Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Gradu Amaierako Lana / Trabajo Fin de Grado

Ingeniaritza Elektronikoko Gradua / Grado en Ingeniería Electrónica

Mapas autoorganizados para clasificación en tiempo real

Implementación digital sobre FPGAs

Egilea/Autor/a:
Mikel Díaz Rodríguez
Zuzendaria/Director/a:
Inés del Campo Hagelström

Leioa, 19 de febrero de 2020

Índice general

Índice de figuras

Índice de tablas

Introducción y objetivos

El aprendizaje automático, en ingles *Machine Learning*, es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan y mejoren su desempeño con la experiencia. En concreto, tratan de crear algoritmos que sean capaces de predecir comportamientos fututos y que puedan reconocer patrones a partir de unos datos que se les suministran.

Existen un gran número de problemas que pueden ser resueltos mediante técnicas de aprendizaje automático. Algunas de las técnicas más comunes son: la regresión, el ranking y la clasificación. La regresión trata de predecir un valor real a partir de hechos que hayan sucedido con anterioridad. Por ejemplo, predecir el comportamiento que tendrá la bolsa basándose en el comportamiento que tuvo ayer. El ranking trata de predecir el orden de relevancia de un conjunto de objetos. Por ejemplo, el orden en el que un buscador devuelve los recursos de internet como respuesta a una búsqueda hecha por un usuario. La clasificación trata de predecir la clase de un objeto a partir de un conjunto de etiquetas prefijadas.

Este trabajo se centra en las técnicas de clasificación. Entre las técnicas de clasificación y dependiendo del tipo de entrada que se dispone, los algoritmos se pueden agrupar en aprendizaje supervisado y no supervisado. El aprendizaje supervisado produce una función que establece una correspondencia entre las entradas y las salidas deseadas, donde la base de datos para entrenar el algoritmo está formada por ejemplos etiquetados. En cambio, el entrenamiento del aprendizaje no supervisado se lleva a cabo solo con entradas, es decir, sin conocer a que categorías pertenecen los ejemplos de entrada. Para ello, descubre patrones ocultos o estructuras intrínsecas en los datos de entrada.

Los mapas autoorganizados (SOM) son un tipo de red neuronal artificial (RNA) no supervisada y son especialmente interesantes para la implementación en hardware de la red ya que el algoritmo es completamente paralelizable. Además, permiten obtener una imagen bidimensional de los resultados de la clasificación de donde se pueden extraer las clases. Para aprovechar la característica de que sea paralelizable se pueden utilizar las FPGAs (*Field-Programmable Gate Array*) debido a que su arquitectura resulta muy útil para este fin. De esta manera, se obtendrán unos tiempos de respuesta del orden de los nanosegundos, suficiente para considerarse que la implementación sea en tiempo real para un gran número de campos de aplicación.

Un campo de interés para la aplicación de métodos de clasificación como las redes SOM es el sector del automóvil. En la actualidad, los automóviles llevan numerosos sistemas de ayuda a la conducción (DAS: Driver Assistance System). Además, llevan complejos sistemas integrados en el diseño de los componentes del automóvil para mejorar la eficiencia energética, pero la mayoría no tienen en cuenta la forma en la que los automóviles son conducidos. Dependiendo del estilo de conducción, el consumo de combustible se ve afectado llegando a ser un factor determinante. El gas emitido por los automóviles está ligado al consumo de

combustible. Por lo tanto, es un tema relevante buscar soluciones para minimizar el impacto que tienen los conductores en el incremento de la polución.

Objetivos del trabajo

El objetivo principal del trabajo es estudiar y analizar los diferentes algoritmos de aprendizaje automático para clasificación no supervisada más utilizados en la actualidad. El interés de estos algoritmos radica en la capacidad que tienen en descubrir patrones ocultos o estructuras intrínsecas en los datos. En concreto se estudian en mayor profundidad los mapas autoorganizados (SOM) ya que ofrecen un compromiso muy bueno entre velocidad e interpretabilidad. Además, permiten que la evaluación de la red se pueda realizar utilizando propiedades intrínsecas del algoritmo capaces de realizarse en paralelo. Por ello, se propone una arquitectura para la implementación en hardware de la evaluación del SOM diseñada para que los procesos se calculen en paralelo.

Para demostrar que se pueden implantar sistemas de clasificación en tiempo real, se realiza un análisis sobre cómo afecta el estilo de conducción al consumo de combustible. Se ha escogido este tema debido a la necesidad de reducir la contaminación producida por los automóviles. Para caracterizar los estilos de conducción que afectan a las emisiones de gases de efecto invernadero y, por ello, al consumo de combustible, se utilizan las técnicas de conducción eficiente (eco-driving) como base para la obtención de los parámetros necesarios para la aplicación de los algoritmos de clasificación. De esta manera, se puede conseguir evaluar el comportamiento del conductor en tiempo real y así poder avisarle para que mejore la conducción.

Capítulo 1

Técnicas de clasificación

Las técnicas de clasificación son algoritmos de aprendizaje automático que son capaces de aprender sin necesidad de tener que suministrar una ecuación predeterminada como modelo, es decir, aprenden directamente de los datos. A medida que se va aumentando el número de muestras disponibles para el aprendizaje, se mejora el rendimiento de los algoritmos, ya que se estarán creando modelos más ajustados y precisos. Se van a analizar las diferentes técnicas con el fin de conocer las características más relevantes de cada una de ellas y conocer qué ventajas y desventajas tiene la elección de una técnica respecto a las otras. Además, se expondrán los algoritmos de cada una de las técnicas y los métodos para su evaluación.

Para visualizar y posteriormente comparar los resultados de las distintas técnicas, se utiliza un ejemplo común para todas. Los datos se obtienen de la base de datos llamada "*Iris de Fisher*". Este conjunto de datos fue creado por Ronald Fisher para su artículo de 1936, *The use of multiple measurements in taxonomic problems (*El uso de medidas múltiples en problemas taxonómicos) [1]. El conjunto de datos contiene 50 muestras de cada una de tres especies de Iris (Iris Setosa, Iris Virginica e Iris Versicolor). Para realizar más fácil las comparaciones entre las distintas técnicas, se denota por colores las tres especies de Iris: Setosa (azul), Virginica (verde) y Versicolor (rojo). Para cada una de las muestras, se midieron cuatro rasgos: el largo y el ancho del sépalo y pétalo. Cada uno de estos rasgos, es un atributo del conjunto de datos. Para la validación de las técnicas de clasificación se divide la base de datos en muestras de entrenamiento y de test.

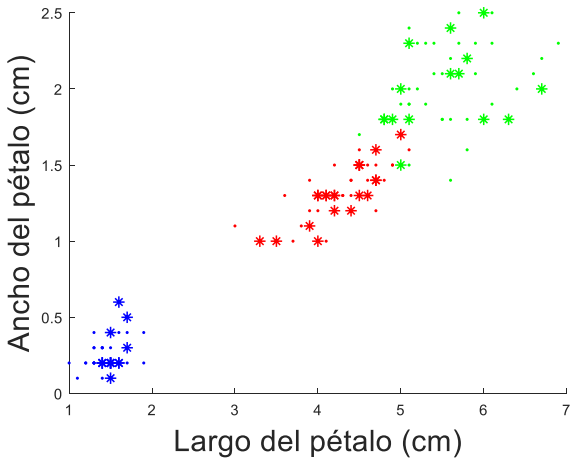

Figura 1.1: Representación del ancho de los pétalos en función del largo de los pétalos. Los puntos (·) indican las muestras de entrenamiento y los asteriscos (*) las muestras de test.

1.1. Aprendizaje supervisado

Las técnicas de aprendizaje supervisado predicen las salidas a partir de un modelo obtenido con muestras de entrada y las etiquetas (clases) a las que pertenecen dichas muestras. En ocasiones, no se conocen las posibles clases de salida a las que pueden pertenecer las muestras de entrada. Para ello se recurre a las técnicas de aprendizaje no supervisado ya que son capaces de descubrir patrones en los datos de entrada. A pesar de ello, se explica una técnica de aprendizaje supervisado como referencia de los resultados que se pueden obtener con un modelo de clasificación y, de esta manera, sirva para comparar los resultados con los algoritmos no supervisados.

En concreto se exponen las redes neuronales artificiales supervisadas puesto que son las técnicas más avanzadas de clasificación y son las más utilizas en la actualidad. Además, se ha escogido está técnica como introducción a los mapas autoorganizados (SOM) que se utilizarán posteriormente. A pesar de que las redes neuronales artificiales (RNA) son las más utilizadas, existen otras técnicas de clasificación supervisada, como por ejemplo los árboles de decisión o la clasificación de Naïve Bayes.

1.1.1. Redes neuronales supervisadas

Las redes neuronales son un modelo computacional basado en la estructura interconectada de las neuronas en el cerebro. Su estructura se divide en capas conectadas entre sí. La información de entrada atraviesa la red neuronal hasta producir los valores de salida. Se puede entrenar una red neuronal para que reconozca patrones, clasifique datos o pronostique sucesos futuros.

Las redes neuronales artificiales (RNA) surgieron como un intento de descubrir la arquitectura del cerebro humano para realizar tareas en las que los algoritmos convencionales habían tenido poco éxito. Normalmente los modelos están asociados con un algoritmo de entrenamiento o una regla de aprendizaje.

Las RNA están formadas por neuronas (nodos) conectadas entre sí formando capas. Las redes constan de una capa de entrada, una o varias capas ocultas y una capa de salida. Las redes neuronales que constan de muchas capas se denominan redes profundas o *deep neural networks*. Estas redes son útiles para aplicaciones de identificación compleja, como pueden ser la traducción de textos o el reconocimiento de imágenes, entre otros [2]. Existen diferentes topologías de redes neuronales que determinan como están interconectadas las neuronas en las capas ocultas. Se pueden clasificar las RNA según el tipo de conexiones o el tipo de aprendizaje:

Según el tipo de conexiones:

- **Redes de propagación hacia delante**, tienen como objetivo aproximar una función determinada. La característica principal es que las conexiones entre neuronas son siempre hacia adelante, es decir, en un solo sentido desde la capa de entrada hacia la capa de salida.
- **Redes recurrentes**, permite conexiones hacia atrás, es decir, las conexiones pueden realimentarse.

El primer modelo matemático de una red neuronal artificial fue descrito por Walter Pitts, junto con Warren McCulloch, en un artículo titulado "Cálculo lógico de ideas inherentes en la actividad nerviosa" [3] en 1943. En él explican la arquitectura de una neurona sencilla y es todavía el estándar de referencia en el campo de las redes neuronales.

La neurona de McCulloch-Pitts (ver Figura 1.2) es la unidad esencial con la que se construye una RNA. El modelo se calcula realizando una suma ponderada de las entradas, seguida de la aplicación de una función no lineal llamada, función de activación f. Esta función se elige de acuerdo a la tarea realizada por la neurona. Entre las más comunes dentro del campo de las RNA destacan la función identidad, escalón, gaussiana y sinusoidal. El primer modelo que propusieron McCulloch-Pitts fue utilizando la función escalón. Con ello conseguían una salida binaria, 0 ó 1.

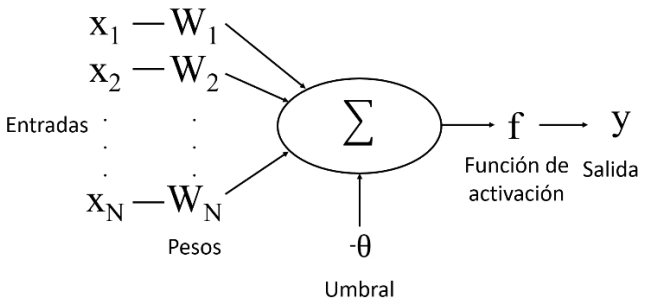

Figura 1.2: Modelo computacional de una neurona.

La salida de una neurona está definida como,

$$y = f\left(\sum_{i=1}^{N} W_i x_i - \theta\right) \tag{1.1}$$

donde W_i representa la intensidad de interacción entre la neurona y las entradas, N es el número de características (atributos) que tienen las muestras de entrada y f es la función de activación. El parámetro W_i se conoce como peso y se va modificando en el llamado proceso de aprendizaje. θ es el valor umbral o sesgo que, en este caso, se puede considerar como un umbral de actuación de la neurona.

Inspirado en el modelo de neurona anterior, Rosenblatt creó el Perceptrón simple (ver Figura 1.3) en 1962 [4]. Este modelo unidireccional está compuesto por dos capas de neuronas, una de entrada y otra de salida, y se puede utilizar como clasificador ya que el algoritmo de entrenamiento puede determinar automáticamente los pesos que clasifican un conjunto de patrones a partir de un conjunto de ejemplos etiquetados.

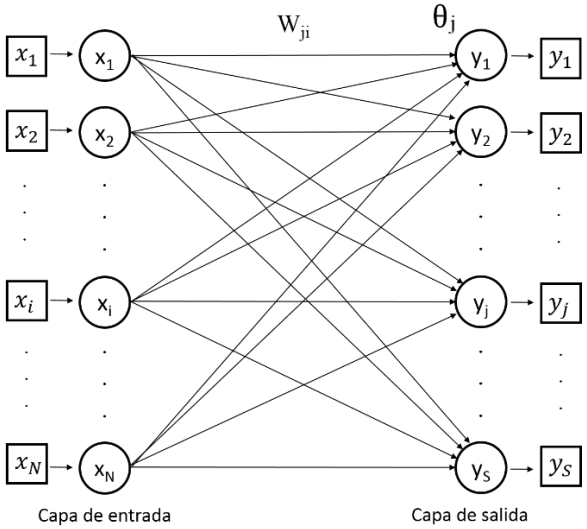

Figura 1.3: Arquitectura de un Perceptrón simple.

Si la red neuronal está constituida por un número j de neuronas de salida, la salida neuronal de la j-ésima neurona y_j se define como,

$$y_j = f\left(\sum_{i=1}^{N} W_{ji}x_i - \theta_j\right) \tag{1.2}$$

El problema que tenía este modelo era que solo podía diferenciar dos regiones separadas por un hiperplano (una recta en el caso de dos neuronas de entrada). Para solventar estas limitaciones se propuso el Perceptrón multicapa, que añadía capas ocultas.

Funahaski demostró en 1989 que con solo una capa oculta se puede aproximar cualquier función continua en un intervalo hasta el nivel deseado [5]. Este hecho proporciona una base sólida al campo de las RNA, aunque en la demostración no se precisa el número de nodos ocultos necesarios para llevar a cabo la aproximación. Además, para el Perceptrón multicapa, el umbral θ de una neurona se trata como una conexión más a la neurona.

El Perceptrón multicapa se puede generalizar para un número de capas ocultas mayor. Sea un Perceptrón multicapa con L capas ($L-2$ capas ocultas) y P_l neuronas en la capa l, para $l = 1, 2, \dots, L$. Sea W_{ji}^l la matriz de pesos que representa el peso de las conexiones de las neuronas de la capa $l-1$ con la neurona j de la capa l para $l = 2, \dots, L$. Se denota a_j^l a la activación (salida) de la neurona j de la capa l. Estas activaciones se calculan del siguiente modo:

- **Activación de las neuronas de la capa de entrada**

$$a_j^1 = x_i \tag{1.3}$$

siendo x_i un atributo $i = 1, 2, \dots, N$.

- **Activación de las neuronas de la capa oculta.**

$$a_j^l = f\left(\sum_{i=1}^{P_{l-1}} W_{ji}^l a_i^{(l-1)} - \theta_j^l\right) \tag{1.4}$$

donde $a_i^{(l-1)}$ son las activaciones de las neuronas de la capa $l-1$.

- **Activación de las neuronas de la capa de salida.**

$$a_j^L = y_j = f\left(\sum_{i=1}^{P_{L-1}} W_{ji}^L a_i^{(L-1)} - \theta_j^L\right) \tag{1.5}$$

siendo y_j la salida de una neurona $j = 1, 2, \dots, S$.

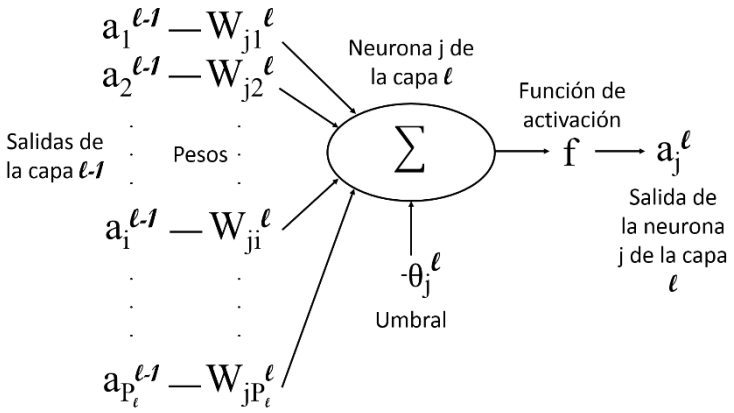

Figura 1.4: Modelo computacional de una neurona *j* de la capa *l*.

Las funciones de activación f, para el Perceptrón multicapa, más utilizadas son la función sigmoidal y la función tangente hiperbólica.

$$Función\ sigmoidal\ \rightarrow\ f_{sigm}(x) = \frac{1}{1+e^{-x}} \tag{1.6}$$

$$Función\ tangente\ hiperbólica\ \rightarrow\ f_{thip}(x) = \frac{1-e^{-x}}{1+e^{-x}} \tag{1.7}$$

Ambas funciones son crecientes con dos niveles de saturación y se relacionan mediante la expresión:

$$f_{thip}(x) = 2f_{sigm}(x) - 1 \tag{1.8}$$

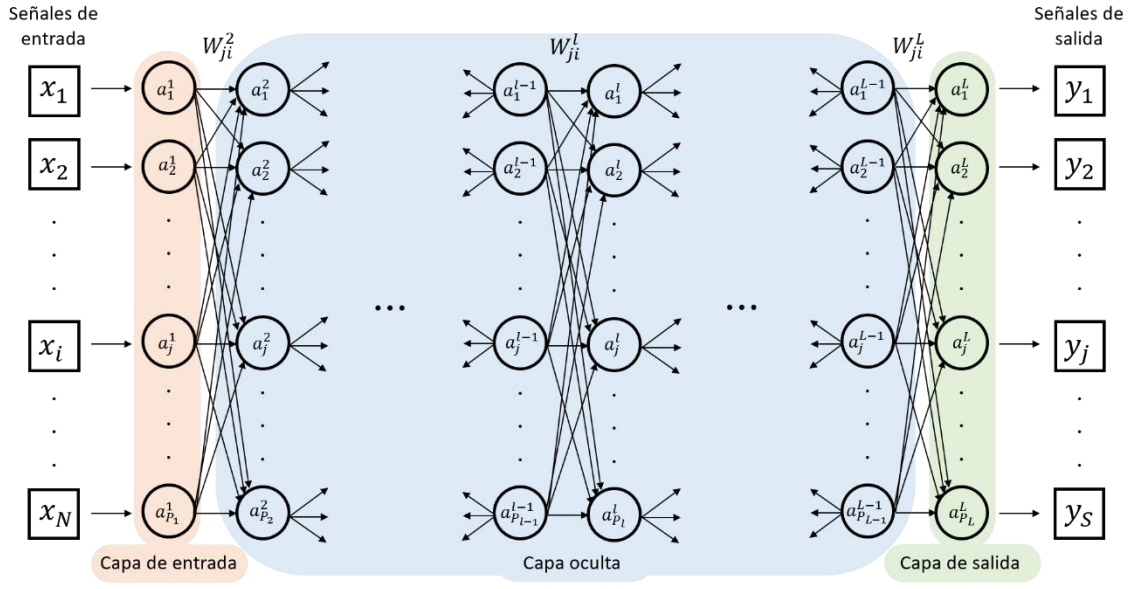

Figura 1.5: Arquitectura general de una RNA

Como se ve en la Figura 1.5, la arquitectura general de una RNA puede estar compuesta por más de una capa oculta. Hay que destacar que en esta arquitectura no se ha considerado las conexiones hacia atrás.

Para modelizar las RNA, es necesaria una fase de entrenamiento, o de aprendizaje, y otra fase de pruebas.

i) Fase de entrenamiento

La fase de entrenamiento tiene como objetivo que la salida de la RNA sea lo más parecida posible a la salida deseada. Por ello, el entrenamiento de la red se puede formular como un problema de minimización donde una función calcule la diferencia entre las salidas de la red y las salidas deseadas. El Perceptrón multicapa se suele entrenar por medio de un algoritmo de retropropagación de errores. El algoritmo fue introducido por primera vez en 1974 por Werboz pero no tuvo repercusión hasta que en 1986 Rumelhart y su equipo lo redescubrieran de manera independiente y comenzaran a popularizarlo ayudados por los avances en computación existentes en la época, los cuales les permitieron satisfacer los requisitos de computación mínimos para poder ejecutar el algoritmo [6] [7].

ii) Fase de pruebas

En la fase anterior, el modelo puede que reproduzca adecuadamente el comportamiento de las muestras de entrenamiento, pero pierda su habilidad de generalizar su aprendizaje para casos nuevos. Este hecho puede deberse al ajustar excesivamente el modelo a las particularidades presentes en las muestras de entrenamiento. A este problema se le suele denominar sobreajuste y puede acentuarse en el caso de que los datos tengan ruido o errores.

Para controlar el proceso de aprendizaje, es aconsejable utilizar un segundo grupo de muestras diferentes a las que se utilizaron en el proceso de entrenamiento. De esta manera, se evita el problema del sobreajuste.

Base de datos *Iris*

Para poner en práctica lo explicado anteriormente, se ha utilizado la base de datos *Iris* y se ha realizado una clasificación utilizando las redes neuronales artificiales (RNA). Aleatoriamente, se han apartado un tercio de las muestras de entrenamiento para la posterior validación de la red. Para la realización del problema se ha buscado una arquitectura capaz de clasificar la base de datos *Iris* utilizando el menor número de recursos posibles. Con ello, se ha comprobado que utilizando una única capa oculta se obtienen unos buenos resultados para este ejemplo. Además, se ha experimentado con diferentes números de neuronas en la capa oculta hasta encontrar un equilibrio entre tiempos de ejecución y resultados que sean suficientemente buenos, pero sin llegar a que se produzca un sobreajuste. Por estos motivos se ha utilizado una RNA que consta de cinco neuronas en la capa oculta y se ha implementado utilizando el *toolbox Machine Learning* de MATLAB. El resultado que se obtiene es una matriz que muestra cual es la neurona de salida que se ha activado para cada muestra de entrenamiento.

En primer lugar, se dividen las muestras de entrada en dos grupos: las muestras de entrenamiento y las muestras de test. Para el caso del *Iris*, a partir de las 150 muestras que contiene la base de datos, se han tomado 100 para el entrenamiento y 50 para los test. Una vez entrenada la red, se ha validado con las muestras de test. En la Figura 1.6 se ha representado el ancho de los pétalos en función del largo de los pétalos.

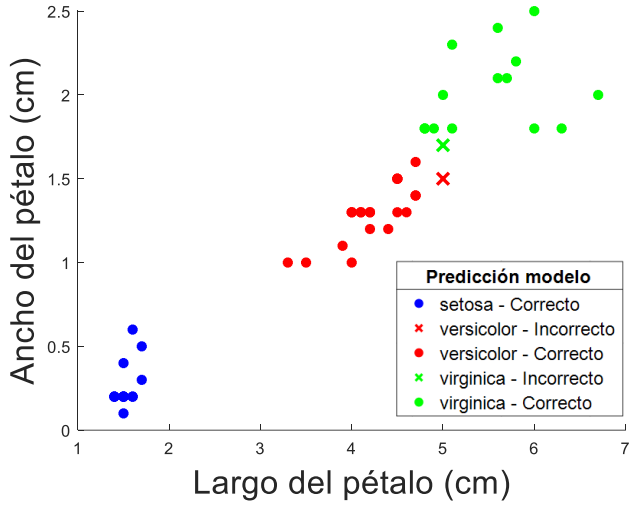

Figura 1.6: Representación del ancho de los pétalos en función del largo de los pétalos utilizando redes neuronales artificiales (RNA). Los puntos marcados con 'x' corresponden a los datos que no están correctamente predichos.

Como se observa en la Figura 1.6, el entrenamiento de la red ha sido muy bueno ya que prácticamente todas las muestras han sido clasificadas correctamente. En este tipo de redes supervisadas se suele utilizar un diagrama de confusión que muestra los porcentajes de clasificación correctos e incorrectos (Figura 1.7).

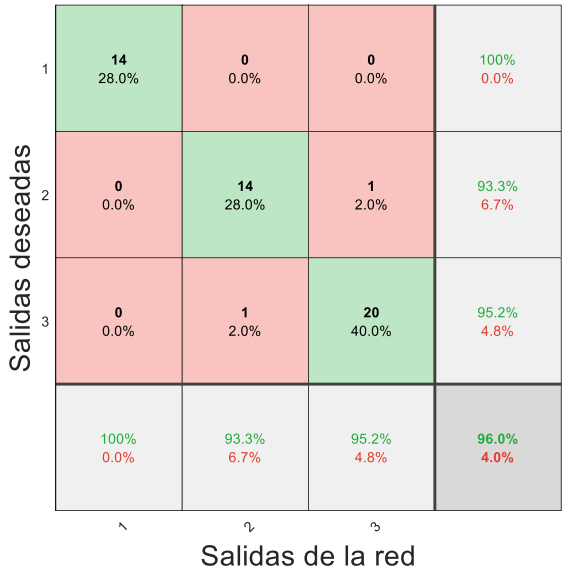

Figura 1.7: Diagrama de confusión. Los cuadrados verdes muestran las clasificaciones correctas y los cuadrados rojos las incorrectas.

El diagrama de confusión indica que el 96% de las muestras de test se han clasificado correctamente. Las dos muestras que no se han clasificado correctamente están entre las especies Virginica (verdes) y las Versicolor (rojas), ya que es una zona que no se puede diferenciar claramente entre los dos tipos de Iris.

1.2. Aprendizaje no supervisado

Los métodos de clustering, o de agrupamiento, a diferencia de los métodos de aprendizaje supervisado, no requieren que se les proporcione un conjunto de muestras de entradas y de salidas, basta con facilitarles un grupo de datos de entrada sin respuestas etiquetadas [8]. A partir de un conjunto de datos de entrada, son capaces de hallar patrones ocultos o estructuras intrínsecas en los datos. Dentro de este grupo, existen diferentes técnicas de agrupamiento que se irán presentando y comparando: agrupación jerárquica, k-means y modelo de mezclas gaussianas.

1.2.1. Agrupación jerárquica

La agrupación jerárquica, o también llamado *Clustering jerárquico*, es un método de análisis que se basa en agrupar objetos similares de manera jerárquica creando un árbol de agrupación o *dendrograma*. El árbol no es un conjunto único de grupos, sino más bien una jerarquía multinivel, donde los grupos en un nivel se unen como grupos en el siguiente nivel. Esto permite decidir el nivel o escala de agrupamiento que sea más apropiado para cada aplicación. Además, a diferencia de las RNA y de los métodos que se irán presentando en este capítulo, este método no depende de las condiciones iniciales ya que las muestras no se entrenan de forma aleatoria, sino siguiendo un orden establecido de forma intrínseca por la arquitectura del algoritmo. Por este motivo, siempre que se entrene el algoritmo con las mismas muestras, el resultado siempre será el mismo. El agrupamiento jerárquico fue introducido por primera vez por Lance y Williams en 1967 [9].

Los métodos jerárquicos se pueden dividir en dos categorías: aglomerativos y disociativos. Cada una de estas categorías presenta diversas variantes. Los métodos aglomerativos, o ascendentes, comienzan el entrenamiento con un número de grupos igual al número de muestras de entrada y se van agrupando de forma ascendente hasta llegar al mínimo de agrupaciones posibles. En cambio, los métodos disociativos, o descendentes, hacen el proceso inverso, comienzan con una agrupación que contiene todas las muestras y van, de forma descendente, formando cada vez más grupos hasta llegar a tener un grupo por cada muestra. A continuación, se describen los pasos del algoritmo aglomerativo básico.

i) Buscar la similitud entre cada par de muestras en el conjunto de datos.

Se considera un conjunto de muestras de entrada para el entrenamiento X^E,

$$X^E = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_i^1 & \cdots & x_N^1 \\ x_1^2 & & & & & \vdots \\ \vdots & & & & & \vdots \\ x_1^e & & & & & \vdots \\ \vdots & & & & & \vdots \\ x_1^E & \cdots & & \cdots & & x_N^E \end{pmatrix} = \begin{pmatrix} \overrightarrow{x^1} \\ \overrightarrow{x^2} \\ \vdots \\ \overrightarrow{x^e} \\ \vdots \\ \overrightarrow{x^E} \end{pmatrix} \equiv \begin{pmatrix} \boldsymbol{x^1} \\ \boldsymbol{x^2} \\ \vdots \\ \boldsymbol{x^e} \\ \vdots \\ \boldsymbol{x^E} \end{pmatrix}_{(NXE)} \tag{1.9}$$

compuesto por un número E de muestras de entrada para el entrenamiento y de N características.

Para cada muestra de entrenamiento se debe calcular la distancia con todas las muestras restantes. Hay muchas formas de calcular esta información de distancia, pero la más común es calculando la distancia euclídea entre objetos,

$$d^2(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{N} (a_i - b_i)^2 \tag{1.10}$$

siendo $\boldsymbol{a}, \boldsymbol{b}$ dos muestras del conjunto de entrenamiento X^E.

El resultado de las medidas de similitud se recoge en una matriz simétrica llamada matriz de distancia, o de proximidad.

$$MD = \begin{pmatrix} d^2(\boldsymbol{x^1}, \boldsymbol{x^1}) & \cdots & d^2(\boldsymbol{x^1}, \boldsymbol{x^E}) \\ \vdots & \ddots & \vdots \\ d^2(\boldsymbol{x^E}, \boldsymbol{x^1}) & \cdots & d^2(\boldsymbol{x^E}, \boldsymbol{x^E}) \end{pmatrix} \tag{1.11}$$

siendo $\boldsymbol{x^e} \subset X^E, 1 \leq e \leq E$.

La diagonal de la matriz de distancia es cero ya que se está calculando la distancia entre una muestra y ella misma. Además, $d(\boldsymbol{x^1}, \boldsymbol{x^2}) = d(\boldsymbol{x^2}, \boldsymbol{x^1})$ por lo que solo se tiene que calcular la mitad de los elementos de la matriz. Por ejemplo, si se tienen 5 muestras de entrada, la matriz de distancia es:

$$MD = \begin{pmatrix} 0 & - & - & - & - \\ d^2(\boldsymbol{x}^2, \boldsymbol{x}^1) & 0 & - & - & - \\ d^2(\boldsymbol{x}^3, \boldsymbol{x}^1) & d^2(\boldsymbol{x}^3, \boldsymbol{x}^2) & 0 & - & - \\ d^2(\boldsymbol{x}^4, \boldsymbol{x}^1) & d^2(\boldsymbol{x}^4, \boldsymbol{x}^2) & d^2(\boldsymbol{x}^4, \boldsymbol{x}^3) & 0 & - \\ d^2(\boldsymbol{x}^5, \boldsymbol{x}^1) & d^2(\boldsymbol{x}^5, \boldsymbol{x}^2) & d^2(\boldsymbol{x}^5, \boldsymbol{x}^3) & d^2(\boldsymbol{x}^5, \boldsymbol{x}^4) & 0 \end{pmatrix} \tag{1.12}$$

donde – representa las distancias que no hacen falta calcular ya que la matriz es simétrica.

ii) Unir las dos agrupaciones más cercanas.

En este paso, se vinculan los pares de muestras más cercanos. A medida que las muestras se emparejan, los grupos recién formados se agrupan en grupos más grandes hasta que se forma un árbol jerárquico.

$$\min_{\boldsymbol{a}, \boldsymbol{b} \in X^E} \{d^2(\boldsymbol{a}, \boldsymbol{b})\} \tag{1.13}$$

siendo $\boldsymbol{a}, \boldsymbol{b}$ dos muestras del conjunto de entrenamiento X^E.

En la Figura 1.8 se observa cómo se van agrupando las muestras.

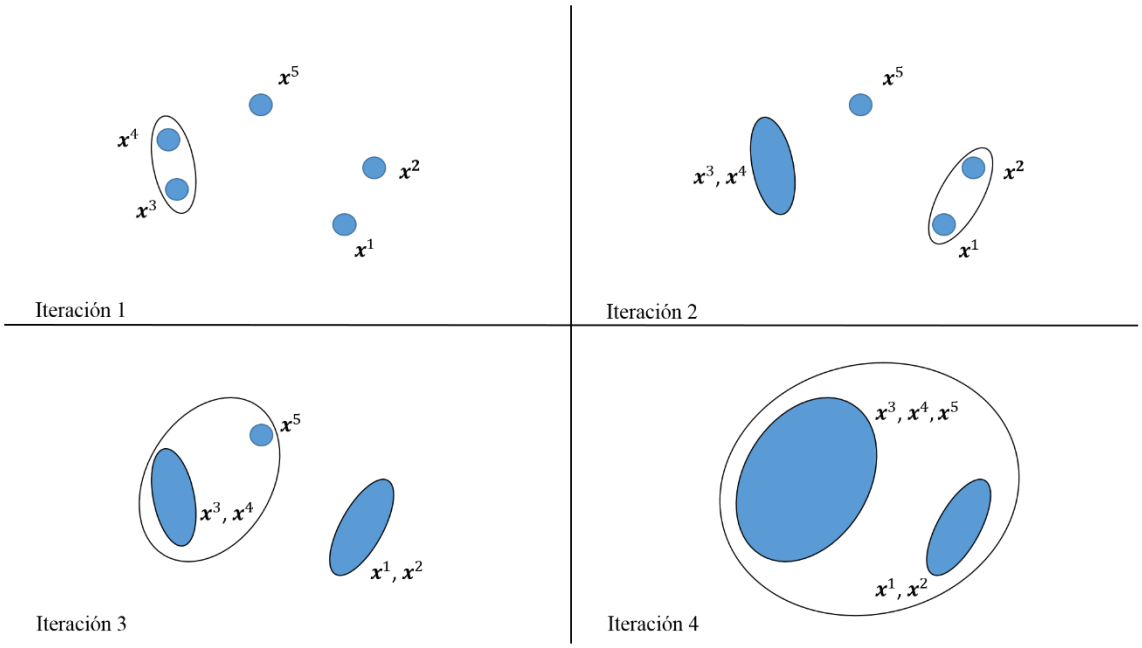

Figura 1.8: Unión de las agrupaciones más cercanas para cada iteración del algoritmo.

iii) Determinar dónde cortar el árbol jerárquico.

Se pueden dividir naturalmente las muestras en grupos que estén lo más alejados posible entre ellos. En el método de agrupación jerárquica se pueden observar donde están las muestras lo suficientemente agrupadas y en donde no, con los denominados d*endrograma* (Figura 1.9).

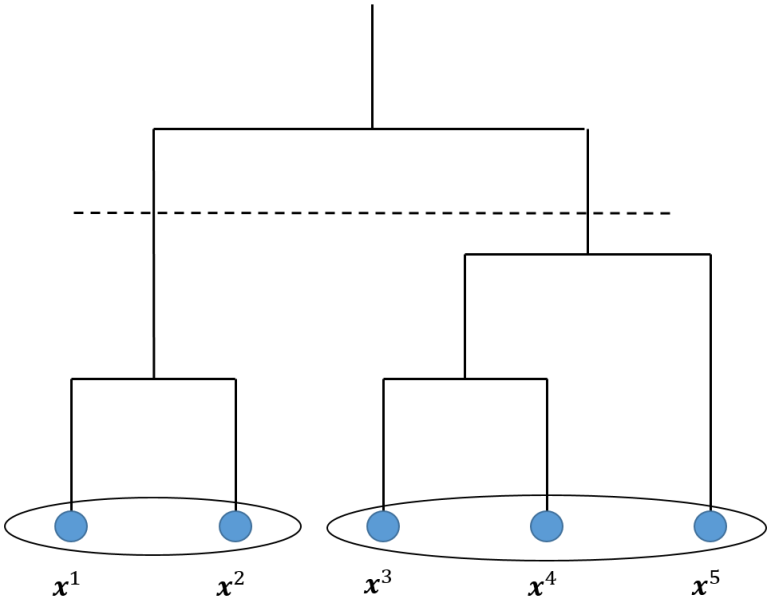

Figura 1.9: Dendrograma.

Un dendrograma es un tipo de representación gráfica o diagrama de datos en forma de árbol que organiza los datos en subcategorías que se van dividiendo. Este tipo de representación permite apreciar claramente las relaciones de agrupación entre los datos e incluso entre grupos de ellos, aunque no las relaciones de similitud o cercanía entre categorías. Cuanta más altura tenga el dendrograma, más distancia habrá entre las muestras y, por lo tanto, mejor separas estarán. Dependiendo a que altura se corte el diagrama, se obtiene el número de

agrupaciones. Por ejemplo, en la Figura 1.9, se tienen 5 muestras con las que se han producido 2 agrupaciones al haber cortado el diagrama a una cierta altura.

iv) Validación o clasificación de nuevos datos.

Una vez clasificadas las muestras de entrenamiento en agrupaciones, con este método no se puede validar con muestras de test o con nuevas muestras que se quiera clasificar. Para solventar esta limitación se pueden utilizar las redes neuronales artificiales (RNA) entrenándolas con las muestras de entrenamiento y con las etiquetas obtenidas mediante el método de agrupación jerárquica.

Utilizando la base de datos *Iris*, se realiza el proceso de agrupamiento utilizando el modelo de agrupación jerárquica. Cabe destacar que los datos de cada flor que se están utilizando son cuatro (ancho y largo del sépalo y del pétalo, por lo tanto, de dimensión cuatro), pero con el fin de representarlo se ha utilizado el ancho y el largo del pétalo. En primer lugar, se clasifican las 100 muestras de entrenamiento en agrupaciones mediante el método de agrupación jerárquica. A continuación, se entrena una RNA utilizando como entradas las muestras de entrenamiento y las agrupaciones a las que pertenecen obtenidas con el método de agrupación jerárquica. Por último, se válida la red haciendo uso de las 50 muestras de test. La RNA que se ha utilizado consta de una única capa oculta compuesta por 5 neuronas, ya que en el capítulo anterior se demostró que para entrenar la base de datos *Iris* era la arquitectura más apropiada.

Antes de realizar la validación, se representa en la Figura 1.10 un dendrograma que indica cómo se han agrupado las muestras de entrenamiento.

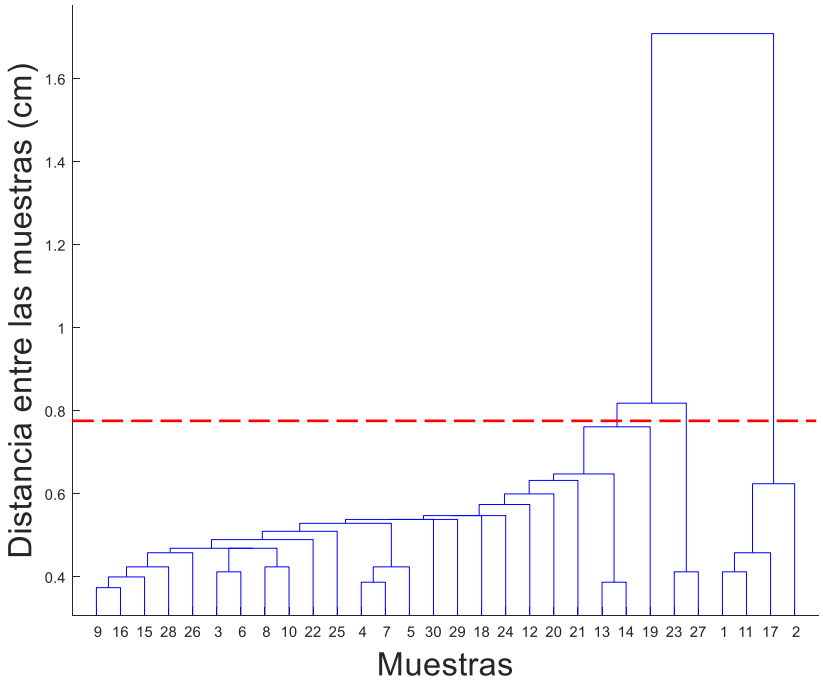

Figura 1.10: Dendrograma para la base de datos *Iris*. La línea discontinua roja indica a que distancia se ha cortado el dendrograma.

Como se observa en la Figura 1.10, hay dos agrupaciones que están muy bien separadas y a partir de ahí las distancias entre las muestras se reduce drásticamente. Ya que se conoce que la clasificación de las flores la forman tres tipos de Iris, se corta el dendrograma a una

altura para la cual se obtienen tres agrupaciones. En la Figura 1.11 se representa la clasificación de las muestras de test obtenidas con la RNA.

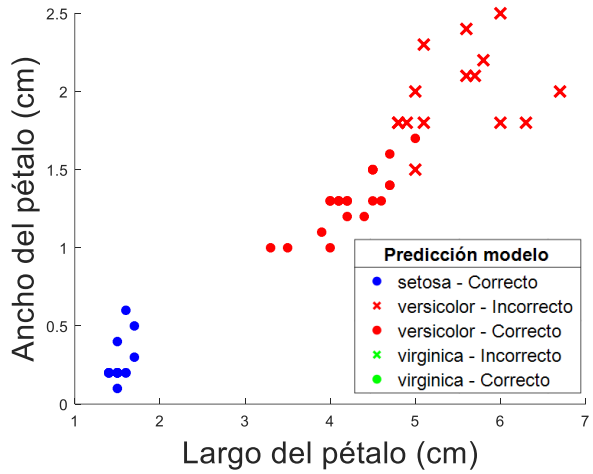

Figura 1.11: Representación del ancho de los pétalos en función del largo de los pétalos utilizando el método de agrupación jerárquica.

Como se puede ver en la Figura 1.11, a pesar de que las muestras de entrenamiento estén agrupadas en tres grupos, la RNA solo ha agrupado las muestras de test en dos agrupaciones. Por lo tanto, utilizando las RNA como evaluación del agrupamiento jerárquico y para el ejemplo de la base de datos *Iris*, el algoritmo solo ha sido capaz de diferenciar entre dos tipos de agrupaciones. Al igual que pasaba en el apartado anterior con las RNA, la mayor dificultad en esta base de datos es diferenciar las V*ersicolor* de las V*irginicas*. En este caso se agrava este problema ya que cuando se generan las agrupaciones jerárquicas solo se tienen en cuanta los dos conjuntos de datos más cercanos. Por lo tanto, el método de agrupación jerárquica es un método sencillo para la clasificación, pero para agrupaciones que están próximas no es un método adecuado debido a que el método de agrupación jerárquica solo tiene en cuenta para hacer la clasificación en cada iteración las dos muestras más cercanas, es decir, no tiene en cuenta a todo el conjunto de muestras.

Para tener una medida que se pueda comparar con otros métodos, se define la *tasa de acierto* como,

$$TA(\%) = \frac{muestras\ de\ test\ bien\ clasificadas}{muestras\ totales\ de\ test} \cdot 100 \qquad (1.14)$$

Este cálculo se puede hacer para esta base de datos ya se conoce a que especies pertenecen las flores. En una situación real, este cálculo no se podría hacer ya que el fin de utilizar el aprendizaje no supervisado es porque no se conocen las agrupaciones a las que pertenecen las muestras. Utilizando este método se obtiene una tasa de acierto del 70%. Este resultado se comparará con otros métodos de aprendizaje no supervisado para demostrar cual es el método que mejor clasifica la base de datos *Iris*.

1.2.2. K-means

El algoritmo estándar fue propuesto por primera vez por S. Lloyd en 1957, aunque no se publicó hasta 1982 [10]. En 1965, E. W. Forgy publicó esencialmente el mismo método por lo que el método k-means también se conoce como método de Lloyd-Forgy [11].

El método de k-means, divide los datos en k agrupaciones distintas en función de la distancia al centroide de una agrupación. El centroide para cada grupo es el punto en el que se minimiza la suma de distancias de todos los objetos en ese grupo. A diferencia de la agrupación jerárquica, la agrupación de k-means utiliza directamente las muestras de entrada, en lugar de utilizar conjuntos más amplios de medidas (medidas de diferencia entre dos muestras), y crea un solo nivel de agrupaciones.

K-means trata cada observación de sus datos como un objeto que tiene una ubicación en el espacio y busca una partición en la que los objetos dentro de cada grupo están lo más cerca posible entre sí y lo más lejos posible de los objetos de otros grupos. Se pueden elegir distintas métricas de distancia, según el tipo de muestras que se esté agrupando. El algoritmo itera entre dos pasos: asignación de los datos y actualización de los centroides. Antes de iniciarlo, se asignan aleatoriamente los centroides, lo que provoca que cada vez que se ejecute el algoritmo de entrenamiento los resultados puedan variar. Este hecho hace que las semillas iniciales tengan un fuerte impacto en los resultados finales.

i) Asignación de los datos.

Los centroides definen cada uno de los grupos. En este paso, cada muestra se asigna al centroide más cercano, en función de la distancia métrica que se haya utilizado. La más común es la distancia euclídea al cuadrado. Sea $C = \boldsymbol{c_1}, \dots, \boldsymbol{c_k}, \dots, \boldsymbol{c_K}$ el conjunto de centroides, entonces cada muestra $\boldsymbol{x^e}$ se asigna a un grupo basado en,

$$\min_{c} \sum_{k=1}^{K} \sum_{e=1}^{E_k} \|\boldsymbol{\mu_k} - \boldsymbol{x^e}\|^2 \tag{1.15}$$

donde $\boldsymbol{\mu_k}$ es la media de las muestras en $\boldsymbol{c_k}$.

ii) Actualización de los centroides.

En este paso, los centroides se vuelven a calcular tomando la media de todos los puntos de datos asignados al grupo de ese centroide,

$$\boldsymbol{c_k} = \frac{1}{E_k} \sum_{e=1}^{E_k} \boldsymbol{x_k^e} \tag{1.16}$$

siendo $\boldsymbol{x_k^e}$ las muestras que pertenecen al grupo $\boldsymbol{c_k}$ y E_k el número de muestras que pertenecen al grupo $\boldsymbol{c_k}$.

El algoritmo itera entre estos dos pasos hasta que se cumpla un criterio de detención, es decir, hasta que ningún dato cambie de centroide o se alcance un número máximo de iteraciones.

Uno de los problemas de este método es determinar el número correcto de grupos. Para poder determinar el número óptimo, hay que ir experimentando hasta encontrar la mejor agrupación de los datos. Otro problema que se debe evitar son los mínimos locales, ya que a menudo, la solución depende de los puntos de partida. Para evitar esto, se debe replicar varias veces el algoritmo y quedarse con la solución que tiene la suma de distancias total más baja.

Base de datos *Iris*

A diferencia del método de agrupación jerárquica, con k-means se puede validar el entrenamiento con muestras de test o con nuevas muestras que se quieran clasificar. Para ello, basta con calcular para cada nueva muestra cual es el centroide más cercano ya que, como se ha explicado anteriormente, cada centroide representa una agrupación. Utilizando el ejemplo de la base de datos *Iris*, se realiza el proceso de agrupamiento utilizando k-means.

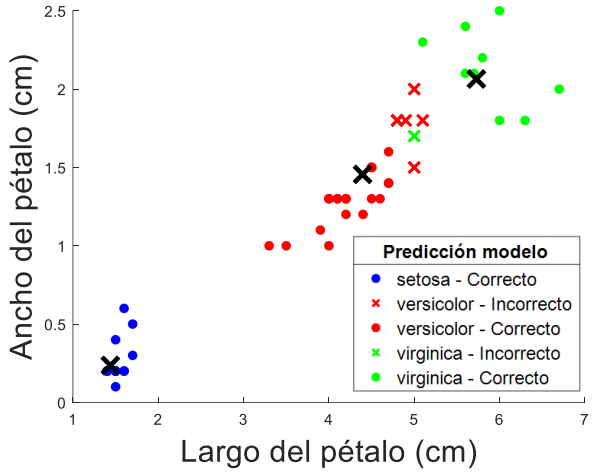

Figura 1.12: Representación del ancho de los pétalos en función del largo de los pétalos utilizando el método k-means. La 'x' en negro marca el centro de las agrupaciones (centroide).

Como se observa en la Figura 1.12, para este ejemplo, el método utilizado se ajusta mejor a los valores reales, con una tasa de acierto del 86%. Este hecho se produce debido a que no tiene en cuenta la separación entre grupos de datos, sino la separación entre sus centros y las muestras de entrenamiento. Además, el algoritmo busca que los centroides tengan el mayor número de muestras. Si para validar los resultados se utiliza una RNA como en el caso de la agrupación jerárquica, se obtiene una tasa de acierto idéntica, 86%. Por lo tanto, para este método, no es necesario utilizar una RNA para validar ya que no aporta ninguna ventaja.

En un caso real, se utiliza el aprendizaje no supervisado cuando no se conocen a que agrupaciones pertenecen las muestras de entrenamiento. Por ello, no se puede conocer si las muestras se han clasificado correctamente. Para solventar esta limitación, Peter J. Rousseeuw introdujo el trazado de silueta como ayuda gráfica para la interpretación y validación del análisis de agrupaciones [12].

El análisis de silueta se puede utilizar para estudiar la distancia de separación entre los grupos resultantes. El trazado de silueta muestra una medida de cuánto de cerca está cada muestra de un grupo a las muestras en los grupos vecinos y, por lo tanto, proporciona una forma de evaluar visualmente parámetros como el número de grupos. Esta medida se define en el rango [-1, 1].

Para determinar si están suficientemente separadas las agrupaciones, el trazado de silueta, muestra, con un valor entre 0 y 1, la cercanía de cada muestra de una agrupación con las muestras de las agrupaciones vecinas. Cuanto mayor sea este número, más distantes estarán las agrupaciones vecinas. Si devuelve un número negativo entre -1 y 0, significa que probablemente las muestras que se han asignado a la agrupación sean incorrectas.

Para cada muestra $\boldsymbol{x_k^e}$ perteneciente al grupo $\boldsymbol{c_k}$ se puede definir $a(\boldsymbol{x_k^e})$ (Ecuación 1.17) como la distancia entre $\boldsymbol{x_k^e}$ y todas las demás muestras pertenecientes a la misma agrupación, $\boldsymbol{c_k}$.

$$a(\boldsymbol{x_k^e}) = \frac{1}{E_k - 1} \sum_{\boldsymbol{x_k^u} \in \boldsymbol{c_k}, \boldsymbol{x_k^e} \neq \boldsymbol{x_k^u}}^{E_k} d(\boldsymbol{x_k^e}, \boldsymbol{x_k^u}) \tag{1.17}$$

En el sumatorio de la Ecuación 1.17 en realidad se están sumado $E_k - 1$ muestras ya que la distancia $d(\boldsymbol{x_k^e}, \boldsymbol{x_k^e})$ da como resultado un valor nulo. Por este motivo, la ecuación se normaliza dividiendo el sumatorio entre $E_k - 1$. $a(\boldsymbol{x_k^e})$ se puede interpretar como una medida de cómo se ha asignado $\boldsymbol{x_k^e}$ en la agrupación $\boldsymbol{c_k}$. Cuanto menor sea ese valor, mejor es la asignación. También se debe definir $b(\boldsymbol{x_k^e})$ como la distancia media de $\boldsymbol{x_k^e}$ a todas las muestras de cualquier otra agrupación de las cuales la muestra $\boldsymbol{x_k^e}$ no pertenece, es decir, menos a las muestras que pertenecen a $\boldsymbol{c_k}$.

$$b(\boldsymbol{x_k^e}) = \min_{\boldsymbol{c_p} \neq \boldsymbol{c_k}} \frac{1}{E_p} \sum_{\boldsymbol{x_p^e} \in \boldsymbol{c_p}} dist(\boldsymbol{x_k^e}, \boldsymbol{x_p^e}) \tag{1.18}$$

Una vez obtenidos $a(\boldsymbol{x_k^e})$ y $b(\boldsymbol{x_k^e})$, se puede definir el valor de la silueta para una muestra $\boldsymbol{x_k^e}$ como,

$$s(\boldsymbol{x_k^e}) = \begin{cases} 1 - \dfrac{a(\boldsymbol{x_k^e})}{b(\boldsymbol{x_k^e})}, & si\ a(\boldsymbol{x_k^e}) < b(\boldsymbol{x_k^e}) \\ 0, & si\ a(\boldsymbol{x_k^e}) = b(\boldsymbol{x_k^e}) \\ \dfrac{b(\boldsymbol{x_k^e})}{a(\boldsymbol{x_k^e})} - 1, & si\ a(\boldsymbol{x_k^e}) > b(\boldsymbol{x_k^e}) \end{cases} \tag{1.19}$$

de la Ecuación 1.19 se puede observar que $s(\boldsymbol{x_k^e})$ toma valores entre [-1,1] y no depende de ninguna unidad ya que es adimensional.

Los coeficientes de silueta $s(\boldsymbol{x_k^e})$ cerca de +1 indican que la muestra $\boldsymbol{x_k^e}$ está muy lejos de los grupos vecinos. Un valor de 0 indica que la muestra está en el límite de decisión entre dos grupos vecinos y los valores negativos indican que esas muestras podrían haberse asignado al grupo incorrecto. También por el grosor del diagrama de silueta se puede visualizar el tamaño de la agrupación. Volviendo con la base de datos *Iris*, se calcula el trazado de silueta para visualizar si el entrenamiento ha agrupado las especies de *Iris* correctamente (Figura 1.13).

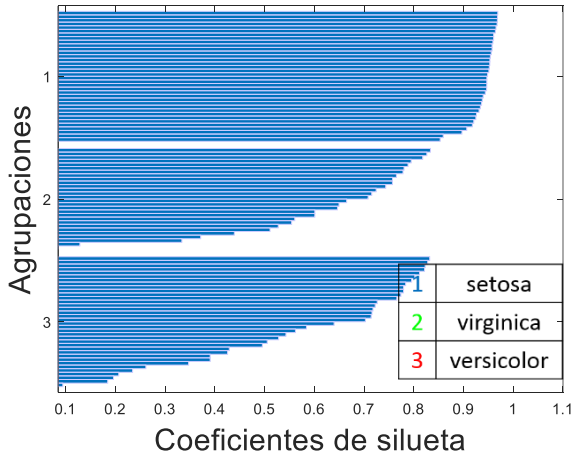

Figura 1.13: Trazado de silueta.

Como se observa en la Figura 1.13, no hay coeficientes de silueta negativos por lo que la clasificación ha sido adecuada. Además, se puede ver que los coeficientes para la agrupación *Setosa* tienen valores altos. Por ello, se puede asegurar con una probabilidad alta de que las *Setosa* las ha clasificado correctamente. Estas afirmaciones, si se comparan con las realizadas anteriormente, se verifican.

También se suele calcular el valor promedio de los coeficientes de silueta como una medida para comprobar si la clasificación ha sido suficientemente correcta. Normalmente se suele considerar para valores superiores a 0.7. En el caso de la base de datos *Iris* ha sido de 0.72, por lo tanto, se considera que la clasificación de la base de datos *Iris* utilizando el método k-means da resultados suficientemente buenos.

1.2.3. Modelo de mezclas gaussianas

El modelo de mezclas gaussianas (MMG) forma grupos al representar la función de densidad de probabilidad de las variables observadas como una mezcla de densidades normales multivariadas, es decir, a cada muestra no le asigna un único grupo sino una probabilidad de pertenecer a cada grupo. Al igual que el método de agrupación de k-means, el modelo de mezclas gaussianas utiliza un algoritmo iterativo que converge a un máximo local. Este modelo puede ser más apropiado que k-means cuando las agrupaciones tienen diferentes tamaños y correlación entre ellos. La idea de utilizar los modelos de mezclas gaussianas para la clasificación fue introducida por R.O. Duda y P. E. Hart en el libro "Pattern Classification and Scene Analysis" [13].

Como la mayoría de los métodos de agrupación, se debe especificar el número de agrupaciones deseadas antes de ajustar el modelo. Al igual que el método de k-means, el modelo de mezclas gaussianas es sensible a las condiciones iniciales y podría converger a un mínimo local. La notación para expresar las agrupaciones será $C = \boldsymbol{c_1}, \dots, \boldsymbol{c_k}, \dots, \boldsymbol{c_K}$. Cada agrupación $\boldsymbol{c_k}$ contiene los siguientes parámetros: media $\boldsymbol{\mu_k}$, matriz de covarianza Σ_k y probabilidad de mezcla π_k.

- La media $\boldsymbol{\mu_k}$ define el centro de la agrupación $\boldsymbol{c_k}$.

$$\boldsymbol{\mu_k} = \frac{1}{E_k} \sum_{e=1}^{E_k} \boldsymbol{x_k^e} \tag{1.20}$$

  siendo $\boldsymbol{x_k^e}$ una muestra perteneciente al grupo $\boldsymbol{c_k}$ y E_k el número de muestras que pertenecen al grupo $\boldsymbol{c_k}$. El conjunto de muestras de entrenamiento X^E se puede expresar como una matriz donde la muestra $\boldsymbol{x^e}$ tiene dimensión N. Por lo tanto, $\boldsymbol{x^e}$ se puede expresar como un vector. Teniendo esto en cuenta, $\boldsymbol{\mu}$ se puede expresar como un vector de dimensión N.

- La matriz de covarianza Σ_k es la generalización natural a dimensiones superiores del concepto de varianza. Se define como el ancho de la función gaussiana, lo que equivaldría a las dimensiones de un elipsoide.

$$\Sigma_k = \frac{1}{E_k} (X_k^E - \boldsymbol{\mu_k})^T (X_k^E - \boldsymbol{\mu_k}) \tag{1.21}$$

  donde X_k^E es la matriz que contiene el conjunto de muestras de entrenamiento que pertenecen a la agrupación $\boldsymbol{c_k}$. El resultado de Σ_k, es una matriz llamada matriz de covarianza.

- Una probabilidad de mezcla π_k define la probabilidad o peso que puede tener cada agrupación. Ya que es un coeficiente de probabilidad, debe cumplir,

$$\sum_{k=1}^{K} \pi_k = 1 \tag{1.22}$$

Antes de explicar el algoritmo, es necesario definir la función de una mezcla gaussiana,

$$p(\boldsymbol{x^e}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_k}, \Sigma_k) \tag{1.23}$$

siendo $\mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_k}, \Sigma_k)$ la función de densidad gaussiana:

$$\mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_k}, \Sigma_k) = \frac{1}{(2\pi)^{N/2}|\Sigma_k|^{1/2}} e^{\left(-\frac{1}{2}(\boldsymbol{x^e}-\boldsymbol{\mu_k})^T {\Sigma_k}^{-1}(\boldsymbol{x^e}-\boldsymbol{\mu_k})\right)} \tag{1.24}$$

La probabilidad conjunta para todas las muestras de entrenamiento viene definida por:

$$p(X^E) = \sum_{e=1}^{E} p(\boldsymbol{x^e}) = \prod_{e=1}^{E} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_k}, \Sigma_k) \tag{1.25}$$

Por conveniencia, se puede calcular la probabilidad logarítmica del modelo $\ln p(X^E)$ (Ecuación 1.26), ya que posteriormente se usará como un cálculo dentro del algoritmo del MMG. Además, a esta función, en teoría de probabilidad, se la denomina función de estimación por máxima verosimilitud (MLE) y convierte el productorio de la Ecuación 1.25 en un sumatorio, lo que hace que a nivel computacional sea una gran ventaja. El resultado de la ecuación es un número escalar.

$$\ln p(X^E) = \sum_{e=1}^{E} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_k}, \Sigma_k) \tag{1.26}$$

Lo que trata el algoritmo del MMG es determinar los valores que maximizan la probabilidad de los parámetros $\boldsymbol{\mu_k}$, Σ_k y π_k. Para ello, irá calculando los valores de manera iterativa hasta converger a los valores óptimos, es decir, hasta que $\ln p(X^E)$ converja o tenga una variación muy pequeña. Los pasos que intervienen en el algoritmo del MMG son los siguientes:

i) Inicialización parámetros

En primer lugar, se deben inicializar los parámetros del MMG ($\pi_k, \boldsymbol{\mu_k}\ y\ \Sigma_k$). Para establecer los valores iniciales $\boldsymbol{\mu_k}$, se puede hacer de manera aleatoria o se pueden inicializar entrenando el algoritmo k-means como paso previo y utilizar las posiciones de los centroides como inicialización de los valores de $\boldsymbol{\mu_k}$. De esta manera, se ayuda al algoritmo a lograr mejores resultados. Los otros parámetros se pueden inicializar como $\pi_k = \frac{1}{N_k}$ y Σ_k como la matriz de identidad.

ii) Cálculo de la esperanza

El segundo paso consiste en calcular la probabilidad de que las muestras pertenezcan a cada agrupación. Se calcula la esperanza $\gamma(\boldsymbol{c_k}|\boldsymbol{x^e})$ mediante la expresión,

$$\gamma(\boldsymbol{c_k}|\boldsymbol{x^e}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_k}, \Sigma_k)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(\boldsymbol{x^e}|\boldsymbol{\mu_j}, \Sigma_j)} \tag{1.27}$$

donde $\gamma(\boldsymbol{c_k}|\boldsymbol{x^e})$ es la probabilidad de que x_i pertenezca a la agrupación $\boldsymbol{c_k}$.

La ecuación anterior se ha obtenido utilizando la teoría de la probabilidad, en particular, utilizando el teorema de Bayes [14].

iii) Actualización de los parámetros

En este paso se actualizan los parámetros del MMG haciendo uso del cálculo de la esperanza. Como $\gamma(\boldsymbol{c_k}|\boldsymbol{x^e})$ es común para las expresiones $\pi_k, \boldsymbol{\mu_k}\ y\ \Sigma_k$, se puede definir:

$$\xi_k = \sum_{i=1}^{N} \gamma(\boldsymbol{c_k}|\boldsymbol{x^e}) \tag{1.28}$$

De este modo se puede redefinir los parámetros anteriores como:

$$\pi_k{}^* = \frac{\xi_k}{E_k} \tag{1.29}$$

$$\boldsymbol{\mu_k}{}^* = \frac{1}{\xi_k} \sum_{e=1}^{E} \gamma(\boldsymbol{c_k}|\boldsymbol{x^e}) \boldsymbol{x^e} \tag{1.30}$$

$$\Sigma_k{}^* = \frac{1}{\xi_k} \sum_{e=1}^{E} \gamma(\boldsymbol{c_k}|\boldsymbol{x^e})(\boldsymbol{x^e} - \boldsymbol{\mu_k})(\boldsymbol{x^e} - \boldsymbol{\mu_k})^T \tag{1.31}$$

iv) Estudio de la convergencia de la función $\ln p(\boldsymbol{X^E})$

En el último paso se evalúa la función $\ln p(X^E)$ con los parámetros actualizados. Si converge o tiene una variación muy pequeña respecto a la iteración anterior, el algoritmo se detiene y, por lo tanto, se obtienen los parámetros $\pi_k, \boldsymbol{\mu_k}\ y\ \Sigma_k$. Si por el contrario no converge, habrá que volver a calcular la expectativa con los nuevos parámetros y actualizarlos hasta que $\ln p(X^E)$ converja. También se puede definir un número máximo de iteraciones, aunque este método garantiza que se alcance un máximo local, es decir, que $\ln p(X^E)$ converja.

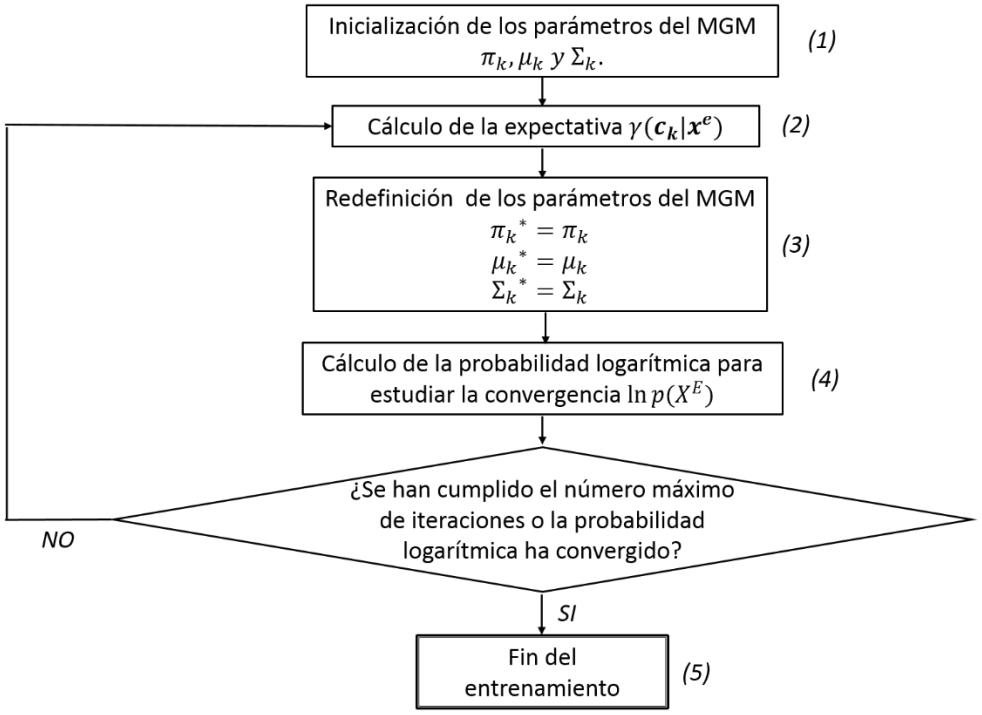

Figura 1.14: Diagrama de flujo de entrenamiento para el MMG.

Para cada agrupación, el algoritmo da como resultado los parámetros $\pi_k, \boldsymbol{\mu_k}$ y Σ_k, donde $\boldsymbol{\mu_k}$ contiene los centros de los grupos (centroides) y Σ_k las dimensiones del elipsoide.

Por último, utilizando la base de datos *Iris*, se realiza el proceso de agrupamiento utilizando el modelo de mezclas gaussianas. Al igual que con el método k-means, se puede validar el entrenamiento con muestras de test o con nuevas muestras que se necesiten clasificar. Para ello, basta con calcular para cada nueva muestra cual es el elipsoide más cercano ya que, como se ha explicado antes, cada elipsoide representa una agrupación. Los resultados pueden verse en la Figura 1.15.

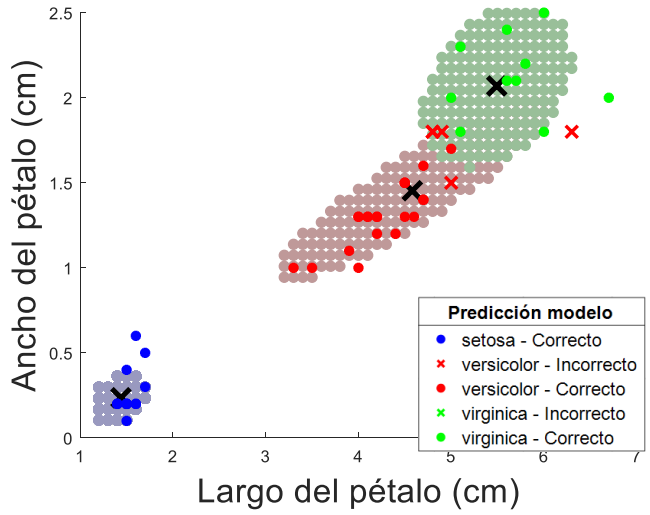

Figura 1.15: Representación del ancho de los pétalos en función del largo de los pétalos utilizando el método MMG. También se han representado los elipsoides del MMG, marcando los centros de las agrupaciones con 'x' en negro.

Como se puede observar en la Figura 1.15, el agrupamiento ha mejorado respecto al k-means ya que este método no solo calcula los centroides, sino que tiene en cuenta cómo se estructuran los datos mediante el cálculo de elipsoides. Este hecho hace que la tasa de acierto se eleve hasta un 90%. El problema que tiene validar este método utilizando los elipsoides es que calcularlos tiene un coste computacional alto y, además, depende del orden en el que se calculen los elipsoides los resultados pueden variar. Este hecho se puede observar en la Figura 1.15 ya que se observa que el elipsoide verde se solapa con el rojo debido a que se ha calculado primero el rojo. En caso de calcular primero el elipsoide verde, se obtienen unos resultados peores, siendo la tasa de acierto del 84%. Si para validar los resultados se utiliza una RNA como en el caso de la agrupación jerárquica, la clasificación ya no depende del orden en el que se agrupan los datos. Por ello, utilizando la RNA se obtiene una tasa de acierto del 100%. La RNA que se ha utilizado consta de una única capa oculta compuesta por 5 neuronas, ya que en el capítulo anterior se demostró que para entrenar la base de datos *Iris* era la arquitectura más apropiada.

A continuación, se representa en la Tabla I las ventajas y desventajas que tienen los métodos de aprendizaje no supervisado que se han visto.

TABLA I: COMPARACIÓN ENTRE LOS MÉTODOS DE APRENDIZAJE NO SUPERVISADO

Métodos de aprendizaje no supervisado	**Tasa de acierto (%)**	**Tasa de acierto con RNA (%)**	**Tiempo de ejecución del algoritmo de entrenamiento[1] (s)**
Agrupación jerárquica	---	70%	4.6365e-04
K-means	86%	86%	0.0021
MMG	90%	100%	0.0039

El rendimiento de los algoritmos se ha comparado calculando el tiempo de ejecución del entrenamiento. Como se observa en la Tabla I, el algoritmo que presenta un tiempo menor es el método de agrupación jerárquica. Este algoritmo tiene un numero de iteraciones fijo, pero depende del número de muestras de entrenamiento. Por lo tanto, si se tiene un gran número de muestras de entrenamiento, el k-means será un método más rápido. En la Tabla II se han argumentado las ventajas y desventajas que hay entre los diferentes métodos de aprendizaje no supervisado.

En el próximo capítulo se introducirán los mapas autoorganizados (SOM). Estos pertenecen a los métodos de aprendizaje no supervisado ya que para entrenarse no se necesita conocer las etiquetas o grupos a las que pertenecen las muestras de entrenamiento. La principal ventaja de este método es que la evaluación de la red es muy sencilla y completamente paralelizable lo que supone que la implementación en hardware pueda ser en tiempo real para un gran número de campos de aplicación. Además, ofrecen un compromiso muy bueno entre velocidad e interpretabilidad.

[1] Los tiempos de ejecución del algoritmo de entrenamiento se han calculado utilizando un procesador Intel® Core™ i7-4790 de 4 núcleos y 8 hilos con una frecuencia turbo máxima de 4GHz.

TABLA II: VENTAJAS Y DESVENTAJAS ENTRE LOS MÉTODOS DE APRENDIZAJE NO SUPERVISADO

Métodos de aprendizaje no supervisado	**Ventajas**	**Desventajas**
Agrupación jerárquica	• Fácil de implementar. • La agrupación jerárquica genera una jerarquía, es decir, una estructura que es más informativa que el conjunto no estructurado de agrupaciones devueltas por k-means. Por lo tanto, es más fácil decidir el número de agrupaciones observando el dendrograma. • Son especialmente potentes cuando el conjunto de datos contiene relaciones jerárquicas reales.	• No es posible deshacer el paso anterior: una vez que la muestra de entrenamiento se ha asignado a una agrupación, ya no se puede mover. • No es adecuado para grandes conjuntos de datos ya que los tiempos de ejecución son linealmente proporcionales al número de muestras de entrada. • Muy sensible a los valores atípicos.
K-means	• Fácil de implementar. • Con una gran cantidad de variables, k-means puede ser computacionalmente más rápido que el agrupamiento jerárquico (si k es pequeño). • Una muestra de entrenamiento puede cambiar de agrupación cuando se vuelven a calcular los centroides.	• El número agrupaciones k es difícil de predecir. • Las semillas iniciales tienen un fuerte impacto en los resultados finales. • El orden de los datos tiene un impacto en los resultados finales. • Es sensible a la escala: reescalar los conjuntos de datos (normalizarlos o estandarizarlos) cambia completamente los resultados. Si bien esto no es malo, hay que dedicarle un tiempo extra a escalar los datos.
MMG	• A diferencia de k-means, MMG tiene en cuenta la forma en la que están distribuidas las muestras de entrenamiento. • El método MMG se puede ver como una mejora del método k-means ya que no solo predice a que agrupaciones pertenecen las muestras de entrenamiento, sino que además proporciona las probabilidades de que una muestra dada pertenezca a cada una de las agrupaciones posibles.	• Al igual que k-means, el número agrupaciones k es difícil de predecir. • Las semillas iniciales tienen un fuerte impacto en los resultados finales. • El orden de los datos tiene un impacto en los resultados finales. • Es más complejo de implementar. • El tiempo de ejecución del algoritmo es mayor.

Capítulo 2

Mapas Autoorganizados (SOM)

Los mapas autoorganizados, o también llamados Self-Organizing Maps (SOM), son un tipo de red neuronal artificial que se entrena utilizando técnicas de aprendizaje no supervisado para producir una representación discreta de baja dimensión del espacio de las muestras de entrada, llamado mapa.

Los mapas autoorganizados difieren de otras redes neuronales artificiales, ya que aplican el aprendizaje competitivo en oposición al aprendizaje de corrección de errores, en el sentido que estos usan una función de vecindad para preservar las propiedades topológicas del espacio de entrada.

El modelo fue presentado en 1982 por el profesor finlandés T. Kohonen, basado en ciertas evidencias descubiertas a nivel cerebral [15].

2.1. Fundamentos del algoritmo SOM

Un mapa autoorganizado está formado por componentes llamados nodos o neuronas. Cada neurona está asociada con un vector de pesos, de la misma dimensión de los vectores de entrada, y es una posición en el mapa (espacio de entrada). Normalmente las neuronas están colocadas en forma de una red bidimensional.

El entrenamiento del SOM consiste en trasladar vectores de peso hacia los datos de entrada (reduciendo la distancia métrica) teniendo en cuenta la topología inducida por el espacio del mapa. Por lo tanto, los mapas autoorganizados son útiles para visualizar un espacio de mayor dimensión a uno de menor dimensión. Una vez entrenado, se ubica un vector del espacio de los datos en el mapa, encontrando la neurona con el vector de pesos más cercano (menor distancia métrica) al vector del espacio de los datos.

Como se ha indicado anteriormente, este modelo emplea el aprendizaje competitivo. En él, las neuronas compiten unas con otras con el fin de llevar a cabo una tarea dada. Se pretende que cuando se presente a la red un patrón de entrada, sólo una de las neuronas de salida (o un grupo de vecinas) se active. Por tanto, las neuronas compiten por activarse, quedando finalmente una como neurona vencedora y anuladas el resto, que son forzadas a sus valores de respuesta mínimos.

El objetivo de este aprendizaje es categorizar los datos que se introducen en la red. Se clasifican valores similares en la misma categoría y, por tanto, deben activar la misma neurona de salida. Las clases o categorías deben ser creadas por la propia red, puesto que se trata de un aprendizaje no supervisado, a través de las correlaciones entre los datos de entrada.

Los pasos del entrenamiento del SOM comienzan al inicializar los vectores de peso. A partir de ahí, se selecciona un vector de muestra al azar y se busca en el mapa de vectores de peso para encontrar qué peso representa mejor esa muestra. Cada vector de peso tiene pesos vecinos que están cerca de él. El peso que se elige, se recompensa al ser más parecido a ese vector de muestra seleccionado al azar. Los vecinos de ese peso también son recompensados al poder parecerse más al vector de muestra elegido. Esto permite que el mapa crezca y genere diferentes formas.

2.2. Arquitectura de la red SOM

Un modelo SOM está compuesto por dos capas de neuronas. La capa de entrada (formada por N neuronas, una por cada atributo de entrada) se encarga de recibir y transmitir a la capa de salida la información procedente del exterior. La capa de salida (formada por M neuronas) es la encargada de procesar la información y formar el mapa de rasgos. Normalmente, las neuronas de la capa de salida se organizan en forma de mapa bidimensional (Figura 2.1).

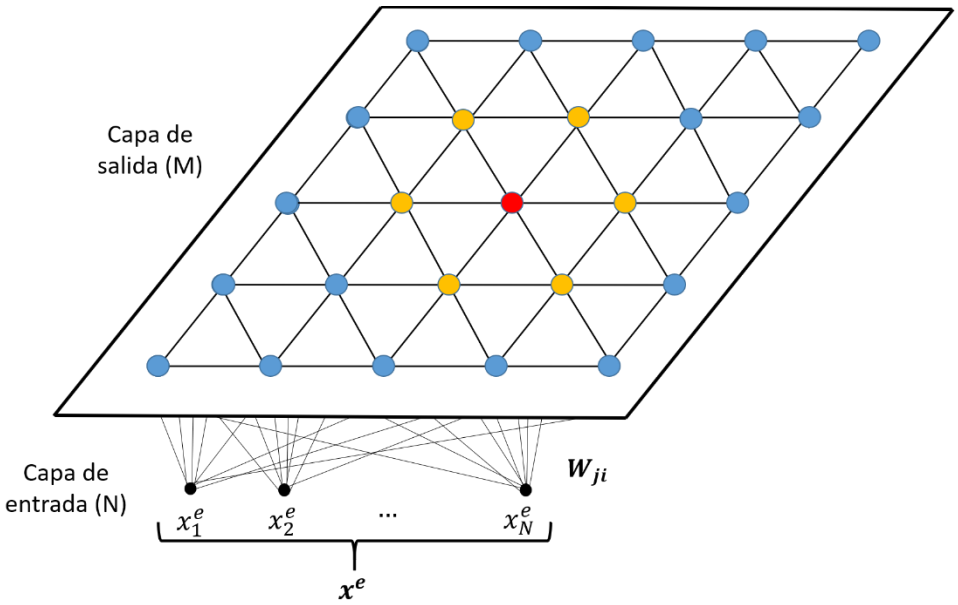

Figura 2.1: Arquitectura de la red SOM correspondiente a una topología hexagonal.

La topología y el número de neuronas de la capa de salida (M) permanece fijo desde el principio. El número de neuronas determina la suavidad de la proyección, lo cual influye en el ajuste y capacidad de generalización del SOM, aunque debe ser inferior al número de muestras de entrenamiento. Como las neuronas de la capa de salida forman un mapa bidimensional, hay que especificar el número de neuronas que se asignan a cada fila y a cada columna. Por ejemplo, en la Figura 2.1, se puede observar que la capa de salida está formada por 5 filas y 5 columnas de neuronas. Por lo tanto, la red está compuesta por $M = 25\ neuronas$. Escoger el número de neuronas es arbitrario, aunque J. Tian, M. H. Azarian y M. Pecht propusieron en el artículo "Anomaly Detection Using Self-Organizing Maps-Based K-Nearest Neighbor Algorithm" [16] utilizar la siguiente ecuación,

$$M \approx 5\sqrt{E} \tag{2.1}$$

siendo E el número de muestras de entrenamiento.

Se pueden utilizar diferentes topologías para ubicar las neuronas de la capa de salida. Cuanto más interconexionada esté una neurona, es decir, cuantas más conexiones tenga con otras neuronas, la actualización de la red será más precisa. Las topologías más frecuentes son la rectangular y la hexagonal. En la rectangular, cada neurona está interconectada con 4 neuronas vecinas. Pero, en cambio, en la hexagonal cada neurona está interconectada con 6 neuronas vecinas. Por lo tanto, la topología hexagonal dará mejores resultados. En la Figura 2.1 se puede observar la topología hexagonal y, para cada neurona (roja), cuáles son sus neuronas vecinas (amarillas).

Las conexiones entre las dos capas que forman la red son siempre hacia delante, es decir, la información se propaga desde la capa de entrada hacia la capa de salida. Cada neurona de entrada i está conectada con cada una de las neuronas de salida j mediante un peso W_{ji} (ver Figura 2.2). De esta forma, las neuronas de salida tienen asociado un vector de pesos $\boldsymbol{W_j} = (W_{j1}, \dots, W_{ji}, \dots, W_{jN})$, de la misma dimensión de los vectores de entrada, indicando una posición en el mapa. Así, el SOM define una proyección desde un espacio de datos en alta dimensión a un mapa bidimensional de neuronas.

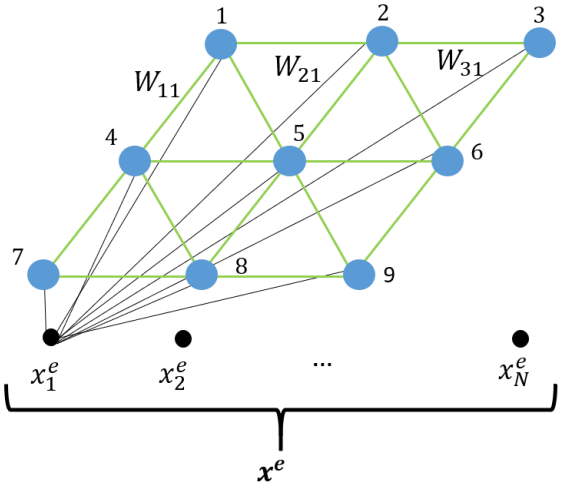

Figura 2.2: Vector de pesos para la neurona de entrada $i = 1$ asociada a cada neurona j.

2.3. Algoritmo de entrenamiento de la red SOM

Durante la fase de entrenamiento, el SOM forma una red elástica que se pliega dentro de la nube de datos originales. El algoritmo controla la red de modo que tiende a aproximar la densidad de los datos. Los vectores de pesos se acercan a las áreas donde la densidad de datos es alta. Eventualmente unos pocos vectores de pesos están en áreas donde existe baja densidad de datos.

Como se mencionó anteriormente, este tipo de red neuronal artificial no supervisada utiliza el aprendizaje competitivo para actualizar sus pesos. El proceso se basa en tres partes: competencia entre neuronas, determinación de las neuronas que pertenecen a la vecindad de la neurona ganadora y adaptación de los pesos.

i) Competencia entre neuronas

A cada neurona de la capa de salida j se le asigna un vector de peso con la misma dimensionalidad que el espacio de entrada $\boldsymbol{W_j} = (W_{j1}, \dots, W_{ji}, \dots, W_{jN})$. Se calculan las distancias entre cada neurona de la capa de salida con las neuronas de entrada. La neurona de

salida con la distancia más baja será la neurona ganadora $\boldsymbol{c}$ (Ecuación 2.2). A esta neurona se la denomina BMU (Best Matching Unit).

$$A_c = \min_j\{\|\boldsymbol{x^e} - \boldsymbol{W_j}\|\} \tag{2.2}$$

siendo $\boldsymbol{x^e} = (x_1^e, \dots, x_i^e, \dots, x_N^e)$ una muestra de la base de datos de entrenamiento X^E.

Normalmente, para calcular la distancia entre las neuronas, se utiliza la distancia euclídea (Ecuación 2.3).

$$d^2(\boldsymbol{x^e}, \boldsymbol{W_j}) = \sum_{i=1}^{N} (x_i^e - W_{ji})^2 \tag{2.3}$$

ii) Determinación de las neuronas que pertenecen a la vecindad de la neurona ganadora

Antes de actualizar los pesos, hay que determinar que neuronas pertenecen a la vecindad de la neurona ganadora. Para ello, primero hay que calcular el tamaño del radio de vecindad centrado en la neurona ganadora y, a continuación, determinar qué neuronas están dentro de dicho radio.

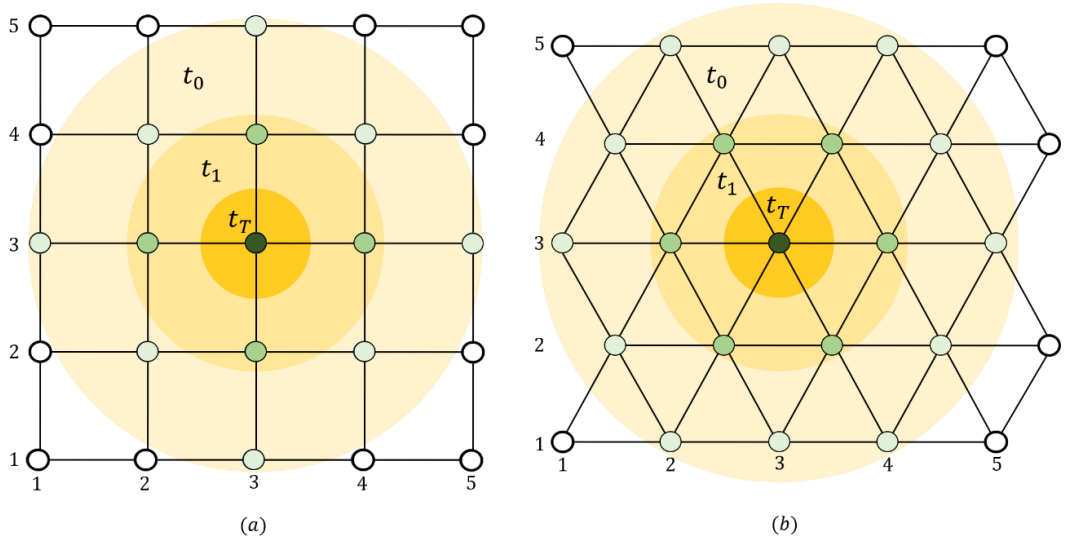

Figura 2.3: Actualización del radio de vecindad para una topología rectangular (a) y hexagonal (b) suponiendo que la neurona del centro es la neurona ganadora (BMU).

Una de las características del algoritmo de aprendizaje del SOM es que el área del vecindario se reduce con el paso del número de iteraciones, es decir, que el radio de vecindad disminuye (ver Figura 2.3). Para ello, se utiliza la función de decaimiento exponencial (Ecuación 2.4), que calcula para cada iteración $t = t_0, t_1, \dots, t_T$, la dimensión del radio de vecindad.

$$\sigma(t) = \sigma_o \, e^{\left(-\frac{t}{\tau}\right)} \tag{2.4}$$

donde σ_o corresponde al radio de vecindad para la iteración t_0:

$$\sigma_o = \frac{\max\,(filas, columnas)}{2} \tag{2.5}$$

Filas y columnas se refiere al número de neuronas que hay en cada fila y en cada columna respectivamente. Por ejemplo, en la Figura 2.3, se puede observar que la capa de salida está formada por 5 filas y 5 columnas de neuronas. Por lo tanto, $\sigma_o = \frac{5}{2}$.

τ es una constante denomina vida media que se define como,

$$\tau = \frac{T}{\log \sigma_o} \tag{2.6}$$

siendo T el número total de iteraciones que realiza el algoritmo de entrenamiento.

Una vez obtenido el radio de vecindad para cada iteración, hay que determinar que neuronas se encuentran dentro del radio. Para ello, hay que calcular la distancia entre la neurona ganadora y todas las neuronas de la red, y comprobar si están dentro del radio de vecindad (Ecuación 2.7).

$$d^2(\boldsymbol{j}, \boldsymbol{c}) = \sum_{i=1}^{N} (j_i - c_i)^2 = \begin{cases} \leq \sigma(t), & \boldsymbol{x^e} \in N_{\boldsymbol{c}}(t) \\ > \sigma(t), & \boldsymbol{x^e} \notin N_{\boldsymbol{c}}(t) \end{cases} \tag{2.7}$$

siendo $\boldsymbol{j}$ una neurona de la capa de salida y $\boldsymbol{j} \neq \boldsymbol{c}$.

Por lo tanto, las neuronas que pertenecen a la vecindad de la neurona ganadora $\boldsymbol{c}$ las denotaremos con $N_{\boldsymbol{c}}(t)$. Volviendo a la Figura 2.3 (a) topología rectangular, para la iteración t_0, hay 12 neuronas que pertenecen a la vecindad de la neurona ganadora y, para la iteración t_1, solo pertenecen a la vecindad 4 neuronas. En cambio, en la Figura 2.3 (b) topología hexagonal, para la iteración t_0, hay 18 neuronas que pertenecen a la vecindad de la neurona ganadora y, para la iteración t_1, pertenecen a la vecindad 6 neuronas.

iii) Adaptación de los pesos

Una vez elegida la neurona ganadora $\boldsymbol{c}$ y las neuronas pertenecientes a su vecindad $N_{\boldsymbol{c}}(t)$, se deben actualizar los pesos para que las neuronas se acerquen a la observación de entrada $\boldsymbol{x^e}$, aunque no de la misma manera (ver Figura 2.4). Cuanto más alejadas estén las neuronas de las muestras de entrada $\boldsymbol{x^e}$, el ajuste de los pesos será menor.

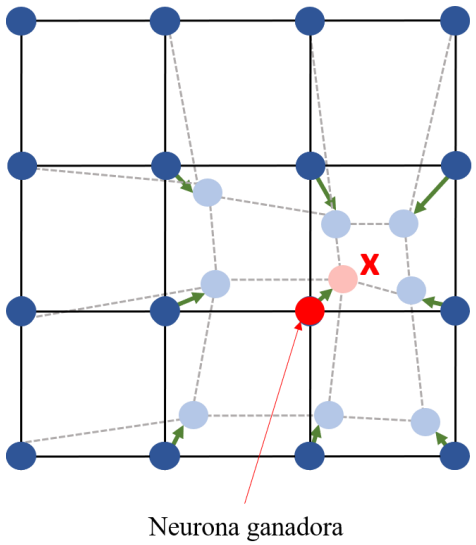

Figura 2.4: Actualización de los pesos de las neuronas de la capa de salida para una topología rectangular.

Los pesos de la neurona ganadora y la de sus vecinas se actualizan mediante la siguiente fórmula,

$$W_{ji}(t+1) = W_{ji}(t) + \alpha(t) \cdot h_{ic}(t) \cdot \left(x_i^e - W_{ji}(t)\right), \quad \boldsymbol{j} \in N_{\boldsymbol{c}}(t) \tag{2.8}$$

siendo $\alpha(t)$ la función de proporción de aprendizaje, que decae con el paso del número de iteraciones. Se puede definir como,

$$\alpha(t) = \alpha_0 \cdot e^{\left(-\frac{t}{T}\right)} \tag{2.9}$$

Además de la tasa de aprendizaje, se debe definir la función núcleo de vecindad $h_{ic}(t)$ (Ecuación 2.10) para que el efecto del aprendizaje sea proporcional a la distancia entre la neurona ganadora y las neuronas pertenecientes a su vecindad. Es decir, cuanto más alejadas estén las neuronas de la neurona ganadora, la actualización de los pesos debe ser menos significativa.

$$h_{ic}(t) = exp\left(-\frac{d_{ic}^2}{2\,\sigma^2(t)}\right) \tag{2.10}$$

donde d_{ic} es la distancia entre la neurona ganadora $\boldsymbol{c}$ y las neuronas pertenecientes a su vecindad $N_{\boldsymbol{c}}(t)$.

En la Figura 2.5 se muestran los pasos del proceso de entrenamiento que se han descrito anteriormente.

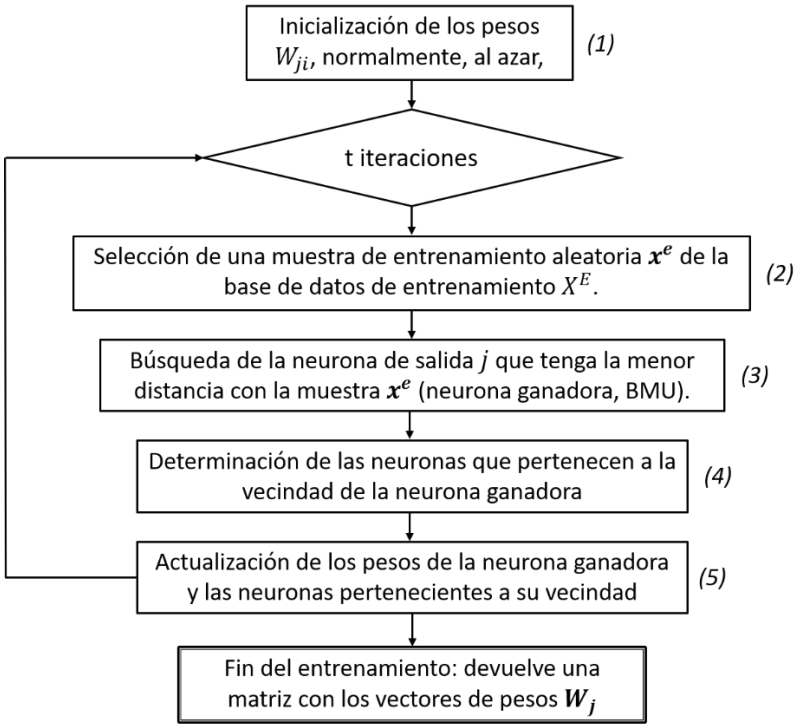

Figura 2.5: Diagrama de flujo de entrenamiento del SOM.

2.4. Evaluación de la red SOM

2.4.1. Matriz-U

Una vez actualizados los pesos de las neuronas, es necesario definir un método para clasificarlas. La manera más común de realizar esto es creando una matriz llamada matriz-U (matriz de distancia unificada), que calcule las distancias entre las neuronas más próximas [17]. Para ello, por cada neurona de salida j, se calcula la distancia entre ella y sus neuronas vecinas más próximas. Por ejemplo, para una topología hexagonal, cada neurona tiene 6 neuronas vecinas próximas (ver Figura 2.1). Cuanto menor sea el resultado, más próxima estará la neurona con sus vecinas. Por lo tanto, el resultado es una matriz de la misma dimensión y tamaño que la matriz de pesos W, que indica cuanto de cerca está cada neurona con sus vecinas más próximas. Normalmente, con la matriz-U se suele hacer una representación gráfica, que muestre las distancia en colores (Figura 2.6).

MATLAB representa la matriz-U con la función plotsomnd(net). En ella, representa las neuronas como hexágonos azules y con líneas rojas la conexión entre las neuronas vecinas. Los colores en las regiones que contienen las líneas rojas indican las distancias entre las neuronas. Los colores más oscuros representan distancias mayores y los colores más claros representan distancias menores.

Para ver esta representación con un ejemplo, se hace uso la base de datos *Iris*. Primero, se entrena la base de datos utilizando el *toolbox* de MATLAB especializado en crear, entrenar y visualizar los mapas autoorganizados (SOM). Haciendo uso de la Ecuación 2.1 se obtiene que se necesitan aproximadamente 50 neuronas para que la red se entrene correctamente. Para este caso, se han utilizado 64 neuronas ya que se ha observado que los resultados se ajustan mejor utilizando este último número de neuronas. Una vez entrenada la red, se crea la representación de la matriz-U. Como se puede observar en la Figura 2.6, hay una banda de segmentos oscuros que cruza desde la región del centro inferior hasta la región superior izquierda, lo que significa que hay dos agrupaciones que están claramente separadas. A continuación, se escoge un método que clasifica las neuronas en agrupaciones basándose en el cálculo de la matriz-U. Existen otras formas de evaluar la red SOM. Por ejemplo, una vez calculados los pesos de las neuronas, clasificarlas utilizando algún método de aprendizaje no supervisado visto en el capítulo anterior.

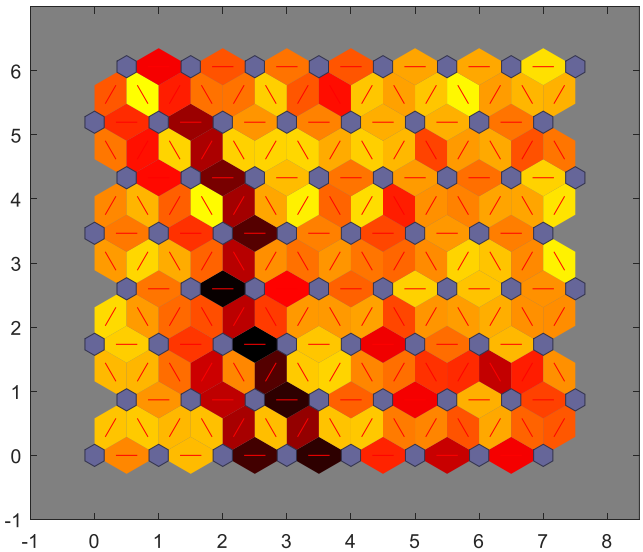

Figura 2.6: Representación de la matriz-U para la base de datos *Iris*.

2.4.2. Clasificación

En este apartado se agrupan las neuronas en clases basándose en el cálculo de la matriz-U. También se podría haber utilizado algún otro método de clasificación, ya que una vez entrenado el SOM, la evaluación es independiente.

Una vez obtenida la matriz-U, se separan las neuronas en agrupaciones mediante un valor umbral que representa la distancia para la cual dos neuronas se consideran que están lo suficientemente separadas como para considerar que pertenecen a dos agrupaciones distintas. Escoger bien este parámetro es una de las dificultades que tiene este método de clasificación ya que en principio no se sabe cuál puede ser el que dé mejores resultados. Haciendo uso del *toolbox* del CIS se consigue clasificar las neuronas utilizando un umbral que contempla valores entre 0 y el $(número\ de\ neuronas) - 1$ [18]. El umbral 0 es la distancia máxima para la cual todas las neuronas pertenecen a una agrupación distinta y $(número\ de\ neuronas) - 1$ es la distancia mínima para la cual todas las neuronas pertenecen a la misma agrupación. A partir de estos dos umbrales el algoritmo realiza la interpolación de las distancias. El la Figura 2.7 se observa la clasificación de los pesos de las neuronas que se han entrenado haciendo uso de la base de datos *Iris*. Para determinar el umbral, se ha buscado uno que dé como resultado tres agrupaciones. Por ello, se ha utilizado un umbral de 8.

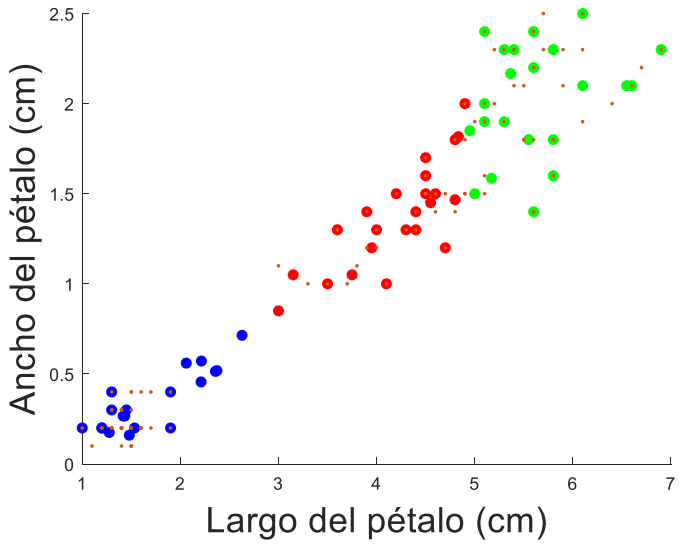

Figura 2.7: Representación de los pesos de las neuronas una vez entrenados. Los puntos más pequeños representan las muestras que se han utilizado en el entrenamiento.

Una vez clasificadas las neuronas en agrupaciones, se pueden clasificar las muestras de entrenamiento o cualquier otra muestra, ya que la clasificación es independiente de las muestras que se han utilizado para entrenar el SOM. Por ello, para cada muestra que se quiera clasificar, se calcula cual es la neurona que tiene la menor distancia a la muestra (neurona ganadora, BMU), y se considera que la muestra pertenece a la misma agrupación que la neurona ganadora. Para verificar el método, se han empleado las muestras de test y se han clasificado utilizando este último método (Figura 2.8).

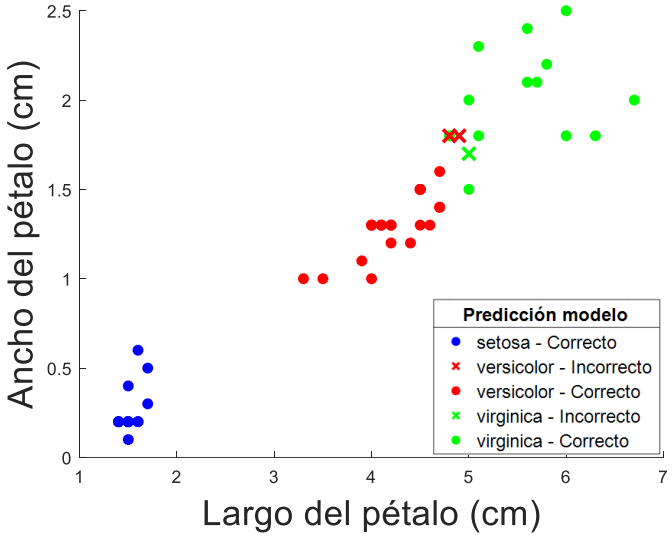

Figura 2.8: Representación del ancho de los pétalos en función del largo de los pétalos utilizando el método de clasificación del SOM.

Como se puede observar en la Figura 2.8, la clasificación ha sido muy buena con una tasa de acierto del 94%. Si se comparan los resultados con los demás algoritmos de aprendizaje no supervisado (Tabla III), se comprueba que es el que más se ajusta a la clasificación real. La parte negativa del algoritmo de clasificación es que las semillas iniciales tienen un fuerte impacto en los resultados finales teniendo que realizar varias veces el entrenamiento hasta obtener los resultados deseados. El tiempo de ejecución del algoritmo de entrenamiento para el SOM es el más elevado entre los diferentes métodos, pero la mayor ventaja que tiene es que la implementación de la evaluación de la red es muy rápida, siendo completamente paralelizable. Esto hace que la clasificación de nuevas muestras se pueda implementar en tiempo real en hardware para un gran número de campos de aplicación. En la Tabla III, se observan las principales ventajas y desventajas que tiene utilizar la red SOM para la clasificación.

TABLA III: COMPARACIÓN ENTRE LOS MÉTODOS DE APRENDIZAJE NO SUPERVISADO

Métodos de aprendizaje no supervisado	**Tasa de acierto (%)**	**Tiempo de ejecución del algoritmo de entrenamiento[1] (s)**
Agrupación jerárquica	70% (RNA)	4.6365e-04
K-means	86%	0.0021
MMG	90%	0.0039
SOM	94%	0.2504

TABLA IV: VENTAJAS Y DESVENTAJAS DEL SOM

Métodos de aprendizaje no supervisado	Ventajas	Desventajas
SOM	• La implementación de la evaluación de la red en hardware se puede realizar en tiempo real para un gran número de campos de aplicación. • Los resultados que se obtienen son visuales y el análisis es más intuitivo que k-means o MMG. • El SOM preserva la topología de las muestras de entrenamiento porque las distancias en el espacio bidimensional reflejan las del espacio de alta dimensión, por lo tanto, es más preciso. • Es menos sensible al ruido presente en las muestras de entrenamiento.	• Hay que determinar el número de neuronas. • Las semillas iniciales tienen un fuerte impacto en los resultados finales. • Visualmente es difícil separar agrupaciones que estén muy próximas. • Después de entrenar el SOM es necesario definir un método para la evaluación de la red.

Para este trabajo, el tiempo de ejecución del algoritmo de entrenamiento no supone ningún inconveniente ya que no es necesario implementarlo en hardware. El algoritmo de entrenamiento se ejecutará en un microprocesador y, una vez obtenidos los parámetros entrenados (los pesos de las neuronas y las etiquetas a las que pertenecen), se realizará la implementación de la evaluación de la red en hardware. Para ello, se utilizarán las FPGAs ya que se podrá realizar la implementación en tiempo real en muchos campos de aplicación.

Capítulo 3

Implementación de una red SOM en FPGA

Para que la evaluación de la red SOM pueda ser utilizada en una aplicación práctica, es necesario que la implementación se pueda llevar a cabo en tiempo real. El dispositivo que se utilice debe ser capaz de calcular los datos y suministrar una salida a alta velocidad. Por ello, para la implementación se hace uso de la FPGA (*Field-Programmable Gate Array*). El dispositivo es una matriz de puertas lógicas programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada desde el exterior del dispositivo. Además, proporciona una rápida ejecución en operaciones producto y son especialmente útiles para aquellas aplicaciones que necesitan un alto grado de paralelismo.

3.1. Arquitectura del SOM

La arquitectura diseñada está compuesta por 5 módulos: los registros de entrada, las neuronas, los comparadores, la ROM interna y el controlador (Figura 3.1). Con el fin de que la respuesta sea dada en el menor tiempo posible, la arquitectura ha sido diseñada para que los procesos se calculen en paralelo. Por este motivo, para calcular cual es la neurona que tiene la menor distancia a la muestra de entrada se hace uso de los comparadores. Estos van comparando las salidas de las neuronas de dos en dos hasta obtener la neurona con la menor distancia. Además, mediante un leguaje de descripción de hardware VHDL (VHSIC (*Very High Speed Integrated Circuit*) y HDL (*Hardware Description Language*)) se consigue escalar la arquitectura dependiendo del número de neuronas que se deseen implementar o del número de características que tenga la base de datos.

Dentro de la arquitectura de la FPGA, se implementa una ROM interna que guarda los pesos de las neuronas y las agrupaciones a las que pertenecen una vez se haya completado el entrenamiento del SOM y se hayan clasificado las neuronas. También, se utiliza una señal de *Reset* que borra todos los registros y una señal de *Enable* que habilita el controlador.

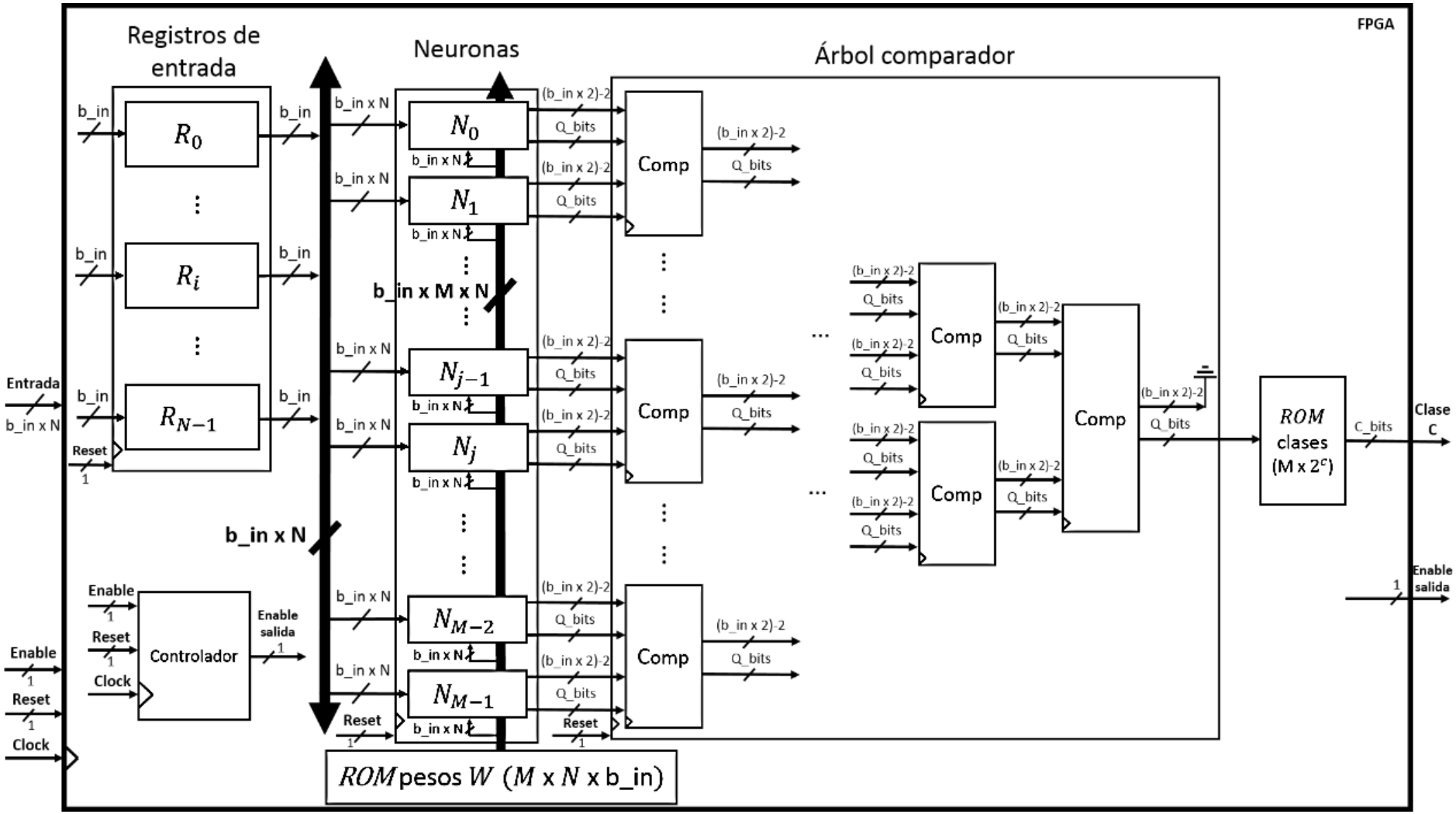

Figura 3.1: Arquitectura del circuito completo.

3.1.1. Registros de entrada

Para llevar las muestras de entrada se utilizan los registros de entrada (Figura 3.2). Éstos llevan la señal a la FPGA de manera síncrona, es decir, con el flanco ascendente del reloj. El número de registros de entrada depende del número de características de entrada ya que, para cada característica, habrá que implementar un registro.

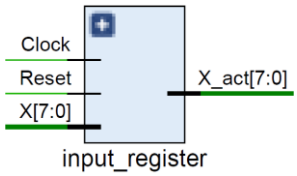

Figura 3.2: Registro de entrada.

Las entradas al registro pueden leerse desde una SRAM, de un bus o de un periférico FIFO (*First In First Out*).

3.1.2. Neuronas

Los componentes neuronas (Figura 3.3) calculan la distancia euclídea al cuadrado entre la muestra de entrada y el peso de una neurona (Ecuación 3.1). La arquitectura del circuito completo debe incorporar *M* componentes de neurona, siendo *M* el número total de neuronas con las que se ha entrenado previamente el SOM.

$$N_j = d^2(\boldsymbol{x}, \boldsymbol{W_j}) = \sum_{i=1}^{N} (x_i - W_{ji})^2 \tag{3.1}$$

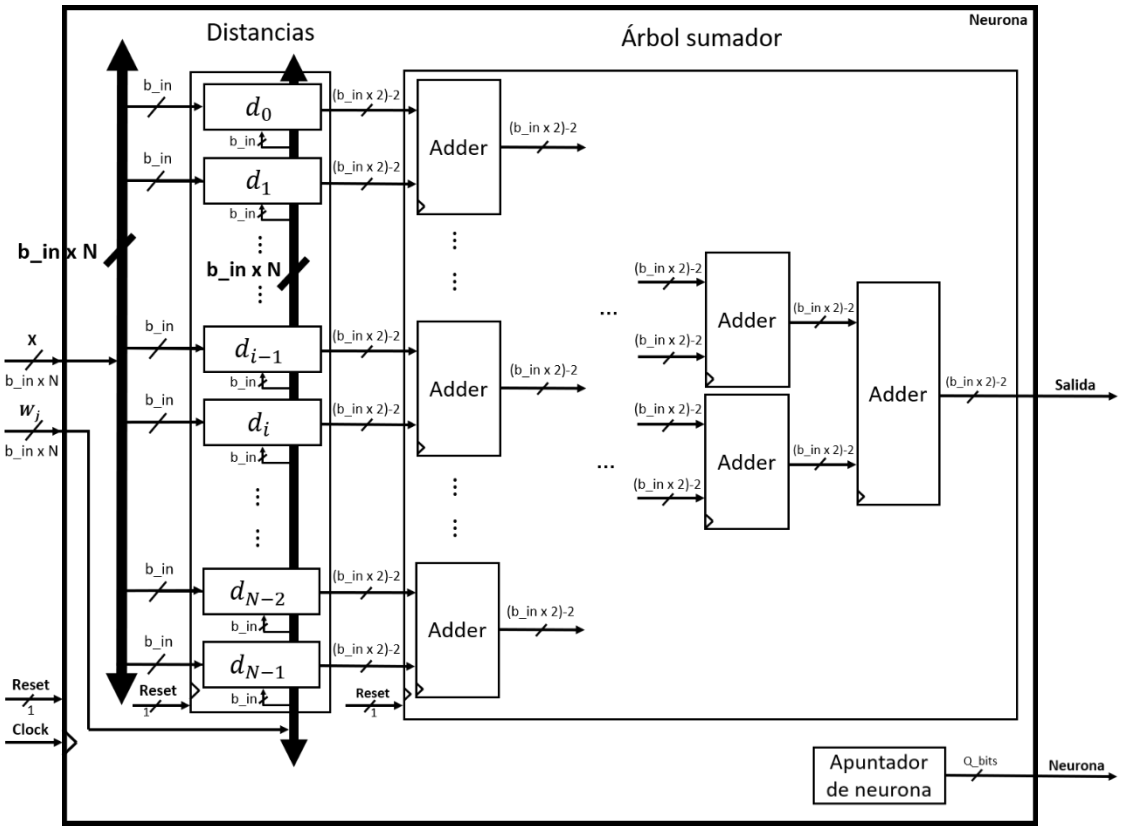

Figura 3.3: Arquitectura de una neurona.

Las neuronas están compuestas por dos componentes: “distance” (Figura 3.4) y “adder” (Figura 3.5). El primero calcula la diferencia entre una característica de la muestra de entrada y el peso asociado a dicha característica y lo eleva al cuadrado (Ecuación 3.2). El segundo componente suma las distancias obtenidas de la salida del componente “distance”, de dos en dos, de la misma manera que los comparadores.

$$d_i = \left(x_i - W_{ji}\right)^2 \tag{3.2}$$

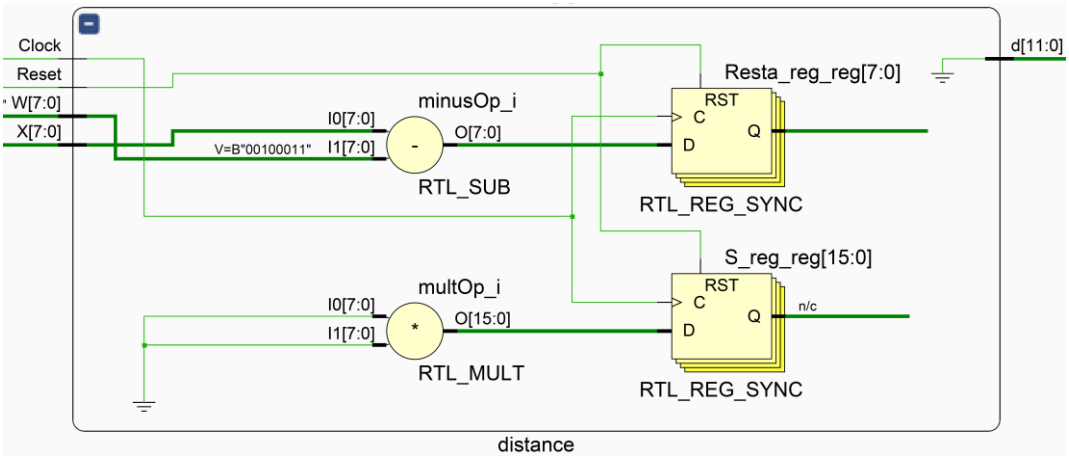

Figura 3.4: Arquitectura del componente “distance”.

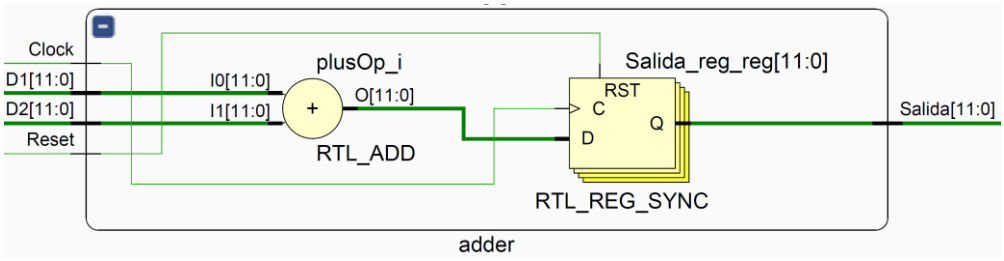

Figura 3.5: Arquitectura del componente “adder”.

La salida del componente neurona es la distancia euclídea, pero también se debe añadir un componente llamado apuntador de neurona que indique a que neurona pertenece la distancia calculada. De esta manera, se puede controlar cual es la neurona a la que pertenece la distancia calculada para, una vez obtenida la distancia más pequeña, poder acceder a la ROM interna encargada de guardar las agrupaciones a las que pertenecen las neuronas.

3.1.3. Árbol comparador

Los árboles comparadores (Figura 3.6), tienen como entradas las distancias de dos neuronas a la muestra de entrada y el número de neuronas al que pertenecen, y devuelven la distancia más pequeña y el número de neurona al que pertenece dicha distancia.

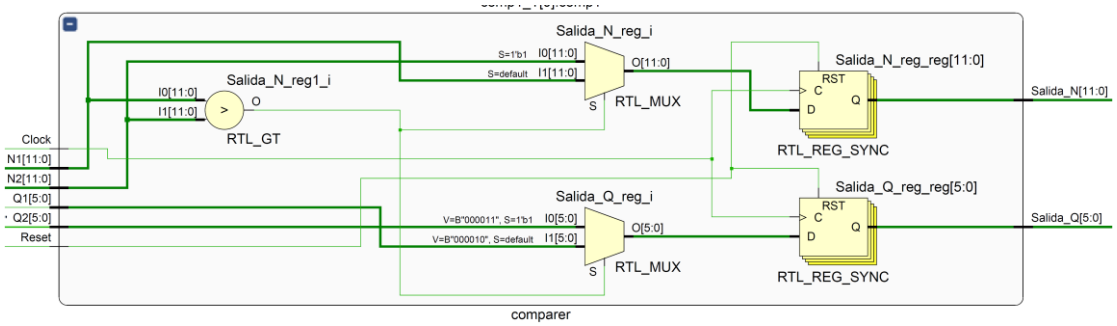

Figura 3.6: Arquitectura de un comparador.

3.1.4. ROM interna

Dentro de la arquitectura, se han utilizado dos ROMs. La primera se utiliza para almacenar los pesos de las neuronas, W, que se han obtenido después del entrenamiento. La segunda almacena las agrupaciones a las que pertenecen los pesos. Esta última se utiliza para, una vez obtenida la neurona ganadora, conocer a que agrupación pertenece dicha neurona (Figura 3.7).

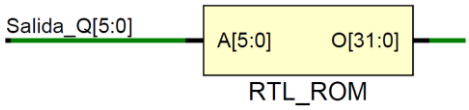

Figura 3.7: Arquitectura de la ROM que guarda las agrupaciones a las que pertenecen los pesos.

3.1.5. Controlador

Para saber cuándo se obtiene la salida de la arquitectura, se utiliza un controlador que indica el flanco de reloj para el cual la salida es válida. Para ello, se diseña un contador que calcula cuantos ciclos de reloj se necesitan hasta obtener la salida. Este cálculo se puede realizar ya que en el diseño, es decir, antes de implementar la arquitectura, se puede conocer el número de ciclos de reloj que se van a necesitar. El controlador se activa cuando recibe la señal *Enable* en alta y se resetea con la señal *Reset* en alta.

El tiempo de ejecución de la arquitectura tiene una parte fija en la que siempre tarda los mismos ciclos de reloj en completarse y otra parte que depende del número de características y del número de neuronas. Como se acaba de explicar, los árboles comparadores calculan que neurona tiene la menor distancia de dos en dos. Por consiguiente, para conocer cuántos ciclos de reloj tienen que transcurrir hasta poder obtener la salida de este componente, se debe calcular el logaritmo en base dos del número total de neuronas M aplicando la función techo (Ecuación 3.3).

$$t_{árbol} = \lceil \log_2 M \rceil = \min\{k \in \mathbb{Z} \mid (\log_2 M) \leq k\} \tag{3.3}$$

Al igual que el árbol comparador, el árbol sumador calcula las salidas de dos en dos. Por ello, el tiempo de ejecución se calcula como la Ecuación 3.3 pero utilizando N ya que depende del número de características. Por otra parte, la arquitectura tarda un número de ciclos de reloj fijos independientemente del número de neuronas o del número de características que conste. El número de ciclos de reloj fijos son tres: uno para los registros de entrada y dos para el

componente “distance” incluido dentro del componente neurona. Por lo tanto, el número de ciclos de reloj que transcurren desde que se recibe la señal hasta que se proporciona una salida válida se calcula mediante la Ecuación 3.4.

$$N° \; ciclos = 3 + \lceil \log_2 N \rceil + \lceil \log_2 M \rceil \tag{3.4}$$

3.2. Implementación

Para implementar el SOM, se utiliza la FPGA perteneciente a la familia Artix-7 y, en particular, la Nexys 4 DDR. La FPGA Artix-7 100T de 450MHz está compuesta por los siguientes recursos:

- 15.850 bloques lógicos, cada uno compuesto por cuatro LUTs de 6 entradas y 8 flip-flops.
- 4,860 Kbits de bloques de RAM rápida.
- 240 DSPs.
- Convertidor analógico-digital.

En el Capítulo 2, se explicó cómo es el entrenamiento del SOM y se vio un método de clasificación. Además, se probó la clasificación con la base de datos *Iris* y se obtuvieron los pesos y las agrupaciones a las que pertenecen las neuronas. Con estos datos, se ha implementado la red SOM en la FPGA para la clasificación de la base de datos *Iris*, introduciendo los pesos de las neuronas y las agrupaciones a las que pertenecen dentro de la ROM interna de la arquitectura.

La arquitectura que se ha implementado consta de 64 neuronas y de 4 características. El formato de datos que se ha escogido para la representación es el de coma fija. Los datos de entrada y los pesos de las neuronas se han representado con 8 bits (4 para la parte entera, 3 para la fraccional y 1 bit de signo).

Antes de la implementación, se ha calculado la frecuencia máxima a la que puede trabajar el diseño. Para ello, se ha implementado la arquitectura con un periodo de reloj de 10ns y se ha buscado, en el informe de entrenamiento, el valor de *Worst Negative Slack* (WNS). Este valor si es negativo, significa que el periodo que se ha utilizado es demasiado pequeño. En cambio, si es positivo, significa que el circuito ha funcionado correctamente. En este caso, es 1.88ns, por lo que el circuito ha funcionado correctamente. Para calcular la frecuencia máxima de trabajo se debe calcular la inversa de la diferencia entre el periodo de prueba y el valor de WNS (Ecuación 3.5).

$$f_{max} = \frac{1}{T_{prueba} - d_{\text{Slack}}} = \frac{1}{10ns - 1.877ns} = 123{,}1MHz \tag{3.5}$$

No conviene utilizar la frecuencia límite ya que puede dar problemas en la implementación. Por este motivo, se ha utilizado un periodo de reloj de 10ns. En la Tabla V se indican los recursos utilizados en la FPGA.

TABLA V: RECURSOS UTILIZADOS POR LA FPGA PARA LA IMPLEMENTACIÓN DE LA ARQUITECTURA DEL SOM PARA LA BASE DE DATOS *IRIS*

Recursos FPGA	Utilizados	Disponibles	% de recursos utilizados
Número de LUTs	5692	63400	8,98
Número de flip-flops	4240	126800	3,34

Para probar la arquitectura, se toma una nueva muestra *X* compuesta por 4 características, y se busca conocer a qué agrupación pertenece. En primer lugar, se introduce en la ROM los pesos de las neuronas y las agrupaciones a las que pertenecen. A continuación, se introduce en los registros de entrada la muestra *X* y se cambia el valor del Reset a '0' y el Enable a '1'. Por último, una vez que el controlador indica que la salida es válida, se observa que la muestra *X* pertenece a la agrupación 3 (*Virginica*) (Figura 3.8).

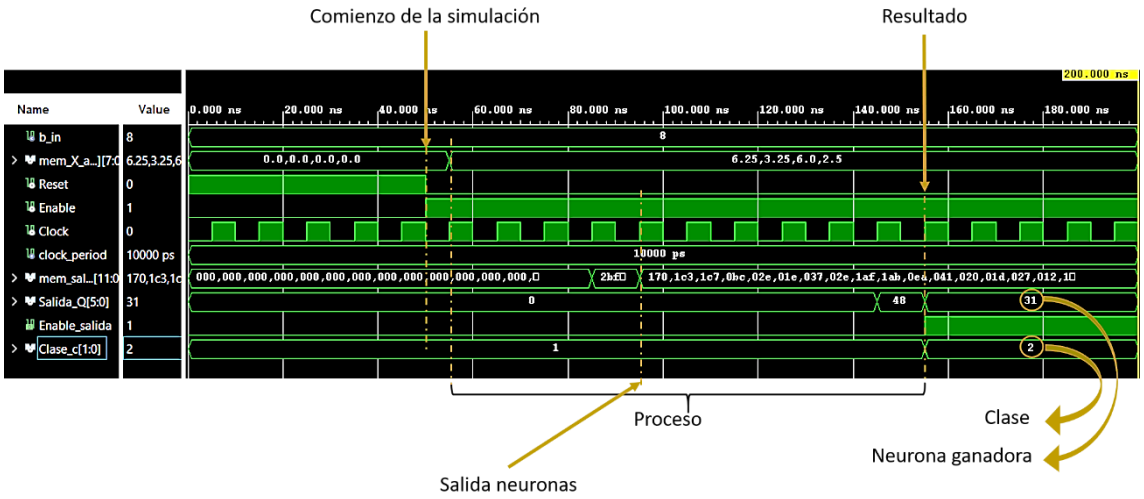

Figura 3.8: Simulación de una muestra para la base de datos *Iris*.

Utilizando MATLAB se ha comprobado que la neurona más cercana a la muestra coincide con la calculada por la FPGA. La duración desde que se ha introducido la muestra de entrada hasta que se ha obtenido los resultados ha sido de 11 ciclos de reloj divididos en: 1 para el registro de entrada, 4 para el componente neurona (2 por el componente "distance" y otros 2 para las salidas del componente "distance" con 3 componentes "adder") y 6 de los comparadores. El número de ciclos de reloj coincide con la Ecuación 3.4 propuesta,

$$N°\ ciclos\ Iris = 3 + \lceil \log_2 4 \rceil + \lceil \log_2 64 \rceil = 3 + 2 + 6 = 11 \qquad (3.6)$$

Capítulo 4

Clasificación de los estilos de conducción en relación al consumo

En el último capítulo se pone en práctica todo lo visto en los capítulos anteriores con un ejemplo. En particular se utiliza la base de datos llamada “UYANIK” obtenida recopilando señales de conducción de más de 100 conductores en un tramo de 25 km en la ciudad y alrededores de Estambul, Turquía [19]. El itinerario consta de tramos en autopista, en carretera y en ciudad y el vehículo que se ha utilizado ha sido un Renault Megane Sedan 1.5 dCi 100 CV (diésel).

A partir de la base de datos, se caracteriza el estilo de conducción para determinar cuáles son los conductores que consumen más combustible. Para ello, se proponen varios atributos que relacionen el estilo de conducción en relación al consumo de combustible y se clasifican. La clasificación se hace utilizando las redes SOM ya que proporcionan una evaluación de la red completamente paralelizable. Con ello, se consigue que la implementación sea en tiempo real utilizando la FPGA.

4.1. Obtención de los atributos

Para clasificar los estilos de conducción en relación al consumo de combustible, se van a seleccionar los datos obtenidos del vehículo con una fuerte correlación con el comportamiento de conducción. Basándonos en el estudio de correlación hecho en el artículo *"Impact of Driver Behavior on Fuel Consumption: Classification, Evaluation and Prediction Using Machine Learning"* [20], se determina la relevancia de cada parámetro para el consumo de combustible del vehículo. Los atributos con una fuerte correlación son: la velocidad, la aceleración positiva, la aceleración negativa y la varianza de velocidad. La aceleración positiva y negativa se tratan como parámetros diferentes porque sus efectos en la economía de combustible difieren. Por ejemplo, al calcular el consumo del combustible, si la aceleración es negativa, el consumo instantáneo de combustible es cero.

De la base de datos “UYANIK”, se han utilizado únicamente los datos obtenidos en autovía con el fin de que los atributos de los conductores no dependan tanto de la situación de la carretera, así como de los posibles obstáculos que se pudieran encontrar como, por ejemplo, los semáforos. Los datos se han obtenido con un muestreo de 32Hz, pero no se han utilizado directamente. Se han tomado los datos por ventanas, calculando la media de cada una y desplazándolas. Las ventanas se utilizan en el análisis y el procesamiento de señales con el fin de suavizar los datos, es decir, que los datos tomados no tengan cambios bruscos puntuales.

Las ventanas se han utilizado con un tamaño de 256 muestras y con un desplazamiento entre ventanas de 32 muestras. Haciendo esto se consigue suavizar y mejorar las muestras obtenidas de la base de datos.

i) Velocidad, aceleración positiva y negativa media

La velocidad y aceleración positiva y negativa media se define como,

$$\mu = \frac{1}{A}\sum_{a=1}^{a} a \tag{4.1}$$

siendo a una muestra de la velocidad o aceleración (+ ó -) muestreada por ventanas y A el número total de ventanas.

ii) Varianza de la velocidad

La varianza de la velocidad se define como,

$$\sigma^2 = \frac{1}{A}\sum_{a=1}^{A}(a-\mu)^2 \tag{4.2}$$

También se propuso utilizar los picos de aceleración como posible atributo para determinar la suavidad de la aceleración calculando el número total de picos de aceleración en la dirección del movimiento para los cuales se supere un cierto umbral. De esta manera, se podría determinar de qué forma el conductor comienza a acelerar. El problema que tiene este atributo es que al considerar solo autovía pierde efectividad ya que no se producen tantos cambios bruscos en aceleración como pudiera suceder en carretera.

Antes de entrenar la red con los atributos propuestos se deben normalizar los datos debido a que tienen diferentes rangos. Para ello, se cambian los valores de los atributos a una escala común [0, 1], sin distorsionar las diferencias en los rangos de valores utilizando la ecuación de escalado de características (Ecuación 4.3).

$$\boldsymbol{x_{i_{norm}}} = \frac{\boldsymbol{x_i} - min\ (\boldsymbol{x_i})}{max\ (\boldsymbol{x_i}) - min\ (\boldsymbol{x_i})} \tag{4.3}$$

4.2. Clasificación de los conductores según su consumo

Una vez obtenidos los parámetros para clasificar a los conductores según su consumo de combustible, se han entrenado utilizando los mapas autoorganizados (SOM). Para la clasificación, se dispone de 20 conductores extraídos de la base de datos "UYANIK". Utilizando la Ecuación 2.1, se recomienda que se utilicen 22 neuronas, es decir, una red de 5 x 5. A pesar de ello, se han utilizado 36 neuronas (6 x 6) ya que se ha comprobado que con un número pequeño de muestras de entrenamiento es recomendable utilizar más neuronas con el fin de que se distingan los grupos en la representación de la matriz –U (Figura 4.1).

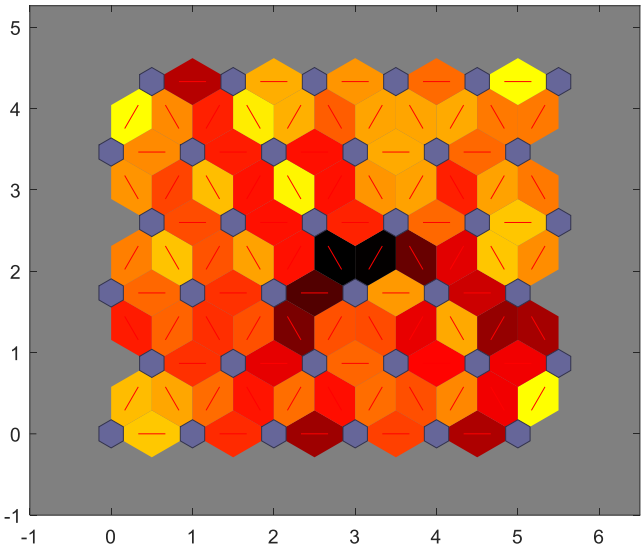

Figura 4.1: Representación de la matriz-U para la base de datos “UYANIK”. Los colores en las regiones que contienen las líneas rojas indican las distancias entre las neuronas. Los colores más oscuros representan distancias mayores y los colores más claros representan distancias menores.

Como se observa en la Figura 4.1, hay una banda de segmentos oscuros que limita la región inferior izquierda y otra banda un poco menos oscura (roja) que parte desde el centro hasta la región superior izquierda, lo que significa que hay tres agrupaciones que están claramente separadas.

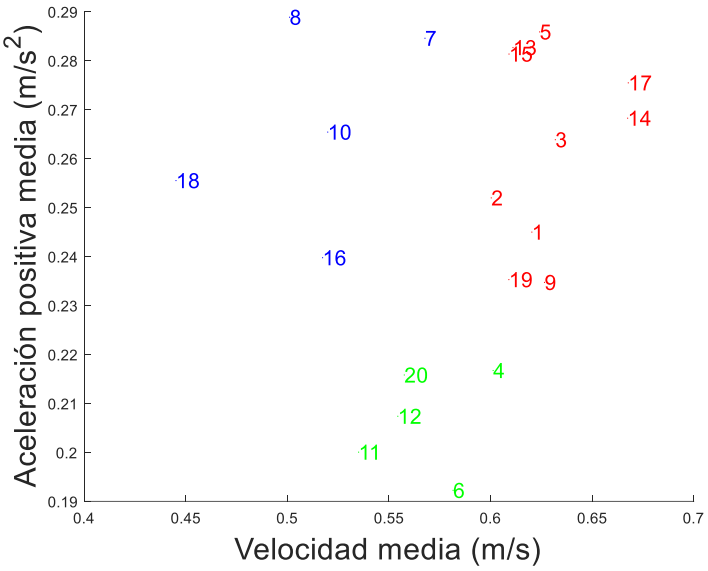

Figura 4.2: Representación la aceleración positiva media en función de la velocidad media utilizando el método de clasificación del SOM.

Para determinar a qué grupo pertenece cada conductor se ha representado en la Figura 4.2 la aceleración positiva media en función de la velocidad media. Las aceleraciones más altas y con mayor velocidad claramente pertenecen al grupo de conductores con mayor consumo. En cambio, para determinar cuál de las dos agrupaciones restantes tiene un mayor consumo nos hemos basado en el mismo artículo que hemos utilizado para determinar la correlación de los

atributos [20]. En consecuencia, se ha determinado que las aceleraciones más bajas pertenecen a los conductores con un menor consumo. Por lo tanto, se consigue poder clasificar a los conductores en tres agrupaciones: los que más consumen (rojo), los que tienen un consumo medio (azul) y los que menos consumen (verde).

En la Tabla VI, se puede observar los resultados obtenidos para cada conductor utilizando los atributos que se han descrito anteriormente antes de ser normalizados: Velocidad, aceleración positiva y negativa media y varianza de la velocidad.

TABLA VI: CÁLCULO DE LOS ATRIBUTOS PARA CADA CONDUCTOR, LAS NEURONAS MÁS CERCANAS A CADA ATRIBUTO Y LAS AGRUPACIONES A LAS QUE PERTENECEN

Conductor	Velocidad media (km/h)	Aceleración positiva media (m/s^2)	Aceleración negativa media (m/s^2)	Varianza velocidad $(km/h)^2$	Neurona más cercana	Agrupación
1	81,87	0,48	0,42	431,57	14	1
2	79,23	0,50	0,49	504,18	10	1
3	83,39	0,52	0,43	574,66	8	1
4	79,35	0,43	0,37	409,38	13	3
5	82,37	0,56	0,47	439,80	3	1
6	76,75	0,38	0,30	296,45	26	3
7	74,93	0,56	0,47	386,21	11	2
8	66,13	0,57	0,51	404,63	12	2
9	82,69	0,46	0,43	404,05	15	1
10	68,61	0,52	0,52	367,73	18	2
11	70,62	0,39	0,37	326,78	33	3
12	73,16	0,41	0,35	248,42	21	3
13	80,64	0,56	0,47	410,42	4	1
14	88,09	0,53	0,47	559,56	1	1
15	80,38	0,55	0,48	543,46	4	1
16	68,28	0,47	0,37	1083,37	34	2
17	88,13	0,54	0,49	592,82	1	1
18	58,76	0,50	0,61	1476,19	24	2
19	80,38	0,46	0,44	499,14	16	1
20	73,58	0,42	0,29	154,35	31	3

4.3. Arquitectura del SOM para la clasificación de los conductores según su consumo

Por último, se ha implementado la arquitectura del SOM para la clasificación de los conductores con 36 neuronas. Como la implementación que se ha hecho en la FPGA es completamente modular y los datos se procesan en paralelo, es sencillo escalarla para este caso concreto.

El formato de datos que se ha escogido para la representación es el de coma fija fraccional en complemento a dos. Los datos de entrada y los pesos de las neuronas se han representado con 9 bits (8 para la fraccional y 1 bit de signo).

Al igual que para el diseño de la base de datos *Iris*, se ha implementado la arquitectura con un periodo de reloj de 10ns. El valor de WNS (*Worst Negative Slack*) ha sido de 1,7ns por lo que la implementación es correcta. Para este caso la frecuencia máxima de trabajo es 120,5MHz. En la Tabla VII se indican los recursos utilizados en la FPGA.

TABLA VII: RECURSOS UTILIZADOS POR LA FPGA PARA LA IMPLEMENTACIÓN DE LA ARQUITECTURA DEL SOM PARA LA CLASIFICACIÓN DE LOS CONDUCTORES

Recursos FPGA	Utilizados	Disponibles	% de recursos utilizados
Número de LUTs	5722	63400	9,03
Número de flip-flops	3349	126800	2,64

Se ha realizado una simulación para comprobar el correcto funcionamiento y se ha comparado con el que se ha calculado en MATLAB (ver Tabla VI), en este caso, se ha utilizado como ejemplo el conductor número 19. Hay que recordar que una vez entrenado el SOM, las muestras, en este caso los conductores, que se han utilizado para el entrenamiento se pueden clasificar calculando cual es la neurona que tiene la menor distancia con ellos y ver a que agrupación pertenecen. Pero también se pueden introducir los datos de otros conductores que no se han utilizado en el entrenamiento y que se deseen clasificar.

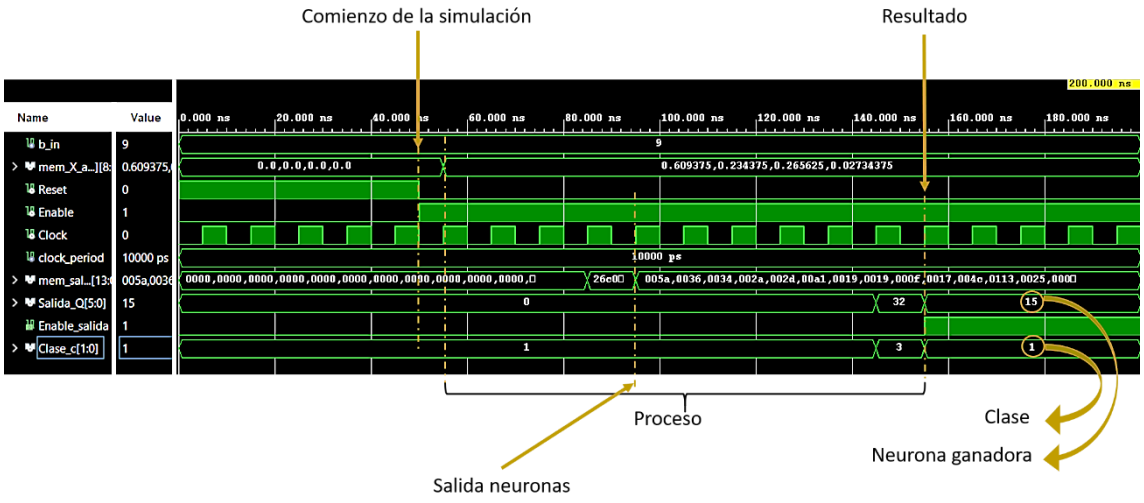

Figura 4.3: Clasificación de un conductor para la base de datos *UYANIK*.

Como se puede ver en la Figura 4.3, la neurona que tiene la menor distancia con la muestra que se ha introducido es la neurona 16 y pertenece a la agrupación 1 tal y como se esperaba. Para obtener la respuesta se han necesitado 11 ciclos de reloj divididos en: 1 para el registro de entrada, 4 para el componente neurona (2 por el componente "distance" y otros 2 para las salidas del componente "distance" con 3 componentes "adder") y 6 de los comparadores. El número de ciclos de reloj coincide con la Ecuación 3.4 propuesta,

$$N°\ ciclos\ conductores = 3 + \lceil \log_2 4 \rceil + \lceil \log_2 36 \rceil = 3 + 2 + 6 = 11 \qquad (4.4)$$

Como se ha podido comprobar la implementación en la FPGA es sencilla y escalable, es decir, es fácil adaptarla si se hace uso de más o menos parámetros o neuronas. Además, la entidad de control es simple ya que solo se utiliza para especificar cuándo la salida es válida.

Conclusiones

En este trabajo se han comparado los métodos de aprendizaje no supervisado más utilizados en la actualidad. Se ha comprobado que la agrupación jerárquica es el método más sencillo de implementar, pero también es el que mayores problemas puede dar ya que no tiene en cuenta a todo el conjunto de datos a la hora de clasificar. Además, si se necesita clasificar nuevas muestras se tiene que hacer uso de algún método supervisado como, por ejemplo, las redes neuronales artificiales (RNA). Con el método k-means se puede escoger el número de agrupaciones que se quieren obtener, aunque en ocasiones puede ser difícil de predecir. Como mejora a este algoritmo existe el modelo de mezclas gaussianas ya que no solo predice a que agrupaciones pertenecen las muestras de entrenamiento, sino que además proporciona las probabilidades de que una muestra dada pertenezca a cada una de las agrupaciones posibles.

Los mapas autoorganizados (SOM) son un tipo de red neuronal artificial que se entrena utilizando técnicas de aprendizaje no supervisado. Con ellos se consigue obtener una imagen visual sobre cómo se han clasificado los datos sin depender del número de características que se haya empleado. El SOM preserva la topología de las muestras de entrenamiento porque las distancias en el espacio bidimensional reflejan las del espacio de alta dimensión. Implementando los SOM con la base de datos *Iris* se han obtenido muy buenos resultados siendo este el método que mejor los ha clasificado. Teniendo en cuenta estos resultados, se han utilizado los SOM como método de clasificación para este trabajo.

Se ha diseñado una arquitectura para el procesado digital de la red SOM en FPGA. Para aprovechar las ventajas de la FPGA, la arquitectura diseñada es paralela para que la respuesta sea de baja latencia. Además, mediante un leguaje de descripción de hardware, VHDL, se consigue implementar el diseño para evaluar cualquier otra base de datos ya que la arquitectura es escalable.

Para demostrar las ventajas de implementar los mapas autoorganizados para la evaluación en tiempo real, se ha realizado una clasificación de conductores en base al consumo de combustible que realizan. Los atributos que se han utilizado para entrenar la red han sido: la velocidad media, la varianza de la velocidad, la aceleración positiva media y la aceleración negativa media. Con ellos, se consigue detectar cuales son los conductores que producen un mayor consumo y cuál es el factor que hace que el consumo se eleve o se reduzca. Además, se puede detectar en qué momento del recorrido el conductor está incrementando el gasto de combustible y se le podría avisar para que mejore la conducción. De esta manera, se consigue ahorrar combustible solamente variando el estilo de conducción.

Los atributos que se han utilizado para detectar a los conductores que tienen un mayor consumo están ligados con las técnicas de conducción eficiente (eco-driving) [21]. Son una serie de pautas de conducción con las que se consigue reducir el consumo de combustible y las emisiones asociadas. Por ello, además de reducir el consumo de combustible también se logra reducir las emisiones de gases de efecto invernadero ya que está completamente ligado al consumo de combustible. Este hecho es realmente importante ya que, según la Agencia Europea de Medio Ambiente, el 93% de las emisiones del sector transporte provienen del

transporte por carretera, que a su vez suponen un 20,4% del total de emisiones [22]. Por ello, existe una gran necesidad de reducir las emisiones y cualquier mejora supone un gran avance a la hora de reducir la contaminación.

La implementación de la evaluación de los conductores se ha realizado mediante la FPGA Nexys 4 DDR Artix-7 ya que se ha comprobado que tiene recursos suficientes para poder realizar los cálculos. La frecuencia de operación máxima ha sido de 120,5 MHz con una latencia de 83ns, un tiempo de respuesta válido para la implementación en tiempo real en muchos campos de aplicación. El porcentaje de recursos utilizados por la FPGA deja margen para poder implementar un diseño con un número mayor de neuronas o de características.

Como líneas de investigación futuras se podría buscar algún otro atributo para la clasificación. Dicho atributo debería ser coherente con los cuatro que se han utilizado en este trabajo. Otra línea de investigación interesante seria implementar junto a la FPGA un microprocesador que se encargue de recibir los datos del bus del automóvil y calcule los atributos para la FPGA. De esta manera, cada cierto tiempo podría calcular la FPGA si el consumo de combustible está siendo elevado y mediante una pantalla avisar al conductor de dicho aumento, indicándole que aspecto de la conducción debe mejorar.

Bibliografía

[1] R. A. Fisher, The use of multiple measurements in taxonomic problems, Annals of Eugenics 7, 1936, pp. 179-188.

[2] Y. Hadad, «30 amazing applications of Deep Learning,» 2017. [En línea]. Available: http://www.yaronhadad.com/deep-learning-most-amazing-applications/.

[3] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» Bulletin of Mathematical Biophysics 5, 1943.

[4] F. Rosenblatt, Principles of Neurodynamics, Spartan Books, 1962.

[5] K. I. Funahaski, «On the approximate realization of continuous mappings by neural networks,» *Neural networks 2,* pp. 183-192, 1989.

[6] P. J. Werboz, «Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences,» Tesis Doctoral, Universidad de Harvard, 1974.

[7] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by backpropagation errors,» *Nature,* nº 323, pp. 533-536, 1986.

[8] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, MIT Press, 2017.

[9] G. N. Lance y W. T. Williams, «A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems,» *The Computer Journal,* vol. 9, nº 4, pp. 373-380, 1967.

[10] S. Lloyd, «Least square quantization in PCM,» Bell Telephone Laboratories Paper, 1957.

[11] E. W. Forgy, «Cluster analysis of multivariate data: efficiency versus interpretability of classifications,» pp. 768-769, 1965.

[12] P. J. Rousseeuw, «Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis,» *Journal of Computational and Applied Mathematics,* vol. 20, pp. 53-65, 1987.

[13] R. O. Duda y P. E. Hart, Pattern Classification and Scene Analysis, A Wiley-Interscience publication, 1973.

[14] T. Bayes y R. Price, «An Essay towards solving a Problem in the Doctrine of Chances,» *Philosophical Transactions of the Royal Society of London,* vol. 53, pp. 370-418, 1763.

[15] T. Kohonen, «Self-Organized Formation of Topologically Correct Feature Maps,» *Biological Cybernetics,* vol. 43, pp. 59-69, 1982.

[16] J. Tian, M. H. Azarian y M. Pecht, «Anomaly Detection Using Self-Organizing Maps-Based K-Nearest Neighbor Algorithm,» de *European conference of prognostics and health management society*, 2014.

[17] A. Ultsch y H. P. Siemon, «Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis,» de *Proceedings of the International Neural Network Conference*, Paris, Francia, 1990.

[18] E. Alhoniemi, J. Himberg, J. Parhankangas y J. Vesanto, «SOM Toolbox,» Laboratory of Information and Computer Science, Univ. of Technology Helsinki, 2000. [En línea]. Available: http://www.cis.hut.fi/projects/somtoolbox/.

[19] H. Abut, H. Erdogan, A. Ercil, B. Çürüklü, H. Can, F. Taş, A. Ö. Argunşah, S. Cosar, B. Akan, H. Karabalkan, E. Çökelek, R. Fıçıcı, V. Sezer, F. S. Daniş, M. Karaca y M. Abbak, «Real-world data collection with "UYANIK",» de *In-Vehicle Corpus and Signal Processing for Driver Behavior*, 2009.

[20] P. Ping, W. Qin, Y. Xu, C. Miyajima y K. Takeda, «Impact of Driver Behavior on Fuel Consumption: Classification, Evaluation and Prediction Using Machine Learning,» *IEEE Access*, vol. 7, pp. 78515-78532, 2019.

[21] M. A. S. Kamal y T. Kawabe, «Eco-driving using real-time optimization,» de *2015 European Control Conference (ECC)*, Linz, 2015.

[22] Agencia Europea de Medio Ambiente, «Air quality in Europe,» EEA Report, 2013.