CIMAT

**Centro de Investigación en Matemáticas, A.C.**

# Entrenamiento de Máquinas de Soporte Vectorial y su Aplicación al Reconocimiento de Patrones

## TESIS

que para obtener el grado de

Maestro en Ciencias con Especialidad en Ciencias de la Computación y Matemáticas Industriales

presenta

*Ricardo del Angel Pérez Flores*

Director de Tesis:

*Dr. Arturo Hernández Aguirre*

Guanajuato, Gto., Julio de 2002.

**A Dios,**
**por haberme dado la**
**paciencia, fe, esperanza,**
**determinación e inteligencia**
**para poder terminar mis**
**estudios de maestría.**

**A mis padres,**
**Ricardo y Dora Emilia,**
**a quienes les agradezco su**
**confianza, apoyo y fe**
**en que podía alcanzar esta meta.**

**A mis hermanos,**
**Benjamín de Jesús**
**y Manuel David,**
**que me motivaron**
**para terminar con bien.**

## Agradecimientos.

Al Dr. Arturo Hernández Aguirre, por el apoyo, paciencia y confianza que me brindó durante el desarrollo del presente trabajo de tesis. Así mismo, por su excelente guía y dirección en la búsqueda del conocimiento necesario para trabajar.

A los Dres. Salvador Botello Rionda y Miguel Angel Moreles Vázquez, por aceptar ser revisores y sinodales de esta tesis, y por sus atinadas observaciones para enriquecerla.

Al Mtro. Andrés Edgardo Salazar Dzib y al Ing. Eduardo del Carmen Reyes Sánchez, por el apoyo y confianza depositados en mi para emprender esta aventura.

A todos los profesores del Centro de Investigación en Matemáticas, por todas sus aportaciones al crecimiento, tanto académico como personal, del autor.

A mis familiares y amigos que me apoyaron en todo momento, y a toda la gente que creyó en mí.

A todo el personal del Centro: administrativo, de seguridad, de mantenimiento, de limpieza, y a la gente de la Facultad de Matemáticas de la Universidad de Guanajuato, por hacer más confortable mi estancia en esta gran institución.

*CONTENIDO.*


**Introducción.** **Pág. 3**

**Capítulo I. Máquinas de Soporte Vectorial.** **Pág. 5**

1.1 Reconocimiento de Patrones y Areas Relacionadas. Pág. 5
1.1.1 Enfoques del Reconocimiento de Patrones. Pág. 6
1.2 El Modelo del Aprendizaje. Pág. 6
1.2.1 El Problema de la Minimización del Riesgo. Pág. 7
1.2.2 El Problema del Aprendizaje. Pág. 7
1.3 La Teoría Estadística del Aprendizaje. Pág. 8
1.3.1 La Dimensión VC. Pág. 9
1.3.2 Minimización del Riesgo Empírico y del Riesgo Estructural. Pág. 11
1.3.3 Algunas Aplicaciones Prácticas de las SVM. Pág. 11
1.4 Formulación del Problema de Clasificación con SVM. Pág. 11
1.4.1 El Caso Linealmente Separable. Pág. 12
1.4.2 El Caso No Lineal. Pág. 17
1.4.3 Selección del Kernel Apropiado. Pág. 20
1.5 Implementación de la SRM en las SVM. Pág. 21

**Capítulo II. Conceptos de Optimización.** **Pág. 24**

2.1 Fundamentos. Pág. 24
2.2 Optimización No Lineal. Pág. 27
2.2.1 Teorema de Lagrange. Pág. 28
2.2.2 Condiciones de Karush-Kuhn-Tucker. Pág. 30
2.3 Algoritmos Genéticos. Pág. 32
2.3.1 Estructura Básica de un Algoritmo Genético. Pág. 33
2.3.2 Operadores Genéticos. Pág. 35
2.3.2.1 Selección. Pág. 35
2.3.2.2 Cruza. Pág. 36
3.3.2.3 Mutación. Pág. 37
2.4 Manejo de Restricciones con Algoritmos Genéticos. Pág. 38
2.4.1 Funciones de Penalización. Pág. 38

**Capítulo III. Entrenamiento de SVM con Algoritmos Genéticos.** **Pág. 44**

3.1 Manejo del Problema y sus Restricciones. Pág. 44
3.2 Implementación con Algorimos Genéticos. Pág. 45
3.3 Resultados Experimentales. Pág. 46

**Capítulo IV. Entrenamiento de SVM con el Método de Chunking Suave.** **Pág. 51**

4.1 Solución del Problema de Optimización en SVM. Pág. 51
4.1.1 Métodos de Solución Utilizados. Pág. 51
4.1.2 Paquetes de Optimización Disponibles. Pág. 52
4.2 Métodos de Conjuntos de Trabajo. Pág. 53
4.2.1 El Método de Chunking. Pág. 55
4.2.2 El Método de Descomposición. Pág. 56
4.2.3 El Método de Optimización Secuencial Mínima. Pág. 57
4.2.4 Posibles Mejoras a los Métodos. Pág. 57
4.3 El Método de Chunking Suave. Pág. 57
4.3.1 Condiciones KKT para las SVM. Pág. 57
4.3.2 Planteamiento Formal del Método. Pág. 59
4.3.3 El Algoritmo de Chunking Suave y su Implementación. Pág. 62
4.3.4 Visualización Gráfica de su Funcionamiento. Pág. 64
4.3.5 Resultados Experimentales. Pág. 66

**Capítulo V. Conclusiones.** **Pág. 72**

5.1 Análisis Final sobre las Aportaciones del Trabajo. Pág. 72
5.2 Expectativas sobre Trabajo Futuro. Pág. 72

**Apéndices.** **Pág. 74**

A. Base de Datos de Dígitos Manuscritos. Pág. 74

**Bibliografía.** **Pág. 75**

**Capítulo IV. Entrenamiento de SVM con el Método de Chunking Suave.** **Pág. 51**

4.1 Solución del Problema de Optimización en SVM. Pág. 51
4.1.1 Métodos de Solución Utilizados. Pág. 51
4.1.2 Paquetes de Optimización Disponibles. Pág. 52
4.2 Métodos de Conjuntos de Trabajo. Pág. 53
4.2.1 El Método de Chunking. Pág. 55
4.2.2 El Método de Descomposición. Pág. 56
4.2.3 El Método de Optimización Secuencial Mínima. Pág. 57
4.2.4 Posibles Mejoras a los Métodos. Pág. 57
4.3 El Método de Chunking Suave. Pág. 57
4.3.1 Condiciones KKT para las SVM. Pág. 57
4.3.2 Planteamiento Formal del Método. Pág. 59
4.3.3 El Algoritmo de Chunking Suave y su Implementación. Pág. 62
4.3.4 Visualización Gráfica de su Funcionamiento. Pág. 64
4.3.5 Resultados Experimentales. Pág. 66

**Capítulo V. Conclusiones.** **Pág. 72**

5.1 Análisis Final sobre las Aportaciones del Trabajo. Pág. 72
5.2 Expectativas sobre Trabajo Futuro. Pág. 72

**Apéndices.** **Pág. 74**

A. Base de Datos de Dígitos Manuscritos. Pág. 74
B. Listado del Código Fuente del Entrenamiento de las SVM con Chunking y Chunking Suave. CD Anexo
C. Listado de Código Fuente del Entrenamiento de las SVM con Algoritmos Genéticos. CD Anexo
D. Listado de Código Fuente del Entrenamiento de Redes Neuronales. CD Anexo

**Bibliografía.** **Pág. 75**

# INTRODUCCION

En este siglo XXI, la ciencia avanza cada vez mas rápido. La tecnología se desarrolla en forma exponencial, de modo que cada nuevo día surgen nuevas herramientas para el mejor aprovechamiento del tiempo y de los recursos del hombre. Las Ciencias de la Computación no podían ser la excepción, sobre todo por su estrecha relación con la gran mayoría, si no es que con todas, las demás disciplinas existentes en este mundo moderno.

En el caso particular de la Inteligencia Artificial, el desarrollo mencionado se enfoca a la obtención de técnicas que permitan a las computadoras una mayor capacidad de "razonamiento" y toma de decisiones en múltiples aplicaciones. El Reconocimiento de Patrones es un área alrededor de la cual gira una gran cantidad de investigación, como sucede con las Maquinas de Soporte Vectorial (SVM) que, a pesar de su reciente surgimiento, han despertado el interés de la comunidad científica mundial.

Entre las mejoras que se buscan esta la manera de construirlas con mayor rapidez, permitiendo el ahorro de tiempo y recursos. El presente trabajo tiene como uno de sus principales objetivos proponer un método para acelerar el entrenamiento de las SVM. El método propuesto se basa en los métodos de conjunto de trabajo y podemos clasificarlo como uno de ellos.

Por otra parte, uno de los detalles con los cuales nos encontramos en la literatura acerca de SVM es la poca información detallada acerca de la teoría de esta técnica. En este trabajo se ofrece una explicación con la mayor precisión posible sobre el desarrollo matemático en el que se basa dicha teoría, con excepción de la Teoría de Kerneles en Espacios de Hilbert y la Teoría Estadística del Aprendizaje, las cuales tienen aspectos fuera del ámbito de este trabajo.

Otro tema abordado es el de la solución del problema de Programación Cuadrática propio de las SVM mediante penalizaciones a las restricciones con Algoritmos Genéticos. Se realizó un pequeño análisis de las ventajas y desventajas que pudiera tener el uso de este recurso para solucionar el problema dado.

Para terminar, se comparan las bondades de las SVM y las Redes Neuronales con un problema de reconocimiento de dígitos manuscritos.

La organización de la tesis es como sigue:

En el Capítulo I se da una breve explicación sobre Reconocimiento de Patrones y los problemas planteados por la Teoría Estadística del Aprendizaje, origen de las SVM. Posteriormente se aborda con detalle matemático el fundamento teórico de las SVM y se plantea el problema a resolver para ellas.

En el Capítulo II se da una breve reseña de las herramientas y conceptos de la Teoría de la Optimización no lineal, necesarias para comprender la teoría de las SVM. Además, se presenta un resumen de los conceptos mas importantes sobre Algoritmos Genéticos y las técnicas basadas en éstos para manejar problemas de optimización con restricciones.

En el Capítulo III se da una explicación de los Algoritmos Genéticos aplicados al problema de optimización inherente a la solución de SVM, y se muestran ejemplos sencillos de solución con esta técnica.

En el Capítulo IV se introduce al lector al concepto de conjuntos de trabajo, como preámbulo para explicar los fundamentos del método propuesto: Chunking Suave. Dicho método se explica, incluyendo ejemplos y resultados, en el mismo capítulo.

En el Capítulo V se plantean las conclusiones obtenidas de la realización del presente trabajo, junto con una visión hacia el futuro de la investigación sobre la línea creada por la tesis.

# CAPITULO I. MAQUINAS DE SOPORTE VECTORIAL.

Este capítulo introduce algunos de los conceptos del Reconocimiento de Patrones y Aprendizaje Estadístico. Además, pretende describir con claridad la parte formal de las Maquinas de Soporte Vectorial en sus posibles casos, planteando el problema a resolver cuando se desea aplicar esta técnica. La explicación sobre SVM comienza con el planteamiento del problema general, seguido del desarrollo matemático del caso linealmente separable. Aquí se plantea la situación de la búsqueda de un hiperplano que separe los puntos de las dos clases, cuyo margen entre las muestras sea máximo. Se desarrolla el problema mediante el Método de Lagrange hasta llegar a un problema dual que se plantea de manera más sencilla, para después explicar la manera en que se calculan los pesos y el sesgo del hiperplano. Posteriormente se aborda el caso no lineal, destacando las similitudes en la forma de obtener el problema dual y los cálculos de los resultados.

## 1.1 Reconocimiento de Patrones y Areas Relacionadas.

Un patrón puede ser un conjunto de medidas u observaciones, tal vez representados en forma de vector o matriz. Estas medidas pueden ser entidades como presión sanguínea, edad, número de ruedas de un auto, líneas dibujadas en 2-D, el estado de un sistema o el arreglo de un conjunto de objetos. Los patrones pueden ser convertidos de una representación en otra.

Reconocimiento de Patrones (también llamado Clasificación de Patrones) es una área de la Inteligencia Artificial que estudia la operación y el diseño de sistemas que clasifican medidas o patrones de datos en dos o más categorías. El Reconocimiento de Patrones se basa en los mecanismos naturales que utilizamos a diario gracias a nuestros sentidos. La automatización de estos mecanismos permiten a una computadora realizar labores repetitivas en las que usamos nuestros sentidos, cambiando el tacto, la vista o el oído por sensores que ayudan a escoger o categorizar objetos en una forma práctica.

Las técnicas de Reconocimiento de Patrones son, a menudo, componentes importantes de los Sistemas Inteligentes y se utilizan tanto para preprocesamiento de datos como para la toma de decisiones. En un sistema de Reconocimiento de Patrones se proporciona información a la computadora sobre las clases de cada elemento de un conjunto de datos (conjunto de entrenamiento). Una vez "entrenado" el sistema, debe ser capaz de clasificar correctamente patrones distintos a los anteriores (conjunto de prueba).

Entre las aplicaciones del Reconocimiento de Patrones están: reconocimiento de voz, de huellas digitales, de caracteres (letras, números, etc.), de caras humanas, identificación de sucesiones en cadenas de ADN, procesamiento de imágenes, análisis sísmico, clasificación y análisis de señales de radar, diagnóstico médico e inspección industrial.

Algunas áreas relacionadas con el Reconocimiento de Patrones son: Procesamiento Digital de Señales, Visión Computacional, Sistemas Expertos, Optimización, Estimación, Teoría de Autómatas y Análisis de Datos.

### 1.1.1 Enfoques del Reconocimiento de Patrones.

De acuerdo a Schalkoff [Sch92], pueden distinguirse tres enfoques para el Reconocimiento de Patrones:

- **Estadístico.** Consiste en extraer un conjunto de características (llamadas rasgos) de los datos de entrada y utilizarlos para asignar las clases. Los rasgos son generados por un estado natural y el modelo obtenido consiste en probabilidades o funciones de densidad de probabilidad de los mismos estados naturales.
- **Sintáctico.** Consiste en relacionar la estructura de los patrones con la sintaxis de un lenguaje definido formalmente, de modo que destaque el cuerpo extenso de conocimiento relacionado a la generación y análisis de patrones.
- **Neuronal.** Consiste en intentar tomar la manera en que los sistemas neuronales biológicos almacenan y manipulan la información. Esto nos lleva a una clase de sistemas llamados Redes Neuronales Artificiales.

Schalkoff comenta que los límites entre estos tres enfoques son difusos, pues comparten aspectos y objetivos comunes.

## 1.2 El Modelo del Aprendizaje.

De acuerdo con Vapnik [Vap95], el Modelo General de Aprendizaje a partir de muestras se describe a través de 3 componentes:

**i)** Un Generador $G$ de vectores aleatorios $x \in R^n$, elegidos independientemente de una función de distribución de probabilidad $F(x)$ fija pero desconocida.

**ii)** Un Supervisor $S$ quien regresa un valor de salida y para cada vector de entrada $x$, de acuerdo a una función de distribución condicional $F(y \mid x)$, también fija.

**iii)** Una Máquina de Aprendizaje $MA$ capaz de implementar un conjunto de funciones $f(x, w)$, $w \in \Lambda$, donde $\Lambda$ es un conjunto de parámetros.

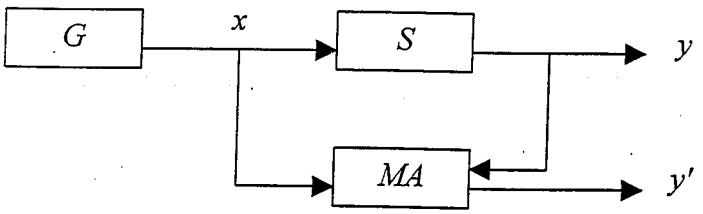


*Figura 1.1. Diagrama de un modelo general de aprendizaje a partir de muestras.*

Vapnik establece que durante el proceso de aprendizaje, la Máquina de Aprendizaje observa las parejas $(x, y)$ (conjunto de entrenamiento). Después del entrenamiento, la máquina debe regresar un valor $y'$, dado un valor $x$, que esté cerca de la respuesta del supervisor $y$.

La selección de la función deseada se basa en un conjunto de entrenamiento de $l$ observaciones independientes e idénticamente distribuidas, elegidas de acuerdo a

$$F(x,y) = F(x)F(y \mid x) : (x_1, y_1), (x_2, y_2), ..., (x_l, y_l) \tag{1.1}$$

en las siguientes subsecciones se hablará de la minimización del riesgo y su relación con el problema de aprendizaje, tal como lo describe Vapnik en [Vap95].

### 1.2.1 El Problema de la Minimización del Riesgo.

Para escoger la mejor aproximación disponible a la respuesta del supervisor, se mide la pérdida o discrepancia, $L(y, f(x,w))$ entre la respuesta $y$ del supervisor a una entrada dada $x$ y la respuesta $f(x,w)$ proveída por la Máquina de Aprendizaje ante la misma entrada. Considérese el valor esperado de la pérdida, dado por el funcional del riesgo

$$R(w) = \int L(y, f(x,w)) dF(x,y) \tag{1.2}$$

El objetivo es encontrar la función $f(x,w^*)$ que minimice el funcional del riesgo $R(w)$ (sobre la clase de funciones $f(x,w)$, $w \in \Lambda$) en la situación donde la función de distribución de probabilidad conjunta $F(x,y)$ es desconocida y la única información disponible está contenida en el conjunto de entrenamiento.

### 1.2.2 El Problema del Aprendizaje.

El planteamiento general del Problema del Aprendizaje se hace de la siguiente manera:
Sea la medida de la probabilidad $F(z)$ definida en el espacio $Z$. Considérese el conjunto de funciones $Q(z,w)$, $w \in \Lambda$. El objetivo es minimizar el funcional del riesgo

$$R(w) = \int Q(z,w) dF(z) \tag{1.3}$$

donde $w \in \Lambda$ y la medida de la probabilidad $F(z)$ es desconocida, pero se tiene una muestra independiente e idénticamente distribuida $z_1, z_2, ..., z_l$.
Los diferentes problemas de aprendizaje son casos particulares de este problema general de minimizar el riesgo funcional sobre una base de datos empíricos, donde $z$ describe un par $(x,y)$ y $Q(z,w)$ es la función de pérdida específica de cada problema. Los principales son:

**Clasificación de Patrones.** En este caso la salida del supervisor $y$ está formada por sólo dos valores $y \in \{0,1\}$ y $f(x,w)$, $w \in \Lambda$, es un conjunto de funciones indicadoras $i_F(f(x,w))$. La función de pérdida es:

$$L(y, f(x,w)) = \begin{cases} 0 & si \quad y = f(x,w) \\ 1 & si \quad y \neq f(x,w) \end{cases} \tag{1.4}$$

Para esta función de pérdida, el funcional del riesgo determina la probabilidad de respuestas diferentes dadas por el supervisor y por la función indicadora $f(x,w)$. Se conoce como error de clasificación al caso de respuestas diferentes. El problema consiste en encontrar una función que minimice la probabilidad del error de clasificación cuando la medida de probabilidad $F(x,y)$ es desconocida pero los datos son dados.

**Estimación de Regresión.** En este caso la respuesta del supervisor $y$ es un valor real y $f(x,w)$, $w \in \Lambda$, es un conjunto de funciones reales que contienen la función de regresión

$$f(x,w) = \int y dF(y \mid x) \tag{1.5}$$

Es conocido que la función de regresión es la única que minimiza el funcional del riesgo con la función de pérdida

$$L(y, f(x,w)) = \left(y - f(x,w)\right)^2 \tag{1.6}$$

El problema consiste en encontrar una función que minimice el funcional del riesgo con dicha función de pérdida cuando la medida de la probabilidad $F(x,y)$ es desconocida pero los datos son dados.

**Estimación de Densidades.** En este caso el problema es estimar una densidad a partir de un conjunto de densidades $p(x,w)$, $w \in \Lambda$. Para este problema la función de pérdida es

$$L(p(x,w)) = -log(p(x,w)) \tag{1.7}$$

Se sabe que la densidad deseada minimiza el funcional del riesgo con la función de pérdida dada. El problema consiste en encontrar una función de densidad que minimice el funcional del riesgo bajo la condición de que la correspondiente medida de la probabilidad $F(x)$ es desconocida pero los datos son dados.

## 1.3 La Teoría Estadística del Aprendizaje.

Las Maquinas de Soporte Vectorial (Support Vector Machines o SVM) constituyen una técnica recientemente desarrollada. En 1995, Vapnik [Vap95] y su equipo de colaboradores establecieron las bases teóricas a partir de la Teoría Estadística del Aprendizaje (Statistical Learning Theory o SLT), con el objetivo de resolver problemas de Reconocimiento de Patrones, aunque últimamente se comenzó a trabajar con SVM para resolver problemas de estimación en Regresión.

Las SVM efectúan una clasificación entre objetos puntuales de dos clases por medio de una superficie de decisión determinada por ciertos puntos del conjunto de entrenamiento, conocidos como vectores de soporte. Dicha superficie se obtiene al resolver un problema de Programación Cuadrática Convexa con restricciones lineales, cuyo número de variables es igual al número de datos de entrenamiento.

Las SVM, afirma Burges [Bur98], son un paradigma aparte de las Redes Neuronales y, aunque tienen muchas similitudes, están mejor fundamentadas en la teoría. Tienen mucho mayor capacidad de generalización; es decir, capacidad para clasificar correctamente datos distintos al conjunto de entrenamiento. Lo anterior, junto con su formulación, las hace muy atractivas, lo que ha motivado una gran cantidad de investigación alrededor del mundo, llamando la atención de las comunidades especializadas en Redes Neuronales y Reconocimiento de Patrones, entre otras.

### 1.3.1 La Dimensión VC [Kec01].

La Dimensión VC (Vapnik-Chervonenkis) es una propiedad del conjunto de funciones $F = \{f_1, f_2, ..., f_\infty\}$ que implementa una Máquina de Aprendizaje, que se utiliza en todos los resultados importantes de la SLT. Constituye una medida de la capacidad o complejidad de la máquina. A través de ella obtenemos una manera de elegir entre varias máquinas según las necesidades.

La dimensión VC de un conjunto de funciones $F = \{f(x,w)\}$, donde $w$ son parámetros adaptables de la máquina y $x$ es un dato a clasificar, se define como el número máximo de puntos de entrenamiento que pueden separarse por dicho conjunto. En el caso de la tarea de clasificación de dos clases, el conjunto de funciones se compone de funciones indicadores definidas como $i_F(x,w) \in \{0,1\}$ o $i_F(x,w) \in \{1,-1\}$, para toda $x$,$w$. En la clasificación con SVM se utiliza la segunda forma, pues facilita el desarrollo algebraico del problema. Estas funciones indicadoras definen hiperplanos orientados; es decir, aquellos en los que todos los puntos de un lado son asignados a la clase 1 y todos los puntos del otro lado a la clase 2. Como puede verse en la figura 1.2. Algebraicamente, un hiperplano orientado en $R^n$ se define como

$$i_F(x,w) = sign(u(x)) = \begin{cases} 1 & si \quad u \geq 0 \\ -1 & si \quad u < 0 \end{cases} \tag{1.8}$$

donde $u(x) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$, con $w_i \in R$ para toda $i$,

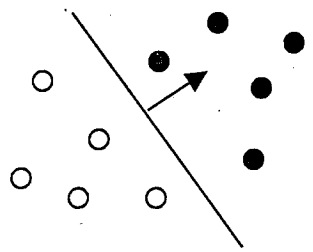

*Figura 1.2. Un hiperplano orientado en $R^2$.*

Cada hiperplano orientado determina una región de decisión lineal, a un lado de la cual hay datos de una clase y en el otro lado datos de la otra clase, de modo que si tomamos un punto cualquiera, este caerá necesariamente en una de las clases definidas por la región.

La dimensión VC de los hiperplanos orientados es igual al número mayor de puntos que pueden separarse correctamente en todas las maneras posibles por miembros del conjunto. Dado al menos un conjunto trivial de $h$ puntos (puntos no colineales) que puedan etiquetarse en las $2^h$ maneras posibles, y por cada caso de etiquetado pueda encontrarse al menos un hiperplano orientado (miembro del conjunto) que asigne correctamente las etiquetas, entonces ese conjunto de puntos es separado por ese conjunto de funciones y, si no es posible separar ni un solo conjunto de $h+1$ puntos por elementos del conjunto de funciones, entonces la dimensión VC de dicho conjunto es $h$.

En dos dimensiones, si tenemos un conjunto de rectas orientadas $u(x) = w_0 + w_1 x_1 + w_2 x_2$, y utilizamos este conjunto de funciones para separar 3 puntos en las $2^3 = 8$ maneras posibles de hacerlo, obtendríamos lo que se ve en la figura 1.3.

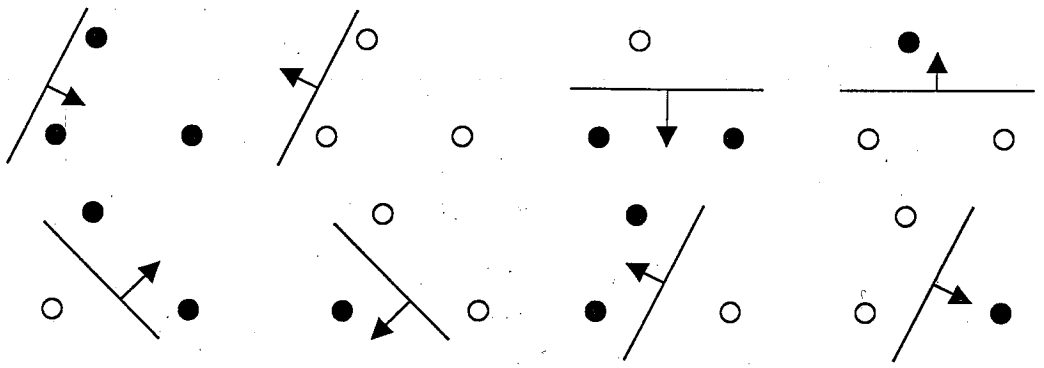

*Figura 1.3. La separación de tres puntos por rectas orientadas en $R^2$.*

Claramente, con tres puntos es posible efectuar todas las separaciones; mientras que con 4 puntos no existe manera alguna de separación para todos los $2^4 = 16$ casos, como puede verse en la figura 1.4, donde se muestran los casos críticos. Menos aún es posible hacerlo con más de 4 puntos, como resulta obvio.

Así, el número máximo de puntos que pueden separarse con el conjunto de rectas orientadas en $R^2$ es 3. Esto es, la dimensión VC de este conjunto de funciones particular es $h = 3$. En general, en un espacio n-dimensional $R^n$, la dimensión VC de conjunto de hiperplanos orientados es $h = n+1$.

*Figura 1.4. La separación de cuatro puntos por rectas orientadas en $R^2$.*

La dimensión VC no tiene relación alguna con el número de parámetros de las funciones, pues en 1995 Vapnik dio un ejemplo de una función de un solo parámetro con dimensión VC infinita (un conjunto de funciones indicadoras tiene dimensión VC infinita si puede separar un número muy grande de puntos). Dicha función es $i_F(x,w) = sign(sen(wx))$ con

$$w = \pi\left(1 + \sum_{i=1}^{l} \frac{(1-y_i)10^i}{2}\right) \tag{1.9}$$

donde $l$ es el número de vectores de entrenamiento y $y_i \in \{0,1\}$ son las clases.

### 1.3.2 Minimización del Riesgo Empírico y del Riesgo Estructural.

Mientras que las Redes Neuronales y otras técnicas de clasificación pretenden minimizar el riesgo empírico (Empirical Risk Minimization o ERM); es decir, el error del conjunto de entrenamiento; las SVM pretenden minimizar el riesgo estructural (Structural Risk Minimization o SRM). El riesgo estructural se controla por medio de dos parámetros: la capacidad de la máquina, expresada por la dimensión VC, y la reducción del error del conjunto de entrenamiento.
La diferencia entre los métodos es notable; la ERM no controla la generalización, sino que asume que ésta será correcta si el error de entrenamiento es pequeño sobre un conjunto de entrenamiento lo suficientemente grande. La SRM, en cambio, controla ambos parámetros: error de entrenamiento y generalización. La generalización depende del tamaño estructural de la máquina, o sea su dimensión VC.

### 1.3.3 Algunas Aplicaciones Prácticas de las SVM.

Entre las aplicaciones específicas de las SVM descritas en [Wri00] tenemos: diagnóstico de cáncer de seno, predicción de temperatura en cables subterráneos, identificación de partículas atómicas en física de alta energía, detección de objetos en 2-D y 3-D, detección de deterioro en motores de combustión, detección de homologías en proteínas remotas, aproximación de funciones y regresión, categorización de textos, predicción de series de tiempo y reconstrucción dinámica en sistemas caóticos, reconocimiento de dígitos escritos a mano, y arboles de decisión para bases de datos de mercadeo.

## 1.4 Formulación del Problema de Clasificación con SVM.

Tenemos $l$ muestras independientes e idénticamente distribuidas, provenientes de una distribución de probabilidad $P(x,y)$ desconocida, consistentes en pares conformados por un vector y una etiqueta de clase $(x_1,y_1),(x_2,y_2),...,(x_l,y_l)$, donde cada vector $x_i \in R^n$ podría ser un vector de tonos de gris de alguna imagen, y cada etiqueta $y_i \in \{1,-1\}$ podría ser 1 si la imagen contiene la huella digital del Señor X y -1 en caso contrario, para $i = 1,2,...,l$.

Deseamos construir una SVM que sepa mapear los valores de los vectores $x_i$ a los valores de las etiquetas $y_i$ correspondientes por medio de una función de decisión $y = f : R^n \rightarrow \{1,-1\}$, la cual está entre dos hiperplanos que definen un margen de tamaño máximo. El uso de $\{1,-1\}$ en vez de $\{1,0\}$ u otros valores nos proporciona una mejor facilidad en el manejo algebraico del problema, como se verá más adelante.

### 1.4.1 El Caso Linealmente Separable.

Es el caso más simple, donde los datos pueden ser clasificados por un hiperplano orientado, conocido como hiperplano de separación, con ecuación $w \cdot x + b = 0$, donde $w$ es un vector normal al hiperplano llamado vector de pesos y $b$ ($w_0$ en la definición de hiperplano orientado) se conoce como sesgo. La distancia del hiperplano al origen se define como

$$d = \begin{cases} \dfrac{-b}{\|w\|} & si \quad b < 0 \\ \dfrac{b}{\|w\|} & si \quad b > 0 \end{cases} = \frac{|-b|}{\|w\|} \tag{1.10}$$

Sea $M_+$ la distancia más corta desde el hiperplano de separación hasta el dato positivo más cercano, y $M_-$ la distancia más corta desde el hiperplano de separación hasta el dato negativo más cercano. Se define el margen de un hiperplano de separación como $M = M_+ + M_-$. Ahora necesitamos encontrar el hiperplano de separación con el margen mayor.

Decimos que el conjunto de entrenamiento es linealmente separable si existen $w \in R^n$ y $b \in R$, tales que todos los datos de entrenamiento satisfacen las siguientes restricciones:

$$w \cdot x_i + b \geq 1 \text{ para } y_i = 1 \tag{1.11}$$

$$w \cdot x_i + b \leq -1 \text{ para } y_i = -1 \tag{1.12}$$

Para combinar ambas restricciones, tomamos en cuenta que $w \cdot x_i + b \leq -1$ es igual a $-(w \cdot x_i + b) \geq 1$. Esto se parece a (1.11), excepto porque el lado izquierdo esta multiplicado por -1. Claramente, el lado izquierdo en ambos casos esta multiplicado por su propia clase, por lo tanto tenemos que $y_i(w \cdot x_i + b) \geq 1$, para $i = 1,2,...,l$. Esto equivale a que

$$y_i(w \cdot x_i + b) - 1 \geq 0 \text{ para } i = 1,2,...,l, \tag{1.13}$$

que define el hiperplano de separación descrito.

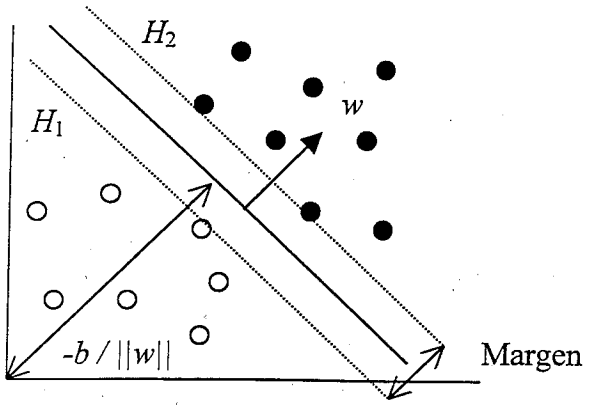


*Figura 1.5. El hiperplano de separación para el caso linealmente separable en $R^2$.*

Para los puntos que cumplen con la ecuación (1.11), algunos cumplen con la igualdad y caen en el hiperplano $H_1 : w \cdot x + b = 1$; esto es, $w \cdot x + b - 1 = 0$, con $w$ normal y distancia al origen

$$d_1 = \frac{|1-b|}{\|w\|} \tag{1.14}$$

Similarmente, para los puntos que cumplen con la ecuación (1.12), algunos cumplen con la igualdad y caen en el hiperplano $H_2 : w \cdot x + b = -1$; esto es, $w \cdot x + b + 1 = 0$, con $w$ normal y distancia al origen

$$d_2 = \frac{|-1-b|}{\|w\|} \tag{1.15}$$

La distancia del hiperplano de separación al hiperplano $H_1$ es

$$M_+ = \frac{|-b|}{\|w\|} - \frac{|1-b|}{\|w\|} = \frac{|-b|-|1-b|}{\|w\|} = \begin{cases} \dfrac{b-(-1+b)}{\|w\|} & si \quad b > 0 \\ \dfrac{-b-(-1-b)}{\|w\|} & si \quad b < 0 \end{cases} = \frac{1}{\|w\|} \tag{1.16}$$

Asimismo, la distancia del hiperplano de separación al hiperplano $H_2$ es

$$M_{-} = \frac{|-1-b|}{\|w\|} - \frac{|-b|}{\|w\|} = \frac{|-1-b|-|-b|}{\|w\|} = \begin{cases} \dfrac{1+b-(b)}{\|w\|} & si \quad b>0 \\ \dfrac{1-b-(-b)}{\|w\|} & si \quad b<0 \end{cases} = \frac{1}{\|w\|} \tag{1.17}$$

Así, al sumar las dos distancias tenemos el margen de separación para las dos clases, el cual es

$$M = \frac{2}{\|w\|} \tag{1.18}$$

Es claro que $H_1$, $H_2$ y el hiperplano de separación son paralelos (tienen el mismo vector normal), y que no hay puntos de entrenamiento entre ellos. Por lo tanto, para encontrar el par de hiperplanos que dan el máximo margen, y al mismo tiempo el hiperplano de separación, es necesario maximizar la distancia $M$, lo que equivale a minimizar el valor de $\|w\|$, sujeto a las restricciones dadas. De esta forma, llegamos al siguiente problema de Programación Cuadrática Convexa:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 = \frac{1}{2} w \cdot w \tag{1.19}$$

sujeto a las restricciones

$$-y_i\left(w \cdot x_i + b\right) + 1 \le 0 \text{ para } i = 1,2,...,l\,. \tag{1.20}$$

Se ha planteado la minimización de $\frac{1}{2}\|w\|^2$ en vez de minimizar $\|w\|$, con el fin de adaptar el problema a la forma estándar de Programación Cuadrática Convexa, como se verá más adelante, pues sólo $\|w\|$ nos llevaría a tratar con una raíz cuadrada, lo que haría muy complejo llegar a la solución.

La solución de este problema de optimización se encuentra en el Punto Silla de la función lagrangiana o Lagrangiano primal del mismo, dada por

$$L_P(w,b,\alpha) = \frac{1}{2} w \cdot w + \sum_{i=1}^{l} \alpha_i \left[-y_i\left(w \cdot x_i + b\right) + 1\right] \tag{1.21}$$

$$L_P(w,b,\alpha) = \frac{1}{2} w \cdot w - \sum_{i=1}^{l} \alpha_i y_i w \cdot x_i - b\sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i \tag{1.22}$$

donde las $\alpha_i$, para $i = 1,2,\mathrm{K}\,,l$, son los multiplicadores de Lagrange. Dicha solución es la misma que para el problema dual.

Para encontrarlo debemos minimizar el Lagrangiano con respecto a $w$ y $b$, al tiempo que es maximizado con respecto a $\alpha = (\alpha_1, \alpha_2, ..., \alpha_l)$. El problema dual está dado por $\max_{\alpha} \; L_D(\alpha) = \max_{\alpha} \; \{\min_{w,b} \; L_P(w,b,\alpha)\}$. Para encontrarlo, calculamos el gradiente de $L_P$ con respecto a $w$ y $b$, y lo igualamos a cero.

$$\frac{\partial L_P(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{l} \alpha_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{1.23}$$

$$\frac{\partial L_P(w,b,\alpha)}{\partial b} = -\sum_{i=1}^{l} \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.24}$$

Las dos expresiones anteriores se conocen como condiciones de optimalidad. Sustituyéndolas en $L_P$ obtenemos el Lagrangiano dual

$$L_D(\alpha) = \frac{1}{2} \sum_{i=1}^{l} \alpha_i y_i x_i \cdot \left( \sum_{j=1}^{l} \alpha_j y_j x_j \right) - \sum_{i=1}^{l} \alpha_i y_i \left( \sum_{j=1}^{l} \alpha_j y_j x_j \right) \cdot x_i - b(0) + \sum_{i=1}^{l} \alpha_i \tag{1.25}$$

$$L_D(\alpha) = \frac{1}{2} \sum_{i=1}^{l} \alpha_i y_i x_i \cdot \left( \sum_{j=1}^{l} \alpha_j y_j x_j \right) - \sum_{i=1}^{l} \alpha_i y_i x_i \cdot \left( \sum_{j=1}^{l} \alpha_j y_j x_j \right) + \sum_{i=1}^{l} \alpha_i \tag{1.26}$$

$$L_D(\alpha) = -\frac{1}{2} \sum_{i=1}^{l} \alpha_i y_i x_i \cdot \left( \sum_{i=1}^{l} \alpha_i y_i x_i \right) + \sum_{i=1}^{l} \alpha_i \tag{1.27}$$

$$L_D(\alpha) = -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum_{i=1}^{l} \alpha_i \tag{1.28}$$

Así, podemos replantear el problema como

$$\max_{\alpha} \quad L_D(\alpha) = -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum_{i=1}^{l} \alpha_i \tag{1.29}$$

sujeto a las restricciones

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{1.30}$$

$$\alpha_i \geq 0 \text{ para } i = 1, 2, ..., l. \tag{1.31}$$

Pero este problema equivale a:

$$\min_{\alpha} \quad -L_D(\alpha) = \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i \cdot x_j - \sum_{i=1}^{l} \alpha_i \tag{1.32}$$

$$\min_{\alpha} \quad -L_D(\alpha) = \frac{1}{2} \alpha^T H \alpha + d^T \alpha \tag{1.33}$$

sujeto a las mismas restricciones; donde $H$ es una matriz de $l \times l$ tal que $H_{ij} = y_i y_j x_i \cdot x_j$, y $d$ es un vector con todos sus elementos con valor de -1.

Este problema cumple con la forma estándar del problema de Programación Cuadrática Convexa con una restricción de igualdad (1.30) y $l$ de desigualdad (1.31). Puede ser manejado con mayor facilidad, pues las restricciones originales son sustituidas por restricciones que incluyen a los multiplicadores de Lagrange, además de que los datos de entrenamiento aparecen comc simples productos punto de vectores. Dicha situación resulta ventajosa en el caso no lineal, que se revisará en la siguiente sección.

En la solución, a cada punto de entrenamiento le corresponde un multiplicador de Lagrange óptimo $\alpha_i^*$. Los puntos para los cuales su respectiva $\alpha_i^* > 0$ están sobre alguno de los hiperplanos $H_1$ o $H_2$, y se conocen como vectores de soporte. Todos los demás puntos de entrenamiento tienen $\alpha_i^* = 0$ y están a los lados de $H_1$ o $H_2$ de modo que se cumple la desigualdad de la restricción.

Para las SVM, los vectores de soporte son los elementos críticos del conjunto de entrenamiento. Ellos son los más cercanos al límite de decisión, cumplen con la igualdad en las restricciones (1.13) y si los elimin---aramos cambiaría la solución encontrada. En cambio, si todos los demás puntos de entrenamiento fueran removidos o movidos de lugar sin cruzar $H_1$ o $H_2$, al repetir el entrenamiento se encontraría el mismo hiperplano de separación.

Una vez solucionado el problema, tendremos un vector $\alpha^*$ de multiplicadores de Lagrange. Para obtener la ecuación del hiperplano de separación óptimo, utilizamos la primera condición de optimalidad (1.23) para encontrar el vector de pesos óptimo

$$w^* = \sum_{i=1}^{l} \alpha^*{}_i x_i y_i \tag{1.34}$$

Todos los vectores de entrenamiento cumplen la condición (1.13). Los vectores de soporte cumplen con la igualdad de la misma y a cada uno le podemos calcular un valor del sesgo despejando este de la forma $b_s = \frac{1 - y_s(w^* \cdot x_s)}{y_s}$; es decir, hay un $b_s = \frac{1}{y_s} - (w^* \cdot x_s)$ para cada vector de soporte. Pero como dividir 1 entre $y_s$ nos da un 1 ó -1, determinado por la clase, sustituimos ese termino por la clase correspondiente y nos queda $b_s = y_s - (w^* \cdot x_s)$ para cada vector de soporte, de la misma manera que para obtener (1.13). El sesgo óptimo se obtiene promediando los sesgos de los vectores de soporte de la siguiente manera:

$$b^* = \frac{1}{k}\left[\sum_{s=1}^{k}\left(y_s - w^* \cdot x_s\right)\right] \tag{1.35}$$

donde $k$ es el número de vectores de soporte; es decir, aquellos cuyo correspondiente multiplicador de Lagrange $\alpha_i^* > 0$. Al resto de los vectores del conjunto de entrenamiento les corresponden los valores $\alpha_i^* = 0$. En SVM, los valores de los multiplicadores de Lagrange tienen relación con la influencia del vector correspondiente sobre el hiperplano de separación. Así pues, para los vectores de soporte, entre mayor sea $\alpha_i^*$, mayor será su influencia, mientras que el resto de los vectores no tiene influencia alguna.

Una vez calculados $w^*$ y $b^*$, obtenemos la ecuación del hiperplano de decisión $u(x) = w^* \cdot x + b$. La función de decisión buscada es entonces $i_F(x, w)$ como en (1.8).

### 1.4.2 El Caso No Lineal.

En la mayoría de los casos, la separación lineal en el espacio de entrada es demasiado restrictiva para un uso práctico. La teoría puede extenderse a regiones de decisión no lineales mapeando los puntos de entrada hacia puntos en un espacio de rasgos, buscando luego el hiperplano óptimo de separación en dicho espacio.

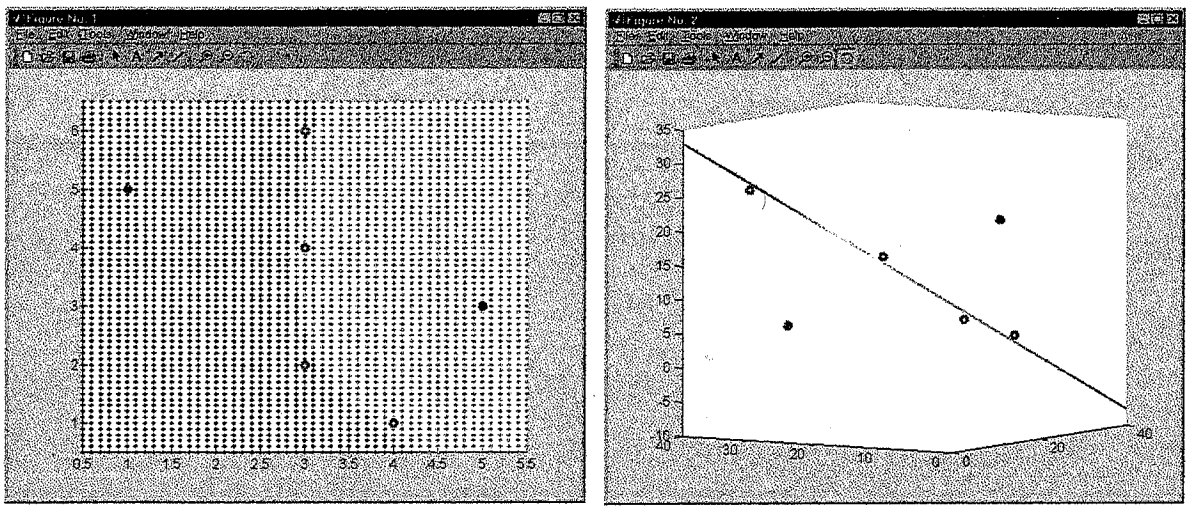

*Figura 1.6. Gráficas de un ejemplo de mapeo de $R^2$ a $R^3$ utilizando un kernel polinomial de grado 2. Pueden verse las equivalencias en las posiciones de los vectores respecto al hiperplano de separación en $R^3$ y la región de decisión no lineal en $R^2$.*

Dados los punto de entrada, utilizamos la función $\varphi : R^n \to R^m$ para mapear dichos puntos hacia el espacio de rasgos $R^m$ (que por lo general es un Espacio de Hilbert de dimensión finita o infinita). A un hiperplano óptimo de separación (una región de decisión lineal) en $R^m$ le corresponde una región de decisión no lineal en el espacio de entrada, que solo puede determinarse si la función $\varphi$ es completamente conocida. La ventaja es que $\varphi$ interviene solamente en el producto punto de dos puntos del espacio de rasgos, mediante una función simétrica conocida como kernel, que se calcula a partir de los puntos en el espacio de entrada como $k(u, v) = \varphi(u) \cdot \varphi(v)$. Aquí no es necesario conocer $\varphi$.

En la figura 1.6 podemos observar que los datos en $R^2$ no son linealmente separables, pero al mapearlos a $R^3$, es posible encontrar un hiperplano que haga la separación lineal en forma adecuada. La región de decisión no lineal en $R^2$ (verde y negra) está determinada por el hiperplano en $R^3$. En este caso, la función de mapeo utilizada es $\varphi(u) = (u_1^2, \sqrt{2}u_1u_2, u_2^2)$, y al hacer el producto punto con $\varphi(v) = (v_1^2, \sqrt{2}v_1v_2, v_2^2)$, se obtiene el kernel

$$k(u, v) = (u_1v_1)^2 + 2u_1u_2v_1v_2 + (u_2v_2)^2 = (u_1v_1 + u_2v_2)^2 = (u \cdot v)^2 \qquad (1.54)$$

Cabe señalar que tendría un altísimo costo computacional efectuar explícitamente estos mapeos y luego calcular los productos punto. Sin embargo, cuando un kernel cumple las Condiciones de Mercer (que se presentan más adelante), el mapeo es implícito. Esto es, trabajamos en $R^n$ aunque la solución se encuentre en $R^m$.

La región de separación no lineal puede encontrarse como la solución al problema de Programación Cuadrática antes mencionado, pero con $\varphi(x_i)$ en vez de cada $x_i$. Es necesario modificar el planteamiento del problema como:

$$\min_{\varphi(w),b} \quad \frac{1}{2}\|\varphi(w)\|^2 = \frac{1}{2}\varphi(w)\cdot\varphi(w) \tag{1.55}$$

sujeto a las restricciones

$$-y_i\left(\varphi(w)\cdot\varphi(x_i)+b\right)+1\le 0 \text{ para } \alpha \ \geq \ 0 \ . \tag{1.56}$$

La solución está en el Punto Silla del Lagrangiano

$$L_P(\varphi(w),b,\alpha)=\frac{1}{2}\varphi(w)\cdot\varphi(w)+\sum_{i=1}^{l}\alpha_i\left[-y_i\left(\varphi(w)\cdot\varphi(x_i)+b\right)+1\right] \tag{1.57}$$

$$L_P(\varphi(w),b,\alpha)=\frac{1}{2}\varphi(w)\cdot\varphi(w)-\sum_{i=1}^{l}\alpha_i y_i\varphi(w)\cdot\varphi(x_i)-b\sum_{i=1}^{l}\alpha_i y_i+\sum_{i=1}^{l}\alpha_i \tag{1.58}$$

donde las $\alpha_i$, para $i=1,2,...,l$, son los multiplicadores de Lagrange.

Minimizando el Lagrangiano con respecto a $\varphi(w)$ y $b$, al tiempo que es maximizado con respecto a $\alpha$. El problema dual está dado por $\max_{\alpha} \quad L_D(\alpha)=\max_{\alpha} \quad \{\min_{\varphi(w),b} \quad L_P(\varphi(w),b,\alpha)\}$.

Ahora encontramos el problema dual, primero calculando el gradiente de $L_P$ con respecto a $\varphi(w)$ y $b$, e igualándolos a cero (condiciones de optimalidad).

$$\frac{\partial L_P(\varphi(w),b,\alpha)}{\partial\varphi(w)}=\varphi(w)-\sum_{i=1}^{l}\alpha_i y_i\varphi(x_i)=0 \quad\Rightarrow\quad \varphi(w)=\sum_{i=1}^{l}\alpha_i y_i\varphi(x_i) \tag{1.59}$$

$$\frac{\partial L_P(\varphi(w),b,\alpha)}{\partial b}=-\sum_{i=1}^{l}\alpha_i y_i=0 \quad\Rightarrow\quad \sum_{i=1}^{l}\alpha_i y_i=0 \tag{1.60}$$

Sustituyendo las condiciones de optimalidad (3.55) y (3.56) en $L_P$ (3.54) tenemos

$$L_D(\alpha)=\frac{1}{2}\sum_{i=1}^{l}\alpha_i y_i\varphi(x_i)\cdot\left(\sum_{j=1}^{l}\alpha_j y_j\varphi(x_j)\right)-\sum_{i=1}^{l}\alpha_i y_i\left(\sum_{j=1}^{l}\alpha_j y_j\varphi(x_j)\right)\cdot\varphi(x_i)-b(0)+\sum_{i=1}^{l}\alpha_i \tag{1.61}$$

$$L_D(\alpha)=-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j\varphi(x_i)\cdot\varphi(x_j)+\sum_{i=1}^{l}\alpha_i \tag{1.62}$$

Por la definición de kernel vista, replanteamos el problema como

$$\max_{\alpha} \quad L_D(\alpha) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i, x_j) + \sum_{i=1}^{l}\alpha_i \tag{1.63}$$

sujeto a las restricciones

$$\sum_{i=1}^{l}\alpha_i y_i = 0 \tag{1.64}$$

$$\alpha_i \geq 0 \text{ para } i = 1,2,...,l\,. \tag{1.65}$$

Lo cual sabemos que equivale a

$$\min_{\alpha} \quad -L_D(\alpha) = \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i, x_j) - \sum_{i=1}^{l}\alpha_i \tag{1.66}$$

sujeto a las mismas restricciones. Este problema tiene la misma forma del definido por (1.32) y (1.33), donde $H$ es la matriz de $l \times l$ tal que $H_{ij} = y_i y_j k(x_i, x_j)$, y $d$ se define de igual manera que en los casos anteriores. La ventaja del problema dual sobre el problema primal, por depender de los multiplicadores de Lagrange y de productos punto de vectores de entrenamiento se hace evidente aquí: si el problema dependiera del vector de pesos, tendríamos tantas incógnitas como la dimensión del espacio de rasgos, que se ha visto que tiene una dimensión demasiado grande.
El sesgo óptimo se obtiene como

$$b^* = \frac{1}{k}\left[\sum_{s=1}^{k}\left(y_s - \sum_{i=1}^{l}\alpha_i y_i \cdot k(x_s, x_i)\right)\right] \tag{1.67}$$

donde $k$ es el número de vectores de soporte.
La clasificación se efectúa mediante la misma función que en el caso linealmente separable, pero en este caso no es posible calcular el vector de pesos explícitamente debido a que necesitaríamos conocer $\varphi$; por lo tanto tenemos

$$u(x) = \sum_{i=1}^{l}\alpha_i y_i k(x_i, x) + b^* \tag{1.68}$$

Así, la extensión de la teoría se traduce en encontrar kerneles que identifiquen a ciertas familias de regiones de decisión que puedan escribirse como productos punto en algún espacio de rasgos.

### 1.4.3 Selección del Kernel Apropiado.

Como se ha visto, cuando un clasificador lineal es inapropiado para separar en dos clases, las SVM se pueden construir mapeando los vectores de entrada hacia un espacio de rasgos de alta dimensión por medio de un kernel. Generalmente, los datos serán linealmente separables en algún espacio de rasgos de muy alta dimensión, por lo que puede no valer la pena intentar dicha separación de manera exacta, sobre todo si sólo tenemos disponibles una cantidad finita de datos de entrenamiento. En la práctica, será necesario emplear el enfoque no separable para el caso no lineal y escoger el kernel que combine la mejor generalización con un mapeo al espacio de rasgos de la menor dimensión posible.
La idea de la función kernel es permitir la realización de operaciones en el espacio de entrada, en vez de hacerlo en el espacio de rasgos de alta dimensión. Como el producto punto no necesita ser evaluado en el espacio de rasgos, el costo computacional es mucho menor.
Un producto punto en un espacio de rasgos tiene un kernel equivalente en el espacio de entrada $k(u,v)=\varphi(u)\cdot\varphi(v)$. Si $k$ es una función simétrica positiva definida en el espacio $L_2$, que cumple con la condición

$$\iint k(u,v)g(u)g(v)dudv>0 \tag{1.70}$$

para toda $g\neq 0$ para la cual $\int g^2(u)du<+\infty$, entonces el kernel representa un producto punto legítimo en el espacio de rasgos; es decir, tiene una expansión

$$k(u,v)=\sum_{m=1}^{+\infty}a_m\psi_m(u)\psi_m(v) \tag{1.69}$$

en $L_2$, donde $a_m\geq 0$. Lo anterior es conocido en su conjunto como Condiciones de Mercer. Los kerneles mas conocidos y utilizados, según describe Gunn [Gun98], son:

**Polinomiales de Grado d.** El mapeo polinomial es un método popular para la modelación no lineal. Dos posibles formas de hacerlo son:

$$k(u,v)=(u\cdot v)^d \tag{1.71}$$

$$k(u,v)=(u\cdot v+1)^d \tag{1.72}$$

El segundo kernel es usualmente preferible porque evita problemas de que el Hessiano llegue a cero. Si tomamos el primer tipo de kernel con $d=1$, tendremos un producto punto en el mismo espacio de entrada. Se suele llamar a esto un kernel lineal.

**Funciones de Base Radial (RBF) Gaussianas.** Las funciones de base radial han recibido atención significativa, más comúnmente con una Gaussiana de la forma

$$k(u,v)=e^{-\frac{\|u-v\|^2}{2\sigma^2}} \tag{1.73}$$

Las técnicas clásicas que utilizan funciones de base radial emplean algún método para determinar un subconjunto de centros. Un rasgo atractivo de las SVM es que esta selección es implícita, con cada vector de soporte contribuyendo a una función Gaussiana local, centrada en ese punto de dato. Para consideraciones posteriores es posible seleccionar el ancho global de la función de base $\sigma$, utilizando el principio de SRM.

**Perceptrones Multicapas.** Este tipo de kernel representa a un Perceptrón Multicapas con una sola capa oculta para ciertos valores de $k$ y $\delta$.

$$k(u,v)=\tanh(ku\cdot v-\delta) \tag{1.74}$$

Cuando se utiliza este kernel, los vectores de soporte corresponden a la primera capa de un Perceptrón Multicapas cuyos multiplicadores de Lagrange son los pesos.

Existen muchos otros tipos de kerneles, tales como RBF exponenciales, series de Fourier, splines y splines B. Además, es posible construir kerneles más complicados mediante la suma o el producto de varios kerneles [Gun98].
Los kerneles mencionados cumplen con las Condiciones de Mercer para cualesquiera vectores $x$ y $y$ de reales.

## 1.5 Implementación de la SRM en las SVM.

En la presente sección se pretende dar un panorama sobre la SRM y como las SVM implementan ese principio. Esta basado en [Kec01], [Sch97], [Vap95] y [Gun98].
La idea básica de la SRM es escoger, de un gran número de modelos candidatos (Máquinas de Aprendizaje), un modelo de la complejidad correcta para describir los pares de datos de entrenamiento. Esto puede hacerse al restringir la cantidad de conjuntos de funciones de aproximación y simultáneamente controlar su complejidad. Así, las Máquinas de Aprendizaje serán estos modelos parametrizados que, al incrementar el número de parámetros (pesos y sesgo), forman una estructura anidada en el sentido

$$S_1\subset S_2\subset\ldots\subset S_n\subset\ldots\subset S \tag{1.75}$$

donde $S_n=\{f(x,w),w\in\Lambda_n\}$, $\Lambda_n\in\Lambda$, con dimensiones VC

$$h_1\le h_2\le\ldots\le h_n\le\ldots h \tag{1.76}$$

En tal conjunto anidado de funciones, cada función siempre contiene una función previa, menos compleja. $S_n$ puede ser un conjunto de polinomios de grado $n$ en una variable, un modelo de Fuzzy Logic con $n$ reglas, perceptrones multicapa o una red RBF con $n$ capas de neuronas ocultas. La definición de conjuntos anidados (1.75) es valida para todos esos modelos porque, por ejemplo, una Red Neuronal de $n$ neuronas es un subconjunto de una Red Neuronal de $n+1$ neuronas, un modelo de Fuzzy Logic con $n$ reglas es un subconjunto de un modelo de Fuzzy Logic con $n+1$ reglas, y así por el estilo. El objetivo del aprendizaje es la selección de uno de los subconjuntos, el cual hace coincidir la complejidad de los datos de entrenamiento con una aproximación de la capacidad del modelo.

Para una Maquina de Aprendizaje, se define el riesgo como el promedio de los errores cometidos al clasificar muestras de prueba independientes, pero que provienen de la misma distribución, y esta dado por

$$R(w) = \int \frac{1}{2} |f(x,w) - y| dP(x,y) \tag{1.77}$$

Desafortunadamente $R(w)$ es desconocido, puesto que $P(x,y)$ lo es. Entonces se trata de minimizar el riesgo empírico

$$R_{emp}(w) = \frac{1}{l} \sum_{i=1}^{l} \frac{1}{2} |f(x_i,w) - y_i| \tag{1.78}$$

Esto no garantiza que el riesgo sea pequeño, pues el número de muestras $l$ es limitado. Puede darse el caso de tener un error muy pequeño pero no una buena capacidad de generalización, lo que se conoce como sobreajuste. Para poder trabajar con las cantidades limitadas de datos, Vapnik [Vap95] estableció el principio de la SRM, en el cual dado el problema anterior, para alguna $w \in \Lambda$ y $l > h$, con probabilidad $1-\eta$ se cumple la cota

$$R(w) \leq R_{emp}(w) + \sqrt{\frac{h\left(\ln\frac{2l}{h}+1\right) - \ln\frac{\eta}{4}}{l}} \tag{1.79}$$

donde el segundo término en el lado derecho es llamado Confianza VC (Vapnik-Chervonenkis). Dado un número fijo de $l$ muestras de entrenamiento, se puede controlar el riesgo al controlar las dos cantidades del lado derecho. El riesgo empírico, que depende de la función escogida $f(x,w)$ (recordar que $x$ representa a las $l$ muestras), puede controlarse al escoger una $w$ adecuada. Mientras tanto, la confianza VC se controla con $h$, que depende de que conjunto de funciones $S_n$ se escoja para implementar la maquina.

Las ecuaciones anteriores muestran que cuando el número de datos de entrenamiento se incrementa, esto es, para $l \to +\infty$ (con otros parámetros fijos), el riesgo verdadero $R(w)$ esta muy cerca del riesgo empírico $R_{emp}(w)$ porque la confianza VC tiende a 0. Por otro lado, cuando la probabilidad $1-\eta$ (nivel de confianza) se aproxima a 1, la cota de generalización crece grande, porque en el caso cuando $\eta \to 0$ $(1-\eta \to 1)$, el valor de la confianza VC tiende a $+\infty$.

Para las SVM, la forma en que se resuelve (1.19) y (1.20) es equivalente a implementar el principio de SRM. Supongamos que la siguiente cota se cumple $\|w\| \leq A$. Entonces, de la ecuación (1.13) y tomando en cuenta que la distancia de un punto $x$ a un hiperplano $w \cdot x + b = 0$ se define como $d = \dfrac{|w \cdot x + b|}{\|w\|}$, entonces tenemos que

$$d \geq \frac{1}{A} \tag{1.80}$$

De acuerdo a esto, los hiperplanos no pueden estar más cerca que $\dfrac{1}{A}$ de algunos de los puntos, lo que reduce los posibles hiperplanos de separación, y por lo tanto la capacidad.

Vapnik [Vap95] redefine la dimensión VC $h$ de un conjunto de hiperplanos en un espacio n-dimensional como

$$h \leq mín\left\lfloor R^2 A^2, n \right\rfloor + 1 \tag{1.81}$$

donde $R$ es el radio de una hiperesfera que encierra todos los puntos. Por lo tanto, resolver (1.19), (1.20) es equivalente a minimizar una cota superior de la dimensión VC.

# CAPITULO II. CONCEPTOS DE OPTIMIZACION Y ALGORITMOS GENETICOS.

En este capítulo se describen las bases para entender los problemas de Programación Cuadrática Convexa con restricciones lineales, los cuales van de la mano con la tarea de entrenamiento de las SVM, así como diferentes opciones para enfocar el manejo de las restricciones que plantean dichos problemas utilizando Algoritmos Genéticos.
En primer lugar se presentan algunas definiciones básicas que se utilizan posteriormente. En segundo lugar tenemos una descripción de los problemas desde Programación No Lineal en general, hasta Programación Cuadrática Convexa. En este contexto se presentan el Teorema de Lagrange y las Condiciones de Karush-Kuhn-Tucker como conceptos importantes en el desarrollo teórico de la solución a las SVM. Finalmente, se hace una rápida exploración sobre los conceptos más importantes de los Algoritmos Genéticos y los métodos con mayor popularidad con los que se manejan problemas con restricciones.

## 2.1 Fundamentos[1].

El problema de aprendizaje (entrenamiento) de una SVM se ha convertido a una forma en que puede ser analizado dentro del marco de la Teoría de la Optimización. La Teoría de la Optimización es la rama de las matemáticas concerniente a la caracterización de soluciones de clases de problemas cuya solución consiste en encontrar un vector de parámetros que minimice o maximice cierta función de costo, típicamente sujeta a ciertas restricciones. Dependiendo de la función de costo específica y de la naturaleza de las restricciones, podemos distinguir un número de clases de problemas de optimización que son bien entendidos y para los cuales existen estrategias de solución eficientes. La Teoría de la Optimización no solo provee técnicas algorítmicas efectivas para encontrar dichas soluciones, sino que también define las condiciones necesarias y suficientes para que una función dada sea una solución.
A continuación se presentan algunas definiciones importantes, que son básicas en el desarrollo de la teoría vista más adelante.

**Formas Cuadráticas.** Una forma cuadrática en $R^n$ es una función de la forma

$$x^T Hx = \sum_{i=1}^{n}\sum_{j=1}^{n} h_{ij} x_i x_j \qquad (2.1)$$

donde $H = (h_{ij})$ es una matriz simétrica de $n \times n$, y $x$ es un vector en $R^n$.

[1] Las definiciones que se presentan en esta sección y en la siguiente están basadas en [Las70], [Min83] y [Sun96].

Se dice que una forma cuadrática es:

- positiva definida si $x^T Hx > 0$ para toda $x$ en $R^n$, $x \neq 0$;
- positiva semi-definida si $x^T Hx \geq 0$ para toda $x$ en $R^n$, $x \neq 0$;
- negativa definida si $x^T Hx < 0$ para toda $x$ en $R^n$, $x \neq 0$;
- negativa semi-definida si $x^T Hx \leq 0$ para toda $x$ en $R^n$, $x \neq 0$.

**Conjunto Convexo.** Se dice que un conjunto de puntos es convexo si, dados dos puntos dentro del conjunto, todos los puntos del segmento de línea que los une están también dentro del conjunto. En términos algebraicos, el segmento de línea entre los puntos $x_A = (x_{A1}, x_{A2}, ..., x_{An})$ y $x_B = (x_{B1}, x_{B2}, ..., x_{Bn})$ en $R^n$ se define como el conjunto $S = \{(x_1, x_2, ..., x_n) \mid x_i = \lambda x_{Ai} + (1-\lambda)x_{Bi}, \forall i\}$, con $0 \leq \lambda \leq 1$.

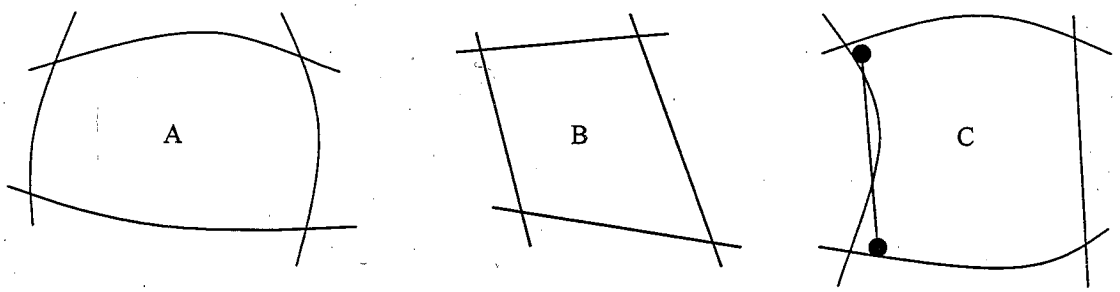


*Figura 2.1. Ejemplos de conjuntos convexos (A y B) y de no convexos (C).*

**Funciones Convexas.** Se dice que una función es convexa si el segmento de línea dibujado entre dos puntos sobre la gráfica de la función nunca pasa por debajo de ella. Una función es estrictamente convexa si el segmento de línea está siempre sobre la función. Algebraicamente, una función $f(x)$, donde $x = (x_1, x_2, ..., x_n)$, es convexa si $f(\lambda x_A + (1-\lambda)x_B) \leq \lambda f(x_A) + (1-\lambda) f(x_B)$ para todas las combinaciones $\{x_A, x_B\}$, dados $x_A = (x_{A1}, x_{A2}, ..., x_{An})$ y $x_B = (x_{B1}, x_{B2}, ..., x_{Bn})$ en el dominio de $f$ y para todo $0 \leq \lambda \leq 1$.

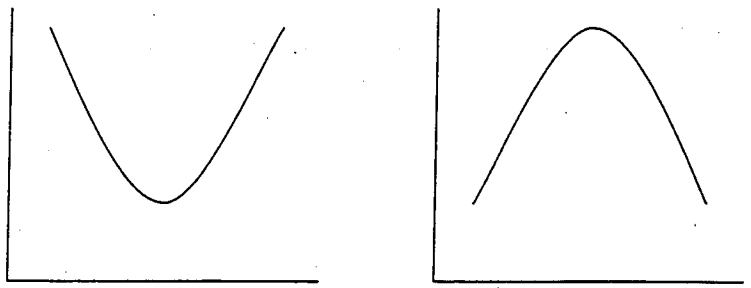

*Figura 2.2. Ejemplos de funciones convexa y cóncava, respectivamente.*

**Funciones Cóncavas.** Se dice que una función es cóncava si el segmento de línea dibujado entre dos puntos debajo la gráfica de la función nunca pasa por sobre de ella. Una función es estrictamente cóncava si el segmento de línea está siempre debajo de la función. Algebraicamente, una función $f(x)$, donde $x=(x_1,x_2,...,x_n)$, es cóncava si $f(\lambda x_A+(1-\lambda)x_B)\geq \lambda f(x_A)+(1-\lambda)f(x_B)$ para todas las combinaciones $\{x_A,x_B\}$, dados $x_A=(x_{A1},x_{A2},...,x_{An})$ y $x_B=(x_{B1},x_{B2},...,x_{Bn})$ en el dominio de $f$ y para todo $0\leq\lambda\leq 1$.

**Propiedades de los Conjuntos y Funciones Convexos.** Algunas propiedades que nos serán útiles son:

- La intersección de dos o más conjuntos convexos es un conjunto convexo.
- Una forma cuadrática positiva semidefinida (matriz de segundas derivadas parciales de una función) es convexa.
- Una combinación lineal positiva de funciones convexas es una función convexa.
- Una función lineal es tanto convexa como cóncava, pero nunca estrictamente convexa o estrictamente cóncava.

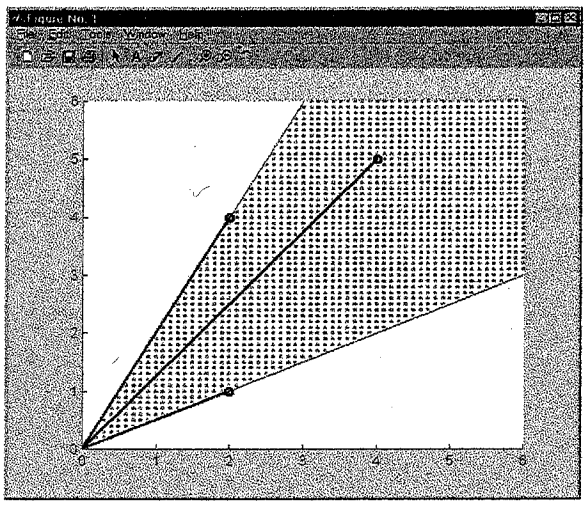


*Figura 2.3. Ejemplo de un cono convexo en dos dimensiones.*

**Cono.** Un cono es un conjunto de puntos $P$, tales que si $x\in P$, entonces $\lambda x\in P$ para toda $\lambda\geq 0$. Un cono convexo es un cono que también es un conjunto convexo. El conjunto de todas las combinaciones lineales no negativas de un conjunto finito de vectores $S=\{x\mid x=\lambda_1x_1+\lambda_2x_2+...+\lambda_mx_m,\lambda_i\geq 0\}$ es un cono convexo y los vectores $x_1,x_2,.....,x_m$ son los generadores del cono.

En la figura 2.4 puede verse un ejemplo de cono en dos dimensiones. Los generadores del cono son los vectores $(2,1)$ y $(2,4)$. El vector $(4,5)$ puede expresarse como una combinación lineal de los otros dos: $1\times(2,1)+1\times(2,4)=(2+2,1+4)=(4,5)$. Cualquier otro vector que pueda ser expresado como una combinación lineal no negativa de algunos otros vectores que caen dentro del cono generado por dichos vectores.

## 2.2 Programación No Lineal.

El problema básico de Programación No Lineal es encontrar $x = (x_1, x_2, ..., x_n)$ que

$$\min_x \quad f(x) \tag{2.2}$$

sujeto a las restricciones

$$g_i(x) \leq 0 \text{ para } i = 1,2,...,l, \tag{2.3}$$

$$g_j(x) = 0 \text{ para } j = l+1, l+2, ..., l+m; \tag{2.4}$$

donde $f : R^n \to R$ es llamada función objetivo, las $g_i : R^n \to R$ son llamadas restricciones de desigualdad y las $g_j : R^n \to R$ son llamadas restricciones de igualdad. El valor óptimo de la función objetivo es llamado valor del problema de optimización. La región del dominio donde la función objetivo está definida y donde todas las restricciones se cumplen es llamada región factible y la denotaremos por

$$\Omega = \{x \in R^n : g_i(x) \leq 0 \quad y \quad g_j(x) = 0\} \tag{2.5}$$

La solución del problema es un punto $x^* = (x_1^*, x_2^*, ..., x_n^*) \in R^n$ tal que no existe otro punto $x \in \Omega$ para el cual $f(x) < f(x^*)$, como puede verse en la figura 2.1.
En una región factible, una cara es el conjunto de puntos que caen en el límite de la misma, y se dice que una restricción es activa cuando la igualdad se cumple.

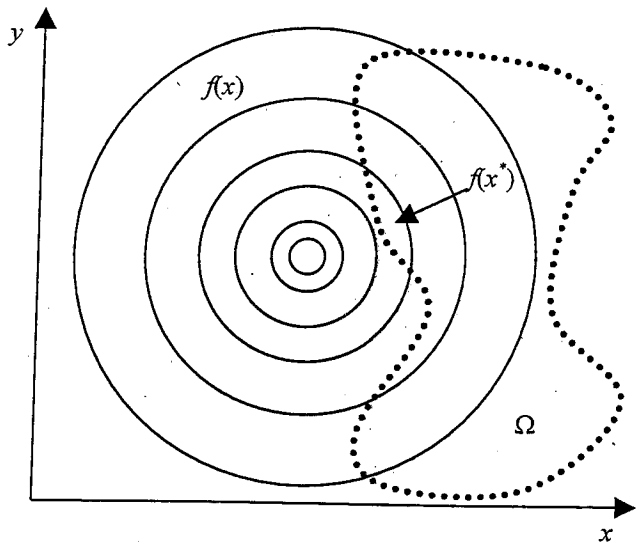


*Figura 2.4. Representación gráfica de un problema de Programación No Lineal. El mínimo global de la función $f(x)$ está en el centro de los círculos, $\Omega$ es la región factible y el mínimo restringido $f(x^*)$ es el punto señalado, es el punto dentro $\Omega$ de que está más cerca del mínimo global.*

**Programación Convexa.** Se dice que un problema de programación matemática es convexo si consiste en minimizar una función convexa (o maximizar una función cóncava) sobre un conjunto de restricciones convexo. Esto es si $f(x)$ es convexa y cada una de las restricciones $g_i(x)$ y $g_j(x)$, son convexas. La propiedad fundamental de un problema de Programación Convexa es que cualquier óptimo local es un óptimo global.

**Programación Cuadrática.** Un problema de Programación Cuadrática es aquel que consiste en minimizar una función de la forma

$$f(x) = x^T Hx + d^T x \tag{2.6}$$

sujeto a las restricciones

$$g_i(x) = A_i \cdot x \leq 0 \text{ para } i = 1,2,...,l, \tag{2.7}$$

$$g_j(x) = A_j \cdot x = 0 \text{ para } j = l+1, l+2,...,m; \tag{2.8}$$

donde $H$ es simétrica y $A$ tiene $l+m$ renglones. La función objetivo $f(x)$ cumple con $\nabla f(x) = Hx + d$, y $\nabla^2 f(x) = H$.

**Programación Cuadrática Convexa.** Un problema de Programación Cuadrática Convexa es un problema de Programación Cuadrática (minimizar una función objetivo convexa), con un conjunto de restricciones convexo.

### 2.2.1 El Teorema de Lagrange.

Dado un problema de optimización no lineal como el anterior, pero tomando en cuenta solamente las restricciones de desigualdad, definimos la función de Lagrange o Lagrangiano como

$$L(x,\alpha) = f(x) + \sum_{i=1}^{l+m} \alpha_i g_i(x) \tag{2.9}$$

Los elementos del vector $\alpha = (\alpha_1, \alpha_2, ..., \alpha_{l+m})$ son llamados multiplicadores de Lagrange. Suponiendo que $x^*$ es un óptimo local de $f(x)$ en $\Omega$, entonces existe un vector $\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_{l+m}^*)$ para el cual

$$\frac{\partial L(x^*,\alpha^*)}{\partial x} = 0 \qquad \frac{\partial L(x^*,\alpha^*)}{\partial \alpha} = 0 \tag{2.10}$$

Los multiplicadores de Lagrange $\alpha_i^*$ miden la sensibilidad del valor de la función objetivo en $x^*$ hacia una pequeña "relajación" de sus respectivas restricciones $g_i(x)$. Si la función $f(x)$ es convexa, entonces estas condiciones son suficientes.

**Punto Silla.** Un punto $(x^*,\alpha^*)$ con $\alpha^* \geq 0$ y $x^* \in \Omega$, es llamado un Punto Silla para el Lagrangiano $L(x,\alpha)$ si satisface $L(x^*,\alpha^*) \leq L(x,\alpha^*)$ para toda $x \in \Omega$ y $L(x^*,\alpha^*) \geq L(x^*,\alpha)$ para toda $\alpha \geq 0$. Esto es, $x^*$ minimiza $L(x,\alpha^*)$ sobre la región factible y $\alpha^*$ maximiza $L(x^*,\alpha)$ sobre todas las $\alpha \geq 0$. Para una función de dos variables con una restricción, si $x=(x_1,x_2)$ y $\alpha \in R$, $L(x,\alpha)$ tiene una forma de silla de montar como puede verse en la figura 2.5.

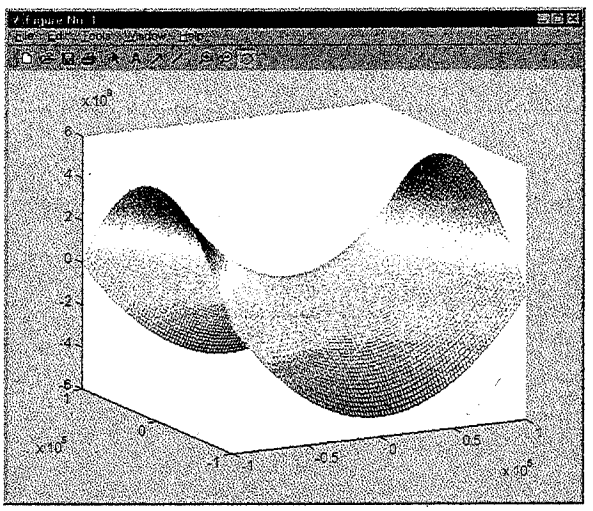

*Figura 2.5. Gráfica de un Lagrangiano en forma de silla de montar en tres dimensiones.*

**Teorema 2.1.** Sean $\alpha^* \geq 0$ y $x_i \in A$. Entonces $(x^*,\alpha^*)$ es un Punto Silla para $L(x,\alpha)$ si y solo si:

a) $x^*$ minimiza $L(x,\alpha^*)$ sobre $\Omega$,

b) $g_i(x) \leq 0$ para $i=1,2,...,l$, y $g_j(x)=0$ para $j=l+1,l+2,...,l+m$;

c) $\alpha^* g_i(x^*)=0$ para $i=1,2,...,l+m$.

Estas son las condiciones necesarias y suficientes para un Punto Silla de $L(x,\alpha)$.

**Teorema 2.2 (Suficiencia del Punto Silla).** Si $(x^*,\alpha^*)$ es un Punto Silla para $L(x,\alpha)$, entonces $x^*$ resuelve el problema primal dado.

***Prueba:*** Como $(x^*,\alpha^*)$ es un Punto Silla, entonces cumple las condiciones del teorema 2.1.

Sea $g(x)=(g_1(x),g_1(x),...,g_{l+m}(x))^T$.

Por la condición (a), $f(x^*)+\alpha^* g(x^*) \leq f(x)+\alpha^* g(x)$ para toda $x \in \Omega$.

Por la condición (c), $\alpha^* g_i(x^*)=0$, por lo tanto $f(x^*) \leq f(x)+\alpha^* g(x)$ para toda $x \in \Omega$.

Para todos los puntos $x \in \Omega$ que satisfacen $g_i(x) \leq 0$ para las restricciones de desigualdad y $g_i(x) = 0$ para las de desigualdad en $g(x)$; es decir, todas las $x$ factibles para el problema primal, el término $\alpha^* g(x)$ es no positivo ($\leq 0$), por lo tanto tenemos que $f(x^*) \leq f(x)$ para toda $x$ factible, por lo tanto $x^*$ resuelve el problema primal.
Este teorema se aplica a cualquier programa matemático, incluyendo programas donde $\Omega$ es un conjunto finito, $f$ o $g_i$ son no convexas, etc. Un Punto Silla puede no existir para tales problemas y su existencia se garantiza sólo para Programas Convexos.

**Dualidad Lagrangiana.** Dado el problema primal de optimización ya mencionado, existe un problema dual asociado de la forma $\max_{\alpha} \; L_D(\alpha) = L_P(x', \alpha')$, donde $x'$ y $\alpha'$ se obtienen igualando a cero el gradiente del Lagrangiano primal $\nabla L_P(x, \alpha)$ y sustituyendo en él las relaciones obtenidas. Las restricciones del problema son las relaciones que no pueden sustituirse, junto con $\alpha_i \geq 0$ para toda $i$.
Definamos ahora para $\alpha \geq 0$ la función $L_D(\alpha) = inf \;\; \{L_P(x, \alpha)\} = \max_{\alpha} \;\; \{\min_{x} \;\; L_P(x, \alpha)\}$.
Suponiendo que las funciones $f$ y $g_i$ son tales que cada punto $\alpha$ donde $L_D(\alpha)$ tiene un valor finito, existe $x^*$ tal que $L_D(\alpha) = L_P(x^*, \alpha)$. Ahora podemos escribir $L_D(\alpha) = \min_{x} \;\; L_P(x, \alpha)$.
Esto se cumple en particular si $f$ y $g_i$ son continuas en $\Omega$ y $\Omega$ es compacto.
La búsqueda del Punto Silla, si existe, puede llevarnos a resolver el problema $\max_{\alpha} \;\; L_D(\alpha)$, donde $L_D(\alpha) = inf \;\; L_P(x, \alpha)$, $\alpha \geq 0$ ($D$). La función dual $L_D$ es una función cóncava de $\alpha$, por lo que $-L_D$ es una función convexa de $\alpha$.

### 2.2.2 Condiciones de Karush-Kuhn-Tucker.

Las Condiciones de Karush-Kuhn-Tucker (Condiciones KKT) son condiciones que debe satisfacer cualquier óptimo restringido, ya sea local o global, de cualquier problema de Programación Lineal y de la mayoría de los problemas de Programación No Lineal. Forman la base de desarrollo de muchos algoritmos computacionales, pues un posible criterio de paro por haber llegado a un óptimo es que se cumplan estas condiciones. Geométricamente, nos dicen que en todo óptimo restringido no hay cambios permisibles en las variables del problema que puedan mejorar la función objetivo.
Supongamos que tenemos un problema de Programación No Lineal que consiste en minimizar $f(x, y)$ sujeto a las restricciones $g_1(x, y)$ y $g_2(x, y)$, las cuales tienen su intersección en el punto $P(x_1, y_1)$, cerca del óptimo global de $f$ que es el punto $Q(x_2, y_1)$, como se muestra en la figura 2.6. El punto $P$, como se ve claramente, resulta ser el óptimo restringido para $f$.
Una dirección factible es un vector tal que un pequeño movimiento a través de él no viola las restricciones. En $P$, el conjunto de todas las direcciones factibles caen entre las tangentes a las curvas $g_1$ y $g_2$ (o las $g_i$ mismas si éstas fueran lineales). Dicho conjunto es el cono generado por esas líneas.

El vector $-\nabla f$ apunta en la dirección de la máxima razón de decremento de $f$ y un pequeño movimiento a lo largo de alguna dirección que haga un ángulo menor a 90° con $-\nabla f$ decrementará $f$. Así, en el óptimo, ninguna dirección factible puede tener un ángulo menor a 90° con $-\nabla f$.

Si consideramos los vectores gradientes $\nabla g_1$ y $\nabla g_2$, vemos que $-\nabla f$ está contenido en el cono generado por dichos gradientes. Si $-\nabla f$ estuviera ligeramente arriba de $\nabla g_2$, podría hacer un ángulo menor a 90° con una dirección factible debajo de la línea $l_2$. Si $-\nabla f$ estuviera ligeramente arriba de $g_1$, podría hacer un ángulo menor a 90° con una dirección factible arriba de la línea $l_1$.

Ninguno de estos casos puede ocurrir en un punto óptimo, y ambos casos son excluidos si y solo si $-\nabla f$ cae en el cono generado por $\nabla g_1$ y $\nabla g_2$. Esto equivale a la necesidad de que $\nabla f$ caiga en el cono generado por $-\nabla g_1$ y $-\nabla g_2$.

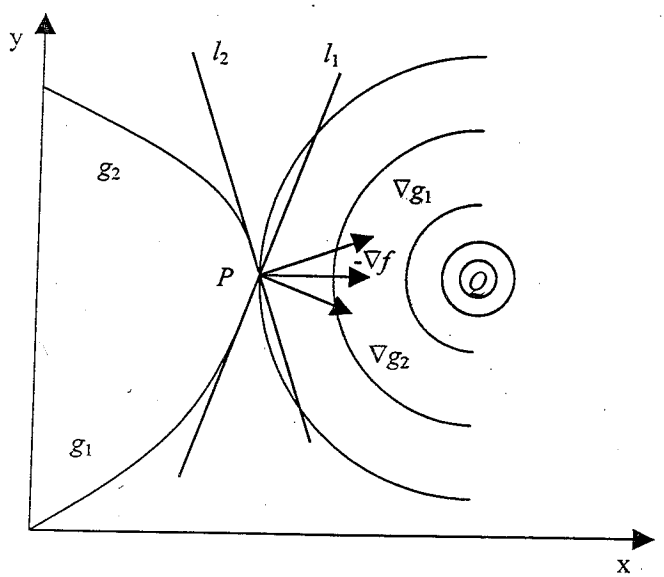


*Figura 2.6. Representación gráfica de las condiciones de Karush-Kuhn-Tucker.*

En términos de enunciado tenemos que si $f$ y $g_i$ son diferenciables, una condición necesaria para que un punto $x^*$ sea un mínimo restringido a un problema de Programación No Lineal es que en $x^*$, $\nabla f$ esté dentro del cono generado por los gradientes negativos de las restricciones de acotamiento. Como $\nabla f$ cae dentro del cono descrito anteriormente, debe ser una combinación lineal no negativa de los gradientes negativos de las restricciones de acotamiento; es decir, deben existir números $\alpha_i^*$ tales que

$$\nabla f(x^*) = \sum_{i=1}^{m} \alpha^*_i \left(-\nabla g_i(x^*)\right) \qquad (2.11)$$

donde $\alpha_i^* \geq 0$ para $i = 1,2,...,l+m$.

Estos resultados pueden ser restablecidos para incluir a todas las restricciones definiendo el coeficiente $\alpha_i^*$ como cero si $g_i(x^*) \leq 0$, con lo que $\alpha_i^* \geq 0$ si $g_i(x^*) = 0$; es decir, el producto $\alpha_i^* g_i(x^*)$ es cero para toda i. Entonces, las Condiciones KKT son:

$$\nabla f(x^*) = -\sum_{i=i}^{m} \alpha^*{}_i \nabla g_i(x^*) \quad \Rightarrow \quad \left( \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n} \right) = \sum_{i=1}^{m} -\alpha^*{}_i \left( \frac{\partial g_i}{\partial x_1} \quad \frac{\partial g_i}{\partial x_2} \quad \cdots \quad \frac{\partial g_i}{\partial x_n} \right) \quad (2.12)$$

$$\alpha_i^* \geq 0 \text{ para } i = 1,2,...,l+m, \quad (2.13)$$

$$\alpha_i^* g_i(x^*) = 0 \text{ para } i = 1,2,...,l+m, \quad (2.14)$$

$$g_i(x^*) \leq 0 \text{ para } i = 1,2,...,l, \quad (2.15)$$

$$g_i(x^*) = 0 \text{ para } i = l+1, l+2,...,l+m. \quad (2.16)$$

## 2.3 Algoritmos Genéticos.

Los Algoritmos Genéticos (AG) son métodos adaptativos que se basan en los procesos evolutivos de los organismos biológicos. Tales procesos siguen los principios de la selección natural y la supervivencia del más apto, establecidos en 1859 por Charles Darwin en su libro "El Origen de las Especies" [Dar00]. Los Algoritmos Genéticos imitan estos procesos pues "evolucionan" soluciones a problemas reales. Fueron formulados inicialmente por John H. Holland a principios de la década de 1960 como parte del Aprendizaje de Máquina (Machine Learning). La presente sección presenta una revisión de los conceptos mas importantes sobre Algoritmos Genéticos, y su contenido esta basado en [Mic96].

En la naturaleza, los individuos de una población compiten entre sí por recursos como comida, agua y refugio; así como para atraer a una pareja. De este modo, los más exitosos o adaptados tendrán relativamente más descendientes que los menos adaptados, lo que hará que sus genes se propaguen en números mayores de individuos en las generaciones sucesivas. Por otro lado, los individuos menos adaptados producirán pocos o nulos descendientes, con lo que sus genes se perderán a través de las generaciones. Esto da como resultado la evolución de una especie hasta adaptarse en forma óptima a su entorno.

La analogía de los Algoritmos Genéticos con la evolución natural es que parten de una población de "individuos", cada uno de los cuales representa una posible solución a un problema dado. A cada uno de tales individuos se le asigna un valor de aptitud (calidad o bondad) que indica lo bueno que es como solución al problema. En base a esto, a los mejores individuos se les da la oportunidad de "reproducirse" (mediante la operación de cruza) y así obtener una nueva población compuesta por sus descendientes. Esta nueva población tendrá una cantidad mayor de individuos con buena adaptación, pues "heredarán" estos rasgos de sus padres. Así mismo, los individuos menos adaptados tenderán a desaparecer. Al repetir el proceso sobre muchas generaciones, las buenas características se propagarán entre la población, mezclándose entre sí para producir poblaciones cuyos individuos serán cada vez más parecidos a la solución buscada.

El poder de los AG viene del hecho de ser una técnica robusta, pues es capaz de adaptarse a una gran cantidad de problemas, incluyendo aquellos difíciles de resolver por otros métodos. Se dice que los AG encuentran una solución aceptablemente buena, aceptablemente rápido. Las técnicas especializadas superan a los AG en velocidad y precisión; pero en áreas difíciles, donde no existen tales técnicas, pueden ser aplicados exitosamente.

Algunas aplicaciones de los AG son: optimización, programación automática, economía, sistemas inmunes, ecología, etc. Entre los problemas de optimización, los problemas más comunes donde se aplican son: ruteo, calendarización, control adaptivo, juegos, modelación del conocimiento, problemas de transporte, problemas del tipo del agente viajero, Aprendizaje de Máquina, control óptimo y diseño de Redes Neuronales.

En comparación con otras técnicas, los AG no se limitan a un dominio pequeño como los métodos de búsqueda aleatoria, gradiente, búsqueda iterada, recocido simulado, etc., pues buscan la solución simultáneamente en varios puntos del espacio, no requieren conocimientos especializados como derivadas, y utilizan reglas de transición probabilísticas en vez de determinísticas.

Lo anterior hace más eficientes a los AG al lograr un buen balance entre la exploración, pues investiga áreas desconocidas del espacio de búsqueda con cada nueva generación, y la explotación, al hacer uso del conocimiento encontrado en los puntos visitados con anterioridad.

Según Holland, la combinación de exploración y explotación en un AG es óptima. Sin embargo, en la práctica el rendimiento está sujeto a errores estocásticos como la deriva genética (el cambio aleatorio en la frecuencia de los alelos de una generación a la siguiente debido a fluctuaciones en las oportunidades de cruza), que también se presenta en la naturaleza.

### 2.3.1 Estructura Básica de los AG.

La estructura de un AG generacional simple se puede ver en la figura 2.7. En cada iteración se tiene una población de posibles soluciones que se evalúan para encontrar su valor de aptitud o simplemente aptitud. Con base a dicha aptitud se hace una selección de parejas de individuos que se recombinan para producir los individuos de una nueva población (cruza), a cuyos miembros se realizan alteraciones en uno o más genes de manera aleatoria (mutación). Después de esto, el proceso se repite a partir de la recién creada población. Cada grupo de individuos se conoce como generación. Se fija un número limitado de generaciones y un tamaño de población adecuado, de acuerdo a la precisión que se desea tenga la solución.

Un AG debe tener los siguientes cinco componentes:

- una representación genética para las soluciones potenciales del problema (individuos);
- una forma de crear una población inicial de soluciones potenciales;
- una función de evaluación que juega el papel del ambiente, midiendo las soluciones en términos de su aptitud;
- operadores genéticos que alteren la composición de los hijos;
- valores de los varios parámetros utilizados (tamaño de la población, probabilidades de aplicación de los operadores genéticos, número límite de generaciones, etc.).

Parámetros del Algoritmo:

tam_pob: tamaño de la población;
$p_c$: probabilidad de cruza;
$p_m$: probabilidad de mutación;
max_iter: número máximo de iteraciones permitidas;
num_iter: contador de iteraciones;
$P_t$: población en la iteración t;
$P'_t$ y $P''_t$: poblaciones auxiliares.

Algoritmo:

$P_0$ = población inicial;
Evaluación: obtener el valor de la aptitud de cada individuo de $P_0$;
num_iter = 0;
Mientras (num_iter < max_iter)
    Selección: seleccionar tam_pob cromosomas de acuerdo a algún criterio de selección y colocarlos en $P'_t$;
    Cruza: elegir parejas de cromosomas de $P'_t$ y, con probabilidad $p_c$ cruzarlas para obtener dos descendientes y colocar todos los descendientes en $P''_t$;
    Mutación: para cada cromosoma en $P''_t$, con probabilidad $p_m$ mutar cada uno de sus genes;
    $P_{t+1} = P''_t$;
    Evaluación: obtener el valor de la aptitud de cada individuo de $P_{t+1}$;
Finm;
Devolver el cromosoma con mayor aptitud.

*Figura 2.7. Forma básica de un Algoritmo Genético generacional simple.*

## Representación o Codificación.

Cada solución potencial o cromosoma puede ser representada como una cadena de $L$ unidades conocidas como genes, rasgos, caracteres o decodificadores. El cromosoma se conoce también como genotipo o estructura, y contiene la información requerida para construir un organismo, conocido como fenotipo, que es el significado de dicho cromosoma que el usuario ha definido externamente mediante la función de aptitud.
Las posiciones de los genes en un cromosoma se conocen como loci o posiciones de cadena, y los diferentes estados en los que puede estar un gen se conocen como alelos o valores de rasgo. Una de las decisiones importantes que se deben tomar al diseñar un algoritmo genético es la forma de representación de las soluciones que se utilizará. Algunos tipos de representación muy utilizados son: binaria, código Gray (binario), entera, real, árboles, expresiones-S (tipo LISP) y representaciones híbridas.

## Población Inicial.

Es la población de individuos que serán los padres de la primera generación. Es obtenida inicializando por separado y de manera aleatoria cada uno de los genes para cada cromosoma que formará parte de ella.

**Convergencia.**

Es el progreso hacia el incremento de uniformidad de una población. Esta evoluciona de manera que la aptitud del mejor individuo y el promedio de las aptitudes de todos los individuos se incrementan, aproximándose al óptimo global. Se dice que un gen converge si en el 95% de los individuos, ese gen tiene el mismo valor. La población converge cuando todos los genes han convergido.

**Función de Aptitud.**

La función de aptitud es el aspecto más crucial de un AG, junto con la forma de representación o codificación. La función de aptitud debe ser suave y regular, de modo que los cromosomas con aptitud razonable estén cerca de los cromosomas ligeramente mejores. Se deben construir funciones de aptitud que no tengan demasiados máximos locales o un máximo global demasiado aislado. Para muchos problemas, dicha construcción no es trivial, pero debe reflejar el valor del cromosoma en forma real. Existen algunos enfoques para distintas situaciones:

- **Función de Castigo o Penalización.** Consiste en utilizar una función de penalización que representa la pobreza del cromosoma y construye la aptitud en base a una constante de penalización. Pueden representar las cantidades por las cuales un cromosoma viola las restricciones o el número de restricciones violadas. Se dice que las primeras son mejores que las segundas.
- **Evaluación de una Función de Aproximación.** Consiste en utilizar una función mucho más rápida de calcular, que aproxima el valor de la verdadera función de aptitud, de manera que se encuentren mejores individuos en un menor tiempo de CPU que si se usara la función de aptitud real.

### 2.3.2 Operadores Genéticos.

Durante la fase reproductiva de los AG, se selecciona un grupo de individuos entre la población para ser recombinados y producir los herederos que constituirán la siguiente generación. Los padres se seleccionan aleatoriamente entre la población, de modo que se favorezca a los mejores individuos. Estos individuos buenos tienen mayores probabilidades de ser elegidos varias veces en una generación, con lo que tendrán más herederos; mientras que los pobremente adaptados difícilmente podrán reproducirse. Una vez seleccionados los padres, se utiliza el operador de cruza para obtener a sus herederos y posteriormente se aplica el operador de mutación a cada individuo.

#### 2.3.2.1 Selección.

Consiste en elegir un grupo de individuos de una población con el objetivo de ser utilizados en la recombinación, de modo que se asignen oportunidades reproductivas a cada individuo. Los elegidos son copiados a una piscina de apareamiento del mismo tamaño de la población.

Una vez que la piscina está llena, se tomarán parejas de individuos de manera aleatoria para cruzarse hasta completar la siguiente generación. Los individuos con mayores aptitudes serán copiados un número mayor de veces que sus congéneres menor adaptados, que incluso pudieran no recibir copia alguna. Los métodos de selección más utilizados son:

**Selección Proporcional de la Aptitud (Ruleta).** Consiste en construir una rueda calculando el valor de aptitud de cada cromosoma y la suma de todas las aptitudes. Después se obtiene la probabilidad de selección de cada cromosoma dividiendo la aptitud de éste entre la aptitud total de la población (normalización). Finalmente se calcula la probabilidad acumulativa para cada cromosoma, sumando la propia a la suma de las anteriores.
Para seleccionar se genera un número aleatorio en el intervalo [0,1] y, si este número es menor a la probabilidad del primer cromosoma, entonces se elige al primer cromosoma; en otro caso se compara el número con las probabilidades de cada uno de los siguientes cromosomas secuencialmente hasta que caiga entre dos probabilidades consecutivas y se elige el cromosoma al que pertenece la probabilidad mayor.

- **Muestreo Estocástico Universal.** Consiste en crear una ruleta como en el método anterior, con un número de marcadores igual al tamaño de la población y separados por espacios de igual tamaño. Así, se genera un número aleatorio y se coloca sobre el primer marcador, con lo cual los demás toman su lugar automáticamente.

**Selección por Torneo.** Consiste en seleccionar un número $k$ de individuos de manera aleatoria y elegir al mejor. El valor $k$ se conoce como tamaño del torneo. Esto se efectúa tantas veces como individuos tenga la población. El valor de $k$ más simple y más utilizado es 2, lo cual se conoce como torneo binario. Entre mayor sea el valor de $k$, se incrementa la presión de selección; es decir, la razón entre la probabilidad de que los individuos con mayor aptitud sean seleccionados como padres y la probabilidad de que un individuo promedio sea seleccionado como padre, lo cual nos indica que tanta diferencia existe entre el mejor individuo y el promedio de la población.

**Elitismo.** Consiste en preservar el mejor cromosoma de una generación para que forme parte de la siguiente: además de tener la oportunidad de cruzarse, lo cual realizará en mayor medida que los demás.

### 2.3.2.2 Cruza.

La cruza consiste en combinar los genes de ambos padres para producir dos herederos, en base a una probabilidad de cruza $P_c$. Por cada par de padres se efectúa la operación de cruza con probabilidad $P_c$ para producir descendientes. En caso contrario, los padres pasan directamente a formar parte de la nueva población. Este proceso se efectúa hasta completar una nueva población de individuos. El valor de $P_c$ está típicamente entre 0.6 y 0.7. El número esperado de cromosomas bajo cruza es $P_c(tam\_pob)$.
Existen varias técnicas para efectuar la cruza, entre ellas están:

**Cruza de 1 Punto.** Consiste en elegir al azar una posición en los cromosomas para producir dos segmentos "cabeza" y dos segmentos "cola". Los segmentos cola son intercambiados para producir dos nuevos cromosomas que heredan algunos genes de cada padre.

**Cruza de 2 Puntos.** Consiste en considerar a los cromosomas como ciclos formados mediante la unión de sus extremos y seleccionando aleatoriamente dos posiciones para producir dos segmentos en cada uno, los cuales se intercambian. La cruza de 1 punto puede verse como una cruza de 2 puntos donde uno de los puntos de corte es el inicio de la cadena. Se dice que la cruza de 2 puntos es mejor que la de 1 punto en términos generales.

**Cruza Uniforme.** Consiste en recorrer simultáneamente las dos cadenas y por cada posición de éstas generar un número aleatorio en el intervalo [0,1], de modo que si es menor a 0.5, el gen del primer padre se copia al primer hijo y el del segundo padre al segundo hijo; en caso contrario, el gen del primer padre va al segundo hijo y el del segundo padre al primero. Cada heredero tendrá una mezcla de genes de ambos padres. El número efectivo de puntos de cruza no es fijo pero en promedio es de $\frac{L}{2}$.

**Cruzas Especiales.** Las cruzas especiales son necesarias en algunos casos específicos de representación. Se trata de problemas que no corresponden a la mayoría, para los cuales se ha estudiado y diseñado funciones de aptitud y representaciones en cromosomas que satisfacen las necesidades de dichos problemas. Un ejemplo de esto es el problema del agente viajero.

### 2.3.2.3 Mutación.

La mutación consiste en alterar uno o más genes en los individuos de la nueva población de manera aleatoria. Generalmente, la mutación se aplica a todos los genes en todos los cromosomas de la población con probabilidad $P_m$. Por cada gen en un cromosoma, se genera un número aleatorio en el intervalo [0,1] y, si es menor a $P_m$, se muta ese gen; si no, se deja igual. El número esperado de genes mutados es $P_m L(tam\_pob)$. El valor de $P_m$ está típicamente entre 0.001 y 0.1, aunque teóricamente debe ser del orden de $\frac{1}{L}$.
Existen varias técnicas para efectuar la mutación, entre ellas están:

**Mutación Uniforme.** Consiste en recorrer cada uno de los genes de cada uno de los cromosomas y con probabilidad $P_m$ hacer un cambio en ellos. Los cambios dependen de la representación utilizada. Por ejemplo: el cambio de bit se utiliza cuando la representación es binaria y consiste en cambiar el bit (gen) de 0 a 1 o viceversa, la cambio aleatorio se utiliza cuando la representación es real o entera y consiste en cambiar el gen correspondiente por un valor dentro de su mismo rango. La mutación minimax consiste en cambiar el gen seleccionado por el valor mínimo o el valor máximo permitidos para los genes, de acuerdo a una selección aleatoria.

**Mutación Definida por el Usuario.** Consiste en efectuar algún cambio a los genes de los cromosomas, de acuerdo a la conveniencia dada por el gusto del usuario y el tipo de problema o representación.

Se dice que la cruza es el operador más importante para la exploración rápida de un espacio de búsqueda, mientras que la mutación reintroduce algunos valores de genes perdidos que previenen la deriva genética y dan una pequeña cantidad de aleatoriedad al proceso, asegurando que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado. Existen, sin embargo, algunos puntos de vista que indican lo contrario, pues aseguran que un operador de mutación adecuadamente modificado puede efectuar todo el trabajo del operador de cruza [Mic96].

## 2.4 Manejo de Restricciones con Algoritmos Genéticos.

Al pensar en solucionar algún problema de optimización mediante Algoritmos Genéticos, siempre se piensa en procedimientos sin restricciones para la solución a encontrar. Sin embargo, las soluciones a los problemas del mundo real casi siempre tienen que cumplir con una o mas restricciones. Cuando se implementa un AG para solucionar problemas, surge la necesidad de incorporar alguna medida de la factibilidad de los individuos a la función de aptitud en forma adecuada, con el fin de preservar tanto a las soluciones factibles como a las que están cerca de serlo.
En general, el manejo de restricciones mediante penalizaciones tiende a ser muy utilizado para solucionar una gran cantidad de problemas de optimización. Sin embargo, existen alternativas para atacar este problema, así como formas de obtener de modo adecuado los factores de penalización que mejores resultados podrían dar.
En esta sección se revisan brevemente los métodos mas interesantes de manejo de restricciones mediante penalizaciones, de acuerdo a Coello [Coe99b].

### 2.4.1 Funciones de Penalización.

La práctica mas utilizada para manejar restricciones (particularmente con restricciones de desigualdad) con AG es el uso de penalizaciones. El enfoque básico consiste en definir el valor de aptitud de un individuo $x_i$, modificando el dominio de la función objetivo $f(x)$ por medio de $aptitud(x_i) = f(x_i) \pm Q(x_i)$, donde $Q(x_i)$ representa una penalización para un individuo no factible $x_i$, o un costo de reparación para tal individuo (el costo de hacerlo factible). Se asume que si $x_i$ es factible, entonces $Q(x_i) = 0$. Durante el proceso, las penalizaciones ayudan a mover a los individuos, a través de las generaciones, hacia la región factible. La forma mas común de penalización es

$$Q(x_i) = c \times \sum_{j=1}^{l+m} \left( g_j(x_i) \right)^2 \qquad (2.17)$$

donde $c$ es un coeficiente de penalización definido por el usuario.

Para elegir el valor de $c$ en forma adecuada, existe una regla llamada Regla de la Mínima Penalización que dice que la penalización debe ser tan baja como sea posible, e indica el límite inferior para el cual las soluciones no factibles son óptimas como lo ideal. Si la penalización es muy alta o baja, entonces el problema llega a ser duro. El concepto es muy simple, pero en la práctica es difícil de implementar, debido a que el lugar exacto del límite entre las regiones factible y no factible se desconoce en la mayoría de los problemas. Pese a ello, hay tres opciones principales para definir una relación entre un individuo no factible y la región factible del espacio de búsqueda son:

- ◊ Penalizar al individuo sólo por ser no factible, sin importar la distancia a la que está de la región factible;
- ◊ Penalizar al individuo de acuerdo a su "cantidad" de no factibilidad;
- ◊ Penalizar al individuo de acuerdo al esfuerzo de "reparación" (costo de hacerlo factible).

Estas relaciones entre individuos y la región factible nos ayudan a construir la función de penalización adecuada a cada problema. Las siguientes guías son básicas para el diseño de funciones de penalización [Coe99b]:

- Los penalizaciones que son funciones de la distancia desde la factibilidad son mejores que aquellas que son solo funciones del número de restricciones violadas.
- Para un problema con pocas restricciones y pocas soluciones completamente factibles, las penalizaciones que sólo son funciones del número de restricciones violadas no son adecuadas para producir algunas soluciones.
- Las buenas funciones de penalización pueden ser construidas a partir de dos cantidades: el costo de terminación máximo y el costo de terminación esperado. El costo de terminación es el costo de hacer factible una solución no factible.
- Las penalizaciones deben estar cerca del costo de terminación esperado, pero no deben caer frecuentemente debajo de éste.

A pesar de que las guías nos llevan a diseñar buenas técnicas para construir funciones de penalización, son difíciles de seguir en algunos casos. Lo más difícil de las funciones de penalización es elegir el(los) valor(es) adecuado(s) para el(los) coeficiente(s) de penalización que asegure(n) la convergencia. A continuación se revisan algunas de las técnicas más importantes.

**Penalizaciones Estáticas.** Consiste en definir varios niveles de violación, y un coeficiente de penalización. escogido para cada nivel, de tal manera que el coeficiente de penalización se incrementa conforme se alcanzan niveles mayores de violación. Este enfoque inicia con una población aleatoria de individuos factibles y no factibles, que se evalúan utilizando la función

$$aptitud(x_i) = f(x_i) + \sum_{j=1}^{l+m} R_{k,j}\left(g_j(x_i)\right)^2 \qquad (2.18)$$

donde $R_{k,j}$ son los coeficientes de violación utilizados, $l+m$ es el número de restricciones, $f(x)$ es la función objetivo no penalizada y $k = 1,2,...,v$, donde $v$ es el número de niveles de violación definidos por el usuario.

La idea es balancear las restricciones individuales separadamente, por medio de la definición de un conjunto diferente de factores para cada una de ellas, a través de la aplicación de un conjunto de reglas determinísticas. Sin embargo, la principal desventaja de esta técnica es el alto número de parámetros requerido, pues son necesarios $(l+m)(2v+1)$ parámetros en total.

**Penalizaciones Dinámicas.** Consiste en utilizar penalizaciones que cambian con el tiempo (las generaciones). Los individuos son evaluados en la generación $t$ utilizando:

$$aptitud(x_i) = f(x_i) + (C \times t)^{\alpha} \sum_{j=1}^{l+m} \left| g_j(x_i) \right|^{\beta} \tag{2.19}$$

donde $C$ y $\alpha$ son constantes definidas por el usuario, $l+m$ es el número de restricciones, y $\beta$ es también una constante definida por el usuario, generalmente con valores de 1 ó 2. Esta función dinámica incrementa la penalización conforme al número de generaciones.
Hay quienes argumentan que las penalizaciones dinámicas trabajan mejor que las estáticas. Sin embargo, en la práctica es tan difícil determinar buenas funciones de penalización dinámica como de penalización estática. Aparentemente, esta técnica provee muy buenos resultados sólo cuando la función objetivo es cuadrática, lo cual es de tomar en cuenta al momento de aplicar estas técnicas al problema de las SVM.

**Penalizaciones de Recocido.** Se basa en la idea del recocido simulado y consiste en cambiar los coeficientes de penalización una vez por cada cierto número de generaciones. En cada iteración se consideran solamente las restricciones activas y el penalización se incrementa con el tiempo (la temperatura se decrementa con el tiempo) de modo que los individuos no factibles son duramente penalizados en las últimas generaciones, hasta alcanzar una temperatura final de congelación previamente definida.
Este método requiere que las restricciones sean divididas en 4 grupos: igualdades lineales, desigualdades lineales, igualdades no lineales y desigualdades no lineales. Se crea un conjunto de restricciones activas $A$ con todas las igualdades no lineales y todas las desigualdades no lineales violadas. La población se evalúa utilizando

$$aptitud(x_i) = f(x_i) + \frac{1}{2\tau} \sum_{j \in A} \left( g_j(x_i) \right)^2 \tag{2.20}$$

donde $\tau$ es el horario de enfriamiento (intervalo de modificación de los factores de penalización).
Un aspecto interesante de este enfoque es que la población inicial no es realmente diversa, sino que consiste en múltiples copias de un solo individuo que satisface todas las restricciones lineales. En cada iteración, la temperatura $\tau$ disminuye y se crea la nueva población utilizando la mejor solución encontrada en la iteración previa como punto inicial para la siguiente iteración.
Dos de las principales desventajas del enfoque son su extrema sensibilidad a los valores de sus parámetros y la dificultad de escoger un esquema de enfriamiento apropiado.

**Penalizaciones Adaptativas.** Consiste en utilizar una función de penalización que toma una retroalimentación del proceso de búsqueda. Cada individuo es evaluado por la fórmula

$$aptitud(x_i) = f(x_i) + \lambda(t)\sum_{j=1}^{l+m}\left(g_j(x_i)\right)^2 \tag{2.22}$$

donde $\lambda(t)$ es actualizado en cada generación $k$ de la siguiente manera

$$\lambda(t+1) = \begin{cases} \left(\frac{1}{\beta_1}\right)\lambda(t) & en\ \ caso\ \ \#1 \\ \beta_2\lambda(t) & en\ \ caso\ \ \#2 \\ \lambda(t) & en\ \ otro\ \ caso \end{cases} \tag{2.23}$$

Cuando el mejor individuo de las últimas $k$ generaciones fue siempre factible tenemos el caso #1. Por el contrario, si el mejor individuo de las últimas $k$ generaciones fue siempre no factible tenemos el caso #2. Se recomienda utilizar $\beta_1, \beta_2 > 1$ y $\beta_1 \neq \beta_2$ para evitar enciclamientos. En otras palabras, el componente de penalización $\lambda(t+1)$ para la generación $t+1$ se decrementa si todos los mejores individuos en las últimas $k$ generaciones fueron factibles o se incrementa si fueron no factibles. Si hay algunos individuos factibles y no factibles empatados como los mejores en la población, entonces el penalización no cambia.
La desventaja obvia es la forma de escoger el valor apropiado para $k$ que provea información razonable para guiar la búsqueda y, más importante, cómo definir los valores de $\beta_1$ y $\beta_2$ para penalizar adecuadamente a una solución dada.

**Penalizaciones Auto-Adaptativas.** Consiste en utilizar una función de penalización de la forma

$$aptitud(x_i) = f(xv) - (coef \times w_1 + viol \times w_2) \tag{2.24}$$

$w_1$ y $w_2$ son factores de penalización (enteros); *coef* es la suma de todas las cantidades por las cuales las restricciones son violadas (sólo para restricciones de desigualdad)

$$coef = \sum_{j=1}^{l} g_j(x_i) \quad \forall g_j(x_i) > 0 \tag{2.25}$$

*viol* es un factor entero que inicia en cero y se incrementa en uno por cada restricción del problema que es violada, no importando la cantidad de la violación.
En este enfoque, el penalización se divide realmente en dos valores (*coef* y *viol*), de modo que el AG tiene suficiente información sobre cuantas restricciones fueron violadas y sobre las cantidades en las cuales dichas restricciones fueron violadas.

Aquí se utilizan dos poblaciones (coevolución). La primera compuesta por soluciones al problema de optimización, y la segunda por combinaciones de los pesos $w_1$ y $w_2$ que serían usados para evaluar a los individuos de la primera.

**Algoritmo Genético Segregado.** Consiste en utilizar dos parámetros de penalización para cada restricción en dos poblaciones distintas, con el objetivo de alcanzar un balance entre los penalizaciones pesados y moderados. Siempre que individuos de estas dos poblaciones se entrecrucen, son "segregados" en términos de satisfacción de una cierta restricción.
Los mejores individuos de cada grupo son escogidos como padres para la siguiente generación, la cual busca combinar individuos factibles y no factibles para mantener la diversidad.
El problema aquí es la manera de escoger los penalizaciones para cada una de las dos subpoblaciones que puedan ser usados en cualquier problema del cual no se tenga información disponible.

**Función de Penalización Basada en Factibilidad.** Consiste en evaluar a los individuos utilizando

$$aptitud(x_i) = \begin{cases} f(x_i) \quad si \quad g_j(x_i) \geq 0 \quad \forall j = 1,2,...,l+m \\ f_{peor} + \sum_{j=1}^{l+m} g_j(x_i) \qquad otro \quad caso \end{cases} \tag{2.26}$$

donde $f_{peor}$ es el valor de la función objetivo de la peor solución factible de la población y las restricciones $g_j(x_i)$ son de desigualdad (las de igualdad se transforman en restricciones de desigualdad usando una tolerancia). Si no hay soluciones factibles en la población, entonces $f_{peor}$ es cero.
Se utilizan la selección por torneo binario y las siguientes reglas para comparar dos individuos:

- Una solución factible se prefiere siempre sobre una no factible;
- Entre dos soluciones factibles, se prefiere a la que tenga mejor función objetivo;
- Entre dos soluciones no factibles, se prefiere a la que tenga menor violación de restricciones.

No se requiere un factor de penalización, pues el procedimiento de selección sólo realiza comparaciones por parejas.
La parte desventajosa es que parece haber problemas para mantener la diversidad en la población y es necesario utilizar métodos de nichos con probabilidades de mutación mayores a las usuales.

**Pena de Muerte.** Consiste en asignar un valor de cero a la aptitud de una solución que viola una restricción, sin necesidad de cálculos para estimar el grado de no factibilidad de la solución. Se dice que es la más fácil y eficiente manera de manejar restricciones. El enfoque normal es recursivo, donde se genera un nuevo punto en cada llamada recursiva, hasta encontrar una solución factible. Esto pudiera ser un proceso muy largo en problemas en los que sea muy difícil de aproximar la región factible.

Un enfoque interesante consiste en determinar la aptitud de un individuo utilizando

$$aptitud(x_i) = \begin{cases} f(x_i) & si \quad solución \quad factible \\ K - \sum_{k=1}^{s} \frac{K}{l+m} & otro \quad caso \end{cases} \tag{2.27}$$

donde $s$ es el número de restricciones satisfechas, y $K$ es una constante grande ($1 \times 10^9$ en experimentos reportados). Nótese que cuando un individuo es no factible, su aptitud no se calcula y todos los individuos que violan el mismo número de restricciones reciben el mismo penalización, independientemente de que tan cerca estén de la región factible.
El pena de muerte está limitada a problemas en los que el espacio de búsqueda factible es convexo y constituye una gran porción del espacio de búsqueda completo. Tiene la desventaja de no explotar información alguna sobre los puntos no factibles que pudieran generarse por el algoritmo evolutivo para guiar la búsqueda.

# CAPITULO III. ENTRENAMIENTO DE SVM CON ALGORITMOS GENETICOS.

En este capítulo, se aborda la solución al problema de optimización planteado como el entrenamiento de las SVM, desde el enfoque de los Algoritmos Genéticos. La utilización de AG en el entrenamiento de SVM se debe a la idea de explorar nuevos caminos en la búsqueda de mejores maneras de solucionar el problema mencionado. Primeramente se plantea la forma en que se manejaron las restricciones del problema, a fin de determinar la función de aptitud adecuada. Posteriormente se explican brevemente las maneras en que se trataron los elementos propios de los AG en la implementación, así como los parámetros de la misma. Finalmente, se muestran algunos resultados obtenidos a partir de ejemplos sencillos de solución con esta técnica, dando paso a conclusiones sobre las ventajas y desventajas de su utilización.

## 3.1 Manejo del Problema y sus Restricciones.

Uno de las manera de resolver el problema de Programación Cuadrática de las SVM es con Algoritmos Genéticos. Para esto debemos tomar en cuenta la formulación del problema, de la cual obtenemos la siguiente función de aptitud

$$f(\alpha) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i, x_j) + \sum_{i=1}^{l}\alpha_i \qquad (3.1)$$

Si nos fijamos bien, el problema dado por las ecuaciones (1.63) a (1.65) tiene $l+1$ restricciones: una de igualdad (1.64) y $l$ de desigualdad (1.65). La región factible esta conformada con valores del vector $\alpha$ todos positivos y que además cumplan con la combinación lineal (1.64). Hacer que las soluciones cumplan con las restricciones de desigualdad es fácil de lograr por medio de la representación, como se plantea más adelante, mientras que lo mas adecuado es incorporar la restricción de igualdad a la función objetivo mediante una penalización. De esa forma se reduce la región factible dentro del AG a estar determinada por la restricción (1.64).

Para hacer lo anterior, debemos tomar en cuenta que dicha restricción nos lleva a una igualdad a cero y podemos darnos cuenta que sea mayor o menor a cero entonces no se cumpliría. Así, le daremos importancia a que tan lejos o cerca está la solución evaluada de la región factible. Esto se determina mediante el valor absoluto de la combinación lineal (o su cuadrado) multiplicada por un factor de penalización.

Cada posible solución, que ahora será un cromosoma, que cumpla con la igualdad deseada eliminará el factor de penalización. Se espera que conforme evolucione la población hacia la solución, cada vez mas cromosomas tendrán penalización cero.

Se hicieron diversas pruebas con una penalización estática con distintos parámetros y con distintos límites para el número de generaciones, pero sin buenos resultados. Finalmente se utilizó la siguiente variante de la estrategia de penalizaciones dinámicas

$$aptitud(\alpha) = f(\alpha) - C\left(\frac{t}{G}\right)R(\alpha) \tag{3.2}$$

donde $C$ es el factor de penalización, $G$ es el número total de generaciones, $t$ es el número de la generación actual y

$$R(\alpha) = \left|\sum_{i=1}^{l} \alpha_i y_i\right| \tag{3.3}$$

Como puede verse, esta estrategia es una variante de la que se presentó en la ecuación (2.19), donde $\alpha = 1$, $\beta = 1$ y en vez de simplemente multiplicar el factor de penalización C por el número de generación $t$, se efectuó el producto con un valor normalizado del mismo. Esto es, el factor de penalización resulta ser proporcional al porcentaje de iteraciones del AG. También se hicieron pruebas con un valor de $\beta = 2$, pero los resultados no fueron lo que se esperaba.

## 3.2 Implementación del Algoritmo Genético.

Para la implementación de la estrategia de solución de las SVM con AG, se utilizó un Algoritmo Genético Generacional simple. El algoritmo, cuyo código puede verse en el Apéndice C, fue implementado en Lenguaje C utilizando el compilador Borland C++ 5.0. Los componentes del AG (ver sección 2.3.1) implementados se describen a continuación:

**Función de Aptitud.** Se utilizó la función de aptitud descrita por (3.1), (3.2) y (3.3). Dicha función tiene la virtud de que al principio será suave, permitiendo a las soluciones encaminarse hacia la región factible con mayor rapidez. Por ahí de la mitad del proceso, las soluciones no factibles mas alejadas empezarán a quedarse marginadas y, por lo tanto, desaparecerán. Al final del proceso, la gran mayoría de los cromosomas cumplirán con la restricción y la búsqueda del máximo de la función de aptitud se dará en forma natural; es decir, como si no hubiera restricción alguna.

**Representación.** Cada cromosoma tendrá tantos genes como elementos tenga el vector $\alpha$ (ya sabemos que tendremos tantas $\alpha_i$ como vectores de entrenamiento). Se utilizó una representación flotante, manejada con un cierto número de dígitos antes y después del punto decimal, dados por el usuario. Entre los parámetros del algoritmo están estas cantidades de dígitos. Los cromosomas se manejan en forma de cadenas de caracteres que sólo pueden ser dígitos del 0 al 9. Cada gen tendrá tantos caracteres como la suma de estas dos cantidades y cada cromosoma constará de tantos caracteres como el número de genes multiplicado por dicha suma.

**Población Inicial.** Para obtener la población inicial se generan números aleatorios entre 0 y 9 para cada uno de los genes, guardando en cada gen el carácter correspondiente valor del dígito obtenido.

**Selección.** Se utiliza como estrategia de selección el torneo binario. Sin embargo, antes de efectuarlo se toma al mejor de la población para que forme parte de los individuos que se cruzarán, de modo que tal individuo siempre tendrá posibilidades de tener descendencia.

**Cruza.** Durante la cruza se utiliza una forma de elitismo que consiste en pasar directamente a la nueva población dos copias del mejor individuo de la generación anterior. Luego se completa el 10% de esta nueva población con hijos producidos por la cruza del mismo mejor con todos los demás, para finalmente efectuar cruzas de todos contra todos, en las que, probabilidad $P_c$ se cruzan y en caso contrario se copian ambos padres a la nueva generación. El tipo de cruza utilizada en todos los casos es la cruza uniforme.

**Mutación.** Para mutar a los individuos se utiliza la mutación uniforme con cambio aleatorio; es decir, se recorren todos los genes y, para cada uno, con probabilidad $P_m$ se les hace una mutación, consistente en obtener un dígito aleatoriamente y asignárselo sólo en caso de que sea diferente de él.

**Parámetros.** Además de los parámetros considerados anteriormente para el algoritmo desarrollado, tenemos: el número máximo de generaciones, el tamaño de la población, la dimensión de los vectores de entrenamiento y el factor de castigo, además del valor del parámetro del kernel utilizado en el entrenamiento.

## 3.3 Resultados Experimentales.

Algunos de los experimentos que se efectuaron son los siguientes.

**Ejemplo 3.1.**
Comencemos por el conjunto de puntos dado por la tabla 3.1. Realmente es un conjunto de entrenamiento que nos da un problema sencillo de resolver. Lo que se desea es ilustrar qué sucede. Se resolvió el problema de Programación Cuadrática de dos maneras: con una sola ejecución de la función QuadProg de Matlab (de la que se hablará más adelante) y con el programa que implementa el Algoritmo Genético.

| *Clase 1 (Azules)* | ***Clase -1 (Negros)*** |
|---|---|
| (1,5), (2,2) | (4,5), (3,4), (5,3) |

*Tabla 3.1. Puntos de entrenamiento para los ejemplos 3.1 y 3.2.*

Para el Algoritmo Genético se utilizaron los siguientes parámetros: número máximo de generaciones 3000, tamaño de la población 50, probabilidad de cruza 0.65, probabilidad de mutación 0.04, dígitos delante del cero 2, dígitos detrás del cero 3, castigo 100 y un kernel gaussiano de ancho 2.

| *Función QuadProg* | *Algoritmo Genético* |
|---|---|
| Vectores de Soporte: 4 (80.00%) | Vectores de Soporte: 4 (80.00%) |
| VS[1]: (1,5). alfa: 1.73. clase: 1. | VS[1]: (1,5). alfa: 1.70. clase: 1. |
| VS[2]: (2,2). alfa: 1.86. clase: 1. | VS[2]: (2,2). alfa: 1.80. clase: 1. |
| VS[4]: (3,4). alfa: 3.13. clase: -1. | VS[4]: (3,4). alfa: 3.00. clase: -1. |
| VS[5]: (5,3). alfa: 0.45. clase: -1. | VS[5]: (5,3). alfa: 0.50. clase: -1. |

*Tabla 3.2. Vectores de soporte resultantes de las corridas del ejemplo 3.1.*

Los resultados de ambas corridas pueden verse en la tabla 3.2. Es claro que el resultado del Algoritmo Genético se aproxima muchísimo a la solución dada por el método de Programación Cuadrática tradicional, implementado en la función utilizada. Sin embargo, las pequeñas diferencias de valores decimales nos dan una región de decisión ligeramente distinta, como puede observarse (con un poco de detenimiento) en la comparación grafica de los resultados que nos proporciona la figura 3.1. Es claro que con un poco más de generaciones se hubiera llegado a la solución, y que el AG identificó los vectores de soporte.

Como convención, las regiones de decisión presentadas en todas las gráficas de la tesis muestran claramente con colores las áreas para las dos clases. Para los puntos azules la región es verde con su mitad del margen de color azul claro, y para los puntos negros la región es púrpura con su mitad del margen amarilla. Cabe señalar que las fronteras de color representan a los hiperplanos: de separación en la frontera azul claro-amarillo, $H_1$ en la frontera verde-azul claro, y $H_2$ en la frontera amarillo-púrpura. Los vectores de soporte se muestran con un espacio en blanco en el centro.

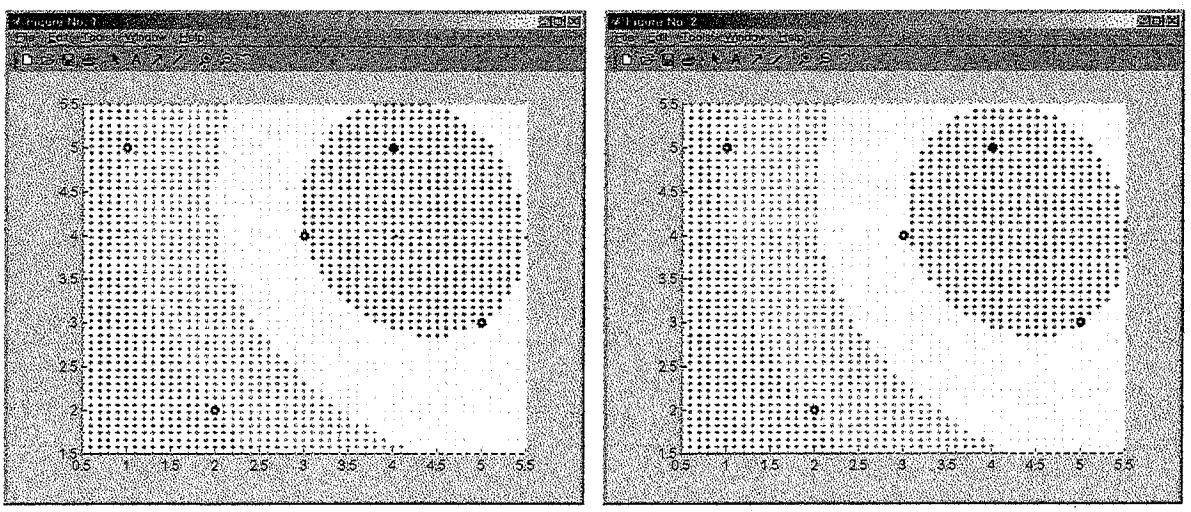

*Figura 3.1. Comparación gráfica de los resultados del ejemplo 3.1.*

**Ejemplo 3.2.**

Para este segundo ejemplo se utilizaron de nuevo los puntos de la tabla 3.1. Nuevamente se resolvió el problema de Programación Cuadrática de las dos maneras ya mencionadas. La diferencia aquí es que se utilizó un kernel lineal para hacerlo, y que se corrió el Algoritmo Genético con dos combinaciones de valores para los parámetros del programa.

La primera, que dio resultados similares a la función de Matlab, fue: número máximo de generaciones 3000, tamaño de la población 50, probabilidad de cruza 0.65, probabilidad de mutación 0.05, dígitos delante del cero 1, dígitos detrás del cero 3, castigo 90 y un kernel lineal.

La segunda combinación tenía los parámetros: número máximo de generaciones 3000, tamaño de la población 50, probabilidad de cruza 0.65, probabilidad de mutación 0.04, dígitos delante del cero 2, dígitos detrás del cero 3, castigo 100 y un kernel lineal.

En la tabla 3.3 podemos ver las salidas de ambas corridas. Es claro que la diferencia en los factores de castigo y algunos parámetros del Algoritmo Genético producen una diferencia en los resultados. Sin embargo, si ponemos atención podremos darnos cuenta de que los vectores de soporte son los mismos en ambos casos. Esto nos dice que el Algoritmo Genético, a pesar de no llevarnos a la solución, si nos aproxima rápidamente a ésta.

| ***Función QuadProg y Primer AG*** | ***Algoritmo Genético*** |
|---|---|
| Vectores de Soporte: 3 (60.00%) | Vectores de Soporte: 3 (60.00%) |
| VS[1]: (1,5). alfa: 0.40. clase: 1. | VS[1]: (1,5). alfa: 5.00. clase: 1. |
| VS[2]: (2,2). alfa: 0.40. clase: 1. | VS[2]: (2,2). alfa: 5.00. clase: 1. |
| VS[4]: (3,4). alfa: 0.80. clase: -1. | VS[4]: (3,4). alfa: 10.00. clase: -1. |

*Tabla 3.3. Vectores de soporte resultantes de las corridas del ejemplo 3.2.*

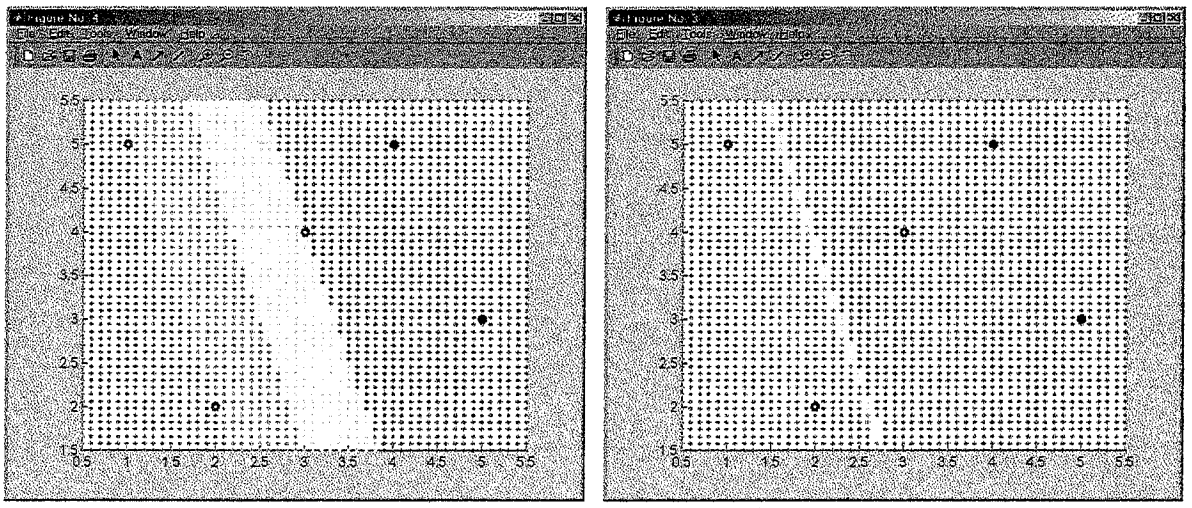

*Figura 3.2. Comparación gráfica de los resultados del ejemplo 3.2.*

Esto se puede notar al observar la figura 3.2, donde comparamos los resultados de ambos enfoques de manera gráfica. Claramente, el hiperplano de decisión tiene prácticamente la misma pendiente y parece estar cerca de la solución real. La diferencia es que el margen es más angosto. Podemos predecir que, conforme los valores de los multiplicadores de Lagrange disminuyen, el margen crece y el hiperplano se ubica en su lugar. Otra situación que debemos mencionar es que en ambos casos, los multiplicadores de la clase 1 son exactamente la mitad del de la clase -1. Los valores son linealmente proporcionales.

**Ejemplo 3.3.**

A continuación veremos como el Algoritmo Genético puede predecir con cierta exactitud cuales de los vectores de entrenamiento serán los vectores de soporte, a pesar de no llegar a la solución correcta. Consideremos los siguientes 40 puntos tomados de dos dimensiones de los datos de la base de datos Iris [Fis36]. Se trata de anchos vs largos de pétalos de las clases setosa y versicolor de la flores de las especies iris[2]. Podemos ver esos datos en la tabla 3.4.

| *Clase 1: Setosa (Azules)* | *Clase -1: Versicolor (Negros)* |
|---|---|
| (1.4,0.2),(1.3,0.2),(1.5,0.2),(1.7,0.4),(1.4,0.3),<br>(1.5,0.1),(1.6,0.2),(1.4,0.1),(1.1,0.1),(1.2,0.2),<br>(1.5,0.4),(1.3,0.4),(1.7,0.3),(1.5,0.3),(1.7,0.5),<br>(1.9,0.2),(1.6,0.4),(1.3,0.3),(1.9,0.4),(1.6,0.6) | (4.7,1.4),(4.5,1.5),(4.9,1.5),(4,1.3),(4.6,1.5),<br>(4.5,1.3),(4.7,1.6),(3.3,1),(4.6,1.3),(3.9,1.4),<br>(3.5,1),(4.2,1.5),(4,1),(3.6,1.3),(4.4,1.4),<br>(4.1,1),(3.9,1.1),(4.7,1.2),(4.6,1.4),(4,1.2) |

*Tabla 3.4. Puntos de entrenamiento para el ejemplo 3.3.*

Es bien sabido que los datos de setosa son linealmente separables respecto de las otras dos clases, aún en 4 dimensiones. Se ejecutaron ambas maneras de solución, donde el algoritmo genético tenía los siguientes parámetros: número máximo de generaciones 3000, tamaño de la población 50, probabilidad de cruza 0.65, probabilidad de mutación 0.0083, dígitos delante del cero 1, dígitos detrás del cero 3, castigo 90 y un kernel lineal. En la tabla 3.5 se muestran los resultados obtenidos con cada enfoque.

| *Función QuadProg* | *Algoritmo Genético* |
|---|---|
| Vectores de Soporte: 2 (5.00%)<br>VS[19]: (1.9,0.4). alfa: 0.86. clase: 1.<br>VS[28]: (3.3,1). alfa: 0.86. clase: -1. | Vectores de Soporte: 4 (10.00%)<br>VS[16]: (1.9,0.2). alfa: 0.10. clase: 1.<br>VS[19]: (1.9,0.4). alfa: 2.10. clase: 1.<br>VS[28]: (3.3,1). alfa: 1.39. clase: -1.<br>VS[31]: (3.5,1). alfa: 0.81. clase: -1. |

*Tabla 3.5. Vectores de soporte resultantes de las corridas del ejemplo 3.3.*

[2] Iris es un conjunto de datos con 150 muestras aleatorias de flores de las especies iris: *setosa*, *versicolor* y *virgínica*. Para cada especie hay 50 observaciones para su longitud del pistilo, ancho del pistilo, longitud de los pétalos y ancho de los pétalos medidos en centímetros. Estos datos fueron utilizados por Fisher [Fis36] en 1936 para ilustrar los principios del análisis del discriminante.

Es bien claro que los vectores de soporte dados por el Algoritmo Genético son un superconjunto de los vectores de soporte reales. Ya habíamos visto como en los ejemplos anteriores el Algoritmo Genético obtiene vectores de soporte coincidentes, aunque los valores de los multiplicadores de Lagrange difieran. En la figura 3.3 podemos observar que sucede algo similar a lo del ejemplo 3.2, pues tenemos un hiperplano de separación paralelo, con un margen mas angosto y con multiplicadores de Lagrange mas grandes. Puede verse que los que no son realmente vectores de soporte tienden a hacerse cero, mientras que los que sí lo son tienen multiplicadores de Lagrange cercanos a lo que deberían ser. Otra vez, el número de generaciones podría darnos mucha mayor precisión en la solución.

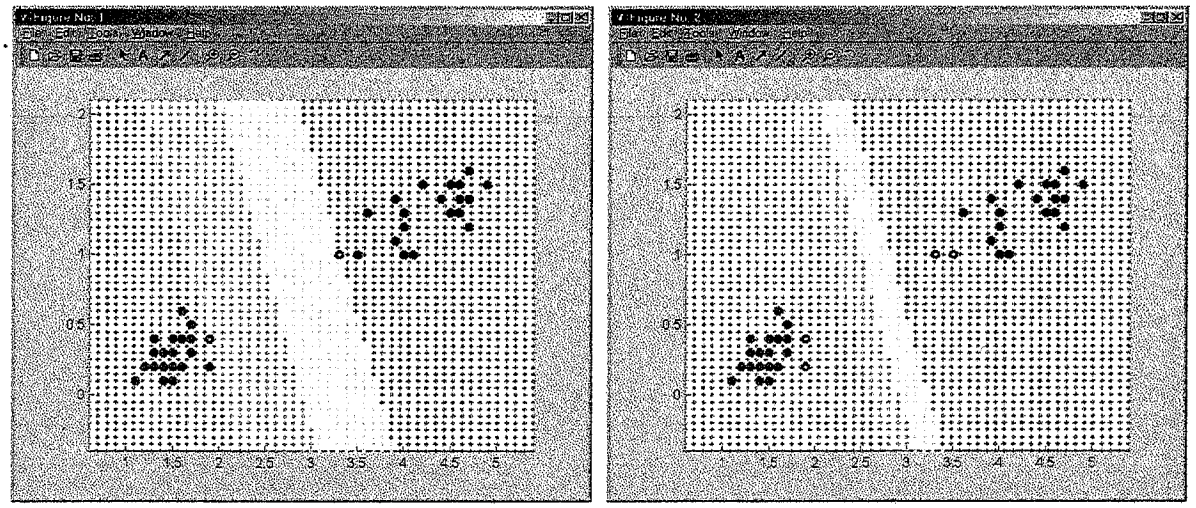

*Figura 3.3. Comparación gráfica de los resultados del ejemplo 3.2.*

Podemos decir que no es tan práctico utilizar Algoritmos Genéticos para solucionar SVM, pero parece ser posible predecir cuales serán los vectores de soporte cuando se haga el entrenamiento. Es bastante interesante la manera en que los multiplicadores de los vectores que no están cerca del margen toman el valor cero en pocas iteraciones (3000). De manera que el AG puede hacer la aproximación gruesa a la solución, y un método de los descritos a continuación la refinaría.

# CAPITULO IV. ENTRENAMIENTO DE SVM CON EL METODO DE CHUNKING SUAVE.

Este capítulo introduce algunos de los métodos utilizados para la solución del problema de Programación Cuadrática propio de las SVM, llevándolo hasta el concepto de conjunto de trabajo. Posteriormente se explican brevemente tres de ellos, en los que se basa el método propuesto: Chunking Suave. Se da una explicación bastante completa y con algunos ejemplos.

## 4.1 Solución del Problema de Optimización en SVM.

Es muy difícil encontrar un problema de optimización para una SVM que puede resolverse en forma analítica. En la inmensa mayoría de los casos del mundo real, dicho problema de optimización debe resolverse numéricamente. Aquí podemos distinguir dos clases de problemas: los problemas pequeños, que se puede resolver utilizando cualquier paquete de optimización de propósito general que resuelva problemas de programación cuadrática convexa con restricciones lineales, y los problemas grandes, para los cuales es necesario utilizar algoritmos especialmente diseñados, un poco más consistentes.

### 4.1.1 Métodos de Solución Utilizados.

La solución del problema de Programación Cuadrática al entrenar una SVM parece ser simple y directo. Se han utilizado varias herramientas de optimización para entrenar las SVM. Inclusive se han utilizado, sobre todo al principio, algoritmos de ascenso de gradiente, gradiente conjugado y otros de Programación Cuadrática de gran escala acotados.
Es bien conocido que la precisión numérica de las computadoras puede causar que las soluciones al problema de Programación Cuadrática no sean lo óptimas que se desean. Pero en la mayoría de la literatura sobre SVM no se discuten las dificultades potenciales que se tienen al resolver dicho problema debido a la precisión numérica limitada. Esto da una impresión de que aparte de la longitud del tiempo requerido para obtener la solución, no importa que optimizador es utilizado.

**Métodos de Conjunto Activo.** Puede verse el problema como el requerimiento de la solución de una secuencia de problemas con restricciones de igualdad. Cada uno de estos problemas puede resolverse en un paso con el Método de Newton o en a lo más $l$ pasos con el Método del Gradiente Conjugado.
Algunos algoritmos se mueven dentro de una cara dada hasta encontrar una nueva restricción, en cuyo caso el algoritmo es reiniciado con la nueva restricción agregada a la lista de restricciones de igualdad. Este método tiene la desventaja de que sólo puede hacerse activa una restricción a un tiempo.

**Métodos de Proyección.** Se consideran también, donde un punto fuera de la región factible es calculado, y entonces se hacen búsquedas y proyecciones en línea, de modo que el movimiento actual permanezca dentro de la región factible. Este enfoque puede agregar varias restricciones nuevas a la vez.

**Métodos de Punto Interior.** Las variables son esencialmente reescaladas de modo que permanezcan dentro de la región factible. Estos métodos son útiles cuando el número esperado de vectores de soporte es una fracción grande de las muestras de entrenamientos.

**Método Reflexivo de Newton [CLi96].** Se aplica a la minimización de una función cuadrática de muchas variables sujeta a límites superiores e inferiores en algunas de las variables. El método se aplica a una función cuadrática (indefinida) general para la cual se requiere un minimizador local sujeto a los límites y es particularmente conveniente para el problema de gran escala. Este método exhibe características fuertes de la convergencia, convergencia global y de segundo orden, y parece tener potencial práctico significativo. Los puntos generados son estrictamente factibles.

A lo largo de los años, se han ideado un buen número de técnicas de optimización, y muchas de ellas pueden aplicarse directamente a problemas de Programación Cuadrática. Los métodos vistos anteriormente no sólo pueden ser directamente aplicados al caso de las SVM, sino que dada la estructura específica del problema, también pueden ser considerablemente simplificados. Conceptualmente, no son muy diferentes de la estrategia del ascenso de gradiente simple, pues todos escalan iterativamente la función objetivo hacia su máximo, aunque son probablemente más eficientes, esencialmente debido a que la dirección y la longitud de cada paso son escogidos en una forma más sofisticada.

### 4.1.2 Paquetes de Optimización Disponibles.

Una de las principales ventajas de las técnicas mencionadas es que están bien entendidas y ampliamente disponibles en un número de paquetes comerciales y de freeware, algunos también accesibles a través de Internet. Estos paquetes fueron utilizados para las SVM antes de que se desarrollaran algoritmos especialmente adaptados. Una de las opciones más comunes es el paquete MINOS, del Stanford Optimization Laboratory, el cual utiliza una estrategia híbrida; otra opción estándar es LOQO, el cual utiliza un método de punto interior primal-dual. Una solución muy conveniente es utilizar los paquetes de SVM existentes, como $SVM^{light}$ de Joachims, el paquete de Royal Holloway, University of London y otros disponibles gratuitamente a través de Internet.
Existen dos subrutinas de programación cuadrática proveídas en la caja de herramientas de optimización de MATLAB 5.3 y MATLAB 6:

**QP (Matlab 5.3).** Este es un método de conjunto activo (active set) también conocido como Método de Proyección, descrito en [GMW91]. Se resuelve un problema de programación lineal para determinar un punto factible inicial. Se inicia un proceso iterativo donde, en cada iteración se toma un paso desde la posición actual hacia la solución óptima. Cada uno de estos pasos se calcula de modo que se minimice el costo de la función, mientras permanece dentro de los límites de las restricciones activas. El conjunto de restricciones activo es actualizado en cada iteración.

**QuadProg (Matlab 5.3).** QuadProg localiza una solución local a menos que el problema sea estrictamente convexo. Si no se suministra un punto inicial o éste no es estrictamente factible, QuadProg escoge un nuevo punto inicial estrictamente factible.
La función utiliza dos algoritmos, dependiendo del problema. Cuando éste tiene solamente cotas superiores o inferiores (es decir, no se especifican desigualdades o igualdades lineales) o tiene solamente igualdades lineales (es decir, no se especifican cotas inferior y superior o desigualdades lineales), se utiliza el algoritmo de un método de gran escala. Este método es un método de un subespacio de la región de confianza, basado en el método Reflexivo de Newton, mencionado en la sección anterior. Cada iteración envuelve la solución aproximada de un sistema lineal grande, utilizando el método de gradientes conjugados precondicionados.
Para optimización de mediana escala, QuadProg utiliza un método de conjunto activo, el cual es también un método de proyección similar al de la función QP. El método encuentra una solución factible inicial resolviendo primero un problema de programación lineal.

A pesar de que estas soluciones computacionales son muy buenas, hay que pagar un costo alto debido al tamaño del problema. Muchas de ellas requieren que la matriz del kernel (Hessiano) se almacene en memoria, lo cual implica que la complejidad del espacio sea cuadrática en el tamaño de la muestra. Por otro lado, existe la ventaja de que el tamaño de dicha matriz depende únicamente del número de muestras, no importando que provengan de un espacio de entrada de baja dimensión, como puntos en el plano cartesiano, o de alta dimensión, como imágenes formadas por 784 pixeles.
Para problemas de tamaño grande, estos enfoques pueden ser ineficientes, por lo cual deben utilizarse en conjunción con las técnicas de descomposición descritas en la sección 4.2. En realidad es imposible describir todos los métodos existentes en este capítulo.

## 4.2 Métodos de Conjuntos de Trabajo.

Las SVM tienen la interesante propiedad de que tanto las funciones de entrenamiento como las de prueba dependen de los datos sólo a través de las funciones de kernel $k(x_i, x_j)$. Aunque corresponde a un producto punto en un espacio de dimensión $m$, donde $m$ puede ser muy grande o infinita, la complejidad del cálculo de $k$ puede estar lejos de ser pequeña. Este hecho es el que nos permite construir hiperplanos en estos espacios aún de muy altas dimensiones con una computación tratable. Así, las SVM evitan ambas formas de dimensionalidad: la proliferación de parámetros que causan una complejidad intratable, y la proliferación de parámetros que causan sobre ajuste.
Resolver un problema de Programación Cuadrática con un tamaño de algunos miles es un reto en términos de requerimientos de memoria y costo computacional. En problemas prácticos de clasificación de patrones, el número de puntos de entrenamiento puede exceder los 50,000; por lo tanto la duración del entrenamiento para conjuntos grandes de datos debe reducirse. Como el número esperado de vectores de soporte es una pequeña fracción del número total de puntos de entrenamiento, los métodos comunes fallan (el costo computacional y requerimiento de memoria son demasiado altos para usos prácticos) cuando el número de vectores de soporte excede unos poco miles. Esto puede pasar en los siguientes casos:

◊ La cantidad de datos de entrenamiento es muy grande, de modo que el número de vectores de soporte es una fracción muy pequeña del tamaño total de datos de entrenamiento, esto es aún demasiado grande para ser manejado por el algoritmo ordinario.

◊ Cuando los datos son altamente no separables, el error de generalización será alto. Como la razón entre el número de vectores de soporte y el número total de datos de entrenamiento es la cota superior del error esperado, podemos esperar que el número de vectores de soporte se incrementará para problemas altamente no separables.

Las técnicas como las descritas en la sección previa requieren que los datos sean mantenidos en memoria en forma de una matriz de kernel. La complejidad del problema de entrenamiento crece con el tamaño de esta matriz, limitando los enfoques a conjuntos de pocos miles de datos.

Parámetros del Algoritmo:

S: conjunto de entrenamiento;
B: working set;
$\alpha$: vector de multiplicadores de Lagrange.

Algoritmo:

$\alpha = 0$;
Seleccionar un working set arbitrario $B \subseteq S$;
Repetir
    Resolver el problema de optimización en B;
    Formar un nuevo working set B con los datos que no satisfacen las condiciones de Karush-Kuhn-Tucker;
Hasta que el criterio de paro sea satisfecho.
Devolver el vector $\alpha$.

*Figura 4.1. Forma básica del algoritmo de un Método de Conjunto de Trabajo.*

Para problemas mayores deseamos tomar las ventajas que forman las bases de los métodos de conjunto de trabajo (working set) en optimización: si uno supiera por adelantado cuales restricciones estuvieran activas, podría ser posible descartar todas las restricciones inactivas y simplificar el problema. Esto lleva a varias estrategias, todas basadas de alguna manera en conjeturas sobre el conjunto activo, y restringiendo el entrenamiento a dichas conjeturas. El punto importante es seleccionar el conjunto de trabajo de tal forma que la optimización del correspondiente subproblema de Programación Cuadrática lleve a una mejora en la totalidad de la función objetivo.

Se ha reportado que estos métodos trabajan muy bien en la práctica y hacen posible repartir con conjuntos de datos de varias decenas de miles de puntos [OFG97]. La figura 4.1 muestra el algoritmo general para estos métodos.

### 4.2.1 El Método de Chunking.

El método de Chunking (Pedazeo) se inicia entrenando una SVM a partir de un subconjunto pequeño de muestras elegidas arbitrariamente, llamado "chunk" ("pedazo"). Las demás muestras se prueban con el clasificador resultante y se eligen los $N_V$ puntos que más violen las condiciones de Karush-Kuhn-Tucker, junto con los $N_S$ vectores de soporte del pedazo entrenado, para formar un nuevo pedazo de tamaño $M = N_S + N_V$. Este proceso se realiza iterativamente, inicializando el vector $\alpha$ para cada nuevo subproblema con los valores de salida del anterior, hasta que todos los puntos satisfacen las condiciones KKT.

En cada iteración se utiliza un optimizador cualquiera para entrenar el pedazo. Dicho pedazo puede cambiar de tamaño en forma variable, dependiendo del valor de $N_V$ dado por el usuario. Generalmente, el pedazo crece (aunque puede decrecer) hasta que en la última iteración contiene a los vectores de soporte de todo el conjunto de entrenamiento que representan a las restricciones activas, lo cual hace que se cumpla la condición de salida. Es por eso que los problemas en que se aplique este método deben tener un número esperado de vectores de soporte bajo.

Nótese que los vectores de soporte finales pueden estar fuera de los pedazos que se van obteniendo en cada iteración, y los vectores de soporte en un pedazo particular pueden no aparecer en la solución final. La figura 4.2 muestra el algoritmo para este método.

Parámetros del Algoritmo:

S: conjunto de entrenamiento;
$B_i$: working set (chunk) en la iteración i;
$\alpha$: vector de multiplicadores de Lagrange;
$N_S$: número de vectores de soporte después de optimizar en el working set;
$N_V$: número de datos que se agregarán al working set;
i: número de iteración.

Algoritmo:

Seleccionar un working set arbitrario $B_0 \subseteq S$;
i = 0;
Repetir
    $\alpha$ = 0;
    Resolver el problema de optimización en $B_i$;
    Seleccionar los $N_V$ datos fuera de $B_i$ que menos satisfacen las condiciones de Karush-Kuhn-Tucker;
    Seleccionar los $N_S$ vectores de soporte de B;
    Formar un nuevo working set $B_{i+1}$ con los $N_S$ vectores de soporte y los $N_V$ datos fuera de $B_i$;
    i = i + 1;
Hasta que todos los datos satisfagan las condiciones de Karush-Kuhn-Tucker;
Devolver el vector $\alpha$.

*Figura 4.2. Forma básica del algoritmo del Método de Chunking.*

### 4.2.2 El Método de Descomposición.

El método anterior requiere que el número de vectores de soporte $N_S$ sea lo suficientemente pequeño para que un Hessiano de tamaño $(N_S + N_V) \times (N_S + N_V)$ $(M \times M)$ quepa en la memoria. Un algoritmo de conjunto de trabajo alternativo que supera esta limitación fue propuesto por Osuna, Freund y Girosi [OFG97]. En este algoritmo, como en el anterior, se entrena una pequeña porción de los datos de entrenamiento en cada iteración, con la diferencia de que el tamaño del conjunto de trabajo permanece fijo y pueden haber más vectores de soporte que datos en dicho conjunto.
Consiste en particionar el conjunto de entrenamiento en dos conjuntos $B$ y $N$, donde $B$ será el conjunto de trabajo de tamaño $M$. El vector $\alpha$ también se divide en $\alpha_B$ y $\alpha_N$, de modo que $\alpha_N$ permanecerá fijo y $\alpha_B$ cambiará en cada iteración. Así mismo, los datos $x$ ($x_B$, $x_N$) y las etiquetas $y$ ($y_B$, $y_N$). En cada iteración, se realiza un intercambio de datos entre los dos conjuntos, reemplazándose algunos valores de $B$ por otros de $N$, de acuerdo a alguna heurística determinada. El subproblema de optimización a resolver ahora es

$$\min_{\alpha_B} \quad -L_D(\alpha_B) = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_{Bi}\alpha_{Bj}y_{Bi}y_{Bj}k(x_{Bi},x_{Bj}) - \sum_{i=1}^{M}\alpha_{Bi} + \sum_{i=1}^{M}\alpha_{Bi}y_{Bi}\sum_{j=1}^{l-M}\alpha_{Nj}y_{Nj}k(x_{Bi},x_{Nj}) \quad (4.1)$$

sujeto a las restricciones

$$\sum_{i=1}^{M}\alpha_{Bi}y_{Bi} = -\sum_{i=1}^{l-M}\alpha_{Ni}y_{Ni} \quad (4.2)$$

$$\alpha_{Bi} \geq 0 \text{ para } i = 1,2,...,l\,. \quad (4.3)$$

Parámetros del Algoritmo:

S: conjunto de entrenamiento;
$B_i$: working set en la iteración i;
$N_i$: complemento del working set en la iteración i;
$\alpha$: vector de multiplicadores de Lagrange;
M: tamaño del working set;
$N_V$: número de datos que se agregarán al working set;
i: número de iteración.

Algoritmo:

$\alpha$ = 0;
Seleccionar un working set arbitrario $B_0 \subseteq S$;
i = 0;
Repetir
    Resolver el problema de optimización en $B_i$, manteniendo fijos los elementos de $\alpha$ que están en $N_i$;
    Seleccionar los $N_V$ datos en $N_i$ que menos satisfacen las condiciones de Karush-Kuhn-Tucker;
    Formar un nuevo working set $B_{i+1}$, intercambiando los $N_V$ datos de $N_i$ seleccionados con otro tanto dentro de $B_i$, y los multiplicadores de Lagrange correspondientes;
i = i + 1;
Hasta que todos los datos satisfagan las condiciones de Karush-Kuhn-Tucker;
Devolver el vector $\alpha$.

*Figura 4.3. Forma básica del algoritmo del Método de Descomposición.*

Como puede verse, el problema es independiente del número de vectores de soporte. Puede probarse que la función objetivo mejora en cada iteración; es decir, el algoritmo no se ciclará infinitamente. Por lo tanto, para un problema de programación cuadrática convexa, este algoritmo debe converger hacia el óptimo global en un número finito de iteraciones. La figura 4.3 muestra el algoritmo para este método.

### 4.2.3 El Método de Optimización Secuencial Mínima (SMO).

El algoritmo de Optimización Secuencial Mínima (Sequential Minimal Optimization) [Pla98] se deriva tomando la idea del método de descomposición hasta su extremo y optimizando un subconjunto mínimo de dos puntos en cada iteración. Fue propuesto por Platt [Pla98]. El poder de esta técnica reside en el hecho de que el problema de optimización para dos puntos permite una solución analítica, eliminando la necesidad de utilizar un optimizador de Programación Cuadrática iterativo como parte del algoritmo.

En cada paso, SMO escoge dos elementos $\alpha_i$ y $\alpha_j$, con $i \neq j$, para optimizarlos, encuentra los valores óptimos para estos dos parámetros dado que todos los otros están fijos, y actualiza el vector $\alpha$ de acuerdo a esto. La opción de los dos puntos es determinada por una heurística, mientras que la optimización de los dos multiplicadores es realizada analíticamente.

### 4.2.4 Posibles Mejoras a los Métodos.

Estos algoritmos de entrenamiento pueden aprovechar las ventajas del procesamiento en paralelo en varias formas. Primero, todos los elementos del Hessiano por si mismos pueden calcularse simultáneamente. Segundo, cada elemento a menudo requiere del cálculo de productos punto de los datos de entrenamiento, los cuales podrían ser paralelizados. Tercero, el cálculo de la función objetivo, o gradiente, el cual es un cuello de botella de la velocidad, puede ser paralelizado (requiere multiplicación de matrices). Finalmente, se puede prever la paralelización en un nivel superior, por ejemplo al entrenar en diferentes pedazos simultáneamente. Esquemas como éstos, combinados con los algoritmos de conjuntos de trabajo pueden hacer tratables a problemas muy grandes ($>> 100,000$ vectores de soporte).

## 4.3 El Método de Chunking Suave.

### 4.3.1 Las Condiciones KKT para las SVM.

Una buena manera de revisar que algún algoritmo trabaja es comprobando que la solución satisface todas las restricciones de Karush-Kuhn-Tucker para el problema primal, pues son condiciones necesarias y suficientes para que la solución sea óptima. En los métodos de conjuntos de trabajo, se realizan iteraciones en cada una de las cuales se construyen SVM, con el fin de probar si todos los datos cumplen con las condiciones KKT. A continuación encontraremos las condiciones KKT particulares para el entrenamiento de las SVM.

Las condiciones KKT para un problema de optimización no lineal están dadas por las ecuaciones (2.8) a (2.12). En el problema que se plantea resolver para las SVM, necesitamos

$$\min_{w,b} \quad f(w,b)=\frac{1}{2}w\cdot w \tag{4.4}$$

sujeto a las restricciones

$$g_i(w,b)=-y_i\left(w\cdot x_i+b\right)+1\le 0 \text{ para } i=1,2,...,l\,. \tag{4.5}$$

Ahora encontraremos las condiciones KKT particulares para este problema. Los gradientes de f y las $g_i$ son:

$$\nabla f=\left(\frac{\partial f}{\partial w},\frac{\partial f}{\partial b}\right)=(w,0) \tag{4.6}$$

$$\nabla g_i=\left(\frac{\partial g_i}{\partial w},\frac{\partial g_i}{\partial b}\right)=(-y_i x_i,-y_i) \tag{4.7}$$

Para cada elemento del vector $\nabla f$, aplicamos la primera condición:

$$\frac{\partial f}{\partial w}=-\sum_{i=1}^{l}\alpha_i^*\frac{\partial g_i}{\partial w} \quad\Rightarrow\quad w=-\sum_{i=1}^{l}\alpha_i^*(-y_i x_i) \quad\Rightarrow\quad w=\sum_{i=1}^{l}\alpha_i^* y_i x_i \tag{4.8}$$

$$\frac{\partial f}{\partial b}=-\sum_{i=1}^{l}\alpha_i^*\frac{\partial g_i}{\partial b} \quad\Rightarrow\quad 0=-\sum_{i=1}^{l}\alpha_i^* y_i \quad\Rightarrow\quad \sum_{i=1}^{l}\alpha_i^* y_i=0 \tag{4.9}$$

Este problema en particular está planteado solamente con restricciones de desigualdad. Por tanto consideramos las condiciones dadas por (2.9), (2.10) y (2.11), de la siguiente manera:

$$\alpha_i^*\ge 0 \text{ para } i=1,2,...,l\,, \tag{4.10}$$

$$\alpha_i^* g_i(w^*,b^*)=\alpha_i^*\left[-y_i\left(w^*\cdot x_i+b^*\right)+1\right]=0 \text{ para } i=1,2,...,l\,, \tag{4.11}$$

$$g_i(w^*,b^*)=-y_i\left(w^*\cdot x_i+b^*\right)+1\le 0 \text{ para } i=1,2,...,l\,. \tag{4.12}$$

Así, para verificar si una SVM entrenada cumple con las condiciones KKT, es suficiente con verificar que todas las $\alpha_i$ satisfacen las ecuaciones (4.9) a (4.12), de acuerdo a lo establecido en la sección 2.2.3.

Si tomamos en cuenta estas condiciones que debe cumplir cada uno de los multiplicadores de Lagrange de la solución de entrenamiento y nos concentramos en tres de ellas: (4.10), (4.11) y (4.12). Cuando termina un entrenamiento tenemos valores de $\alpha^*$ y, por consiguiente, de $w^*$ y $b^*$. Generalmente los valores del vector $\alpha^*$ encontrado cumplen con las condiciones establecidas por (4.9) y (4.10), pues son las condiciones del problema de programación cuadrática ya resuelto. Por lo tanto, nos resta verificar si para cada punto de entrenamiento los multiplicadores cumplen con (4.11) y (4.12). Para ello, hagamos $u(x_i)=w^*\cdot x_i+b^*$ y veamos a las ecuaciones (4.11) y (4.12) como

$$\alpha_i^*\left[-y_i u(x_i)+1\right]=0 \text{ para } i=1,2,...,l\,, \tag{4.13}$$

$$-y_i u(x_i)+1\le 0 \text{ para } i=1,2,...,l\,. \tag{4.14}$$

De acuerdo a (4.11), hay dos factores que pueden hacer 0 dicha igualdad: $\alpha_i$ y $-y_i u(x_i)+1$; y de acuerdo a (4.10), tenemos dos casos posibles para cada $x_i$ de un conjunto de entrenamiento: $\alpha_i^*=0$ y $\alpha_i^*>0$.

**Caso 1.** Cuando $\alpha_i^*=0$, entonces $-y_i u(x_i)+1$ puede o no ser distinto de 0. Pero como ese factor debe cumplir con (4.12), no puede ser positivo y, por lo tanto, al despejar adecuadamente tenemos que $y_i u(x_i)\ge 1$. El cero en el multiplicador nos dice que el vector de entrenamiento $x_i$ no es un vector de soporte y por lo tanto no cae en ninguno de los hiperplanos que acotan a éste; por lo tanto su valor no puede ser 1.
Así, para toda $\alpha_i^*=0$ debe cumplirse que $y_i u(x_i)>1$.

**Caso 2.** Cuando $\alpha_i^*>0$, entonces necesariamente $-y_i u(x_i)+1=0$. De esa manera, el factor cumple con (4.12) y, por lo tanto, al despejar adecuadamente tenemos que $y_i u(x_i)=1$. En este caso, el vector de entrenamiento $x_i$ sí es un vector de soporte.
Así, para toda $\alpha_i^*>0$ debe cumplirse que $y_i u(x_i)=1$.
Es claro que cualesquiera vectores que no hayan tomado parte de un conjunto de entrenamiento (pertenecientes a un conjunto de prueba) no tienen un valor de $\alpha$ determinado, por lo cual para ellos no se toman en cuenta las condiciones (4.8) a (4.11). Sin embargo, cuando al utilizar el método de Chunking queremos probar si un vector de entrenamiento cumple con las condiciones KKT, asumimos valores de $\alpha=0$ para los vectores que no fueron parte del pedazo; pero si el método es el de descomposición, entonces cada vector de entrenamiento, aunque en ese momento no forme parte del conjunto de trabajo, debe tener un valor de $\alpha$ asignado.

### 4.3.2 Planteamiento Formal del Método de Chunking Suave.

Cuando probamos un patrón $x_i$ en algún clasificador (SVM) tenemos varias posibles salidas, a cada una de la cual le corresponde una explicación:

1. Si $y_i u(x_i)>1$, el patrón es correctamente clasificado y no cae en el margen.
2. Si $y_i u(x_i)=1$, el patrón cumple exactamente la restricción (3.10). En este caso, $x_i$ cae sobre alguno de los hiperplanos que delimitan el margen, al igual que los vectores de soporte.
3. Si $0<y_i u(x_i)<1$, el patrón cae dentro del margen, pero aun permanece en el lado correcto del límite de decisión, con lo que la clasificación es correcta.
4. Si $y_i u(x_i)<0$, el patrón será clasificado incorrectamente.

Nótese que cuando un patrón de la clase 1 está correctamente clasificado, el valor de $u(x_i)$ es positivo o cero, y al multiplicarlo por la clase, $y_i u(x_i)$ será también positivo o cero. Por otra parte, cuando un patrón de la clase -1 está correctamente clasificado, el valor de $u(x_i)$ es negativo, y al multiplicarlo por la clase, $y_i u(x_i)$ será también positivo. Esto concuerda con lo visto en los dos casos en los que se cumplen las condiciones KKT de la sección anterior.

Hemos visto que, tanto en el método de Chunking como en el de Descomposición, ciertas cantidades de vectores de entrenamiento entran y salen del conjunto de trabajo, de acuerdo a si cumplen o no las condiciones descritas anteriormente. Generalmente, y de acuerdo a como haya sido elegido el primer conjunto de trabajo, nuestra primera región de decisión clasificará incorrectamente una cantidad proporcionalmente alta de vectores de entrenamiento. Conforme vayan sucediéndose las iteraciones, dicha proporción disminuirá hasta que en la última iteración haya cero intercambios entre el conjunto de trabajo y los vectores restantes.

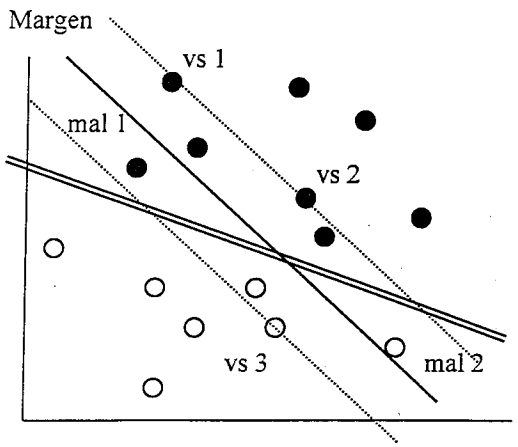


*Figura 4.4. Un caso del estado de un método de conjunto de trabajo entre iteraciones.*

El método propuesto, llamado Método de Chunking Suave consiste en llegar a una situación en la que todos los vectores están bien clasificados, no importando que algunos estén dentro del margen (los cuales no cumplen con las condiciones KKT). A esta etapa se le llama Fase Suave. Una vez ahí, de manera opcional puede crearse un conjunto de trabajo formado por los vectores de soporte de la última iteración y todos los vectores que quedaron dentro del margen, y con esos datos obtener una solución final, efectuando lo que llamamos Iteración Adicional. La idea es que el usuario del método elija que desea hacer pues, como veremos mas adelante, la generalización proporcionada por ambas alternativas es bastante buena.

Para explicar lo que sucede en la primera fase consideremos la figura 4.4. Supongamos que corre la iteración *i* del método de Chunking. Podemos observar que hay varios patrones que no cumplen las condiciones KKT como se describieron anteriormente. Tres de ellos están bien clasificados pero dentro del margen y dos más están mal clasificados.

De acuerdo a los criterios del Chunking, tenemos 5 vectores que no cumplen las condiciones KKT y esto significa que tenemos que agregar estos 5 vectores al conjunto de trabajo (pedazo). La primera diferencia entre éste método y el propuesto es que ahora, en cada iteración de la primera fase, toleraremos a los vectores que quedan bien clasificados aunque estén dentro del margen.

Para el caso de la figura 4.4, solamente tendremos que agregar al conjunto de trabajo los dos vectores mal clasificados. Esta sencilla modificación permite al método efectuar una cantidad menor de cambios por iteración y tener una cantidad menor de vectores en cada conjunto de trabajo, por lo cual se ahorrará tiempo de entrenamiento. Al final de esta fase tendremos la situación deseada: todos los vectores estarán bien clasificados.

Lo anterior significa que para toda $\alpha_i^* = 0$, es suficiente que se cumpla que $y_i u(x_i) \geq 0$; que es lo que se verificara para todos los vectores fuera del conjunto de trabajo en cada iteración, al momento de seleccionar a los vectores entrantes. Como en el método de Chunking, la selección de los que entran se hace a través de la cantidad de violación a las condiciones KKT, en este caso modificadas.

- La iteración adicional se efectúa una vez que se termina la primera fase del método, con un conjunto de trabajo formado por los vectores de soporte de la última iteración de la fase suave y todos los vectores que al final quedaron dentro del margen. En esta iteración se aplica todo el rigor de las condiciones KKT, no permitiéndose ya que queden vectores dentro del margen.

Este último entrenamiento se hace como si el conjunto de trabajo estuviera formado por todos los vectores de entrenamiento. De hecho, entrenar a este conjunto de trabajo equivale a entrenar a todo el conjunto de entrenamiento, como se verá enseguida.

Es bien sabido que si eliminásemos los vectores afuera del margen (los que no son vectores de soporte) y entrenáramos de nuevo, la región de decisión sería exactamente la misma que al utilizar todos los datos. En este caso, se utilizarán los vectores de soporte junto con los datos que están dentro del margen para entrenar la SVM final. El siguiente razonamiento nos lleva a demostrar que esta SVM final clasifica correctamente a todos los datos del conjunto de entrenamiento (en el conjunto de trabajo).

***Prueba de que la iteración adicional resuelve la SVM para el conjunto de entrenamiento.***

**Definición 4.1.** Dados un conjunto de vectores $x_i \in A$, si $u(x_i) = w \cdot x_i + b$ es una máquina de soporte vectorial entrenada a partir de ellos, entonces la máquina clasifica adecuadamente a todos los vectores y éstos cumplen con la desigualdad $y_i u(x_i) \geq 1$.

**Definición 4.2.** Si todos los elementos de un conjunto de vectores $x_i \in A$ satisfacen la desigualdad $y_i u(x_i) \geq 1$, donde $u(x_i) = w \cdot x_i + b$ es una máquina de soporte vectorial, entonces la máquina clasifica adecuadamente a todo el conjunto.

**Teorema 4.1.** Dados una máquina de soporte vectorial $u(x) = w \cdot x + b$ entrenada a partir de un conjunto de vectores $x_A \in A$ y un conjunto de vectores $x_B \in B$ en el margen de $u(x)$, es decir, que cumplen con la desigualdad $0 < y_B u(x_B) < 1$ entonces, la máquina de soporte vectorial $v(x) = w_1 \cdot x + b_1$ entrenada a partir de los vectores de soporte de $u(x)$ y los vectores del conjunto $B$, clasifica adecuadamente también los vectores del conjunto $A$ que no son vectores de soporte.

***Prueba:*** Sea el conjunto $C$ compuesto por los vectores de soporte del conjunto $A$ y todos los vectores del conjunto $B$. Como los vectores de soporte del conjunto $A$ cumplen con la igualdad $y_A u(x_A) = 1$, y los vectores del conjunto $B$ cumplen con $0 < y_B u(x_B) < 1$; entonces tenemos que los vectores del conjunto $C$ cumplen con la desigualdad $y_C u(x_C) \leq 1$.
Debido a que la máquina de soporte vectorial $v(x)$ fue obtenida al entrenar con el conjunto $C$, entonces todos los elementos del conjunto $C$ cumplen con la desigualdad $y_C v(x_C) \geq 1$.
Despejando en estas dos desigualdades, tenemos que para cada elemento de $C$ es cierto que $v(x_C) \geq \frac{1}{y_C}$ y que $u(x_C) \leq \frac{1}{y_C}$; y por la ley transitiva tenemos que $u(x_C) \leq \frac{1}{y_C} \leq v(x_C)$; por consiguiente $u(x_C) \leq v(x_C)$. Esto significa que para cualquier punto $x$ se cumplirá que $u(x) \leq v(x)$.
Consideremos ahora los vectores del conjunto $A$ que están fuera del margen; es decir, los que cumplen con la desigualdad $y_A u(x_A) > 1$, o sea que para esos vectores $u(x_A) > \frac{1}{y_A}$.
Además, como ya vimos $u(x_A) \leq v(x_A)$; por lo tanto, por la ley transitiva se cumple que $\frac{1}{y_A} < u(x_A) \leq v(x_A)$, y por consiguiente $\frac{1}{y_A} < v(x_A)$. Despejando nos da que para esos vectores se cumple que $y_A v(x_A) > 1$; es decir, $v(x)$ clasifica adecuadamente a dichos vectores que, además, están fuera de su margen.
Toda esta explicación se ha hecho en base a un caso linealmente separable, pues este razonamiento se cumple también al utilizar cualquier kernel no lineal, pues en realidad se trabaja con datos linealmente separables en algún espacio de rasgos $R^m$.
Sin embargo, en la practica esto no se cumple en algunas ocasiones; es decir, no siempre producirá el margen óptimo para todo el conjunto de entrenamiento, pues algunos puntos fuera del conjunto de trabajo pueden no quedar bien clasificados. Esto depende de la diferencia en inclinación (ángulo) entre el hiperplano obtenido por la fase suave y el hiperplano de máximo margen, y de las posiciones que tengan los vectores mas cercanos al margen de la fase suave. En la tabla 4.12 se discute sobre el tamaño del margen en una aplicación y se explica porqué el Chunking Suave es una opción posible, sobre todo con márgenes pequeños.

### 4.3.3 El Algoritmo de Chunking Suave y su Implementación.

A continuación describiremos el algoritmo para el Método de Chunking Suave. Dicho método esta basado en el de Chunking, por lo que cada vez que entrena con un pedazo de los vectores de entrenamiento, considera como cero a los multiplicadores de Lagrange del resto. Esta implementación se hizo en un solo programa para ambos métodos, de modo que la única diferencia entre ellos es el rigor con el que se toman las condiciones KKT.

**Parámetros.** Es necesario determinar algunos parámetros del algoritmo. El primero de ellos es el tamaño del conjunto de trabajo inicial. Esto depende del tamaño del problema y del número esperado de vectores de soporte para ese problema. A continuación debemos saber en que cantidad se le permitirá crecer al conjunto de trabajo, cada vez que sea insuficiente, como se vera mas adelante.

**Selección del Conjunto de Trabajo Inicial.** Originalmente se utilizaba el azar para elegir a los vectores de entrenamiento que conformarían el conjunto de trabajo inicial. Cuando el número de vectores de cada clase son muy parecidos, esto no supone problema alguno. Pero si una clase tiene un número de vectores proporcionalmente menor que la otra, se corre el riesgo de que muchos vectores, si no es que todos sean de una misma clase, con lo que podrían obtenerse resultados muy malos en las primeras iteraciones que hicieran muy lento al algoritmo. Existen algunas técnicas de selección que intentan encontrar un mejor conjunto de trabajo inicial con el fin de acelerar aún más los métodos, como los mencionados en [ROU99] o el Algoritmo Genético, como se vio en el Capítulo 3.
Para el caso nuestro, se construye el conjunto de trabajo de manera aleatoria, pero estableciendo un balance en el numero de vectores de cada clase; es decir, se van eligiendo uno de cada clase de manera intercalada hasta completar el conjunto de trabajo. Si acaso la cantidad de vectores de una clase es menor a la mitad del tamaño del conjunto de trabajo inicial, entonces todos ellos serán incluidos en él.

**Primera Fase (Suave).** En esta fase, lo que sigue se repetirá mientras existan vectores candidatos a entrar al conjunto de trabajo; es decir, hasta que no existan vectores fuera del conjunto de trabajo que estén mal clasificados.
Se resuelve el problema de Programación Cuadrática para los elementos del conjunto de trabajo, se extraen los vectores de soporte y se obtienen el sesgo.
Se calculan los valores de $y_i u(x_i)$ para todos los vectores del conjunto de entrenamiento, y se hacen dos listas, ordenadas de acuerdo a los valores recién calculados: una para los vectores que violan la condición de no estar bien clasificados (candidatos a entrar al conjunto de trabajo), y otra para los que no fueron vectores de soporte en el conjunto de trabajo (candidatos a salir).
Se determina un número de intercambios entre los dos conjuntos, de acuerdo a la menor de las cantidades de vectores entrantes y salientes.
En caso de haber cambios posibles, estos se efectúan intercambiando al vector de afuera que mas viola la condición con el no vector de soporte de adentro que este mas alejado del margen, y así sucesivamente con los segundos, terceros, etc., de cada lista.
En caso de no haber candidatos a salir pero si a entrar, o no haber obtenido vectores de soporte, se incrementa el tamaño del conjunto de trabajo de acuerdo al parámetro ya establecido, y esos lugares se ocupan con los vectores candidatos a entrar con mayor violación a la condición.
Si hubieron cambios o un incremento para el conjunto de trabajo, se pasa a la siguiente iteración con el nuevo conjunto de trabajo.
En caso de no haber candidatos a entrar, no importa si hay o no candidatos a salir, la fase iterativa termina.

**Segunda Fase (Iteración Adicional).** Se calculan los valores de $y_i u(x_i)$ para todos los vectores del conjunto de entrenamiento que no estaban en el conjunto de trabajo, y se hace una lista para los vectores que quedan dentro del margen de la última región de decisión calculada. Se reúnen estos datos con los vectores de soporte en el conjunto de trabajo para formar un conjunto de trabajo.

Finalmente, se resuelve el problema de Programación Cuadrática para los elementos del conjunto de trabajo, se extraen los vectores de soporte y se obtienen el sesgo.

Esta última región de decisión obtenida es la solución para todo el conjunto de entrenamiento. La figura 4.5 muestra el algoritmo del método.

Parámetros del Algoritmo:

S: conjunto de entrenamiento;
$B_i$: working set en la iteración i;
$N_i$: complemento del working set en la iteración i;
$\alpha$: vector de multiplicadores de Lagrange;
M: tamaño del working set;
$N_S$: número de vectores de soporte después de optimizar en el working set;
$N_V$: número de datos que se agregarán al working set;
$N_M$: número de datos que están bien clasificados pero dentro del margen;
i: número de iteración.

Algoritmo:

$\alpha = 0$;
Seleccionar un working set arbitrario $B_0 \subseteq S$;
$i = 0$;
Repetir
    Resolver el problema de optimización en $B_i$, manteniendo fijos los elementos de $\alpha$
    Que están en $N_i$;
    Seleccionar los $N_V$ datos en $N_i$ que menos satisfacen las condiciones de
    Karush-Kuhn-Tucker;
    Formar un nuevo working set $B_{i+1}$, intercambiando los $N_V$ datos de $N_i$ seleccionados
    Con otro tanto dentro de $B_i$, y los multiplicadores de Lagrange correspondientes;
$i = i + 1$;
Hasta que todos los datos estén bien clasificados, aunque no todos satisfagan las
Formar un nuevo working set $B_{i+1}$ con los $N_S$ vectores de soporte y los $N_M$ datos que estén dentro del margen de dichos vectores de soporte;
Resolver el problema de optimización en $B_{i+1}$;
Devolver el vector $\alpha$.

*Figura 4.5. Forma básica del algoritmo del Método de Chunking Suave.*

### 4.3.4 Visualización Gráfica de su Funcionamiento.

**Ejemplo 4.1.**

Para entender mejor lo que sucede tanto con el método de Chunking como con el de Chunking Suave, comparemos los resultados que se obtuvieron al correr los algoritmos de ambos métodos con los mismos datos que utilizamos en el ejemplo 3.3, y que se muestran en la tabla 3.4. El kernel utilizado fue lineal. En ambos métodos se utilizaron conjuntos de trabajo iniciales de tamaño 5 con incrementos de tamaño 1.

En las figuras que veremos a continuación, las gráficas de la izquierda corresponden al método de Chunking Suave y las de la derecha al de Chunking original. Veamos primero la figura 4.6. Como la forma de escoger el conjunto de trabajo inicial es la misma, podemos ver que las regiones de decisión son exactamente iguales.
Nos damos cuenta, de acuerdo a las características particulares de cada método, que esa situación es incorrecta para el Chunking, pero no para el Chunking Suave, pues podemos ver que todos los vectores están bien clasificados, aunque no todos cumplen con las restricciones KKT.

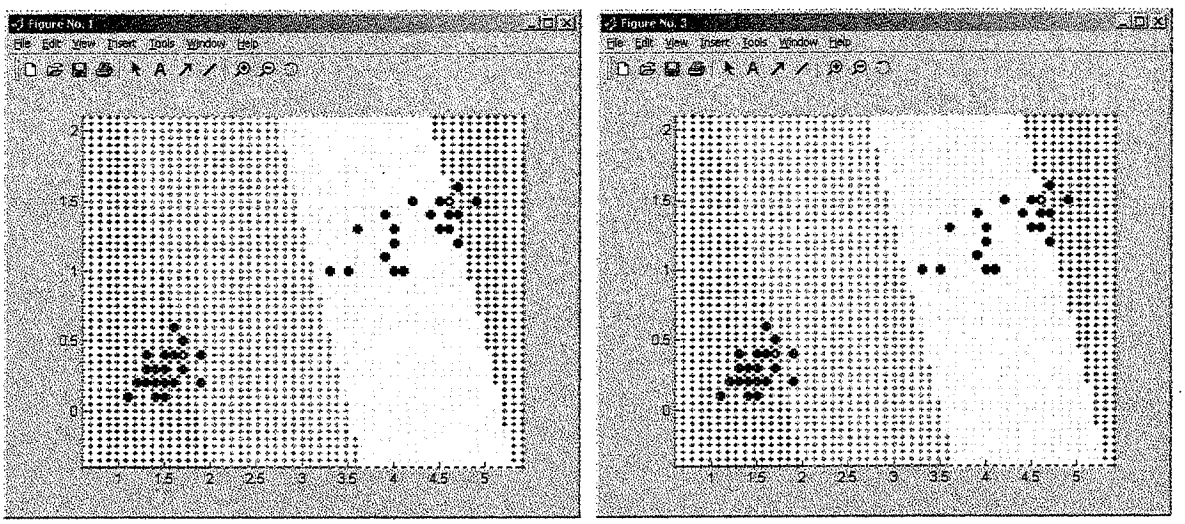

*Figura 4.6. Regiones de decisión producidas por la primera iteración de los algoritmos de los métodos de Chunking y de Chunking Suave en el ejemplo 4.1.*

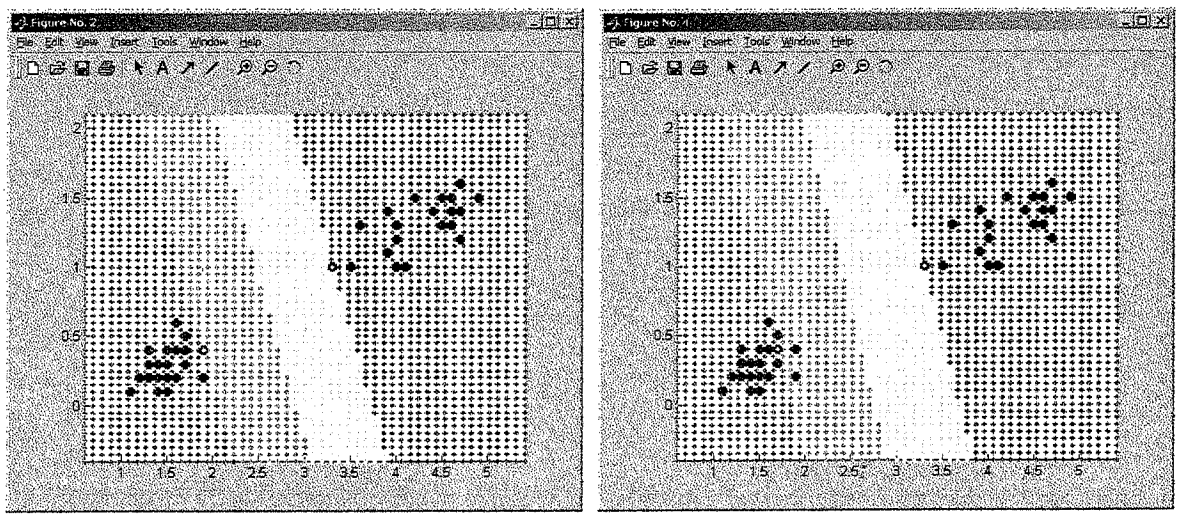

*Figura 4.7. Regiones de decisión producidas por la segunda iteración del algoritmo del método de Chunking y por la iteración adiconal del método de Chunking Suave en el ejemplo 4.1.*

De ese modo, el método de Chunking necesita seguir iterando para llegar a la solución, mientras que el de Chunking Suave podría terminar ahí. Para este ejemplo se ejecutó el segundo método para que corriera la iteración adicional, por lo que podemos comparar este resultado con la segunda iteración del Chunking.
En la figura 4.7, se observa que el método de Chunking Suave ha terminado, puesto que realizó la iteración adicional con todos los vectores del margen, lo que produjo como salida el margen óptimo. Mientras tanto, el método de Chunking hizo intercambios de vectores, tomando los peores de la clase -1 y al encontrar un nuevo margen, aún le quedaron datos de la clase 1 "malos", es decir dentro del margen, aunque bien clasificados.
Obviamente el Chunking toma los vectores de clase 1 mas cercanos al hiperplano encontrado y al recalcular se llega al margen óptimo, como puede verse en la figura 4.8.

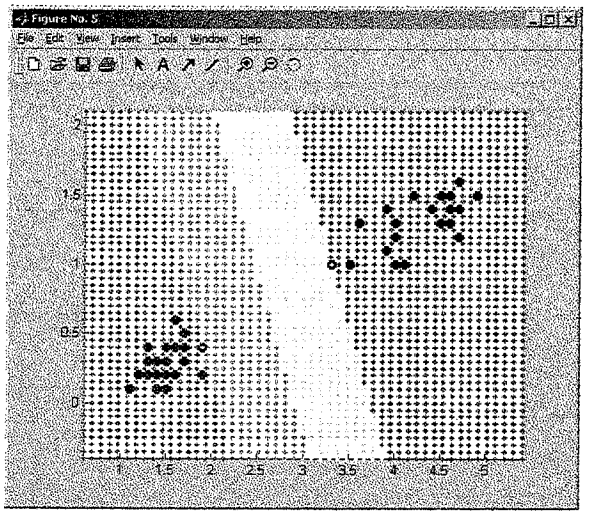


*Figura 4.8. Región de decisión producidas por la última iteración del algoritmo del método de Chunking en el ejemplo 4.1.*

### 4.3.5 Resultados Experimentales.

Se ha probado este nuevo método utilizando la base de datos de caracteres manuscritos MNIST (ver Apéndice A), que consta de imágenes de $28 \times 28$ en tonos de gris, lo que nos da vectores de 784 elementos enteros de 0 a 255; es decir, nuestro espacio de entrada es $R^{784}$. Los experimentos que arrojaron los experimentos aquí mostrados fueron hechos para el problema de separar el dígito 0 de los demás. Se han hecho pruebas con dos diferentes cantidades de datos de entrenamiento: 250 y 500; y se ha calculado la generalización para 4500 datos de prueba.

**Ejemplo 4.2.**
Primeramente veremos el caso del entrenamiento de SVM utilizando 25 muestras de cada dígito, lo que nos da 250 patrones de entrenamiento. Se encontraron las regiones de decisión para cada enfoque y se tomaron algunos datos resultantes. Las tablas 4.1 a 4.5 nos muestran en forma comparativa dichos resultados. Cabe señalar que el tiempo que aparece en el renglón de la iteración adicional del método de Chunking Suave es exclusivo de la iteración, efectuada al final de la primera parte del método.

Cuando se menciona a la función QuadProg, nos referimos a que se entrenó una SVM con todos los vectores utilizando dicha función. Cabe destacar que para entrenar las SVM en cada iteración de los metodos de descomposición se utilizó tambien QuadProg.

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 99.00 | 96.29 | 29.31 | 166 (66.40%) |
| Chunking | 5 | 99.00 | 96.20 | 57.81 | 104 (41.60%) |
| Chunking Suave | 2 | 99.20 | 96.24 | 23.20 | 96 (38.40%) |
| I. Adicional | 1 | 99.00 | 96.22 | 18.02 | 147 (58.80%) |

*Tabla 4.1. Comparación de resultados de los tres métodos con un kernel polinomial de grado 3 con 250 muestras, para el ejemplo 4.2.*

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 98.80 | 95.13 | 32.95 | 179 (71.60%) |
| Chunking | 4 | 98.80 | 95.13 | 51.11 | 114 (45.60%) |
| Chunking Suave | 2 | 98.80 | 95.22 | 23.23 | 99 (39.60%) |
| I. Adicional | 1 | 99.00 | 95.20 | 17.52 | 165 (66.00%) |

*Tabla 4.2. Comparación de resultados de los tres métodos con un kernel polinomial de grado 4 con 250 muestras, para el ejemplo 4.2.*

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 98.60 | 94.58 | 31.09 | 190 (76.00%) |
| Chunking | 9 | 98.60 | 94.64 | 135.14 | 130 (52.00%) |
| Chunking Suave | 2 | 98.40 | 94.71 | 22.81 | 103 (41.20%) |
| I. Adicional | 1 | 98.60 | 94.64 | 21.27 | 178 (71.20%) |

*Tabla 4.3. Comparación de resultados de los tres métodos con un kernel polinomial de grado 5 con 250 muestras, para el ejemplo 4.2.*

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 98.40 | 94.07 | 29.30 | 191 (76.40%) |
| Chunking | 8 | 98.40 | 94.11 | 222.27 | 146 (58.40%) |
| Chunking Suave | 2 | 98.40 | 94.13 | 22.78 | 112 (44.80%) |
| I. Adicional | 1 | 98.40 | 94.11 | 18.86 | 182 (72.80%) |

*Tabla 4.4. Comparación de resultados de los tres métodos con un kernel polinomial de grado 6 con 250 muestras, para el ejemplo 4.2.*

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 98.00 | 93.64 | 25.16 | 197 (78.80%) |
| Chunking | 12 | 98.20 | 93.76 | 288.34 | 155 (62.00%) |
| Chunking Suave | 2 | 98.00 | 93.71 | 23.00 | 110 (44.00%) |
| I. Adicional | 1 | 98.40 | 93.84 | 15.66 | 196 (78.40%) |

*Tabla 4.5. Comparación de resultados de los tres métodos con un kernel polinomial de grado 7 con 250 muestras, para el ejemplo 4.2.*

Podemos observar en estos resultados la manera en que la velocidad del Chunking Suave supera a la del Chunking original. El porcentaje de ahorro es superior al 50% en todos los casos, e incluso hay algunos en los que ese porcentaje es mucho mayor. Aún si sumamos el tiempo de la iteración adicional, la ventaja sigue siendo del Chunking Suave. Ya si comparamos con el enfoque de resolver el problema con todos los datos (QuadProg), las diferencias se reducen y la iteración adicional provoca que el Chunking Suave sea mas lento que QuadProg. En este caso, los resultados del Chunking Suave son tan buenos que es aceptable utilizarlo para ahorrar tiempo de entrenamiento.

En cuanto a los vectores de soporte, tal parece que su cantidad no es relevante para obtener buena generalización. Lo que sucede es que los multiplicadores de Lagrange resultan ser tan pequeños que muchos pasan por vectores de soporte, aunque su valor pequeño sea mas tendiendo a cero. Sería muy difícil encontrar una cota que distinguiera unos de otros a esa escala.

Resulta interesante observar las tendencias en el número de vectores de soporte que se obtienen en cada metodo. Al utilizar QuadProg y tener a todo el conjunto de entrenamiento, se obtiene un porcentaje alto de vectores de soporte, mientras que con los metodos de Chunking (original y suave), esas cantidades disminuyen. Podemos observar que la iteración adicional aproxima esta cantidad a la de la función QuadProg, debido a que toma una buena cantidad de vectores que quedaron dentro del margen del Chunking Suave. Esto se debe a que el tamaño del margen es muy pequeño, como se discutira mas adelante. Otra situación es que el número de vectores de soporte aumenta conforme aumenta el grado del kernel polinomial y los porcentajes de estos sobre el total de vectores de entrenamiento aumenta de maneras similares.

No solamente la velocidad de entrenamiento y el número de vectores de soporte encontrados son factores a favor del Chunking Suave, sino la capacidad de generalización que se obtiene. Comparando con los otros métodos, resulta ser casi la misma. Incluso al efectuar la iteración adicional lo que se gana no es significativo. Puede decirse que la región de decisión del Chunking Suave es tan buena como las de las SVM convencionales, con la gran ventaja de la rapidez con que se obtienen.

Finalmente, podemos ver que el kernel de grado 3 es el que nos da una mejor generalización por sobre los otros. Conforme aumenta el grado del polinomio, la generalización empeora, al tiempo que el número de vectores de soporte disminuye. Esto es una clara muestra que el kernel que tiene el balance entre minimización del riesgo y capacidad de generalización es el polinomial de grado 3, para este problema.

**Ejemplo 4.3.**
Ahora nos ocuparemos del caso del entrenamiento de SVM utilizando 50 muestras de cada dígito, lo que nos da 500 patrones de entrenamiento. Como en el ejemplo anterior, se encontraron las regiones de decisión para cada enfoque y se tomaron algunos datos resultantes. Las tablas 4.6 a 4.8 nos muestran en forma comparativa dichos resultados.

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 100.00 | 96.36 | 350.11 | 330 (66.00%) |
| Chunking | 3 | 100.00 | 96.36 | 256.26 | 203 (40.60%) |
| Chunking Suave | 3 | 100.00 | 96.51 | 229.38 | 181 (36.20%) |
| I. Adicional | 1 | 100.00 | 96.51 | 192.62 | 261 (52.20%) |

*Tabla 4.6. Comparación de resultados de los tres métodos con un kernel polinomial de grado 4 con 500 muestras, para el ejemplo 4.3.*

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 100.00 | 95.73 | 365.80 | 333 (66.60%) |
| Chunking | 7 | 100.00 | 95.87 | 546.30 | 206 (41.20%) |
| Chunking Suave | 3 | 100.00 | 95.80 | 223.44 | 188 (37.60%) |
| I. Adicional | 1 | 100.00 | 95.93 | 156.33 | 280 (56.00%) |

*Tabla 4.7. Comparación de resultados de los tres métodos con un kernel polinomial de grado 5 con 500 muestras, para el ejemplo 4.3.*

| *Algoritmo* | *Número de Iteraciones* | *% Correctos Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* | *Núm. VS (%)* |
|---|---|---|---|---|---|
| QuadProg | 1 | 100.00 | 95.29 | 372.72 | 361 (72.20%) |
| Chunking | 7 | 100.00 | 95.44 | 591.83 | 241 (48.20%) |
| Chunking Suave | 3 | 100.00 | 95.00 | 202.91 | 181 (36.20%) |
| I. Adicional | 1 | 100.00 | 95.44 | 137.37 | 308 (61.60%) |

*Tabla 4.8. Comparación de resultados de los tres métodos con un kernel polinomial de grado 6 con 500 muestras, para el ejemplo 4.3.*

Ahora la diferencia en velocidades se ha acentuado. Obviamente, al ser este un problema mayor, no serán iguales los 25 segundos de diferencia en el ejemplo anterior con los 300 de este caso. Esto se cumple a excepción del polinomio de grado 4, donde la diferencia fue pequeña. El porcentaje de ahorro sigue siendo superior al 50%. Aún si sumamos el tiempo de la iteración adicional, la ventaja sigue siendo del Chunking Suave. La diferencia con el enfoque total es mayor debido a la escala del problema. Las diferencias con el total, aún con la iteración adicional son mucho menores en porcentaje y en ocasiones gana un método y en otras el otro.

La capacidad de generalización, como era de esperarse, ha mejorado debido a la cantidad de vectores de entrenamiento. Al compararlas resultan parecerse al igual que con 250 muestras. Esto nos confirma las bondades del Chunking Suave.
En cuanto a los vectores de soporte, ocurre el mismo fenómeno que en el ejemplo 4.2. Es posible notar algo curioso en los porcentajes de estos arrojados por cada método, pues son muy parecidos y podemos ver como aumentan de número (y porcentaje) conforme el kernel deja de ser adecuado y la generalización disminuye. Este ejemplo también nos confirma que el kernel de grado 3 es el mejor para separar al 0 de los demás dígitos.
En este punto podemos esperar que cuanto mayor sea el problema, la diferencia de velocidad entre el Chunking y el Chunking suave se incrementara. También habrá ventaja de este último con el entrenamiento con todos los vectores, pero en un momento dado ya no se podría por el tamaño del problema que crearía conflictos en la memoria de la computadora, problema que no tendrían los Chunking.

**Ejemplo 4.4.**
En este ejemplo compararemos el rendimiento de las SVM contra el de las Redes Neuronales, que resulta ser una comparación lógica por la naturaleza de ambas. Para esto se entrenaron tres Redes Neuronales de distintas configuraciones con el mismo conjunto de 250 datos utilizado para el ejemplo 4.2. Se utilizó el programa que viene en los apendices de [Pao89]. En la tabla 4.9 se muestran los resultados obtenidos por los entrenamientos de las redes y la tabla 4.10 compara la generalización obtenida por estas en comparación con las SVM.

**Redes Neuronales.**

| *Tipo de Red* | *Error del Sistema* | *Máximo Error de Entrenamiento* | *% Correctos Prueba* | *Tiempo (Segundos)* |
|---|---|---|---|---|
| 60-60 | 0.003620 | 0.256418 | 91.89 | 71 |
| 60-54 | 0.000099 | 0.000266 | 85.47 | 112 |
| 54-54 | 0.001476 | 0.002349 | 88.62 | 18 |

*Tabla 4.9. Comparación de resultados de tres Redes Neuronales con 250 muestras, para el ejemplo 4.4.*

En esta comparación podemos observar tres redes con dos capas ocultas, cada una. El número de neuronas en cada capa hace la diferencia. La generalización es buena, pero no tanto como la que dan las SVM. Para que una RN aproxime a la capacidad de un kernel dado, es necesario buscar por medio de ensayo los parámetros adecuados para dicha red (número de capas ocultas, número de neuronas por capa, etc.), que incrementen esta generalización. Podemos ver las notables diferencias entre las generalizaciones producidas por ambos enfoques. La tabla 4.10 nos revela cómo la mejor red no se acerca a la peor SVM de estos ejemplos.

| *Enfoque* | *% Prueba* | *Enfoque* | *% Prueba* |
|---|---|---|---|
| SVM Polinomial^3 | 96.29 | SVM Polinomial^7 | 93.64 |
| SVM Polinomial^4 | 95.13 | NN 60-60 | 91.89 |
| SVM Polinomial^5 | 94.58 | NN 54-54 | 88.62 |
| SVM Polinomial^6 | 94.07 | NN 60-54 | 85.47 |

*Tabla 4.10. Comparación de la generalización obtenida de varias SVM con la obtenida con Redes Neuronales, todas con 250 muestras.*

Para terminar este capítulo analizaremos un poco lo que sucede con los margenes de las SVM en los ejemplos hechos con los dígitos manuscritos. Con esto nos explicamos porque el Chunking Suave es tan "bueno" como el Chunking original. Nos hemos dado cuenta de que a pesar de las cantidades distintas de vectores de soporte arrojadas por los métodos con distintos vectores de entrenamiento y distintos tipos de kerneles, los resultados resultan ser muy similares. Esto puede explicarse por la cercanía que hay entre los vectores de cada clase entre sí, y de las distintas clases una de la otra, en el espacio de rasgos. Para confirmar esto medimos los márgenes óptimos obtenidos al entrenar los ejemplos con el algoritmo total. Los resultados se pueden ver en las tablas 4.11 y 4.12.

| *Kernel* | **Polinomial de Grado 3** | **Polinomial de Grado 4** | **Polinomial de Grado 5** | **Polinomial de Grado 6** | **Polinomial de Grado 7** |
|---|---|---|---|---|---|
| *Margen* | $2.0262\times10^{-19}$ | $5.0501\times10^{-26}$ | $1.3834\times10^{-32}$ | $3.9758\times10^{-39}$ | $1.1747\times10^{-45}$ |

*Tabla 4.11. Comparación de los anchos de los márgenes de cinco SVM entrenadas con 250 patrones, del ejemplo 4.2.*

| *Kernel* | **Polinomial de Grado 4** | **Polinomial de Grado 5** | **Polinomial de Grado 6** |
|---|---|---|---|
| *Margen* | $1.485\times10^{-25}$ | $4.2029\times10^{-32}$ | $1.2878\times10^{-38}$ |

*Tabla 4.12. Comparación de los anchos de los márgenes de tres SVM entrenadas con 500 patrones, del ejemplo 4.3.*

Es claro que los márgenes son tan pequeños que los tres métodos producen respuestas muy similares, pues las tres regiones de decisión pasan demasiado cerca una de la otra. Además, es muy probable que compartan buena parte de sus trayectorias en el espacio de entrada, lo que significa que los ángulos entre los hiperplanos de separación obtenidos por cada uno de ellos son muy pequeños en el espacio de rasgos.
Esta pequeñez del margen hace demasiado parecidos a la región de decisión con todos los puntos bien clasificados y al margen óptimo. Las diferencias son ínfimas y esto nulifica al margen óptimo, que se pierde en un mar de vectores cuyas clases tienen una frontera casi imperceptible.

# CAPITULO V. CONCLUSIONES.

## 5.1 Análisis Final sobre las Aportaciones del Trabajo.

Recapitulando sobre todo lo visto y analizado, podemos destacar el presente trabajo las siguientes aportaciones:

- Se propuso una variante del método Chunking con SVM para Clasificación de Patrones, al cual se le llamó Chunking Suave. Dicho método es iterativo y se basa en el entrenamiento de una o más SVM sobre pedazos de un conjunto grande de datos para encontrar una región decisión que clasifica con una generalización muy buena, comparable con el entrenamiento de una SVM con el total del conjunto de vectores. El método tiene la ventaja de ser muchísimo más rápido.
- Se analizó el enfoque de Algoritmos Genéticos como alternativa para la solución del problema de optimización derivado de las SVM, sentando las bases para un análisis más a fondo de esta opción. Si bien el AG no resulta muy adecuado para entrenar una SVM debido a su lentitud y a lo difícil que es encontrar el factor de penalización y el número de iteraciones adecuados para cada caso, es una muy buena opción para detectar un conjunto de los vectores de entrenamiento que contenga a los vectores de soporte, por lo que bien pudiera ser utilizado como técnica de preproceso previa al entrenamiento de una SVM.
- Se realizaron comparaciones entre los resultados de cuatro posibles formas de entrenar una SVM y de otras tantas Redes Neuronales, con el fin de entender el valor de ambos enfoques, así como las similitudes y diferencias entre ellos, y entre sus diferentes formas de solución. La excelente capacidad de generalización que poseen las SVM las coloca por encima de las Redes Neuronales en tareas de clasificación de dos clases. Bien se ha visto que resulta mucho mas eficiente, cómodo y rápido entrenar una SVM que una Red Neuronal, con mejores resultados.
- Se trabajó en el desarrollo de la parte formal de las SVM, permitiendo a un lector sin conocimientos previos entender más rápido y con mayor énfasis la manera en que se construyen. Esto es muy difícil de encontrar en la literatura sobre SVM existente.
- Las SVM resultan ser una técnica muy eficiente en las tareas de Reconocimiento de Patrones, tanto que muchos investigadores del mundo entero se encuentran trabajando con ellas. En México parecen haber pocas personas dedicadas a estos estudios, entre las que se cuentan los involucrados en esta tesis.

## 5.2 Expectativas sobre Trabajos Futuros.

Consideramos que después de haber realizado el presente trabajo, quedan varios caminos posibles a seguir en cuanto al estudio de las SVM:

- Desarrollar de manera formal una demostración más detallada de la validez del método Chunking Suave y encontrar, en la teoría, la manera de guiarlo hacia un estado en el que sea posible elegir un conjunto de trabajo adecuado para llegar en todos los casos al margen óptimo con una sola iteración adicional.

- ◊ Lo anterior incluye encontrar alguna manera adecuada de poder predecir la inclinación del hiperplano de separación definitivo y así poder incluir en el conjunto de trabajo de la iteración adicional a los vectores que quedan fuera del último margen, y que podrían no cumplir las Condiciones KKT al cambiar la inclinación del hiperplano de separación.
- ◊ Desarrollar algoritmos con métodos en paralelo, encontrando la región de decisión correcta a partir de regiones de decisión proveídas por SVM entrenadas a partir de diversos conjuntos de trabajo.
- ◊ Continuar la investigación y experimentación sobre el enfoque de AG para construir SVM. Es necesario encontrar el tipo de castigo adecuado, así como la manera de poder estimar los valores de sus parámetros.
- ◊ Trabajar en la búsqueda de mejores técnicas de preproceso para los vectores de entrenamiento, entre ellas la utilización de los AG que, como ya se vio, son buenos señalando a los candidatos a ser vectores de soporte.
- ◊ Estudiar el enfoque de las estrategias evolutivas como alternativa al uso de AG para entrenar las SVM.
- ◊ Combinar las técnicas vistas con métodos de preprocesamiento de los datos que detectan a los vectores candidatos a ser vectores de soporte, y así poder partir de un punto mas favorable a la hora de iniciar el entrenamiento de una SVM.
- ◊ Trabajar en la búsqueda de mejores y mas adecuados kerneles para ciertas aplicaciones.
- ◊ Aplicar las SVM a problemas donde el Reconocimiento de Patrones sea necesario y que tenga un aporte benéfico a la sociedad.
- ◊ Estudiar la parte de las SVM para regresión.

# APENDICES.

## A. Base de Datos de Dígitos Manuscritos.

La base de datos de dígitos manuscritos MNIST es un subconjunto de uno mayor creado por NIST (National Institute of Standards and Technology). Contiene un conjunto de entrenamiento de 60,000 muestras y un conjunto de prueba de 60,000. Los tamaños de las imágenes de los dígitos fueron normalizados para ajustarse a un rectángulo de $20 \times 20$ pixeles, mientras se conservó su razón de aspecto. Las imágenes resultantes con fueron centradas en imágenes de $28 \times 28$ mediante el cálculo del centro de masa de los pixeles y el traslado de la imagen de modo que este punto quedara en el centro del campo de $28 \times 28$.
MNIST es una buena base de datos para la gente que desea probar técnicas de aprendizaje y métodos de reconocimiento de patrones sobre datos del mundo real con un mínimo esfuerzo de preprocesamiento y formateo. Es ampliamente utilizada por investigadores como prueba para métodos de reconocimiento de patrones y por estudiantes para proyectos de clase en reconocimiento de patrones, aprendizaje de máquina y estadísticas.
La construcción de MNIST fue hecha a partir de NIST Special Database 1 y NIST Special Database 3. La mitad de los patrones en cada conjunto pertenece a SD-1 y la otra mitad a SD-3. Originalmente NIST determinó a SD-3 como su conjunto de entrenamiento y a SD-1 como su conjunto de prueba. Pero SD-3 es mucho más clara y fácil de reconocer, debido a que proviene de empleados del Census Bureau (Buró de Censos), y SD-1 proviene de estudiantes de preparatoria. Actualmente existe una versión actualizada de NIST llamada NIST Special Database 19, que contiene caracteres y otras formas escritas a mano.
Para nuestros experimentos se utilizaron muestras de MNIST obtenidas de la página web de Yann LeCun [LeC99], cuyo conjunto de entrenamiento está compuesto por 30,000 muestras de SD-3 y 30,000 de SD-1, y cuyo conjunto de entrenamiento consta de 5,000 muestras de SD-3 y 5,000 de SD-1.

## Bibliografía.


[BBM93] David Beasley, David R. Bull, Ralp R. Martin; An Overview of Genetic Algorithms: Part 1, Fundamentals; University Computing, Vol. 15, No. 2, pp. 58-69; 1993.

[BBM93a] David Beasley, David R. Bull, Ralp R. Martin; An Overview of Genetic Algorithms: Part 2, Research Topics; University Computing, Vol. 15, No. 4, pp. 170-181; 1993.

[Bur98] Christopher C. Burges; A Tutorial on Support Vector Machines for Pattern Recognition; Data Mining and Knowledge Discovery, Kluwer Academis Publishers, Boston; 1998.

[Chi98] K. K. Chin; Support Vector Machines Applied to Speech Pattern Classification; Marter's Thesis, Department of Computer Science, University of Cambridge; 1998.

[CLi96] Thomas F. Coleman, Yuying Li; A Reflective Newton Method for Minimizing a Quadratic Function Subject to Bounds on some of the Variables; SIAM Journal on Optimization, Vol. 6, No. 4, pp. 1040-1058; 1996.

[Coe99] Carlos Coello Coello; An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of Art and Future trends; Congress on Evolutionary Computation 1999, Vol. 1, pp. 3-13, IEEE Service Center; 1999.

[Coe99a] Carlos Coello Coello; A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques; Knowledge and Information Systems, Vol. 1, No. 3, pp. 269-308; 1999.

[Coe99b] Carlos Coello Coello; A Survey of Constraint Handling Techniques used with Evolutionary Algorithms; Reporte Técnico Lania-99-04, Laboratorio Nacional de Informática Avanzada; 1999.

[CST00] Nello Cristianini, John Shawe-Taylor; An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods; Cambridge University Press; 2000.

[CT00] Carlos Coello Coello, Gregorio Toscano Pulido; A Micro-Genetic Algorithm for Multiobjective Optimization; Reporte Técnico Lania-2000-06, Laboratorio Nacional de Informática Avanzada; 2000.

[CT01] Carlos Coello Coello, Gregorio Toscano Pulido; Multiobjective Optimization Using a Micro-Genetic Algorithm; Proceedings of the Genetic and Evolutionary Computation Conference 2001, pp. 274-282, Morgan Kaufmann Publishers, San Francisco; 2001.

[Dar00] Charles Darwin; El Origen de las Especies (1859), Introducción de Richard E. Leakey, Séptima Edición; Editorial Porrúa, México; 2000.

[Fis36] Fisher; The Use of Multiple Measurements in Axonomic Problems; Annals of Eugenics, Vol. 7, pp. 179-188; 1936.

[GMW91] Philip E. Gill, Walter Murray, Margaret H. Wright; Numerical Linear Algebra and Optimization, Volume 1; Addison-Wesley Publishing Company; 1991.

[Gun98] Steve Gunn; Support Vector Machines for Classification and Regression; ISIS Technical Report, Image Speech and Intelligent Systems Group, University of Southampton; 1998.

[HBC99] Arturo Hernández Aguirre, Bill P. Buckles, Carlos Coello Coello; Estrategias Evolutivas: La Versión Alemana del Algoritmo Genético (Parte I); Soluciones Avanzadas: Tecnologías de Información y Estrategias de Negocios, Año 7, No. 69, pp. 50-56; 1999.

[HBC99a] Arturo Hernández Aguirre, Bill P. Buckles, Carlos Coello Coello; Estrategias Evolutivas: La Versión Alemana del Algoritmo Genético (Parte II); Soluciones Avanzadas: Tecnologías de Información y Estrategias de Negocios, Año 7, No. 70, pp. 44-48; 1999.

[HL97] Frederick S. Hillier, Gerald J. Lieberman; Introduccion a la Investigación de Operaciones; McGraw-Hill; 1997.

[Kec01] Vojislav Kecman; Learning and Soft Computing: Support Vector Machines, Neural Networks and Fuzzy Logic Models; Massachusetts Institute of Technology; 2001.

[Las70] Leon S. Lasdon; Optimization Theory for Large Systems; MacMillan Publishing Co. Inc., MacMillan Series For Operations Research; 1970.

[LeC99] Yann Le Cun; The MNIST Database of Handwritten Digits; Dirección de Internet: http://yann.lecun.com/exdb/mnist/; 1999.

[Mez01] Efrén Mezura Montes; Uso de la Técnica NPGA para el Manejo de Restricciones en Algoritmos Genéticos; Tesis de Maestría, Universidad Veracruzana; 2001.

[Mic95] Zbigniew Michalewicz; A Survey of Constraint Handling Techniques in Evolutionary Computation Methods; Proceedings of the 4th Annual Conference on Evolutionary Programming; 1995.

[Mic96] Zbigniew Michalewicz; Genetic Algorithms + Data Structures = Evolution Programs; Springer-Verlag, Berlin Heidelberg; 1996.

[MiS96] Zbigniew Michalewicz, Marc Schoenauer; Evolutionary Algorithms for Constrained Parameter Optimization Problems, Evolutionary Computation, Vol. 4, No. 1, pp. 1-32; 1996.

[Min83] Michel Minoux; Mathematical Programming: Theory and Algorithms; John Wiley & Sons Ltd.; 1983.

[Mit96] Melanie Mitchell; An Introduction to Genetic Algorithms; MIT Press; 1996.

[MMR00] Kishan Mehrotra, Chilukuri K. Mohan, Sanjay Ranka; Elements of Artificial Neural Networks; MIT Press (1997); 2000.

[MVL99] Jonathan Miller, Borin Van Loon; Darwin para Principiantes; Icon Books Ltd., Era Naciente SRL (1995); 1999.

[NKT89] G. L. Nemhauser, A. H. G. Rinooy Kan, M. J. Todd, editores; Handbooks in Operations Research and Management Science, Volume I: Optimization; Elsevier Science Publishers B. V.; 1989.

[OFG97] Edgar Osuna, Robert Freund, Federico Girosi; An Improved Training Algorithm for Support Vector Machines; Proceedings of IEEE Workshop on Neural Networks for Signal Procesing, Amelia Island, Florida, USA; 1997.

[OFG97a] Edgar Osuna, Robert Freund, Federico Girosi; Training Support Vector Machines: An Application to Face Detection; Proceedings of CVPR'97; 1997.

[Pao89] Yoh-Han Pao; Adaptive Pattern Recognition and Neural Networks; Addison-Wesley Publishing Company, Inc.; 1989.

[Pla98] John C. Platt; Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines; Technical Report MSR-TR-98-14, Microsoft Research; 1998.

[Rey00] Juan Reyes ; Reconocimiento de Patrones: Una Aproximación para Expresión de Instrumentos y Arte Interactiva; Dirección de Internet: http://ccrma-www.stanford.edu/~juanig/articles/charlAndes/charlAndes.html; 2000.

[ROU99] Mathias Rychetsky, Stefan Ortmann, Michael Ullmann, Manfred Glesner; Accelerated Training of Support Vector Machines; Proceedings of International Joint Conference on Neural Networks, Washington D.C., USA; 1999.

[Sch92] Robert Schalkoff; Pattern Recognition: Statistical, Structural and Neural Approaches; John Wiley & Sons, Inc.; 1992.
[Sch97] Bernhard Scholkpof; Support Vector Learning; Ph. D. Dissertation Published by Oldenbourg Verlag, Munich; 1997.
[Sun96] Rangarajan K. Sundaram; A First Course in Optimization Theory; Cambridge University Press; 1996.
[Vap95] Vladimir Vapnik; The Nature of Statistical Learning Theory; Springer-Verlag; 1995.
[Wri00] Matt Wrigth; SVM Application List Web Page; Dirección de Internet: http://www.clopinet.com/isabelle/Projects/SVM/applist.html; 2000.