red neuronal recurrente con el grado de precisión que se desee y sin restricciones en cuanto a la compacidad del espacio de estados, siempre que la red esté equipada con un número adecuado de neuronas ocultas. Posteriormente, en 2006, Anton M. Schäfer y Hans G. Zimmermann probaron en [34] un resultado análogo pero con funciones de activación sigmoideas.

3.5.1. Algoritmos de optimización

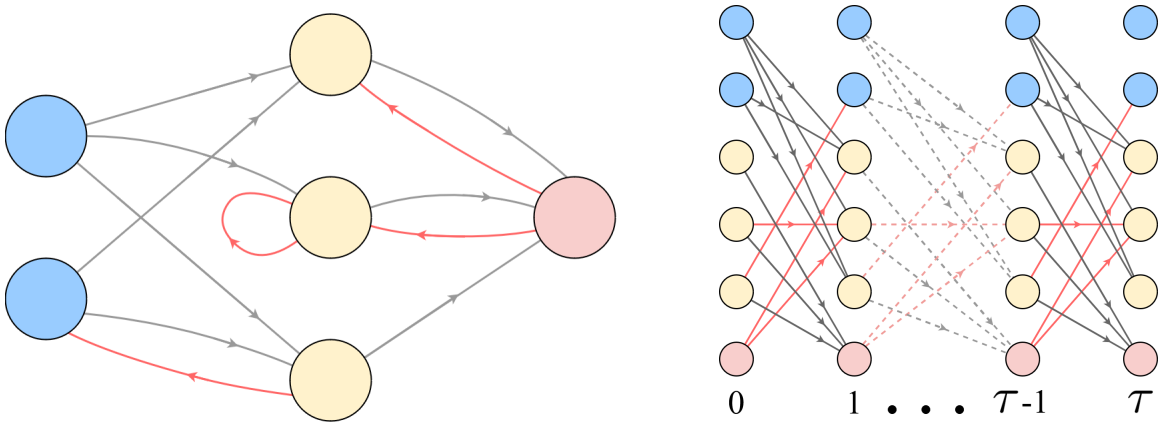
Los algoritmos de optimización habituales para redes recurrentes con aprendizaje supersivado son la retropropagación a través del tiempo (BPTT, del inglés *backpropagation through time*) y el aprendizaje recurrente en tiempo real (RTRL, del inglés *real-time recurrent learning*). Estos algoritmos son distintos de los habituales para redes neuronales prealimentadas porque es necesario que se propague información también a través de las conexiones recurrentes.

El algoritmo BPTT es una generalización del algoritmo clásico de descenso de gradiente estocástico junto con la retropropagación asociada. Para ello, se considera una red neuronal prealimentada desdoblada formada por τ capas, donde cada una contiene la información de la red neuronal recurrente, y luego se aplica el algoritmo conocido.

De esta forma, para el entrenamiento de una etapa se consideran τ pasos temporales discretos y se construye la red desdoblada como una red con τ capas y $|V_n|$ neuronas por capa, donde $|V_n|$ es el tamaño de la red recurrente. Además, en cada capa de la red desdoblada hay una copia de cada neurona de la RNN, y los pesos sinápticos $w_{n,j}^{(z)}$ que unen la neurona j de la capa $z-1$ de la red desdoblada con la neurona n de la capa z son una copia de los pesos sinápticos que unen la neurona n y la j en la red recurrente [13]. Puede verse una comparativa en las figuras 3.5a y 3.5b.

Observación 3.5.2. *Una cuestión interesante es que, ordenando convenientemente las neuronas, en la red desdoblada prealimentada los arcos ascendentes u horizontales corresponden a conexiones recurrentes en la RNN, mientras que los descendentes en la desdoblada corresponden a conexiones sinápticos en la RNN. Esto se aprecia de manera clara en las figuras 3.5a y 3.5b.*

Dado que cada capa z de la red desdoblada representa un paso temporal de la red recurrente, se aplica así el algoritmo de descenso de gradiente estocástico presentado en la Sección *3.1*, utilizando también la retropropagación descrita en la Sección *3.4.2*, sobre la red neuronal desdoblada. Esto permite optimizar tanto los pesos sinápticos como los pesos recurrentes, y se repite el proceso para la siguiente etapa [40].



(a) *Arquitectura de una RNN de tamaño 6.* (b) *Red desdoblada de tamaño* $6 \cdot (\tau + 1)$.

Figura 3.5: *Desdoblamiento de una red recurrente para* τ *instantes temporales.*

Observación 3.5.3. *El algoritmo de retropropagación a través del tiempo aumenta su complejidad computacionalmente de manera exponencial según se aumenta el tamaño de la red neuronal recurrente o el número de pesos recurrentes que contiene, por lo que solo es útil para RNN pequeñas.*

El algoritmo BPTT se basa en la actualización de los pesos sinápticos mediante el descenso de gradiente estocástico presentado en la Sección *3.1*, utilizando también el método de retropropagación presentado en la Sección *3.4.2*, pues la red recurrente se desdobla en redes neuronales prealimentadas. Por otro lado, el algoritmo RTRL se dice que optimiza "en línea" porque modifica los pesos sinápticos a medida que se va propagando la información, es decir, durante el paso hacia delante, luego no necesita ningún desdoblamiento. Su desarrollo puede consultarse en [41].

3.5.2. Gradiente evanescente

Uno de los principales problemas de las redes neuronales recurrentes que se entrenan con algoritmos de optimización basados en el descenso de gradiente y la retropropagación es el gradiente evanescente. Esto se debe a que las señales de error retropropagadas tienden a crecer o decrecer con cada paso temporal, luego si crecen en exceso tienden a pesos oscilantes y no convergentes, mientras que si decrecen en exceso el aprendizaje se ralentiza y no mejora [38].

La explicación es sencilla. Como los pesos $w_{n,j}$ (incluyendo los $\tilde{w}_{n,j}$ recurrentes) se actualizan en base a la relación (3.3) con una razón de aprendizaje η y una etapa entre los tiempos t_1 y t_2, entonces:

$$\Delta_{n,j} = -\eta \cdot \frac{\partial \mathcal{E}(t_1, t_2)}{\partial w_{n,j}} = -\eta \cdot \sum_{t=t_1}^{t_2} \delta_n(t) \hat{y}_j(t) \, .$$

La señal de error retropropagada en un tiempo $t \in [t_1, t_2)$ de la neurona n es:

$$\delta_n(t) = \psi_n'(a_n(t)) \cdot \sum_{r \in U_n} w_{r,n} \cdot \delta_r(t+1) \, , \tag{3.22}$$

donde U_n es el conjunto de neuronas para las que existe un peso que las conecta con la neurona n.

Por ende, la señal de error dada para una neurona s de la capa de salida en el paso temporal t se retropropaga por la red durante $t - t_1$ pasos temporales hasta una neurona arbitraria n. La señal de error se modifica entonces de la forma:

$$\frac{\partial \delta_n(t_1)}{\partial \delta_s(t_2)} = \begin{cases} \psi_n'(a_n(t_1)) \cdot w_{s,n} & \text{si} \quad t_2 - t_1 = 1 \\ \psi_n'(a_n(t_1)) \cdot \left[\sum_{r \in U_n} \dfrac{\partial \delta_r(t_1+1)}{\partial \delta_s(t_2)} \cdot w_{r,n} \right] & \text{si} \quad t_2 - t_1 > 1 \end{cases} . \tag{3.23}$$

Si se desarrolla la ecuación (3.23) sobre el tiempo y se denota por n_t una neurona en el tiempo t, entonces se obtiene la siguiente expresión:

$$\frac{\partial \delta_n(t_1)}{\partial \delta_s(t_2)} = \sum_{n_{t_1} \in U_n} \cdots \sum_{n_{t_2-1} \in U_n} \left(\prod_{t=t_1+1}^{t_2} \psi_{n_t}'(a_{n_t}(t_2 - t + t_1)) \cdot w_{n_t, n_{t-1}} \right) . \tag{3.24}$$

De esta relación se deduce que si $\left|\psi'_{n_t}(a_{n_t}(t_2 - t + t_1)) \cdot w_{n_t,n_{t-1}}\right| > 1$ para todo t, entonces el producto crece exponencialmente y la señal de error explota. Por otro lado, si se tiene que $\left|\psi'_{n_t}(a_{n_t}(t_2 - t + t_1)) \cdot w_{n_t,n_{t-1}}\right| < 1$ para todo t, entonces la señal de error desaparece al decrecer exponencialmente, y esto implica que la señal de error global también desaparece. Este es el llamado fenómeno de evanescencia de gradiente, que ha sido descrito, entre otros, en [38].

3.5.3. Memoria a corto-largo plazo

Un tipo de red neuronal recurrente que es de especial relevancia por sus múltiples aplicaciones prácticas es el de las redes de memoria a corto-largo alcance (LSTM, del inglés *long short-term memory*). Estas fueron propuestas para mitigar el problema del gradiente evanescente, y en ellas las neuronas pasan a formar unidades LSTM, denominadas "células de memoria", que constan de un carrusel de errores constantes (CEC, del inglés *constant error carousel*), una puerta de entrada y una puerta de salida [14].

El fundamento teórico de estas células se basa en conseguir neuronas con un flujo de señal de error constante, para lo que se considera una neurona n con un único arco a si misma. De esta manera, su señal de error en un instante t es simplemente, por la ecuación (3.22):

$$\delta_n(t) = \psi'_n(a_n(t)) \cdot w_{n,n} \cdot \delta_n(t+1) \,. \tag{3.25}$$

Para asegurar un flujo de señal de error constante, i.e. $\delta_n(t) = \delta_n(t+1)$, se debe cumplir que $\psi'_n(a_n(t)) \cdot w_{n,n} = 1$, luego integrando se obtiene que:

$$\psi_n(a_n(t)) = \frac{a_n(t)}{w_{n,n}} \,.$$

Se deduce que ψ_n debe ser lineal y constante en el tiempo, luego basta considerar como función de activación la identidad y establecer que $w_{n,n} = 1$, que son las condiciones de CEC, pues así:

$$\hat{y}_n(t+1) = \psi_n(a_n(t+1)) = \psi_n(\hat{y}_n(t) \cdot w_{n,n}) = \hat{y}_n(t) \,.$$

En definitiva, considerando una célula de memoria m_j como la de la figura 3.6, la puerta de entrada (In) controla las señales de la red a la célula de memoria, mientras que la puerta de salida (Out) protege a otras células de memoria de las perturbaciones originadas en m_j. Además, la célula consta de un CEC entre las funciones de activación anterior $\psi_{m_j}^{(\text{pre})}$ y posterior $\psi_{m_j}^{(\text{pos})}$, y las células de memoria m_j están agrupadas en "bloques de memoria" m, que son conjuntos de células de memoria que comparten una puerta de entrada y una puerta de salida.

Para un tiempo t, una célula de memoria recibe tres tipos de valores distintos, los procedentes de la puerta de entrada, $\hat{y}^{\text{in}_j}(t)$, los procedentes de la puerta de salida, $\hat{y}^{\text{out}_j}(t)$, y los que entran

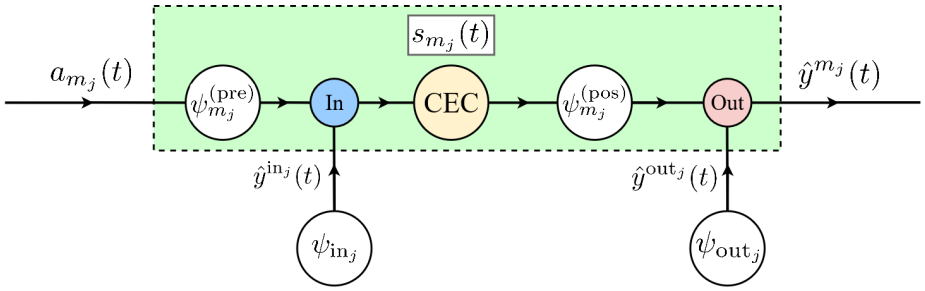


Figura 3.6: *Arquitectura de una célula de memoria m_j.*

de manera directa a la célula, $a_{m_j}(t)$. Dadas funciones de activación ψ_{in_j} y ψ_{out_j} de las puertas de entrada y salida, respectivamente, usualmente logísticas sigmoides, los valores que recibe son:

$$\begin{aligned}
\hat{y}^{\mathrm{in}_j}(t) &= \psi_{\mathrm{in}_j}\left(a_{\mathrm{in}_j}(t)\right) = \psi_{\mathrm{in}_j}\left(\sum_{r\in U} w_{\mathrm{in}_j,r}\hat{y}_r(t-1) + \sum_{i\in I} w_{\mathrm{in}_j,i}\hat{y}_i(t)\right) \\
\hat{y}^{\mathrm{out}_j}(t) &= \psi_{\mathrm{out}_j}\left(a_{\mathrm{out}_j}(t)\right) = \psi_{\mathrm{out}_j}\left(\sum_{r\in U} w_{\mathrm{out}_j,r}\hat{y}_r(t-1) + \sum_{i\in I} w_{\mathrm{out}_j,i}\hat{y}_i(t)\right) \\
a_{m_j}(t) &= \sum_{r\in U} w_{m_j,r}\hat{y}_r(t-1) + \sum_{i\in I} w_{m_j,i}\hat{y}_i(t)
\end{aligned} \tag{3.26}$$

Además, se puede definir el "estado" de la célula $s_{m_j}(t)$ en función de su estado previo y de la multiplicación de los valores de entrada a través de la puerta de entrada y los directos, estos últimos multiplicados por una función de activación anterior $\psi^{(\mathrm{pre})}_{m_j}$:

$$s_{m_j}(t) = s_{m_j}(t-1) + \hat{y}^{\mathrm{in}_j}(t)\cdot\psi^{(\mathrm{pre})}_{m_j}\left(a_{m_j}(t)\right)\ , \tag{3.27}$$

donde se considera que $s_{m_j}(0) = 0$ y $\psi^{(\mathrm{pre})}_{m_j}(a) = \frac{4}{1-\exp(-a)} - 2$, que reescala al intervalo $(-2, 2)$.

El valor de salida de la célula, $\hat{y}^{m_j}(t)$ se obtiene multiplicando el valor de la puerta de salida por estado de la célula $s_{m_j}(t)$, bajo el efecto de una función de activación posterior $\psi^{(\mathrm{pos})}_{m_j}$:

$$\hat{y}^{m_j}(t) = \hat{y}^{\mathrm{out}_j}(t)\cdot\psi^{(\mathrm{pos})}_{m_j}\left(s_{m_j}(t)\right)\ , \tag{3.28}$$

donde se considera $\psi^{(\mathrm{pos})}_{m_j}(a) = \frac{2}{1-\exp(-a)} - 1$, que reescala al intervalo $(-1, 1)$.

En resumen, las redes LSTM son un tipo de redes neuronales recurrentes formadas por células con puertas, que modifican las señales de entrada y de salida. Esta estructura permite que en cada iteración la red almacene y olvide información de iteraciones anteriores, lo que las hace especialmente útiles para modelizar problemas con dependencia temporal a largo plazo.

Observación 3.5.4. *El algoritmo de optimización de las redes LSTM es una combinación de los algoritmos BPTT y RTRL de las redes recurrentes, pudiendo consultarse en [38]. La base de este algoritmo es que se utiliza BPTT para entrenar los componentes de la red posteriores a las células y RTRL para las componentes posteriores a las células, y también para las componentes de las células.*

Capítulo 4

Modelo predictivo de la turbidez del agua

La turbidez del agua es una variable física difícil de describir analíticamente, pues su dependencia con el resto de variables no es clara. Además, al tratarse de un parámetro físico con una definición un tanto ambigua, esto complica aun más su análisis.

Definición 4.0.1. *La turbidez es la medida del grado de transparencia, o claridad, de un líquido. Es una característica óptica que se mide a través de la cantidad de luz que se dispersa al atravesar el material, y su unidad es NTU (de las siglas en inglés, Nephelometric Turbidity Unit).*

Por otro lado, la caracterización y predicción de la turbidez del agua resultan de vital importancia en muchas circunstancias, entre las que destaca la producción de agua potable. Los eventos de turbidez suponen un problema para las plantas desaladoras de agua, pues los mecanismos de desalinización basados en membranas y ósmosis inversa no son capaces de funcionar con altos niveles de turbidez[1], provocando que las plantas tengan que detener su funcionamiento.

Así, para la operatividad de una planta desaladora es interesante poder anticiparse a estos eventos y planificar la detención y limpieza de las plantas, pudiendo optimizar su funcionamiento. Este es el motivo por el que la empresa ACCIONA se interesó por desarrollar un modelo de estas características, proporcionando los datos y medios necesarios para ello.

En definitiva, con la intención de poner a prueba todos los conceptos descritos en los capítulos anteriores y también por el interés propio transmitido por ACCIONA, se plantea el problema siguiente: *Dado el histórico de turbidez recogido en una planta desaladora, ¿se puede desarrollar un modelo predictivo que estime la turbidez en ella en un periodo de dos horas en el futuro?*

[1] Véase [1] para más información sobre la ósmosis inversa y el efecto de la turbidez en ella.

4.1. Definición del problema y datos considerados

Una vez entendido el problema, el primer desafío es decidir qué resultados se quieren obtener de la predicción, y, en base a ello, qué variables son necesarias.

De manera natural, la primera idea que surge es predecir el valor de la turbidez exacta en NTU que habrá en un tiempo τ en el futuro, pero tras los primeros intentos se observa que este procedimiento lleva a un resultado poco preciso. Por ende, con la intención de mejorar el nivel de precisión en la predicción sin perder una excesiva información, se plantea la opción alternativa de predecir la probabilidad de que la turbidez esté en un cierto nivel en un tiempo τ. A lo largo de este capítulo se desarrollan las dos opciones, comparando sus similitudes y diferencias, y presentando los resultados obtenidos de las dos maneras. Se denominarán **Modelo 1** (predicción de la turbidez) y **Modelo 2** (predicción del nivel de turbidez).

En cualquier caso, para ambos desarrollos es necesario el valor de la turbidez como variable de entrada, lo que se ha obtenido de la planta que opera ACCIONA como una sucesión temporal de valores en NTU para el periodo desde 2017 hasta 2023. El paso temporal entre los datos considerados es de un minuto, luego se tienen 3.680.640 datos de turbidez. Todos ellos se relacionan con distintas variables externas que, se entiende, deben influir en la producción de turbidez.

4.1.1. Variables externas consideradas

La turbidez del agua tiene distintos orígenes y, por ello, distintos parámetros que la modifican, entre los que destacan la presencia de materia en suspensión y sedimentos relacionados con la erosión, la materia orgánica como plancton u otros organismos microscópicos, y el crecimiento de algas.

Estos parámetros son muy complicados de medir de manera directa, por lo que se han estudiado otras variables meteorológicas y biológicas con las que pueden estar relacionados pero que sí son medibles con satélites. Las nueve variables externas elegidas se han tomado de la base de datos de *Meteomatics*[2] y son las que siguen:

- **Velocidad de las corrientes marinas (V.C.M.)**: medida en km/h, las corrientes marinas pueden desplazar sedimentos y microorganismos.
- **Dirección de las corrientes marinas (D.C.M.)**: medida de 0 a 360 grados.
- **Velocidad del viento (V.V.)**: medida en km/h, el viento puede desplazar polvo y otros materiales que luego caen al agua y la enturbian.
- **Dirección del viento (D.V.)**: medida de 0 a 360 grados.

[2]Para más información, consúltese `https://www.meteomatics.com/`.

- **Polvo (P)**: medida su densidad en $\mu g/m^3$, se consideran partículas de polvo con un tamaño entre 0,9 y 20 μm, que pueden afectar de manera directa a la turbidez.
- **Dióxido de nitrógeno (NO_2)**: medida su densidad en $\mu g/m^3$, el NO_2 puede servir como una medida de la materia orgánica presente en el agua.
- **Temperatura del agua (T.A.)**: medida en grados Celsius, la temperatura puede acelerar la proliferación de microorganismos y algas.
- **Dióxido de azufre (SO_2)**: medida su densidad en $\mu g/m^3$, el SO_2 también puede servir como una medida de la materia orgánica presente en el agua.
- **Salinidad (S)**: medida su densidad en $\mu g/m^3$, la salinidad puede favorecer el desarrollo de algas y la presencia de materia viva.

Para todas ellas se ha considerado un punto espacial a un kilómetro (mar adentro) de la entrada de agua de la planta desaladora, y se han obtenido los datos desde 2017 hasta 2023 con un paso temporal de cinco minutos, luego se han incluido 6.625.152 datos meteorológicos y biológicos.

Resulta interesante realizar un análisis previo de los coeficientes de correlación, para entender qué relaciones existen entre estas variables y la turbidez. Para ello, se utiliza el coeficiente correlación de Pearson (véase A.7 del capítulo de apéndices) y se obtienen los resultados mostrados en la figura 4.1. De esta forma, se observa que ninguno de los coeficientes de correlación es representativo, pues todos tienen valor absoluto menor que 0,3 [5].

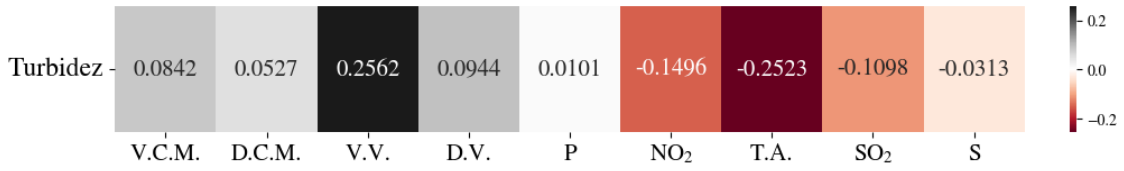
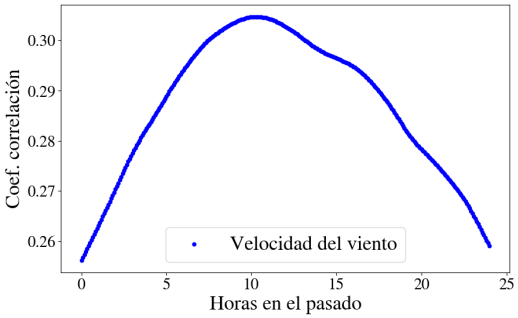


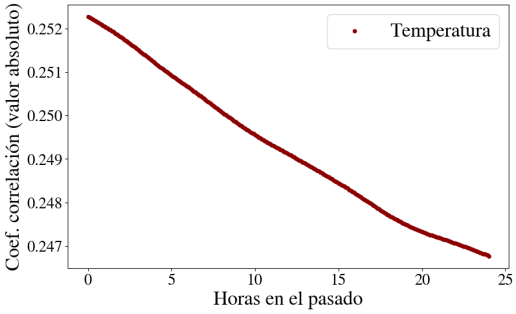
Figura 4.1: *Coeficientes de correlación de Pearson para la turbidez con las variables externas.*

Por otro lado, y en la búsqueda de correlaciones que pudiesen facilitar el desarrollo del modelo predictivo, se intenta relacionar el histórico de las distintas variables con la turbidez actual. El procedimiento a seguir consiste en obtener el coeficiente de correlación de Pearson entre la turbidez en el instante t y las variables externas en el instante $t - t_1$, con $t_1 \in \mathbb{N}$ variando en el intervalo de 0 a 288, considerando que t son los puntos temporales discretos con un paso de cinco minutos. El intervalo utilizado es tal que las variables externas llegan a estudiarse hasta las 24 horas anteriores al instante considerando para la turbidez.

Una vez hecho esto, para cada variable externa se presenta el resultado en una gráfica comparando el coeficiente de correlación frente al instante temporal en el pasado y se buscan los valores máximos de correlación. Dos ejemplos de estas gráficas pueden verse en las figuras 4.2a y 4.2b para la velocidad del viento y la temperatura, respectivamente.



(a) *Correlación con la velocidad del viento.*



(b) *Correlación con la temperatura.*

Figura 4.2: *Coeficientes de correlación de Pearson para la turbidez con desfase temporal.*

Como se puede observar, algunas variables, como la velocidad del viento, tienen mayor correlación con la turbidez si se considera un desfase temporal entre ellas, mientras que otras, como la temperatura, tienen una mayor correlación con la turbidez en el momento presente. En definitiva, esto apoya la necesidad de emplear un modelo de aprendizaje automático capaz de relacionar series temporales en distintos instantes: *es necesario utilizar un modelo con memoria a corto y largo plazo.*

4.1.2. Adaptaciones de los datos necesarias para la predicción

Previo al desarrollo del modelo predictivo son necesarias tres tipos de adaptaciones para los datos de entrada: calibración temporal, supresión del ruido y clasificación en clases.

En primer lugar, los datos de turbidez considerados presentan un espaciado temporal de un minuto, mientras que las variables externas tienen un intervalo temporal de cinco minutos. Con el fin de aumentar la precisión del modelo, los datos de turbidez se promedian (media aritmética) en intervalos de cinco minutos y se unen con los de las variables externas, pasando a tener un *dataset* de 7.361.280 datos.

Matemáticamente, la sucesión de datos de turbidez $\{(\tilde{t}^k, \tilde{x}^k)\}_{k=1}^{\tilde{N}}$ con $\tilde{N} = 3680640$ se transforma mediante:

$$x^i = \frac{1}{5}\sum_{j=0}^{4} \tilde{x}^{5i-j} \quad \text{y} \quad t^i = \tilde{t}^{5i}\,, \qquad \forall i \in \left\{1, ..., \frac{\tilde{N}}{5}\right\}\,,$$

y se obtiene una nueva sucesión de datos de turbidez $\{(t^i, x^i)\}_{i=1}^{N}$ con $N = 736128$. Ahora, se considera una matriz bidimensional de tamaño 736128×10, con los instantes temporales como filas y la turbidez y las otras nueve variables externas como columnas.

Por otro lado, los datos recibidos de turbidez presentaban valores anómalos que no se relacionaban con eventos reales de turbidez alta, denominados *ruido*. Esto se pudo identificar gracias a que las fluctuaciones reales de la turbidez no pueden ser exponenciales, y en los datos estudiados había intervalos temporales muy cortos con grandes fluctuaciones. En definitiva, el ruido se ha definido cuidadosamente y se ha corregido para la implementación del modelo.

Definición 4.1.1. *Un valor de turbidez x^i en un instante t^i, i.e. (t^i, x^i), se considera ruido si su diferencia con el promedio de los últimos 10 valores, no considerados ruido, supera en 10 veces dicho promedio. Esto es, se define $\mathcal{R}$ el ruido del dataset como $i \in \mathcal{R}$ si*

$$|x^i - \bar{x}^i| < 10 \cdot \bar{x}^i \quad con \quad \bar{x}^i = \frac{1}{10} \sum_{j=1}^{10} x^{i_j} \ ,$$

donde $i_{j'} < i_j < i$ si $j' > j$ y tal que $\forall i_k \in (i_{j'}, i)$ con $i_k \neq i_j$, $i_k \in \mathcal{R}$.

Una vez identificado el ruido $\mathcal{R}$, se suprime mediante una interpolación lineal de los valores que no sean ruido más próximos para cada valor de ruido, es decir,

$$\forall i \in \mathcal{R} \quad \Longrightarrow \quad x^i = x^{j_1} + \frac{x^{j_2} - x^{j_1}}{j_2 - j_1} \cdot (i - j_1) \ ,$$

con $j_1 < i$ tal que $\forall j' \in (j_1, i)$, $j' \in \mathcal{R}$ y $j_2 > i$ tal que $\forall j' \in (i, j_2)$, $j' \in \mathcal{R}$. Un ejemplo de este procedimiento puede verse en la figura 4.3.

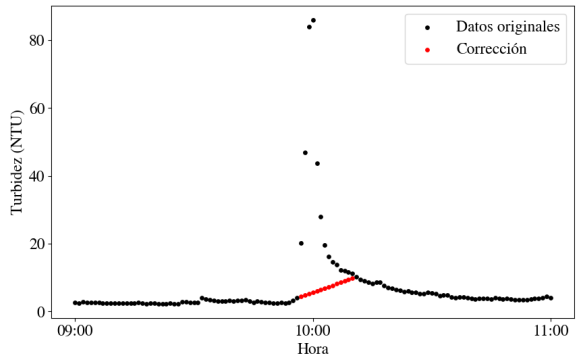


Figura 4.3: *Transformación de los datos de turbidez para eliminar el ruido.*

Por último, para el modelo predictivo centrado únicamente en predecir el nivel de turbidez en un instante τ dado (Modelo 2), se definen tres clases de turbidez distintas con valores $0, 1$ y 2, y se añade una columna al *dataset* con la clase asociada al valor de turbidez real x_i en cada instante t_i:

- **Clase 0**: nivel de turbidez baja, con $x \in [0, a_1)$. En este caso se ha utilizado $a_1 = 15$ NTU.
- **Clase 1**: nivel de turbidez media, con $x \in [a_1, a_2)$. En este caso se ha utilizado $a_2 = 40$ NTU.
- **Clase 2**: nivel de turbidez alta, con $x \in [a_2, \infty)$. En este caso el límite superior es 100 NTU debido a la saturación del sensor.

Esta transformación puede observarse en la figura 4.4, en la que se muestra en negro el valor exacto de la turbidez para cada instante temporal, y con franjas de colores cada una de las tres clases: verde para la clase 0, amarillo para la 1 y rojo para la 2. Se puede comprobar a simple vista que el número de puntos que forma parte de cada una de las tres clases no es equivalente (un 97,2 % de eventos de clase 0, un 2,3 % de clase 1 y un 0,5 % de clase 2), siendo que las clases no están balanceadas y se deberán introducir distintos pesos para compensarlo.

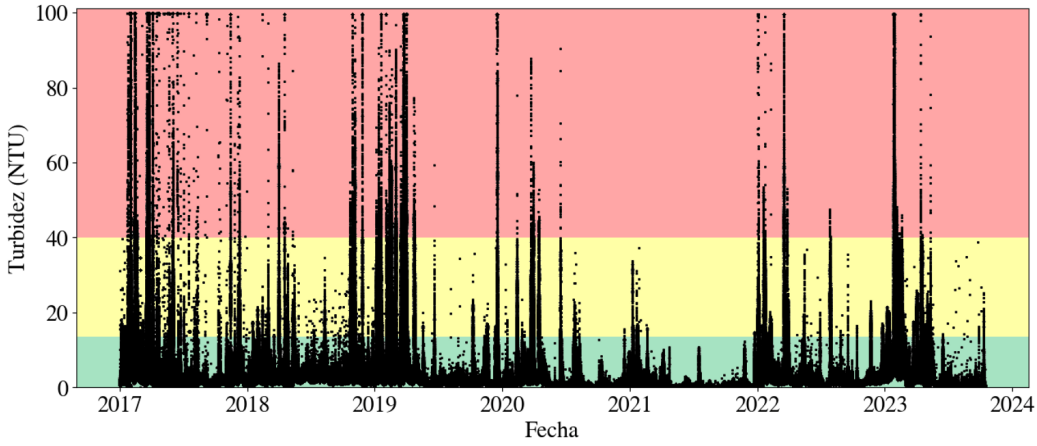


Figura 4.4: *Clasificación en tres clases (presentadas en verde, amarillo y rojo) de los datos de turbidez (en negro) empleados en el desarrollo del modelo.*

4.2. Características del modelo

Una vez preprocesados los datos, el siguiente paso es decidir las características principales que van a indicar el tipo de modelo que se va a utilizar. Esto incluye: inicialización, número de capas, tamaño de las capas, tipo de red, funciones de activación, optimizador y función de pérdida.

La inicialización en ambos modelos se realiza mediante la distribución de inicialización normalizada presentada por Xavier Glorot y Yoshua Bengio, y que se ha descrito en la Sección *3.3.1*. El empleo de esta inicialización se ha debido a su adaptabilidad a las redes LSTM y su impacto en la mejora del rendimiento de la red.

Por otro lado, dada la necesidad de emplear una red neuronal que pudiese relacionar varias variables con su histórico temporal, la red utilizada ha sido, para ambos modelos, una red LSTM. Este tipo de red, que se ha comentado en profundidad en la Sección *3.5.3*, es un tipo de red neuronal recurrente (RNN) que evita la pérdida de información que podría ser relevante a largo plazo, lo que es clave en el problema que se está tratando dado el bajo número de eventos de turbidez con los que se entrena y su separación temporal.

Además, para ambos modelos se han usado redes neuronales con 1.259.235 parámetros distribuidos en seis capas, cuyo tamaño se incrementa en las dos capas centrales y disminuye en los extremos. Entre cada par de capas se ha introducido una capa de *dropout* con proporción también creciente en las capas intermedias, que, como se ha explicado en la Sección *3.2*, disminuye el riesgo de sobreajuste en la red. El motivo de la variabilidad en las capas está relacionada con una mejor convergencia del modelo, así como con evitar problemas debidos a funciones de pérdida no tratables computacionalmente.

La función de activación empleada en las capas ha sido la función lineal rectificadora (ReLU), presentada en la Sección *3.3.2*, dada su optimalidad para mantener un valor cero verdadero y su simpleza computacional, lo que mejora el rendimiento.

El optimizador elegido ha sido *Adam*, que, como se ha descrito en la Sección *3.1.1*, se adapta al entrenamiento de grandes volúmenes de datos, considerando momentos de primer y segundo orden en el descenso de gradiente, que aumentan el ritmo de convergencia y evitan inestabilidades.

Por último, aunque todas las características descritas son comunes a los dos modelos, estos se diferencian en su última capa y en su función de pérdida:

- Para el Modelo 1, la función de pérdida empleada ha sido el error cuadrático medio (3.11), que, como se ha estudiado en la Sección *3.3.3*, permite optimizar el resultado obtenido a través de su diferencia con el valor real de la turbidez en un instante temporal. Además, la última capa elegida ha sido una capa densa completamente conectada y con la identidad como función de activación, adaptada al modelo de regresión desarrollado.
- Para el Modelo 2, la función de pérdida utilizada ha sido de entropía cruzada categórica (3.15). En este caso es una función de pérdida adecuada porque se tiene un problema de clasificación multiclase y, como se comentó en la Sección *3.3.3*, permite emplear más de dos clases y asignar una probabilidad a cada una, que es el valor que se compara con el resultado real. Asimismo, la última capa empleada también ha sido una capa densa completamente conectada, pero con función de activación *softmax* para adaptarla al modelo de clasificación desarrollado.

4.2.1. Pesado de las distintas clases

Dada la complejidad de los datos de entrada y la gran diferencia en el volumen de datos en cada una de las clases (véase la figura 4.4), en el Modelo 2 se ha asignado un peso variable a cada clase y para cada época en la función de pérdida, como se indica en la ecuación (3.16). Esto ha permitido conseguir una mejor convergencia del modelo y una mayor precisión en los resultados obtenidos. El pesado de las clases no ha sido constante en todas las épocas del entrenamiento, apoyándose esta decisión en el siguiente argumento:

Inicialmente se ha considerado un pesado brusco que penalizase en exceso la predicción errónea de las clases menos representadas, para conseguir una sobrepredicción de ellas y una mejor convergencia del modelo. Después, este pesado se ha ido reduciendo y adaptando al balance adecuado de las clases en función de la cantidad de datos en cada una, para finalizar con una época de entrenamiento en la que el pesado es inversamente proporcional al número de eventos de cada clase.

En definitiva, sea $\epsilon \in \{1, \dots, \tilde{\epsilon}\}$ la época del entrenamiento considerada, el peso de cada una de las clases ha sido, sabiendo que el número de datos de clase 0 es N_0, el de la clase 1 es N_1 y el de la clase 2 es N_2, cumpliendo que $N_0 \gg N_1 > N_2$:

$$\begin{aligned} W_0 &= \frac{N_0 + N_1 + N_2}{N_0} \cdot \exp\left(-\frac{\tilde{\epsilon} - \epsilon}{\tilde{\epsilon}}\right) , \\ W_1 &= \frac{N_0 + N_1 + N_2}{N_1} \cdot \exp\left(\frac{\tilde{\epsilon} - \epsilon}{2\tilde{\epsilon}}\right) , \\ W_2 &= \frac{N_0 + N_1 + N_2}{N_2} \cdot \exp\left(\frac{\tilde{\epsilon} - \epsilon}{\tilde{\epsilon}}\right) . \end{aligned} \tag{4.1}$$

4.3. Resultados obtenidos

En esta sección, además de presentar los resultados de los dos modelos desarrollados, se comentarán las diferencias y similitudes encontradas en los entrenamientos, validaciones y capacidad predictiva de ambos. Para ambos modelos se entrenará con el periodo 2017-2023 (630.908 registros) y se validará con el periodo 2023-2024 (82.393 registros).

4.3.1. Modelo 1

La principal limitación en el modelo 1 es su sensibilidad a los datos y la complicada convergencia que presenta. Dado que el 97,2 % de los datos de entrada ya normalizados son inferiores a 0,1, el modelo difícilmente se aleja de una predicción estacionaria en valores bajos de turbidez constantes, y cuando sí se acerca a otros tipos de predicciones, tras una o dos épocas de entrenamiento sale de dicha tendencia para volver a la estacionaria. Así, la mayoría de modelos entrenados de esta forma consisten en una predicción constante de turbidez nula.

Por otro lado, dado que solo un 2,8 % de los datos afecta de manera intensa a la función de pérdida cuando esta está en la solución estacionaria, los valores de precisión para modelos erróneos son más altos que para modelos que tienden a otros tipos de resultados. Esto implica que la métrica habitual para definir la precisión no es correcta, por lo que se ha considerado como precisión la siguiente relación:

$$\text{Precisión} = 1 - \frac{1}{N} \sum_{i=1}^{N} \frac{|y^i - \hat{f}(\boldsymbol{x}^i)|^2}{y^i} , \tag{4.2}$$

donde $\hat{f}(\boldsymbol{x}^i)$ son los valores predichos y y^i los valores reales, que para considerarse en la suma deben cumplir que $y^i \geq a_1$, con a_1 el valor de la clase 1 del modelo 2. Esto es, *grosso modo*, la precisión que tiene el modelo para predecir los picos intensos de turbidez en función del valor promedio de dichos eventos, que son los que tienen interés práctico.

De esta forma, tras entrenar el modelo durante 100 épocas, en lotes de 1000 muestras y adaptando la razón de aprendizaje en las distintas épocas para facilitar la convergencia, se ha llegado a los resultados mostrados en las figuras 4.5 y 4.6, para el entrenamiento y la validación, respectivamente.

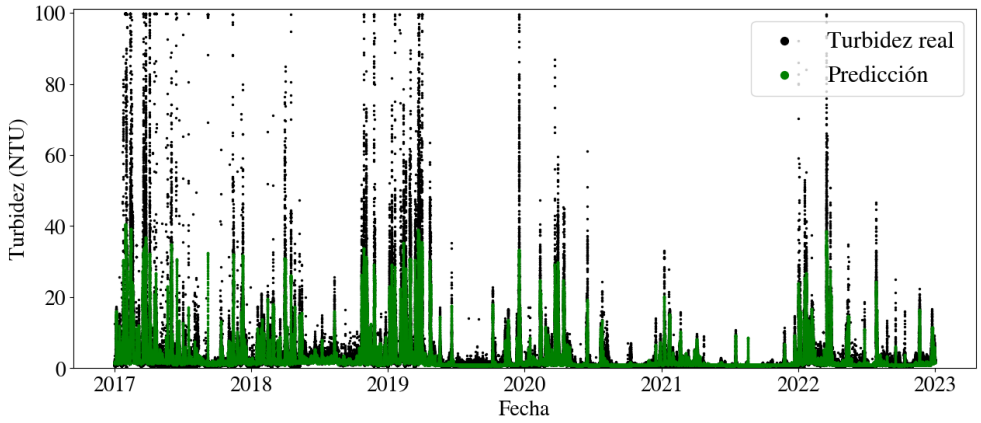


Figura 4.5: *Predicciones del modelo 1 para el periodo de entrenamiento.*

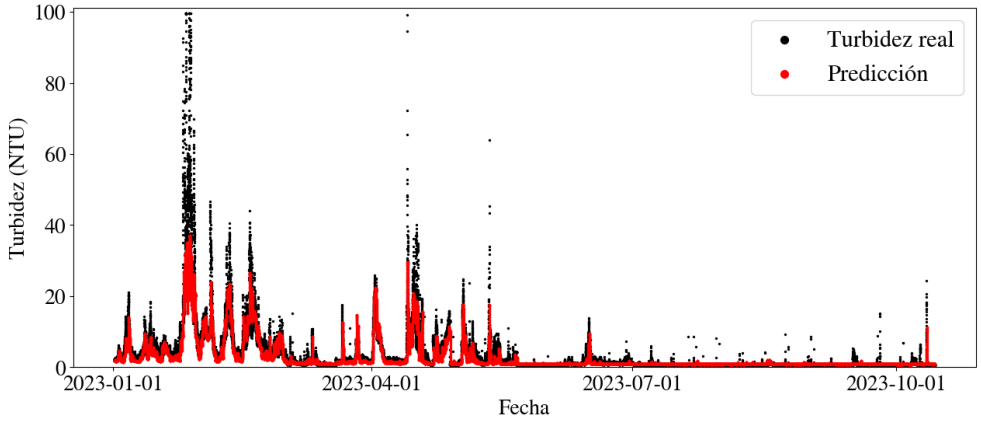


Figura 4.6: *Predicciones del modelo 1 para el periodo de validación.*

Se comprueba que, con la precisión definida en (4.2), los resultados han sido los mostrados en la tabla 4.1. La precisión del modelo no es aceptable a nivel práctico, pues no consigue acercarse a los valores reales de turbidez en los momentos de turbidez elevada. De hecho, se muestra así que abordar la resolución del problema de la turbidez mediante su predicción exacta resulta complejo y poco preciso, siendo necesario un enfoque alternativo, como el del modelo 2.

	Entrenamiento	Validación
Precisión	0,553	0,496

Cuadro 4.1: *Precisión obtenida con el modelo 1, dada por la ecuación* (4.2).

4.3.2. Modelo 2

Para el modelo 2 la principal dificultad encontrada ha sido su convergencia. Previo a introducir el pesado de las clases la red mostraba características similares a las del modelo 1, siendo muy improbable que se produjese una predicción distinta de la clase 0, por ser la clase mayoritaria. Sin embargo, una vez introducidos los pesos, el problema ha sido la convergencia, pues según se entrenaba el modelo se obtenían otros que divergían, con funciones de pérdida tendiendo a infinito, o que caían en la solución estacionaria de clase 0. Aun así, una vez solventado este problema se ha conseguido un modelo que, tras distintos entrenamiento, se mantiene en una solución estable distinta de la que asigna clase 0 en cada ocasión.

En este caso, dado que los errores en la clasificación no son equivalentes, la definición de la precisión es importante. De manera clara, una clasificación de un evento de clase 0 como clase 2, o viceversa, es menos aceptable que uno de clase 1 como clase 2, luego pierde el sentido utilizar la definición usual de la precisión como la proporción de eventos que clasifica correctamente. En este caso, se define la precisión como 1 menos el promedio de la diferencia en valor absoluto entre la clase correcta y la predicha, lo que penaliza más las clasificaciones extremas incorrectas. Así:

$$\text{Precisión} = 1 - \frac{1}{N}\sum_{i=1}^{N} |y^i - \hat{f}(\boldsymbol{x}^i)| \,, \tag{4.3}$$

donde y^i es la clase correcta y $\hat{f}(\boldsymbol{x}^i) \in \{0, 1, 2\}$ es la clase predicha.

En este caso, considerando la función de pérdida con los pesos (4.1), se entrena el modelo durante 100 épocas, en lotes de 1000 muestras y adaptando la razón de aprendizaje, llegando así a los resultados mostrados en las figuras 4.7 y 4.8, para el entrenamiento y la validación, respectivamente.

Como se puede comprobar, las precisiones mostradas en la tabla 4.2 para el modelo 2 mejoran considerablemente a la precisión del modelo 1, siendo entonces el modelo de clasificación más preciso y fiable. De esta manera, cabe destacar que, al no predecirse un valor exacto de turbidez, se ha decidido reducir la exactitud de los resultados en favor de mejorar la precisión de estos.

	Entrenamiento	Validación
Precisión	0,951	0,938

Cuadro 4.2: *Precisión obtenida con el modelo 2, dada por la ecuación* (4.3).

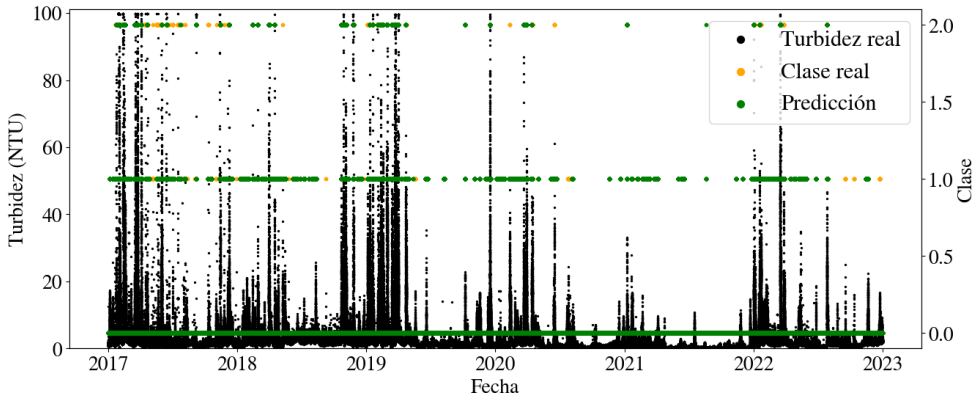


Figura 4.7: *Predicciones del modelo 2 para el periodo de entrenamiento.*

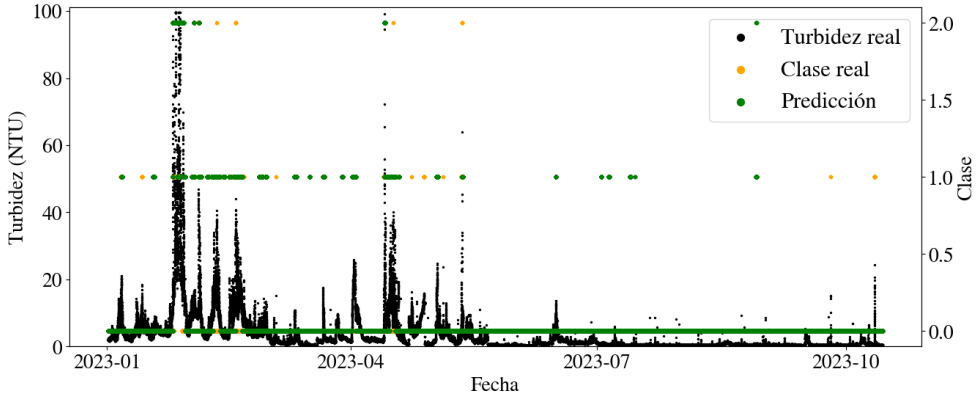


Figura 4.8: *Predicciones del modelo 2 para el periodo de validación.*

De hecho, para comprobar el desempeño real del modelo predictivo, se puede definir una precisión alternativa que únicamente tenga en cuenta los sucesos de turbidez reales catalogados como clase 1 o 2, i.e. modificar la ecuación (4.3) con los y_i tales que $y_i \in \{1, 2\}$. Para esta precisión alternativa se obtienen los resultados mostrados en la tabla 4.3, que indican que, en efecto, el modelo presentado es adecuado y consigue predecir correctamente lo eventos de turbidez.

	Entrenamiento	Validación
Precisión	0,703	0,659

Cuadro 4.3: *Precisión alternativa (solo turbidez alta) obtenida con el modelo 2.*

Por otro lado, en las figuras 4.9a y 4.9b se puede observar la matriz de confusión (véase A.8 del capítulo de apéndices para más información sobre matrices de confusión) para los resultados del modelo, en entrenamiento y validación, respectivamente. Entre muchas otras conclusiones, de dichas matrices de confusión se puede extraer que el 87,9 % de los picos de turbidez se predicen al menos como un pico intermedio, y que solo el 4,4 % de las veces que el modelo predice que va a haber un pico intenso, en realidad no había pico.

(a) *Periodo de entrenamiento.* (b) *Periodo de validación.*

Figura 4.9: *Matrices de confusión asociadas al modelo 2.*

Debido a estos y otros beneficios que este modelo predictivo de la turbidez supone para la empresa ACCIONA, a día de hoy se está implementando como un modelo preliminar en una de sus plantas desaladoras, con la intención que sea una herramienta de utilidad diaria.

Capítulo 5

Conclusiones

Como se ha podido vislumbrar en este documento, y como se puede comprobar en la literatura, el aprendizaje automático y las redes neuronales abarcan una gran cantidad de métodos para atajar problemas reales y abstractos. Tantas son las posibilidades que en ocasiones es complicado decidir qué tipo de red y qué tipo de aprendizaje son los más adecuados para un problema particular.

Aunque se han deducido resultados que afirman la universalidad de las redes neuronales como aproximadores, sus demostraciones no son constructivas, lo que lleva al problema de identificar cómo optimizar los modelos de predicción para que estimen adecuadamente el caso tratado. Además, debe entenderse también cómo conseguir que el modelo generalice y prediga correctamente datos de entrada con los que no ha entrenado. Esto es precisamente lo que se ha intentado, y conseguido, para la predicción de la turbidez del agua en plantas de desalinización.

Dado que el problema planteado por ACCIONA no era sencillo e implicaba buscar correlaciones temporales entre distintas variables y la turbidez, el empleo de redes prealimentadas no era posible. Por ello, se han desarrollado y comparado dos modelos que involucran redes neuronales recurrentes, en particular con memoria a corto-largo plazo, capaces de aportar información sobre la turbidez en un instante futuro. Se ha visto entonces que priorizar la precisión en la predicción frente a la obtención de resultados más informativos, i.e. priorizar un modelo que prediga la clase de turbidez y no el valor exacto de ella, es conveniente y útil para la resolución del problema en este caso particular.

Finalmente, se ha conseguido abordar con éxito un problema real relacionado con la predicción de eventos meteorológicos, aunque con un carácter más sencillo por el menor número de variables implicadas. Dados los beneficios del modelo desarrollado, este se ha implementado en la planta desaladora, por lo que será útil para su operatividad y repercutirá de manera directa en una parte fundamental de la vida de las personas: el **suministro de agua**.

Apéndice A

Definiciones y teoremas relevantes

A continuación, se presentan definiciones y teoremas complementarios al desarrollo presentado en el trabajo, pero cuya consulta puede ser interesante.

A.1. Teorema de Bayes

El teorema de Bayes relaciona la probabilidad condicionada de un suceso en una hipótesis con la probabilidad *a priori* y la probabilidad *a posteriori* [18]:

Teorema A.1.1. *Sean $\{A_1, \ldots, A_n\}$ un conjunto de sucesos excluyentes y exhaustivos tales que $P(A_i) \neq 0$, $\forall i \in \{1, \ldots, N\}$, para un suceso B cualquiera se tiene que:*

$$P(A_j|B) = \frac{P(A_j) \cdot P(B|A_j)}{\sum_{i=1}^{N} P(A_i) \cdot P(B|A_i)} \quad \textit{para todo } j \in \{1, \ldots, n\}.$$

Demostración. Dado que $P(A_i \cap B) = P(A_i) \cdot P(B|A_i) = P(B) \cdot P(A_i|B) = P(B \cap A_i)$ para todo $i \in \{1, \ldots, N\}$, entonces, por la probabilidad total de los conjuntos exahustivos y excluyentes:

$$P(B) = \sum_{i=1}^{N} P(B \cap A_i) = \sum_{i=1}^{N} P(A_i) \cdot P(B|A_i) \quad \Longrightarrow$$

$$\Longrightarrow \quad P(A_j|B) = \frac{P(A_j) \cdot P(B|A_j)}{P(B)} = \frac{P(A_j) \cdot P(B|A_j)}{\sum_{i=1}^{N} P(A_i) \cdot P(B|A_i)} \, .$$

■

A.2. Modelos bayesianos de clasificación lineal

En este los modelos bayesianos de clasificación de K clases no se busca obtener directamente la probabilidad *a posteriori* $p(C_k|\boldsymbol{x}^i)$, sino que se infieren mediante modelización las densidades de

probabilidad condicionadas a las clases, $p(\boldsymbol{x}^i|C_k)$, y las probabilidades de las distintas clases, $p(C_k)$. Posteriormente, con dichas probabilidades y la ecuación (2.8) se pueden obtener analíticamente las probabilidades *a posteriori* buscadas para la clasificación.

En general, si el conjunto de entrenamiento T es suficientemente grande, la probabilidad de la clase $p(C_k)$ tiene sentido que se estime como la fracción de puntos de puntos del entrenamiento cuya etiqueta correcta pertenece a la clase k, como se demuestra en el teorema A.2.2 siguiente. Por otro lado, para las densidades de probabilidad condicionadas a las clases puede realizarse un estudio análogo al realizado en la SECCIÓN *2.1.1*.

En este caso, la distribución que siguen estas probabilidades es una distribución gaussiana multivariable, i.e. una generalización a un espacio K-dimensional de la distribución normal, donde se tiene un vector aleatorio $\boldsymbol{x}$, un vector de medias $\boldsymbol{\mu}$ y la matriz de covarianza Σ [4]:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right) .$$

En particular, la matriz de covarianza es la generalización del concepto de varianza para una variable aleatoria escalar al vector de variables aleatorias. Esta matriz es cuadrada e incluye la covarianza de los elementos del vector [15]:

Definición A.2.1. *Sea $X = (X_1, \ldots, X_n)^T$ un vector de variables aleatorias, entonces la matriz de covarianza es Σ de dimensión $n \times n$ tal que:*

$$\Sigma_{ij} = Cov(X_i, X_j) = E\left[(X_i - E(X_i))(X_j - E(X_j)\right] ,$$

donde $E(X)$ es el valor esperado de la variable aleatoria X.

Teorema A.2.2. *Dado un conjunto de entrenamiento $T = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^N, y^N)\}$, se considera un modelo bayesiano con dos clases, C_0 y C_1, de forma que $y^i = 0$ si se tiene la clase C_0 y $y^i = 1$ si se tiene la clase C_1. Suponiendo que las densidades de probabilidad de las clases son $p(C_0) = \rho$ y $p(C_1) = 1 - \rho$, y las densidades de probabilidad condicionadas a las clases siguen distribuciones gaussianas con matriz de covarianza $\boldsymbol{\Sigma}$ compartida, entonces:*

- *La estimación de máxima verosimilitud para ρ es la fracción del número de puntos en la clase C_0 frente al total:*

$$\rho^* = \frac{N_0}{N} = \frac{N_0}{N_0 + N_1} .$$

- *La estimación de máxima verosimilitud para las medias $\boldsymbol{\mu}_0$ y $\boldsymbol{\mu}_1$ de las densidades de probabilidad condicionadas a las clases son, respectivamente, la media de los datos de entrada asociados a cada una de las clases:*

$$\boldsymbol{\mu}_0 = \frac{1}{N_0}\sum_{i=1}^{N}(1-y^i)\boldsymbol{x}^i \quad y \quad \boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{i=1}^{N} y^i\boldsymbol{x}^i .$$

Demostración. [4]. Dado un dato de entrada $\boldsymbol{x}^i$ de la clase C_0, se tiene que $y^i = 0$ y entonces:

$$p(\boldsymbol{x}^i, C_0) = p(C_0)p(\boldsymbol{x}^i|C_0) = \rho \cdot \mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}) \,.$$

Por otro lado, para la clase C_1 y $y^i = 1$ se tiene que:

$$p(\boldsymbol{x}^i, C_1) = p(C_1)p(\boldsymbol{x}^i|C_1) = (1-\rho) \cdot \mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) \,.$$

De esta manera, la función de verosimilitud es, siendo $Y = (y_1, \ldots, y_N)^T$:

$$\mathcal{L}(\rho, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1) = p(Y|\rho, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} \left(\rho \cdot \mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_0, \boldsymbol{\Sigma})\right)^{1-y^i} \cdot \left([1-\rho] \cdot \mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right)^{y^i} \,.$$

Considerando el logaritmo neperiano de la función de verosimilitud y maximizándolo:

$$\begin{aligned}\mathcal{E}(\rho, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1) &= \log\left(p(Y|\rho, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right) = \\ &= \sum_{i=1}^{N} (1-y^i) \cdot \left[\log(\rho) + \log\left(\mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_0, \boldsymbol{\Sigma})\right)\right] + y^i \cdot \left[\log(1-\rho) + \log\left(\mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right)\right]\end{aligned}$$

$$\frac{\partial \mathcal{E}}{\partial \rho} = \sum_{i=1}^{N} \frac{1-y^i}{\rho} - \frac{y^i}{1-\rho} = \sum_{i=1}^{N} \frac{1-y^i-\rho}{\rho(1-\rho)} = \frac{-1}{\rho(\rho-1)} \left(N\rho - \sum_{i=1}^{N}(1-y^i)\right) = 0 \,.$$

De aquí se deduce directamente que:

$$\rho^* = \frac{1}{N}\sum_{i=1}^{N}(1-y^i) = \frac{N_0}{N} \,.$$

Considerando ahora la maximización con respecto a $\boldsymbol{\mu}_0$:

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \boldsymbol{\mu}_0} &= \frac{\partial}{\partial \boldsymbol{\mu}_0}\left(\sum_{i=1}^{N} y^i \log\left(\mathcal{N}(\boldsymbol{x}^i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma})\right)\right) = \frac{\partial}{\partial \boldsymbol{\mu}_0}\left(-\sum_{i=1}^{N}\frac{y^i}{2}(\boldsymbol{x}-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_0)\right) = \\ &= \sum_{i=1}^{N} y^i\left(\boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x}^i - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0\right) = -\boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1} \cdot \left(N_1\boldsymbol{\mu}_0 - \sum_{i=1}^{N} y^i\boldsymbol{x}^i\right) = 0 \quad \Longrightarrow \\ &\Longrightarrow \quad \boldsymbol{\mu}_0^* = \frac{1}{N_1}\sum_{i=1}^{N} y^i\boldsymbol{x}^i \qquad \text{y, análogamente,} \quad \boldsymbol{\mu}_1^* = \frac{1}{N_1}\sum_{i=1}^{N}(1-y^i)\boldsymbol{x}^i \,.\end{aligned}$$

Finalmente, se ha obtenido el resultado buscado. ■

A.3. Condiciones de Karush-Kuhn-Tucker

Las condiciones de Karush-Kuhn-Tucker (KKT) imponen ciertas características, i.e. condiciones necesarias, sobre los mínimos locales de una función f y N restricciones g_i. En particular, se basan en el siguiente teorema, cuya demostración puede encontrarse en [23]:

Teorema A.3.1. *Sea un problema de optimización, con* $\boldsymbol{x} \in \mathbb{R}^M$*, dado por*

$$\begin{aligned} &\text{mín} && \{f(\boldsymbol{x})\} \\ &s.a && g_i(\boldsymbol{x}) \leq 0\,, \quad \forall\, i \in \{1,\ldots,N\} \end{aligned}$$

y se define $\mathcal{G}(\boldsymbol{x})$ *el conjunto de índices de las restricciones activas, i.e.* $g_i(\boldsymbol{x}) = 0$*. Si las funciones* f, g_i *con* $i \in \mathcal{G}(\boldsymbol{x})$ *son diferenciables y las* g_i *con* $i \notin \mathcal{G}(\boldsymbol{x})$ *son continuas, y además los gradientes* $\nabla g_i(\boldsymbol{x})$ *con* $i \in \mathcal{G}(\boldsymbol{x})$ *son linealmente independientes. Entonces, si* $\tilde{\boldsymbol{x}}$ *es un mínimo local de* f *y es factible, i.e. cumple las condiciones* $g_i(\tilde{\boldsymbol{x}}) \leq 0$*, existen escalares* $u_i \geq 0$ *con* $i \in \{1,\ldots,N\}$ *tales que la función lagrangiana* $L(\tilde{\boldsymbol{x}}) = f(\tilde{\boldsymbol{x}}) + \sum_{i=1}^{N} u_i g_i(\tilde{\boldsymbol{x}})$*, cumple que:*

$$\begin{aligned} &\frac{\partial L}{\partial \tilde{x}_j} = 0\,, \quad \forall\, j \in \{1,\ldots,M\} \\ &u_i g_i(\tilde{\boldsymbol{x}}) = 0\,, \quad \forall\, i \in \{1,\ldots,N\}\,. \end{aligned}$$

Cabe destacar que, si $\boldsymbol{x}_0$ es un punto que cumple las condiciones anteriores, i.e. las condiciones KKT, y las funciones f, g_i con $i \in \mathcal{G}(\boldsymbol{x}_0)$ son convexas, entonces $\boldsymbol{x}_0$ es un mínimo global. La demostración puede consultarse en [23].

Observación A.3.2. *Una función* $f : \mathbb{R}^M \to \mathbb{R}$ *es convexa si para cualesquiera* $\boldsymbol{x}_1$ *y* $\boldsymbol{x}_2$ *se tiene que* $f(\lambda \boldsymbol{x}_1 + (1 - \lambda \boldsymbol{x}_2) \leq \lambda \cdot f(\boldsymbol{x}_1) + (1 - \lambda) \cdot f(\boldsymbol{x}_2)$*, para todo* $\lambda \in [0, 1]$*.*

A.4. Teoría de grafos

En teoría de grafos se utilizan ciertos conceptos que son útiles a la hora de representar las redes neuronales [24]:

Definición A.4.1. *Un grafo orientado* G *es un par* $G = \{V, A\}$ *donde* V *es un conjunto de vértices y* A *es un conjunto de pares ordenados* (i, j) *con* $i, j \in V$*, llamados arcos; el vértice* i *se llama vértice inicial y el vértice* j *se llama vértice final. Un arco del tipo* (i, i) *se llama un bucle.*

Un grafo orientado G se dice que es simple no tiene bucles ni arcos repetidos. Además, se define el conjunto de sucesores de un vértice i como $\Gamma^+(i) = \{j : (i, j) \in A\}$, y el de predecesores como $\Gamma^-(i) = \{j : (j, i) \in A\}$.

Definición A.4.2. *Si existe al menos un arco uniendo cada par de vértices, el grafo es completo.*

A.5. Resultados empleados en la demostración del teorema de aproximación universal de Cybenko

En la demostración del teorema de aproximación universal de Cybenko se han utilizado tres resultados importantes que se muestran por orden: el teorema de Hanh-Banach, el teorema de

representación de Riesz y un lema sobre funciones continuas sigmoideas.

Teorema A.5.1. *Sea S un subespacio vectorial de un espacio vectorial normado X y $\boldsymbol{x}_0 \in X$. Entonces $\boldsymbol{x}_0 \in \bar{S}$ si y solo si no existe un funcional lineal acotado L definido en X tal que $L(\boldsymbol{x}) = 0$ para todo $\boldsymbol{x} \in S$ pero $L(\boldsymbol{x}_0) \neq 0$.*

Demostración. La demostración puede consultarse en [10]. ■

Teorema A.5.2. *Sea Y un espacio de Hausdorff localmente compacto, como puede ser $[0,1]^M$, entonces para cada funcional lineal acotado L definido en $\mathcal{C}(Y)$ existe una medida de Borel μ tal que, para cada $f \in \mathcal{C}(Y)$, se tiene que:*

$$L(f) = \int_Y f(\boldsymbol{x}) d\mu(\boldsymbol{x}) \, .$$

Demostración. La demostración puede consultarse en [2]. ■

Lema A.5.3. *Una función $\sigma : \mathbb{R} \to \mathbb{R}$ continua sigmoidea satisface que, dada μ una medida de Borel sobre $[0,1]^M$, la medida es nula, i.e. $\mu = 0$, si se cumple que, para todo $\boldsymbol{w} \in \mathbb{R}^M$ y $b \in \mathbb{R}$:*

$$\int_{[0,1]^M} \sigma\left(\boldsymbol{x}^T \cdot \boldsymbol{w} + b\right) d\mu(\boldsymbol{x}) = 0 \, .$$

Demostración. La demostración puede consultarse en [26]. ■

A.6. Señales de error para la capa de salida

Las señales de error de las neuronas de la capa de salida, asociadas a las funciones de pérdida descritas en este documento para una iteración, son, haciendo uso de la relación (3.17):

- **Error cuadrático medio** (3.11), con $\hat{f}(\boldsymbol{x}) = \hat{y}^{(Z)} \in \mathbb{R}$, luego $|V_Z| = 1$:

$$\begin{aligned} \delta^{(Z)} = \frac{\partial \mathcal{E}}{\partial a^{(Z)}} = \frac{\partial}{\partial a^{(Z)}} \left[\frac{1}{N}\left(y - \hat{f}(\boldsymbol{x})\right)^2\right] = \frac{\partial}{\partial a^{(Z)}} \left[\frac{1}{N}\left(y - \hat{y}^{(Z)}\right)^2\right] = \\ = \frac{\partial}{\partial a^{(Z)}} \left[\frac{1}{N}\left(y - \psi^{(Z)}\left(a^{(Z)}\right)\right)^2\right] = -\frac{2}{N} \cdot \left(y - \psi^{(Z)}\left(a^{(Z)}\right)\right) \cdot \left[\psi^{(Z)}\left(a^{(Z)}\right)\right]' \, . \end{aligned}$$

  Expresando la relación en términos de la salida de la red, $\hat{f}(\boldsymbol{x})$, dado que, para un modelo de regresión con error cuadrático medio, la función de activación de la última capa es la identidad $\psi^{(Z)}(a^{(Z)}) = a^{(Z)}$, entonces $\left[\psi^{(Z)}(a^Z)\right]' = 1$ y queda:

$$\delta^{(Z)} = \frac{2}{N} \cdot \left(\hat{f}(\boldsymbol{x}) - y\right) \, . \tag{A.1}$$

- **Entropía cruzada** (3.13), con $\hat{f}(\boldsymbol{x}) = \hat{y}^{(Z)} \in [0,1]$, luego $|V_Z| = 1$:

$$\begin{aligned}
\delta^{(Z)} &= \frac{\partial \mathcal{E}}{\partial a^{(Z)}} = \frac{\partial}{\partial a^{(Z)}} \left[-y \cdot \log\left(\hat{f}(\boldsymbol{x})\right) - (1-y) \cdot \log\left(1 - \hat{f}(\boldsymbol{x})\right) \right] = \\
&= \frac{\partial}{\partial a^{(Z)}} \left[-y \cdot \log\left(\psi^{(Z)}\left(a^{(Z)}\right)\right) - (1-y) \cdot \log\left(1 - \psi^{(Z)}\left(a^{(Z)}\right)\right) \right] = \\
&= -y \cdot \frac{\left[\psi^{(Z)}\left(a^{(Z)}\right)\right]'}{\psi^{(Z)}\left(a^{(Z)}\right)} + (1-y) \cdot \frac{\left[\psi^{(Z)}\left(a^{(Z)}\right)\right]'}{1 - \psi^{(Z)}\left(a^{(Z)}\right)} = \\
&= \frac{\psi^{(Z)}\left(a^{(Z)}\right) - y}{\left[\psi^{(Z)}\left(a^{(Z)}\right)\right] \cdot \left[1 - \psi^{(Z)}\left(a^{(Z)}\right)\right]} \cdot \left[\psi^{(Z)}\left(a^{(Z)}\right)\right]' .
\end{aligned}$$

Ahora, como la función de activación de la última capa es la función logística sigmoide (3.7) con $\alpha = 1$, entonces se tiene que:

$$\begin{aligned}
\left[\psi^{(Z)}\left(a^{(Z)}\right)\right]' &= \left[\sigma\left(a^{(Z)}\right)\right]' = \left[\frac{1}{1 + \exp\left(-a^{(Z)}\right)}\right]' = \frac{\exp\left(-a^{(Z)}\right)}{\left[1 + \exp\left(-a^{(Z)}\right)\right]^2} = \\
&= \frac{1}{1 + \exp\left(-a^{(Z)}\right)} \cdot \left[1 - \frac{1}{1 + \exp\left(-a^{(Z)}\right)}\right] = \sigma\left(a^{(Z)}\right) \cdot \left[1 - \sigma\left(a^{(Z)}\right)\right] = \\
&= \left[\psi^{(Z)}\left(a^{(Z)}\right)\right] \cdot \left[1 - \psi^{(Z)}\left(a^{(Z)}\right)\right] .
\end{aligned}$$

De esta forma, para la entropía cruzada queda:

$$\delta^{(Z)} = \hat{f}(\boldsymbol{x}) - y \,. \tag{A.2}$$

- **Entropía cruzada categórica** (3.15), con $\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{y}}^{(Z)} \in [0,1]^K$, luego $|V_Z| = K$:

$$\begin{aligned}
\delta_n^{(Z)} &= \frac{\partial \mathcal{E}}{\partial a_n^{(Z)}} = \frac{\partial}{\partial a_n^{(Z)}} \left[-\sum_{k=1}^{K} y_k \cdot \log\left(\hat{f}_k(\boldsymbol{x})\right) \right] = \\
&= \frac{\partial}{\partial a_n^{(Z)}} \left[-\sum_{k=1}^{K} y_k \cdot \log\left(\psi_k^{(Z)}\left(a_k^{(Z)}\right)\right) \right] = -y_n \cdot \frac{\left[\psi_n^{(Z)}\left(a_n^{(Z)}\right)\right]'}{\psi_n^{(Z)}\left(a_n^{(Z)}\right)} .
\end{aligned}$$

Obteniendo la derivada de la función de activación de la última capa, que en este caso es el término n-ésimo de la función *softmax* (3.8):

$$\begin{aligned}
\left[\psi_n^{(Z)}\left(a_n^{(Z)}\right)\right]' &= \left[\frac{\exp\left(a_n^{(Z)}\right)}{\sum_{k=1}^{K} \exp\left(a_k^{(Z)}\right)}\right]' = \frac{\exp\left(a_n^{(Z)}\right)}{\sum_{k=1}^{K} \exp\left(a_k^{(Z)}\right)} \left[1 - \frac{\exp\left(a_n^{(Z)}\right)}{\sum_{k=1}^{K} \exp\left(a_k^{(Z)}\right)}\right] = \\
&= \left[\psi_n^{(Z)}\left(a_n^{(Z)}\right)\right] \cdot \left[1 - \psi_n^{(Z)}\left(a_n^{(Z)}\right)\right] .
\end{aligned}$$

Así, se obtiene la relación para la señal de error de la neurona n de la capa de salida:

$$\delta_n^{(Z)} = -y_n \cdot \left[1 - \hat{f}_n(\boldsymbol{x})\right] . \tag{A.3}$$

A.7. Coeficiente de correlación de Pearson

El coeficiente de correlación de Pearson de dos variables aleatorias mide la dependencia lineal que presentan, o, dicho de otra manera, el grado de covariación entre ellas. Para el caso de dos muestras la interpretación es análoga: cuál es el grado de relación lineal entre ellas.

Definición A.7.1. *Sean* $\{x_i\}_{i=1}^N$ *y* $\{y_i\}_{i=1}^N$ *dos muestras de datos de tamaño* N*, el coeficiente de correlación muestral de Pearson,* r_{xy}*, se define como*

$$r_{xy} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}},$$

donde $\bar{x}$ *y* $\bar{y}$ *son las medias muestrales:* $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$, $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$.

Este coeficiente de correlación puede variar entre -1 y 1, siendo en ambos casos una dependencia lineal perfecta (inversa o directa, respectivamente). Por el contrario, cuando $r_{xy} = 0$ quiere decir que no existe dependencia lineal entre las muestras [15].

A.8. Matriz de confusión

Las matrices de confusión son herramientas que permiten visualizar la precisión y rendimiento de un modelo predictivo en una tarea de clasificación. Para ello, muestran en una matriz cuadrada el número de casos predichos de manera correcta e incorrecta en base a la etiqueta real.

Definición A.8.1. *Una matriz de confusión es una matriz cuadrada de tamaño* $K \times K$ *y que incluye el número de predicciones de las* K *clases en las filas, teniendo en cuenta el respectivo valor real de cada dato en las* K *columnas.*

De esta manera, en una matriz de confusión se incluyen las predicciones correctas (términos en la diagonal), los falsos positivos (términos en la parte triangular inferior), y los falsos negativos (términos en la parte triangular superior). Esto puede verse en la figura A.1. Cabe destacar que esta matriz puede encontrarse también representada como su transpuesta.

Por lo general, la escala de color presentada en la matriz equivale al porcentaje de desempeño por filas, es decir, corresponde al porcentaje de las predicciones que han asignado una cierta clase y que han sido correctas o incorrectas. Aun así, en casos con clases muy desbalanceadas, los valores que se representan no son los porcentajes sino los registros absolutos [3].

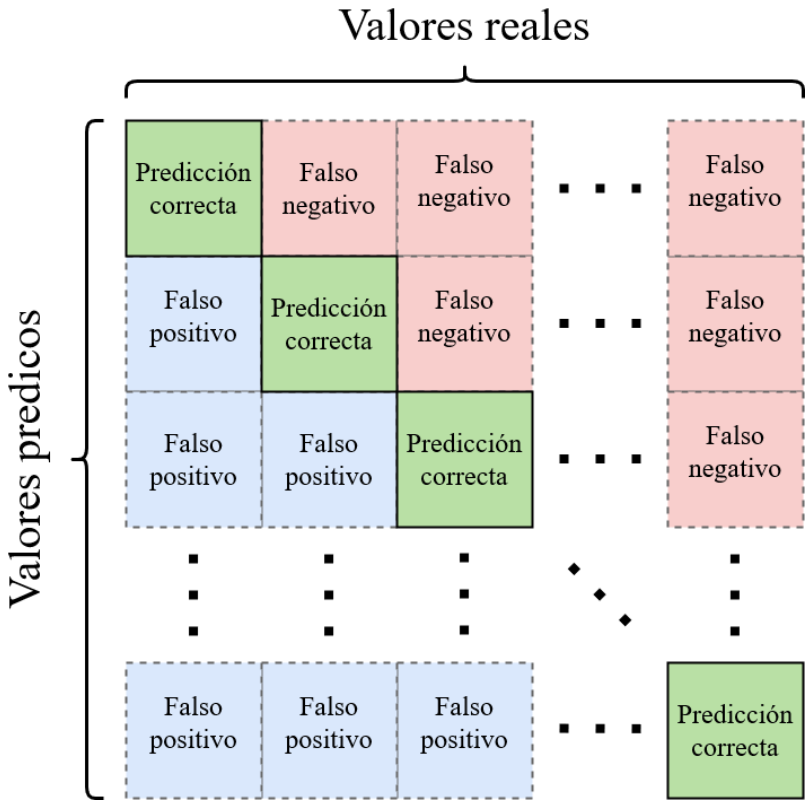


Figura A.1: *Matriz de confusión típica en un modelo de clasificación.*

Bibliografía

[1] A. Altaee, M. Hafiz, A. Al Hawari, R. Alfahel, M. Hassan, S. Zaidi, and A. Yasir. Impact of High Turbidity on Reverse Osmosis: Evaluation of Pretreatment Processes. *Desalination and Water Treatment*, 208:96–103, 08 2020.

[2] G. Bachman and L. Narici. *Functional Analysis*. Dover Publications, Mineola, New York, second edition, 2000.

[3] E. Beauxis-Aussalet and L. Hardman. Visualization of Confusion Matrix for Non-Expert Users. 2014.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, Berlin, Germany, August 17 2006.

[5] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, New York, 2nd edition, 1988.

[6] J. M. Cohen, S. Kaur, Y. Li, J. Z. Kolter, and A. Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability, 2022.

[7] T. M. Cover. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, 1965.

[8] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[9] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, Cambridge, UK, April 2020.

[10] R. E. Edwards. *Functional Analysis: Theory and Applications*. Dover Publications, 1995.

[11] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International*

Conference on Artificial Intelligence and Statistics, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] S. Haykin. *Neural Networks and Learning Machines*. Pearson, Upper Saddle River, NJ, 3rd edition, November 2008.

[14] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[15] R. V. Hogg, J. W. McKean, and A. T. Craig. *Introduction to Mathematical Statistics*. Pearson, 8th edition, 2019.

[16] S. Kim. Mathematical Foundations of Machine Learning, Enero 2024.

[17] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2017.

[18] P. M. Lee. *Bayesian Statistics: An Introduction*. Wiley, 4th edition, 2012.

[19] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[20] X. Li and W. Yu. Dynamic system identification via recurrent multilayer perceptrons. *Information Sciences*, 147(1):45–63, 2002.

[21] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, volume 30, pages 6231–6239. Curran Associates, 2017.

[22] V. Maiorov and A. Pinkus. Lower bounds for approximation by mlp neural networks. *Neurocomputing*, 25(1):81–91, 1999.

[23] P. M. Menéndez. *Condiciones de Karush-Kuhn-Tucker*. Universidad Complutense de Madrid, 2024.

[24] P. M. Menéndez. *Teoría de Grafos*. Universidad Complutense de Madrid, 2024.

[25] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A*, 209(441–458), 1909.

[26] A. V. Morales. *Apuntes de GC*. Universidad Complutense de Madrid, 2018.

[27] N. J. Nilsson. *Introduction to Machine Learning.* Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA, November 3 1998. E-mail: nilsson@cs.stanford.edu.

[28] E. Orhan. Cover's Function Counting Theorem, December 16 2014. eorhan@cns.nyu.edu.

[29] F. Ortega, A. González-Prieto, and J. Bobadilla. Providing reliability in recommender systems through bernoulli matrix factorization. *Information Sciences*, 553:110–128, Apr. 2021.

[30] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[31] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[32] J. M. V. Rubio. Origen y desarrollos actuales de la predicción meteorológica. *Encuentros Multidisciplinares*, 2013.

[33] M. Ruiz-Garcia, G. Zhang, S. S. Schoenholz, and A. J. Liu. Tilting the playing field: Dynamical loss functions for machine learning, 2021.

[34] A. M. Schäfer and H. G. Zimmermann. Recurrent neural networks are universal approximators. *International Journal of Neural Systems*, 17(04):253–263, 2007. PMID: 17696290.

[35] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, Cambridge, UK, 2014.

[36] Z. Shen, H. Yang, and S. Zhang. Optimal approximation rate of ReLU networks in terms of width and depth. *Journal de Mathématiques Pures et Appliquées*, 157:101–135, 2022.

[37] N. Srivastava, G. Hinton, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014.

[38] R. C. Staudemeyer and E. R. Morris. Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks. *ArXiv*, abs/1909.09586, 2019.

[39] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 2000.

[40] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[41] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[42] D. Wolpert and W. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.