RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

FPGAs en el entrenamiento de SVM para clasificación de grandes volúmenes de datos

Ing. Lázaro Bustio Martínez,
Dr. C. José Hernández Palancar,
Dr. C. René Cumplido Parra

RT_009 **Noviembre 2009**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
Ciudad de La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260
Versión Digital

REPORTE TÉCNICO
Minería de Datos

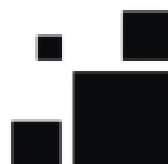
SERIE GRIS

FPGAs en el entrenamiento de SVM para clasificación de grandes volúmenes de datos

Ing. Lázaro Bustio Martínez,
Dr. C. José Hernández Palancar,
Dr. C. René Cumplido Parra

RT_009 **Noviembre 2009**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
Ciudad de La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu



Índice

1 Introducción 2
2 Clasificación por Vectores de Soporte 4
2.1 Caso linealmente separable..4
2.2 Caso linealmente no separable..8
2.3 Superficie de decisión no lineal..10
2.4 Algoritmos para el entrenamiento de SVM 12
3 Cómputo Reconfigurable 17
3.1 Conceptos básicos..18
3.2 FPGA..18
3.3 Flujo de diseño 20
3.4 DSVM 22
3.5 Clasificación con SVM en Sistemas Numéricos Logarítmicos 25
4 Conclusiones y trabajo futuro...27
Referencias bibliográficas 27

FPGAs en el entrenamiento de SVM para clasificación de grandes volúmenes de datos

Ing. Lázaro Bustio Martínez, Dr. C. José Hernández Palancar, Dr. C. René Cumplido Parra

Centro de Aplicaciones de Tecnología de Avanzada (CENATAV)
7ª/ 21812 e/ 218 y 222, Rpto. Siboney, Playa, C.P. 12200, La Habana, Cuba
lbustio@cenatav.co.cu

RT_009 CENATAV
Fecha del camera ready: 31 de julio de 2009

Resumen: En este trabajo se detallan las Máquinas de Soporte Vectorial y el basamento teórico sobre el que se sustentan. Se hace una descripción de los principales algoritmos que las implementan y que han mostrado mejores resultados según la bibliografía analizada, explicando sus características y particularidades. Teniendo en cuenta que el entrenamiento de una Máquina de Soporte Vectorial es un problema de complejidad cuadrática (enfoque optimista) respecto a las dimensiones de los datos sobre de entrenamiento, es que se proponen los Arreglos de Compuertas Lógicas Programables (o Field Programmable Logic Arrays o FPGA) como una posible solución a este problema y se analiza una arquitectura digital para el entrenamiento de SVM y otra para la clasificación de datos; ambas sobre FPGA.

Palabras claves: clasificación de datos, SVM, SMO, FPGA

Abstract: In this work the Support Vector Machines and the theoretical foundation on which they are based is described. They also have some of the algorithms that implement and they have shown better results in the literature reviewed, detailing its characteristics and peculiarities. Due to training a Support Vector Machine is a problem that has quadratic complexity (optimistic point of view) compared to the dimension of the training data is that also introduces the Field Programmable Logic Arrays (FPGA) and a detailed implementation of training and classification of SVM on FPGA showing their particularities.

Keywords: Data Classification, SVM, SMO, FPGA

1 Introducción

En la actualidad se genera en cuestión de segundos un enorme volumen de datos resultado del quehacer de la actividad humana. Ser capaz de manejar esta información es sinónimo de Poder y organizarla eficientemente es un reto para los científicos a nivel mundial. Muchas son las técnicas para intentar dotar de un valor cualitativo a esos datos como son la clasificación, el agrupamiento, el descubrimiento o minado de patrones frecuentes, las reglas de asociación, etc., enmarcadas todas dentro de diferentes líneas de investigación de lo que se conoce como Minería de Datos y Reconocimiento de Patrones. En este trabajo se hablará específicamente de la Clasificación de Datos. Todas éstas técnicas tienen por objetivo común extraer conocimiento potencialmente útil y comprensible de esos enormes volúmenes de información.

Hasta la fecha se han desarrollado varios algoritmos de clasificación los cuales se seleccionan y emplean dependiendo del problema a enfrentar, pero todos resultan inoperantes cuando se enfrentan

a elevados volúmenes de datos[1]: requieren un elevado período de tiempo para devolver los resultados o sencillamente no pueden procesarlos. Contar con un algoritmo capaz de clasificar grandes volúmenes de datos, con un consumo computacional adecuado y en un período de tiempo relativamente corto resultaría sumamente conveniente en muchísimas áreas de la ciencia y la vida cotidiana. Una técnica de aprendizaje automático que está siendo empleada en una gran multitud de áreas, entre las que se destaca la clasificación de datos, con resultados notablemente alentadores son las Máquinas de Soporte Vectorial (Support Vector Machines o SVM por sus siglas en inglés). Su aparición a principio de los años '90 provocó una explosión de aplicaciones y de profundos análisis teóricos lo cual hizo sea SVM en la actualidad una de las herramientas más poderosas en el de Aprendizaje Automático Supervisado y el Reconocimiento de Patrones[1]. Las SVM fueron desarrolladas por Vladimir Vapnik sobre la base de un problema de clasificación lineal binario; es decir, donde sólo se trabaja con dos clases, e implementa el Principio Inductivo de Minimización Estructural del Riesgo[2] para obtener buenos resultados de generalización sobre un número limitado de patrones de aprendizaje.

El proceso de clasificación mediante una Máquina de Soporte Vectorial consta de dos pasos: entrenamiento y clasificación, donde en el primero se reconocen los patrones del conjunto de datos de entrenamiento con el fin de crear un modelo que luego será empleado en la clasificación de nuevos datos. Este proceso presenta complejidad de orden cuadrático (enfoque optimista) respecto a las dimensiones de los datos de entrenamiento por lo que los problemas que se pueden solucionar con esta técnica se ven limitados. Actualmente existen tres algoritmos fundamentales[3] para el entrenamiento de SVM en software: Chunking[4], Sequential Minimum Optimization (SMO)[5] y SVM^{light}[6] (este último es una mejora propuesta al algoritmo planteado en el trabajo de Osuna "Improved Training Algorithm for Support Vector Machines"[7]); siendo SMO el de mayor impacto ya que logra mejorar la rapidez del entrenamiento de la SVM, consume menos recursos computacionales respecto a los algoritmos anteriores, es más fácilmente programable, no requiere de cálculos complejos para resolver el problema de programación cuadrática que SVM plantea donde todo lo dicho representa sus principales ventajas[8].

Dado el hecho en que SVM es un excelente clasificador, aunque inadecuado para la clasificación de grandes volúmenes de datos debido a los altos tiempos de entrenamiento y costo en recursos computacionales que requiere, es que entran a jugar su papel técnicas que puedan ayudar a aumentar el desempeño de SVM. Tal es el caso del uso de FPGAs (Field Programmable Gate Arrays) como posible solución a ésta problemática. A grandes rasgos, un FPGA es un circuito electrónico reconfigurable que permite crear nuevos circuitos que siguen un comportamiento definido según su programación para resolver un problema específico (más adelante se aborda el tema de los FPGAs con mayor profundidad) lo cual los dotan de una gran versatilidad. Un ejemplo del desempeño de los FPGAs se puede ver en el campo de la industria aeroespacial, donde un satélite artificial puede generar 900 Gigabytes de información por día. Se ha demostrado que con el uso de FPGAs se puede acelerar la clasificación de los datos obtenidos en 8 veces[9].

Lo explicado anteriormente sugiere que contar con un sistema de clasificación de datos mediante SVM sobre FPGAs podría ofrecer resultados sumamente alentadores y superar los que han sido obtenidos mediante las técnicas de clasificación por software.

Por otro lado, pocos han sido los intentos de llevar las Máquinas de Soporte Vectorial a hardware. Fuera de [10] no se han propuesto arquitecturas que implementen algoritmos para el entrenamiento de SVM en hardware, específicamente sobre FPGA.

---

[1] Cuando se habla de "enormes volúmenes de datos" se hace referencia a conjuntos de datos con millones de elementos. Vistos dichos datos en forma matricial, se está hablando de matrices con millones de filas y columnas.

Este trabajo ha sido estructurado en cuatro secciones, incluyendo "Introducción" y "Conclusiones y trabajos futuros". En la segunda sección se presenta SVM y se describe el basamento teórico sobre el que se sustenta. En esta sección igualmente varios algoritmos para el entrenamiento de una SVM son descritos mostrándose las características de cada uno y se realiza un estudio más detallado de sus ventajas y desventajas. En la tercera sección se explica el Cómputo Reconfigurable y se introducen los FPGAs mostrando sus características y además se presentan dos arquitecturas para el trabajo con SVM sobre FPGAs: la primera de ellas se encarga del entrenamiento de una Máquina de Soporte Vectorial mediante el algoritmo DSVM el cual fue diseñado específicamente para ser implementado sobre una arquitectura digital[10]. El segundo trabajo[11] de esta sección emplea las ventajas que ofrece la representación en el sistema numérico logarítmico en la clasificación de muestras desconocidas partiendo de que se tienen los modelos de entrenamiento.

2 Clasificación por Vectores de Soporte

El problema de la clasificación de datos, visto de manera general, puede ser restringido hasta considerarlo como un problema de clasificación entre dos clases sin que ocurra una pérdida de generalidad. El objetivo es separar los objetos en dos grupos mediante una función que es inducida a partir de los propios objetos. En la fig. 1 se muestran varios hiperplanos de separación que pueden separar lo datos, pero solamente uno maximiza la distancia entre las muestras más cercanas de las distintas clases. Esta distancia máxima es llamada *margen de separación* y el hiperplano que maximice la distancia entre él y las muestras más cercanas de ambas clases es llamado *hiperplano óptimo de separación*[12].

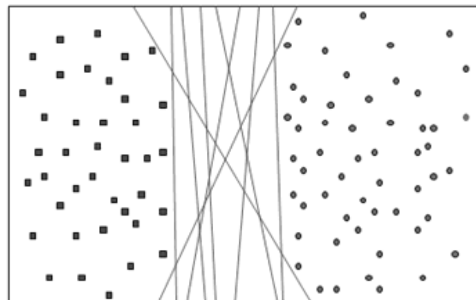


Fig. 1 Posibles hiperplanos separadores para muestras de dos clases

La clasificación binaria de datos usando SVM presenta tres variantes diferentes:

1. Linealmente separable.
2. Linealmente no separable.
3. Superficie de decisión no lineal.

2.1 Caso linealmente separable

La idea básica de la clasificación binaria con SVM es encontrar un hiperplano como superficie de decisión que separe los dos tipos de clases con el mayor margen posible.

La ecuación general del plano en n-dimensiones es:

$$w \cdot x = b$$

o lo que es lo mismo:

$$\langle w \cdot x \rangle + b = 0 \tag{1}$$

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

$$d_{min} = |b|/||w||$$

Pero existe un error de redundancia en (1) y sin que se pierda generalidad se pueden considerar que los hiperplanos canónicos[1] y sus parámetros $\boldsymbol{w}$ y b están restringidos por:

$$min_i |\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b| = 1 \tag{2}$$

La idea se ilustra en la Fig. 2 donde se muestra la distancia del punto más cercano a cada posible hiperplano de separación.

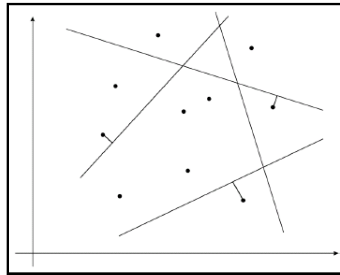


Fig. 2 Distancia más cercana de los datos a cada posible hiperplano de separación

Sea un problema de clasificación binaria donde la extracción de características se realiza inicialmente, se clasifican los datos de entrenamiento $\boldsymbol{x_i} \in \mathbb{R}^d$ con la etiqueta $y_i \in \{-1; +1\}$ para todos los datos de entrenamiento $i = 1... l$ donde $\boldsymbol{l}$ es la cantidad de datos y d es la dimensión del problema. Cuando dos clases son potencialmente linealmente separables en $\mathbb{R}^d$, se necesita encontrar un hiperplano separador el cual ofrezca el menor error de generalización entre el número infinito de posibles hiperplanos. Dicho hiperplano es aquel que maximice el margen de separación entre 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, los que son llamados *Vectores de Soporte*.

Lo anteriormente dicho se resume a que se cuenta con los patrones de entrenamiento $(\boldsymbol{x_1}, y_1), \cdots, (\boldsymbol{x_l}, y_l)$ donde $\boldsymbol{x_i}$ es un vector d-dimensional y

$y_i = 1$ Si $\boldsymbol{x_i}$ pertenece a la clase A
$y_i = -1$ Si $\boldsymbol{x_i}$ pertenece a la clase B

Si los datos son linealmente separables, entonces existe un vector d-dimensional $\boldsymbol{w}$ y un escalar b tales que

$$\boldsymbol{w} \cdot \boldsymbol{x_i} - b \geq 1 \quad \text{si } y_i = 1; \qquad \boldsymbol{w} \cdot \boldsymbol{x_i} - b \leq 1 \quad \text{si } y_i = -1; \tag{3}$$

De forma compacta, (2) puede escribirse como

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} - b) \geq 1 \tag{4}$$

o

$$-(y_i(w \cdot x_i - b) - 1) \leq 0$$

donde el par *(w, b)* define el hiperplano que separa las dos clases de datos. Con el propósito de hacer cada superficie de decisión *(w, b)* única, se normaliza la distancia perpendicular desde el origen al hiperplano separador dividiéndola por $||w||$, dando la distancia como $\frac{|b|}{||w||}$. Como se aprecia en la Fig. 3**,** la distancia perpendicular del origen al hiperplano H_1 ($\boldsymbol{w} \cdot \boldsymbol{x_i} - b = 1$) es $\frac{|1+b|}{||\boldsymbol{w}||}$ y del origen al hiperplano $\boldsymbol{H_2}$ **(**$\boldsymbol{w} \cdot \boldsymbol{x_i} - b = 1$**)** es $\frac{|b-1|}{||\boldsymbol{w}||}$**.** Aquellos objetos que se encuentran sobre los hiperplanos $\boldsymbol{H_1}$ **y** $\boldsymbol{H_2}$ son llamados *Vectores de Soporte*.

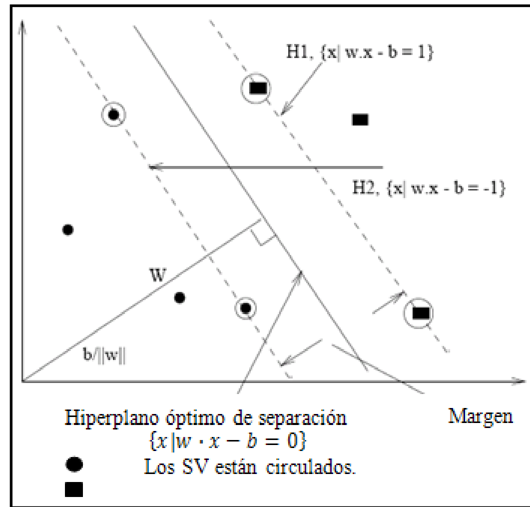


Fig. 3 Plano óptimo de separación y los Vectores de Soporte

Eliminar cualquiera de los puntos que no se encuentran sobre H_1 y H_2 no afecta el resultado de la clasificación, pero eliminar alguno de los vectores de soporte sí. La distancia entre los dos hiperplanos $\boldsymbol{H_1}$ y $\boldsymbol{H_2}$ es $\frac{2}{||w||}$, donde $\frac{1}{||w||}$ es el margen y determina la capacidad de la máquina de aprendizaje.

El objetivo para lograr una correcta clasificación es maximizar $\frac{2}{||w||}$ lo cual es equivalente a minimizar $\frac{||w||^2}{2}$. Es este punto se pude formular el problema como:

$$\text{Minimizar} \quad f = \frac{||\boldsymbol{w}||^2}{2}$$
$$\text{Sujeto a} \quad g_i = -(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} - b) - 1) \leq 1, \quad i = 1, \cdots, l$$

donde *l* es el número de muestras de entrenamiento.

Este problema puede resolverse usando las técnicas de Programación Cuadrática (*Quadratic Programming* o QP por sus siglas en inglés), sin embargo, usando el método de Lagrange[13] para resolver el problema hace fácil extender el análisis realizado al caso linealmente no separable.

La función Lagrangiana para este problema es:

$$
\begin{aligned}
L_p(\boldsymbol{w}, b, \boldsymbol{\Lambda}) &= f(\boldsymbol{x}) + \sum_{i=1}^{l} \lambda_i g_i(\boldsymbol{x}) \\
&= \frac{\boldsymbol{w}\cdot\boldsymbol{w}}{2} - \sum_{i=1}^{l} \lambda_i (y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} - b) - 1) \\
&= \frac{\boldsymbol{w}\cdot\boldsymbol{w}}{2} - \sum_{i=1}^{l} \lambda_i y_i \boldsymbol{w}\cdot\boldsymbol{x_i} + \sum_{i=1}^{l} \lambda_i y_i b + \sum_{i=1}^{l} \lambda_i
\end{aligned}
$$

donde $\Lambda = \{\lambda_1, \lambda_2, \cdots \lambda_l\}$ son los multiplicadores de Lagrange del conjunto de entrenamiento.

Las condiciones de Karush-Kuhn-Tucker (condiciones KKT) [14] para este problema son:

Condición del gradiente:

$$
\frac{\partial L_p}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{l} \lambda_i y_i \boldsymbol{x_i} = 0 \tag{5}
$$

donde $\frac{\partial L_p}{\partial \boldsymbol{w}} = \left(\frac{\partial L_p}{\partial \boldsymbol{w_1}}, \frac{\partial L_p}{\partial \boldsymbol{w_2}}, \cdots, \frac{\partial L_p}{\partial \boldsymbol{w_d}},\right)$

$$
\frac{\partial L_p}{\partial b} = \sum_{i=1}^{l} \lambda_i y_i = 0 \tag{6}
$$

$$
\frac{\partial L_p}{\partial \lambda_i} = g_i(\boldsymbol{x}) = 0 \tag{7}
$$

Condición de Ortogonalidad:

$$
\lambda_i g_i = -\lambda_i (y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} - b) - 1) = 0, \qquad i = 1, \cdots, l \tag{8}
$$

Condición de viabilidad:

$$
-y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} - b) + 1 \leq 0, \qquad i = 1, \cdots, l \tag{9}
$$

Condición de no-negatividad:

$$
\lambda_i \geq 0, \qquad i = 1, \cdots, l \tag{10}
$$

El punto estacionario del Lagrangiano determina las soluciones para el problema de optimización. Sustituyendo las ecuaciones (5) y (6) en la parte derecha de la función Lagrangiana se reduce la función a la forma dual con λ_i como variable dual. El problema después de la sustitución es:

Maximizar

$$
L_D = \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} \lambda_i \lambda_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j} \tag{11}
$$

Sujeto a:

$$\sum_{i=1}^{l} \lambda_i y_i = 0$$
$$\lambda_i > 0, \qquad i = 1, \dots, l.$$

Se pueden obtener todos los valores de λ resolviendo la ecuación **(11)** mediante QP y $\boldsymbol{w}$ puede ser obtenido usando la ecuación **(5)**:

$$\boldsymbol{w} = \sum_{i=1}^{l} \lambda_i y_i \boldsymbol{x_i}$$

Los valores de λ_i que sean mayores que cero para las restricciones dadas corresponden a los vectores de soporte del sistema mientras que los demás no son vectores de soporte.

Se puede calcular el valor de b usando la ecuación **(8)**:

$$\lambda_i(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} - b) - 1) = 0, \qquad i = 1, \dots, l \tag{12}$$

seleccionando el valor de $\boldsymbol{x_i}$ con λ distinto de cero.

La clase de un dato de entrada $\boldsymbol{x}$ es entonces determinada por:

$$class(\boldsymbol{x}) = sign(\boldsymbol{w} \cdot \boldsymbol{x} - b) \tag{13}$$

En la práctica, debido a implementaciones numéricas, el valor de la media de b es calculado mediante el promedio de todos los valores de b usando la ecuación **(12)**.

2.2 Caso linealmente no separable

Existen casos en que el hiperplano de separación lineal no existe, sin embargo aún así se puede buscar alguno que realice la clasificación introduciendo un conjunto de variables ξ las cuales medirán el grado de violación de las restricciones para el caso linealmente separable.

La formulación del problema sería para este caso:

Minimizar

$$f(\boldsymbol{w}, \Xi) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l} \xi$$

Sujeto a:

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} - b) \geq 1 - \xi_i, \qquad i = 1, \dots l$$
$$\xi_i \geq 0, \qquad i = 1 \dots, l$$

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

La función Lagrangiana quedaría:

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\lambda_i(y_i(\boldsymbol{x_i}\cdot\boldsymbol{w} - b) - 1 + \xi) - \sum_{i=1}^{l}\mu_i\xi_i$$

donde μ_i son los multiplicadores de Lagrange introducidas por la restricción $\xi_i \geq 0$.
Las condiciones KKT para este problema serían:

Condición del gradiente:

$$\frac{\partial L_p}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{l}\lambda_i y_i \boldsymbol{x_i} = 0 \tag{14}$$

donde $\frac{\partial L_p}{\partial \boldsymbol{w}} = \left(\frac{\partial L_p}{\partial \boldsymbol{w_1}}, \frac{\partial L_p}{\partial \boldsymbol{w_2}}, \cdots, \frac{\partial L_p}{\partial \boldsymbol{w_d}},\right)$

$$\frac{\partial L_p}{\partial b} = \sum_{i=1}^{l}\lambda_i y_i = 0 \tag{15}$$

$$\frac{\partial L_p}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \tag{16}$$

$$\frac{\partial L_p}{\partial \lambda_i} = -(y_i(\boldsymbol{x_i}\cdot\boldsymbol{w} - b) - 1 + \xi_i) \tag{17}$$

Condición de ortogonalidad:

$$\lambda_i(y_i(\boldsymbol{x_i}\cdot\boldsymbol{w} - b) - 1 + \xi_i) = 0, \qquad i = 1, \cdots, l \tag{18}$$

Condición de viabilidad:

$$(y_i(\boldsymbol{x_i}\cdot\boldsymbol{w} - b) - 1 + \xi_i) = 0, \qquad i = 1, \cdots, l \tag{19}$$

Condición de no-negatividad:

$$\xi_i \geq 0, \qquad i = 1, \cdots, l \tag{20}$$
$$\lambda_i \geq 0, \qquad i = 1, \cdots, l \tag{21}$$
$$\mu_i \geq 0, \qquad i = 1, \cdots, l \tag{22}$$
$$\mu_i\xi_i = 0, \qquad i = 1, \cdots, l \tag{23}$$

Sustituyendo las ecuaciones **(14)** y **(15)** en la parte derecha de la función Lagrangiana se obtiene el siguiente problema dual con las mismas variables Lagrangiana que en el caso anterior:

Maximizar

$$L_D = \sum_{i=1}^{l}\lambda_i - \frac{1}{2}\sum_{i,j=1}^{l}\lambda_i\lambda_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j}$$

Sujeto a:

$$0 \leq \lambda_i \leq C,$$

$$\sum_{i=1}^{l} \lambda_i y_i = 0$$

La solución para $\boldsymbol{w}$ puede ser determinada por la ecuación (14) la que describe una de las condiciones de KKT:

$$\boldsymbol{w} = \sum_{i=1}^{l} \lambda_i y_i \boldsymbol{x_i}$$

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 KKT:

$$\lambda_i(y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} - b) - 1 + \xi_i) = 0 \quad (24)$$
$$(C - \lambda_i)\xi_i = 0 \quad (25)$$

La ecuación (25) también indica que $\xi_i = 0$ si $\lambda_i < C$. Por lo tanto, b puede ser calculado solamente sobre aquellas muestras que cumplan que $0 < \lambda_i < C$.

Si $\lambda_i < C$ entonces $\xi_i = 0$. (Los vectores de soporte se encuentran a una distancia de $\frac{1}{\|w\|}$ del hiperplano separador (estos vectores son llamados *Vectores de Soporte Marginales*.) Cuando $\lambda_i = C$ los vectores de soporte son puntos mal clasificados si $\xi_i > 1$. Cuando $0 < \xi_i < 1$ los vectores de soporte son clasificados correctamente pero están más cerca que $\frac{1}{\|w\|}$ del hiperplano. Estos son los V*ectores de Soporte Frontera*. Ver la figura siguiente:

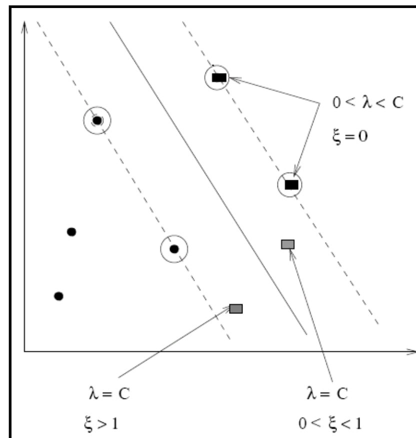


Fig. 4 Vectores de Soporte Marginales y Frontera

2.3 Superficie de decisión no lineal

En la mayoría de los casos en los que se precisa clasificar datos el hiperplano de separación no es una superficie lineal. Sin embargo, la teoría de SVM puede ser extendida para manejar estos casos. La idea que se sigue es llevar los datos de entrada $\boldsymbol{x}$ a un espacio de mayor dimensionalidad (un espacio de Hilbert de dimensiones finitas o infinitas) y allí realizar la separación lineal de los datos donde la separación lineal resulte más evidente. Esto sería:

$$\boldsymbol{x} \rightarrow \varphi(\boldsymbol{x}),$$
$$\boldsymbol{x} = (x_1, x_2, \dots, x_n),$$
$$\varphi(\boldsymbol{x}) = (\varphi_1(\boldsymbol{x}), \varphi_2(\boldsymbol{x}), \dots, \varphi_n(\boldsymbol{x}), \dots,)$$

La solución del SVM tiene la misma forma que para el caso linealmente separable, lo que ahora cuenta con la transformación entre los espacios de dimensionalidad diferentes anteriormente explicada.

$$clasificación = sign(\varphi(\boldsymbol{x}) \cdot \boldsymbol{w} - b) = sign\left(\sum_{i=1}^{l} y_i \lambda_i \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x}) - b\right) \quad (26)$$

No es necesario conocer la función φ para resolver este problema. Una función K llamada *función kernel* que cumpla con la propiedad que[15]:

$$K(\boldsymbol{x}, y) = \varphi(\boldsymbol{x}) \cdot \varphi(y)$$

simplificará la ecuación (26) hasta convertirla en:

$$sign\left(\sum_{i=1}^{l} y_i \lambda_i K(\boldsymbol{x_i}, \boldsymbol{x}) - b\right)$$

Entre los kernels más empleados están los siguientes:

- Lineal: $\boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i^T x_j}$
- Polinomial: $\boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{\gamma x_i^T x_j} + \boldsymbol{r})^{\boldsymbol{d}}, \boldsymbol{\gamma} > 0$
- RBF: $\boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{exp}\left(-\boldsymbol{\gamma}\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2\right), \boldsymbol{\gamma} > 0$
- Sigmoid: $\boldsymbol{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{tanh}(\boldsymbol{\gamma x_i^T x_j} + \boldsymbol{r})$

donde $\boldsymbol{\gamma}$, $\mathbf{r}$ y $\mathbf{d}$ son parámetros del kernel en cuestión.

Usando los Kernels, la ecuación (11) se puede reescribir como:

$$W(\lambda) = \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j K(\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \lambda_i \lambda_j$$

que en el caso de SVM se reduce a maximizar $\boldsymbol{W}$ sujeto a:

$$0 \le \lambda_i \le C \quad i = 1, \dots\dots, l \;\; \text{y} \;\; \textstyle\sum_{i=1}^{l} y_i \lambda_i = 0$$

el cual es un problema de QP.

La Fig. 5 representa el caso de clasificación donde la superficie de decisión no es lineal, se pueden ver los vectores de soporte y el hiperplano de separación.

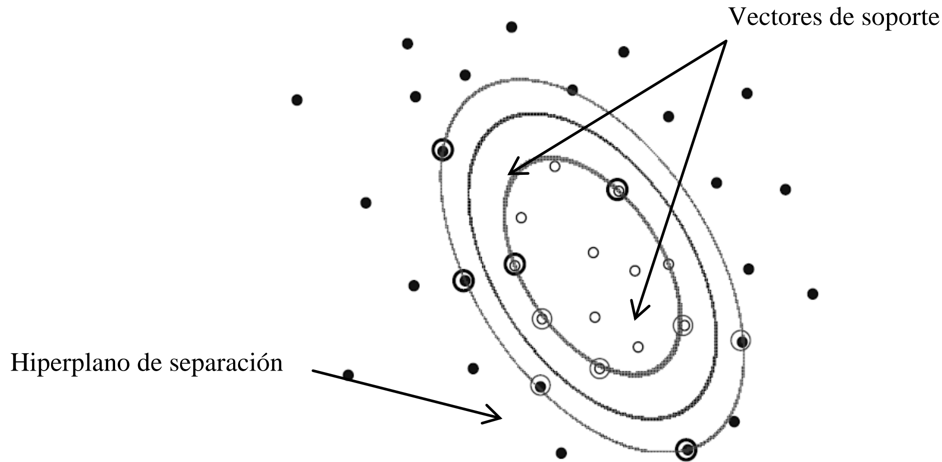


Fig. 5 Superficie de decisión no lineal

2.4 Algoritmos para el entrenamiento de SVM

Entrenar un SVM es equivalente a resolver un problema de QP N-dimensional, donde N es el número de muestras en la matriz de datos de entrenamiento. Resolver este problema usando las técnicas tradicionales de QP involucra grandes operaciones matriciales, las cuales son altamente consumidoras de tiempo y de recursos computacionales; las que en la mayoría de los casos no resultan prácticas y sí extremadamente lentas para problemas grandes. Por ejemplo, un conjunto de datos de entrenamiento de 4000×4000 cuyos elementos fuesen valores doubles de simple precisión no cabrían siquiera en 128 Mbyte de memoria[16]. Producto a esta limitante se han creado varios algoritmos para solucionar el problema del entrenamiento de SVM. A grandes rasgos, un pseudo-algoritmo para calcular SVM sería:

1. Seleccionar el parámetro C (que representa un compromiso entre minimizar el error de entrenamiento y la maximización del margen de separación entre los elementos de las clases), la función kernel y cualquier otro parámetro requerido por la función kernel empleada.
2. Resolver el problema de Programación Cuadrática o la formulación alternativa usando el algoritmo apropiado de Programación Cuadrática o Programación Lineal para obtener los vectores de soporte.
3. Obtener el umbral b usando los vectores de soporte ya calculados[17].

Existen varios algoritmos para el entrenamiento de SVM. Por los resultados reportados se destacan básicamente tres[3]:

Chunking: Este algoritmo fue propuesto por Vapnik en [4]. Utiliza la gran esparsidad que poseen los valores obtenidos para los multiplicadores de Lagrange, los cuales varían entre cero y el límite superior dado por C. El valor de la forma cuadrática será idéntico para el caso en que se mantienen todas las muestras y para el caso que se eliminan las filas y columnas de la matriz de muestras de entrenamiento que correspondan a multiplicadores de Lagrange con valor cero y algunos de aquellos que violen las condiciones de KKT. Por lo tanto, los grandes problemas de QP pueden descomponerse en una serie de pequeños subproblemas cuyo objetivo es identificar los multiplicadores de Lagrange con valores iguales a cero para descartarlos. Las dimensiones del problema varían pero son finalmente igual al número de multiplicadores de Lagrange diferentes de cero. Esta técnica está limitada por el número máximo de vectores de soporte que se pueden manejar y además requiere de optimizadores cuadráticos para resolver la secuencia de subproblemas de optimización[18]. Los subproblemas de optimización que se obtienen con este algoritmo continúan siendo demasiado grandes para poder realizar el entrenamiento de SVM de

manera adecuada puesto que consume demasiado tiempo y requiere de demasiada memoria para enfrentarse a los subproblemas de optimización.

Osuna: Osuna et al en [19] proponen un algoritmo para entrenar las SVMs que pretende hacer frente a las limitantes que presenta Chunking en cuanto a la selección de los objetos que formarán los subproblemas y el tamaño de éstos subproblemas. Este algoritmo, al igual que el anterior, se basa en dos suposiciones **1)** el número de vectores de soporte es pequeño respecto al número total de muestras de entrenamiento; **2)** el número total de vectores de soporte no excede a unos cuantos miles (menos de 3000). Se han desarrollado variantes a este algoritmo como son SVM^{Light} [6] para la clasificación y SVM_{Torch} [20] para la regresión.

Para este algoritmo se definen dos conjuntos de índices de particionamiento del conjunto de entrenamiento: $\boldsymbol{B}$ y $\boldsymbol{N}$. El algoritmo se basa en la suposición que se puede definir un conjunto de tamaño fijo $\boldsymbol{B}$ tal que $|\boldsymbol{B}| \leq l$, donde l es el número de muestras de entrenamiento. Además, $\boldsymbol{B}$ debe ser lo suficientemente grande para contener todos los vectores de soporte (o lo que es lo mismo, cuyos valores de $\lambda_i > 0$) pero lo suficientemente pequeño tal que la computadora pueda manejarlo. Bajo estas condiciones, este algoritmo puede ser enunciado como sigue:

1. Arbitrariamente seleccionar $|\boldsymbol{B}|$ muestras del conjunto de entrenamiento.
2. Resolver el problema de optimización definido por las muestras en $\boldsymbol{B}$.
3. Mientras exista algún $j \in N$ tal que $g(x_j)y_j < 1$ donde:

$$g(x_j) = \sum_{p=1}^{l} \lambda_p y_p K(x_j, x_p) + b$$

Reemplazar $\lambda_i = 0$, $i \in B$ con $\lambda_j = 0$ y resolver el nuevo subproblema.

Como se puede ver, la complejidad de este algoritmo aumenta en correspondencia con la cantidad de muestras de entrenamiento y no es factible para problemas de varias decenas de miles de muestras de entrenamiento (ver la Fig. 6).

El trabajo [19] fue presentado en marzo de 1997 y ya en septiembre aparece [7], una modificación al algoritmo presentado anteriormente que pretendía mejorar sus limitaciones. En este nuevo algoritmo no se tienen en cuenta las suposiciones **1**) y **2**) enunciadas anteriormente. Igual que en [19] se definen dos conjuntos de índices $\boldsymbol{B}$ y $\boldsymbol{N}$ y presenta igualmente dos puntos medulares. La primera proposición plantea que si se mueve una muestra de $\boldsymbol{B}$ a $\boldsymbol{N}$ ésto hace que la función de costo se mantenga inalterable, y la solución es factible en el subproblema. La segunda proposición plantea que mover una muestra que viola la condición de optimalidad de $\boldsymbol{N}$ a $\boldsymbol{B}$ da una mejora estricta en la función de costo cuando el problema es re-optimizado (ver [7] para la demostración).

Tomando como base las proposiciones anteriores es que se formula el algoritmo de descomposición:

1. Se escogen arbitrariamente $|\boldsymbol{B}|$ muestras del conjunto de entrenamiento.
2. Resolver el subproblema definido por las muestras en $\boldsymbol{B}$.
3. Mientras exista algún $j \in N$ tal que:
   a. $\lambda_j = 0$ y $g(x_j)y_j < 1$
   b. $\lambda_j = C$ y $g(x_j)y_j > 1$
   c. $0 < \lambda_j < C$ y $g(x_j)y_j \neq 1$

   Reemplazar los λ_i, $i \in B$ con λ_j y resolver el nuevo subproblema.

Este algoritmo sugiere que se adicione y se elimine una muestra en cada iteración, lo cual puede ser ineficiente debido a que debería utilizar un paso completo de optimización para obligar a que

una muestra obedezca las condiciones de KKT. En la práctica, los investigadores adicionan y restan múltiples muestras acorde a una heurística no publicada.

SMO: (*Sequential Minimal Optimization* o *SMO*) puede resolver los problemas de QP generados por SVM sin agregar ninguna matriz extra de almacenamiento y sin usar optimizaciones numéricas. SMO descompone el problema de QP en pequeños subproblemas de QP bajo determinadas condiciones y emplea el teorema de Osuna et al en [7] para garantizar la convergencia y cada subproblema es resuelto por separado[5].

A diferencia de los otros métodos, SMO selecciona el menor problema de optimización posible en cada paso del algoritmo: dos multiplicadores de Lagrange ya que los multiplicadores de Lagrange deberán obedecer la restricción de igualdad lineal. De esta forma, en cada iteración SMO selecciona 2 multiplicadores de Lagrange para optimizar conjuntamente, encuentra el valor óptimo para estos dos multiplicadores y actualiza las variables de SVM para reflejar los nuevos valores óptimos. De esta manera se necesitan más iteraciones para lograr la convergencia que los algoritmos anteriores pero los cálculos necesarios para realizar el paso de optimización con 2 multiplicadores de Lagrange es notablemente más rápido que cuando se seleccionan varios multiplicadores.

Básicamente, SMO está compuesto por dos elementos básicos: un método analítico para realizar la optimización con los dos multiplicadores de Lagrange escogidos; y una heurística para seleccionar con cuáles multiplicadores se realizará la optimización.

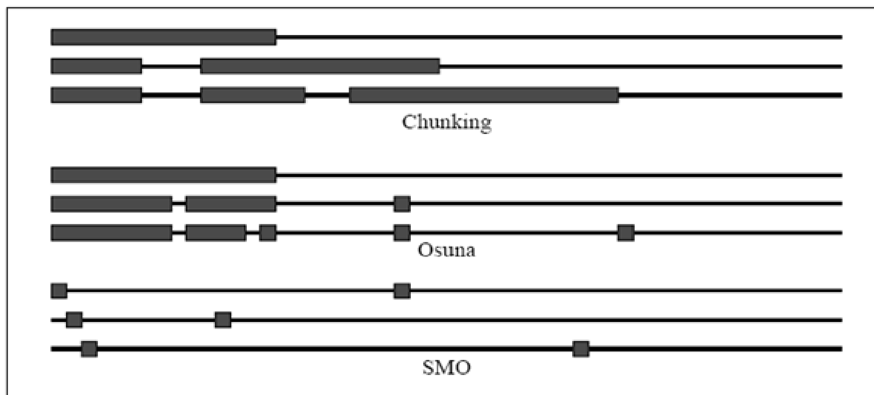


Fig. 6 En la figura se muestran los tres métodos analizados para resolver SVM: Chunking, el algoritmo de Osuna y SMO. Cada línea horizontal representa el conjunto de entrenamiento y los cuadros representan los multiplicadores de Lagrange que serán optimizados en cada paso. En Chunking el número de muestras a entrenar por paso tiende a crecer. Para el algoritmo de Osuna, un número fijo de muestras serán optimizadas por cada paso, las cuales serán insertadas y eliminadas del problema en cada paso. Para SMO solamente dos muestras son analíticamente optimizadas en cada paso por lo que cada paso es más rápido

A pesar de que con SMO se obtienen mejores resultados en el desempeño que otros algoritmos, resulta demasiado lento para enfrentarse a problemas realmente grandes. En [16] se prueba que SMO puede tornarse ineficiente cuando las muestras exceden un determinado número, por lo tanto, ya que SMO es el algoritmo más rápido para entrenar una SVM no habrá manera a enfrentarse a determinados problemas. Cuando se trata de un conjunto de entrenamiento muy grande el proceso será especialmente lento ya que sólo se optimizará una pareja de muestras en cada iteración. Muchas de estas iteraciones serán empleadas en un despreciable avance en la optimización o deshaciendo cambios previos. Además, si los datos son muy grandes, la memoria caché del algoritmo (caché de errores, etc.) debería ser limpiada cada cierto número de iteraciones, lo cual lejos de ayudar al desarrollo del algoritmo constituiría un limitación.

Pero no solamente las dimensiones del conjunto de entrenamiento es el único problema a enfrentar. Por ejemplo, incrementar el valor del parámetro C puede hacer que la clasificación mejore, pero incrementaría dramáticamente el tiempo para llegar a la solución[16]. En [21] se identifica lo que a juicio de los autores es el punto de ineficiencia de SMO: se calcula la desviación del clasificador en cada iteración, la que frecuentemente no es exacta y por lo tanto causa que la selección de las muestras a optimizar no sea la más eficiente y que además incrementa el tiempo de procesamiento del algoritmo. En ese mismo trabajo se muestra un ejemplo de la principal limitación de SMO y afirman que su propuesta acelera en 2 veces el desempeño del SMO clásico. Básicamente, la modificación que se hace es emplear dos variables para el umbral b: b_{up} y b_{low} y emplean otras restricciones para chequear la optimalidad (ver [21] para mayor referencia). A todo lo antes mencionado puede sumársele el costo que implica el cálculo de la función kernel para datos de alta dimensionalidad.

Los algoritmos mencionados anteriormente son variantes para resolver problemas de programación cuadrática que por demás es una tarea lenta que requiere demasiada memoria extra y un profundo conocimiento de análisis numérico. Como se demostró en [22], [8],[23] , SMO resultó ser entre todos los algoritmos analizados el más eficiente y que mejores resultados ofrece. En este punto ya se está en condiciones de ofrecer una descripción de SMO, donde se usará el pseudocódigo original propuesto por Platt en [5]:

```
target = desired output vector
point = training point matrix
procedure takeStep( i1,i2 )
     if ( i1 == i2 ) return 0
     alph1 = Lagrange multiplier for i1
     y1 = target[i1]
     E1 = SVM output on point[i1] – y1 (check in error cache)
     s = y1 * y2
     Compute L, H via equations (13)[2] and (14)[3]
     if ( L == H )
       return 0
     k11 = kernel( point[i1], point[i1] )
     k12 = kernel( point[i1], point[i2] )
     k22 = kernel( point[i2], point[i2] )
     eta = k11 + k22 – 2 * k12
     if ( eta > 0 )
     {
       a2 = alph2 + y2 * (E1-E2)/eta
       if ( a2 < L ) a2 = L
       else if ( a2 > L ) a2 = H
     }
     else
     {
       Lobj = objective function at a2 = L
       Hobj = objective function at a2 = H
       if ( Lobj < Hobj - eps )
```

[2]En el artículo original de Platt, eq. 13: $L = \max(0, \alpha_2 - \alpha_1)$, $H = min(C, C + \alpha_2 - \alpha_1)$
[3]En el artículo original de Platt, eq. 14: $L = \max(0, \alpha_2 + \alpha_1 - C)$, $H = min(C, \alpha_2 + \alpha_1)$

```
            a2 = L
          else if ( Lobj > Hobj + eps)
            a2 = H
          else
            a2 = alph2
        }
        if ( |a2 - alph2| < eps * (a2 + alph2 + eps))
          return 0
        a1 = alph1 + s * (alph2 - a2)
        Update threshold to reflect change in Lagrange Multipliers
        Update weight vector to reflect change in a1 & a2, if SVM is linear
        Update error cache using new Lagrange Multipliers
        Store a1 in the alpha array
        Store a2 in the alpha array
        Return 1
endprocedure

procedure examineExample(i2)
        y2 = target[i2]
        alph2 = Lagrange multiplier for i2
        E2 = SVM output on point[i2] - y2 (check in error cache)
        r2 = E2 * y2
        if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))
        {
          if ( non-zero & non-C alpha > 1)
          {
            i1 = result of second choice heuristic (section 2.2)
            if takeStep(i1,i2)
              return 1
          }
          Loop over all non-zero and non-C alpha, starting at a random point
          {
            i1 = identity of current alpha
            if takeStep( i1,i2 )
              return 1
          }
          Loop over all possible i1, starting at a random point
          {
            i1 = loop variable
            if takeStep( i1, i2 )
              return 1
          }
        }
        return 0
endprocedure

main routine:
        numChanged  = 0;
        examineAll = 1;
        while ( numChanged > 0 | examineAll)
        {
          numChanged = 0;
          if ( examineAll )
```

```
        loop I over all training examples
          numChanged += examineExample(I)
      else
        loop I over examples where alpha is not 0 & not C
          numChanged += examineExample(I)
      if ( examineAll == 1)
        examineAll == 0
      else if (numChanged == 0)
        examineAll = 1
    }
endRoutine
```

Para mayor claridad, se reproduce en un sumario el algoritmo de Platt:

1. Iterar sobre todas las muestras del conjunto de entrenamiento buscando un λ_1 que viole las condiciones de Karush-Kuhn-Tucker.
    a. Si λ_1 es encontrado, entonces ir al paso 2.
    b. Si termina de iterar sobre el conjunto de entrenamiento y no se encuentra ningún λ_1, entonces iterar sobre el conjunto de muestras cuyos λ son distintos de 0 o C (conjunto de las no fronteras.)
    c. Si λ_1 es encontrado, entonces va al paso 2
    d. Se alternarán las iteraciones sobre todo el conjunto de entrenamiento y el conjunto de las no fronteras buscando un λ_1 que viole las condiciones de KKT hasta que todos los λ de las muestras cumplan con la condición de KKT.
    e. Termina.
2. Buscar un λ_2 en el conjunto de las no fronteras.
    a. Calcular E_1 y E_2
    b. Tomar el λ que obtenga el mayor valor de $|E_1\text{-}E_2|$ como λ_2.
    c. Si las dos muestras son iguales, entonces ir al paso 3. Si no, calcular los valores de L y H para λ_2.
    d. Si $L = H$ entonces la optimización no puede realizarse. Se desecha este valor de λ_2 y va al paso 3. De otra forma, se calcula el valor de η.
    e. Si η es negativo, entonces se calcula un nuevo valor para λ_2.
    f. Si η es positivo, entonces calcular la función objetivo para los puntos L y H y usar el λ que de mayor resultado para la función objetivo como λ_2.
    g. Si $|\lambda_2^{Nuevo} - \lambda_2^{Viejo}|$ es menor que ξ, entonces abandonar λ_2 e ir a 3. De otra forma, ir a 4.
3. Iterar sobre el conjunto de las no fronteras comenzando en un punto aleatorio hasta que λ_2 descrito en proceso de la optimización 2 sea encontrado.
    a. Si no es encontrado, entonces iterar sobre todo el conjunto de entrenamiento hasta que un λ_2 que haga el progreso de la optimización sea encontrado.
    b. Si no se encuentra ningún λ_2 en esas 2 iteraciones, entonces se salta λ_1 encontrado y regresa al paso 1 para encontrar un nuevo λ que viole las condiciones de KKT.
4. Calcular el nuevo valor de λ_1
    a. Actualizar el valor del umbral b, la caché de error y almacenar los nuevos valores de λ_1 y λ_2.

Ir al paso 1.

3 Cómputo Reconfigurable

La computación reconfigurable es un campo de investigación y desarrollo relativamente nuevo: las primeras investigaciones comenzaron a finales de los 80. Se trata de un intento de superar el

tradicional desfase entre hardware y software en el campo de la Informática y dotar al hardware de la flexibilidad del software para lograr que un mismo hardware se someta a los requerimientos sean cuales sean éstos durante el tiempo.

3.1 Conceptos básicos

Existen dos enfoques para implementar y ejecutar un algoritmo. El primero se refiere al uso de *Application Specific Integrated Circuits* (ASIC, por sus siglas en inglés). Un ASIC es un dispositivo electrónico, diseñado específicamente para realizar operaciones de propósito específico. Solo están formados por los elementos de hardware indispensables para ejecutar la tarea para la que fueron diseñados, y es por ello que tienen alto rendimiento. Sin embargo, la nula flexibilidad en la modificación de estos elementos es una gran desventaja. El circuito no puede ser alterado después de ser fabricado. Si cualquier elemento del chip necesita ser modificado, debe pasar por un proceso de rediseño y refabricación, el cual es muy costoso tanto económicamente como en tiempo.

El segundo enfoque recurre al uso de microprocesadores programados a través de software. Los microprocesadores ejecutan un conjunto de instrucciones para realizar las operaciones que necesita un algoritmo. Al cambiar las instrucciones del software, la funcionalidad del sistema es alterada sin que sea necesario modificar el hardware, lo que permite tener una gran flexibilidad para implementar diversos tipos de algoritmos. Sin embargo, esta flexibilidad afecta directamente al rendimiento en la ejecución de un algoritmo. El microprocesador, antes de poder ejecutar cualquier instrucción, primero debe leerla de la memoria y decodificarla para saber qué acción debe realizar. En consecuencia se introduce un retraso en la ejecución de cada instrucción. Más aún, si el algoritmo necesita de alguna operación que no existe en el conjunto de instrucciones del microprocesador, ésta debe construirse a partir de las existentes.

El cómputo reconfigurable toma las ventajas de ambos enfoques. Mientras trata de mantener un alto rendimiento respecto al tiempo de ejecución del software, también trata de dar la flexibilidad necesaria en hardware que los ASIC no tienen. Los primeros conceptos del cómputo reconfigurable se remontan a la década de los 60's [24], pero fue hasta mediados de los 80's cuando el área tomó relevancia gracias a la introducción de los FPGAs. Estos dispositivos ofrecen la flexibilidad que los ASIC no tienen, ya que diversos algoritmos pueden ejecutarse en un FPGA con tan solo configurarlo cuando se quiera modificar su comportamiento. Esta flexibilidad es análoga a ejecutar diversos algoritmos en software sobre un microprocesador. Otra ventaja importante que tienen las implementaciones de algoritmos en FPGAs, es un aumento considerable en la velocidad de ejecución respecto a implementaciones hechas solo en software. Por ejemplo, el hardware podría ejecutar un algoritmo 10 o hasta 100 veces más rápido que el mismo implementado en software. Esta ventaja se deriva del hecho de que el hardware programable es adaptado a un algoritmo en particular tal como se haría con un ASIC, incluyendo solo los elementos necesarios para la ejecución de las operaciones que requiere dicho algoritmo.

Las primeras aplicaciones del cómputo reconfigurable estuvieron orientadas solamente al procesamiento de señales e imágenes. Recientemente, existen otras aplicaciones entre las que se incluyen: algoritmos de cifrado, reconocimiento automático de objetivos y compresión de datos por solo por mencionar algunos.

3.2 FPGA

Los circuitos integrados son omnipresentes en la sociedad moderna. Una de sus alternativas, los circuitos FPGA son una variante sumamente atractiva a los desarrolladores e ingenieros no solo por sus costos relativamente accesibles sino por las particularidades que presentan y por las ventajas que trae su uso.

Los FPGA aparecen en el mercado en 1984 como continuidad lógica de los CPLDs y se puede decir que mejoran en muchos aspectos a éstos últimos. Si se mide la densidad de los elementos lógicos programables en compuertas lógicas equivalentes (número de compuertas NAND equivalentes que se podrían programar en un dispositivo) un CPLD contiene valores del orden de decenas de miles de compuertas equivalentes mientras que un FPGA contiene un número de compuertas del orden de cientos de miles hasta varios millones de ellas. Aparte de las diferencias en densidad entre ambos tipos de dispositivos, la diferencia fundamental entre las FPGAs y los CPLDs es su arquitectura. La arquitectura de los CPLDs es más rígida y consiste en una o más sumas de productos programables cuyos resultados van a parar a un número reducido de biestables síncronos o flip-flops. La arquitectura de las FPGAs, por otro lado, se basa en un gran número de pequeños bloques utilizados para reproducir sencillas operaciones lógicas, que cuentan a su vez con flips-flops. La enorme libertad disponible en la interconexión de dichos bloques confiere a las FPGAs una gran flexibilidad con lo cual se logra desarrollar en los FPGAs funciones de alto nivel como sumadores y multiplicadores embebidas en la propia matriz de interconexiones permitiendo construir desde una simple compuerta AND hasta un microprocesador. Con el empleo de FPGAs se pueden desarrollar circuitos a la medida sin los riesgos tecnológicos asociados a las otras opciones de desarrollo. Los FPGAs tienen aplicaciones en todas las áreas donde se requiera computación de alta velocidad y con fines específicos y sus aplicaciones son muy diversas.

Las principales características de las FPGAs son:

- Alta complejidad.
- Bajo costo de desarrollo.
- Fácil de depurar.
- Tolerante a errores.
- Pocas unidades.
- Tamaño reducido.
- Fiabilidad alta.
- Área-time-power intermedio.
- Confidencialidad baja.

El FPGA es un componente estándar (re)programable por el usuario. Esto Implica que la interconexión debe ser (re)programable y las funciones lógicas y la entrada/salida también deben ser (re)programables.

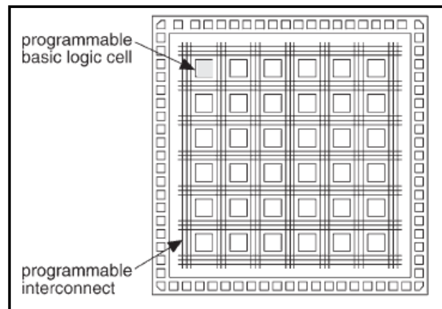


Fig. 7 Aspecto genérico de un FPGA: un array de interconexiones y tablas de look-ups configurables externamente por una corriente de bits en serie.

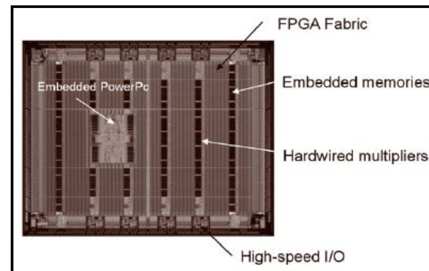


Fig. 8 Microfotografía de una FPGA actual, donde además de los componentes anteriores existen bloques HW tales como procesadores, transceivers de comunicaciones serie, memorias, y multiplicadores.

Debido a las ventajas que presentan las FPGAs, la mira de los investigadores está dirigida a su uso en aplicaciones donde el poder de procesamiento sea elevado. Actualmente, los FPGAs están siendo empleados en investigaciones referentes a la criptografía, decodificación de video y análisis

del habla. Aquellos que emplean FPGAs en sus estudios se basan en la premisa de que si se implementan algoritmos rápidos por software en un dispositivo FPGA, éstos deberían acelerarse notablemente, lo cual no siempre es cierto o la relación costo-beneficio no favorece este tipo de desarrollos.

3.3 Flujo de diseño

Los sistemas reconfigurables regularmente están formados por un elemento de lógica reprogramable (usualmente uno o más FPGAs) y un microprocesador de propósito general (ver la figura 9). El microprocesador realiza las tareas que no pueden ser realizadas eficientemente por la lógica reprogramable, además de programar y controlar el dispositivo reprogramable. El FPGA tiene la función de un coprocesador y realiza las tareas que al software le tomarían la mayor parte del tiempo, en especial, operaciones en donde no exista dependencia de datos, y así poder realizar varias acciones al mismo tiempo dentro del FPGA.

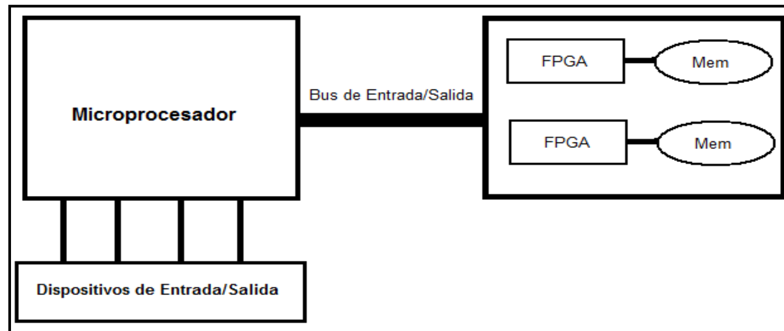


Fig. 9 Sistema de Cómputo reconfigurable basado en un microprocesador de propósito general y uno o varios FPGAs

El flujo de diseño que se sigue para implementar algoritmos bajo este tipo de sistema de cómputo reconfigurable se ilustra en la figura 10. El paso inicial consiste en particionar el algoritmo, para así identificar, a criterio del diseñador, aquellas partes que puedan ser implementadas eficientemente ya sea en hardware o en software.

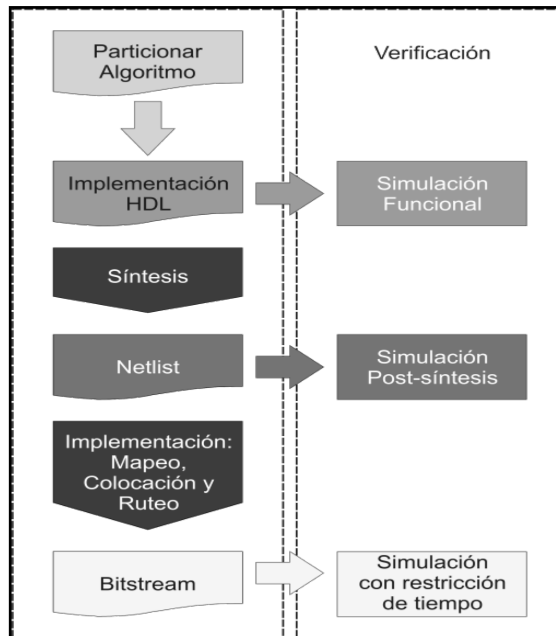


Fig. 10 Flujo del diseño para FPGA

Para cada sección que se implemente en hardware, se crea un diseño utilizando los medios disponibles para ello, como lo pueden ser lenguajes de descripción de hardware (Hardware Description Languages, HDL) como VHDL, Verilog o System C por mencionar algunos; o también herramientas para el diseño de esquemáticos. Ambos medios describen el funcionamiento que el FPGA asumirá en tiempo de ejecución.

Posteriormente, estos diseños son sintetizados (Systhesis phase) a nivel de compuertas y registros, con lo que se obtiene un netlist de elementos lógicos, aún independiente de la tecnología del FPGA a usar.

La netlist es mapeada (Map phase) hacia los bloques lógicos del FPGA, es decir, el diseño lógico obtenido en la síntesis, es convertido a un diseño que involucra a la tecnología utilizada en los bloques lógicos del FPGA objetivo.

En la etapa de colocación (Place phase) se asignan las funciones lógicas en bloques particulares del FPGA. Enseguida, estos bloques son ruteados (Routing phase) por medio de las interconexiones programables para comunicar unos bloques con otros de acuerdo al diseño planteado. Al final del flujo de diseño, se obtiene el archivo de programación (bitstream) con el cual se configurará el FPGA.

En cualquiera de sus etapas, el flujo de diseño puede realizarse de forma manual o automática. La forma automática requiere un mínimo esfuerzo del programador para la creación de los diseños, además no necesita tener conocimiento acerca de la tecnología del FPGA donde se implementará el diseño. Sin embargo, si se comparara contra una implementación manual, la implementación automática podría tener un menor rendimiento en la ejecución del algoritmo, y el diseño resultante podría ocupar una mayor cantidad de los recursos disponibles del FPGA.

La forma manual resulta en una implementación mucho más eficiente en uso de recursos y rendimiento, pero se requiere que el programador conozca muy bien la tecnología y estructura interna del FPGA objetivo. Además, constantemente se deben realizar procesos de simulación y depuración, no solo para asegurar el correcto funcionamiento del circuito, sino también para prevenir posibles conexiones equivocadas que puedan causar cortos dentro del FPGA, y en consecuencia dañarlo.

A continuación se describen los trabajos más relevantes donde se ha implementado SVM sobre FPGAs para la clasificación de datos.

3.4 DSVM

Probablemente uno de los trabajos más citados sobre FPGAs y SVM fue presentado por Davide Anguita et al en [25]. En este trabajo los autores describen la implementación de kernels digitales para clasificadores, especialmente se refieren al perceptrón. Aquí los autores se basan en una arquitectura digital de punto fijo y ofrecen una variante para el perceptrón llamada Digital Kernel Perceptron. Comparan sus resultados con los resultados obtenidos por software para las operaciones de punto flotante y fijo obteniendo similares resultados en cuanto a métricas entre los distintos modelos probados y consideraron al diseño por hardware como la mejor opción dado a lo reducido del diseño y a los resultados obtenidos. En un trabajo posterior de los mismos autores [10] proponen una arquitectura digital para el aprendizaje mediante Support Vector Machines y discuten su implementación en un FPGA. En este trabajo se analiza brevemente los efectos de cuantización en el performance del SVM en problemas de clasificación para mostrar su robustez frente a las implementaciones de las operaciones matemáticas de punto fijo. La arquitectura que se propone implementa un algoritmo menos sensible a los errores de cuantización respecto a los algoritmos reportados en la literatura.

La solución que se propone en [10] consta de 2 partes: el algoritmo FIBS y el algoritmo DSVM donde el primero, iterativamente, invoca al segundo para calcular los parámetros de SVM con un valor de b fijo y luego, con estos parámetros y mediante un proceso de bisección, se calculan los nuevos valores de b hasta que el criterio de convergencia sea alcanzado. Una vez terminado el proceso los vectores de soporte son aquellos tales que $SV = \{i: \alpha_i \epsilon (0, C)\}$.

Este algoritmo utiliza las potencialidades de los FPGAs para resolver los problemas de QP que surgen al realizar el aprendizaje por SVM. Para llegar a esta implementación se parte de la idea de diseñar un algoritmo de aprendizaje en dos fases las cuales trabajarían iterativamente. La primera parte se encarga de resolver los problemas de QP para un valor de b fijo (en el origen en este caso) mientras que el segundo implementa un procedimiento para actualizar el valor de b con objetivo de obtener iterativamente el valor óptimo de b. El algoritmo DSVM puede ser visto como un dispositivo al cual se le suministra la matriz kernel Q, el umbral fijo b, y comenzando con un valor inicial de $\boldsymbol{\alpha}^0$, el algoritmo resuelve los problemas de QP dando una solución intermedia $\boldsymbol{\alpha}^i$. El segmento DSVM es el encargado de calcular los valores de $\boldsymbol{\alpha}$ (multiplicadores de Lagrange) dados la matriz Kernel, un valor inicial para b y un valor inicial para $\boldsymbol{\alpha}^i$. Los multiplicadores de Lagrange se calculan mediante un proceso iterativo cuyas operaciones se realizan de manera concurrente aprovechando las facilidades que ofrecen los FPGA en el paralelismo de instrucciones. Estos valores de $\boldsymbol{\alpha}$ son valores intermedios obtenidos para el b intermedio suministrado a DSVM. Con $\boldsymbol{\alpha}$ se calcula en FIBS un b más refinado a el cual se suministra a DSVM para obtener un $\boldsymbol{\alpha}$ más exactos y este proceso se repite hasta que la condición de convergencia se alcance y se obtenga el b optimo y a partir de él, los valores finales para los multiplicadores de Lagrange.

El segmento FIBS se ejecuta de manera secuencial. Tiene dos secciones principales donde en la primeramente se obtienen los valores de α con valores de b fijos mediante DSVM. En dependencia de la evaluación de $s = sgn(y^T \alpha')$, para los α obtenidos en la fase anterior, se obtienen los valores de los extremos posibles dónde se encuentra el valor óptimo de b al cual se llega mediante un proceso de bisección y se repite el proceso nuevamente para el nuevo valor de b calculado.

A pesar de que este algoritmo presenta varias estructuras secuenciales, fue pensado y creado para ejecutarse sobre un FPGA aprovechando las potencialidades que estos presentan para paralelizar procesos ya que las operaciones básicas de algoritmo sí pueden ejecutarse concurrentemente. La idea en la que se basa este algoritmo radica en el hecho de que el uso de un

kernel Gaussiano mapea los datos a un espacio de características infinito donde el efecto de eliminar algún parámetro del SVM es despreciable. En particular, con esta idea si se hace $b = 0$ se está forzando a que el hiperplano separador pase por el origen del espacio de características y la restricción de igualdad desaparece de la formulación del problema de QP. Esta idea es sólo aplicada a un kernel Gaussiano y no se explica en éste artículo cuáles serían los efectos si se empleara otro tipo de kernel.

A continuación se muestra el algoritmo *DSVM*:

ALGORITHM 1: DSVM WITH FIXED BIAS

Step	Description
1.	input: Q, b, $\boldsymbol{\alpha}^0$;
2.	set $\tilde{\boldsymbol{r}} = (\boldsymbol{r} - \boldsymbol{y}b)$; $\eta = \frac{2}{m \max_{i,j}(q_{ij})}$
3.	set endrun=false; $k = 0$
4.	while not endrun do
5.	$\boldsymbol{g} = \left(-Q\boldsymbol{\alpha}^k + \tilde{\boldsymbol{r}}\right)$
6.	$\boldsymbol{z}^{k+1} = \boldsymbol{\alpha}^k + \eta\boldsymbol{g}$
7.	$\alpha_i^{k+1} = \max\left(0, \min\left(z_i^{k+1}, C\right)\right), \forall i = 1, \ldots, m$
8.	if $\forall i \left(\alpha_i^{k+1} - \alpha_i^k\right) \leq \varepsilon$ then
8.	endrun=true
10.	else k=k+1
11.	enddo
12.	output: $\boldsymbol{\alpha}' = \boldsymbol{\alpha}^{k+1}$

Tabla 1 Algoritmo *DSVM* para un valor de b fijo[10]

ALGORITHM 2: FIBS

Step	Description
1.	$b_{low} = b = -1; \quad \boldsymbol{\alpha}^0 = 0; \quad b_{up} = +1;$ endFibs=false
2.	$\boldsymbol{\alpha}' = \mathcal{A}_b(Q, b, \boldsymbol{\alpha}^0); \quad s = \text{sgn}\left(y^T\boldsymbol{\alpha}'\right); \boldsymbol{\alpha}^0 = \boldsymbol{\alpha}'$
3.	if $s < 0$ then
4.	while $s < 0$ do
5.	$b_{up} = b; \quad b = 2b; \quad b_{low} = 2b_{low}$
6.	$\boldsymbol{\alpha}' = \mathcal{A}_b(Q, b, \boldsymbol{\alpha}^0); \quad s = \text{sgn}\left(y^T\boldsymbol{\alpha}'\right); \boldsymbol{\alpha}^0 = \boldsymbol{\alpha}'$
7.	enddo
8.	else if $s > 0$ then
9.	$b = b_{up}$
10.	$\boldsymbol{\alpha}' = \mathcal{A}_b(Q, b, \boldsymbol{\alpha}^0); \quad s = \text{sgn}\left(y^T\boldsymbol{\alpha}'\right); \boldsymbol{\alpha}^0 = \boldsymbol{\alpha}'$
11.	if $s > 0$ then
12.	while $s > 0$ do
13.	$b_{low} = b; \quad b = 2b; \quad b_{up} = 2b_{up}$
14.	$\boldsymbol{\alpha}' = \mathcal{A}_b(Q, b, \boldsymbol{\alpha}^0); \quad s = \text{sgn}\left(y^T\boldsymbol{\alpha}'\right); \boldsymbol{\alpha}^0 = \boldsymbol{\alpha}'$
15.	enddo
16.	else endFibs = true
17.	while not endFibs do
18.	$b = (b_{low} + b_{up})/2$
19.	$\boldsymbol{\alpha}' = \mathcal{A}_b(Q, b, \boldsymbol{\alpha}^0); \quad s = \text{sgn}\left(y^T\boldsymbol{\alpha}'\right); \boldsymbol{\alpha}^0 = \boldsymbol{\alpha}'$
20.	if $s = 0$ or $(b_{low} - b_{up}) \leq \varepsilon_b$ then endFibs=true
21.	else if $s < 0$ then $b_{up} = b$
22.	else $b_{low} = b$
23.	enddo

Tabla 2 Algoritmo *FIBS*[10]

En [10] se puede ver en detalle el análisis realizado al respecto.

La arquitectura propuesta puede ser vista como un bloque general, *SVMblock*, para el cual sus funcionalidades pueden ser divididas en tres fases básicas:

- **Fase de carga**: Cuando se recibe la señal `start`, (transición de 0 a 1) comienza la carga del vector de clasificación y y la matriz de kernel Q.

- **Fase de aprendizaje**: Tan pronto como la fase de carga se completa, se comienza la fase de aprendizaje según el algoritmo *Fibs* detallado en [10] y cuando esta fase termina envía una señal de activación para comenzar la fase de salida.
- **Fase de salida**: Luego remite los resultados del aprendizaje ($b^*, \alpha^*_1,\ldots, \alpha^*_m$) a la salida del dispositivo.

Con el fin de comprobar el desempeño de esta implementación se realizaron un conjunto de pruebas y se seleccionó la base de datos Sonar[26] y algunos datasets derivados de problemas de telecomunicaciones propios de los autores. La base de datos Sonar ha sido ampliamente estudiada y aplicada en la comprobación de diversos algoritmos de aprendizaje como patrón de prueba. Esta base de datos está compuesta por 208 muestras caracterizadas por 60 rasgos cada una usualmente dividida en 104 muestras de entrenamiento y 104 de prueba. Es conocido que un clasificador lineal clasifica erróneamente 23 muestras de entrenamiento, mientras que un RBF-SVM se equivoca en clasificar 6 muestras si se usa el umbral y 8 patrones en caso contrario. Se realizó un experimento empleando un kernel gaussiano con $\sigma^2 = 1$ y $C = 10$. La solución consistió de 119 vectores de soporte (59 pertenecientes a la clase positiva y 60 pertenecientes a la clase negativa). Posteriormente fueron aplicadas perturbaciones aleatorias de tamaño Δ a los parámetros y luego los errores de clasificación mínimos y máximos fueron medidos para 10^6 pruebas diferentes. A medida que la perturbación fue mayor los errores de clasificación fueron aumentando pero siempre obteniendo resultados semejantes a los reportados en la literatura para la clasificación de esta base de datos. El algoritmo fue implementado sobre un FPGA VIRTEX II de XILINX, específicamente la tarjeta XC2V8000.

A modo de resumen con esta implementación se logra desarrollar un algoritmo para realizar el entrenamiento de SVM que logra minimizar los efectos de cuantización, obtiene resultados semejantes a los algoritmos reportados en la literatura y se mejora el tiempo de convergencia respecto a los algoritmos por software. Lo interesante de esta implementación es que logra poner en un solo chip todo el hardware necesario para el entrenamiento SVM de manera que solo le sean entregados al dispositivo los parámetros de entrada y éste devuelve los resultados: todo el procesamiento se realiza en el FPGA.

3.5 Clasificación con SVM en Sistemas Numéricos Logarítmicos

En otra arista del mismo problema, Khan et al en [11] proponen la implementación de clasificador SVM digital usando los sistemas numéricos logarítmicos (LNS por sus siglas en inglés). Los sistemas numéricos logarítmicos son una alternativa para la aritmética de puntos fijo y flotante y emplean la propiedad de la compresión logarítmica para las operaciones numéricas. Dentro del dominio logarítmico, las multiplicaciones y divisiones pueden ser tratadas como simples adiciones o restas[11]. Los cálculos de estas operaciones en hardware son significativamente rápidos y además de complejidad muy reducida. El LNS es válido para números positivos y negativos y el signo del número es almacenado en un bit de signo. Emplear LNS acarrea un retraso en la conversión desde y hacia el dominio logarítmico lo cual es irrelevante frente a la reducción de la complejidad computacional de los cálculos realizados.

Entre las desventajas del uso de LNS está el hecho de que se requiere más hardware para la suma y la resta que para la multiplicación y la división además del retraso en la conversión antes mencionado[11].

Partiendo de este punto los autores proponen una arquitectura digital para la clasificación sobre FPGA, la que se detalla a continuación. En este trabajo los autores demuestran que el kernel lineal es la mejor opción para la clasificación con el conjunto de datos representativos que emplean y es debido a ello que escogen este kernel para su implementación, además, con el uso del kernel lineal durante el entrenamiento es posible almacenar el vector que representa a los pesos de cada muestra

de entrada. Como consecuencia, con el kernel lineal solo se requiere una multiplicación “punto” entre la muestra desconocida y el vector de pesos, no es necesario almacenar todos los vectores de soporte ni sus valores α, lo cual hace del kernel lineal la opción más fácil de implementar en una arquitectura de hardware. En la figura 11 se muestra la arquitectura propuesta en [11]:

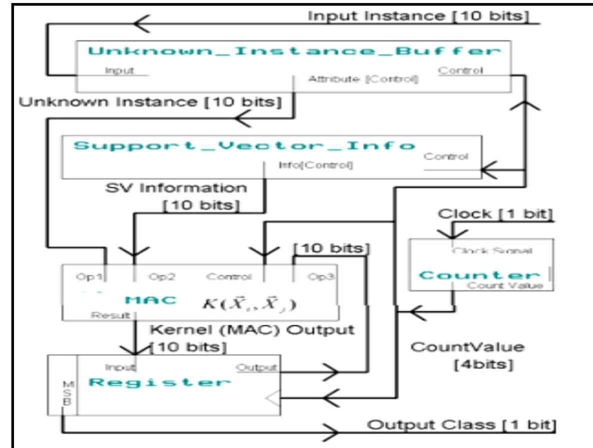


Fig. 11 Arquitectura de un clasificador SVM con kernel lineal para LNS

En general, la arquitectura funciona de mantera tal que cuando una muestra desconocida se introduce al clasificador, el registro *Register* es inicializado en *b* y el contador *Counter* es inicializado en cero. La muestra a ser clasificada es almacenada en un buffer de muestras desconocidas (*Unknown Instance Buffer*), y la memoria de información de vectores de soporte (*Support Vector Info*) almacena los pesos. La unidad MAC (*Multiply and Accumulate*) realiza tantas operaciones matemáticas como atributos tenga la muestra de entrada. Finalmente, el resultado de la clasificación es el signo del valor almacenado en *Register*. Esta arquitectura tiene como ventajas que puede ser fácilmente adaptada para implementar cualquier otro kernel sólo mediante el reemplazo de la unidad MAC con alguna otra unidad matemática que realice las operaciones del kernel en LNS.

Luego de realizar varios experimentos, y tomando como referencia a [27] se concluyeron ciertos resultados los que fueron resumidos en la tabla 3. Las diferencias están dadas por el hecho de que el cero no se puede representar en LNS, por lo que resultados extremadamente cercanos o iguales a cero no se pueden representar. Sin embargo, los autores enfatizan que estos casos son extremadamente raros.

Dataset	LNS Hardware	LNS Software Simulation	Floating Point Software
Diabetes	75.0	78.5	79.2
Votes	93.8	94.3	94.4
Heart Risk	79.2	81.2	81.2
SONAR	83.6	78.2	74.0

Tabla 3 Exactitud de la clasificación para un kernel lineal con una precisión de 4 bits

En la tabla anterior se puede apreciar que los resultados obtenidos entre la implementación usando LNS y hardware, LNS en software y punto flotante por software son muy similares. Lo interesante de esta arquitectura frente a las propuestas por software usando LNS y Punto Fijo radica en la sencillez de la implementación y en las facilidades de extensión a otros tipos de kernels.

A pesar de que con esta arquitectura se describe la clasificación de muestras desconocidas a partir de un modelo de entrenamiento previamente obtenido lo que resulta interesante es la idea de emplear el LNS en lugar de una representación en Punto Fijo o Punto Flotante.

4 Conclusiones y trabajo futuro

En este reporte se abordó el problema del entrenamiento de grandes volúmenes de datos, el cual no puede ser realizado en un tiempo viable y con un consumo adecuado de recursos computacionales con los algoritmos existentes actualmente para software. Se mostraron las SVM, el marco teórico que las sustentan y tres algoritmos para su entrenamiento los cuales fueron descritos explicando sus características, fortalezas y limitaciones. De igual manera se describen las bases matemáticas del funcionamiento de dichos algoritmos.

Con base en los resultados derivados de este reporte se propone como posible línea de investigación y desarrollo el análisis de una arquitectura digital que implemente el clasificador SVM sobre FPGA mediante uno de los algoritmos presentados, específicamente SMO (o alguna variante de este algoritmo). SMO ha demostrado ser el algoritmo más eficiente para el entrenamiento de una SVM dado a que es el que mejor uso hace de los recursos computacionales y que menor tiempo de entrenamiento emplea. Combinar esta cualidad junto a las facilidades que ofrecen los FPGAs en cuanto al paralelismo permitiría acelerar el proceso de entrenamiento. Para ello sería necesario realizar estudio más detallado del algoritmo SMO identificando las secciones del algoritmo más convenientes de paralelizar y plantear una arquitectura que permita materializar dichas transformaciones.

Referencias bibliográficas

1. Vapnik, V., *Statistical Learning Theory.* 1998: Wiley-Interscience.
2. Vapnik, V.N., *The Nature of Statistical Learning Theory.* 1999.
3. Wang, G., *A Survey on Training Algorithms for Support Vector Machine Classifiers.* Networked Computing and Advanced Information Management, International Conference on, 2008. **1**: p. 123-128.
4. Vapnik, V., *Estimation of Dependences Based on Empirical Data: Empirical Inference Science (Information Science and Statistics).* 2006: Springer.
5. Platt, J., *Sequential minimal optimization: A fast algorithm for training support vector machines.* 1998.
6. Joachims, T., *Making large-scale support vector machine learning practical.* Advances in kernel methods: support vector learning, 1999: p. 169-184.
7. Osuna, E., R. Freund, and F. Girosi. *An improved training algorithm for support vector machines.* in *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop.* 1997.
8. Platt, J., *Fast training of support vector machines using sequential minimal optimization.* 1999: p. 185-208.
9. Mirchandani, C., *Cost & Performance Measures for Optimizing Configurable Computer Architectures.* 2000.
10. Anguita, D., A. Boni, and S. Ridella, *A digital architecture for support vector machines: theory, algorithm, and FPGA implementation.* Neural Networks, IEEE Transactions on, 2003. **14**(5): p. 993-1009.
11. Khan, F.M., M.G. Arnold, and W.M. Pottenger. *Hardware-based support vector machine classification in logarithmic number systems.* in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on.* 2005.

12. Byun, H. and S.-W. Lee, *Applications for Support Vector Machines for Pattern recognition: A Survey.* Lecture Notes in Computer Science, 2002. **2388/2002**(Pattern Recognition with Support Vector Machines): p. 571-591.
13. Gunn, S., *Support vector machines for classification and regression.* 1998.
14. Eiker, J., M. Kupferschmid, and Jhon, *Introduction to Operations Research.* Introduction to Operations Research. 1988, New York: John Wiley & Sons.
15. Weisstein, E.W., *"Riemann-Lebesgue Lemma."*
16. Etin, S., U. Elias, and S. Mannor, *Parallelizing SMO for solving SVMs.*
17. Ahmad, et al., *The Comparative Study of SVM Tools for Data Classification.* p. 1-9.
18. Muller, K.R., et al., *An introduction to kernel-based learning algorithm.* IEEE Transactions., 2001. **12**(Neural Networks): p. 181-202.
19. Osuna, E., R. Freund, and F. Girosit. *Training support vector machines: an application to face detection.* in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on.* 1997.
20. Collobert, R. and S. Bengio, *SVMTorch: support vector machines for large-scale regression problems.* J. Mach. Learn. Res., 2001. **1**: p. 143-160.
21. Keerthi, S., et al., *Improvements to Platt's SMO Algorithm for SVM Classifier Design.* Neural Computation, 2001. **13**(3): p. 637-649.
22. Mangasarian, O. *Mathematical Programming for Support Vector Machines.* 2001 [cited; Available from: citeulike-article-id:4431930.
23. Platt, J. *Using Analytic QP and Sparseness to Speed Training of Support Vector Machines.* in *In Neural Information Processing Systems 11.* 1999.
24. Estrin, G., et al., *Parallel Processing in a Restructurable Computer System.* Electronic Computers, IEEE Transactions on, 1963. **EC-12**(6): p. 747-755.
25. Anguita, D., A. Boni, and S. Ridella, *Digital kernel perceptron.* Electronics Letters, 2002. **38**(10): p. 445-446.
26. Asuncion, A. and D.J. Newman, *{UCI} Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences, 2007.
27. Khan, F.M., M.G. Arnold, and W.M. Pottenger. *Finite precision analysis of support vector machine classification in logarithmic number systems.* in *Digital System Design, 2004. DSD 2004. Euromicro Symposium on.* 2004.

RT_009, Noviembre 2009
Aprobado por el Consejo Científico CENATAV
Derechos Reservados © CENATAV 2009
Editor: Lic. Lucía González Bayona
Diseño de Portada: DCG Matilde Galindo Sánchez
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
Ciudad de La Habana. Cuba. C.P. 12200
Impreso en Cuba