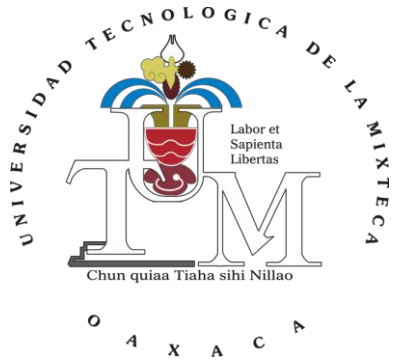

UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"DISEÑO Y MODELADO DE UNA ARQUITECTURA HARDWARE DE UN CLASIFICADOR BASADO EN MÁQUINAS DE SOPORTE VECTORIAL"

TESIS PARA OBTENER EL GRADO DE:

MAESTRO EN ELECTRÓNICA, OPCIÓN: SISTEMAS INTELIGENTES APLICADOS

PRESENTA:
ING. ALDO AVILA CASTRO

DIRECTOR:
DR. ENRIQUE GUZMÁN RAMÍREZ

CODIRECTOR:
DR. ANTONIO ORANTES MOLINA

H. CIUDAD DE HUAJUAPAN DE LEÓN, OAXACA; OCTUBRE DE 2022

Dedicatoria

A mis padres, por su amor y apoyo incondicional.

A toda mi familia, por su comprensión y el cariño que me brindan.

Aldo

Agradecimientos

A mi madre, Dolores Hortensia Castro Rodríguez, por su comprensión cariño y consejos que me brinda.

A mi padre, Alonso Avila Castillo, por su apoyo y comprensión.

A familia, por estar siempre presentes, apoyándome.

A mi novia, Yubesny Iriani Ventura Sigüenza, por su amor, compañía y consejos que me brinda.

A los amigos, Omar Castro Heredia, Rubén Heredia Barba y Luisa Margarita Chávez Reyes, por la colaboración en los trabajos y las risas que nos hicieron sobrellevar la maestría.

A mi director de tesis, Dr. Enrique Guzmán Ramírez por su amistad, conocimiento y guía durante todo el trabajo de tesis.

A mi amigo, Heriberto I. Hernández Martínez, por todos los consejos que aporta para mi crecimiento personal, profesional y cultural.

A Roberto Barragán, por su ayuda durante el primer semestre de maestría.

A mis sinodales, Dr. Oscar David Ramírez Cárdenas, MTCA. Moisés Emmanuel Ramírez Guzmán, MTCA. Erik Germán Ramos Pérez, Dr. Rosebet Miranda Luna, por sus observaciones y consejos.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la beca otorgada, a el Programa para el Desarrollo Profesional Docente (PRODEP) por el apoyo económico al proyecto y a la Universidad Tecnológica de la Mixteca (UTM).

Aldo

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

Aprendizaje automático (ML, *Machine Learning*) es un término que en la actualidad se escucha con mucha frecuencia debido a su cercanía con tecnologías como Inteligencia Artificial (AI, *Artificial Intelligence*) y *big data,* entre otros. Tiene como objetivo desarrollar técnicas que proporcionen a los sistemas la capacidad de aprender y mejorar automáticamente, por medio de la generación de modelos, permitiendo la predicción de comportamientos a partir de información en forma de datos.

La revolución digital ha facilitado la captura de datos y disminuido los costos de su almacenamiento, como resultado, es posible almacenar continuamente enormes cantidades de información, con diferentes tipos de datos, en diversos medios para un posterior análisis o uso. Lo anterior ha provocado que los métodos habituales empleados para su análisis sean desactualizados al ser empleados en grandes conjuntos de datos. Debido a esto, se requieren métodos semiautomáticos de análisis cuyo objetivo principal es entender y analizar enormes cantidades de datos.

La información almacenada en bases de datos crece significativamente día con día. Debido al incremento en la cantidad de datos y sus características asociadas se necesitan técnicas de exploración versátiles, robustas y eficientes que puedan hacer frente a los cambios actuales. Estas técnicas pueden pertenecer al área del aprendizaje automático, tomando en

cuenta los diferentes tipos de aprendizaje existentes, como pueden ser supervisado, no supervisado, semi-supervisado o aprendizaje por refuerzo. La clasificación de datos se describe como una subcategoría del aprendizaje supervisado y un modelo de clasificación puede servir como una herramienta para distinguir entre objetos de diferente clase. Los procesos de clasificación constan de dos fases, entrenamiento y prueba. En la fase de entrenamiento, se utiliza un conjunto de datos inicial para decidir qué parámetros deberán ser ponderados y combinados con el objetivo de separar varias clases de objetos. El aprendizaje intenta predecir una representación óptima a partir del conjunto de datos cuyas etiquetas son conocidas. En la fase de prueba, se aplican los pesos determinados en la fase de entrenamiento a un conjunto de objetos (conjunto de prueba) cuyas etiquetas de clase son desconocidas, con el objetivo de determinar a qué clase pertenecen.

Existen varias técnicas de clasificación convencionales, por ejemplo: reglas basadas en clasificadores de vecinos más cercanos (*nearest neighbor*), clasificadores Bayesianos, redes neuronales artificiales, árboles de decisión y máquinas de soporte vectorial (SVM, *Support Vector Machine*), por nombrar algunas. De las técnicas antes mencionadas, las SVM son unas de las técnicas más conocidas para optimizar la solución esperada. Se ha mostrado que las SVM son superiores a otros métodos de aprendizaje supervisado [1], [2], [3], [4]. Debido a sus fundamentos teóricos y su capacidad de generalización, en los últimos años, las SVM han llegado a ser uno de los métodos de clasificación más utilizados.

Una ventaja que sobresale de las SVM radica en el hecho de que éstas obtienen un subconjunto de vectores de soporte durante la fase de aprendizaje, que a menudo es sólo una pequeña parte del conjunto de datos original. Este conjunto de vectores de soporte representa una tarea de clasificación dada y está formado por un conjunto compacto de datos.

Con la finalidad de aprovechar las ventajas ofrecidas por una SVM y aplicarlas en situaciones que requieran trabajar en tiempo real y operar en forma autónoma, la presente tesis plantea el diseño y modelado de una arquitectura hardware de un clasificador basado en SVM y su implementación en un arreglo de compuertas programable en el campo (FPGA, *Field Programmable Gate Array*). Con la obtención de una arquitectura hardware eficiente se pretende proponer técnicas en el modelado de hardware que permitan aprovechar las estructuras paralelas inherentes y la concurrencia potencial que sustenta la operación de una

SVM. Un FPGA es un dispositivo que, debido a las prestaciones que ofrece, favorece la evaluación concurrente de procesos, resultando ser una tecnología ideal para aplicaciones que requieran operar a altas velocidades de procesamiento, en tiempo real y en forma autónoma.

1.1. Planteamiento del Problema

Uno de los objetivos del ML es dotar a un sistema con habilidades que le permitan percibir, aprender e interactuar con su entorno. El reconocimiento de objetos es una tarea básica para que un sistema de este tipo pueda dar cumplimiento a una de sus actividades principales, clasificar objetos de interés.

La tarea de clasificación consiste en construir un mapa de relaciones entre el espacio de características y un conjunto de las clases definidas, de modo que sea capaz de identificar a qué clase corresponde un patrón de entrada representado por un vector de características. Este clasificador debe ser capaz de resolver diversos problemas, como la dimensionalidad del espacio de características, el número de ejemplos presentes en el conjunto de entrenamiento, construir un sistema que presente alta eficiencia al discriminar clases linealmente no separables y cumplir los requerimientos velocidad-rendimiento-recursos del sistema, entre otros.

En este sentido, las SVM representan un método de aprendizaje automático supervisado ampliamente utilizado en procesos eficientes de clasificación. Las SVM han demostrado una alta precisión de clasificación en numerosas aplicaciones, como reconocimiento de voz, detección de objetos, clasificación de imágenes, bioinformática y diagnóstico médico, entre otras [5]. Incluso, las tasas de precisión de clasificación presentadas por las SVM superan a otros algoritmos de clasificación populares en numerosos casos y aplicaciones [6], [7], [8], [9].

El problema que este trabajo de investigación enfrenta es diseñar un sistema que permita el uso de un clasificador basado en SVM en un contexto práctico, ofreciendo una implementación que funcione en tiempo real y pueda ser utilizado en aplicaciones autónomas, impulsada por tecnologías que favorezcan la ejecución concurrente de procesos y, por lo tanto, adecuada para aplicaciones del mundo real. El desarrollo de un sistema que

cumpla con las características mencionadas y sea capaz de adecuarse a diversas aplicaciones cotidianas, es un trabajo en demasía desafiante debido a las limitaciones críticas de rendimiento, área, potencia y costo requeridas para desarrollar un sistema integrado como el descrito.

1.2. Justificación

Los sistemas que realizan tareas de clasificación y que requieren operar en forma autónoma y en tiempo real son cada vez más requeridos en aplicaciones cotidianas y especializadas como las mencionadas en el apartado anterior.

Por tal motivo, el presente trabajo de investigación tiene por objetivo diseñar y modelar una arquitectura de una SVM que explote las estructuras paralelas inherentes y la concurrencia potencial que sustenta la operación matemática de este modelo, y que pueda ser implementada en sistemas autónomos que requieran operar en tiempo real.

Los procesadores en general son sistemas que basan su operación en un paradigma secuencial, es decir, su funcionamiento es regido por un conjunto de instrucciones que son ejecutadas en un determinado orden. Estos dispositivos son descartados para ser usados en el presente trabajo debido a que su forma de operar implica grandes tiempos de ejecución y que no favorecen el explotar el paralelismo existente en los algoritmos.

Por su parte, los GPUs resultan otra alternativa, ya que al poseer una gran cantidad de unidades de procesamiento pueden realizar tareas con un alto grado de concurrencia y ofrecer altas velocidades de procesamiento. Sin embargo, cuentan con una arquitectura cerrada y desde el punto de vista de manejar un arreglo masivo de unidades de procesamiento, no permite el modelado de hardware específico.

Para dar solución a la problemática planteada, esta tesis propone utilizar un FPGA como elemento central de procesamiento. Un FPGA resulta ser la opción más equilibrada debido a las siguientes razones: a) la característica más importante de un FPGA es la capacidad de realizar tareas en forma concurrente, permitiendo explotar el paralelismo existente en los algoritmos, lo que genera altas velocidades de procesamiento; b) la arquitectura de un FPGA brinda una gran flexibilidad de diseñar, lo que permite modelar fácilmente una arquitectura para una tarea específica; c) el costo de estos dispositivos y de

las herramientas de diseño resultan bastante accesibles para el ámbito académico; d) las herramientas de diseño permiten modelar, implementar y probar un sistema en lapsos de tiempo relativamente cortos; y e) son fácilmente adaptables para trabajar en un sistema autónomo, debido a su moderado consumo de energía y su portabilidad.

1.3. Hipótesis

El modelado de una arquitectura hardware de una SVM en lógica reconfigurable puede operar a altas velocidades de procesamiento y generar un alto desempeño en su tarea de clasificación considerando aspectos concurrentes y resultados en tiempo real.

1.4. Objetivos

1.4.1. Objetivo principal

Diseñar y modelar una arquitectura hardware de un clasificador basado en máquinas de soporte vectorial e implementación en un FPGA.

1.4.2. Objetivos específicos.

Diseñar un modelo conceptual del SVM utilizando un lenguaje de alto nivel.

Implementar el SVM sobre lógica reconfigurable, tomando en cuenta el paralelismo inherente de este modelo

1.5. Metas

Las metas por cumplir son las siguientes:

- Modelar de manera eficiente la SVM a utilizar en el presente estudio, identificando los elementos y operaciones básicas.
- Realizar un estudio de técnicas utilizadas en cada elemento e implementarlo en un FPGA.

1.6. Delimitación de la propuesta

El trabajo propuesto en esta tesis se delimita de la siguiente forma:

- Únicamente se considera el diseño, modelado e implementación de una SVM binaria.
- El modelo SVM generado en esta tesis únicamente cubrirá la operación de clasificación.
- El entrenamiento del algoritmo será realizado en el modelado conceptual.

1.7. Descripción de la Solución Propuesta

El alcance de este trabajo abarca cuatro áreas temáticas: a) modelado conceptual, b) diseño e implementación de arquitecturas hardware, c) procesamiento digital de señales y d) aprendizaje automático. Estas áreas se relacionan con la finalidad de generar un producto tangible representado por una arquitectura hardware de una SVM (véase Figura 1.1).

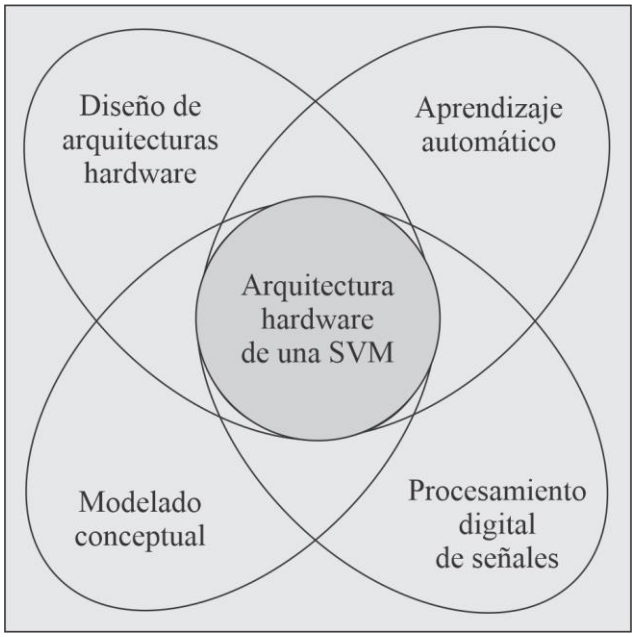

Figura 1.1. Diagrama de Venn que ilustra la relación de las áreas temáticas utilizadas para generar las contribuciones de la investigación en la tesis.

La Figura 1.2 muestra el diagrama de bloques del sistema construido y que funciona como plataforma de desarrollo en la consecución del objetivo de este trabajo de investigación. Este sistema está formado por un subsistema basado en FPGA, cuya función es albergar la arquitectura hardware del clasificador basado en SVM y una interfaz de usuario que permite al usuario interactuar con el subsistema mencionado.

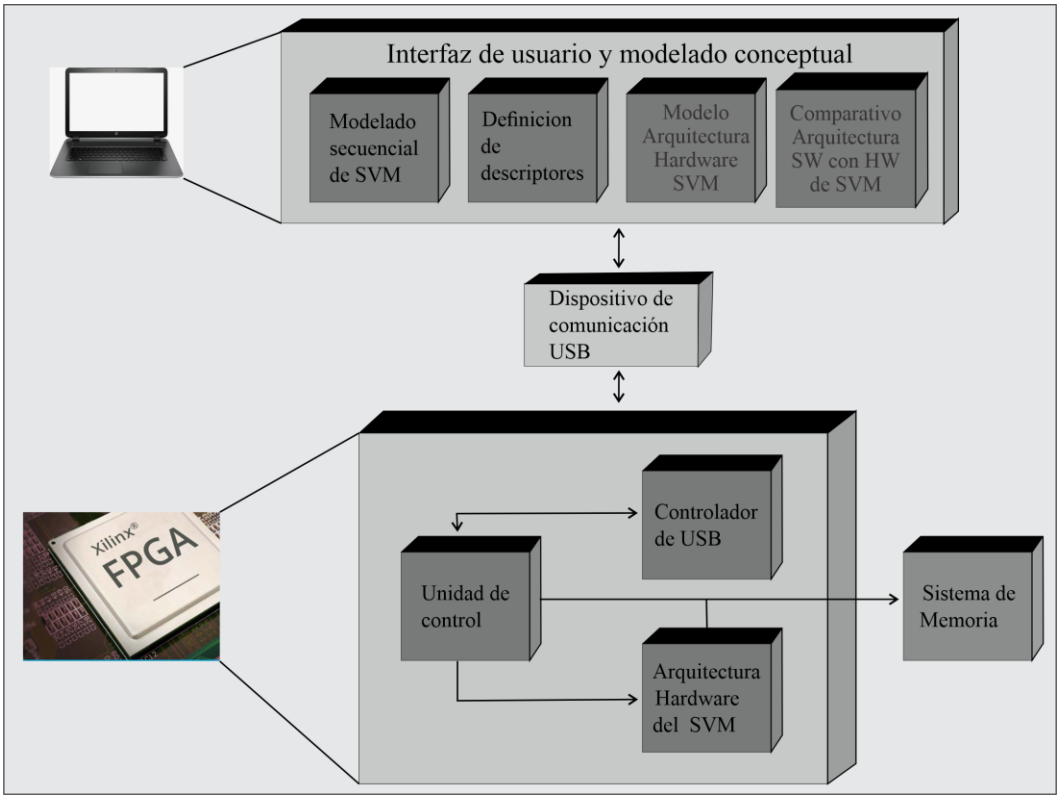

Figura 1.2. Diagrama de bloques plataforma de desarrollo usada en el proyecto.

El subsistema basado en FPGA está integrado por:

- El elemento central de procesamiento es la parte medular de este subsistema, se implementa mediante la plataforma de diseño Nexys 3 de la compañía Digilent Inc., diseñada con base en el FPGA Spartan-6 LX16 de la compañía Xilinx. Realiza las funciones de administrar los recursos del subsistema mencionado y de realizar el procesamiento de los datos en cuestión, es decir, albergar la arquitectura hardware de la SVM.
- El controlador USB-UART tiene por función permitir el intercambio de información entre el subsistema basado en FPGA y la interfaz de usuario vía el protocolo UART. Consiste en el dispositivo integrado a la tarjeta Nexys 3, el cual es un módulo de interfaz USB-UART basado en el FT232RQ de la compañía FTDI.
- El sistema de memoria se utiliza como una memoria temporal de almacenamiento (buffer) donde el subsistema basado en FPGA puede respaldar información como los datos que debe procesar o los resultados obtenidos del procesamiento. Este

sistema está compuesto por la RAM distribuida incorporado en la tarjeta Nexys 3.

- La interfaz de usuario tiene la finalidad de ofrecer al usuario una forma eficiente de acceder a todos los recursos ofrecidos por el subsistema basado en FPGA de una manera conveniente y cómoda. La interfaz de usuario es una GUI (*Graphical User Interface*) que se ejecuta en una computadora personal y que tiene por función dar apoyo en la evaluación del desempeño de la arquitectura hardware de la SVM diseñada. Para el desarrollo de la GUI se utilizó el entorno de desarrollo C++Builder.

La Figura 1.3 muestra la metodología de desarrollo de este trabajo. A continuación, se describen las fases que la conforman:

- Fase 1. Análisis, diseño y modelado conceptual de una SVM. En esta fase de la metodología, un modelado secuencial de una SVM se desarrolló en el lenguaje de alto nivel C++. La finalidad de esta fase es identificar las etapas y/u operaciones básicas que constituyen a una SVM, lo cual facilita la visualización de posibles características presentes en el algoritmo que permitan aplicar técnicas apropiadas en el diseño y modelado una arquitectura hardware eficiente y su implementación en lógica reconfigurable.
- Fase 2. Diseño e implementación de una interfaz de usuario. Con la finalidad de tener una forma eficiente de comunicación con la arquitectura hardware de la SVM diseñada, en esta fase se diseñó una GUI, utilizando el entorno de desarrollo C++Builder, la cual cumple con las siguientes funciones:
  - Tener acceso al modelado conceptual de la SVM modelada en la fase anterior.
  - Establecer una comunicación con la arquitectura hardware de la SVM diseñada mediante la definición de un protocolo de comunicación entre la GUI y el sistema implementado en el FPGA que facilite el intercambio de información entre ambos elementos.
  - Mostrar al usuario los resultados obtenidos tanto por el modelado conceptual de la SVM como por su arquitectura hardware.

  - Ofrecer métodos que faciliten la evaluación de la arquitectura hardware de la SVM diseñada.
- Fase 3. Diseño de la arquitectura hardware. En esta fase se diseña y modela la arquitectura hardware de la SVM explotando las características detectadas en la Fase 1. Para el diseño de la arquitectura se utiliza la metodología descendente (*top-down*), la cual es una metodología apropiada para diseños basados en FPGA. El modelado de la arquitectura obtenida se realiza en un lenguaje descriptor de hardware (HDL, *Hardware Descriptor Language*) y elementos pre-optimizados que ofrece el FPGA en uso. Finalmente, la implementación se realiza en un FPGA Spartan 6 de la compañía Xilinx. La herramienta que brindó soporte a las actividades de esta fase fue ISE Foundation (*Integrated Synthesis Environment*), también de la compañía Xilinx.
- Fase 4. Pruebas del sistema y análisis de resultados. Esta fase cumple principalmente con dos actividades:
  - Establece métodos que permite evaluar la arquitectura hardware de la SVM diseñada, considerando parámetros como velocidad de procesamiento y recursos utilizados.
  - Compara el desempeño que se obtuvo por el modelado conceptual de la SVM.

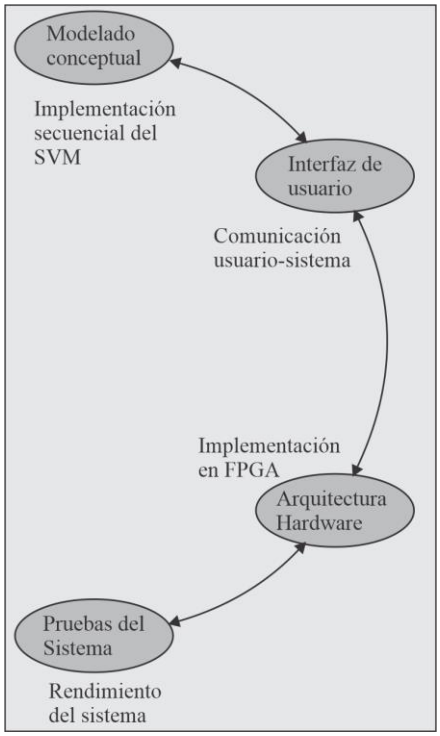

Figura 1.3. Metodología utilizada en el desarrollo del presente trabajo de investigación.

Capítulo 2. Marco Teórico

El presente capítulo contiene una breve discusión referente al ML, subcampo de las ciencias al que pertenece el modelo sobre el cual se centra este trabajo de investigación, las SVM. Además, incluye una discusión sobre las tecnologías utilizadas en este trabajo y el estado del arte referente al tema que esta tesis cultiva.

2.1. Aprendizaje Automático

La historia de la Inteligencia Artificial (AI) empieza en el año 1943, cuando Warren McCulloch y Walter Pitts presentaron el primer modelo de red neuronal [10]. El siguiente trabajo notable en el desarrollo de la AI fue presentado en el año 1950 por Alan Turing. Este científico formuló la famosa pregunta ¿pueden pensar las máquinas?, también presentó un tipo B de redes neuronales y definió el concepto de prueba de AI [11].

En 1956, John McCarthy, Marvin Minsky y Nathan Rochester de IBM junto a Claude Shannon organizaron la primera conferencia de verano de AI en la universidad de Dartmouth en Estados Unidos [12]. En esta conferencia, se utilizó por primera vez el término de AI. En el mismo año, surgió el término ciencia cognitiva durante un Simposio de Ciencias de la Información en el Instituto Tecnológico de Míchigan.

En 1958, Frank Rosenblatt presenta el modelo neuronal llamado perceptrón. El mismo año, Oliver Selfridge propone el uso de las computadoras para el reconocimiento de patrones. En 1969, Marvin Minsky y Seymour Papert publicaron su libro *Perceptrones*, en el cual demostraban las limitaciones de las redes neuronales y como resultado las organizaciones frenaron las investigaciones fundadas sobre redes neuronales. En 1979, Paul Werbos propone la primera red neuronal eficiente con *Backpropagation*. Sin embargo, David Rumelhart, Geoffrey Hinton y Ronald Williams descubren un método que permite a una red neuronal aprender a discriminar entre dos clases linealmente no separables y también lo nombraron *Backpropagation* [13].

Por bastante tiempo muchos expertos creen que la AI es capaz de acercarse al nivel de cognición del ser humano al crear un conjunto de reglas muy grande capaz de manipular el conocimiento y generar máquinas inteligentes. En este sentido, la AI demuestra ser adecuada para resolver problemas lógicos y bien definidos, como jugar al ajedrez; pero el aplicar reglas explícitas para resolver problemas más complejos resulta poco fiable, en estos problemas se incluyen la clasificación de imágenes, reconocimiento de voz y traducciones en lenguajes naturales. En respuesta, surge un nuevo enfoque para complementar a la AI. Este enfoque fue denominado ML, que surge a partir de preguntas como: *¿puede una computadora ir más allá de lo que le ordenamos y aprender por sí misma una tarea específica?* [14].

El ML es un subcampo de las Ciencias de la Computación, y rama de la AI, que tiene como objetivo el desarrollo de técnicas que proporcionan a los sistemas computacionales la capacidad de aprender y mejorar automáticamente sin la necesidad de ser reprogramados, mediante la generación de modelos que permitan la generalización de comportamientos a partir de información suministrada en forma de datos.

El concepto ML fue introducido por Arthur Samuel en 1959, bajo la siguiente definición: "*Aprendizaje automático: Campo de estudio que proporciona a las computadoras, la capacidad de aprender sin la necesidad de ser reprogramadas"*.

Por su parte, Russell, S. y Norvig definen al ML como el "*proceso mediante el cual una máquina se adapta a nuevas circunstancias buscando mejorar las capacidades de un sistema*" [15]. Siendo ésta una de las principales características de los sistemas inteligentes.

En 1998 Tom Mitchell, otro investigador de ML, da una definición más precisa, "*Un problema de aprendizaje bien planteado, se dice que es un programa informático que aprende de la experiencia E con respecto a alguna tarea T y alguna medida de rendimiento P, sí el rendimiento en T comparado con P, mejora con la experiencia E"* [16].

También se considera que ML es una consecuencia natural de la intersección de informática y estadística [17].

2.2. Tipos de Aprendizaje

Existen diversas formas de clasificar al ML, la más común está en función del método de aprendizaje que utiliza. Un concepto importante dentro del proceso de aprendizaje es el conjunto de entrenamiento.

Definición 1 (conjunto de entrenamiento). Sean $(\mathbf{x}^1, \boldsymbol{c}^1), (\mathbf{x}^2, \boldsymbol{c}^2), \dots, (\mathbf{x}^N, \boldsymbol{c}^N)$ N asociaciones conocidas, cada una formada por un vector de entrada y un vector de salida. Cuando el conjunto formado por estas asociaciones se utiliza en la fase de aprendizaje de un sistema para definir la relación existente entre **x** y **c**, se denomina conjunto fundamental de entrenamiento o simplemente conjunto de entrenamiento (CE) y es denotado por:

$$\{(\mathbf{x}^\mu, \mathbf{c}^\mu) | \mu = 1,2, \dots, N\}, \mathbf{x} \in \mathcal{X} \quad (1)$$

Donde $\mathbf{x}^\mu$ son los vectores de características, $\boldsymbol{c}^\mu$ las respuestas y $\mathcal{X}$ es el espacio de características que agrupa todos los vectores de características posibles y es denominado conjunto universo o espacio muestral.

A continuación, se describen los métodos de aprendizaje más comunes.

2.2.1. Aprendizaje Supervisado

El aprendizaje supervisado consiste en inferir modelos predictivos mediante un proceso denominado entrenamiento, a partir de asociaciones entre datos de entrada y datos de salida previamente definidas. Es decir, este tipo de aprendizaje incluye algoritmos que, a partir del análisis de un conjunto de entrenamiento, son capaces de generar una función utilizada en la asignación del vector de salida adecuada para cada nuevo valor presentado al

sistema, prediciendo de esta forma el valor de salida que corresponde a una entrada determinada.

Dependiendo del tipo de los valores de salida, el problema del aprendizaje supervisado se divide en dos categorías, regresión y clasificación. Cuando la salida es discreta, se trata de un problema de clasificación; por el contrario, si la salida es continua se trata de un problema de regresión. La clasificación y la regresión son técnicas que el aprendizaje supervisado utiliza para desarrollar sus modelos predictivos.

Para el modelo de clasificación, en el CE, $\mathbf{x}^{\mu} \in \mathbb{R}^n$ y $c^{\mu} \in \mathbb{N}$, c^{μ} representa la clase a la que pertenece el vector $\mathbf{x}^{\mu}$.

La clasificación es un problema que consiste en inducir, con el mayor grado de exactitud posible, a partir de la observación de un conjunto finito de casos resueltos, el valor de un caso incompleto que lo identifica dentro de una categoría fijada previamente por un supervisor. Es decir, consiste en una tarea que mediante una función f aproxima un mapeo de variables de entrada $\mathbf{x}$ a variables de salidas c. Las variables de salida a menudo son llamadas categorías, la función de mapeo predice la categoría para el dato de entrada $\mathbf{x}$. Si el sistema de clasificación está formado por dos clases recibe el nombre de clasificación binaria y con más de dos clases se conoce como multiclase.

Formalmente, el problema de clasificación supervisada consiste en asignar un vector $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ a una de las N clases definidas en la aplicación. La clase verdadera se denota por c y toma valores en $\{1,2, \dots, N\}$. Se puede contemplar el clasificador como una función f que asigna etiquetas a observaciones:

$$f: (x_1, x_2, \dots, x_n) \rightarrow \{1,2, \dots, N\} \tag{2}$$

El objetivo es construir un clasificador que mediante una función minimice el coste total de los errores cometidos, esta función se estima a partir de un CE. Existen diversos paradigmas que cumplen con este objetivo, por ejemplo, el análisis de discriminantes [18], el vecino más cercano [19], Naïve Bayes [20], las redes neuronales [21] y las SVM [22], entre otros.

En el modelo de regresión, los elementos del CE tienen las siguientes características, $\mathbf{x}^{\mu} \in \mathbb{R}^{n}$ y $c^{\mu} \in \mathbb{R}$. c^{μ} representa el valor que más se aproxima a las características definidas por $\mathbf{x}^{\mu}$.

La regresión es un procedimiento estadístico que permite establecer una función f que define una relación entre una variable dependiente c^{μ} y un conjunto de variables independientes definidas por los elementos de $\mathbf{x}^{\mu}$. Las variables independientes, también llamadas explicativas o exógenas, representan las características que se usan para predecir el valor de la variable dependiente, que también suele denominarse endógena.

Los modelos de regresión utilizan el modelo predictivo inferido como estimador de los valores de salida dada una entrada.

Para definir formalmente el problema de regresión, considerando la definición de CE de la expresión (1) y asumiendo que existe f y ϵ tal que:

$$c^{\mu} = f(\mathbf{x}^{\mu}) + \epsilon \tag{3}$$

siendo $f: \mathcal{X} \rightarrow \mathbb{R}$ la función subyacente a los datos y ϵ el posible ruido aleatorio presente en los datos. El objetivo es construir un estimador para la función $\bar{f} = f$ de manera que dado cualquier vector $\mathbf{x}^{\mu} \in \mathcal{X}$ sea capaz de predecir el valor de respuesta c^{μ} como $\bar{f}(\mathbf{x}^{\mu})$.

En general, existen cuatro posibles formas en que las variables dependientes e independientes se pueden relacionar, relación lineal directa o simple, relación lineal inversa, relación no lineal directa y relación no lineal inversa. La relación utilizada es aquella que mejor se ajuste a los datos que definen el problema a resolver y por la validación estadística de los pronósticos realizados.

Para comprender el concepto de regresión, se considera el modelo de regresión más sencillo, el lineal simple. Este modelo de regresión está definido por la expresión (4):

$$c = \beta_0 + \beta_1 x + \epsilon \tag{4}$$

En este modelo, c es una función lineal de x (definida por $\beta_0 + \beta_1 x$) más ϵ que representa un componente aleatorio de error que manifiesta la variabilidad que no puede ser explicada con la relación lineal. El término $\beta_0 + \beta_1 x$ es conocido como la *función de*

regresión poblacional (FRP) o recta poblacional. β_1 se denominado parámetro pendiente y mide la relación entre c y x, es decir, cómo cambia c cuando se producen modificaciones en x. β_0 es el término independiente u ordenada al origen y es el valor que toma c cuando x y ϵ son cero. β_0 y β_1 son constantes desconocidas que suelen llamarse coeficientes de regresión y que deben estimarse a partir de los datos de la muestra utilizando el método de mínimos cuadrados. Por lo tanto, β_0 y β_1 son estimados de tal manera que la suma de los cuadrados de las diferencias entre las observaciones c_i y la línea recta sea mínima.

Suponiendo que se dispone de una muestra aleatoria de tamaño N, $\{(\mathbf{x}^\mu, \boldsymbol{c}^\mu)|\mu = 1,2,\dots,N\}$, extraída de la población estudiada, entonces el modelo poblacional para cada observación de la muestra puede ser expresado como:

$$c_i = \beta_0 + \beta_1 x_i + \epsilon_i, i = 1,2,\dots,N \tag{5}$$

La expresión (3) representa un modelo de regresión muestral. El objetivo principal del modelo de regresión es la determinación o estimación de β_0 y de β_1 a partir de una muestra dada. Los estimadores por mínimos cuadrados de β_0 y de β_1 son definidos como:

$$\hat{\beta}_0 = \bar{c} - \hat{\beta}_1 \bar{x} \tag{6}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})\,(c_i - \bar{c})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{7}$$

donde $\bar{c} = \frac{1}{N}\sum_{i=1}^{N} c_i$ y $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.

Entonces, el modelo ajustado de regresión lineal simple, que es una estimación de la FRP, se define por:

$$\hat{c}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \tag{8}$$

Esta expresión es la *función de regresión muestral* (FRM), la cual es la contrapartida de la FRP. Dado que la FRM se obtiene para una muestra dada, una nueva muestra genera otra estimación distinta. La FRM permite calcular el valor ajustado $\hat{c}_i$ para c cuando $x = x_i$. En la FRM $\hat{\beta}_0$ y $\hat{\beta}_1$ son los estimadores de los parámetros β_0 y β_1. Para cada x_i se tiene un valor observado c_i y un valor ajustado $\hat{c}_i$.

Los algoritmos representativos del modelo de regresión son árboles de decisión [23], redes neuronales [21], regresión de soporte vectorial (SVG, *Support Vector Regression*) [24], procesos gaussianos (GPR, *Gaussian Process Regression*) [25], modelo lineal generalizado (GLM, *General Linear Model*) [26] y SVM [22].

2.2.2. Aprendizaje No Supervisado

Un problema que aparece frecuentemente en el ML es el no contar con información acerca de las relaciones y las estructuras subyacentes del conjunto de los datos, ni una función de distribución o modelo matemático que describa estas estructuras. La única información disponible es un gran volumen de datos multidimensionales de entrada y una forma de medir la similitud entre ellos. Al carecer de las salidas asociadas a cada entrada, el CE es redefinido como:

$$\mathbf{X} = \{\mathbf{x}^{\mu} | \mu = 1,2, \dots, N\}, \mathbf{x} \in \mathcal{X} \tag{9}$$

El aprendizaje no supervisado es un paradigma capaz de encontrar y descubrir, de manera automática, patrones de similitud dentro del nuevo CE y agrupar a los elementos de este conjunto en regiones, de manera que se agrupen datos similares dentro de la misma región. Estos descubrimientos pueden realizarse sin ningún tipo de retroalimentación con el medio externo y sin la utilización de información *a priori*.

El agrupamiento (*clustering*) es considerado la técnica de aprendizaje no supervisado de mayor interés en el análisis y procesamiento de datos cuyo objetivo es reducir la cantidad de datos mediante la caracterización o agrupamiento de datos con características similares. Para el modelo de agrupamiento, en el CE, $\mathbf{x}^{\mu} \in \mathbb{R}^{n}$ y, como ya se mencionó, c^{μ} no está disponible.

Formalmente, el problema de agrupamiento consiste en encontrar una partición óptima en el sentido que los elementos de la partición se entiendan como clases o tipos de datos. De acuerdo con Brian Ripley "*los algoritmos de agrupamiento son métodos para dividir un conjunto* **X** *de N observaciones en k grupos de tal manera que miembros del mismo grupo son más parecidos que miembros de distintos grupos*" [27].

Las siguientes definiciones son necesarias para clarificar el concepto de agrupamiento.

Definición 2 (Agrupación). Una **agrupación** de **X** es un conjunto $\mathbf{C} = \{C_1, C_2, \dots, C_k\}$, tal que **C** es una partición de **X** obtenida a partir de un **algoritmo de agrupamiento**. A cada elemento $C \in \mathbf{C}$ se le denomina **cluster** y agrupa a un conjunto de vectores de características **x**. Dado que un agrupamiento es una partición, ésta determina una relación de equivalencia entre los elementos de **X**.

Definición 3 (centroide). Un elemento $\mathbf{x}^{\mu} \in \mathcal{X}$ es un **centroide**, **c**, del clúster C si éste es, a partir de un criterio, representativo de los elementos de C.

Un centroide no tiene por qué ser elemento del clúster, sin embargo, al ser elemento de $\mathcal{X}$ el centroide puede ser comparado con los elementos de **X**. Una forma típica de calcular el centroide **c** de un clúster C está dada por:

$$\mathbf{c} = \frac{1}{\#C} \sum_{\mathbf{x} \in C} \mathbf{x} \tag{10}$$

donde $\#C$ representa la cardinalidad del clúster C.

La similitud entre un elemento $\mathbf{x} \in \mathbf{X}$ y un clúster $C \in \mathbf{C}$ es determinada a partir de una función que calcula la distancia entre el objeto **x** y el centroide **c** del clúster, por ejemplo, considerando como medida de similitud a la norma Euclidiana, la distancia entre **x** y **c** se define como:

$$d(\mathbf{x}, \mathbf{c}) = \|\mathbf{x} - \mathbf{c}\| \tag{11}$$

2.2.3. Aprendizaje Semi-Supervisado

El aprendizaje semi-supervisado posee características del aprendizaje supervisado y del no supervisado. En la información contenida en el CE existen entradas con salidas asociadas (etiquetadas) y entradas que carecen de ellas (sin etiquetar), además, el algoritmo proporciona cierta información de supervisión que no necesariamente se aplica a todos los ejemplos.

El CE del aprendizaje semi-supervisado puede ser dividido en dos secciones, la que agrupa a las asociaciones que incluyen etiqueta y la que agrupa asociaciones que no la incluyen. Entonces, el CE se puede redefinir como:

$$\begin{gathered}\{(\mathbf{x}^{\mu}, \mathbf{c}^{\mu}) | \mu = 1,2, \dots, p\}, \mathbf{x} \in \mathcal{X} \\ \{\mathbf{x}^{\mu} | \mu = p+1, p+2, \dots, p+u\}, \mathbf{x} \in \mathcal{X}\end{gathered} \tag{12}$$

Para este tipo de aprendizaje, se consideran $\mathbf{x}^{\mu} \in \mathbb{R}^{n}$ y $c^{\mu} \in \mathbb{N}$ para un proceso de clasificación y $c^{\mu} \in \mathbb{R}$ para un proceso de regresión.

El aprendizaje semi-supervisado será más útil siempre que haya muchos más datos sin etiquetar, que etiquetados. Es probable que esto ocurra si obtener puntos de datos es barato, pero obtener las etiquetas cuesta mucho tiempo, esfuerzo o dinero. Es el caso de muchas aplicaciones del área de ML, por ejemplo:

- En el reconocimiento de voz, no cuesta casi nada grabar grandes cantidades de voz, pero etiquetarlo requiere que algún humano lo escuche y escriba una transcripción.
- Miles de millones de páginas web están disponibles directamente para su procesamiento automatizado, pero para clasificarlas de manera confiable, las personas deben leerlas.

Dado que los datos no etiquetados llevan menos información que los datos etiquetados, se requieren grandes cantidades para aumentar significativamente la precisión de la predicción. Esto implica la necesidad de algoritmos de aprendizaje semi-supervisados más rápidos y eficientes.

2.2.4. Aprendizaje por Refuerzo

El aprendizaje por refuerzo es un paradigma de aprendizaje dentro de ML que modela un problema de forma que pueda ajustarse a un escenario de agente-ambiente (véase Figura 2.1), siendo esto un enfoque distinto al cómo se modela un problema en aprendizaje supervisado. En el algoritmo de aprendizaje se recibe un tipo de valoración acerca de la ideal de la respuesta dada. A diferencia del escenario de aprendizaje supervisado, el agente no recibe pasivamente un conjunto de datos etiquetados. Sin embargo, recopila información a través de un curso de acciones al interactuar con el entorno. En respuesta a una acción, el

agente recibe dos tipos de información, su estado actual en el entorno y una recompensa de valor real, que especifica la tarea y su objetivo correspondiente (véase Figura 2.1).

El objetivo del aprendizaje por refuerzo es extraer qué acciones deben ser elegidas en los diferentes estados para maximizar la recompensa; de cierta manera, se busca que el agente aprenda una política, que formalmente es una aplicación que dice en cada estado qué acción tomar.

Uno de los desafíos que surge en el aprendizaje por refuerzo y no en otros tipos de aprendizaje es el intercambio entre exploración y explotación. Para obtener una gran recompensa, un agente de aprendizaje por refuerzo debe preferir las acciones que ha intentado en el pasado y que se han encontrado para ser eficaz en la producción de recompensa. Pero para descubrir tales acciones tiene que intentar acciones que no ha seleccionado antes. Otra característica clave del aprendizaje por refuerzo es que considera explícitamente todo el problema de un agente dirigido a un objetivo que interactúa con un entorno incierto.

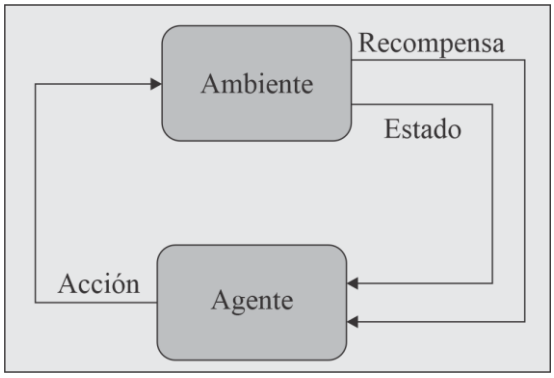

Figura 2.1. Escenario de un aprendizaje por refuerzo (reproducida de [28]).

Además de los tipos de aprendizajes descritos, existen otros con cierto grado de importancia y uso, por ejemplo, aprendizaje Hebbiano, competitivo, mín-máx, estocástico, genético, analítico y conexionista, por mencionar algunos.

Considerando los tipos de aprendizaje anteriores se puede decir que los algoritmos de clasificación se usan cuando el resultado deseado es una etiqueta discreta; existen 2 tipos de clasificación: a) clasificación binaria, es el tipo de clasificación en el que sólo se pueden asignar dos clases diferentes (0 o 1); y b) clasificación multiclase, en esta clasificación pueden asignarse múltiples categorías a las observaciones. Algunos de los algoritmos de

clasificación más usados son Naïve Bayes, vecinos más cercanos (KNN), árboles de decisión, redes neuronales y SVM, entre otros.

Las SVM son el punto medular de esta investigación, por tal motivo, en el Capítulo 3 se estudia detalladamente.

2.3. Tecnologías Utilizadas

El hardware reconfigurable es una tecnología ideal para acelerar cálculos y proporcionar computación de alto rendimiento (HPC, *High Performance Computing*) a bajo costo y bajo consumo de energía [29]. Los FPGA son dispositivos de procesamiento reconfigurables altamente paralelos y potentes que se utilizan para lograr HPC en sistemas integrados con un uso eficiente de los recursos de hardware. Los FPGA han demostrado recientemente mejoras significativas en rendimiento para muchas aplicaciones como procesamiento de imágenes, procesamiento digital de señales y reconocimiento de patrones entre otros.

2.3.1. Arreglo de Compuertas Programable de Campo

Un FPGA es un circuito digital reconfigurable que posee una arquitectura flexible y permite el modelado de sistemas utilizando un paradigma concurrente consiguiendo que el sistema trabaje a frecuencias altas, lo que mejora considerablemente la velocidad de procesamiento. Además, un FPGA soporta el modelado de sistemas digitales por medio de un lenguaje de descripción de hardware (HDL, *Hardware Description language*).

La tecnología de los FPGA fue introducida por la compañía Xilinx en 1985. La arquitectura de un FPGA consiste en un arreglo bidimensional de bloques lógicos configurables (CLB, *Configurable Logic Block*), recursos de interconexión, bloques de entrada/salida (IOB, *Input Output Block*) y bloques embebidos de aplicación específica [30].

El FPGA resulta ser muy versátil al momento de implementar funciones secuenciales o combinacionales, esto se encuentra relacionado con las capacidades de los CLB, elementos que representan la unidad lógica en un FPGA y su flexibilidad para interconectarse entre sí. Los componentes internos de un CLB varían entre los diferentes fabricantes, por lo general contienen un circuito lógico llamado tabla de búsqueda (LUT, *look-up table*). Una LUT

consiste en una SRAM de $m \times 1$, la cual puede ser usada para representar una tabla de verdad de cualquier función lógica de *m* entradas. El bus de direcciones de la SRAM corresponde a las entradas de la tabla de verdad y el bus de datos de la SRAM provee el valor de la función lógica [31]. Los CLB pueden ser conectados mediante la estructura de enrutamiento configurable.

Considerando la complejidad de los CLB, se puede distinguir de dos tipos de FPGA, aquellos que tienen CLB de complejidad baja son denominados de granularidad fina mientras que los de granularidad gruesa están constituidos por CLB de complejidad alta [31]. Los CLB se pueden conectar eficientemente con CLB contiguos en la misa columna o renglón. Además, las conexiones configurables permiten interconectar CLB con IOB, lo que habilita una conexión con el exterior.

La función de los IOB es la de comunicar la lógica interna del FPGA con el exterior, controlando la entrada y salida de datos a través de los elementos que los integran como son los flip-flops, latch y buffer de tercer estado, entre otros. Las características que éstos otorgan a la lógica implementada en un FPGA son bidireccionalidad, alta impedancia, polaridad de salida programable, resistencia *pull-up* y *pull down*, control del *slew-rate* y minimizar la aparición de la metaestabilidad [32].

La interconexión es reconfigurable, tiene por función comunicar a los CLB entre sí, y a estos con los IOB. La interconexión está constituida principalmente por líneas horizontales y verticales que recorren los espacios entre los CLB. Además, existen elementos adicionales que forman parte de estos recursos, llamados puntos de interconexión y matrices de interconexión (véase Figura 2.2).

Por último, los bloques de aplicación específica embebidos son aquellos elementos de arquitectura fija que forman parte de la estructura del FPGA. Funciones que comúnmente son integradas como elementos de arquitectura fija son aquellos que son requeridos por una gran variedad de aplicaciones y/o consumen bastos recursos en su implementación. Los elementos embebidos comúnmente encontrados en un FPGA son: bloques RAM, (BRAM), multiplicadores y administradores de señales de reloj (DCM, *Digital Clock Manager*). Cada vez es más frecuente que fabricantes de FPGA incluyan en sus arquitecturas procesadores embebidos, esta unión es conocida como FPSoC (*Field-programmable system-on-chip*).

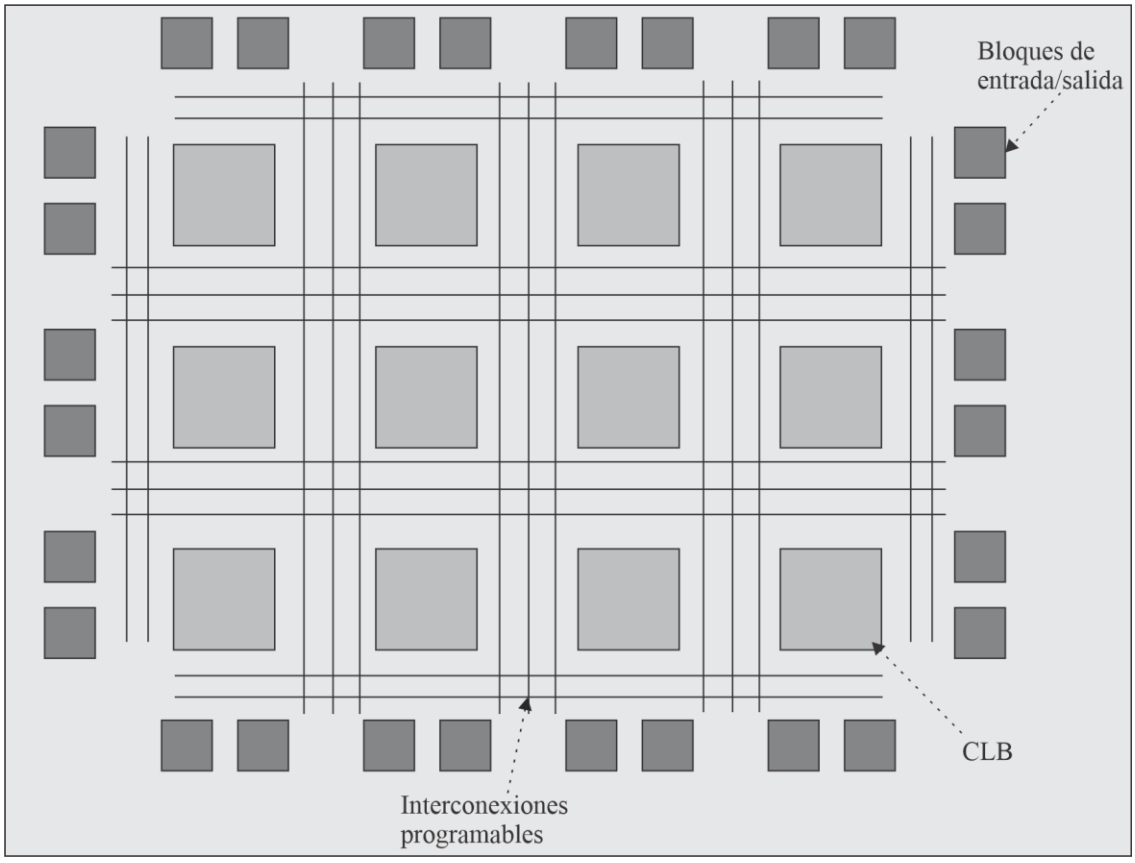

Figura 2.2. Arquitectura del FPGA.

En la actualidad, diversas compañías como Xilinx, Altera, Lattice Semiconductor, QuickLogic, Atmel y Achronix Semiconductor, fabrican dispositivos FPGA. Los campos de aplicación de los FPGA son bastos, tales como: uso militar, automotriz, medicina, video y audio (véase Tabla 2.1). La empresa Xilinx ofrece una amplia gama de dispositivos FPGA integrados en familias, como Spartan, Artix, Kintex y Virtex.

Tabla 2.1. Familias de FPGA en grados de uso.

Automotriz	Militar	Aeroespacial	Familias adicionales
XA Spartan-7	XQ Kintex UntraScale+	Virtex-5QV	Virtex-6
XA Spartan-6	XQ Virtex UntraScale+	Virtex-4QV	Virtex-5
XA Spartan-3A	XQ Virtex-7		Virtex-4
XA Spartan 3A DSP	XQ Kintex-7		Spartan-3A
XA Spartan 3E	XQ Artix-7		Spartan-3AN
	XQ Virtex-6		Spartan-3A DSP
	XQ Spartan-6		Spartan-3A Extended
	XQ Virtex-5		Spartan-3E
	XQ Virtex-4		Spartan-3

En este trabajo de investigación se hace uso de la tarjeta de desarrollo Nexys-3 de la compañía Diligent Inc., la cual cuenta con un FPGA Spartan 6, y que se puede emplear en un sin número de aplicaciones. Esta tarjeta ofrece los siguientes recursos: cuatro conectores de expansión de 12 terminales (pines), denominados PMOD y utilizados para integrar periféricos al sistema como controladores de motor, ADC y DAC, circuitos de audio, y una serie de interfaces de sensores y actuadores. Todas las señales de la tarjeta Nexys-3 cuentan con protección contra descargas electrostáticas (ESD, *electrostatic discharge*) y contra cortocircuitos, lo que garantiza una larga vida útil en cualquier entorno. Adicionalmente, la tarjeta Nexys-3 cuenta con las siguientes características:

- Puerto USB para la configuración del FPGA y trasferencia de datos.
- 16 MB de PSDRAM Micron y 16MB de ROM Intel StrataFlash.
- Plataforma Flash Xilinx para las configuraciones del FPGA.
- Oscilador de 100MHz.
- Conectores de Entrada/Salida de propósito general.
- Puerto de datos VGA, PS2 y LAN.
- 8 leds, 4 displays de 7 segmentos, 5 botones y 8 interruptores.

2.3.2. Herramienta para la descripción

La herramienta para la Automatización del Diseño Electrónico (EDA, *Electronic Design Automation*) y el Diseño Asistido por Computadora (CAD, *Computer Aided Design*) utilizadas en la síntesis e implementación del sistema diseñado es el software ISE Foundations versión 14.7 de la compañía Xilinx Inc.

La herramienta CAD ISE Foundation es un entorno de desarrollo (IDE, *Integrated Development Environment*) que permite navegar fácilmente por las diferentes fases del proceso de modelado de un sistema digital desde el diseño hasta la implementación sobre una arquitectura configurable. Esta herramienta permite modelar de manera adecuada las fases a desarrollar, capturando los diseños en diagrama esquemático y en lenguaje descriptor de hardware. En el caso de captura de esquemático, Xilinx cuenta con bibliotecas bastante extensas de componentes de uso común, facilitando las interconexiones entre componentes para describir el sistema digital; mientras que, para el modelado descriptor, Xilinx integra

dos de los estándares más populares VHDL y Verilog, teniendo así la facilidad de desarrollo con uno u otro lenguaje.

También incluye una herramienta para realizar una síntesis lógica del código HDL o del diagrama esquemático. A través de una compilación es posible producir un *netlist* a partir de cualquiera de los métodos de captura. Un *netlist* es un archivo que registra la descripción de las celdas lógicas (primitivas) y las conexiones entre ellas para conformar un diseño. El formato de netlist más utilizado es el Formato de Intercambio de Diseño Electrónico (EDIF, *Electronic Design Interchange Format*). El proceso que le continua a la síntesis es la implementación y comprende las siguientes fases:

- Mapeo o Planeación de superficie (Floorplanning), el archivo netlist generado por la síntesis se mapea sobre el FPGA seleccionado. La configuración de las celdas lógicas y sus respectivas conexiones, descritas por el netlist, se distribuyen sobre la superficie del circuito integrado a manera de identificar recursos. Los algoritmos involucrados en este proceso permiten minimizar espacio físico y retardos de señal, en una primera aproximación.
- Colocación (*Placement*): estratégicamente, el software decide la colocación de las primitivas sobre un bloque del dispositivo físico (módulo lógico). Los algoritmos inteligentes con los que trabaja el sistema de colocación deciden la mejor ubicación para cada celda lógica, considerando las redundancias en el diseño (componentes o conexiones repetidas) y una segunda aproximación para eliminar los retardos críticos.
- Trazado o Ruteo (*Rutting*): realiza la conexión física entre los módulos lógicos y determina el tipo de interconexión proponiendo rutas cortas o largas, eligiendo la mejor opción.
- Traducción (*Translate*): es la extracción de características del circuito, determinando la resistencia y capacitancia eléctrica entre las interconexiones, para verificar un correcto ruteo de las líneas, El software se encarga automáticamente de generar el ruteo y la extracción de impedancias.

Después de que se han colocado y ruteado las celdas primitivas y los módulos que las contienen, es posible simular la configuración física que ha sido adquirida por el dispositivo,

a esta simulación se le conoce como física, ya que se aproxima al comportamiento real que tendrá el diseño. Aquí es posible considerar los retardos de propagación que sufren las señales.

Posteriormente viene la programación del dispositivo, que consiste en un conjunto de módulos que permiten configurar el dispositivo elegido; estos módulos son, el módulo generador de un archivo con la información necesaria para la configuración del dispositivo y el módulo que permite la interfaz con la herramienta hardware necesaria para la configuración.

2.4. Estado del Arte

El presente trabajo versa sobre la implementación de SVM sobre lógica reconfigurable. Por ello, a continuación, se hace mención de algunos trabajos relacionados con esta área.

Cadambi *et al.* (2009) presentan un coprocesador basado en FPGA para entrenamiento y clasificación de SVM, esta arquitectura utiliza una aritmética de baja precisión basada en los DSP *slices* del FPGA, usando multiplicadores- acumuladores (MAC, *multiply–accumulate*) en paralelo. Para el entrenamiento de SVM observaron velocidades de cálculo a nivel de aplicación de más de 9 mil millones de multiplicaciones acumuladas por segundo (GMACs). Con esta arquitectura se obtuvo velocidades de procesamiento aceptables, comparadas con un CPU Opteron dual de 2.2Ghz [33].

Una implementación de un sistema de reconocimiento de altavoces, basado en SVM e implementado en un FPGA Spartan 3, es presentado por Ramos Lara *et al.* (2009), donde con una frecuencia de operación de 50Mhz obtuvieron rendimientos similares a los logrados en una computadora con procesador Pentium IV. El sistema propuesto es capaz de realizar una extracción de vectores de entidades con una base de datos de 3,634 vectores en 285μs en el FPGA y 436μs en la Pentium IV, el sistema implementa las operaciones en formato de punto fijo para reducir el área necesaria en el FPGA [34].

Papadonikolakis *et al.* (2009) presentan una comparación de implementaciones del algoritmo SVM en un FPGA y una GPU, centrando su investigación en la explotación del paralelismo del algoritmo utilizando los recursos del hardware que ofrece una GPU y un

FPGA. Los tiempos de procesamiento del kernel en el FPGA y el kernel de la GPU son normalizados para permitir la generalización de resultados a dispositivos más grandes. La tarjeta de desarrollo para el FPGA es una Altera Stratix III EP3SE110-F780C2, mientras que la GPU es una NVidia GeForce 8500GT. El tamaño del conjunto de datos fue de 315,000 [35].

Papadonikolakis y Bouganis (2010) proponen una arquitectura FPGA heterogénea completamente escalable para impulsar el aprendizaje de una SVM. Esta arquitectura explota en forma eficiente la potencia de procesamiento paralelo del dispositivo y la diversidad de rango dinámico de los requisitos de precisión entre las características de problemas de entrenamiento [36].

Un clasificador dinámico sobre lógica reconfigurable de propósito general es presentado por Gomes Filho *et al.* (2010), implementando el algoritmo SVM-SMO para la fase de entrenamiento y soportando diversos tamaños de conjuntos de entrenamiento con un amplio número de muestras y elementos. Este modelado ofrece un ahorro del área del 22.38%, la implementación se hace sobre un FPGA Virtex-IV XC4VLX25 [37].

Bauer *et al.* (2010) realizan tres arquitecturas SVM diferentes para un multisensor de detección de peatones y mejorar la seguridad en intersecciones. Estas arquitecturas son relizadas en un FPGA, una CPU y una GPU, utilizando un kernel gaussiano acelerado por hardware para el procesamiento en tiempo real. La propuesta consiste en una arquitectura de hardware flexible que puede ser implementada en una PC. Como etapa previa a la clasificación, se calcula un descriptor de Histograma de Gradientes Orientados (HOG, *Histogram of Oriented Gradient*), la extracción de características se realiza en el FPGA en tiempo real con un esquema de convolución que tiene una latencia de 312µs, La clasificación SVM del kernel se calcula en paralelo en la GPU con esta combinación, la evaluación de 1,000 ventanas toma menos de 100ms al presentar candidatos basados en la fusión de imágenes multiespectrales [38].

Pan *et al.* (2013) presentan un diseño de la función de decisión del SVM con un kernel de bajo costo en recursos y fácil implementación en un FPGA. El diseño fue modelado para cubrir problemas de clasificación y regresión. Para las pruebas se utiliza el conjunto de datos estándar UCI de cáncer de mama. En este estudio eligen operaciones con punto fijo para el

estudio de clasificación y reducen el número de bits de los parámetros de entrada para la regresión. El error cuadrático medio es menor de 0.004 en experimentos de regresión. Verifican el rendimiento con un experimento de regresión simple y encuentran que a mayor dimensionalidad del conjunto presentaba un incremento en el costo computacional [39].

Pietron *et al.* (2013) proponen una comparación del algoritmo SVM implementado en un FPGA y en una GPU y aplicado a la segmentación rápida de imágenes. Se basan en el rendimiento, tanto del hardware como del software, observan que para imágenes con un mayor número de pixeles el rendimiento de la GPU es mejor respecto al del FPGA. Los tiempos de rendimiento obtenidos para GPU y FPGA fueron para 51,200 pixeles, 12.15ms y 42.24ms respectivamente, mientras que para 2,048 pixeles fueron 0.6ms y 0.16896ms respectivamente [40].

Una implementación de un clasificador SVM sobre lógica reconfigurable para la clasificación de microarreglos, es propuesta por Hussain, Benkrid, y Seker (2013). Esta es una solución personalizada de alto rendimiento analizando el paralelismo disponible de los kernel de la SVM. Al comparar con un procesador de propósito general (GPP), encuentran que la implementación en el FPGA es hasta 85 veces más rápida, concluyen que en FPGA proporcionan una solución de alto rendimiento para análisis de microarreglos [41].

Groléat, Arzel y Vaton (2014) proponen un clasificador SVM sobre FPGA y lo aplicaron a la clasificación de tráfico en internet. El diseño está presentado en tiempo real, procesando cientos de Gb/s para permitir la detección en línea de categorías de aplicaciones. Utilizan un FPGA como netFPGA-10G y aprovechado el acceso de muy bajo nivel a las interfaces de red en conjunto con paralelismo masivo, prueban dos kernel. Un kernel creado con una función de base radial (RBF, *radial basis function*) y otro un diseño más adaptado a hardware, mediante el algoritmo CORDIC (*COordinate Rotation DIgital Computer*). Ambas implementaciones proporcionan niveles similares de precisión de clasificación, sin embargo, la implementación de CORDIC permite frecuencias de trabajo más altas y menos uso de área en el FPGA [42].

Rabieah y Bouganis (2015) presentan un sistema completo basado en FPGA para impulsar el entrenamiento SVM no lineal. Los autores utilizaron un enfoque general de aprendizaje automático buscando un mejor rendimiento predictivo combinando múltiples

modelos (*Ensemble Learning*). Además, proponen un flujo de entrenamiento de precisión múltiple en cascada, que explota la re-configurabilidad del FPGA y la diversidad del problema de entrenamiento para el manejo de grandes conjuntos de datos [43].

Una arquitectura hardware paralela enfocada en la clasificación de imágenes en tiempo real (detección de objetos) utilizando un modelo SVM implementado en un FPGA es presentada por Qasaimeh, Sagahyroon, y Shanableh (2015). Dicha arquitectura se complementa con los algoritmos de transformación de característica invariantes a escala (SIFT, *Scale-invariant feature transform*) y bolsa de características (BoF, *Bag of Features*), usados en la fase de extracción de características [44].

Ho *et al.* (2016) proponen un entorno integrado con apache Spark para facilitar la implementación de aplicaciónes mixtas de hardware y software que se encuentran en clústeres distribuidos acelerados por FPGA, especificamente, utilizando el entrenamiento de un clasificador SVM para imagendes de celulas biologicas. Mejorando el rendimiento general de los datos entre la CPU y el FPGA, comparando con el procesamiento al usar solo clusters en el CPU, reportan velocidades de 1.6x en el entrenamiento dentro del FPGA con nodos del cluster entre 1 a 8 [45].

Lopes, Ferreira y Fernandes (2019) muestran una implementación sobre FPGA del algoritmo SVM utilizando gradiente de descenso estocástico (SGD, *Stochastic Gradient Descent*) como método de entrenamiento. Esta implementación tiene como objetivo lograr una alta tasa de datos procesados con el fin de satisfacer las demandas de aplicaciones computacionalmente intensivas. Observan que la implementación de esta técnica en hardware permite mejoras significativas en rendimiento [46].

Elgawi, Mutawa y Ahmad (2019) proponen una arquitectura de SVM binarizado llamado (eBSVM, *embedded binarized support vector machine*) sobre un FPGA. La arquitectura propuesta calcula los valores binarizados utilizando los pesos de Hamming y remplazando las operaciones de multiplicación y adición por operaciones XNOT y *popcount* bit a bit, lo que reduce el tiempo de ejecución y consumo de energía. Esta propuesta es evaluada para los conjuntos de MNOST y CIFAR-10, muestran que las demandas computacionales son satisfechas mediante la aceleración en hardware propuesta [47].

Una implementación paralela sobre FPGA de SVM es propuesta por Noronha, Torquato y Fernandes (2019), la fase de entrenamiento se implementa mediante la optimización mínima secuencial (SMO, *sequential minimal optimization*) que es una implementación de bajos recursos y compatible con el hardware (HFK, *Hardware Friendly Kernel*), utilizado para reducir el área del kernel. Realizan un análisis detallado sobre la implementación, analizan tanto la simulación y como la síntesis. A partir de esto validan la arquitectura y realizan un análisis del comportamiento del sistema respecto a parámetros esenciales como área de ocupación y rendimiento del sistema, verifican que con el desarrollo de este algoritmo es posible alcanzar alto rendimiento [48].

Capítulo 3. Máquinas de Soporte Vectorial

Las SVM son una técnica diseñada para tareas de clasificación y regresión. Para tareas de clasificación, una SVM puede verse como un conjunto de métodos de ML que toman los datos de entrada como dos conjuntos de vectores en un espació n −dimensional, donde cada conjunto representa una categoría y la SVM construye un hiperplano $(n-1)$ −dimensional de separación que maximiza el margen entre los dos conjuntos (problema de clasificación de dos clases).

3.1. SVM para Tarea de Clasificación

El objetivo principal es separar los objetos en dos grupos mediante una función que se define a partir de los propios objetos; pueden existir diferentes hiperplanos para la separación de dichos objetos, pero sólo uno es el que maximiza la distancia entre las muestras más cercanas de las diferentes clases como se muestra en la Figura 3.1. Esta distancia máxima se conoce como *Margen de separación* y el hiperplano que maximiza esa distancia entre él y las muestras más cercanas de las dos clases se nombra *Hiperplano Optimo de Separación.* [49].

El análisis de un proceso de clasificación de datos usando SVM contempla los siguientes casos: linealmente separable, linealmente no separable y multiclase.

3.1.1. Caso linealmente separable

Un conjunto de datos n-dimensionales, que define un problema, es linealmente separable si existe un hiperplano de $n-1$ dimensiones que es capaz de separarlos perfectamente en grupos. Por tanto, se debe encontrar dicho hiperplano como superficie de decisión que separe las clases con el mayor margen posible.

La ecuación general de un hiperplano en n −dimensiones es:

$$(\mathbf{x} * \mathbf{w}) + b = 0 \tag{13}$$

donde $\mathbf{x}$ es un vector de tamaño $n \times 1$; $\mathbf{w}$ es la normal al hiperplano y b es un valor escalar. De todos los puntos del hiperplano sólo uno tiene la menor distancia al origen:

$$S_{min} = \frac{|b|}{\|\mathbf{w}\|} \tag{14}$$

Sin embargo, existe un error de redundancia en (14), por lo que, sin que se pierda generalidad, se puede considerar que los hiperplanos canónicos y sus parámetros $\mathbf{w}$ y b están restringidos por:

$$min_i |\langle \mathbf{w}, \mathbf{x}^{\mu} \rangle + b| = 1 \tag{15}$$

La Figura 3.1 muestra la distancia al punto más cercano a cada posible hiperplano de separación.

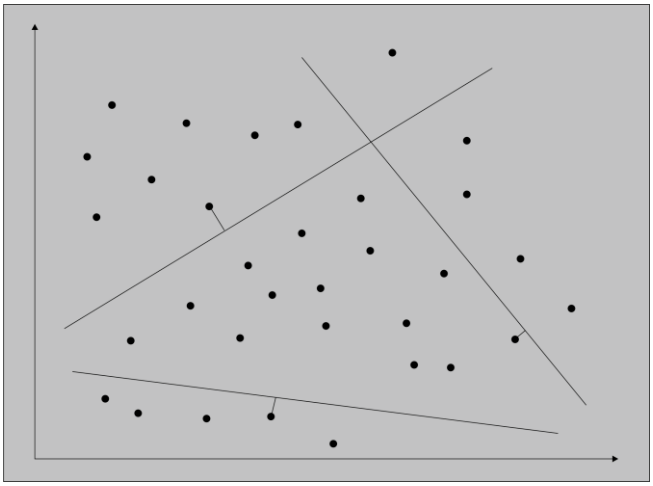

Figura 3.1. Distancia más cercana de los datos a cada posible hiperplano de separación.

Ahora, sea un problema de clasificación binaria donde la extracción de características se realiza inicialmente, entonces se clasifican los datos de entrenamiento $\mathbf{x}^{\mu} \in \mathbb{R}^n$ con la etiqueta $\mathbf{c}^{\mu} \in \{-1; +1\}$ para todos los datos o muestras de entrenamiento $\mu = 1 \dots N$, donde N es la cantidad de muestras de entrenamiento y n es la dimensión del problema.

Cuando dos clases son linealmente separables en $\mathbb{R}^n$, es necesario encontrar un hiperplano que ofrezca el menor error de generalización entre el número infinito de posibles hiperplanos. Este hiperplano debe ser el que maximice el margen de separación de las dos clases y dicho margen es la suma de las distancias del hiperplano a los puntos más cercanos de cada una de las clases. Es decir, dado los patrones de entrenamiento $(\mathbf{x}^1, \mathbf{c}^1), \dots, (\mathbf{x}^N, \mathbf{c}^N)$ donde $\mathbf{x}^{\boldsymbol{\mu}}$ es un vector $n-$dimensional y N es el número de muestras de entrenamiento, entonces:

$c^{\mu} = 1$ Si $\mathbf{x}^{\mu}$ pertenece a la clase A

$c^{\mu} = -1$ Si $\mathbf{x}^{\mu}$ pertenece a la clase B

Si los datos con linealmente separables, entonces existe un vector $\mathbf{w}$ n-dimensional y un escalar b tales que:

$$c^{\mu}(\boldsymbol{w} * \mathbf{x}^{\boldsymbol{\mu}} - b) \geq 1 \tag{16}$$

O bien $-(c^{\mu}(\mathbf{w} * \mathbf{x}^{\mu} - b) - 1) \leq 0$, donde el par $(\mathbf{w}, b)$ define el hiperplano que separa las dos clases de datos. Con el propósito de hacer cada superficie de decisión $(\mathbf{w}, b)$ única, se normaliza la distancia perpendicular desde el origen al hiperplano separador dividiéndola entre $\|\boldsymbol{w}\|$, dando la distancia como $\frac{|b|}{\|\mathbf{w}\|}$. La distancia perpendicular del origen al hiperplano $H_1(\mathbf{w} * \mathbf{x}^{\mu} - b = 1)$ es $\frac{|1+b|}{\|\mathbf{w}\|}$ y del origen al hiperplano $H_2(\mathbf{w} * \mathbf{x}^{\mu} - b = -1)$ es $\frac{|b-1|}{\|\mathbf{w}\|}$. Aquellos objetos que se encuentren sobre los hiperplanos H_1 y H_2 son llamados *Vectores de Soporte* (véase Figura 3.2).

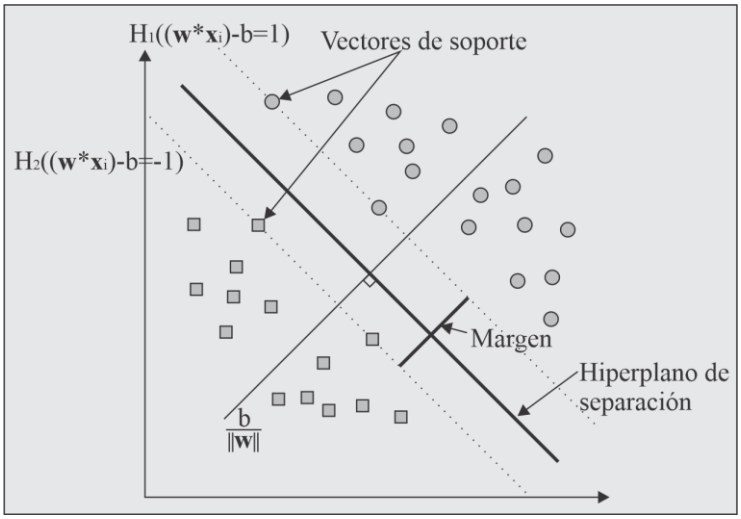

Figura 3.2. Plano óptimo de separación y vectores de soporte.

Si se elimina alguno de los puntos que se encuentra sobre los hiperplanos H_1 y H_2 no afecta el resultado de la clasificación, pero si se elimina alguno de los vectores de soporte, sí. La distancia entre los hiperplanos es $\frac{2}{\|\mathbf{w}\|}$, donde $\frac{1}{\|\mathbf{w}\|}$ es el margen y determina la capacidad de la máquina de aprendizaje.

Para lograr una clasificación correcta el objetivo es maximizar $\frac{2}{\|\mathbf{w}\|}$, lo cual es equivalente a minimizar $\frac{\|\mathbf{w}\|^2}{2}$. Es ese punto donde se puede formular como:

Maximizar:

$$f = \frac{\|\mathbf{w}\|^2}{2} \tag{17}$$

sujeto a:

$$g^{\mu} = -(c^{\mu}(\mathbf{w} * \mathbf{x}^{\mu} - b) - 1) \leq 1, \ \ \mu = 1, \dots, N \tag{18}$$

donde N es el número de datos de entrenamiento.

Este problema puede resolverse por medio de programación cuadrática (QP, *quadratic programming*). Sin embargo, usar el método de Lagrange [50] facilita el entender el análisis realizado al caso linealmente separable.

La función Lagrangiana para un problema de clasificación es:

$$L_p(\mathbf{w}, b, \mathbf{\Lambda}) = f(\mathbf{x}) + \sum_{\mu=1}^{N} \lambda^{\mu} g^{\mu}(\mathbf{x})$$

$$= \frac{\mathbf{w} * \mathbf{w}}{2} - \sum_{\mu=1}^{N} \lambda^{\mu} (c^{\mu}(\mathbf{w} * \mathbf{x}^{\mu} - b) - 1) \tag{19}$$

$$= \frac{\boldsymbol{w} * \boldsymbol{w}}{2} - \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} \mathbf{w} * \mathbf{x}^{\mu} + \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} b + \sum_{\mu=1}^{N} \lambda^{\mu}$$

donde $\Lambda = \{\lambda^1, \lambda^2, \dots, \lambda^N\}$ son los multiplicadores de Lagrange del conjunto de entrenamiento.

Ahora, considerando las condiciones de Karush-Kuhn-Tucker [50] para el problema anterior, se considera la condición de gradiente y está dada por:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} \mathbf{x}^{\mu} = 0 \tag{20}$$

donde $\frac{\partial \boldsymbol{L_p}}{\partial \mathbf{w}} = \left(\frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w_1}}, \frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w_2}}, \dots, \frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w_N}}\right)$

$$\frac{\partial L_p}{\partial b} = \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} = 0 \tag{21}$$

$$\frac{\partial L_p}{\partial \lambda^{\mu}} = g^{\mu}(\mathbf{x}) = 0 \tag{22}$$

La condición de ortogonalidad por:

$$\lambda^{\mu} g^{\mu} = -\lambda^{\mu} (c^{\mu}(\mathbf{w} * \mathbf{x}^{\mu} - b) - 1) = 0, \quad \mu = 1, \dots, N \tag{23}$$

Y la condición de no-negatividad por:

$$\lambda^{\mu} \geq 0, \qquad \mu = 1, \dots, N \tag{24}$$

El punto estacionario del Lagrangiano determina las soluciones para el problema de optimización. Sustituyendo las ecuaciones (21) y (22) en la parte derecha de la función

Lagrangiana, se reduce la función a la forma dual con λ^{μ} como variable dual. Así el problema después de la sustitución queda de la siguiente manera:

Maximizar:

$$L_D = \sum_{\mu=1}^{n} \lambda^{\mu} - \frac{1}{2} \sum_{\mu,j=1}^{N} \lambda^{\mu} \lambda^{j} c^{\mu} c^{j} \mathbf{x}^{\mu} * \mathbf{x}^{j} \tag{25}$$

sujeto a:

$$\sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} = 0, \qquad \lambda^{\mu} > 0, \qquad \mu = 1, \dots, N \tag{26}$$

Se pueden obtener todos los valores de λ resolviendo la (25) mediante QP y $\boldsymbol{w}$ puede calcularse usando la ecuación (20):

$$\sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} (\mathbf{x}^{\mu})^{T} \mathbf{x}^{\mu} = 0$$

$$\mathbf{w} = \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} \mathbf{x}^{\mu} \tag{27}$$

Los valores de λ_i que sean mayores que cero para las restricciones dadas corresponden a los vectores de soporte del sistema. Se puede calcular el valor de *b* usando la ecuación (23):

$$\lambda^{\mu}(c^{\mu}(\mathbf{w} * \mathbf{x}^{\mu} - b) - 1) = 0, \qquad \mu = 1, \dots, N \tag{28}$$

seleccionado el valor de $\mathbf{x}^{\mu}$ con λ distinto de cero.

La clase de un dato de entrada **x** es entonces determinada por:

$$Clasificación(\mathbf{x}) = signo(\mathbf{w} * \mathbf{x} - b) \tag{29}$$

3.1.2. Caso linealmente no separable

En la mayoría de los problemas reales el hiperplano de separación lineal no existe, sin embargo, se puede buscar alguno que permita la clasificación introduciendo un conjunto de variables ξ que permiten medir el grado de violación a las restricciones para el caso linealmente no separable.

La formulación del problema sería para este caso es:

Maximizar

$$f(\boldsymbol{w}, \boldsymbol{\Xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + P\sum_{\mu=1}^{N}\xi \tag{30}$$

sujeto a:

$$\begin{aligned} &\boldsymbol{c}^{\mu}(\mathbf{w} * \mathbf{x}^{\mu} - b) \geq 1 - \xi^{\mu} && \mu = 1, \dots, N \\ &\xi^{\mu} \geq 0, && \mu = 1, \dots, N \end{aligned} \tag{31}$$

donde $\boldsymbol{\Xi} = (\xi_1, \dots, \xi_n)$ y P son parámetros determinados *a priori* que pueden ser vistos como valores de penalización. Un valor pequeño de P maximiza el margen y el hiperplano es menos sensible a los datos anómalos en las muestras de entrenamiento; mientras que un valor grande de P minimiza el número de los puntos mal clasificados.

La función Lagrangiana quedaría como:

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + P\sum_{\mu=1}^{N}\xi_i - \sum_{\mu=1}^{N}\lambda^{\mu}(c^{\mu}(\mathbf{x}^{\mu} * \mathbf{w} - b) - 1 + \xi^{\mu}\ ) - \sum_{\mu=1}^{N}\zeta^{\mu}\xi^{\mu} \tag{32}$$

donde ζ^{μ} son los multiplicadores de Lagrange introducidos por la restricción $\xi^{\mu} \geq 0$.

Las condiciones de e-Kuhn-Tucker para este problema serían:

Condición del gradiente:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} \mathbf{x}^{\mu} = 0 \tag{33}$$

donde $\frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w}} = \left(\frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w_1}}, \frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w_2}}, \ldots, \frac{\partial \boldsymbol{L_p}}{\partial \boldsymbol{w_N}}\right)$.

$$\frac{\partial L_p}{\partial b} = \sum_{\mu=1}^{N} \lambda^{\mu} c^{\mu} = 0 \tag{34}$$

$$\frac{\partial L_p}{\partial \xi^{\mu}} = P - \lambda^{\mu} - \zeta^{\mu} = 0 \tag{35}$$

$$\frac{\partial L_p}{\partial \lambda^{\mu}} = -(c^{\mu}(\mathbf{x}^{\mu} * \mathbf{w} - b) - 1 + \xi^{\mu}) \tag{36}$$

Condición de ortogonalidad:

$$\lambda^{\mu}(c^{\mu}(\boldsymbol{x}^{\mu} * \mathbf{w} - b) - 1 + \xi^{\mu}) = 0, \qquad \mu = 1, \ldots, N \tag{37}$$

Condición de viabilidad:

$$(c^{\mu}(\mathbf{x}^{\mu} * \mathbf{w} - b) - 1 + \xi^{\mu}) = 0, \qquad \mu = 1, \ldots, N \tag{38}$$

Condición de no negatividad:

$$\xi^{\mu} \geq 0, \qquad \mu = 1, \ldots, N \tag{39}$$

$$\lambda^{\mu} \geq 0, \qquad \mu = 1, \ldots, N \tag{40}$$

$$\zeta^{\mu} \geq 0, \qquad \mu = 1, \ldots, N \tag{41}$$

$$\zeta^{\mu}\xi^{\mu} \geq 0, \qquad \mu = 1, \ldots, N \tag{42}$$

Sustituyendo las ecuaciones (33) y (34) en la parte derecha de la función lagrangiana se obtiene el mismo problema dual con las mismas variables Lagrangianas que el caso anterior:

Maximizar:

$$L_D = \sum_{\mu=1}^{N} \lambda^{\mu} - \frac{1}{2} \sum_{\mu,j=1}^{N} \lambda^{\mu} \lambda^{j} c^{\mu} c^{j} \boldsymbol{x}^{\mu} * \mathbf{x}^{j} \tag{43}$$

sujeto a:

$$0 \leq \lambda^{\mu} \leq P$$

$$\sum_{\mu=1}^{N} \lambda^{\mu} C^{\mu} = 0 \tag{44}$$

La solución para **w** puede ser determinada por la ecuación (33) que describe una de las condiciones de Karush-Kuhn-Tucker $\mathbf{w} = \sum_{\mu=1}^{N} \lambda^{\mu} C^{\mu} \boldsymbol{x}^{\mu}$.

Nuevamente, b puede ser encontrado promediando todos los valores de b entre todas las muestras de entrenamiento; los cuales pueden ser calculados usando las siguientes condiciones de Karush-Kuhn-Tucker:

$$\lambda^{\mu}(c^{\mu}(\mathbf{x}^{\mu} * \mathbf{w} - b) - 1 + \xi^{\mu}) = 0 \tag{45}$$

$$(P - \lambda^{\mu})\xi^{\mu} = 0 \tag{46}$$

La ecuación (46) también indica que $\xi^{\mu} = 0$ si $\lambda^{\mu} < P$. Por lo tanto, b puede ser calculado solamente sobre aquellas muestras que cumplan que:

$$0 < \lambda^{\mu} < P \tag{47}$$

Si $\lambda^{\mu} < P$ entonces $\xi^{\mu} = 0$ en este caso los vectores de soporte se encuentran a una distancia de $\frac{1}{\|\mathbf{w}\|}$ del hiperplano separador (Vectores de soporte Marginales). Cuando $\lambda^{\mu} = P$ entonces los vectores de soporte son puntos mal clasificados si $\xi^{\mu} > 0$. Cuando $0 < \xi^{\mu} < 1$ los vectores de soporte son clasificados correctamente, pero están más cerca que $\frac{1}{\|\mathbf{w}\|}$ del hiperplano (véase Figura 3.3).

La función de decisión es descrita en la siguiente ecuación:

$$D(x) = \sum_{\mu \in N} \alpha_i y_i x_i^T x + b \tag{48}$$

donde N es el conjunto de índices de vectores de soporte. Debido a que α_i no es cero para los vectores de soporte en la ecuación (48), es sólo para estos vectores.

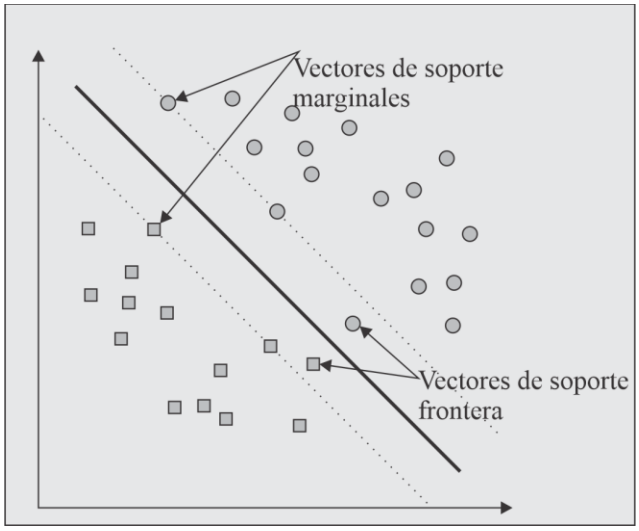

Figura 3.3. Vectores de soporte marginales y frontera.

El modelo de las SVM es un método de aprendizaje automático supervisado que se utiliza para una clasificación eficiente con alta precisión en diversas aplicaciones como detección de objetos, reconocimiento de voz, bioinformática, clasificación de imágenes y diagnóstico médico, entre otras.

3.1.3. Truco de kernel

En una SVM el hiperplano óptimo está determinado para maximizar el margen de separación, sin embargo, si los datos de entrenamiento no son linealmente separables, el clasificador obtenido puede tener un margen de separación que no es óptimo. Por lo tanto, para mejorar la separación de los datos, al espacio de entrada original se le aplica una función llamada kernel para hacer un mapeo a una dimensión más, creando un espacio de características mayor [51].

Usando una función vectorial no lineal $g(x) = (g_1(x), \dots, g_n(x))^T$ que asigna el vector de entrada x de dimensión m al espacio de características de dimensión l, la función de decisión lineal en el espacio ésta dada por:

$$D(x) = \mathbf{w}^T g(x) + b \tag{49}$$

donde **w** es un vector de dimensión n y b es el término de bias.

De acuerdo a la teoría de Hilbert-Schmidt [52], si una función simétrica $H(\boldsymbol{x}, \boldsymbol{x}')$ satisface:

$$\sum_{\mu,j=1}^{N} h_\mu h_j H(\boldsymbol{x}_\mu, \boldsymbol{x}_j) \geq 0 \tag{50}$$

para todo $N, \boldsymbol{x}_i, h_i$ donde N es un numero natural y h_i son números reales, existe una función, $g(x)$, que asigna un x en un espacio de características con producto punto en $g(x)$ que satisface:

$$H(\boldsymbol{x}, \boldsymbol{x}') = g^t(x) g(x') \tag{51}$$

sí en (51) cumple con:

$$\sum_{\mu,j=1}^{N} h_\mu h_j H(x_\mu, x_j) = \left(\sum_{\mu=1}^{N} h_\mu g^t(x_\mu) \right) \left(\sum_{\mu=1}^{N} h_i g(x_\mu) \right) \geq 0 \tag{52}$$

A la condición descrita en la ecuación (50) o la ecuación (52) se le llama condición de Mercer y la función que da solución a dichas ecuaciones se llama kernel semidefinido positivo o kernel de Mercer.

La ventaja de usar kernel es que no se necesita usar el espacio de características de mayor dimensión. A esta técnica se le conoce como “truco de kernel” (*kernel trick*), es decir, se usa $H(\boldsymbol{x}, \boldsymbol{x}')$ para entrenamiento y clasificación en lugar de $g(x)$, como se muestra a continuación.

Usando el kernel, el problema dual en el espacio de características está dado de la siguiente manera:

Maximizar:

$$L_D = \sum_{\mu=1}^{N} \lambda^{\mu} - \frac{1}{2} \sum_{\mu,j=1}^{N} \lambda^{\mu} \lambda^{j} c^{\mu} c^{j} H(\boldsymbol{x}^{\mu}, \mathbf{x}^{j}) \tag{53}$$

sujeto a:

$$\sum_{\mu=1}^{N} \lambda^{\mu} C^{\mu} = 0$$
$$0 \leq \lambda^{\mu} \leq P \tag{54}$$

Como $H(\boldsymbol{x}, \boldsymbol{x}')$ es un kernel semidefinido positivo, el problema de optimización es un problema de programación cuadrática cóncava. Y debido a que $\alpha = 0$ es una solución factible, el problema tiene una solución óptima global.

Las condiciones complementarias de KKT están dadas por:

$$\alpha^{\mu} \left(c^{\mu} \left(\sum_{j=1}^{N} C^{j} \lambda^{j} H(x_{\mu}, x_{j}) + b \right) - 1 - \xi^{\mu} \right) = 0 \tag{55}$$

para $\mu = 1, \ldots, N.$

$$(P - \alpha^{\mu}) \xi^{\mu} = 0 \quad \text{para} \quad \mu = 1, \ldots, N. \tag{56}$$

$$\alpha^{\mu} \geq 0, \ \xi^{\mu} \geq 0 \quad \text{para} \quad \mu = 1, \ldots, N. \tag{57}$$

La función de decisión está dada por:

$$L_D = \sum_{\mu \in N} \lambda^{\mu} C^{\mu} H(x^{\mu}, x) + b \tag{58}$$

si $L_D = 0, x$ no puede ser clasificado.

3.1.3.1. Tipos de kernel

Los tipos de kernel más utilizados en las SVM para la clasificación de datos son:

- Kernel lineal, también llamado kernel polinomial homogéneo se puede definir como:

$$K(x_i, x_j) = (x_i, x_j)^d \tag{59}$$

- Kernel Polinomial no homogéneo se define como:

$$k(x_i, x_j) = (x_i \cdot x_j + c)^d \tag{60}$$

- Kernel Gaussiano o base radial está definido como:

$$k(x_i, x_j) = exp\left(-\frac{(x_i - x_j)^2}{2\sigma^2}\right) \tag{61}$$

- Kernel tangente hiperbólica:

$$k(x_i, x_j) = \tanh\left(k(x_i \cdot x_j) + c\right) \tag{62}$$

3.1.4. Caso Multiclase

Las SVM son creadas para problemas de dos clases. Dado que las SVM emplean funciones de decisión directa, una extensión para SVM multiclase no es sencilla, hay aproximadamente cuatro tipos de SVM que manejan problemas multiclase:

- SVM uno contra todos.
- SVM por pares.
- SVM de código de salida de corrección de errores (ECOC, *error correcting output codes*).
- SVM todo junto.

De las cuatro mencionadas, esta investigación se centra únicamente en dos de ellas, SVM uno contra todos y SVM en pares.

De acuerdo con Vapnik [53] en las SVM uno contra todos, un problema de clase n se convierte en n problemas de dos clases y para el i-ésimo problema de dos clases, la clase i se separa de las clases restantes. Sin embargo, para este tipo de clasificación existen regiones inclasificables si se utilizan las funciones de decisión discretas.

Para dar solución a este problema de SVM por pares, Krebel [54] convierte el problema de n clases en problemas de dos clases $n(n-1)/2$, que cubre todos los pares de clases, sin embargo, aun con este método existen regiones linealmente no separables.

Se pueden resolver estas regiones linealmente no separables, introduciendo árboles de decisión [55], [56], [57], [58], ECOC [59] o determinando las funciones de decisión para cada una [60], [61].

3.1.4.1. SVM uno contra todos

Considerando un problema de n-clases, para una SVM uno contra todos, se determinan funciones de decisión directa que separan una clase de las clases restantes. Sea la i-ésima función de decisión, con el margen máximo que separa la clase i de las clases restantes:

$$D(x) = \mathbf{w}_i^T g_i(x) + b_i \tag{63}$$

donde $\mathbf{w}_i$ es un vector n-dimensional, $g_i(x)$ es la función que asigna x al espacio de características n-dimensional y b_i es el termino bias.

El hiperplano $D_i(x) = 0$ forma el hiperplano de separación óptimo, y si el problema de clasificación es separable, los datos de entrenamiento pertenecientes a la clase i satisfacen $D_i \geq 1$ y los pertenecientes a las clases restantes satisfacen $D_i \leq -1$. Especialmente, los vectores de soporte satisfacen $y_i D_i = 1$. Si el problema es linealmente no separable, los vectores no acotados satisfacen a $y_i D_i = 1$ y los vectores de soporte acotados satisfacen $y_i D_i \leq 1$. Los datos de entrenamiento restantes satisfacen $y_i D_i \geq 1$.

En clasificación, si para el vector x:

$$D_i(x) > 0 \tag{64}$$

se satisface para un i, x se clasifica en clase i. Debido a que sólo se usa el signo de la función de decisión, la decisión es discreta.

Si la ecuación (64) satisface para todo i o si no hay un i que no satisface la ecuación entonces x no es clasificable. Considerando el problema de tres clases con entrada bidimensional, donde las flechas muestran el lado positivo de los hiperplanos (véase Figura 3.4). Para el dato 1, x_1, las tres funciones de decisión son:

$$D_1(x) > 0, \qquad D_2(x) > 0, \qquad D_3(x) < 0 \tag{65}$$

donde x_1 pertenece a las clases 1 y 2, x_1 es un dato que no pude ser clasificado, del mismo modo, para el dato 2, x_2, las tres funciones de decisión son:

$$D_1(x) < 0, \qquad D_2(x) < 0, \qquad D_3(x) < 0 \tag{66}$$

así, x_2 no puede ser clasificado.

Para evitar esto, en lugar de funciones de decisión discretas, se proponen funciones de decisión continuas para su clasificación. Es decir, el dato x se clasifica en la clase:

$$\arg\max D_i(x), \qquad i = 1, \dots, N \tag{67}$$

Entonces el dato 1 de la Figura 3.4 se muestra clasificado dentro de la clase 1 ya que $D_1(x_1)$ es el más cercano de los 3. Del mismo modo, el dato 2 se clasifica en la clase 1.

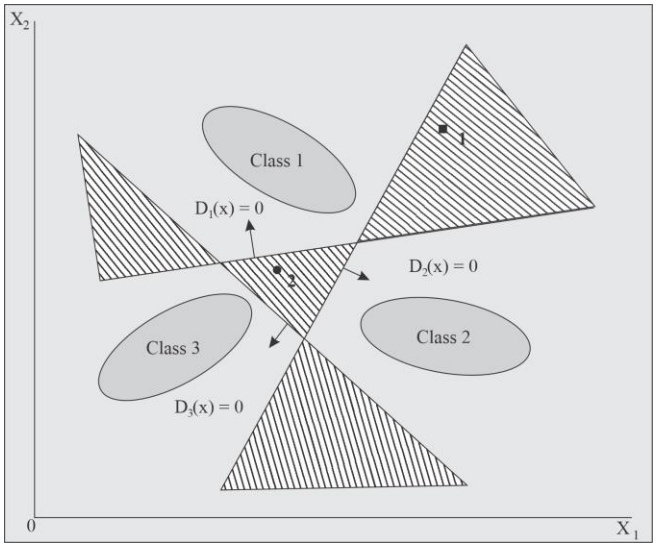

Figura 3.4. Regiones linealmente no separables con la clasificación uno contra todos.

3.1.4.2. SVM por pares

En este método de clasificación SVM por pares, se entrena un clasificador para cada posible par de clases. Para n clases, los clasificadores binarios se calculan $n(n-1)/2$. Este número, suele ser mayor que el número de clasificadores uno contra todos, por ejemplo, si $n = 10$, se requiere entrenar a 45 clasificadores binarios en lugar de 10 como en el método anterior. Aunque esto sugiere más tiempo de entrenamiento, los problemas individuales que se necesitan entrenar son significativamente más pequeños y si el algoritmo de entrenamiento es escalado de forma lineal con el tamaño del conjunto de entrenamiento, en realidad es posible ahorrar tiempo.

Los clasificadores individuales, sin embargo, son generalmente de menor tamaño y tienen menos vectores de soporte (SV) de los que serían en el enfoque de uno contra el resto. Esto se debe a dos razones: en primer lugar, los conjuntos de entrenamiento son más pequeños y, en segundo lugar, los problemas a aprender suelen ser más fáciles ya que las clases se superponen menos [54].

Sin embargo, si n es demasiado grande y se evalúan los clasificadores en $n(n-1)/2$, entonces el sistema resultante puede ser más lento que el SVM correspondiente de uno contra todos.

En conjunto con este tipo de clasificación multiclase, se consideró el kernel polinomial para hacer en conjunto el entrenamiento y la clasificación.

3.1.5. Kernel polinomial no homogéneo

La construcción de nuevas características es computacionalmente muy costosa, especialmente si se procesan datos de mayor dimensionalidad, el kernel es una sustitución del producto escalar de las SVM convencionales por el producto escalar de la función de mapeo. El kernel polinomial es una función comúnmente usada con las SVM que representa la similitud de vectores de entrenamiento, en un espacio de características sobre polinomios de las variables originales, lo que permite el aprendizaje de modelos no lineales.

3.1.5.1. Definición del kernel

Para un polinomio de grado d, el kernel polinomial está dado por:

$$K(x, y) = (x^T y + c)^d \tag{68}$$

donde x e y son vectores de entrada, es decir, vectores de características a partir de las muestras de entrenamiento o prueba, y $c \geq 0$ es un parámetro libre que compensa los términos superiores e inferiores, cuando $c = 0$ se le denomina kernel homogéneo [53].

Para un kernel polinomial la nueva dimensionalidad del espacio de mapeo está dada por [54]:

$$C(n + d, d) = \frac{(n + d)(n + d - 1) \dots (n + 1)}{d!} \tag{69}$$

donde n es el número de características y d es el grado del polinomio.

Como el kernel corresponde a un producto escalar en el espacio de características basado en algún mapeo ø, se tiene que:

$$K(x, y) = \langle \emptyset(x), \emptyset(y) \rangle \tag{70}$$

Un ejemplo del comportamiento del kernel es el siguiente. Sea $d = 2$, se obtiene el siguiente resultado.

$$K(x, y) = \left(\sum_{i=1}^{n} x_i y_i + c \right)^2$$

$$= \sum_{i}^{n} (x_i^2)(y_i^2) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} \left(\sqrt{2} x_i x_j \right) \left(\sqrt{2} y_i y_j \right) \tag{71}$$

$$+ \sum_{i}^{n} \left(\sqrt{2c} x_i \right) \left(\sqrt{2c} y_i \right) + c^2$$

reagrupando términos, el nuevo mapa de características queda de la siguiente forma:

$$\emptyset(x) = \begin{matrix} x_n^2, \dots, x_1^2, \sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \dots, \sqrt{2}x_{n-1}x_1, \dots, \sqrt{2}x_2 x_1, \\ \dots, \sqrt{2c}x_n \dots, \sqrt{2c}x_1, c \end{matrix} \quad (72)$$

donde $\emptyset(x)$ es el nuevo vector de características de mayor dimensionalidad.

3.2. Modelado Conceptual

El modelado conceptual se considera una fase del proceso de diseño de una arquitectura hardware y es una representación de un sistema descrito utilizando el paradigma secuencial mediante un lenguaje de software. El modelado conceptual permite identificar las fases que integran al sistema y se puede utilizar como base para el diseño de una arquitectura hardware. La finalidad de dicho modelado es identificar y facilitar la definición de los elementos de hardware requeridos en el diseño de la arquitectura, permite definir y planificar las técnicas adecuadas para su implementación en un dispositivo especifico.

Tomando como referencia los métodos y ecuaciones descritas en el Apartado 3.1.1, se implementa el algoritmo SVM siguiendo un enfoque secuencial mediante funciones, las cuales son invocadas por el programa principal. Para la implementación son necesarios los vectores de entrada, los cuales son cargados mediante una función, estos vectores son necesarios para calcular la distancia menor entre ellos; con la finalidad de encontrar los vectores de soporte, los cuales son utilizados para el cálculo de los multiplicadores de Lagrange. Una vez calculados los multiplicadores, se puede calcular el vector **w**, que permite maximizar el margen del hiperplano de separación. En la Figura 3.5 se muestra el diagrama general del entrenamiento se la SVM.

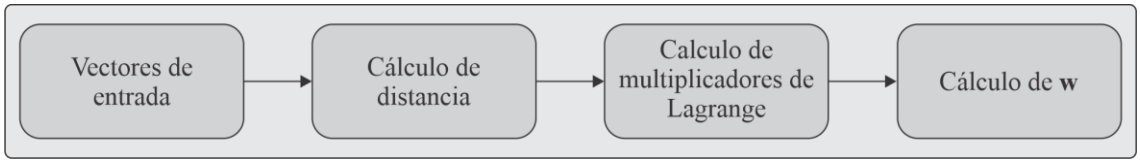

Figura 3.5. Diagrama general del entrenamiento de la SVM.

Las pruebas que se realizan al sistema tanto en el modelado conceptual como en la implementación en hardware son: a) Para el caso linealmente separable, la compuerta AND, b) para el caso linealmente no separable, la compuerta XOR y c) para el caso multiclase, el set de datos de la Planta Iris.

3.2.1. Interfaz de Usuario

Esta sección describe la funcionalidad de la **GUI** diseñada especialmente para realizar las evaluaciones correspondientes al modelado secuencial de la SVM y su posterior implementación en Hardware; esta **GUI** es desarrollada en un lenguaje de alto nivel debido a que éste permite flexibilidad para cada uno de los problemas para la SVM. Para comunicar la **GUI** y la implementación en hardware del clasificador SVM, se utiliza un convertidor **USB-UART** que además permite que aplicaciones de PC puedan comunicarse con la Nexys 3 por medio del dispositivo **DLP-FT232RQ**.

Las funciones de la **GUI** son las siguientes:

- Entrenamiento y definición de la estructura de la SVM. En este trabajo de investigación el aprendizaje del modelo SVM se lleva a cabo por medio de la **GUI**. Por otro lado, debido a que el modelado de la función de clasificación del SVM se implementa en el FPGA, el usuario debe hacer uso de la **GUI** para definir la estructura del clasificador mediante el envío y/o actualización del vector **w.**
- Prueba de operación del SVM. Definida la estructura de la SVM, el desempeño del **Clasificador SVM** implementado en el FPGA puede ser analizado mediante la función de "envío de patrones" ofrecida al usuario por la **GUI.**
- Recepción y visualización del resultado. En cuanto la SVM ha terminado de procesar el patrón de entrada, envía el resultado a la GUI, cuya función es recibirla y mostrarla al usuario para su interpretación.

Para el intercambio de información entre la GUI y el sistema embebido, se establece un protocolo de comunicación. La finalidad de este protocolo es permitir la transferencia de datos eficaz y segura, además de ofrecer una forma de distinguir la información que se transmite y abstraer al usuario de los procesos de envío y recepción de datos.

La implementación de las SVM en la GUI se compone de las siguientes fases:

- Implementación del algoritmo de aprendizaje de la SVM.
- Fase de pruebas.

La GUI calcula el vector **w** para enviarlo al FPGA junto a los nuevos patrones de clasificación.

3.2.2. Caso linealmente separable

3.2.2.1. Funciones para cargar el vector de entrenamiento

La función encargada de cargar los vectores de entrenamiento se llama ini_pat(), la cual está previamente descrita dentro del programa principal y los vectores se describen en una matriz con nombre X, donde son 4 vectores de 2 características; se añade el bias para hacer que la matriz sea de tamaño 4 x 3. En esta función se añade la etiqueta asociada a cada vector en un arreglo de 4 dimensiones d. La Tabla 3.1 muestra el pseudocódigo de dicha función.

Tabla 3.1. Implementación en pseudocódigo del patrón a clasificar.

```
00| Función ini_pat()
01| Variables
02|    Global X[][], d[]
03| Inicio
04|    X[0][0] =0   X[0][1] =0   X[0][2] =1   d[0]=-1
05|    X[1][0] =0   X[1][1] =1   X[1][2] =1   d[1]=-1
06|    X[2][0] =1   X[2][1] =0   X[2][2] =1   d[2]=-1
07|    X[3][0] =1   X[3][1] =1   X[3][2] =1   d[3]= 1
08| Fin
```

3.2.2.2. Obtención de la distancia de los vectores

La función que realiza el cálculo de la distancia es denominada distancia_e(). El parámetro de entrada de esta función son los vectores previamente cargados, el parámetro de salida es el cálculo de la distancia euclidiana entre los vectores; los vectores de menor distancia serán seleccionados como vectores de soporte, los cuales ayudan a calcular los multiplicadores de Lagrange. El pseudocódigo de la función descrita se muestra en la Tabla 3.2.

Tabla 3.2. Implementación del cálculo de la distancia en pseudocódigo.

```
00| Función distancia_e()
01| Variables
02|    Global Float trainData
03|    Int p1, p2
04| Inicio
05|    Regresa (√((trainData[p1][0] − trainData[p2][0])² + (trainData[p1][1] − trainData[p2][1])²))
06| Fin
```

El resultado obtenido en la función distancia_e() se utiliza para comparar las distancias y separar los vectores de soporte que se utilizarán para calcular los multiplicadores de Lagrange.

3.2.2.3. Cálculo de los multiplicadores de Lagrange y vector w.

Para encontrar los multiplicadores de Lagrange se considera la ecuación (25) con los vectores encontrados y se resuelve el sistema de ecuaciones correspondiente, para posteriormente calcular el vector $\boldsymbol{w}$. El pseudocódigo se presenta en la Tabla 3.3.

*Tabla 3.3. Implementación para encontrar multiplicadores de Lagrange y vector **w** en pseudocódigo.*

```
00| Función vector_support()
01| Variables
02|    Global trainData
03|    Float A1[][], A2[][], A3[][], A1T[][], A2T[][], A3T[][]
04|    Float alpha1, alpha2, alpha3
05|    Int i, j pos_c1[], pos_c2[] pos_1, pos_2 mini[][]
06| Inicio
07|    para(i=0 hasta pos_1)
08|       para(j=0 hasta pos_2)
09|          sí(distancia(pos_c1[i], pos_c2[j]) <= distancia(mini[0][0], mini[0][1])
10|             mini[2][0] = mini[1][0]
11|             mini[2][1] = mini[1][1]
12|             mini[1][0] = mini[0][0]
13|             mini[1][1] = mini[0][1]
14|             mini[0][0] = pos_c1[i]
15|             mini[0][1] = pos_c2[j]
16|    //transpuesta de los vectores
17|    para (i=0 hasta 4)
18|       A1T[i][0]=A1[0][i]
19|       A2T[i][0]=A2[0][i]
20|       A3T[i][0]=A3[0][i]
21|    para (i=0 hasta 4)
22|       C11 += A1T[j][0] * A1[0][j]
23|       C12 += A2T[j][0] * A1[0][j]
24|       C13 += A3T[j][0] * A1[0][j]
25|       C21 += A1T[j][0] * A2[0][j]
26|       C22 += A2T[j][0] * A2[0][j]
27|       C23 += A3T[j][0] * A2[0][j]
28|       C31 += A1T[j][0] * A3[0][j]
29|       C32 += A2T[j][0] * A3[0][j]
30|       C33 += A3T[j][0] * A3[0][j]
31|    detg = (C11 * ((C22 * C33) - (C23 * C32))) - (C12 * ((C21 * C33) – (C23 * C31))) +
32|           (C13 * ((C21 * C32) - (C22 * C31)))
33|    detx = (b1 * ((C22 * C33) - (C23 * C32))) - (C12 * ((b2 * C33) - (C23 * b3))) +
34|            (C13 * ((b2 * C32) - (C22 * b3)))
35|    dety = (C11 * ((b2 * C33) - (C23 * b3))) - (b1 * ((C21 * C33) - (C23 * C31))) +
```

```
36|            (C13 * ((C21 * b3) - (b2 * C31)))
37|    detz = (C11 * ((C22 * b3) - (b2 * C32))) - (C12 * ((C21 * b3) - (b2 * C31))) +
38|           (b1 * ((C21 * C32) - (C22 * C31)))
39|    sí(detg != 0) entonces
40|       alpha1 = detx / detg;
41|       alpha2 = dety / detg;
42|       alpha3 = detz / detg
43|    para (i=0 hasta 4)
44|       w[i]= alpha1*A1[i]+alpha[2]*A2[i]+alpha3*A3[i]
45|    regresa w
46| fin
```

El resultado que se obtiene al calcular el vector se utiliza para la función de decisión, esta función determina la clase a la que pertenecen los nuevos vectores que se le presentan a la SVM.

3.2.2.4. Función de decisión

La función de decisión, representada en la ecuación (29), permite conocer a la clase a la cual pertenece un nuevo vector de entrada, esta función recibe el vector ***w*** para hacer el procesamiento de clasificación. En la Tabla 3.4 se muestra su pseudocódigo.

Tabla 3.4. Implementación de la función test_svm() en pseudocódigo.

```
01| Función test_svm(float w)
02| Variables
03|    Global testDataK[]
04|    Float a[]
05|    Int i, j
05| Inicio
06|    para (i=0 hasta 4)
07|       para (j=0 hasta 3)
08|          a[i]=a[i]+(w[j] *testDatak[i][j])
09|    para (i=0 hasta 4)
10|       sí (a[i] <= 0) entonces
11|          a[i]=-1.0
12|       sí no
13|          a[i]=1.0
14|       fin sí
15|    regresa a
16| fin
```

3.2.2.5. Resultado del caso linealmente separable

Para probar el desempeño de este caso se utiliza la compuerta AND con el patrón de la Tabla 3.5.

Tabla 3.5. Conjunto de entrenamiento de la compuerta AND.

Patrón	Descriptor 1	Descriptor 2	Valor Deseado
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

Con la finalidad de determinar el desempeño de la SVM para el problema correspondiente, la GUI utiliza los patrones de entrenamiento de la compuerta AND trasladadas a un sistema de coordenadas en un plano cartesiano como muestra la Figura 3.6.

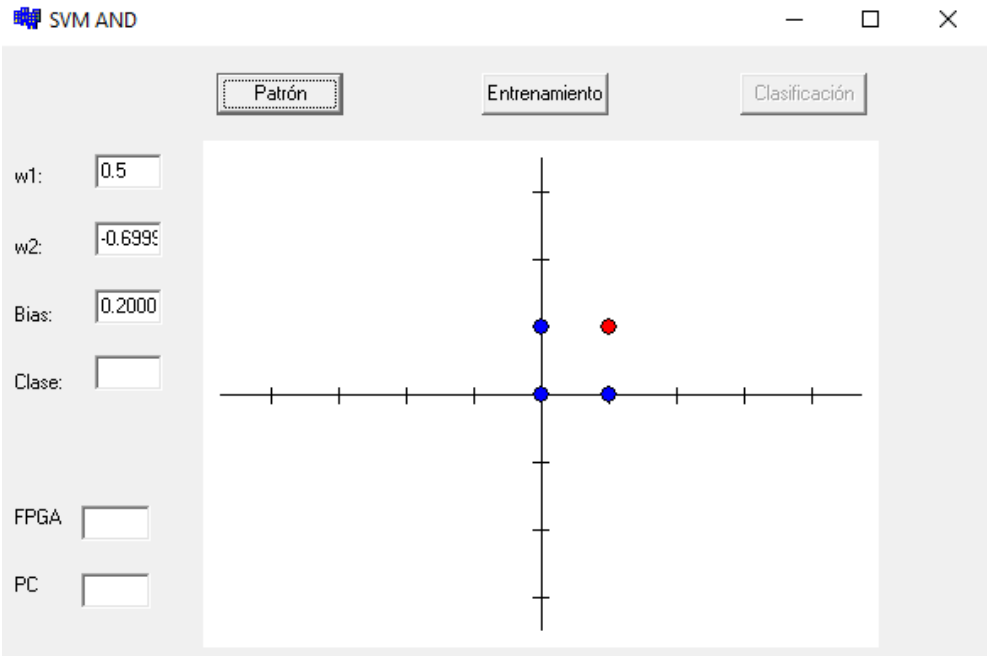

Figura 3.6. Patrones de entrenamiento caso linealmente no separable.

Posteriormente, se realiza el proceso de entrenamiento para determinar el hiperplano óptimo de separación de las clases, como se muestra en la Figura 3.7; para este caso el hiperplano de separación se dibuja formando la distancia máxima entre los vectores de soporte.

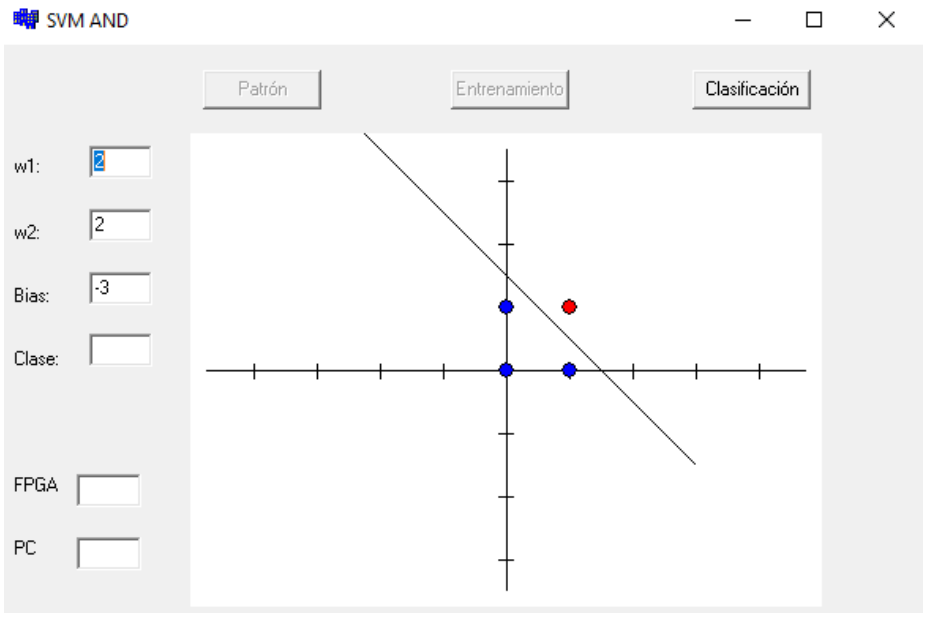

Figura 3.7. Hiperplano óptimo de separación.

Encontrado el hiperplano óptimo de separación de las clases, se envían nuevos datos diferentes para su clasificación; los resultados se pueden ver en la Figura 3.8.

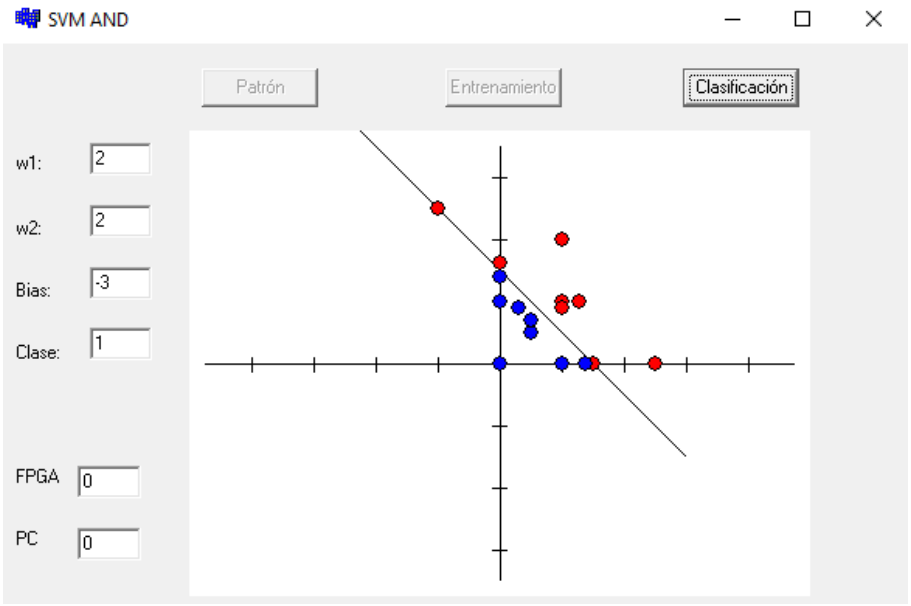

Figura 3.8. Nuevos datos clasificados por la SVM.

3.2.3. Caso linealmente no separable

Las pruebas realizadas en el caso linealmente no separable contemplan el problema XOR. Ahora, tomando como referencia las funciones descritas en el caso linealmente separable, las funciones son reutilizadas y sólo se describe el kernel utilizado para el caso linealmente no separable.

3.2.3.1. Modelado conceptual para el kernel lineal

El kernel usado para la clasificación linealmente no separable para dos clases en su descripción matemática es:

$$K(x^1, x^2) = (x^1, x^1 \cdot x^2) \tag{73}$$

El modelado conceptual del kernel lineal para la clasificación de dos clases linealmente no separables se describe en la función Linear_kernel(), este kernel es una transformación lineal de dos dimensiones a dos dimensiones, en la Tabla 3.6 se muestra su pseudocódigo.

Tabla 3.6. Implementación de la función Linear_kernel() en pseudocódigo.

```
01| Function Linear_Kernel(Float p1, Float p2)
02| Inicio
03|   return(p1*p2)
04|Fin
```

3.2.3.2. Resultado del caso linealmente no separable

Considerando las funciones descritas en los apartados 3.2.2.1, 3.2.2.2 y 3.2.2.3, se añade el kernel para la clasificación del caso descrito y con la finalidad de determinar el desempeño de la SVM se utiliza el caso del problema XOR.

El sistema linealmente no separable clasifica la compuerta XOR el cual se describe en la Tabla 3.7.

Tabla 3.7. Conjunto de entrenamiento XOR.

Patrón	Descriptor 1	Descriptor 2	Valor deseado
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Para evaluar la SVM con la compuerta descrita, se pasa el problema a un sistema de coordenadas para poder evaluarlo, éste se muestra en la Figura 3.9, los valores mostrados son aleatorios para el vector $\boldsymbol{w}$ y el bias.

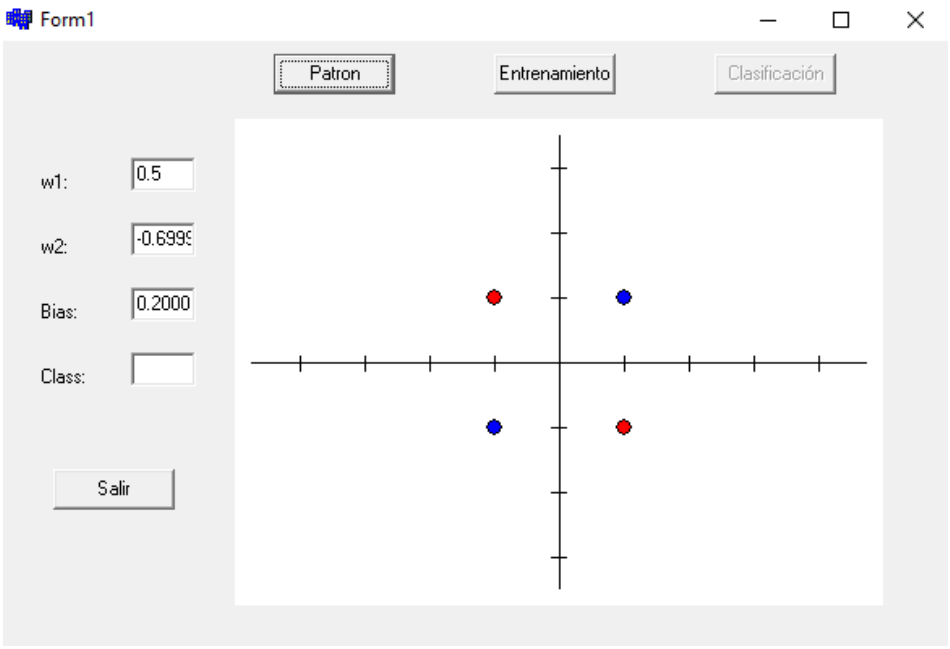

Figura 3.9. Patrón de la XOR para clasificar.

Ahora se debe aplicar el kernel descrito en la Ecuación (73), cuyos resultados son mostrados en la Figura 3.10 y donde se aprecia que el problema XOR ha sido convertido a un caso linealmente separable. Posteriormente, se realiza el proceso de entrenamiento, la Figura 3.10 se muestra el hiperplano óptimo de separación calculado.

Una vez encontrado el hiperplano óptimo se pueden agregar nuevos datos para su clasificación, en la Figura 3.11 se muestra un ejemplo de los datos que se mandaron para su clasificación. En la Tabla 3.8 se muestra un ejemplo de datos de entrada antes de aplicar el kernel y los resultados obtenidos después del kernel.

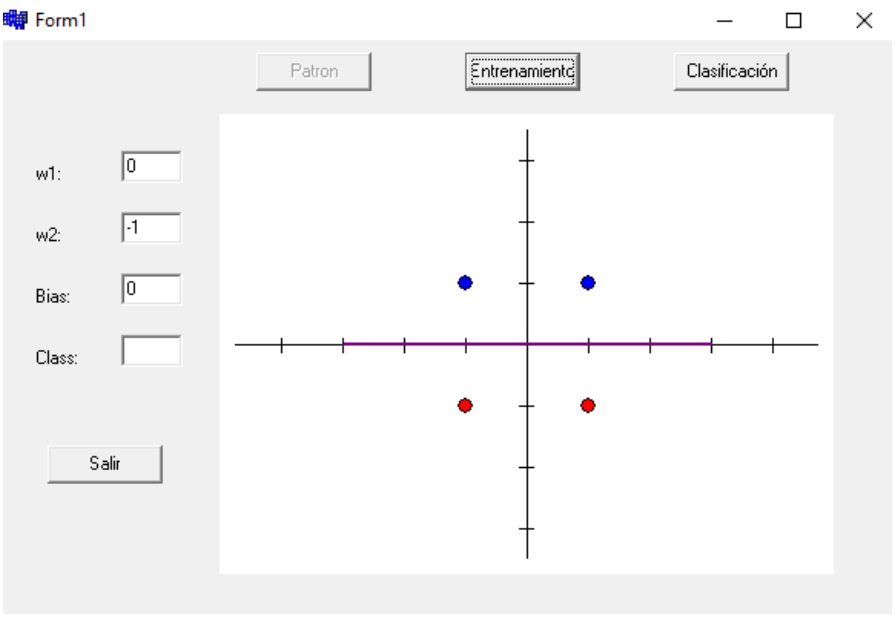

Figura 3.10. Entrenamiento de la SVM con el hiperplano calculado.

Tabla 3.8. Ejemplo de datos de clasificación antes y después de la transformación de datos.

Datos antes del kernel		Datos después de aplicar el kernel	
1.5	.5	1.5	0.75
-.5	1.5	-.5	-0.75
1	-.5	1	-.5
-1	-1.5	-1	1.5

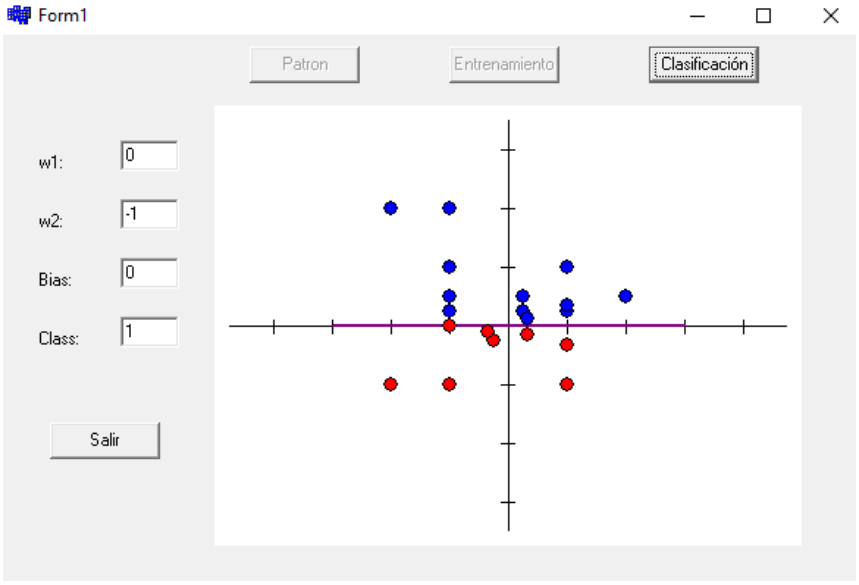

Figura 3.11. Datos nuevos para su clasificación.

3.2.4. Caso multiclase

Tomando como referencia las funciones descritas en el caso linealmente separable, se reutilizan las funciones y sólo se actualiza el kernel polinomial para el caso multiclase. Considerando que la dimensionalidad de los nuevos vectores y el número de vectores, también se actualiza la función que carga los vectores de entrada y se redimensionan las variables usadas.

3.2.4.1. Funciones para cargar los vectores para caso multiclase

Las funciones loadtrain() y loadtest() se encargan de cargar los vectores de entrenamiento y de prueba desde un archivo de texto, depositándolos en dos matrices para poder ser ocupados por las funciones del programa; estas matrices tienen por nombre trainData y testData, la primera consta de 75 vectores con una dimensión de 4 características cada vector, la segunda es de 150 vectores con 4 características cada vector, el cuerpo de la función es la misma para ambas funciones, por tal motivo sólo se describe una función en el pseudocódigo que puede consultarse en la Tabla 3.9.

Tabla 3.9. Implementación para cargar los vectores para entrenamiento en pseudocódigo.

```
00| Función loadTrain()
01| Variables
02|    Global trainData[][], trainL
03|    String line
04|    Int i, j, count
05|    Float x
06| Inicio
07|    Ifstream("trainData.txt")
08|    mientras(Getline(in, line))
09|       strigstream ss(line)
10|       mientras(ss>>x)
11|          count=count+1
12|          sí (count<5) entonces
13|             trainData[i][j] =x;
14|             j=j+1
15|          sí no
16|             trainL[i]=int(x)
17|             i=i+1
18|             j=0
19|             count=0
20| Fin
```

3.2.4.2. Modelado conceptual del kernel polinomial

En la sección 3.1.5 se presenta la descripción matemática del kernel polinomial y en esta sección se describe el modelado conceptual del kernel. La función encargada de realizar el cálculo del kernel se muestra en la Tabla 3.10.

Tabla 3.10. Implementación de la función kernel_poly() en pseudocódigo.

```
01| Función Kernel_poly()
02| Variables
03|    Global TrainDT
04|    Int i=0
05| Inicio
06|    para (i=0 hasta 150)
07|       TrainDT[i][0] = (float)pow(trainData[i][0], 2);
08|       TrainDT[i][1] = (float)pow(trainData[i][1], 2);
09|       TrainDT[i][2] = (float)pow(trainData[i][2], 2);
10|       TrainDT[i][3] = (float)pow(trainData[i][3], 2);
11|       TrainDT[i][4] = (float)sqrt(2) * trainData[i][3] * trainData[i][2];
12|       TrainDT[i][5] = (float)sqrt(2) * trainData[i][3] * trainData[i][1];
13|       TrainDT[i][6] = (float)sqrt(2) * trainData[i][3] * trainData[i][0];
14|       TrainDT[i][7] = (float)sqrt(2) * trainData[i][2] * trainData[i][1];
15|       TrainDT[i][8] = (float)sqrt(2) * trainData[i][2] * trainData[i][0];
16|       TrainDT[i][9] = (float)sqrt(2) * trainData[i][1] * trainData[i][0];
17|       TrainDT[i][10] = (float)sqrt(2) * trainData[i][3];
18|       TrainDT[i][11] = (float)sqrt(2) * trainData[i][2];
19|       TrainDT[i][12] = (float)sqrt(2) * trainData[i][1];
20|       TrainDT[i][13] = (float)sqrt(2) * trainData[i][0];
21|       TrainDT[i][14] = 1.0;
22|    regresa
23| Fin
```

3.2.4.3. Función de decisión del caso multiclase

La función de decisión permite discriminar entre clases, cuyas regiones de decisión pueden separarse mediante una única condición lineal o hiperplano, que se realiza a través de la ecuación $C^{\mu}(\langle \boldsymbol{w}, \boldsymbol{x}^{\mu} \rangle + b) \geq 0$ y que describe el hiperplano de separación de la SVM. Lo anterior representa un discriminador lineal, separando las clases en las regiones que representan dos clases con diferentes patrones. Esta misma función permite clasificar un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías; un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo, cuya categoría se desconoce, pertenece a una categoría o a otra.

La función de decisión encargada del procesamiento de datos nuevos se expresa a continuación en la siguiente Tabla 3.11.

Tabla 3.11. Implementación de la función test_svm() en pseudocódigo.

```
01| Función float* test_svm(float wA[16])
02| Variables
03|    Global testDatak, a
04|    Int i=0, j=0;
05| Inicio
06|    para (i=0 hasta 150)
07|       para (j=0 hasta 16)
08|          a[i] = a[i]+(wA[j]*testDatak[i][j])
09|       para (i = 0 hasta 150)
10|          Sí (a[i] <= 0)
11|             a[i] = -1.0;
12|          Sí no
13|             a[i] = 1.0;
14| regresa a;
15| Fin
```

3.2.4.4. Resultados del caso multiclase

Tomando en cuenta las funciones descritas en los apartados 3.2.2.2 y 3.2.2.3, y remplazando el kernel lineal por un polinomial se hace la siguiente clasificación multiclase utilizando el set de clasificación de la Planta Iris, este set de datos es propuesto por el botánico Ronald Fisher. La base de datos contiene tres clases:

- Iris Setosa
- Iris Virgínica.
- Iris Versicolor.

La base de datos de la planta Iris consiste de 50 ejemplos de cada una de las tres clases que la conforman. Cada ejemplo o patrón está compuesto por cuatro rasgos que corresponden a las medidas de largo y ancho de sépalo y largo y ancho de pétalo, dadas en centímetros. Una característica importante de esta base de datos es que dos de sus clases son linealmente separables y una no es linealmente separable. Cada patrón que representa una planta está formado por cuatro descriptores (véase Tabla 3.12).

Tabla 3.12. Descriptores de la Planta Iris.

Rango	Descripción(cm)
1	Largo del sépalo
2	Ancho del sépalo
3	Largo del pétalo
4	Ancho del pétalo

Para mostrar el desempeño del sistema propuesto, se hace uso de matrices de confusión. Una matriz de confusión es una tabla frecuentemente usada para describir el comportamiento que presenta un modelo de clasificación después de ser aplicado sobre un conjunto de datos de pruebas. La matriz de confusión agrupa los casos positivos que fueron clasificados correctamente conocidos como verdaderos positivos (TP, *True Positives*); los casos positivos que son clasificados incorrectos son llamados falsos positivos (FP, *False Positives*); los casos negativos clasificados correctamente son verdaderos negativos (TN, *True Negatives*); y los casos negativos que son clasificados incorrectamente son llamados, falsos negativos (FN, *False Negatives*). Esta matriz también permite conocer más métricas sobre el sistema:

- Exactitud (*accuracy*): Indica la fracción de observaciones correctamente clasificadas en el conjunto.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \tag{74}$$

- Tasa de error (*Te*): Indica qué tan frecuente el sistema se equivoca en sus predicciones.

$$Te = \frac{FP + FN}{TP + TN + FP + FN} \tag{75}$$

- Sensibilidad o Tasa de verdaderos positivos (TPR, *True Positives Rate*). Indica la fracción de observaciones positivas que han sido predichas correctamente.

$$TPR = \frac{TP}{TP + FN} \tag{76}$$

- Especificidad o Tasa de verdaderos negativos (TNR, *True Negatives Rate*). Indica la fracción de observaciones negativas que han sido correctamente clasificadas.

$$TNR = \frac{TN}{TN + FP} \tag{77}$$

- Precisión. Indica la fracción de predicciones positivas que en realidad son positivas y cuanto menor es la dispersión mayor la precisión.

$$Precisión = \frac{TP}{TP + FP} \tag{78}$$

Tomando en cuenta la matriz de confusión descrita en la Tabla 3.13 para las tres clases de la base de datos Iris, se pueden calcular las demás métricas utilizando las ecuaciones correspondientes y así obtener los parámetros de exactitud, tasa de error, tasa de errores verdaderos, tasa de errores negativos y precisión de cada una de las clases, para con ello obtener un promedio general (avg, *average*) de cada parámetro.

Tabla 3.13. Matriz de confusión del resultado de clasificación de la Planta Iris

	Predicciones del sistema conceptual		
Clases	**(a)**	**(b)**	**(c)**
Setosa	50	0	0
Versicolor	0	39	11
Virgínica	0	8	42

Los resultados obtenidos en la Tabla 3.14 brindan la información necesaria para la comprensión del comportamiento y desempeño del sistema.

Tabla 3.14. Métricas resultantes usando los datos de la matriz de confusión.

Clase\rate	**TP**	**TPR**	**TN**	**TNR**	**Te**	**Precisión**	**Exactitud**
Setosa	50	1	100	1	0	1	1
Versicolor	39	0.78	92	0.8932	0.1266	0.78	0.8733
Virgínica	42	0.84	89	0.9175	0.1266	0.84	0.8733
avg		0.8733		0.9369	0.0844	0.8733	0.9155

Capítulo 4. Arquitectura Hardware SVM

El presente capítulo describe el diseño y modelado de una arquitectura hardware de una SVM y muestra los resultados obtenidos de su implementación en lógica reconfigurable. El diseño de esta arquitectura utiliza como base el modelado conceptual descrito en el Capítulo 3, realizado mediante un enfoque modular.

4.1. Modelado SVM sobre Lógica Configurable

Un importante resultado de esta investigación es la obtención de un entorno de diseño de arquitecturas hardware de clasificadores SVM (AHC-SVM). Un punto clave de la mencionada arquitectura es que es diseñada siguiendo un enfoque modular. Este enfoque es un punto clave en el diseño de arquitecturas hardware que permite y/o facilita la interoperabilidad entre los elementos de la arquitectura, el rediseño de elementos individuales y mejora la fiabilidad del sistema. Además, genera un ambiente propicio para la experimentación y favorece el mantenimiento y evolución del diseño, lo que permite generar entornos de desarrollo de áreas específicas.

El entorno ofrece los componentes necesarios para implementar arquitecturas hardware de un modelo SVM que den solución a problemas linealmente separables, linealmente no separables y multiclase. Además, este entorno se complementa con el uso de

la **Interfaz de usuario** descrita en el modelo conceptual, a través de ésta el usuario pueda interactuar con la AHC-SVM, enviándole datos para su procesamiento o visualizando resultados generados por ella.

Además, el entorno se puede actualizar fácilmente, es decir, se pueden crear nuevos componentes, añadirlos al entorno y probar su desempeño. Por ejemplo, se puede generar una nueva forma de implementar la suma de las entradas ponderadas, un nuevo kernel y una nueva función de resolución de la clase, entre otros. Los componentes nuevos y aquellos desarrollados en esta investigación son considerados un repositorio que el usuario puede usar para crear diversos esquemas de AHC-SVM, esta característica favorece la evolución de la herramienta propuesta.

La Figura 4.1 muestra la estructura general de la herramienta propuesta, la cual está formada por dos sistemas, **Interfaz de Usuario** y **Clasificador SVM**. El sistema **Interfaz de Usuario** es un programa desarrollado en el lenguaje Builder C++, que modela una **GUI** que tiene como finalidad permitir al usuario interactuar con el sistema **Clasificador SVM**. El diseño de la **Interfaz de Usuario** y las funciones que desempeña están explicadas en el apartado 3.2.1 de este documento.

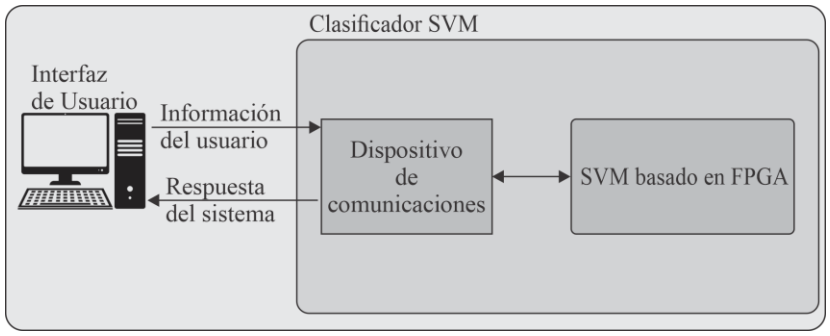

Figura 4.1. Estructura general de la herramienta propuesta.

El **Clasificador SVM** es un sistema de procesamiento basado en FPGA, dotándolo de la capacidad de diseñar y modelar arquitecturas hardware que exploten la concurrencia existente en un algoritmo o método que requiera ser implementado en lógica reconfigurable. El **Clasificador SVM** está integrado por dos componentes, **Dispositivo de comunicaciones** y **SVM basado en FPGA**.

El componente **Dispositivo de comunicaciones** es el circuito integrado DLP-FT232RQ de la compañía FTDI, que forma parte de la tarjeta de evaluación Nexys 3,

utilizada en el desarrollo de esta investigación. El DLP- FT232RQ opera como una pasarela (*Gateway*) USB-232 y permite la comunicación entre la **Interfaz de Usuario** y el **Clasificador SVM**.

Por su parte el componente **SVM basado en FPGA** es un dispositivo XC6LX16-CS324 de la familia Spartan-6 de la compañía Xilinx Inc., también incluido de la tarjeta Nexys 3 y representa al elemento central de procesamiento del **Clasificador SVM**. El componente **SVM basado en FPGA** está conformado por módulos de control y módulos que permiten la construcción de la SVM, descritos en los siguientes apartados.

4.2. Interfaz de Usuario

El diseño de la **Interfaz de Usuario** depende de la aplicación y deberá ser adaptada a las especificaciones de ésta. Con este fin, los aspectos a considerar son los parámetros de salida y entrada del **Clasificador SVM**. Para esto es necesario recordar que el entrenamiento de la SVM se realiza en la **Interfaz de Usuario**, por lo que el primer parámetro que la SVM recibe es el vector **w**, cuya función es definir la estructura de la SVM. Ahora, en modo de operación, la SVM recibe un vector de entrada que necesita ser clasificado. Finalmente, el resultado generado por la SVM que indica la clase a la que pertenece en vector introducido, debe ser enviado a la **Interfaz de Usuario**. Si el problema a resolver lo permite, la **Interfaz de Usuario** puede mostrar los resultados en forma gráfica (caso de 2 dimensiones) o en simple texto (caso de 3 o más dimensiones).

El apartado 3.2.1 documenta aspectos generales de este sistema y los incisos 0, 3.2.3.2 y 3.2.4.4 lo han hecho para cada ejemplo considerado para mostrar el desempeño del sistema propuesto. En el presente apartado sólo se explica el protocolo que ha sido definido para garantizar que la transferencia de información entre los sistemas **Interfaz de Usuario** y **Clasificador SVM** se lleve a cabo en forma correcta y abstrae al usuario del proceso general de envío-recepción de datos entre estos sistemas.

El protocolo propuesto está formado por tres tramas (denominadas Vector **w**, Vector de entrada y Clase de vector) y cada trama se constituye por tres campos: `ID`, `NumDat` y `Data`, como se muestra en la Figura 4.2. El campo `ID` representa el identificador de cada trama permitiendo diferenciarlas entre sí; el campo `NumDat` indica la cantidad de bytes que

incluye el campo Data y el campo Data contiene la información que será transferida entre la **Interfaz de Usuario** y el **Clasificador SVM**. La Tabla 4.1 presenta un resumen de las tramas incluidas en el mencionado protocolo, sus funciones y los valores de los campos que las forman.

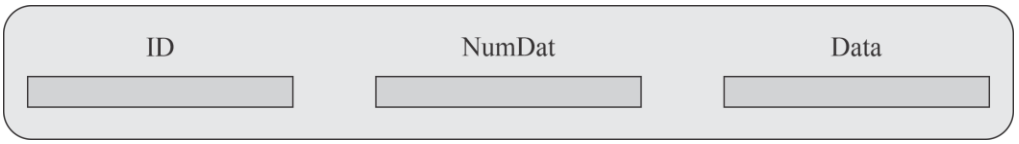

Figura 4.2. Estructura de las tramas del protocolo de comunicaciones.

Tabla 4.1. Resumen del protocolo de comunicaciones propuesto.

Nombre de la Trama	Función de la trama (dirección)	Valor de los campos			Observaciones
		ID (8 bits)	**Num_Dat** (8 bits)	**Data** (variable)	
Vector **w**	Definición de la estructura de la SVM (**Interfaz de usuario** a **Clasificador SVM**)	0x31 hasta 0x34	5 o 16	16	El campo ID de este vector puede cambiar dependiendo de la aplicación y número de clasificadores; el número de datos que se va a enviar va definido en el campo Num_Dat; cada elemento del campo Data está representado en punto fijo Q1.0.15
Vector de entrada	Envía un vector para su clasificación (**Interfaz de usuario** a **Clasificador SVM**)	0x35	4	8	El ID de este vector es fijo; el número de datos Num_Data se define en la interfaz de usuario; los elementos dentro del campo Data está representado en punto fijo Q1.3.4
Clase de vector	Envío de resultado del proceso de clasificación para su visualización gráfica (**Clasificador SVM** a **Interfaz de usuario**)	0x37	1	8	El campo ID de este vector es fijo; el número de datos Num_Data se define dentro del FPGA; los elementos dentro del campo Data está representado en punto fijo Q0.8.0

4.3. Clasificador SVM

Los circuitos digitales reconfigurables son dispositivos que permiten diseñar sistemas con un enfoque de hardware, organizados en un sólo circuito integrado para implementar aplicaciones de forma flexible y con la capacidad de reconfiguración. Debido a estas

características y a los avances de la tecnología, actualmente el uso de estos dispositivos se ha extendido ampliamente tanto en el ámbito académico como en el industrial. Como consecuencia, cada vez es más frecuente su uso en entornos software-hardware destinados al desarrollo de prototipos. Por estas razones, el entorno de diseño AHC-SVM generado en esta tesis es una herramienta orientada al desarrollo de clasificadores SVM sobre lógica reconfigurable, que integra, como uno de sus elementos principales, un sistema cuyo elemento central de procesamiento es un FPGA, denominado **Clasificador SVM**. Este sistema está conformado por los componentes denominados **Dispositivo de comunicaciones** y **SVM basado en FPGA**.

4.3.1. Dispositivo de comunicaciones

El **Dispositivo de comunicaciones** provee una transferencia de información robusta entre la **Interfaz de Usuario** y el **Clasificador SVM** mediante el protocolo USB. Este componente es un dispositivo DLP- FT232RQ, diseñado por la compañía FTDI, incluido en la tarjeta de evaluación Nexys 3. La estructura del DLP-FT232RQ incluye FIFOs independientes para los procesos de transmisión y recepción de datos, asegurando un alto rendimiento. Además, soporta velocidades de transferencia de datos desde 300 hasta 3 mega baudios. Los controladores del DLP-FT232RQ permiten que la **Interfaz de Usuario** lo identifique como un puerto COM virtual (VCP, *Virtual COM Port*). Por lo que, una vez instalados estos controladores, la **Interfaz de Usuario** puede utilizar simples comandos de acceso al puerto COM para producir tráfico de datos hacia/desde el **Clasificador SVM** a través del DLP-FT232RQ. Por otra parte, cuando el DLP-FT232RQ se configura en modo Interfaz-RS-232, el **Clasificador SVM** lo percibe como un dispositivo que responde al estándar RS-232.

4.3.2. SVM basado en FPGA

Es necesario mencionar que las operaciones que se utilizan en la implementación conceptual, descrita en el Capítulo 3, requieren el uso de aritmética de punto flotante, en precisión simple y doble. La descripción hardware de este tipo de aritmética genera una arquitectura que usa una gran cantidad de recursos. Con la finalidad de buscar reducir estos recursos se implementaron alternativas que no afectaran el rendimiento del sistema, por este

motivo en la implementación del componente **SVM basado en FPGA** se decidió utilizar aritmética de punto fijo. Además, también es importante hacer énfasis en que los módulos descritos en este apartado pueden o no formar parte de una estructura SVM construida mediante el AHC-SVM y que esto depende directamente del tipo de problema a resolver, sea linealmente separable, linealmente no separable o multiclase.

El componente **SVM basado en FPGA** es el elemento central de procesamiento del **Clasificador SVM**. El diseño de la arquitectura del componente **SVM basado en FPGA** fue realizado mediante un enfoque modular siguiendo la metodología *top-down*. Un enfoque modular garantiza la creación de un diseño orientado al re-uso, permitiendo que los módulos sean utilizados en diferentes proyectos sin necesidad de realizarles modificaciones, y favorece la evolución del sistema, ya que se le pueden agregar fácilmente nuevos módulos. Por otro lado, el modelado de los módulos resultantes del diseño del **SVM basado en FPGA** fue elaborado en el entorno de desarrollo ISE Foundation versión 14.7 mediante el lenguaje VHDL utilizando niveles de abstracción RTL (*Register, Transfer Level*) y algorítmico, lo que implica que el diseño generado sea portable. Finalmente, la implementación del **SVM basado en FPGA** se realizó en un dispositivo FPGA XC6LX16-CS324 de la familia Spartan-6 de la compañía Xilinx Inc.

El **SVM basado en FPGA** es un sistema síncrono que trabaja con una frecuencia de reloj de 100MHz y está integrado por los siguientes módulos: **Driver del dispositivo de comunicaciones**, **Unidad de control principal** y **Sistema modular SVM** (véase Figura 4.3).

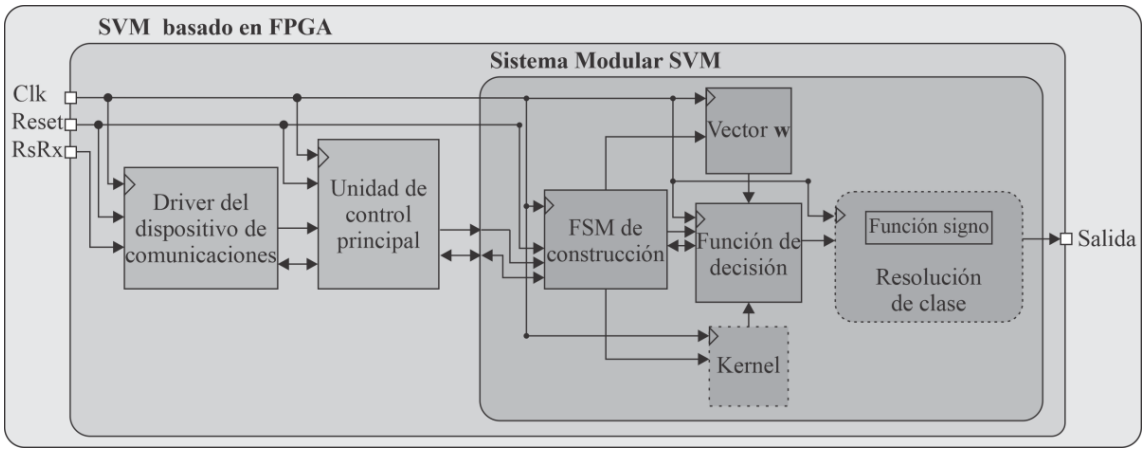

*Figura 4.3. Diagrama de bloques del **SVM basado en FPGA**.*

4.3.2.1. Driver del dispositivo de comunicaciones

Se ha mencionado que se utiliza el dispositivo DLP-FT232RQ para implementar una transferencia de información entre la **Interfaz de Usuario** y el **Clasificador SVM** vía el puerto USB. Desde que el DLP-FT232RQ se configura en modo Interfaz-232, el módulo **Driver del dispositivo de comunicaciones** describe los procesos de recepción y de transmisión (nombrados UART_RX y UART_TX en la Figura 4.4) del estándar RS-232 con las siguientes características: comunicación asíncrona, 8 bits de datos, 1 bit de paro y sin bit de paridad. Cada proceso se modeló independientemente mediante máquinas de estados finitos (FSM, *finite state machine*).

La Figura 4.4 muestra el diagrama de bloques del módulo **Driver del dispositivo de comunicaciones** y la Tabla 4.2 incluye una descripción de sus señales.

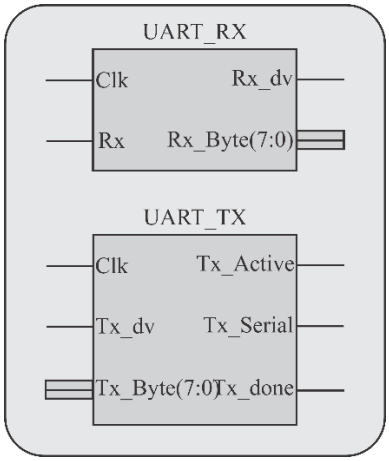

*Figura 4.4. Diagrama de bloques del módulo **Dispositivo de comunicaciones**.*

*Tabla 4.2. Descripción de las señales de entrada/salida del módulo **Driver de comunicaciones**.*

Proceso de recepción UART_RX	
Señal	**Descripción**
Clk	(Entrada) Señal de reloj del sistema, 100Mhz
Rx	(Entrada) Señal de recepción de datos RS232
Rx_dv	(Salida) Señal que indica cuando existe un dato disponible
Rx_Byte(7:0)	(Salida) Bus de datos que contiene la trama de datos recibida
Proceso de transmisión UART_TX	
Señal	**Descripción**
Clk	(Entrada) Señal de reloj del sistema, 100Mhz
Tx_dv	(Entrada) Señal que indica que existe un dato disponible para ser enviado
Tx_Byte(7:0)	(Entrada) Bus de datos resultante para ser enviado
Tx_Serial	(Salida) Señal que se envía bit a bit hasta completar el dato resultante
TX_Done	(Salida) Señal que notifica que el dato se ha enviado

4.3.2.2. Unidad de control principal

La **Unidad de control principal** es la responsable de iniciar los procesos de cada fase del **SVM basado en FPGA**. Una vez que el **Driver del dispositivo de comunicaciones** le asigna los datos, la **Unidad de control principal** verifica e indica en qué momento se define la estructura del clasificador por medio del vector **w**. Cuando la estructura está definida, identifica los nuevos datos a clasificar y habilita la FSM que permite esta acción (**FSM de construcción**).

La **Unidad de control principal** se implementó en el módulo **Ctrl_Principal** y se encarga de dirigir el flujo de datos que, dependiendo de la aplicación, permite definir la estructura de la SVM (véase apartado 3.2.1). Esta acción define cuántos clasificadores se van a utilizar en el proceso. Una vez definida la estructura, se encarga de dirigir los nuevos datos a clasificar y, una vez terminado el proceso de clasificación, permite el envío de resultados para su interpretación en la **Interfaz de Usuario**. En la Figura 4.5 se muestra el símbolo del módulo **Ctrl_Principal** y en la Tabla 4.3 se listan las señales que intervienen en su funcionamiento.

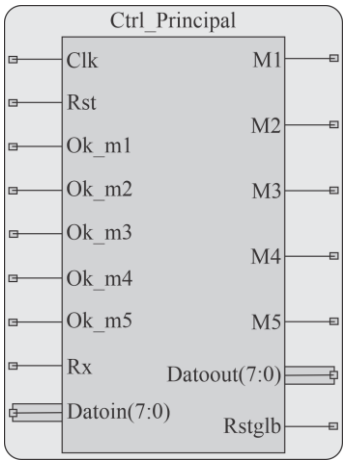

Figura 4.5. Símbolo del control principal descrito en el FPGA.

El módulo **Ctrl_Principal** basa su funcionamiento en la FSM que se muestra la Figura 4.6, la cual se describe a continuación.

*Tabla 4.3. Descripción de las señales Entrada/Salida de **Ctrl_principal**.*

Señal	Descripción
Clk	(Entrada) Señal del sistema 100Mhz
Rst	(Entrada) Señal de *reset*
Ok_m1, Ok_m2, Ok_m3, Ok_m4.	(Entrada) Señal de regreso de los módulos que controlan la estructura de los clasificadores, esta señal es activa cuando el vector **w** es inscrito en la **Sistema modular SVM**
Ok_m5	(Entrada) Señal de regreso del módulo **FSM de construcción** indicando que ha terminado de mandar los datos para la clasificación
Rx	(Entrada) Señal que avisa si existe un dato disponible en el buffer
Datoin(7:0)	(Entrada) Bus que indica que existe un nuevo dato
Ok1, Ok2, Ok3, Ok4.	(Salida) Señal de control, indica que los datos están listos para describir la arquitectura de la SVM
Ok5	(Salida) Señal de control que indica que hay datos nuevos para el proceso de clasificación
Datoout(7:0)	(Salida) Bus de salida para datos de descripción o clasificación
Rstglb	(Salida) Un *reset* global para asegurar que se inicializan las señales en cada nuevo dato que ingresa para ser clasificado

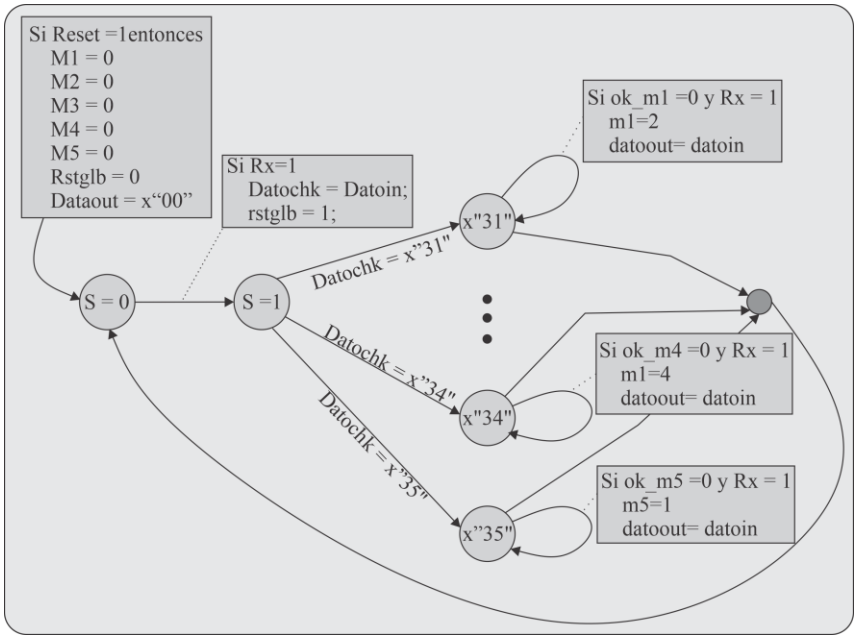

*Figura 4.6. Máquina de estados del módulo **Ctrl_principal**.*

Cuando el usuario activa la señal Rst, la FSM inicializa las señales de control. Una vez desactivada la señal Rst, comienza el proceso de espera de una trama proveniente de la **Interfaz de Usuario**. El campo ID de la trama del vector de entrada define si es un dato para describir la estructura de un clasificador o es un vector que se va a clasificar. Una vez definida la acción dirige el vector de entrada hacia el camino que corresponda dentro del componente **Sistema Modular SVM**. Si en el campo ID se recibe un número entre 0x31 a 0x34 es para

describir una estructura de la SVM esto depende de cuantos clasificadores se están usando y los datos son dirigidos hacia el vector **w**.

4.3.2.3. Sistema Modular SVM

El **Sistema Modular SVM** se encarga de guiar los datos de entrada para definir la estructura de uno o más clasificadores, también dirige a los nuevos datos de clasificación hacia el módulo correspondiente, ya sea al kernel o directamente a la función de decisión, y consta de los módulos **FSM de construcción**, **Kernel**, **Vector w**, **Función de decisión**, **Función signo** y **Resolución de clase**.

4.3.2.3.1. FSM de construcción

El módulo **FSM de construcción** se encuentra estructurado por tres FSM encargadas de concatenar los datos dentro del FPGA y dirigirlos hacia los módulos correspondientes, que pueden ser: **Kernel** (inciso 4.3.2.3.2**)** o **Vector W** (inciso 4.3.2.3.3) e iniciar la **Función de decisión** (inciso 4.3.2.3.4). En la Figura 4.7 se muestra el símbolo del **FSM de construcción** y en la Tabla 4.4 se listan las señales que hacen posible su operación

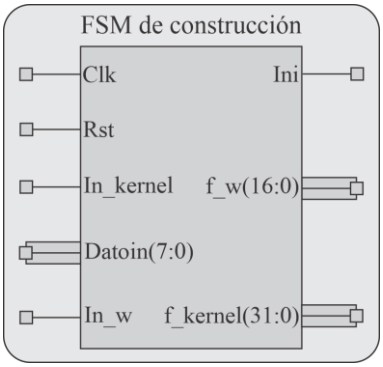

*Figura 4.7. Símbolo del módulo **FSM de construcción.***

*Tabla 4.4. Descripción de las señales de Entrada/Salida de **FSM de construcción.***

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100Mhz.
Rst	(Entrada) Señal de *reset*
In_kernel	(Entrada) Señal que notifica el envío de los datos al Kernel
Datoin(7:0)	(Entrada) Bus de datos que pueden ser enviados para el vector **w** o para el Kernel
In_w	(Entrada) Señal que notifica que el envío de datos es para el vector **w**
Ini_FD	(Salida) Señal de control, se manda para indicar el inicio de la función de decisión
S_w(16:0)	(Salida) Bus de datos que se origina del módulo vector **w**
S_ker(31:0)	(Salida) Bus de datos que se origina del módulo Kernel

El objetivo de este módulo es dirigir los datos hacia los módulos correspondientes para el cambio de dimensión cuando son dirigidos hacia el **Kernel** o para describir la estructura del clasificador cuando son dirigidos hacia el módulo **Vector W**. Esto depende directamente de las señales de control **In_kernel** e **In_w**; una vez concluidas estas operaciones, la **FSM de construcción** indica a la **Función de decisión** que puede realizar la clasificación de los datos.

4.3.2.3.2. Módulo Kernel

El módulo **Kernel** realiza el cambio de dimensionalidad del conjunto de datos original, recibe los datos y, con base en las ecuaciones descritas en el apartado 3.1.5, transforma el conjunto de datos a dimensiones mayores; mediante operaciones aritméticas y auxiliándose en una FSM envía los datos para ser clasificados. La Figura 4.8 muestra el símbolo del módulo **Kernel** y en la Tabla 4.5 se describen las señales que infieren en su operación.

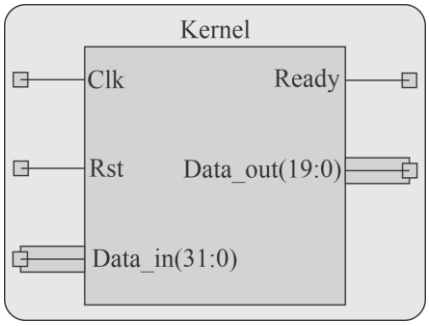

*Figura 4.8. Símbolo esquemático del módulo **Kernel**.*

*Tabla 4.5. Descripción de las señales de Entrada/Salida del módulo **Kernel**.*

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100Mhz.
Rst	(Entrada) Señal de reset.
Data_in (31:0)	(Entrada) Bus de datos, que contiene el set de datos original.
Ready1	(Salida) Señal que indica que el cálculo del kernel ha terminado e inicia el envío de los datos transformados.
Data_out (19:0)	(Salida) Bus de datos que envía los datos transformados

En la Figura 4.9 se muestra el diseño de la arquitectura hardware concurrente que describe la transformación de datos del **Kernel**. Considerando la Ecuación (72), el valor $\sqrt{2}$ se aproxima a 1.4375 para implementarse en el FPGA con el número binario 10111, tomando el bit más significativo como la parte entera y los restantes como la parte decimal.

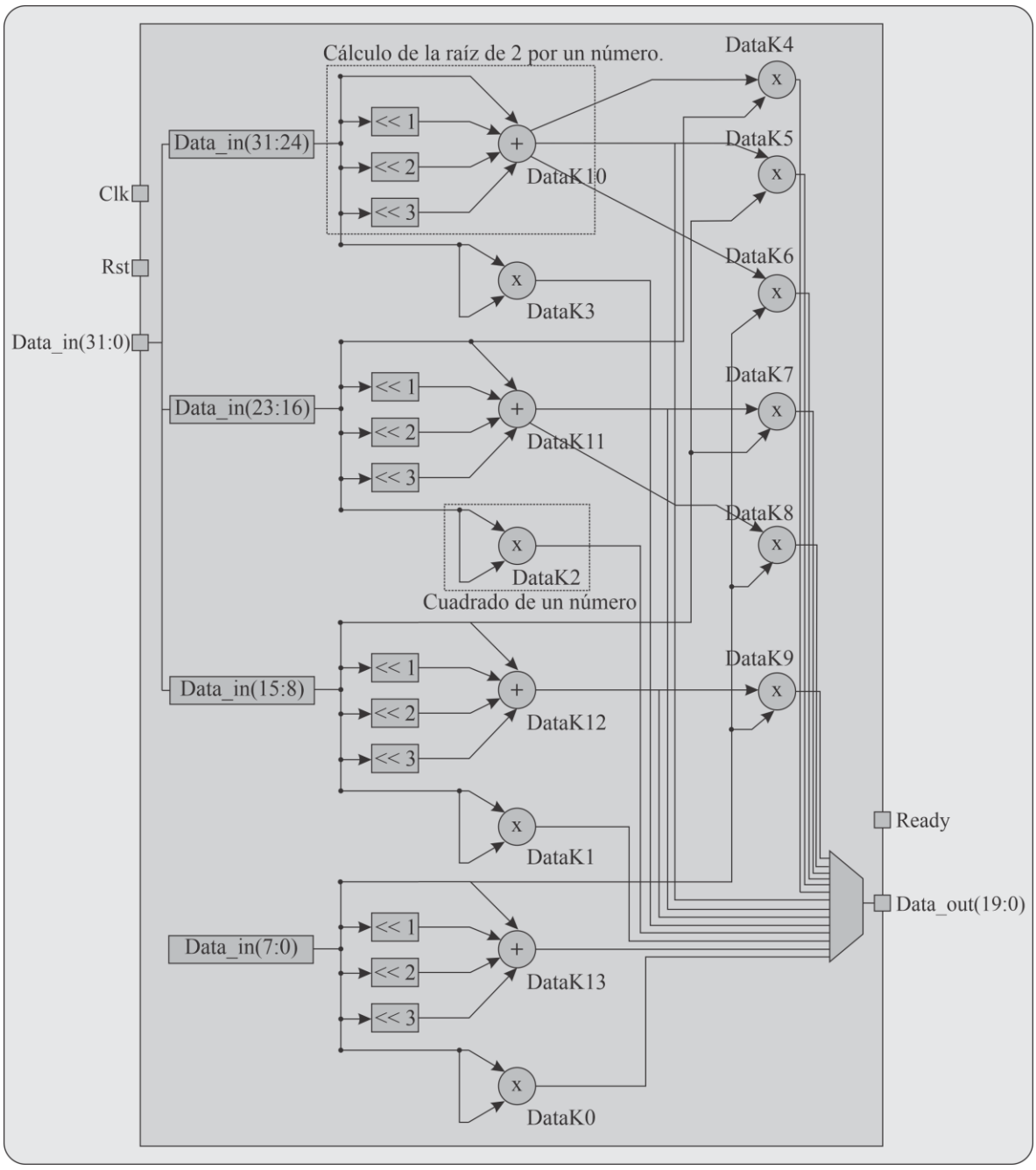

*Figura 4.9. Arquitectura hardware del **Kernel**.*

El bus de datos entrante es de 32 bits, separados en 4 datos de 7 bits; estos datos representan un vector de 4 características y cada característica se utiliza para calcular la nueva dimensionalidad del espacio de características transformado, usando las ecuaciones descritas en la sección 3.1.5.1. La multiplicación $\sqrt{2}$ por un número x se hizo por medio de desplazamientos de datos, para ahorrar multiplicadores y recursos en el FPGA, después de realizar las operaciones, la nueva dimensión del vector de características resultante es de 16

datos y cada dato tiene una longitud de 20 bits, los datos se envían a la **Función de decisión** para realizar la clasificación de los datos.

4.3.2.3.3. Vector $\boldsymbol{w}$

La función del módulo **Vector w** es almacenar la información del vector **w**, obtenido con la ecuación(27), que describe la estructura de la SVM; cada clasificador cuenta con un módulo de este tipo. Este módulo está construido por una RAM distribuida *single port* de tamaño $n \times m$, donde n es el número de localidades de la RAM que depende del clasificador que se está implementando y m es el tamaño en bits del del vector **w** que se está empleando, este tamaño es constante para cualquier clasificador.

El proceso de escritura de las memorias es síncrono. Una vez terminado el proceso de escritura para todos los clasificadores, esta operación queda desactivada y los datos almacenados están disponibles para su uso. La Figura 4.10 muestra el símbolo del módulo y la Tabla 4.6 muestra la descripción de sus señales.

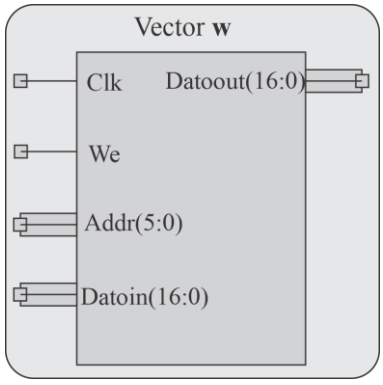

Figura 4.10. Símbolo del módulo ***Vector w***.

Tabla 4.6. Descripción de las señales Entrada/Salida del módulo ***Vector w***.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100Mhz
We	(Entrada) Señal que habilita la operación de escritura de la RAM
Addr (5:0)	(Entrada) Bus de direcciones, designa la localidad donde se va a guardar el dato
Datoin (16:0)	(Entrada) Bus de datos, contiene el dato que se va a guardar
Datoout (16:0)	(Salida) Bus de datos disponible para ser usado por la función de decisión

4.3.2.3.4. Función de decisión

La **Función de decisión** está presente en los clasificadores y se basa en un método que devuelve un número que representa si una muestra predicha se encuentra en el lado izquierdo o derecho del hiperplano, para ello se implementa la Ecuación (29). En la Figura 4.11 se muestra el símbolo del esquemático **fun_dec** y en la Tabla 4.7 se describen sus señales.

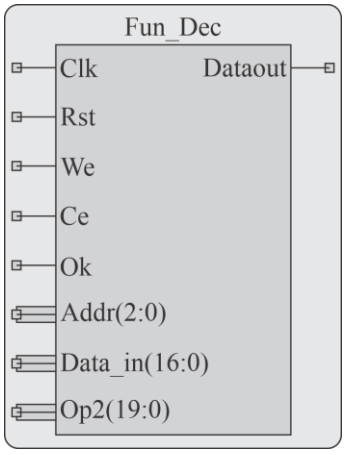

*Figura 4.11. Símbolo del módulo **Función de decisión (fun_dec)**.*

*Tabla 4.7. Descripción de las señales Entrada/Salida del módulo **Función de decisión(fun_dec)**.*

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100Mhz.
Rst	(Entrada) Señal de reset.
We	(Entrada) Señal de control de escritura de la memoria.
Ce	(Entrada) Señal de control para habilitar el biestable.
Ok	(Entrada) Señal de control para habilitar el comparador una vez concluido cada dato.
Addr(2:0)	(Entrada) Bus de dirección para la memoria
Data_in(16:0)	(Entrada) Bus de datos para la memoria
Op2(19:0)	(Entrada) Bus de datos que recibe los datos a clasificar después de pasar por el kernel.
Dataout	(Salida) Dato de un bit que define la clase del dato.

En la Figura 4.12 se muestra la arquitectura del módulo **Función de decisión**. Una vez que el módulo **Kernel** ha cambiado la dimensionalidad de los datos, envía la señal al módulo **Función de decisión** para iniciar el proceso de clasificación. Este proceso se repite con cada característica del **Vector w** que se envía a clasificar; la **Función de decisión** está descrita por medio de una arquitectura **MAC**. Una vez terminado de procesar los datos del vector, éste es evaluado por la **Función signo**, quien determina la resolución de clase final para su envío a la **Interfaz de usuario**. Con cada nuevo vector se reinician tanto el

comparador como el acumulador para evitar que tengan residuos del vector anterior, asegurando de esa manera una clasificación correcta de los datos.

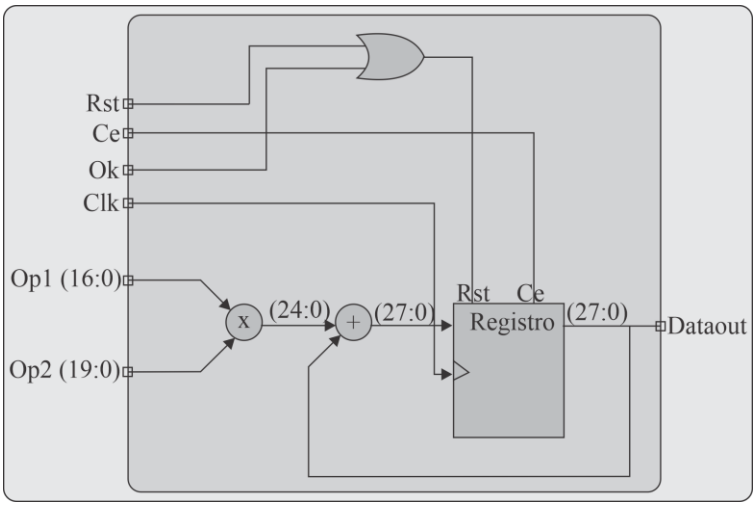

Figura 4.12. Arquitectura del módulo ***Función de decisión****.*

4.3.2.3.5. Resolución de clase

El módulo **resolución de clase** recibe el resultado del o de los clasificadores que se han usado para la aplicación. En caso de que sea un solo clasificador como puede ser el caso linealmente separable y linealmente no separable, mediante una función signo se evalúa el bit más significativo (MSB, *most significant bit*) asignando un valor de salida de cero o uno.

$$sig = \begin{cases} 1\ si\ \ Dataout(17) < 0 \\ \\ 0\ si\ Dataout(17) \geq 0 \end{cases}$$

En el caso multiclase, recibe el resultado de los clasificadores, estos resultados también son evaluados mediante una función signo, posteriormente se construye un vector con los valores que se obtiene de cada clasificador, una vez que este vector se ha construido es interpretado por un decodificador como se muestra en la Tabla 4.8, a continuación es enviado hacia la **Interfaz de usuario** para desplegar el resultado, es preciso decir que el vector puede ser de una longitud mayor y depende de cuantos clasificadores se usen para la aplicación deseada.

Tabla 4.8. Ejemplo de cómo se forma el vector resultante.

Resultado de función signo			Interpretación	Vector resultante
Clasificador 1	Clasificador 2	Clasificador 3		
1	0	0	01	00000001
0	1	0	10	00000010
0	0	1	11	00000011

El **Clasificador SVM** se puede construir por un solo clasificador o más clasificadores para cualquier aplicación. Estos clasificadores se comportan como un clasificador binario y es la forma en la que se combinan que permiten el caso no lineal o caso multiclase. La implementación hardware del sistema final se muestra en la Figura 4.13.

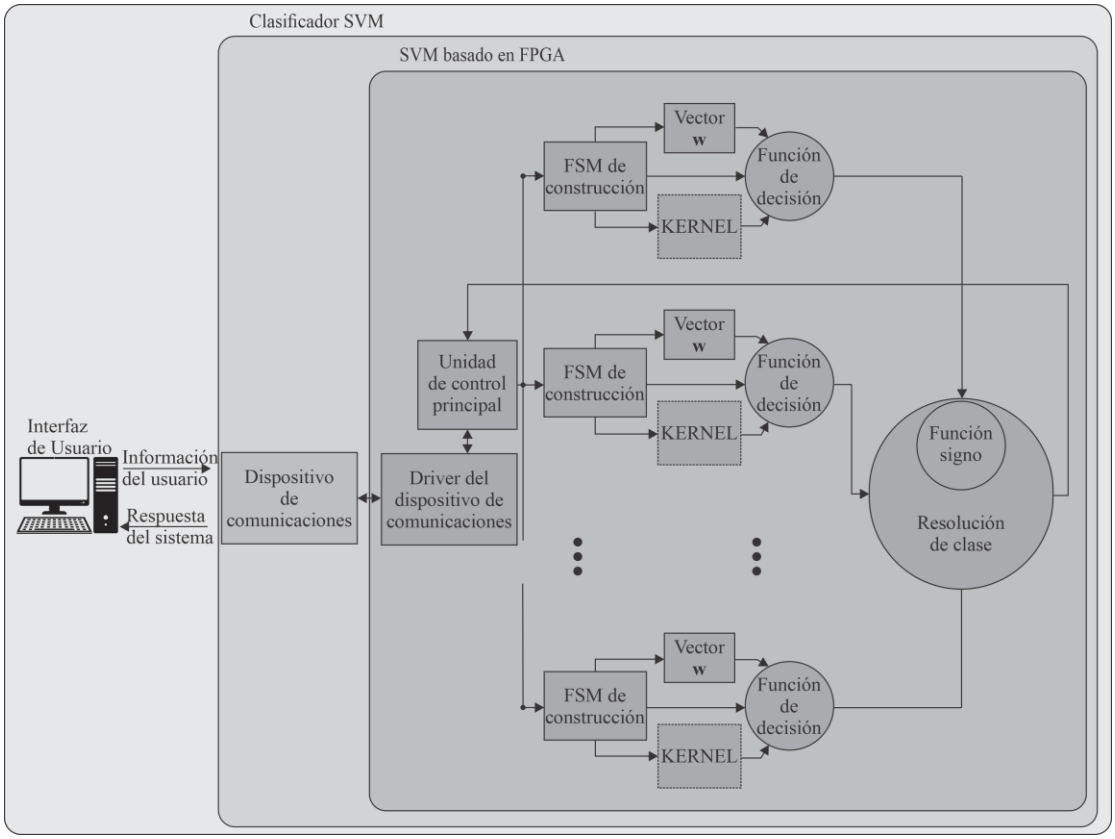

Figura 4.13. Diagrama a bloques del sistema en general de la implementación a Hardware.

4.4. Resultado de la implementación en hardware

Con la finalidad de medir el desempeño del AHC-SVM, se hizo uso de los módulos diseñados y modelados, descritos en el apartado 4.3.2, para implementar estructuras de SVMs para las 3 aplicaciones generales: caso linealmente separable, caso linealmente no separable y multiclase. Estas implementaciones son realizadas en un FPGA XC6lX16-cs324,

perteneciente a la familia Spartan-6 LX de la compañía Xilinx Inc., integrado en la tarjeta de evaluación Nexys 3 de la firma Digilent Inc.

4.4.1. Resultado del caso linealmente separable

Para el caso linealmente separable (apartado 3.1.1) se planteó el problema de la compuerta AND, que es un clásico ejemplo de problema de clasificación lineal en dos dimensiones, para probar el desempeño de la SVM. Este problema cuenta con dos clases, cuyos valores son 1 y 0; la Tabla 4.9 muestra los vectores de dos dimensiones del problema AND junto con su clasificación. El problema consiste en proporcionar un vector de dos dimensiones y se debe determinar a qué clase pertenece.

Tabla 4.9. Conjunto de entrenamiento de la compuerta AND.

Patrón	Descriptor 1	Descriptor 2	Valor deseado
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

El modelado e implementación del clasificador SVM se lleva a cabo mediante instanciación de los diferentes módulos ofrecidos por el sistema propuesto. Estos módulos determinan la interacción y el funcionamiento del clasificador. La Figura 4.14 muestra el diagrama a bloques del SVM construido para dar solución al problema linealmente separable.

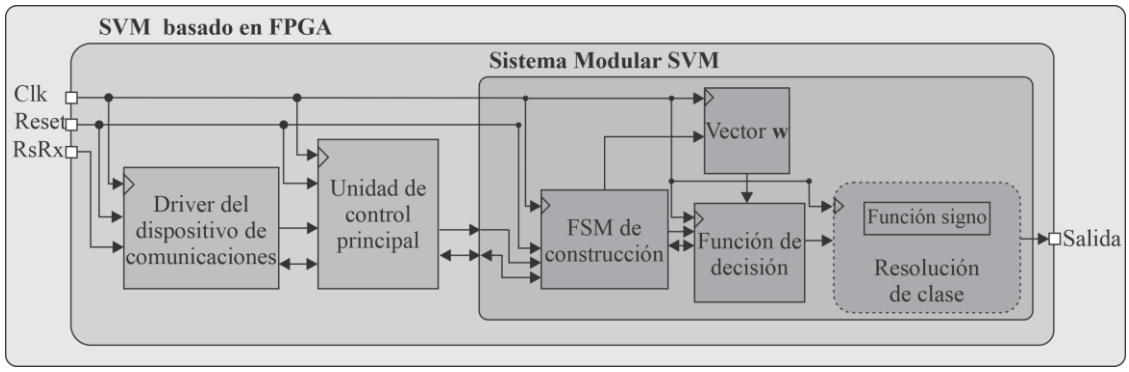

Figura 4.14. Diagrama a bloques del del SVM construido para dar solución al problema AND.

Se envían los pesos calculados en el modelado conceptual al FPGA; como la aritmética dentro del FPGA es de punto fijo en un formato *Q*1.4.3 se toma 1 bit para

representar el signo, 3 bits para representar la parte entera y 4 bits para la parte fraccionaria. La Tabla 4.10 muestra una aproximación de los pesos calculados y como lo recibe el FPGA.

Tabla 4.10. Aproximaciones del vector de pesos en formato punto fijo.

Representación en punto fijo de los pesos	
PC	**FPGA**
2	00010.000
2	00010.000
-3	10011.000

La Figura 4.15 muestra el camino de datos para la clasificación de nuevos datos, las veces que esta arquitectura es usada durante la clasificación de un patrón es igual a la dimensión de este. Debido a que tanto los elementos del patrón como los vectores **w** están representados en 8 bits y a que su dimensión es 2, el resultado final generado por esta arquitectura quedara representado en 17 bits, en un formado Q1.9.7, se desprecia el LSBit para obtener una representación de 16 bits en Q1.9.6.

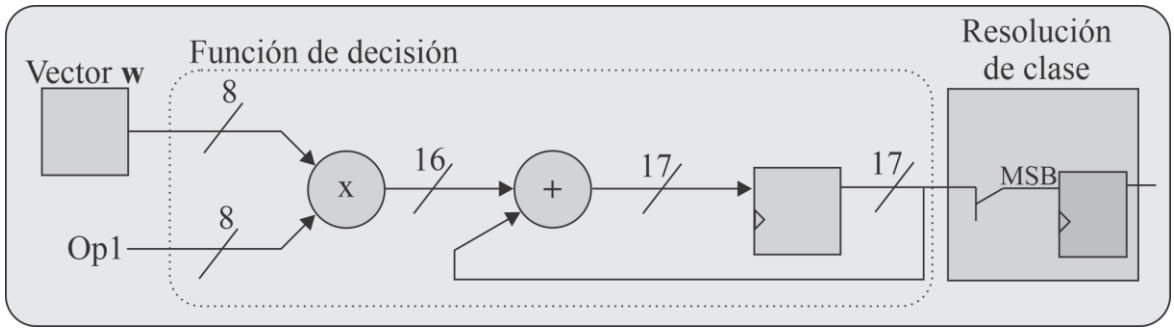

Figura 4.15. Arquitectura del clasificador linealmente separable.

La **Interfaz de usuario** permite activar el proceso de clasificación con el botón clasificación (véase Figura 4.16), en donde mediante la ventana mostrada en la Figura 4.17 se introduce un patrón de prueba de cualquier valor. En la Figura 4.16 se muestra los resultados generados por SVM entrenada, el hiperplano de separación y los pesos que se envían al FPGA.

La implementación del sistema para el caso linealmente separable, proporciona un porcentaje de clasificación del 100% para los ejemplos que se muestran en la Figura 4.18, estos resultados se utilizan para determinar el desempeño del clasificador SVM.

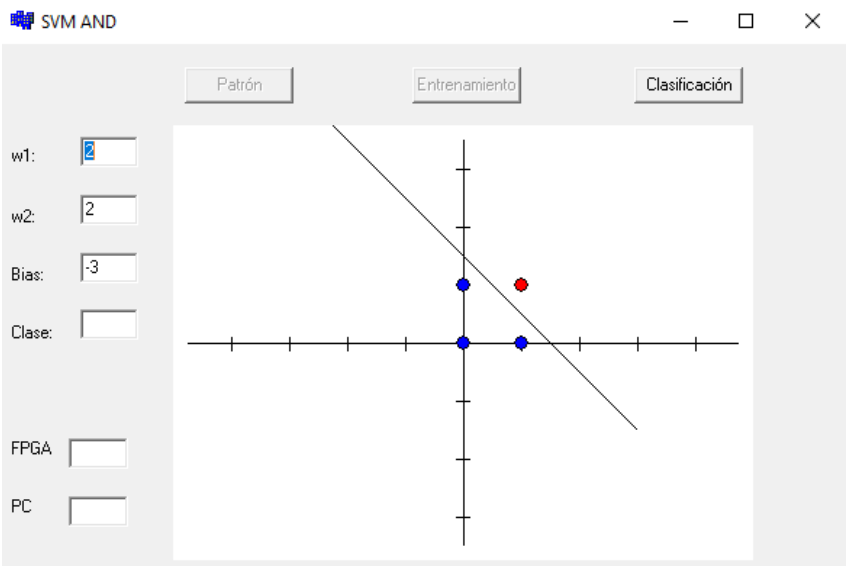

Figura 4.16. Resultado del clasificador SVM.

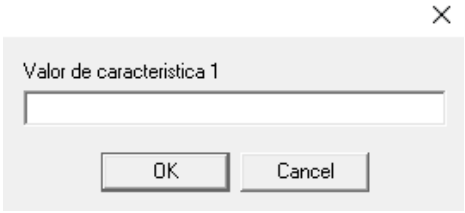

Figura 4.17. Ventana para introducir las características de un patrón.

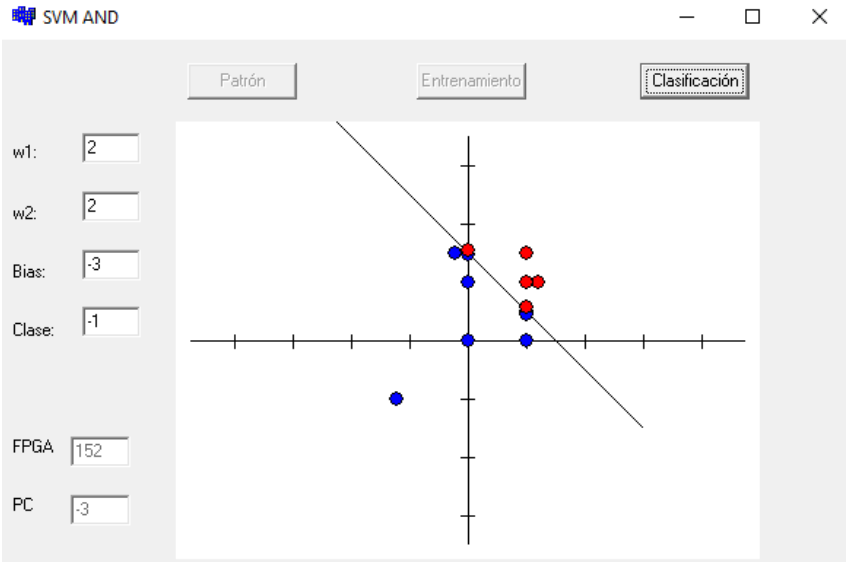

Figura 4.18. Clasificador con diversos patrones de prueba.

En la Tabla 4.11 se muestra la cantidad de recursos que se utilizan del FPGA y la frecuencia máxima del sistema de la implementación para el caso linealmente separable. Dicha información se obtiene del reporte de síntesis de la herramienta ISE Foundation de Xilinx.

Tabla 4.11. Recursos utilizados por la implementación linealmente separable.

Recursos	Recursos usados	Recursos de la tarjeta	Porcentaje	Frecuencia del sistema
Número de Slices	156	18,224	1%	124.517Mhz
Número de Luts	180	9,112	2%	
Usados como memoria	11	2,176	1%	
Número de DSP48A1s	1	32	3%	

4.4.2. Resultado del caso linealmente no separable

Para el caso linealmente no separable (apartado 3.1.2) se considera el problema XOR que es un clásico ejemplo de problema de clasificación no lineal en dos dimensiones. Este ejemplo cuenta con dos clases cuyos valores son 1 y 0. La Tabla 4.12 muestra los vectores de dos dimensiones del problema XOR junto con su clasificación. Como en el caso anterior, el problema consiste en proporcionar un vector de dos dimensiones y se debe determinar a qué clase pertenece.

Tabla 4.12. Conjunto de entrenamiento de la compuerta XOR.

Patrón	Descriptor 1	Descriptor 2	Valor deseado
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Para este caso, como ya se ha mencionado, es necesario convertirlo en un problema linealmente separable. Esto se logra mediante su mapeo a un espacio de diferente dimensión al que pertenece o aplicándole una transformación que lo deje en el mismo espacio, ya se han discutido las bases que sustentan la aplicación de estos métodos. Por lo tanto, para este caso se agregó el módulo **Kernel** como parte del clasificador. En la Figura 4.19 se muestra el diagrama a bloques del SVM construido para dar solución al problema linealmente no separable que ha sido planteado.

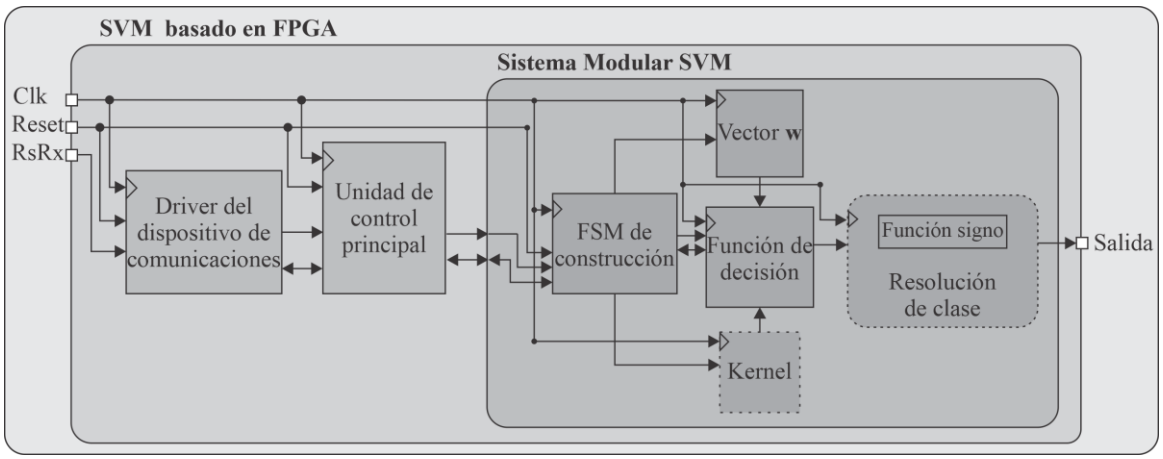

Figura 4.19. Diagrama a bloques del SVM construido para dar solución al problema XOR

De la misma manera que en el caso linealmente separable, existe un botón que permite introducir nuevos patrones de clasificación, estos patrones se introducen por medio de una ventana como la que se muestra en la Figura 4.17.

La Figura 4.20 muestra el camino de datos para la clasificación de nuevos datos, las veces que esta arquitectura es usada durante la clasificación de un patrón es igual a la dimensión de este. Debido a que los elementos del patrón están representados en 16 bits y los vectores **w** en 8 bits y a que su dimensión es 2, el resultado final generado por esta arquitectura quedara representado en 25 bits, en un formado Q1.14.10, se desprecia el LSBit para obtener una representación de 24 bits en Q1.14.9.

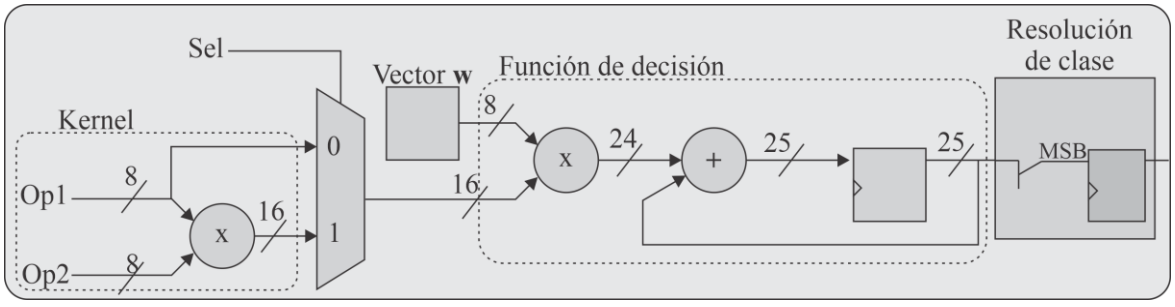

Figura 4.20. Arquitectura del clasificador con el kernel correspondiente.

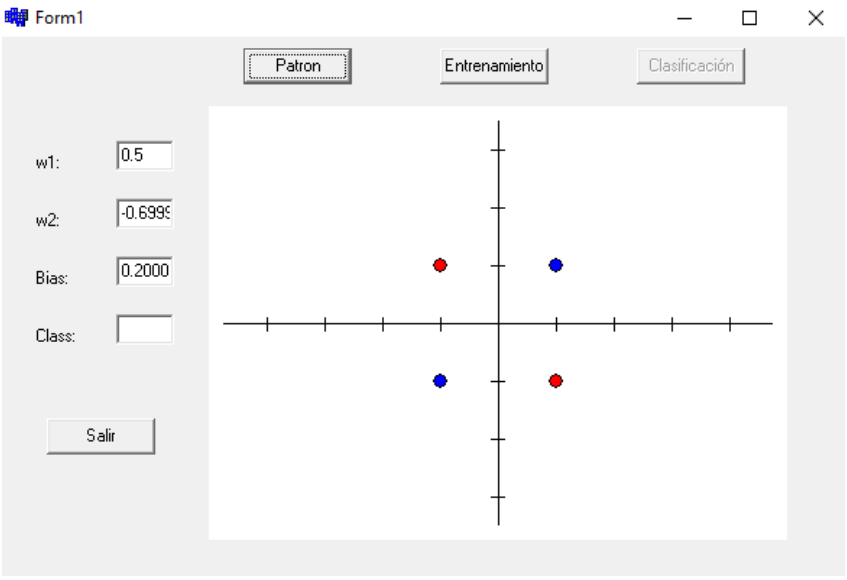

Figura 4.21. Patrón del set de datos XOR

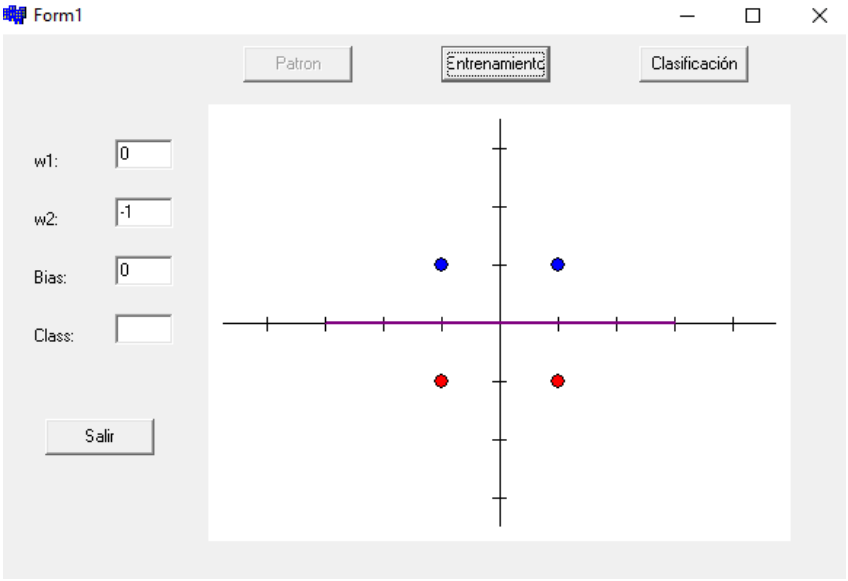

Figura 4.22. Resultado del clasificador entrenado de la compuerta XOR.

La Figura 4.21 muestra el mapeo del conjunto de datos del problema XOR. En la Figura 4.22 se aprecia como después de aplicar el kernel descrito en la ecuación (73) al problema XOR, este ha sido transformado a un problema linealmente separable. La Figura 4.22 también muestra el hiperplano óptimo de separación obtenido.

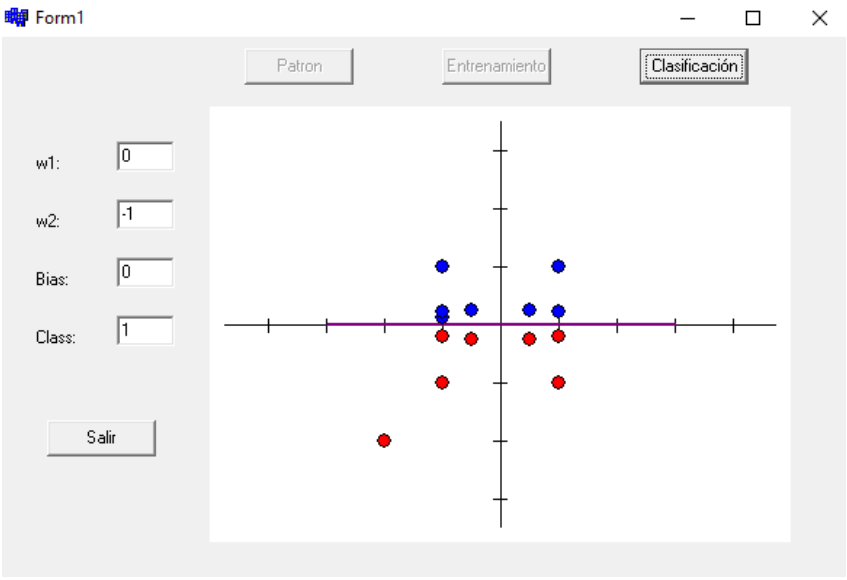

Figura 4.23. Clasificador con diversos patrones de prueba.

La implementación del sistema para el caso linealmente no separable, proporciona un porcentaje de clasificación del 100% para los ejemplos que se muestran en la Figura 4.23, estos resultados son utilizados para determinar el desempeño del clasificador SVM.

En la Tabla 4.13 muestra los recursos que se utilizan para el caso linealmente no separable y la frecuencia de máxima del sistema. Dicha información se obtiene del reporte de síntesis de la herramienta ISE Foundation de Xilinx.

Tabla 4.13. Recursos utilizados por la implementación linealmente no separable.

Recursos	Recursos usados	Recursos de la tarjeta	Porcentaje	Frecuencia del sistema
Número de Slices	159	18,224	1%	82.804MHZ
Número de Luts	196	9,112	2%	
Usados como memoria	10	2,176	1%	
Número de DSP48A1s	2	32	6%	

4.4.3. Resultados del caso multiclase

En el modelado conceptual se describe el ejemplo que se utiliza para evaluar la SVM en el caso multiclase (apartado 3.1.4). Se usa la base de datos de la Planta Iris, que consiste en un conjunto con 150 datos con 3 clases diferentes. Una característica importante de esta base de datos es que incluye clases linealmente separables y no linealmente separables. Cada patrón que representa una planta está formado por cuatro descriptores como muestra la Tabla 4.14 y, para su procesamiento dentro de la arquitectura SVM, cada descriptor es representado en punto fijo Q1.4.3.

Tabla 4.14. Descriptores de la Planta Iris.

Rasgo	Descripción
1	Largo del sépalo
2	Ancho del sépalo
3	Largo del pétalo
4	Ancho del pétalo

El entrenamiento del **SVM basado en FPGA** para el problema de la Planta Iris se realiza en la **Interfaz de Usuario** utilizando el 50% de los vectores de esta base de datos. Los vectores **w** generados por este proceso definen la arquitectura del SVM que soluciona el problema de la Plante Iris. Considerando que los valores de los vectores **w** obtenidos en el proceso de aprendizaje son menores a 1, la arquitectura del SVM que da solución a este problema utiliza una representación de los datos en punto fijo Q1.0.15.

La implementación multiclase para el set de datos de la Planta Iris consideró una arquitectura de cuatro clasificadores, esto se determinó considerando el número de clases del set de datos; se debe mencionar que cada clasificador es binario como se dijo anteriormente y se construyó usando los módulos descritos en los apartados anteriores, con el propósito de dar solución al set de datos, se utiliza el enfoque descrito en los incisos 3.1.4.1 y 3.1.4.2:

- El kernel que se utiliza para el set de datos es un Kernel polinomial no homogéneo descrito el apartado 3.1.5 ecuación (60).
- El enfoque uno contra todos, se utiliza para la clasificación de la clase uno contra tres y dos; también para la clasificación de la clase tres contra dos y uno; este enfoque ve a las clases restantes como una sola clase.
- El enfoque por pares, se utiliza para la clasificación dos contra uno y después dos contra tres, al separar las clases de esta forma permite una clasificación más precisa, el resultado de las dos evaluaciones es dirigida hacia una compuerta AND para unir los clasificadores y que se comporte como un solo clasificador.

Una vez que realiza el proceso de clasificación para cada vector nuevo, el resultado de cada clasificador va hacia el módulo **resolución de clase** descrito en el párrafo 4.3.2.3.5, para posteriormente ser enviado a la **interfaz de usuario** y ser mostrado en pantalla.

LaFigura 4.24 muestra el diagrama general de la arquitectura SVM que da solución al problema Planta Iris.

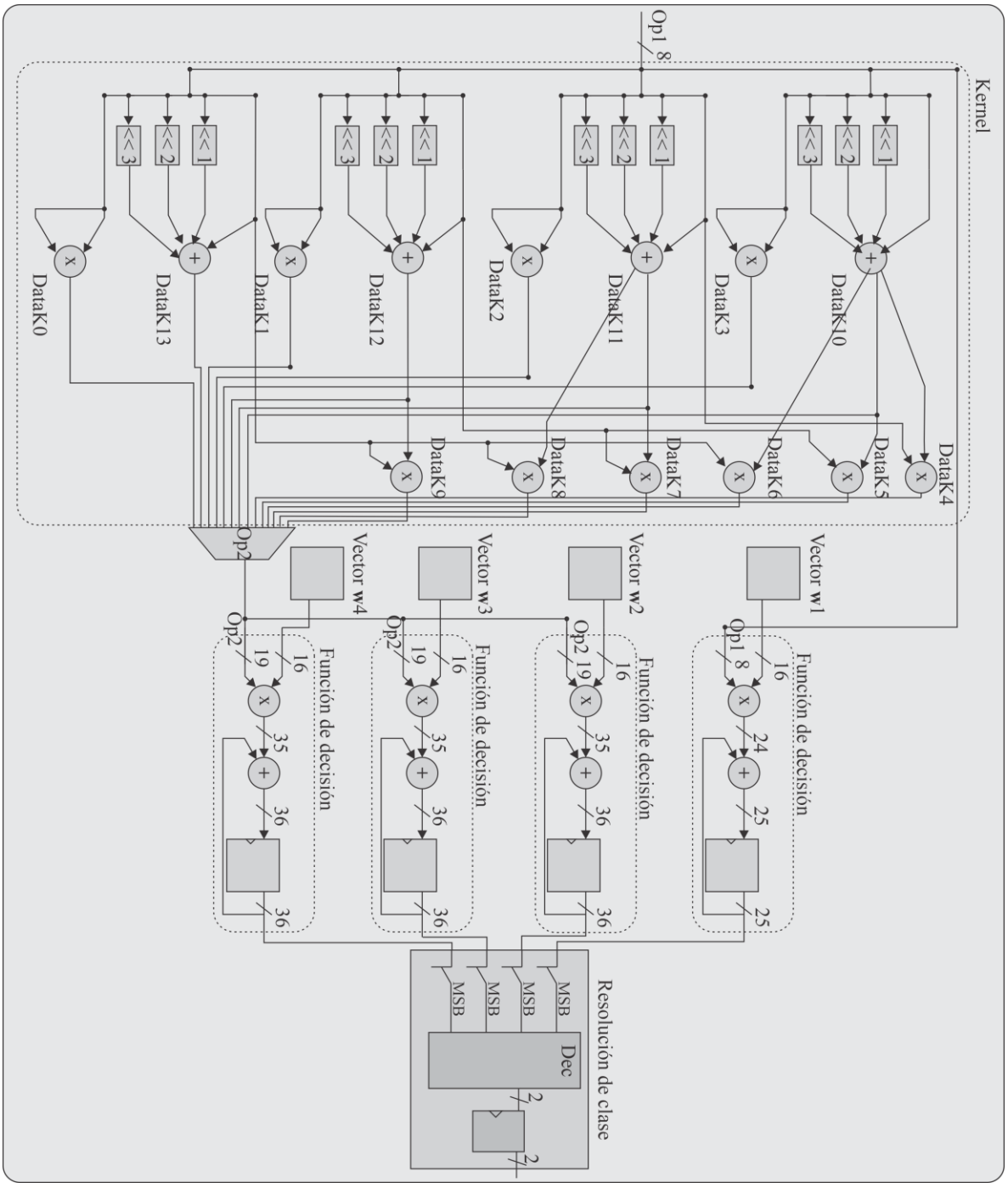

Figura 4.24. Arquitectura del clasificador multiclase con su correspondiente kernel.

Para determinar el desempeño de la arquitectura del clasificado SVM construida mediante la AHC-SVM, se hace uso del 50% de los vectores de esta base de datos de la Planta Iris restantes, los cuales se envían mediante la **Interfaz de Usuario**. La Tabla 4.15 muestra la matriz de confusión que representa los resultados obtenidos por el proceso de clasificación.

Tabla 4.15. Matriz de confusión del clasificador SVM en Hardware.

	Predicciones del sistema conceptual		
clases	**(a)**	**(b)**	**(c)**
Setosa	50	0	0
Versicolor	0	38	12
Virgínica	0	3	47

Los resultados que se obtienen se muestran en la Tabla 4.16 brindan la información necesaria para la comprensión del comportamiento y desempeño del sistema, considerando las métricas descritas anteriormente. Se destaca el porcentaje de clasificación obtenidos que es del 93%.

Tabla 4.16. Métricas resultantes usando los datos de la matriz de confusión del resultado en Hardware.

Clase\rate	**TP**	**TPR**	**TN**	**TNR**	**Te**	**Precisión**	**Exactitud**
Setosa	50	1	100	1	0	1	1
Versicolor	38	0.9268	97	0.8899	0.1	0.76	0.9
Virgínica	47	0.7966	88	0.9670	0.1	0.94	0.9
Avg		0.9078		0.9523	0.0666	0.9	0.9333

Tabla 4.17 muestra los recursos que se utilizan para el caso multiclase. La frecuencia máxima de operación es 95.632MHz. Dicha información se obtuvo del reporte de síntesis de la herramienta ISE Foundation de Xilinx.

Tabla 4.17. Recursos utilizados por la implementación multiclase.

Recursos	**Recursos usados**	**Recursos de la tarjeta**	**Porcentaje**
Número de Slices	682	18,224	3%
Número de Luts	924	9,112	10%
Usados como memoria	48	2,176	2%
Número de DSP48A1s	14	32	43%

Capítulo 5. Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones a las que se ha llegado después de analizar los resultados obtenidos por la propuesta planteada al inicio de esta investigación. Adicionalmente, se proponen perspectivas o trabajos futuros que plantean la continuidad de esta tesis.

5.1. Conclusiones

La metodología empleada en el diseño y modelado de una arquitectura hardware de un clasificador basado en una SVM exige un modelado conceptual como una de sus etapas. Resulta evidente que esta etapa facilita enormemente el diseño de la arquitectura hardware del sistema propuesto debido a que hace posible comprender el funcionamiento de dicho sistema e identificar las fases que lo integran. Este conocimiento permite identificar los elementos de hardware requeridos en el diseño de la arquitectura para definir y planificar las técnicas adecuadas para su implementación en un dispositivo especifico.

Un enfoque modular rige el diseño del AHC-SVM, este enfoque ofrece una forma eficiente de adaptar nuestra propuesta en la solución de diversos problemas de clasificación, facilita la interoperabilidad entre los elementos de la arquitectura, permite el rediseño de elementos individuales y mejora la fiabilidad del sistema. Además, este enfoque permite

reutilizar cualquiera de los módulos creados en otros proyectos sin necesidad de realizarle modificaciones.

En el diseño de la arquitectura se explota la concurrencia existente en cada fase del clasificador SVM. Esto permite implementar un sistema con una alta velocidad de procesamiento volviéndolo ideal para aplicaciones en tiempo-real. Además, es un sistema fácilmente adaptable para trabajar en aplicaciones autónomas debido a su moderado consumo de energía.

Los módulos que forman parte del clasificador SVM fueron descritos mediante un lenguaje de descripción de hardware, el VHDL. Este lenguaje, al estar estandarizado, otorga al sistema propuesto las siguientes ventajas: 1. Ser un diseño portable, el modelado puede ser implementado en cualquier tecnología FPGA. 2. Es independiente a la metodología de diseño. 3. Es independiente a la herramienta EDA utilizada. Además, el VHDL permite el uso de diversos niveles de abstracción, incluso el que involucra paradigmas de software en el diseño lógico, y permite el modelado de un sistema definido a partir de niveles jerárquicos, fomentando el modelado orientado a la reutilización.

Con la finalidad de validar el funcionamiento del AHC-SVM, se utiliza para construir arquitecturas hardware que dan solución a problemas que son comúnmente utilizados en la verificación del desempeño de sistemas de clasificación. Los resultados que se obtienen por estas arquitecturas muestran que en el parámetro de eficiencia es competitivo con implementaciones tradicionales en software (programa secuencial ejecutado en una computadora personal o en un microcontrolador), incluso superándola en algunos casos como el de la Planta Iris, mientras que en los parámetros de velocidad de procesamiento y consumo de potencia los supera. Esto permite concluir que las técnicas de aproximación utilizadas en el diseño de arquitecturas hardware, p. ej. el uso de punto fijo en lugar de punto flotante, no afectan su desempeño y minimiza el uso de recursos y consumo de potencia.

Es importante mencionar que la presente investigación generó una herramienta de diseño de arquitecturas hardware de clasificadores SVM, denominada AHC-SVM, la cual reduce el tiempo de desarrollo de este tipo de arquitecturas. Se trata de un entorno de desarrollo que ofrece los componentes y módulos necesarios para construir arquitecturas hardware de un modelo SVM, representando de esta manera un ambiente propicio para la

experimentación que permite aplicarla en el diseño, modelado e implementación de arquitecturas hardware que den solución a problemas de clasificación linealmente separables, linealmente no separables y multiclase. Además, este entorno incluye una interfaz de usuario que habilita una interacción eficiente y amigable con el AHC-SVM.

Aunque el AHC-SVM incluye un repositorio de módulos hardware y métodos relacionados con la operación de un SVM, nuevos componentes y módulos pueden ser creados y añadidos a la herramienta propuesta facilitando y promoviendo la creación de nuevo conocimiento y por ende favoreciendo su mantenimiento y evolución.

5.2. Trabajos Futuros

Se proponen los siguientes trabajos futuros:

- Ampliar el repositorio de módulos de la AHC-SVM, mediante el diseño y modelado de arquitecturas hardware que ofrezcan alternativas a los módulos ya incluidos, por ejemplo:
  - Módulos que implementen otros métodos kernel.
  - Módulo alternativo a la arquitectura MAC, creado con aritmética distribuida.
- Diseñar y modelar una arquitectura hardware para la definición de los vectores **w**.

Referencias

[1] C. J. Burges, «A tutorial on Support Vector Machine for Pattern Recognition,» *Data. Min. Knowl. Discov,* vol. 2, nº 2, pp. 121-167, 1998.

[2] D. L. Olson y D. Delen, Advanced Data Mining Techniques, Sringer, 2008.

[3] M. Rychetsky, Algorithms and Architectures for Machine Learning based on Regularized Neural Networks and Support Vector Approaches, Shaker Verlag, 2001.

[4] J. C. N. Shawe-Taylor, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

[5] J. Nayak, B. Naik y H. Behera, «A compreensive Survey on Support Vector Machine in Data Mininf Tast: Aplications & challenges,» *International Journal of Database Theor and Appplications,* vol. 8, pp. 169-186, 2015.

[6] P. Sabouri, H. GholamHosseinni, T. Larsson y J. Collins, «A cascade Classifier for Diagnosis of Melanoma in Clinical Images,» *IEEE,* 2014.

[7] G. M. Foody and A. Mathur, «A relative Evaluation of Multiclass Image Classification by Support Vector Machine,» *IEEE Transactions on Geoscience and Remote Sensing,* vol. 42, pp. 1335-1343, 2004.

[8] R. Entezari Maleki, A. Rezaei y B. Minaei Bidgoli, «Comparisson of Classification Methods Based on The Tyoe of Attributes and Sample Size,» *Journal of Convergence Information Technology,* vol. 4, pp. 94-102, 2009.

[9] J. Kim, B. S. Kim y S. Savarese, «Comparing Image Classification Methods: K-Nearest Neighbor and Support Vector Machines,» *Ann Arbor,* vol. 1001, pp. 48109-2122, 2012.

[10] G. Palm, Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity, Berlin Heidelberg: Brain Theory, 1984.

[11] A. M. Turing, «Computing Machinery and Intelligence,» *Mind,* pp. 433-460, 1950.

[12] J. Moor, «The Dartmouth College Artificial Conference: The Next Fifty Years,» *AI Magazine,* pp. 87-91, 2006.

[13] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors,» *Nature,* pp. 533-536, 1986.

[14] M. Cajamarca, «Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo,» 6 Enero 2019. [En línea]. Available: https://planetachatbot.com/inteligencia-artificial-aprendizaje-autom%C3%A1tico-y-aprendizaje-profundo-862ca9790bb9.

[15] S. Russell y P. Norving, Inteligencia Artificial un Enfoque moderno., Prenrice Hall, 1996.

[16] IBM, «What Is Machine Learning?,» 20 Febrero 2019. [En línea]. Available: https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning?lang=en.

[17] T. M. Mitchell, «The disipline of Machine Learning,» *School of Computer Science,* p. 1, 2006.

[18] R. V. Mendoza Méndez, E. J. Dorantes Coronado, J. Cedillo Monroy y X. Jasso Arriaga, «El método estadístico de análisis discriminante como,» *Revista Iberoamericana para la Investigación y el Desarrollo Educativo ,* vol. 7, nº 14, 2007.

[19] J. Mora Florez, G. Morales España y R. Barrera Cárdenas, «Evaluating a k-nearest neighbours-based classifier for locating faulty areas in,» *REVISTA INGENIERÍA E INVESTIGACIÓN,* vol. 28, nº 3, pp. 81-86, 2008.

[20] S. D. Pacheco Leal, L. G. Díaz Ortiz y R. García Flores, «El Clasificador Naive Bayes en la extracción de conocimiento de bases de datos,» *Ingenierias,* vol. 8, nº 27, 2005.

[21] X. Basogain Olabe, Redes Neuronales Artificiales y sus Aplicaciones, Bilbao: Publicaciones de la Escuela de Ingenieros, 1998.

[22] C. Cortes y V. Vapnik, «Support-vector networks,» *Machine,* vol. 20, nº 3, 1995.

[23] R. E. Barrientos Martínez, N. Cruz Ramírez, H. G. Acosta Mesa, I. Rabatte Suárez, M. d. C. Gogeascoechea Trejo, P. Pavón León y S. Blázquez Morales, «Árboles de decisión como herramienta en el diagnóstico médico,» *Rev Med UV,* pp. 19-24, 2009.

[24] A. J. Smola y B. Scholkopf, «A tutorial on support vector regression,» *Statistics and Computing,* nº 14, p. 199–222, 2004.

[25] H. N. Carl Edward Rasmussen, «Gaussian Processes for Machine Learning (GPML) Toolbox,» *Journal of Machine Learning Research,* nº 11, p. 3011−3015, 2010..

[26] J. M. Chambers y T. J. Hastie, Statistical models in S, Pacific Grove, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1992.

[27] B. D. Ripley, Pattern Recognition and Neural Networks, Oxford: Cambridge University Press, 1996.

[28] C. Szepesvári, Algorithms for Reinforcement Learning, Morgan & Claypool, 2010.

[29] M. P. Vésitas, «High-Performance Reconfigurable Computing Granularity,» *Encyclopedia of information Science an Technology,* pp. 355-3567, 2017.

[30] P. P. CHU, FPGA Prototyping by VHDL Examples, Wiley-Interscience, 2008.

[31] N. Battezzati, L. Sterpone y M. Violante, Reconfigurable Field Programmable Gate Arrays for Mission Critical Aplications, Berlin: Springer, 2011.

[32] M. P. García Juárez, *Implementación de redes neuronales sobre lógica reconfigurable,* Huajuapan de León, Oaxaca, 2013.

[33] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass , E. Cosatto, S. Chakradhar y H. P. Graf, «A Massively Parallel FPGA-based Coprocessor for Support Vector Machines,» *Symposium on Field Programable Custom Computing Machines,* pp. 115-122, 2009.

[34] R. Ramos Lara, M. López García, E. Cantó Navarro y L. Puente Rodriguez, «SVM Speaker Verfication Syste Based on a Low Cost FPGA,» de *International Conference on Field Programmable Logic and Applications*, 2009.

[35] M. Papadonikolakis , C. S. Bougnais y G. Constantinides, «Performance Comparison of GPU and FPGA architectures for the SVM Training Problem,» *International Conference on Fiel-Programmable Technology,* pp. 388-391, 2009.

[36] M. Papadonikolakis y C. S. Bouganis, «A Heterogeneous FPGA Architecture for Support Vector Machine Training,» de *IEEE Symposium on Field-Programmable Custom Computing Machines-FCCM*, 2010.

[37] J. Gomes Filho, M. Raffo, M. Strum y W. Jiang Chau, «A general Purpose Dynamically Reconfigurable SVM,» de *VI Southern Programmable Logic Conferende*, 2010.

[38] S. Bauer, S. Köhler, K. Doll y U. Brunsmann, «FPGA-GPU Architecture for Kernel SVM Pedestrian Detection,» de *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[39] X. Pan, H. Yang, L. Li, Z. Liu y L. Hou, «FPGA Implementation of SVM Decision Function Based on Hardware-Friendly Kernel,» de *International Conference on Computational and Information Sciences*, 2013.

[40] M. Pietron, M. Wielgosz, D. Zurek, E. Jamro y K. Wiatr, «Comparison of GPU and FPGA Implementation of SVM Algorithm for Fast Image Segmentation,» de *Lecture Notes in Computer Science*, Berlin, 2013.

[41] H. M. Hussain, K. Benkrid y H. Seker, «Reconfiguration Based Implementation of SVM Classifier on FPGA for Classifiying Microarray Data,» de *35th Annual International Conference of the IEEE EMBS*, Osaka, Japón, 2013.

[42] T. Groléat, M. Arzel y S. Vaton, «Streching the Edges of SVM Traffic Classification With FPGA Acceleration,» *IEEE Transactions on Network And Service Management,* pp. 278-291, 2014.

[43] M. B. Rabieah y C. S. Bouganis, «FPGA Based Nonlinear Support Vector Machine Training Using an Ensemble Learning,» de *25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015.

[44] M. Qasaimeh, A. Sagahyroon y T. Shanableh, «"FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification,» *IEEE Transactions on Computational Imaging,* vol. XIII, pp. 56-70, 2015.

[45] S. M. H. Ho, Wang M., H. C. Ng y H. So, «Towards FPGA assisted Spark: An SVM Training Acceleration Case Study,» de *International Conference on ReConFigurable Computing and FPGAs*, 2016.

[46] F. F. Lopes, J. C. Ferreira y M. A. C. Fernandes, «Parallel Implementation on FPGA of Support Vector Machines Using Stochastic Gradient Descent,» 2019.

[47] O. Elgawi, A. M. Mutawa y A. Ahmad, «Energy-Efficient Embedded Inference of SVMs on FPGA,» *IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* pp. 164-168, 2019.

[48] D. H. Noronha, M. F. Torquato y M. A. C. Fernandes, «A parrallel implementation of sequential minimal optimization on FPGA,» *Microprocessors and Microsystems,* pp. 138-151, 2019.

[49] H. Byun, S.-W. Lee y A. Verri, Applications of Support Vector Machines for Pattern Recognition: A Survey, in Pattern Recognition with Support Vector Machines., Springer Berlin Heidelberg, 2002.

[50] J. Ecker y M. Kupferschmid, Introdution to Operations Research, John Wiley & Sons Inc, 1998.

[51] S. Abe, Support Vector Machines for Pattern Classification, Japón: Springer, 2005.

[52] C. Berg, J. P. Reus Christensen y P. Ressel, Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions, New York: Springer Science + Business Media, 1984.

[53] V. N. Vapnik, The Nature of Stadistical Learning Theory, New York: Springer-Verlag, 1995.

[54] U. H. KreBel, «Pairwise classification and support vector machines,» *Advances in Kernel Methods: Support Vector Learning,* vol. MIT Press, pp. 255-68, 1999.

[55] B. Kijsirikul y N. Ussivakul, «Multiclass support vector machines using adaptive directed acyclic graph.,» *Proceedings of the 2002 Internatioan Joint Conference on Neural Networks,* vol. 1, nº 02, pp. 980-5, 2002.

[56] J. C. Platt, N. Cristianini y J. Shawe-Taylor, «Large Margin DAGs for multiclass classification,» *Advanced in Neuronal Information Processing System,* nº 12, pp. 547-53, 2000.

[57] M. Pontil y A. Verri, «Support vector machines for 3-D object recognition,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 20, nº 6, pp. 637-46, 1998.

[58] F. Takahashi y S. Abe, «Decision-tree-based multiclass support vector machines,» *Proceedings of the Ninth International Conference on Neural Information Processing,* vol. 3, nº 02, pp. 1418-22, 2002.

[59] T. G. Dietterich y G. Bakiri, «Solving multiclass learning problems via correcting output codes.,» *Journal of Artificial Intelligence Research,* vol. 2, pp. 263-86, 1995.

[60] K. Crammer y Y. Singer, «On the learnability and design of output codes for multiclass problems,» *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory,* pp. 35-46, 2000.

[61] J. Weston y C. Watkins, «Multi-class support vector machines,» de *Technical Report CSD-TR-98-04,Royal Holloway.*, Londres, 1998.

[62] A. Shaushua, Introduction to Machine Leraning, Jerusañem, Israel: School of computer Science and Engineering, 2008.

[63] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard y C.-J. Lin, «Training and testing low-degree Polynomial Data Mappings via Linear SVM,» *Journal of Machine Learning Research,* pp. 1471-1490, 2010.

[64] O. G. Selfridge, «Pandemonium: A paradigm for learning,» de *Mechanisation of Thought Processes*, Londres, 1958.

[65] F. Rosenblatt, «The Perceptron: A probabilistic model for information storage and organization in the brain,» *Psychological Review,* pp. 386-408, 1958.

[66] H. Yu, J. Yang y J. Han, «Classifying Large Data Sets Using SVMs with Hierarchical Clusters,» *in Proceedings of the 9th ACM SIGKDD 2003,* pp. 3006-315, 2003.

[67] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors,» *Nature,* pp. 533-536, 1986.