INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

MÉTODOS DE OPTIMIZACIÓN PARA EL PROBLEMA CUADRÁTICO GENERADO POR LAS MÁQUINAS DE SOPORTE VECTORIAL

TESIS QUE PARA OPTAR EL GRADO DE
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
PRESENTA

RODOLFO ESTEBAN IBARRA OROZCO

Asesor:	DR. NEIL HERNÁNDEZ GRESS.	
Coasesor:	DR. JUAN FRAUSTO SOLÍS.	
Comité de tesis:	DR. JAIME MORA VARGAS.	
	DR. JOSEPH AGUILAR MARTIN.	
Jurado:	DR. JAIME MORA VARGAS.	Presidente
	DR. JUAN FRAUSTO SOLÍS.	Secretario
	DR. NEIL HERNÁNDEZ GRESS.	Vocal

Atizapán de Zaragoza, Edo. Mex., agosto de 2004

CONTENIDO

1. INTRODUCCIÓN

La construcción de máquinas capaces de aprender ha experimentado un gran desarrollo en los últimos años gracias a la invención de las computadoras. Una gran variedad de algoritmos se han desarrollado para realizar este aprendizaje debido a que existen muchos problemas que no pueden ser resueltos por técnicas de programación clásicas, ya que no se cuenta con modelos matemáticos para dichos problemas.

El aprendizaje computacional se ha desarrollado por dos vertientes principales: 1) por métodos estadísticos y 2) por métodos que tratan de imitar el comportamiento del cerebro (redes neuronales). El primer tipo de estos métodos tiene el inconveniente de que para poder efectuar el aprendizaje es necesario conocer la distribución de probabilidad $p(x)$ de los datos que se quieren aprender. Las redes neuronales no necesitan conocer esta distribución de probabilidad pero traen consigo un nuevo problema: determinar los parámetros y la estructura de la red que realicen un aprendizaje correcto que simule el modelo de las mismas. Los obstáculos para obtener este aprendizaje son el sobreaprendizaje y el bajo aprendizaje y, aunque se han desarrollado heurísticas que tratan de resolver estos problemas, no es posible asegurar un aprendizaje óptimo en estos tipos clásicos de redes neuronales.

En el aprendizaje supervisado, la máquina de aprendizaje es entrenada con un conjunto de ejemplos (o entradas) con etiquetas asociadas (o valores) de salida. Usualmente los ejemplos son dados en forma de vectores de atributos, de tal forma que el espacio de entrada es un subconjunto de $\Re^n$. Una vez que los vectores de atributos se encuentran disponibles, se puede seleccionar un conjunto

de funciones para resolver el problema. Las funciones lineales son las mejores entendidas y más fáciles de aplicar. Los métodos estadísticos clásicos y la literatura de redes neuronales clásicas han desarrollado muchos métodos para diferenciar entre dos clases de instancias usando funciones lineales. Estas técnicas han construido la base para el desarrollo de sistemas más complejos.

Las Máquinas de Soporte Vectorial (MSV) son una técnica para clasificación, regresión y estimación de densidad recientemente desarrollada por Vladimir Vapnik [1]. Surgen como resultado práctico de la Teoría de Aprendizaje Estadístico. Esta técnica de clasificación utiliza funciones lineales y la solución se encuentra en una dimensión más grande de la que se encuentran los datos originalmente. Los datos son proyectados a este espacio con el objeto de encontrar una función lineal (hiperplano) que separe correctamente los datos.

Una de las características principales de las MSV, y por la que en los últimos años se ha observado un incremento considerable en su utilización, es que presentan un muy buen rendimiento al generalizar [1] en problemas de clasificación pese a no incorporar conocimiento específico sobre el dominio.

Para entrenar las MSV es necesario resolver un problema de optimización cuadrática con restricciones de igualdad y desigualdad lineales, dicho problema es convexo, lo que asegura que la solución encontrada es la óptima, esto en contraste con las técnicas clásicas de redes neuronales, donde es necesario resolver un problema de optimización sin restricciones y no convexo, en las cuales no es posible garantizar una solución óptima global.

El inconveniente que presentan las MSV es que la etapa de entrenamiento es demasiado dilatada para que puedan ser utilizadas en muchas aplicaciones reales.

El objetivo de este trabajo de tesis es encontrar, por medios matemáticos, algoritmos que ayuden a aumentar la velocidad de la etapa de entrenamiento.

Una de las características importantes de las MSV que debe de ser explotada para lograr este objetivo es que el entrenamiento de los datos que se encuentran más cerca de la función de decisión, llamados vectores de soporte, produce el mismo resultado que con el conjunto completo de datos de entrenamiento. A partir de esta característica se han desarrollado algoritmos para aumentar la velocidad de entrenamiento y para evitar el problema de memoria [2] [3]. También partiendo de esta característica surge la hipótesis de que si se implementa un algoritmo que busque de una manera muy rápida un subconjunto de los datos de entrenamiento que estén cerca de la función de decisión y posteriormente se realize el entrenamiento de la MSV sólo con esos datos, el tiempo total de entrenamiento será menor que el utilizado si éste se realiza con el conjunto completo de

[1] Generalizar: Abstraer lo que es común y esencial a muchas cosas, para formar un concepto general que las comprenda todas. Diccionario de la Real Academia Española

datos.

Organización de la tesis.

El capítulo 2 comienza con una breve reseña histórica de las redes neuronales: desde el primer modelo de una neurona desarrollado por McCullloch y Pitts en 1943 hasta el desarrollo de las MSV en el año de 1995 realizado por Vladimir Vapnik. En este mismo capítulo se realiza una descripción de los diferentes tipos de redes neuronales mencionados, así como de los métodos clásicos de optimización utilizados en el desarrollo de este trabajo.

El capítulo 3 detalla el los métodos implemantados para aumentar la velocidad en el entrenamiento de las MSV.

En el capítulo 4 se muestran los resultados obtenidos por cada uno de los métodos elaborados y se analizan dichos resultados.

Finalmente, en el capítulo 5, se presentan las conclusiones finales y el trabajo futuro que sugen de esta investigación.

Este trabajo fué realizado en parte gracias al apoyo financiero de CONACyT: Proyecto 37368.

2. MARCO TEÓRICO

2.1. REDES NEURONALES.

2.1.1. RESEÑA HISTÓRICA DE LAS REDES NEURONALES.

Frecuentemente se considera que la historia de las redes neuronales comienza con el artículo publicado por Warren McCulloch y Walter Pitts en el año de 1943 [4]. Este artículo presenta el primer modelo matemático de una neurona y muestra que una función aritmética o lógica puede ser computada usando incluso tipos simples de redes neuronales, sin embargo, este modelo carecía de un algoritmo de aprendizaje.

En 1949, Donald Hebb escribe el libro titulado ”The Organization of Behavior” [5], donde introduce dos ideas fundamentales que han influido de manera decisiva en el campo de las redes neuronales: i) la idea de que una percepción o un concepto se representa en el cerebro por un conjunto de neuronas activas simultáneamente y ii) la idea de que la memoria se localiza en las conexiones entre las neuronas (sinápsis). Las hipótesis de Hebb, basadas en investigaciones psicofisiológicas, presentan de manera intuitiva el modo en que las neuronas memorizan información y se plasman sintéticamente en la famosa regla aprendizaje de Hebb. Esta regla indica que las conexiones entre dos neuronas se refuerzan si ambas son activadas. Muchos de los algoritmos actuales proceden de los conceptos de este psicólogo y, a pesar de las críticas recibidas, como la existencia de conexiones inhibitorias y no sólo excitatorias, siguen teniendo una gran influencia.

En 1957, Frank Rosenblatt inventó el Perceptrón [6]. Este sistema es una extensión del modelo matemático concebido por McCulloch y Pitts y funciona basado en el principio de activar neuronas a partir de un valor de entrada el cual modifica un peso asociado a una neurona. El Perceptrón es el primer algoritmo de aprendizaje autónomo conocido, sin embargo, tiene la desventaja de clasificar sólo patrones linealmente separables.

En 1962, Bernard Widrow desarrolló otro elemento neuronal computacional al que llamó Adaline (ADAptive LInear NEuron)[7]. Este elemento es similar al Perceptrón pero incorpora una función de umbral y un algoritmo de aprendizaje diferente. El algoritmo de aprendizaje propuesto por Widrow para ser utilizado en el Adaline es la regla Delta. Esta regla es un método de gradiente descendente. Con la adición de la regla Delta, Adaline tiene el potencial para aprender con mayor precisión que el Perceptrón, manteniendo la certeza de su convergencia.

En 1969 Marvin Minsky y Seymour Papert publicaron el libro: ”Perceptrons: An introduction to Computational Geometry”[8]. Este libro significó para muchos investigadores el final de las redes neuronales. En él se presenta un análisis detallado del Perceptrón en términos de sus capacidades y limitantes, en especial en cuanto a las restricciones que existen para los problemas que un Perceptrón puede resolver; enfatiza que la mayor desventaja de este algoritmo es su incapacidad para solucionar problemas que no sean linealmente separables.

En 1986 se publicó un artículo presentado por Rumelhart, Geofrey Hinton y Ronald Williams [9] en el cual se describe el algoritmo de retropropagación para redes neuronales multicapas. Con este tipo de redes neuronales es posible crear funciones de separación no lineales y de esta forma es resuelto el problema que, para Minsky y Papert, parecía un obstáculo insuperable para la explotación y desarrollo de las redes neuronales. Con la publicación de este artículo el interés en la neurocomputación alcanzó su máximo nivel. Después se mostró que el algoritmo de retropropagación estaba siendo redescubierto ya que haba sido conocido y publicado con anterioridad por varios investigadores (por ejemplo, Arthur Bryson y Yu-Chi Ho en 1969, Paul Werbos en 1974 y David Parker en 1985).

Con el algoritmo de retropropagación vienen asociados problemas como 1) la inicialización, ya que con una mala inicialización el algoritmo puede llegar a no converger, 2) tiempo de cálculo, puesto que este algoritmo es muy lento, y 3)la selección de una estructura óptima para tener un máximo de generalización. Para tratar de resolver el último punto se han creado métodos como *crossvatidation*[1], algoritmos *pruning* y métodos de construcción incrementales. Sin embargo, a pesar de que con estos algoritmos se obtienen buenas estructuras de la red, no es posible asegurar que

[1]Las técnicas de *crossvalidation* consisten básicamente en tomar un porcentaje de la base de datos para realizar el aprendizaje y el porcentaje restante para probar la generalización.

la estructura es la óptima.

Las MSV, desarrolladas por Valdimir Vapnik en 1995, resuelven el problema de escoger una buena estructura de la red neuronal ya que este algoritmo, el cual se encuentra bien fundamentado matemáticamente, mientras realiza la etapa de entrenamiento también obtiene la estructura óptima. El inconveniente de este nuevo algoritmo es que para realizar el entrenamiento es necesario resolver un problema de optimización cuadrática el cual crece con el cuadrado de los datos de entrenamiento, por esta razón, para problemas grandes (mayores a 10 000 datos), el problema se vueve muy difícil debido a que es necesario emplear mucha memoria y tiempo computacional para resolverlo.

En las secciones siguientes se presenta una breve introducción de cada uno de los algoritmos mencionados.

2.1.2. REDES DE UNA CAPA.

Se describen dos neuronas clásicas, el Perceptrón y el Adaline. Ambas difieren en su origen y fueron desarrolladas por investigadores de diferentes ramas, uno de la neurofisiología y el otro de la ingeniería. El Perceptrón fué desarrollado por Frank Rosenblatt con la motivación de resolver problemas de visión computacional y para realizar tareas de reconocimiento de patrones, mientras que Bernard Widrow desarrolló el Adaline para tratar problemas provenientes del campo del proceso de señales, específicamente del problema de la cancelación adaptiva del ruido.

2.1.2.1. Perceptrón

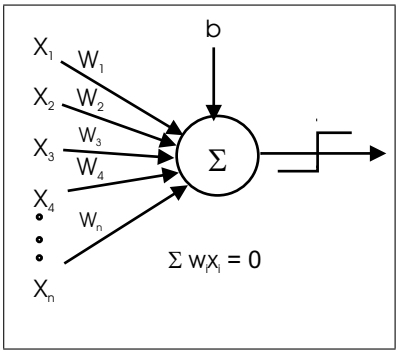


Figura 2.1: Perceptrón

El Perceptrón fue uno de los primeros elementos de procesamiento capaces de aprender. En 1947, año en que fue inventado, el problema de aprendizaje era una tarea sin solución. Para ese entonces, la idea de una adaptación autónoma de pesos usando datos de entrenamiento (ejemplos,

patrones, medidas, observaciones, imágenes digitales) causó gran expectación. El Perceptrón realiza un tipo de aprendizaje supervisado.

En este esquema de aprendizaje supervisado, el vector de pesos ($\boldsymbol{w}$) es inicializado aleatoriamente y el Perceptrón es entrenado con ejemplos de atributos ($\boldsymbol{x}_i$) y la salida deseada ($\boldsymbol{y}_i$) para cada ejemplo. El algoritmo de aprendizaje del Perceptrón es una regla de corrección de error que cambia los pesos $\boldsymbol{w}$ de los elementos que se encuentren mal clasificados. El cambio de los pesos es modificado por un factor multiplicativo α con valor entre 0.1 y 1. El propósito de este factor multiplicativo es hacer más pequeño el cambio en los pesos, llamado paso de aprendizaje, y hacer que el Perceptrón tome pasos más pequeños hacia la solución, disminuyendo con esto oscilaciones que son causadas en el algoritmo con un paso grande.

El algoritmo se puede resumir de la siguiente forma:

1. Inicializar los pesos $\boldsymbol{w}$
2. Inicializar el tamaño del paso de aprendizaje α
3. Mientras no se cumpla la condición de paro (mientras que haya error)
   - Calcular la respuesta de la red

$$a = b + w_i x_i \qquad f(a) = \begin{cases} 1 & \text{si } a > 0 \\ 0 & \text{si } a = 0 \end{cases}$$

   - Si hay error, es decir, si $f_i \neq y_i$

$$w_i(new) = w_i(old) + \alpha \Delta w_i$$

$$donde \qquad \Delta w_i = x_i y_i$$

   - Si no hay error, es decir, si $f_i = y_i$

$$w_i(new) = w_i(old) \qquad \forall\, i$$

Cuadro 1.1 Algoritmo de aprendizaje del Perceptrón.

Ventajas del Perceptrón

- Primer algoritmo con aprendizaje autónomo conocido. Este algoritmo fue el primero con la capacidad de realizar un aprendizaje autónomo a partir de ejemplos. El algoritmo es sencillo de programar y realiza un aprendizaje rápido.
- Es la base para el algoritmo de retropropagación del gradiente.

Desventajas del Perceptrón

- Únicamente puede clasificar patrones linealmente separables. Ésto motivó a la creación de otros tipos de redes neuronales como el Adaline que, aunque la función de decisión continúa siendo lineal, cuenta con un algoritmo que minimiza el número de datos mal clasificados, Poco después surgen otro tipo de algoritmos que forman una estructura con estos elementos de aprendizaje para obtener funciones de decisión no lineales (redes neuronales a capas múltiples).
- No garantiza una buena generalización. Esto debido a que el algoritmo se detiene inmediatamente después de encontrar un hiperplano que separe correctamente los datos (sin tener la posibilidad de mejorar esa solución) o que se cumplan cierto número de iteraciones, por lo que no es posible garantizar que la solución encontrada sea la óptima para la generalización.

2.1.2.2. Adaline.

El algoritmo de aprendizaje Adaline [7] (también conocido como regla del error cuadrático medio, la regla delta ó la regla Widrow-Hoff) minimiza el error cuadrático medio durante la etapa de entrenamiento. Este algoritmo se basa en el método del gradiente descendente. A continuación se muestra la derivación matemática de este algoritmo de aprendizaje.
El vector de pesos $\boldsymbol{w}$ se obtiene minimizando el error cuadrático medio:

$$E = \frac{1}{2}(y - s)^2 \tag{2.1}$$

donde $s = \sum_{i=1}^{n} x_i w_i$. Esto se logra alterando los pesos en la dirección que produce el máximo descenso en la superficie de error:

$$\boldsymbol{w} = \boldsymbol{w} + \Delta \boldsymbol{w} \tag{2.2}$$

La dirección de cambio se obtiene mediante el gradiente de la función. El gradiente de una función nos indica la dirección hacia la que mayor incremento tiene la función. Como en este caso se quiere

minimizar la función, se utiliza el negativo del gradiente:

$$\Delta \boldsymbol{w} = -\nabla E \tag{2.3}$$

Derivando la función de error (aplicando la regla de la cadena):

$$\Delta w_i = (y - s)x_i \tag{2.4}$$

El algoritmo de aprendizaje se implementa como se muestra en el siguiente cuadro:

1. Inicializar los pesos $\boldsymbol{w}$
2. Mientras no se cunpla la condición de paro
   a) Para cada elemento i de n
      * Calcular
      $$s = \sum_{i=1}^{n} x_i w_i$$
      * Realizar aprendizaje
      $$w_i(new) = w_i(old) + \alpha(y - s)x_i$$
3. Revisar condición de paro: si el cambio más importante es más pequeño que un umbral predeterminado.

Cuadro 1.2 Algoritmo de aprendizaje del Adaline.

Ventajas del Adaline

- Método de paro del aprendizaje para problemas no linealmente separables.
- El algoritmo minimiza el error cuadrático medio, esto permite a la red mejorar el aprendizaje aún cuando haya alcanzado la solución deseada.

Desventajas del Adaline

- La función de decisión continúa siendo lineal. El algoritmo trabaja con funciones de decisión lineales y, aunque minimiza el error en la clasificación de patrones no linealmente separables, en la mayoría de los problemas reales, ya que se requieren de otro tipo de funciones de decisión (con mayor capacidad) para realizar una clasificación satisfactoria (polinomiales, radiales). En la siguiente sección se verán las redes de capas múltiples, estas redes tienen la capacidad de elaborar este tipo de funciones.

2.1.3. REDES NEURONALES A CAPAS MÚLTIPLES

Los algoritmos de una capa permiten clasificar patrones linealmente separables. Aunque el algoritmo Adaline provee un método basado en el gradiente descendente para disminuir el error, incluso cuando los patrones no son linealmente separables, las soluciones que pueden ser obtenidas por estas redes -hiperplanos de decisión- no son las apropiadas para obtener un error mínimo en muchas aplicaciones reales.

Las redes multicapas implementan funciones de decisión lineales pero en un espacio donde las entradas han sido mapeadas no linealmente. La clave del poder que proveen estas redes neuronales es que admiten algoritmos simples y la forma de la no linealidad puede ser aprendida de los datos de entrenamiento.

2.1.3.1. Algoritmo de Retropropagación del gradiente.

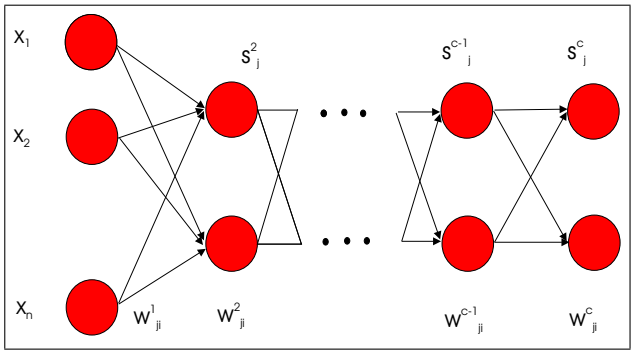


Figura 2.2: Red neuronal a capas múltiples

Uno de los métodos más populares y el más ampliamente usado para el entrenamiento de redes multicapas es el algoritmo de retropropagación del gradiente [10] [11]. Este algoritmo se basa en el método del gradiente descendente y constituye una extensión natural al algoritmo Adaline.

La mayoría de los algoritmos de entrenamiento involucran un proceso iterativo para minimizar el error de una función mediante el ajuste de pesos en una secuencia de pasos. Una vez que se ha

aplicado un patrón a la entrada de la red como estímulo, éste se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas. Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa escondida que contribuyen directamente a la salida. Sin embargo, las neuronas de la capa escondida sólo reciben una fracción de la señal total de error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total. Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento.

Las funciones de activación (fac) utilizadas en cada neurona, deben de cumplir los siguientes requisitos:

1. La función de activación necesita ser continua, no decreciente y diferenciable, con objeto de calcular la regla delta para minimizar el error.
2. Fácil de calcular computacionalmente.

En el siguiente cuadro se resume, de forma general, el algoritmo de retropropagación del gradiente:

1. Inicialización aleatoria de pesos
2. Mientras la condición de paro sea falsa
   - Para cada elemento de la base de datos
     - Generalización
     - Retropropagación
     - Actualización de pesos
   - Revisar condición de paro: ya no hay error o se cumplió cierto número de iteraciones

Cuadro 1.3 Algoritmo de aprendizaje de retropropagación del gradiente.

El algoritmo de retropropagación realiza tres pasos importantes que son: 1) Generalización, 2) Retropropagación y 3) Actualización de pesos.

Los siguientes algoritmos describen cada uno de estos pasos:

- Para todas las capas de la red: c=1,. . . ,nc

$$SP_j^c = \sum_{i=0}^{n(c-1)} w_i^r s_i^{(c-1)}$$

$$s_j^c = fac_j^c(SP_j^c)$$

$$s_i^0 = x_i$$

Cuadro 1.4 Etapa de generalización del algoritmo de retropropagación.

.

- Para todas las capas, comenzando de la capa de salida c=cr,. . . ,2
  - Para todas las neuronas de las capas de salida i=1,. . . , NNcr

$$\delta_i = (y_i - s_i) fac_i'(SP_i)$$

$$\Delta w_{ij}^c = \alpha \delta_i s_j^{c-1}$$

  - Para todas las neuronas de las capas ocultas i=1,. . . , NNc
  - c=cr-1, . . . , 2

$$(S\delta_i^c) = \sum_{k=1}^{n(c+1)} \delta_k^{c+1} w_{ik}^{c+1}$$

$$(\delta_i^c) = (S\delta_i^c) fac_i'(SP_i^c)$$

$$\Delta w_{ij}^c = \alpha \delta_i^c s_j^{c-1}$$

  donde $\delta_k = (y_k - s_k^c) fac'(SP_k^c)$

Cuadro 1.5 Etapas de retropropagación y actualización de pesos.

Ventajas del algoritmo de retropropagación del gradiente

- Este algoritmo resucitó el estudio de las redes neuronales. Con las redes neuronales de capas múltiples y una forma para entrenarlas (algoritmo de retropropagación del gradiente), las redes neuronales son retomadas para su estudio, puesto que vencen el obstáculo de las redes neuronales de una capa de poder clasificar correctamente sólo datos linealmente separables.

Desventajas del algoritmo de retropropagación del gradiente

- Convergencia hacia mínimos locales.
- Bajo cierta inicialización se pueden tener oscilaciones o hasta movimientos caóticos.
- Es necesario indicar la arquitectura de la red. Para realizar el entrenamiento es necesario indicar la estructura de la red (lo cual determinará la capacidad de la función de decisión) y esto involucra el problema de poder encontrar la estructura óptima (o al menos una estructura con la cual se obtengan buenos resultados en clasificación y en generalización) de la red. Algoritmos de construcción y técnicas como *crossvalidation* fueron diseñados para tratar de resolver este problema.

2.1.4. ALGORITMOS DE CONSTRUCCIÓN.

La arquitectura de una red (el número de unidades y la topología de conexiones) tiene un impacto importante en el aprendizaje y en la capacidad de generalización de una red. Varias técnicas, como las mostradas en [13], [14] y [15], han sido desarrolladas para optimizar la arquitectura, en algunos casos como parte del proceso de entrenamiento de la red. Es importante distinguir entre dos aspectos distintos del problema de selección de arquitectura. Primero, se necesita un procedimiento sistemático para explorar algún espacio de posibles arquitecturas, Segundo, es necesario tener una forma de decidir cual de las arquitecturas consideradas debe de ser seleccionada. Este es usualmente determinado por el requisito de lograr la mejor posible generalización. El método más simple de optimización de la estructura de la red consiste en una búsqueda exhaustiva en una clase de arquitecturas neuronales. Este método puede requerir de un esfuerzo computacional muy significativo y sólo buscaría en un conjunto muy restringido de clases de redes. Si se incrementara el conjunto de modelos pronto se llegaría al punto de tener recursos computacionales insuficientes para una búsqueda completa. Una desventaja obvia de estos métodos es que tienen que ser entrenadas muchas redes. Esto puede ser evitado considerando una red que es inicialmente relativamente muy pequeña y permitir que nuevas neuronas y conexiones sean agregadas durante

el entrenamiento. Técnicas de este tipo son llamadas algoritmos de construcción. El algortimo Cascade Correlation, descrito a continuación, es un ejemplo de este tipo de algoritmos.

2.1.4.1. Cascade Correlation

La arquitectura Cascade Correlation [16] tiene varias ventajas sobre los algoritmos existentes: aprende muy rápidamente, la red determina su propio tamaño y arquitectura, mantiene la estructura de la red incluso si el conjunto de entrenamiento cambia y no requiere retropropagación de señales de error. El algoritmo Cascade Correlation empieza con una red mínima, después entrena automáticamente y agrega nuevas unidades en la capa escondida, creando una estructura multicapa. Para cada nueva unidad agregada, se intenta maximizar la magnitud de la correlación entre la salida de la nueva unidad y el error residual que se está tratando de eliminar.
Se agregan unidades escondidas a la red una por una. Cada nueva unidad en la capa escondida recibe una conexión por cada una de las entradas originales de la red y de cada unidad en la capa escondida preexistente. Los pesos de las unidades en la capa escondida son congeladas cada vez que son agregadas: sólo las conexiones de salida son entrenadas repetidamente.

Ventajas de Cascade Correlation

- En general, los algoritmos de construcción tienen una buena generalización.
- No se necesita que el usuario indique la arquitectura de la red.
- El aprendizaje es muy rápido.

Desventajas de Cascade Correlation

- No es posible asegurar la estructura estructura óptima de la red.
- El algoritmo de aprendizaje de Cascade Correlation trabaja con una sola capa oculta, esto puede propiciar el uso de muchos más elementos de aprendizaje que si se utilizara un número mayor de capas ocultas.

2.2. MÁQUINAS DE SOPORTE VECTORIAL

2.2.1. FUNDAMENTOS TEÓRICOS

La Teoría de de Aprendizaje Estadístico tuvo sus inicios a finales de los años 60's. En esa época se utilizaba para el análisis del problema de la estimación de una función para una colección dada de datos. Basados en esta teoría, a mediados de los años 90's, fue propuesto un nuevo tipo de algoritmo de aprendizaje llamado Máquina de Soporte Vectorial (MSV). Este nuevo algoritmo hizo que la Teoría de Aprendizaje Estadístico no fuera sólo utilizada como una herramienta para análisis teórico, sino que además sea utilizada como una herramienta para crear algoritmos prácticos para la estimación de funciones multidimensionales [17]. Después del éxito de las MSV en la solución de problemas de la vida real (reconocimiento de caracteres, reconocimiento de voz, etc), el interés en la Teoría de Aprendizaje Estadístico se ha incrementado significativamente. Por primera vez, resultados matemáticos abstractos de la Teoría de aprendizaje estadístico han tenido impacto directo en algoritmos para el análisis de datos.

Para una determinada tarea de aprendizaje, con una cantidad finita de datos de entrenamiento, el mejor rendimiento al generalizar se conseguirá si se equilibra la balanza entre la exactitud obtenida sobre ese conjunto de entrenamiento y la capacidad de la máquina, esto es, la habilidad de la máquina de aprender cualquier conjunto de entrenamiento sin error. Un ejemplo de una máquina con demasiada capacidad es como un botánico con memoria fotográfica a quien cuando se le presenta un nuevo árbol dice que no es un árbol porque tiene diferente número de hojas que cualquiera de los que ha visto con anterioridad, en el caso contrario, una máquina con muy baja capacidad es como un botánico flojo que dice que si es verde, es un árbol. Ninguno de los dos tiene la capacidad de generalizar correctamente. La exploración y formalización de estos conceptos ha abierto una de las líneas más prometedoras dentro de la Teoría de Aprendizaje Estadístico.

Los capítulos 2.2.1.1 y 2.2.1.2 introducen los Principios de Minimización de Riesgo Empírico y Minimización de Riesgo Estructurado [1] [2] [10]. Ambos Principios son de gran importancia porque son la base teórica de las MSV y nos permiten entender el poder de clasificación y generalización de éstas.

2.2.1.1. Minimización de Riesgo Empírico.

Supóngase que se tiene una máquina cuya tarea sea aprender el mapeo $x_i \mapsto y_i$. La máquina está actualmente definida por un conjunto de posibles mapeos $x \mapsto f(x, \lambda)$, donde las funciones $f(x, \lambda)$ son etiquetadas por parámetros ajustables λ. Se asume que la máquina es determinística:

para una entrada dada x, y una selección de λ, siempre se tendrá la misma salida. Una selección particular de λ generan lo que se llama una máquina entrenada. El riesgo esperado para la máquina entrenada es:

$$R(\boldsymbol{\lambda}) = \int \big| f_\lambda(\boldsymbol{x}) - y \big| P(\boldsymbol{x}, \boldsymbol{y}) dx dy \tag{2.5}$$

Las funciones f_λ son llamadas hipótesis, y el conjunto $\{f_\lambda(x) : \lambda \in \Lambda\}$ es llamado el espacio de hipótesis y se denota con $\mathcal{H}$. El riesgo esperado es la medida de que tan bien una hipótesis predice la etiqueta correcta y para un punto x. El conjunto de funciones $f(\lambda)$ puede ser, por ejemplo, un conjunto de funciones radiales o un Perceptrón multicapa con un cierto número de capas escondidas. En este caso, el conjunto Λ es el conjunto de pesos de la red. Ya que la distribución de probabilidad es desconocida, no se tiene la capacidad de calcular, y por lo tanto minimizar, el riesgo esperado $R(\lambda)$. Sin embargo, debido a que se tiene acceso a una muestra de $P(x, y)$, podemos calcular una aproximación de $R(\lambda)$, a esta aproximación se le llama riesgo empírico:

$$R_{emp}(\boldsymbol{\lambda}) = \frac{1}{n} \sum_{i=1}^{n} \big| f_\lambda(x_i) - y_i \big| \tag{2.6}$$

Un método común consiste en minimizar el riesgo empírico en lugar del riesgo esperado. A este método se le llama Principio de Minimización de Riesgo Empírico, y surge de la intuición de que si R_{emp} converge a R, el mínimo de R_{emp} debe converger al mínimo de R puesto que $R < R_{emp}$. Si la convergencia del mínimo de R_{emp} a el mínimo de R no existe, el Principio de Minimización de Riesgo Empírico no permite hacer ninguna inferencia basada en el conjunto de datos y se dice que el Principio es no consistente. Como lo muestra Vapnik y Chervonenkis [18] [19] [20], la consistencia toma lugar si y sólo si la convergencia en la probabilidad de R_{emp} a R es reemplazada por convergencia uniforme en probabilidad. Vapnik y Chervonenkis también mostraron que la condición necesaria y suficiente para la consistencia del Principio de Riesgo Empírico es que la dimensión VC (h) sea finita en el espacio de hipótesis $\mathcal{H}$. La dimensión VC del espacio de hipótesis $\mathcal{H}$ (o dimensión VC del clasificador f_λ es un número natural, posiblemente infinito, el cual es el número máximo de datos que pueden ser separados de todas las posibles formas por el conjunto de funciones f_λ. La dimensión VC es una medida de la complejidad del conjunto $\mathcal{H}$ y frecuentemente, pero no necesariamente, proporcional al número de parámetros libres del clasificador f_λ.

La teoría de convergencia uniforme en probabilidad desarrollada por Vapnik y Chervonenkis también provee asíntotas en el desvío del riesgo empírico del riesgo esperado. Una asíntota uniforme típica de Vapnik y Chervonenkis, que tiene una probabilidad de $1 - \eta$, tiene la siguiente forma:

$$R(\lambda) \leq R_{emp}(\lambda) + \sqrt{\frac{h\Big(ln\frac{2n}{h} + 1\Big) - ln\frac{n}{4}}{n}} \tag{2.7}$$

donde h es la dimensión VC de f_λ. De esta asíntota es claro que con la finalidad de lograr un riesgo esperado pequeño, lo cual es bueno para realizar generalización, el riesgo empírico y la relación entre la dimensión VC y el número de datos tiene que ser pequeño. Ya que el riesgo empírico es usualmente una función decreciente de h, por consecuencia, para un número dado de datos de entrenamiento, hay un valor óptimo de la dimensión VC. La selección de un valor apropiado de h (que en muchas técnicas es controlado por el número de parámetros libres del modelo) es crucial para obtener un buen funcionamiento, especialmente cuando el número de datos es pequeño. Este problema es difícil y es resuelto usualmente por algún tipo de técnica de *crossvalidation*.

2.2.1.2. Minimización de Riesgo Estructurado.

La técnica de Minimización de Riesgo Estructurado es un intento por evitar el problema de escoger una dimensión VC apropiada. De la ecuación (2.7) es claro que un valor pequeño del riesgo empírico no implica necesariamente un valor pequeño del riesgo esperado. Un principio de inducción diferente, llamado Principio de Riesgo Estructurado, fue propuesto por Vapnik. El principio se basa en la observación de que, con la finalidad de hacer el riesgo esperado pequeño, ambos lados de la ecuación (2.7) deben ser pequeños. Consecuentemente, la dimensión VC y el riesgo empírico deben ser minimizados al mismo tiempo. Con la finalidad de implementar el Principio de Minimización de Riesgo Estructurado se necesita una estructura anidada de espacios de hipótesis:

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_n \subset \ldots \tag{2.8}$$

Con la propiedad de que $h(n) \leq h(n+1)$ donde $h(n)$ es la dimensión VC del conjunto $\mathcal{H}_n$. Entonces, la ecuación (2.7) sugiere que, no tomando en cuenta los factores logarítmicos, el siguiente problema debe ser resuelto:

$$min_{\mathcal{H}_n}\left(R_{emp}[\lambda] + \sqrt{\frac{h(n)}{n}}\right) \tag{2.9}$$

Este principio se encuentra claramente fundamentado matemáticamente, pero puede ser difícil de implementar por las siguientes razones:

a) La dimensión VC de $\mathcal{H}_n$ puede ser difícil de calcular, y sólo hay un número pequeño de modelos para los cuáles se conoce como calcular la dimensión VC.

b) Aún cuando se asuma el cálculo de la dimensión VC de $\mathcal{H}_n$, no es fácil solucionar el problema de minimización (2.9).

En la mayoría de los casos se tiene que minimizar el riesgo empírico para cada conjunto $\mathcal{H}_n$, y luego escoger el $\mathcal{H}_n$ que minimiza la ecuación (2.9).

2.2.2. DERIVACIÓN MATEMÁTICA DE LAS MÁQUINAS DE SOPORTE VECTORIAL.

A continuación se describe la derivación matemática de las MSV. Se presentarán los siguientes tres casos:

1. El primer caso es el más simple: el clasificador lineal y el problema linealmente separable. Éste trabaja sólo con datos linealmente separables y por lo tanto no puede ser usadas en muchas aplicaciones reales. Sin embargo, es el algoritmo más fácil de entender y forma el bloque principal de construcción para MSV más complejas. Este caso muestra las principales características de este tipo de máquinas de aprendizaje.

2. Posteriormente se describe el caso donde, a pesar de que las funciones de decisión continúan siendo hiperplanos, se permite cierto grado de datos mal clasificados. A este caso se le llama clasificador lineal y problema no linealmente separable.

3. Por último se presenta el caso más interesante y de mayor uso, donde los datos de entrada son mapeados a un espacio de una dimensión más alta (las capas escondidas en un modelo de red neuronal), resultando en un clasificador no lineal. A este caso se le llama clasificador no lineal y problema no linealmente separable.

Las figuras 2.3, 2.4 y 2.5 ejemplifican cada uno de estos tres casos. En estas figuras se puede observar que los márgenes de separación se encuentran sobre un número muy reducido del conjunto total de datos de entrenamiento. Estos datos son los llamados vectores de soporte y son de gran importancia porque el entrenamiento de la MSV únicamente con esos datos produce el mismo resultado que con el conjunto de datos completo.

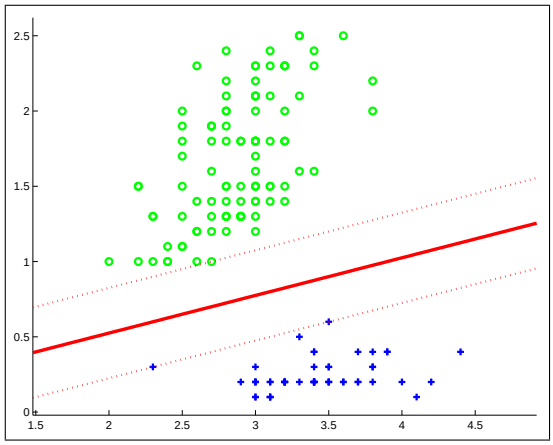


Figura 2.3: Caso 1. Clasificador lineal y problema linealmente separable.

Como se muestra en la figura 2.3, con las características del caso 1 únicamente es posible separar

datos linealmente separables, es decir, sin translape. mediante funciones lineales -hiperplanos-. Si los datos se encuentran translapados la MSV, con las condiciones del caso 1, no podrá encontrar una solución.

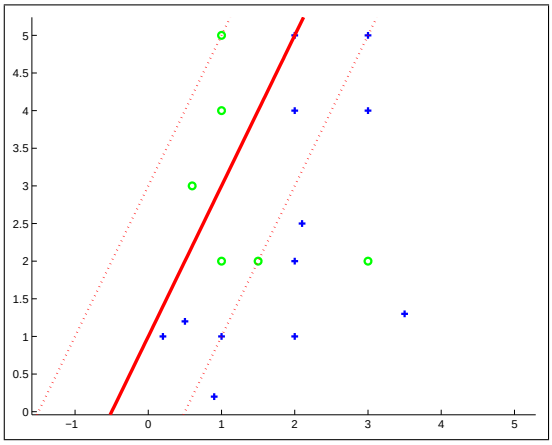


Figura 2.4: Caso 2. Clasificador lineal y problema no linealmente separable.

Las características del caso 2 (figura 2.4), permiten clasificar datos de manera incorrecta, esto es muy importante para cuando los datos no son linealmente separables o tienen ruido. Es importante notar que la función de decisión continúa siendo lineal.

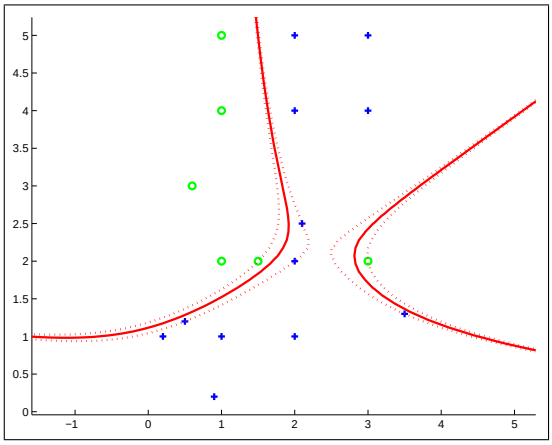


Figura 2.5: Caso 3. Clasificador no lineal y problema no linealmente separable.

La figura 2.5 muestra datos de entrenamiento translapados (no linealmente separables). Con las características del caso 3 la clasificación se realizó mediante una función de decisión no lineal. Como se verá mas adelante, los datos de entrenamiento son llevados a una dimensión más alta y ahí son clasificados linealmente, y si se mapean nuevamente los datos a su dimensión original, se puede observar que las funciones obtenidas son no lineales.

Las deducciones matemáticas de estos tres casos son las siguientes:

2.2.2.1. Clasificador lineal y problema linealmente separable

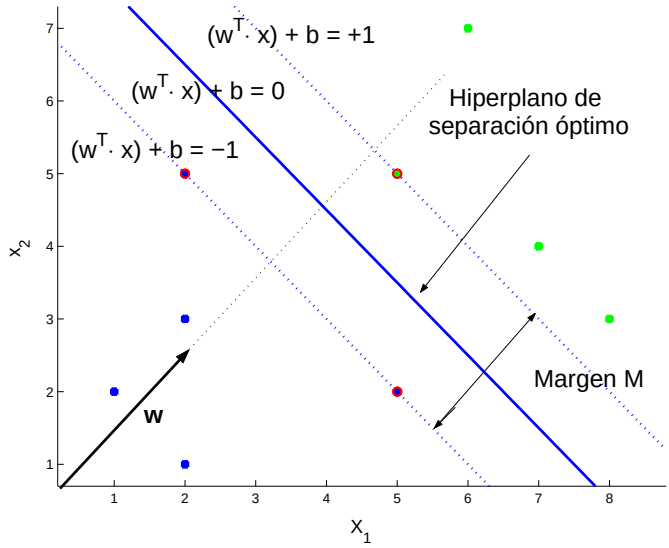


Figura 2.6: Hiperplano de separación óptimo con el margen de separación máximo entre las dos clases

Se considera primero el caso en que el conjunto de datos es linealmente separable y se desea encontrar el mejor hiperplano que separe los datos. Linealmente separable significa que se puede encontrar un par $(\boldsymbol{w}, \boldsymbol{b})$ tal que:

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + \boldsymbol{b} \geq \quad 1 \qquad \forall\, \boldsymbol{x_i} \in clase1 \tag{2.10}$$

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + \boldsymbol{b} \leq \quad -1 \qquad \forall\, \boldsymbol{x_i} \in clase2 \tag{2.11}$$

Ambas ecuaciones, 2.10 y 2.11 se pueden representar como:

$$\boldsymbol{y_i}(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 \tag{2.12}$$

La representación geométrica de esta ecuación se muestra en la figura 2.6 mediante las 2 líneas punteadas (hiperplanos en dimensiones mayores), la línea continua que se encuentra a la mitad de las líneas punteadas representa la función de separación. Para encontrar el hiperplano que mejor separe los datos es necesario maximizar el margen de separación que se encuentra entre los datos más cercanos de las diferentes clases y finalmente se traza un hiperplano a la mitad de dicho margen. Se sabe que la distancia D de un punto $P(x_{1p}, x_{2p}, \ldots, x_{np})$ a un hiperplano d($\boldsymbol{x}, \boldsymbol{w}, b$) es:

$$D = \frac{|\boldsymbol{w} \cdot x_i + b|}{\| \boldsymbol{w} \|} = \frac{|w_1 x_{1p} + w_2 x_{2p} + \ldots + w_n x_{np} + b|}{\| w_1^2 + w_2^2 + \ldots + w_n^2 \|} \tag{2.13}$$

Se tiene que la distancia D entre un vector de soporte y el hiperplano de separación es:

$$D = \frac{M}{2} = \frac{|\boldsymbol{w}^T \cdot \boldsymbol{x}_2 + b|}{\| \boldsymbol{w} \|} = \frac{1}{\| \boldsymbol{w} \|} \tag{2.14}$$

Esta ecuación (2.14) representa un resultado muy interesante, muestra que la minimización de la norma de los pesos de un vector normal a un hiperplano $\| \boldsymbol{w} \|$ lleva a la maximización del margen M. Debido a que $\sqrt{f}$ es una función monótona[2], la minimización de $\sqrt{f}$ es equivalente a la minimización de f. Consecuentemente, la minimización de la norma $\| \boldsymbol{w} \|$ es igual a la minimización de $\boldsymbol{w^T w} = (\boldsymbol{ww}) = \sum_{i=1}^{n} w_1^2 + w_2^2 + \ldots + w_n^2$. y esto trae como resultado la maximización del margen M.

Con la finalidad de encontrar el hiperplano óptimo que tenga un margen máximo, una máquina de aprendizaje debe minimizar $\| \boldsymbol{w} \|^2$ sujeta a la restricción de desigualdad (2.2.2.1), la cual impone la restricción de que mientras se maximiza el margen los datos deben de estar clasificados correctamente. Si los datos no pueden ser clasificados correctaente con un hiperplano, hasta este punto el problema no tiene solución. Este es un problema clásico de optimización cuadrático con restricciones de desigualdad. Dado lo anterior se formula el siguiente problema de optimización cuadrático:

Minimizar $\quad F(\boldsymbol{w}) = \frac{1}{2} \| \boldsymbol{w} \|^2$

sujeta a:

$\boldsymbol{y_i}(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 \quad i = 1, \ldots, n$

Problema 2.1: Forma primal del problema de optimización cuadrático para optimizar las MSV. Caso linealmente separable y clasificador lineal

En esta etapa el problema puede ser resuelto utilizando técnicas de optimización cuadrática clásicas, sin embargo, para que posteriormente se pueda extender estos resultados a las MSV al caso no lineal se hace uso de la representación dual, para esto utilizamos la técnica de los multiplicadores de Lagrange.

$$L_p = \frac{1}{2} \| \boldsymbol{w} \|^2 - \sum_{i=1}^{n} \lambda_i y_i (\boldsymbol{x_i} \cdot \boldsymbol{w}\cdot + b) + \sum_{i=1}^{n} \lambda_i \tag{2.15}$$

donde $\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_n)$ es un vector con multiplicadores de Lagrange no negativos correspondientes a las restricciones 2.12. La transformación del problema de la forma primal a la forma

[2]Una función es monótona cuando es siempre creciente o bien siempre decreciente.

dual causa que ahora el problema se encuentre en función de una sola variable (λ) a diferencia del problema en su forma primal, el cual se encontraba en función de dos variables ($\boldsymbol{w}$ y b).

La solución de este problema de optimización esta determinado por un punto silla de este Lagrangiano, el cual debe ser minimizado con respecto a $\boldsymbol{w}$ y b, y maximizada con respecto a $\boldsymbol{\Lambda} \geq 0$. Diferenciando (2.15) e igualando el resultado a cero se obtiene:

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\Lambda})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \lambda_i y_i x_i = 0 \tag{2.16}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\Lambda})}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{2.17}$$

Usando el superíndice * para denotar los valores óptimos de la función de costo, de la ecuación (2.16) se tiene que:

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \lambda_i^* y_i x_i = 0 \tag{2.18}$$

la cual muestra que el hiperplano óptimo puede ser escrito como una combinación lineal de los vectores de entrenamiento. Es fácil percatarse que sólo los vectores de entrenamiento con valor de λ estrictamente mayor a cero ($\lambda_i > 0$) contribuyen en la expresión (2.18).

Sustituyendo (2.18) en (2.15) se obtiene que:

$$F(\boldsymbol{\Lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \sum_{i=1}^{n} \lambda_i \Big(y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) - 1 \Big)$$

$$F(\boldsymbol{\Lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \sum_{i=1}^{n} \lambda_i y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) + \sum_{i=1}^{n} \lambda_i$$

$$F(\boldsymbol{\Lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{w} \cdot \boldsymbol{x} + \lambda_i y_i b + \sum_{i=1}^{n} \lambda_i$$

Utilizando (2.17), la expresión se reduce a:

$$F(\boldsymbol{\Lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{w} \cdot \boldsymbol{x} + \sum_{i=1}^{n} \lambda_i$$

Sustituyendo (2.18) en la ecuación anterior:

$$F(\boldsymbol{\Lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \boldsymbol{w} \cdot \boldsymbol{w} + \sum_{i=1}^{n} \lambda_i$$

Como la función es monótona, esta ecuación equivale a:

$$F(\boldsymbol{\Lambda}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 - \parallel \boldsymbol{w} \parallel^2 + \sum_{i=1}^{n} \lambda_i$$

Ordenando lo términos se obtiene la siguiente ecuación:

$$F(\mathbf{\Lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \parallel \boldsymbol{w} \parallel^2$$

$$F(\mathbf{\Lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j \tag{2.19}$$

Escribiendo (2.19) en notación matricial e incorporando las restricciones de no negatividad de Λ y las restricciones de (2.17), se obtiene el siguiente problema cuadrático dual:

Maximizar $F(\mathbf{\Lambda}) = \mathbf{\Lambda} \cdot 1 - \frac{1}{2}\mathbf{\Lambda} \cdot H\mathbf{\Lambda}$

sujeta a:

$$\mathbf{\Lambda} \cdot \boldsymbol{y} = \mathbf{0}$$

$$\mathbf{\Lambda} \geq \mathbf{0}$$

Problema 2.2: Forma dual del problema de optimización cuadrático para optimizar las MSV. Caso linealmente separable y clasificador lineal

donde $\boldsymbol{y} = (y_1, \ldots, y_n)$ y H es una matriz simétrica, llamada Hessiana, de tamaño nxn y con elementos $H_{ij} = y_i y_j (x_i \cdot x_j)$.

A aquellos patrones de entrenamiento que tienen el valor de su correspondiente λ estrictamente mayor a cero se les llama vectores de soporte. Para datos de entrenamiento linealmente separables, todos los vectores de soporte se encuentran el el margen y son generalmente una porción muy pequeña del total de los datos de entrenamiento. Las soluciones λ_i de este problema determinan los parámetros w_0 y b_0 del hiperplano óptimo:

$$\boldsymbol{w}_o = \sum_{i=1}^{n} \lambda_i y_i x_i \quad i = 1, \ldots, n \tag{2.20}$$

$$b_0 = \frac{1}{N_{SV}} \left(\sum_{s=1}^{N_{SV}} \left(\frac{1}{y_s} - \boldsymbol{x_s^T w_0} \right) \right) \tag{2.21}$$

N_{SV} denota el número de vectores de soporte. El vector de pesos $\boldsymbol{w}_o$, así como el bías b_0, son calculados usando únicamente los vectores de soporte. Esto es porque los multiplicadores de Lagrange que no son vectores de soporte son iguales a cero. Finalmente, habiendo calculado $\boldsymbol{w}_0$ y b_0,

se obtiene el hiperplano de decisión $d(\boldsymbol{x})$ y una función indicadora $i_F = sign(d(\boldsymbol{x}))$:

$$d(\boldsymbol{x}) = \sum_{i=1}^{n} w_{0i}x_i + b_0 = \sum_{i=1}^{n} y_i\lambda_i \boldsymbol{x^T x_i} + b_0 \tag{2.22}$$

$$i_F = sign(d(\boldsymbol{x})) \tag{2.23}$$

2.2.2.2. Clasificador lineal y problema no linealmente separable.

El algoritmo presentado en en la sección precedente es válido sólo para datos linealmente separables, eso es, para datos sin traslape. Tales problemas son muy raros en la práctica. La solución del problema cuadrático presentado previamente (problema 2.2) no puede ser usado en el caso de datos con traslape porque la restricción de desigualdad (2.17) no puede ser satisfecha.
En este caso continuamos buscando una superficie de separación lineal, pero ahora se tienen datos no linealmente separables, por lo que no existe un hiperplano de separación, por lo tanto no es posible satisfacer todas las restricciones del problema 2.2. Con la finalidad de tratar con este caso se introduce un nuevo conjunto de variables $\xi_{i\,i=1}^{n}$, que miden la cantidad de violación de las restricciones. Entonces el margen es maximizado pagando una penalización proporcional a la cantidad de restricciones violadas. Formalmente, el problema a resolver es:

$$\begin{aligned} &Minimizar \quad \Phi(\boldsymbol{w}, \boldsymbol{\Xi}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C(\sum_{i=1}^{n} \xi_i)^k \\ &sujeta\ a \\ &\qquad y_i(\boldsymbol{w} \cdot x_i + b) \geq 1 - \xi_i \\ &\qquad \xi_i \geq 0 \end{aligned} \tag{2.24}$$

C es un parámetro seleccionado por el usuario. El incremento de C corresponde a asignar una penalización más alta a los errores, lo que resulta en pesos más grandes. Este es un problema de programación convexo y, escogiendo el exponente $k = 1$, ni las variables de holgura ni los multiplicadores de Lagrange β_i aparecen en el Lagrangiano en su forma dual. Como en el problema linealmente separable, la solución de un problema de optimización cuadrático sujeto a restricciones de desigualdad (2.24), está dada por el puntio silla del Lagrangiano:

$$L_p(\boldsymbol{w}, b, \xi, \boldsymbol{\Lambda}, \boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\left(\sum_{i=1}^{n} \xi_i\right) - \sum_{i=1}^{n} \lambda_i \left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i\right) - \sum_{i=1}^{n} \beta_i\xi_i \tag{2.25}$$

donde λ_i y β_i son los multiplicadores de Lagrange. Este problema puede ser resuelto ya sea en el espacio primal o en el espacio dual (el espacio de los multiplicadores de Lagrange λ_i *y* β_i). Como se procedió en el caso anterior, se encuentra una solución en el espacio dual usando las condiciones de optimalidad para un función con restricciones

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\Lambda})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \lambda_i y_i x_i = 0 \tag{2.26}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\Lambda})}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{2.27}$$

$$\frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\Lambda})}{\partial \xi} = C - \lambda_i - \xi_i = 0 \tag{2.28}$$

Despejando C de (2.28) se tiene:

$$C = \lambda_i + \xi_i \tag{2.29}$$

Las variables del Lagrangiano $F(\boldsymbol{\Lambda})$ ya no se encuentran en función de β, quedando sólo en función de Λ como en el caso linealmente separable.

$$F(\boldsymbol{\Lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i \cdot x_j \tag{2.30}$$

Con la finalidad de encontrar un hiperplano óptimo, la forma dual del lagrangiano tiene que ser maximizada con respecto a λ_i no negativas.

El problema de optimización cuadrático final es prácticamente el mismo que en el caso linealmente separable, la única diferencia es que se modificó la cota superior de los multiplicadores de Lagrange λ_i. El parámetro de penalización C, el cual es ahora el límite superior de sobre λ_i es determinado por el usuario. Es importante notar que el el caso linealmente separable el límite superior = ∞. Expresando este problema dual en forma matricial, se tiene el siguiente problema:

donde $\boldsymbol{y} = (y_1, \ldots, y_n)$ y H es una matriz simétrica, llamada Hessiana, de tamaño nxn y con elementos $H_{ij} = y_i y_j (x_i \cdot x_j)$.

Un punto clave para la extensión a las MSV que se hará en la siguiente sección es notar que la optimización no depende directamente de los vectores de entrada $\boldsymbol{x}_i$, sino únicamente del producto escalar de estos vectores $\boldsymbol{x}_i$.

2.2.2.3. Funciones kernel.

Las máquinas de aprendizaje lineal tienen muchas limitantes en aplicaciones reales, es por eso se han propuesto múltiples métodos que han llevado al desarrollo de redes neuronales con capas

Maximizar $F(\boldsymbol{\Lambda}) = \boldsymbol{\Lambda} \cdot 1 - \frac{1}{2}\boldsymbol{\Lambda} \cdot H\boldsymbol{\Lambda}$

sujeta a:

$$\boldsymbol{\Lambda} \cdot \boldsymbol{y} = \mathbf{0}$$

$$\mathbf{0} \leq \boldsymbol{\Lambda} \leq \boldsymbol{C}$$

Problema 2.3: Forma dual del problema de optimización cuadrático para optimizar las MSV. Caso no linealmente separable y clasificador lineal

múltiples y algoritmos de aprendizaje como retropropagación para el entrenamiento de dichos sistemas.

Una de las ideas básicas en el diseño de las MSV es mapear el vector de entrada $\boldsymbol{x} \in \Re^n$ a un vector $\boldsymbol{z}$ en un espacio de mayor dimensionalidad $F(z = \Phi(\boldsymbol{x}))$ [10] [12] , donde Φ representa el mapeo $\Re^n \longrightarrow \Re^f$), y resolver un problema de clasificación lineal en ese espacio:

$$\boldsymbol{x} \in \Re^n \longrightarrow \boldsymbol{z}(\boldsymbol{x}) = [a_1\phi_1(x), a_2\phi_2(x), \ldots, a_n\phi_n(x)]^T 1n\Re^f \tag{2.31}$$

Con este mapeo se espera que la MSV sea capaz de separar linealmente los elementos de la base de datos. La solución obtenida es la de una clasificación lineal en el espacio F, la cual creará una superficie de separación no lineal en el espacio de entrada original n. Existen dos problemas muy importantes si aplicamos este método: 1) la selección de una función $\Phi(\boldsymbol{x})$ con la que se mapean los datos de entrada y 2) el cálculo de los productos escalares $\boldsymbol{z^T(x)z(x)}$, el cual puede ser computacionalmente muy costoso si la dimensionalidad f del espacio característico es muy grande. El segundo problema tiene que ver con un problema llamado curso de dimensionalida. Por ejemplo, para construir una superficie de decisión que sea una función polinomial de grado 2 en el espacio de entrada, la dimensionalidad en el espacio característico $f = n(n+3)/2$. Así, la construcción de un polinomio de grado 2 en un espacio de entrada, lleva a una dimensionalidad en el espacio característico $f = 33,152$. Realizar el producto escalar en el espacio característico es una tarea muy difícil.

Esta explosión en dimensionalidad puede ser evitada si se nota que el problema de optimización cuadrática dado por (2.30), así como la expresión de decisión dada por el clasificador (2.22), los datos de entrenamiento aparecen sólo en la forma de productos escalares $\boldsymbol{x_i^T x_j}$. Estos productos son reemplazados, en un espacio característico F, por los productos escalares $\boldsymbol{z_i^T z_j} = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})][\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_n(\boldsymbol{x})]^T$. Esta última expresión puede ser expresada

utilizando la función kernel [10] [12]:

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{z_i^T}, \boldsymbol{z_j} = \Phi^T(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_j) \tag{2.32}$$

La función kernel trabaja en el espacio de entrada. Así, la principal ventaja de utilizar funciones kernel es que se evita realizar un mapeo $\Phi(\boldsymbol{x})$. En lugar de realizar este mapeo, los productos escalares requeridos en el espacio característico $\Phi^T(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_j)$ son calculados directamente por la función kernel $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ para los datos de entrenamiento en el espacio de entrada.

Lineal	$k = u * v'$
Polinomial de grado d	$k = (u * v' + 1)^d$
Radial Basis Function	$exp(-(u - v) * (u - v)'/(2 * \sigma^3))$;

Cuadro 2.4: Ejemplos de funciones kernel

2.2.2.4. Clasificador no lineal y problema no linealmente separable

Utilizando las funciones kernel descritas en la sección anterior, se puede construir una MSV que trabaje en un espacio característico con una dimensión muy alta, incluso infinita. Reformulando, la solución de una MSV tiene la siguiente forma:

$$f(\boldsymbol{x}) = sign\left(\sum_{i=1}^{n} y_i \lambda_i^* K(\boldsymbol{x}, \boldsymbol{x}_i) + b^*\right) \tag{2.33}$$

Esta superficie de decisión es una función no lineal, dada por una superposición lineal de funciones kernel, una por cada vector de soporte.
El problema de programación cuadrática queda de la siguiente forma:
donde H es una matriz llamada Hessiana, la cual es simétrica, semidefinida positiva, de tamaño nxn y con la diferencia de que para calcularla los datos de entrada x son elevados a otra dimensión mediante las funciones kernel: $H_{ij} = y_i y_j K(x_i, x_j)$.

Ventajas de las Máquinas de Soporte Vectorial

- Alto nivel de generalización garantizado matemáticamente.
- El entrenamiento mediante la solución del problema de optimización cuadrática nos asegura el óptimo global, caso contrario a los algoritmos de redes neuronales donde lo más común es llegar a un óptimo local.

$$\begin{aligned} &\text{Maximizar} \quad F(\boldsymbol{\Lambda}) = \boldsymbol{\Lambda} \cdot 1 - \tfrac{1}{2}\boldsymbol{\Lambda} \cdot H\boldsymbol{\Lambda} \\ &\text{sujeta a:} \\ &\qquad \boldsymbol{\Lambda} \cdot \boldsymbol{y} = \mathbf{0} \\ &\qquad \boldsymbol{\Lambda} \geq \mathbf{0} \end{aligned}$$

Problema 2.5: Forma dual del problema de optimización cuadrático para optimizar las MSV. Caso linealmente separable y clasificador no lineal

Desventajas de las Máquinas de Soporte Vectorial

- Para problemas de gran dimensión (mayores a 10,000 datos) el tiempo de aprendizaje puede ser muy extenso como para ser aplicadas en muchas aplicaciones reales. Se han desarrollado múltiples algoritmos [2] para acelerar el tiempo de entrenamiento y resolver el problema de almacenamiento cuando el problema es de gran dimensión. Sin embargo, aunque estos algoritmos aceleran de manera muy significativa el tiempo de entrenamiento y resuelven de manera muy eficiente el problema de memoria, existen muchas heurísticas y mejoras que se pueden realizar para aumentar la velocidad de entrenamiento. En este trabajo de tesis se buscó implementar un método híbrido que conste de una primera parte donde se busque una solución aproximada de manera muy rápida y después alcance la solución óptima exacta mediante un algoritmo tradicional.

2.2.3. ALGORITMO DE DESCOMPOSICIÓN *CHUNKING*.

El algoritmo de *Chunking*, propuesto por Valdimir Vapnik, se fundamenta en que la solución obtenida del problema de programación cuadrática generado por una SVM es la misma si el problema se resuelve con la matriz Hessiana construida con todos los elementos de la base de aprendizaje que si se resuelve con una matriz Hessiana calculada sólo con aquellos valores de patrones cuyo lagrangiano sea diferente de cero (vectores de soporte). Una de las principales desventajas que se presenta en este algoritmo es que cuando el número de vectores de soporte es muy grande es necesario construir una matriz Hessiana de gran tamaño que puede llegar a ser imposible de almacenar en memoria.

El algoritmo de *Chunking* se puede resumir de la siguiente forma:

1. Seleccionar q elementos aleatorios de la base de aprendizaje y formar el conjunto de trabajo.
2. Realizar la optimización del conjunto de trabajo mediante algún método de optimización cuadrático.
3. Identificar loa datos cuyo multiplicador de Lagrange es diferente de cero ($\lambda \neq 0$) (vectores de soporte) y agregarlos al conjunto de trabajo.
4. Identificar aquellos patrones de la base de datos que se encuentren mal clasificados y agregarlos al conjunto de trabajo.
5. Si aún se encontraron elementos mal clasificados, regresar al punto 3. En caso contrario, terminar el algoritmo.

Cuadro 1.6 Algoritmo de descomposición *Chunking*.

Se asegura la convergencia del algoritmo ya que en cada iteración el hiperplano de separación se mueve en la dirección de aquellos patrones que no cumplen las condiciones de optimalidad hasta lograr que el hiperplano quede lo mejor posicionado posible.

2.3. ALGORITMOS GENÉTICOS

Los algoritmos genéticos son algoritmos de búsqueda basados en mecanismos de la selección natural y de la genética. Combinan la supervivencia del individuo más apto entre las estructuras de las cadenas con un intercambio de información aleatoria para formar un algoritmo de búsqueda [26]. En cada generación, un nuevo conjunto de individuos artificiales (cadenas) es creado usando bits y piezas de los individuos anteriores con mayor aptitud. Los algoritmos genéticos no realizan simplemente una búsqueda aleatoria, ellos explotan eficientemente información histórica para especular en nuevos puntos de búsqueda con la finalidad de encontrar puntos con un valor de aptitud mejor.

Los algoritmos genéticos fueron desarrollados por John Holland. Los objetivos de su investigación son: 1) Abstraer y explicar rigurosamente el proceso de adaptación natural y 2) diseñar sistemas artificiales que empleen el mecanismo de los sistemas naturales. Esto ha llevado a descubrimientos importantes en las ciencias naturales y en la ciencia de los sistemas artificiales.

El tema central de la investigación en algoritmos genéticos ha sido la robustez, el balance entre la eficiencia y la eficacia necesaria para la supervivencia en muchos medios diferentes. Si los sistemas artificiales tienen robustez, el costo de rediseños pueden ser reducido o eliminado. Los algoritmos genéticos han probado teórica y empíricamente que producen una búsqueda robusta en espacios complejos.

2.3.1. ALGORITMO GENÉTICO SIMPLE.

Los mecanismos de un algoritmo genético son muy simples. Consisten únicamente en copiar cadenas e intercambiar partes de cadenas. La simplicidad en la operación y el poder del efecto producido son dos de las principales atracciones de los algoritmos genéticos. El algoritmo genético comienza con una población inicial generada aleatoriamente. Esta población inicial es modificada por un conjunto de operaciones simples que toman esta población inicial y generan poblaciones sucesivas que se espera mejoren a través del tiempo.

El algoritmo genético simple, con el que se han obtenido buenos resultados en muchos problemas, se compone de tres operadores: selección, cruza y mutación.

2.3.1.1. Selección.

La reproducción es un proceso en el cual las cadenas son copiadas de acuerdo a los valores de su función objetivo f. Intuitivamente se puede pensar en la función f como la medida de aptitud,

utilidad o mejora que se quiere maximizar. Copiar la cadena de acuerdo a sus valores de aptitud significa que las cadenas con un mayor valor de aptitud tienen mayor probabilidad de contribuir en una o más descendencias de las siguientes generaciones. Este operador es una versión artificial de la seleeción natural, una estrategia de supervivencia Darwiniana del individuo más apto. En las poblaciones naturales, la aptitud es determinada por la habilidad del individuo de sobrevivir a depredadores, clima, y otros obstáculos que le impide crecer y reproducirse.
El operador de reproducción puede ser implementado computacionalmente en muchas formas. Uno de los procedimientos más utilizado es el denominado ruleta, en donde cada individuo tiene una sección circular de una ruleta que es directamente proporcional a su calidad. De esta forma, las cadenas con mayor aptitud tienen un número mayor de descendencia en las generaciones sucesivas. Cada vez que una cadena es seleccionada para reproducirse, se realiza una réplica exacta de dicha cadena.

2.3.1.2. Cruza.

Después de la selección, el operador de cruza procede en dos pasos: primero, miembros de la población seleccionada son escogidos al azar y, segundo, se hace intercambio de bits. Los operadores de cruzamiento más utilizados son:

- De un punto: Se elige aleatoriamente un punto de ruptura en los padres y se intercambian sus bits.
- De dos puntos: Se eligen dos puntos de ruptura al azar para intercambiar.
- Uniforme: En cada bit se elige al azar un padre para que contribuya con su bit al del hijo, mientras que el segundo hijo recibe el bit del otro padre.
- PMX, SEX: Son operadores más sofisticados que consisten en mezclar y aleatorizar los anteriores.

2.3.1.3. Mutación.

En el algoritmo genético simple, la mutación es la alteración aleatoria ocasional (con probabilidad pequeña) de los valores de una posición de la cadena. En la codificación binaria, esto significa simplemente el cambio de 1 a 0 y viceversa. La mutación es necesaria porque, aun cuando la selección y cruza buscan y recombinan efectivamente, ocasionalmente tienden a estabilizarse y realizar la búsqueda en un espacio muy limitado donde posiblemente no se encuentre el óptimo global.

La frecuencia de mutación para obtener buenos resultados debe de ser muy pequeña. Las tasas de mutación en las poblaciones naturales también son muy pequeñas, es por eso que la mutación es considerada como un mecanismo secundario en la adaptación de los algoritmos genéticos.

Ventajas de los Algoritmos Genéticos

- Pueden ser usados en problemas donde no conozcamos alguna forma para resolverlos pero se tenga la posibilidad de evaluar la solución. Con los algoritmos genéticos se puede optimizar cualquier problema en el cual se pueda medir la aptitud de la función objetivo. Esto permite que los algoritmos genéticos sean muy poderosos ya que pueden ser utilizados para resolver muchos problemas de los cuales no se conoce otra forma de resolverlos.
- Son robustos.
- Pueden combinarse con otros algoritmos para formar métodos híbridos.

Desventajas de los Algoritmos Genéticos

- Son lentos, aunque en algoritmos híbridos han llegado a acelerar algunos métodos [21] [22].
- Muchas veces los algoritmos genéticos son usados indiscrimidadamente para resolver todo tipo de problemas sin importar que se conozca otra forma de resolverlo. Esto ha causado que los algoritmos genéticos muchas veces no sean bien vistos, ya que para dichos problemas pueden existir formas mucho más eficientes (menor tiempo de optimización y mayor exactitud del resultado) para resolverlos. Por lo anterior es muy importante identificar que tipos de problemas son los indicados para ser resueltos por algoritmos genéticos y cuales no. Si los algoritmo genéticos son utilizados dentro de un método híbrido normalmente sólo se permite que evolucionen pocas generaciones para obtener una solución aproximada para después hacer una búsqueda partiendo de esa solución con algún algoritmo clásico.

2.4. MÉTODO DE ZOUTENDIJK

El método de Zoutendijk [23] pertenece a la clase de métodos de direcciones factibles. Esta clase de métodos resuelve problemas de programación no lineal moviéndose de un punto factible a otro mejor. La siguiente estrategia es típica de los algoritmos de direcciones factibles: dado un punto factible $\boldsymbol{x}_k$, una dirección $\boldsymbol{d}_k$ es calculada tal que para $h > 0$ suficientemente pequeña, las siguientes dos propiedades son verdaderas: 1) $\boldsymbol{x}_k + h\boldsymbol{d}_k$ es factible, y 2) el valor de la función objetivo en $\boldsymbol{x}_k + h\boldsymbol{d}_k$ es mejor que el valor de la función objetivo en $\boldsymbol{x}_k$. Después de que dicha dirección es determinada, es necesario resolver un problema de optimización unidimensional para determinar el tamaño de h. Esto lleva a un nuevo punto $\boldsymbol{x}_{k+1}$ y el proceso se repite. El método de Zoutendijk puede tratar con restricciones y no lineales, se mostrará unicamente el caso de restricciones lineales ya que es el único tipo de restricciones con las que cuentan las MSV.

Método de Zoutendijk (caso con restricciones lineales)
Con la finalidad de resolver un problema no lineal de la forma:

$$\begin{aligned} &Maximizar && f(\boldsymbol{x}) \\ &sujeta\ a && \\ & && \boldsymbol{Ax} \leq \boldsymbol{b} \\ & && \boldsymbol{Ex} = \boldsymbol{e} \end{aligned} \tag{2.34}$$

Este método sigue el procedimiento indicado en el cuadro 1.7:

1. Encontrar $\boldsymbol{x_1}$ con $\boldsymbol{A_1 x_1 = b_1}$, $\boldsymbol{A_2 x_2 < b_2}$, particionando $\boldsymbol{A^T = \{A_1^T, A_2^T\}}$ y $\boldsymbol{b = \{b_1, b_2\}}$. Hacer k=1.

2. Dado $\boldsymbol{x_k}$, $\boldsymbol{A_1 x_k = b_1}$ and $\boldsymbol{A_2 x_k < b_2}$, encontrar la dirección $\boldsymbol{d_k}$, que se obtiene de la solución óptima de:

$$Maximizar \qquad \nabla f(\boldsymbol{x})^T d \tag{2.35}$$

$$sujeta\ a \tag{2.36}$$

$$\boldsymbol{A_1 d \leq 0} \tag{2.37}$$

$$\boldsymbol{Ed = 0} \tag{2.38}$$

3. Si $\nabla f(\boldsymbol{x_k})^T \boldsymbol{d}$ = 0. Parar. Si no, ir al paso 4.

4. Encontrar el tamaño del paso h_k, el cual resulta de solucionar el siguiente problema:

$$Maximizar \qquad \nabla f(\boldsymbol{x_k} + h\boldsymbol{d}) \tag{2.39}$$

$$sujeta\ a \tag{2.40}$$

$$\boldsymbol{0} \leq h \leq h_{max} \tag{2.41}$$

donde

$$h_{max} = \begin{cases} min_{d_1 > 0} & \text{si } d \leq 0 \\ \infty & \text{si } d \leq 0 \end{cases}$$

Cuadro 1.7 Algoritmo de Zoutendijk.

2.4.1. DESARROLLO DEL MÉTODO DE ZOUTENDIJK PARA LAS MÁQUINAS DE SOPORTE VECTORIAL.

El problema de optimización cuadrática que es necesario resolver para entrenar las MSV no cuenta con restricciones de desigualdad, por este motivo el paso 1 del método de Zoutendijk no se realiza y el paso 2 queda de la siguiente forma:

$$
\begin{aligned}
&Maximizar \quad f(\boldsymbol{d}) = \mathbf{1} - (\boldsymbol{dH\Lambda^T}) \\
&sujeta\ a \\
&\qquad \boldsymbol{y \cdot d} = \mathbf{0} \\
&\qquad \mathbf{1} \leq \boldsymbol{d} \leq \mathbf{0} \quad si\ \Lambda_i = C \\
&\qquad \mathbf{0} \leq \boldsymbol{d} \leq \mathbf{1} \quad si\ \Lambda_i = 0 \\
&\qquad -\mathbf{1} \leq \boldsymbol{d} \leq \mathbf{1} \quad de\ otra\ forma
\end{aligned}
\tag{2.42}
$$

Para encontrar el tamaño del paso:

De la serie de Taylor se tiene:

$$f(\boldsymbol{\Lambda} + h\boldsymbol{\Lambda}) - f(\boldsymbol{\Lambda}) = \nabla f(\boldsymbol{\Lambda})^T \boldsymbol{d} + \frac{1}{2}\, \boldsymbol{d} h \boldsymbol{H} \boldsymbol{d}^T h = 0 \tag{2.43}$$

Dividiendo (2.43) entre h se tiene:

$$f(\boldsymbol{\Lambda} + h\boldsymbol{\Lambda}) - f(\boldsymbol{\Lambda}) = \nabla f(\boldsymbol{\Lambda})^T \boldsymbol{d} + \frac{1}{2}\, \boldsymbol{d} h \boldsymbol{H} \boldsymbol{d}^T = 0 \tag{2.44}$$

Estando en el punto óptimo sabemos que $f(\boldsymbol{\Lambda} + h\boldsymbol{\Lambda}) - f(\boldsymbol{\Lambda}) = 0$, por lo tanto:

$$\frac{1}{2}\, \boldsymbol{d} h \boldsymbol{H} \boldsymbol{d}^T = 0 = \nabla f(\boldsymbol{\Lambda})^T \boldsymbol{d} \tag{2.45}$$

Despejamos h y se obtiene el paso óptimo

$$h = \frac{2\boldsymbol{d} \cdot \mathbf{1} - \boldsymbol{dH\Lambda}^T}{\boldsymbol{d}^T \boldsymbol{H} \boldsymbol{d}} \tag{2.46}$$

Finalmente, el algoritmo de Zoutendijk para optimización de MSV se resume en el siguiente cuadro:

1. Dada una solución factible $\boldsymbol{x_k}$, encontrar la dirección $\boldsymbol{d_k}$ que se obtiene de la solución óptima de:

$$Maximizar \quad f(\boldsymbol{d}) \quad = \mathbf{1} - (\boldsymbol{dH\Lambda}^T) \tag{2.47}$$

$$sujeta\ a \tag{2.48}$$

$$\boldsymbol{y \cdot d} = \mathbf{0} \tag{2.49}$$

$$\mathbf{1} \leq \boldsymbol{d} \leq \mathbf{0} \qquad si\ \Lambda_i = C \tag{2.50}$$

$$\mathbf{0} \leq \boldsymbol{d} \leq \mathbf{1} \qquad si\ \Lambda_i = 0 \tag{2.51}$$

$$-\mathbf{1} \leq \boldsymbol{d} \leq \mathbf{1} \qquad de\ otra\ forma \tag{2.52}$$

2. Si $\nabla f(\boldsymbol{x_k})^T \boldsymbol{d}$ = 0. Parar. Si no, ir al paso 3.

3. Encontrar el tamaño del paso h_k, el cual resulta de solucionar el siguiente problema:

$$h_k = \frac{2\boldsymbol{d} \cdot \mathbf{1} - \boldsymbol{dH\Lambda}^T}{\boldsymbol{dHd^T}} \tag{2.53}$$

Cuadro 1.8 Algoritmo de Zoutendijk para el caso de MSV.

2.5. MÉTODO SIMPLEX REVISADO ACOTADO

El método Simplex es un procedimiento sistemático para generar y probar soluciones candidatas de los vértices en un problema de programación lineal. Comienza en una esquina arbitraria del conjunto de soluciones. En cada iteración, el método Simplex selecciona la variable que producirá el cambio más grande hacia un mínimo o máximo. Esta variable, que se encuentra en el conjunto de variables no básicas, reemplaza a aquella que la restrinja más, ésto hace que el algoritmo se mueva a otra deferente esquina del conjunto de soluciones. Por las propiedades de convexidad de los problemas de programación lineal, cuando el algoritmo no puede encontrar una mejor solución que en la que se encuentra es porque está en el óptimo global. En cada iteración del método Simplex [24], una solución básica factible $\boldsymbol{x}_B$ es reemplazada por otra.

El método Simplex Revisado describe el problema de programación lineal en términos matriciales y presenta al método Simplex como una serie de cálculos algebraicos lineales. Cada iteración del método Simplex Revisado puede o no tomar menos tiempo que la correspondiente iteración del método Simplex. El resultado de esta comparación depende no sólo de la implementación particular del método Simplex Revisado sino que también de la naturaleza de los datos. En problemas

de programación lineal grandes y con esparcidad, el método Simplex Revisado trabaja más rápido que el método Simplex estándard.

Primero, el problema de programación lineal tiene que presentarse en forma matricial:

$$maximizar \qquad \boldsymbol{c}x \tag{2.54}$$

$$sujeta\ a: \tag{2.55}$$

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{2.56}$$

$$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{2.57}$$

En el sistema $\boldsymbol{Ax} = \boldsymbol{b}$ hay variables básicas y no básicas. Para enfatizar que sólo las variables básicas son tratadas como desconocidas, escribimos $\boldsymbol{Ax}$ como $\boldsymbol{A_B x_B} + \boldsymbol{A_N x_N}$ y el sistema $\boldsymbol{Ax} = \boldsymbol{b}$ queda como

$$\boldsymbol{A_B x_B} = \boldsymbol{b} - \boldsymbol{A_N x_N} \tag{2.58}$$

Multiplicando ambos lados de 2.58 por $\boldsymbol{A_B^{-1}}$ se tiene que:

$$\boldsymbol{x_B} = \boldsymbol{A_B^{-1} b} - \boldsymbol{b} - \boldsymbol{A_B^{-1} A_N x_N} \tag{2.59}$$

Sabemos que la función objetivo z es igual a $\boldsymbol{cx}$, o para distinguir de nuevo las variables básicas y las no básicas $z = \boldsymbol{c_B x_B} + \boldsymbol{C_N x_N}$. Sustituyendo $\boldsymbol{x_B}$ de (2.59) se obtiene:

$$z = \boldsymbol{c_B A_B^{-1}} + (\boldsymbol{c_N} - \boldsymbol{c_B A_B^{-1} A_N})\boldsymbol{x_N} \tag{2.60}$$

1. Resolver el sistema $\boldsymbol{yB} = \boldsymbol{c}_B$

2. Escoger la variable entrante x_j. Puede ser cualquier variable no básica tal que, para $\boldsymbol{a}$ correpondiente a $\boldsymbol{A}$, se tiene que $\boldsymbol{ya} < c_j,\ x_j^* < u_j$ o $\boldsymbol{ya} > c_j,\ x_j^* > l_j$. Si no existe esta variable, parar, la solución actual $\boldsymbol{x}_i$ es óptima.

3. Resolver el sistema $\boldsymbol{Bd} = \boldsymbol{a}$

4. Definir $x_j(t) = x_j^* + t$ y $\boldsymbol{x}_B$+t$\boldsymbol{d}$ cuando $\boldsymbol{ya} < c_j$ y $x_j(t) = x_j^* - t$ y $\boldsymbol{x}_B$+t$\boldsymbol{d}$ cuando $\boldsymbol{ya} > c_j$. Si las restricciones

$$l_j \leq x_j(t) \leq u_j,\ \ \boldsymbol{l} \leq \boldsymbol{x}_B(t) \leq \boldsymbol{u}_B$$

   son satisfechas para toda t positiva, parar, el problema es no Acotado. De otra manera, dar a t el valor máximo permitido por estas restricciones. Si la cota superior impuesta en t por las restricciones $\boldsymbol{l} \leq \boldsymbol{x}_B(t) \leq \boldsymbol{u}_B$ es más estricta que la cota superior impuesta por $l_j \leq x_j(t) \leq u_j$, determinar la variable saliente. Puede ser cualquier variable básica x_j tal que la cota superior impuesta en t por $l_j \leq x_j(t) \leq u_j$ es tan estricta comola cota superior impuesta por todas las restricciones en $\boldsymbol{l} \leq \boldsymbol{x}_B(t) \leq \boldsymbol{u}_B$

5. Reemplazar x_j^* por $x_j(t)$ y $\boldsymbol{x}_B^*$ por $\boldsymbol{x}_B(t)$. Si el valor de la variable entrante x_j sólo cambio de una de seu cotas a otra, ir directamente al paso 2 en la siguiente iteración. De otra forma, remplazar la variable saliente x_i por la variable entrante x_j y remplazar la columna saliente de $\boldsymbol{B}$ por la columna entrante $\boldsymbol{a}$

Cuadro 1.9 Algoritmo Simplex Revisado Acotado.

Hasta este punto el algoritmo Simplex sólo puede resolver el problema de programación lineal si las restricciones del problema son de desigualdad. Dado que el problema para el caso de las MSV cuenta con una restricción de igualdad, la dificultad restante es encontrar una solución factible inicial. Esta dificultad puede ser vencida con el método Simplex de dos fases. Primero las restricción $\boldsymbol{Ax} = \boldsymbol{b}$ es extendida por medio de variables artificiales, haciendo que haya solución básica factible disponible para el problema extendido y posteriormente, mediante un problema auxiliar, los valores de estas variables son llevadas para encontrar una solución básica factible. A la solución de este problema se le llama fase I y el resultado obtenido es la solución básica factible con la que inicializamos el método Simplex Revisado, que en este método se le llama fase II.

3. DESARROLLO DE ALGORITMOS DE OPTIMIZACIÓN PARA MÁQUINAS DE SOPORTE VECTORIAL

Durante el desarrollo de este trabajo de tesis existieron dos ideas principales para reducir el tiempo de entrenamiento en las MSV:

1. Realizar modificaciones al optimizador que se ocupa para resolver el problema de optimización cuadrático, adaptándolo a los requerimientos específicos de este problema con la finalidad de aumentar la velocidad del entrenamiento, y

2. Ya que sabemos que si entrenamos la MSV con los datos que se encuentren sobre los hiperplanos que representan el margen de separación, la segunda idea consiste en hacer un pretratamiento de los datos para obtener un subconjunto de entrenamiento dentro del cual se encuentren los vectores de soporte.

Para llevar a cabo la primera idea, primero se trató de realizar una linealización del problema por medio del método de Dantzig-Wolfe. Después se realizaron modificaciones al algoritmo de Zoutendijk. Este algoritmo, al igual que el anterior linealiza el problema, esto lo hace resolviendo iterativamente el problema a través de aproximaciones lineales. Para la solución de los problemas lineales que se van generando se emplea el algoritmo Simplex. Esto resultó de gran atracción ya

que se conocen los métodos Simplex Genético y Simplex Annealing, descritos en [21] y [22], que aceleran de manera significativa el método Simplex y por consiguiente se esperaba una reducción importante en el tiempo de optimización que emplea el algoritmo de Zoutendijk. Como se mostrará en la sección 3.2 el problema del algoritmo de Zoutendijk consiste en que existen etapas en la optimización donde el algoritmo oscila y eso hace que tarde en converger hacia la solución, es por eso que se elaboró un algoritmo híbrido donde se localiza cuando el algoritmo comienza a oscilar y la solución obtenida en ese momento es utilizada como solución inicial para el algoritmo de optimización cuadrática Quadprog.

Al llevar a cabo la segunda idea, primero se verificó el trabajo realizado en [27] y [28], donde se realiza un pretratamiento de los datos con algoritmos genéticos, esto con la finalidad de encontrar los datos que se encuentren más cerca del hiperplano de separación, dando a los datos más lejanos una penalización que permita que los datos más cercanos obtener un mejor evaluación de aptitud. Como el tiempo empleado para realizar el pretratamiento con los Algoritmos Genéticos se buscó otra forma para realizar el pretratamiento. Para esto retomamos al algoritmo de Zoutendijk, el cual es ahora es empleado con una penalización muy baja para obtener una solución aproximada donde es posible identificar cuales datos tienen una mayor posibilidad de comportarse como vectores de soporte. El entrenamiento es realizado posteriormente sólo con esos datos. A este nuevo método se le llamó método ZQP.

El desarrollo de estos dos puntos se describe a continuación. Todos los experimentos se hicieron utilizando como lenguaje de programación Matlab y se llevaron a cabo en una computadora Intel(R) Pentium(R) 4 a 2.40 GHz con 1 GB de memoria RAM.

3.1. ACELERACIÓN DEL ALGORITMO DE ZOUTENDIJK.

Una primera idea para resolver el problema de optimización cuadrática generado por las MSV fue mediante la utilización de la regla Dantzig-Wolfe. Esta regla consiste en hacer aproximaciones lineales por medio de las series de Taylor. Como se muestra en la figura 3.1, la idea de este algoritmo es que, comenzando de un punto inicial x_0, se calcula la dirección hacia la que cual la función crece más rápidamente y se avanza un paso muy pequeño hacia esta dirección. Para calcular la dirección es necesario resolver el problema de programación lineal obtenido por medio de aproximaciones lineales con ayuda de las series de Taylor

$$f(x_{k+1}) = f(x_k) + \frac{\partial f}{\partial x_i}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2 \tag{3.1}$$

sujeta a las restricciones del problema de programación cuadrática.

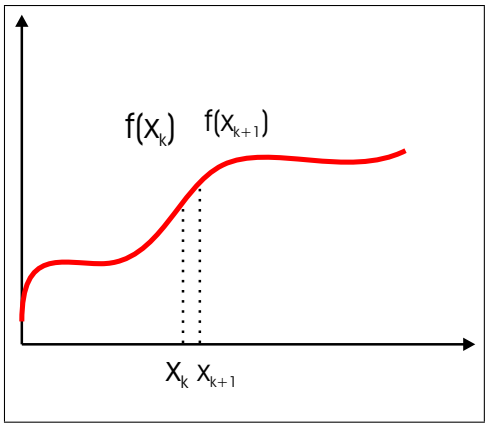


Figura 3.1: Aproximaciones lineales

Este algoritmo no pudo ser implementado para el caso de las MSV debido a que al hacer la linealización el problema a resolver es un problema no acotado.

El algoritmo de Zoutendijk resuelve el problema encontrado con la regla de Dantzig-Wolfe. Este algoritmo parte de que para encontrar una dirección factible de mejora es necesario minimizar la función $\nabla f(\boldsymbol{x})^t \boldsymbol{d}$ sujeta, en el caso de las SVM, a las restricciones $\boldsymbol{Ad} = \boldsymbol{0}$. Sin embargo, si un vector $\boldsymbol{d}$ existe, entonces el valor de la función objetivo es ∞. Así, se introuce una restricción que afecta al vector $\boldsymbol{d}$. Tal restricciónes usulmente referida como restricción de normalización. Esta restricción se encuentra en el algoritmo presentado en el cuadro 1.8.

El algoritmo de Zoutendijk se compone de tres etapas fundamentales, por el orden en el que se llevan a cabo en el algoritmo, son las siguientes:

1. Se calcula una dirección factible, esta se lleva a cabo resolviendo un problema de optimización lineal, la cual se lleva a cabo con el método Simplex acotado,

2. Se prueba si el punto en el que se encuentra el algoritmo se encuentra en el óptimo, esto se hace verificando si es un punto Karush-Kuhn-Tucker, si se cumple esta condición, el algoritmo se detiene y se tiene que la solución del problema, de otra forma, se continua con el punto 3.

3. Se calcula el tamaño del paso para avanzar en la dirección obtenida se continua con el punto 1.

Como se comentó anteriormente la idea es optimizar el algoritmo Simplex, para esto se siguieron los pasos que se explican enseguida:

3.1.1. IMPLEMENTACIÓN DEL ALGORITMO DE ZOUTENDIJK CON EL ALGORITMO SIMPLEX A LA MEDIDA.

En este trabajo se le llamará algoritmo Simplex a la medida al algoritmo Simplex que se adapta a los requerimientos específicos para resolver el problema de optimización lineal obtenido en cada iteración del algoritmo de Zoutendijk.
Primero se implementó el algoritmo de Zoutendijk con el Simplex de Linprog con la finalidad de tener un punto de comparación con las mejoras realizadas.
Las mejoras realizadas al algoritmo Simplex de Linprog se pueden dividir en dos partes:

1. Se eliminaron todas las condiciones innecesarias que utiliza el Simplex de Linprog, ya que el problema a resolver tiene características particulares y siempre las sigue.

2. Como se vió anteriormente, para calcular la dirección dentro del algoritmo de Zoutendijk es necesario resolver un problema de optimización lineal con ciertas características particulares, este problema se reescribe a continuación:

$$
\begin{aligned}
&Maximizar \quad f(\boldsymbol{d}) = \mathbf{1} - (\boldsymbol{d}\boldsymbol{H}\boldsymbol{\Lambda}^T) \\
&sujeta\ a \\
&\qquad \boldsymbol{y}\cdot\boldsymbol{d} = \mathbf{0} \\
&\qquad \mathbf{1} \leq \boldsymbol{d} \leq \mathbf{0} \quad si\ \Lambda_i = C \\
&\qquad \mathbf{0} \leq \boldsymbol{d} \leq \mathbf{1} \quad si\ \Lambda_i = 0 \\
&\qquad -\mathbf{1} \leq \boldsymbol{d} \leq \mathbf{1} \quad de\ otra\ forma
\end{aligned}
\tag{3.2}
$$

   Este problema consta de cotas superiores e inferiores y sólo cuenta con una restricción de igualdad, esto debido a que las cotas impuestas por este algoritmo a las variables $\boldsymbol{d}$ son manejadas implícitamente por el método Simplex Revisado Acotado, de forma similar a como maneja la restricción de que el resultado siempre sea igual o mayor a cero. Al resolver este problema con el método Simplex Revisado Acotado de dos fases, es fácil notar que se puede eliminar la fase I del algoritmo Simplex y empezar con una solución inicial fija que siempre sea factible. En este caso una solución que siempre es factible es el vector de ceros, ya que éste, multiplicado por el vector Y siempre siempre tendrá como resultado el valor de cero. Como la fase I del algoritmo Siplex encuentra una solución factible inicial cualquiera, se espera que con esta inicialización se reduzca el tiempo para resolver el problema lineal.

3.2. MÉTODO HÍBRIDO: ZOUTENDIJK Y QUADPROG

Al obtener los resultados de las implementaciones anteriores, se observó que las modificaciones realizadas al algoritmo de Zoutendijk tuvieron como efecto una disminución significativa en el tiempo de optimización. Sin embargo, comparando los resultados con el algoritmo Quadprog se pudo notar que este tiempo todavía es muy extenso. Esto se debe a que, tal como lo menciona Osuna en [2], este algoritmo oscila mucho y eso consume demasiado tiempo para llegar a la solución. Observando el comportamiento del algoritmo con respecto a que se cumpla la condición de optimalidad $\nabla f(\Lambda) \cdot d = 0$, se puede notar que existe una optimización muy rápida en las primeras iteraciones y después comienza a oscilar hasta llegar a la solución final, este comportamiento se puede ver en la siguiente figura:

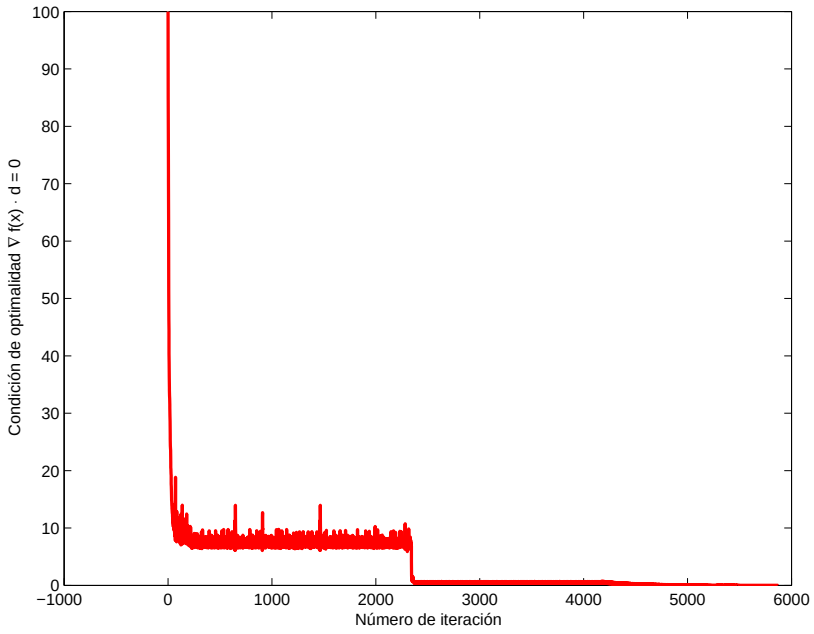


Figura 3.2: Comportamiento del algoritmo de Zoutendijk

En la gráfica se pueden advertir zonas donde el algoritmo optimiza de una manera efectiva y otras zonas donde el algoritmo oscila, es por eso que se elaboró un método híbrido que consiste de las siguientes partes:

- **Comenzar el funcionamiento del método con el algoritmo Zoutendijk.**
- **Identificar cuando comienza el algoritmo a oscilar.** Para identificar esta etapa se guardó un registro de los resultados de la prueba de optimalidad anterior a la actual, después se com-

paró con la actual y si la anterior tiene un valor menor, es indicativo de que el algoritmo de Zoutendijk está oscilando.

- **Cuando comience a oscilar, utilizar el optimizador cuadrático Quadprog.** Una vez que se detectó que el algoritmo de Zoutendijk está oscilando, la solución obtenida por este algortimo es utilizada como solución inicial por Quadprog.

3.3. PRETRATAMIENTO DE LOS DATOS CON ALGORITMOS GENÉTICOS

En [27] y [28] se aborda la solución del problema de programación cuadrática mediante el uso de Algoritmos Genéticos. Proponen hacer que las soluciones cumplan con las restricciones de desigualdad mediante una representación que no admita valores negativos, y las restricciones de desigualdad las incorporan a la función objetivo mediante una penalización dinámica. La finalidad es que el algoritmo Genético prediga cuales de los vectores de entrenamiento serán los vectores de soporte. En los resultados obtenido en el artículo muestran que que los vectores de soporte obtenidos con el Algoritmo Genético son un superconjunto de los vectores de soporte reales.

$$f(\Lambda) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j\, k(x_i, x_j) + \sum_{i=1}^{n}\lambda_i \tag{3.3}$$

La restricción de igualdad la incorporan a la función objetivo mediante una penalización dinámica. Dicha restricción es una igualdad a cero, así, dependiendo de que tan lejos esté el valor de la restricción de cero, podemos evaluar 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 unfactor 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 más cromosomas tendrán penalización cero. La función de aptitud con la estrategia de penalizaciones dinámicas queda expresada de la siguiente forma:

$$aptitud(\Lambda) = f(\Lambda) - C\left(\frac{t}{G}\right)R(\Lambda) \tag{3.4}$$

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

$$R(\Lambda) = \sum_{i=1}^{n}\lambda_i y_i \tag{3.5}$$

3.3.1. IMPLEMENTACIÓN DEL ALGORITMO GENÉTICO

La función de aptitud, la representación de la solución como cromosomas, la forma de crear una población inicial, los operadores genéticos y los parámetros utilizados (número máximo de generaciones, tamaño de la población) fueron implementados exactamente como se describe en [27]. A continuación se resumen los componentes del Algoritmo Genético implementados:

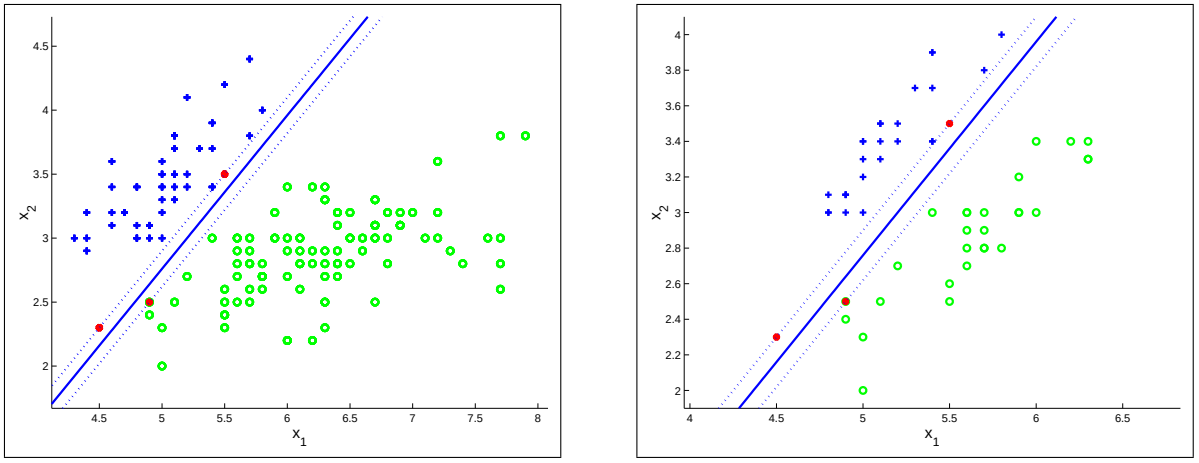
- Función de aptitud. Se utilizó la función de aptitud descritas en las ecuaciones 3.4 y 3.5.
- Representación de la solución. Cada cromosoma tiene tantos bits como elementos tiene el vector λ. Se utilizó una representación de números reales.
- Población inicial. Se generan números aleatorios entre 0 y 9 que son representados por un caracter.
- Selección. Se utilizó la estrategia de torneo.
- Cruza. Se pasan directamente a la nueva población dos copias del mejor individuo de la generación anterior y se completa el 10 % de esta nueva generació con la cruza del mismo mejor con todos los demás y finalmente se realizan cruzas uniformes de todos contra todos.
- Mutación. A cada gen, con probabilidad P_m, se le realiza una mutación que consiste en obtener un dígito aleatoriamente y asignárselo.

3.4. MÉTODO ZQP

Como se puede observar en la tabla 4.1, entre menor sea la constante de penalización C, el algortimo de Zoutendijk tiende a ser más rápido que el algoritmo Quadprog. Este resultado parecería no ser útil, ya que como la constante de penalización tiene que ser muy pequeña para que el algoritmo de Zoutendijk sea más rápido que el algoritmo de Quadprog, el resultado obtenido con esta constante permite tener muchos datos clasificados incorrectamente.
Sin embargo, revisando los valores asignados a cada multiplicador de Lagrange (λ), se puede notar que algunos de estos multiplicadores fueron asignados con el valor de cero y además estos multiplicadores corresponden a los datos que se encuentran más lejos de la frontera de decisión.
Así, los multiplicadores de Lagrange que después del entrenamiento con Zoutendijk utilizando una penelización muy baja, se pueden considerar como un subconjunto de los datos de entrenamiento donde los vectores de soporte están incluidos. Esto es un resultado muy importante ya que, como

se vió anteriormente, el entrenamiento con este subconjunto produce el mismo resultado que el conjunto completo, además, el tiempo utilizado por el algoritmo de Zoutendijk, como se puede ver nuevamente en la tabla 4.1, para encontrar este subconjunto es muy pequeño.

En la figura 3.3 se puede ver como los vectores de soporte encontrados con el conjunto de datos de entrenamiento y por el subconjunto de datos encontrados por el algoritmo de Zoutendijk son los mismos. En la figura figuras 3.3(a) se muestran los vectores de soporte y la frontera de decisión encontrados con el conjunto de datos de la base Iris completo y en la figura 3.3(b) se observa que los vectores de soporte y la frontera de decisión son los mismos que con el subconjunto de datos encontrados con el algoritmo de Zoutendijk utilizando una constante de penalización C= 0.001.



(a) (b)

Figura 3.3: Entrenamiento de la MSV con (a) el conjunto de datos completo y (b) con el subconjunto de datos obtenido con Zoutendijk

1. Utilizar el algoritmo de Zoutendijk con una constante de penalización muy pequeña, de esta forma se obtine de manera muy rápida una solución aproximada a la óptima.
2. De la solución obtenida se descartan los datos de entrenamiento cuyos multiplicadores de Lagrange hayan tenido el valor de cero.
3. Se reconstruye la matriz Hessiana pero sólo con los datos de entrenamiento obtenidos en la etapa enterior y
4. se resuelve el nuevo problema reducido con el algoritmo de programación cuadrática Quadprog.

Cuadro 1.2 Método de optimización propuesto ZQP.

3.5. *CHUNKING* CON ZQP

Para poder realizar la aproximación a la solución con el algoritmo propuesto en la sección anterior es necesario calcular la matriz Hessiana con el conjunto de datos completo, lo cual resulta muy costoso computacionalmente (memoria y tiempo computacional) para problemas de mayor tamaño, pero se conocen técnicas de descomposición como *Chunking*[1], Osuna [2] y *Sequential Minimal Optimization* (SMO) [3] que descomponen y resuelven el problema iterativamente. Se desarrolló un método que utiliza el algoritmo de *Chunking* para descomponer el problema en subproblemas, pero en lugar de resolver directamente el subproblema con Quadprog, se obtiene un conjunto de datos reducido con el algoritmo de Zoutendijk para después resolverlo con Quadprog.

4. EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS

4.1. BASES DE DATOS UTILIZADAS PARA LAS PRUEBAS

Las base utilizadas para las pruebas son : Iris, Sonar, Pima Indians Diabetes y Phonemes [29]. A continuación se dá una breve descripción de cada una de ellas.

4.1.1. IRIS

Esta base de datos en la más conocida en la literatura de reconocimiento de patrones. Contiene 150 instancias y 4 atributos y la clase.

Los atributos contienen la siguiente información:

1. Longitud del tallo en cm.
2. Ancho del tallo en cm.
3. Longitud del pétalo en cm.
4. Ancho del pétalo en cm.

Mediante los cuales se puede distinguir entre tres clases de flores:

Clase 1: Setosa.

Clase 2: Versicolor.

Clase 3: Virginica.

4.1.2. SONAR

Esta base de datos sirve para diferenciar entre señales acústicas producidas por un cilindro metálico y las producidas por un cilindro de piedra. La base contiene 111 patrones obtenidos por un cilindro metálico desde varios ángulos y bajo distintas condiciones, también contiene 97 patrones obtenidos del cilindro de piedra bajo condiciones similares. Cada patrón contiene 60 atributos numéricos en el rango de 0.0 a 1.0. Cada número representa la energía de una banda de frecuencia particular, integrada en un cierto periodo de tiempo. La etiqueta de clase contenida con cada patrón es:

Clase 1: Si el sonido es producido por el cilindro de roca y, Clase 2: Si el sonido es producido por el cilindro de metal.

4.1.3. PIMA INDIANS DIABETES.

Esta base de datos sirve para diagnosticar si un paciente muestra signos de Pima Indians Diabetes de acuerdo a criterios de la Organización Mundial de Salud. La población vive en Phoenix, Arizona.

Se utilizaron varias restricciones en la selección de estas instancias de una base mayor. En particular, todos los pacientes son del sexo femenino y tienen al menos 21 años de edad y de descendencia de Indios Pima.

Número de atributos: 8.

1. Número de veces que ha estado embarazada.
2. Concentración de glucosa después de dos horas de una prueba de tolerancia a la glucosa.
3. Presión sanguínea diastólica.
4. Índice de masa corporal.
5. Edad.

4.1.4. PHONEMES

Muchos de los sistemas de reconocimiento del lenguaje son globales (típicamente cadenas de Markov) que reconocen señales y no utilizan las características del lenguaje. Por el contrario, los sistemas analíticos toman en cuanta el proceso de articulación quienes describen diferentes

fonemas del lenguaje en cuestión. La idea es deducir la presencia de cada una de las características fonéticas a partir de observaciones acústicas.

La dificultad principal de los sistemas analíticos es la de obtener parámetros acústicos suficientes cuyas medidas (observaciones) deben contener:

- Toda la información relativa a la característica fonética buscada.
- Ser independiente de la persona que habla.
- Ser independiente del contexto.
- Ser robusta al ruido.

En esta base se observan las frecuencias y se representan en una escala logarítmica (frecuencias centrales) que representan la respuesta de los nervios auditivos. La salida de los filtros es observada entre 2 y 8msec (integración de 4 a 16 ms) dependiendo del tipo de fonema observado (estacionario o transitorio). El objetivo de esta base es el de distinguir entre:

Clase 0 : Nasales.

Clase 1 : Orales.

La base de aprendizaje cuenta con 1027 observaciones en 5 atributos (5 columnas) y la salida deseada también cuenta con 1027 observaciones y una sola columna de 0 o 1.

4.2. ACELERACIÓN DEL ALGORITMO DE ZOUTENDIJK

4.2.1. RESULTADOS

4.2.1.1. Pruebas con la la base de datos Iris

La tabla 4.1 compara los resultados obtenidos con el algoritmo de Zoutendijk, de las modificaciones realizadas a éste y los resultados obtenidos con Quadprog para el entrenamiento de la base de datos Iris. Se utilizaron diferentes valores de C. Zoutendijk con Simplex 1 y Zoutendijk con Simplex 2 se refieren al Simplex con las 2 modificaciones descritas en la sección 3.1.1 respectivamente.

4.2.2. ANÁLISIS DE LOS RESULTADOS.

Con las modificaciones realizadas al algoritmo Simplex se obtuvieron buenos resultados ya que con ambas modificaciones se redujo el tiempo de entrenamiento obtenido con el algoritmo de Zoutendijk con el Simplex de Linprog. Sin embargo, como se muestra en la tabla 4.1, el algoritmo

C	Zoutendijk con linprog	Zoutendijk con Simplex 1	Zoutendijk con Simplex 2	Quadprog
10000	385.34 s	191.18 s	143.00 s	3.78 s
1000	68.23 s	35.23 s	26.52 s	3.78 s
800	56.76 s	27.15 s	20.60 s	3.82 s
600	50.43 s	24.5 s	19.06 s	3.84 s
400	39.78 s	20.32 s	15.42 s	3.80 s
200	24.34 s	15.78 s	12.07 s	3.78 s
100	18,34 s	10.42 s	8.40 s	3.84 s
10	6.87 s	4.31 s	4.03 s	3.90 s
1	5.90 s	3.57 s	3.39 s	3.40 s
0.1	4.14 s	2.56 s	2.54 s	3.25 s
0.01	1.6 s	0.98 s	0.98 s	3.20 s
0.001	0.31 s	0.18 s	0.17 s	3.20 s
0.0001	0.22 s	0.20 s	0.17 s	3.30 s
0.00001	0.20 s	0.17 s	0.23 s	3.20 s

Tabla 4.1: Tabla comparativa del tiempo de entrenamiento utilizado por cada método con la base de datos IRIS.

de Zoutendijk sólo fue más rápido que Quadprog si se utiliza una C muy pequeña. Esto se debe a que el algoritmo de Zoutendijk al tratar de encontrar una solución utilizando una penalización alta comienza a oscilar y en algunos casos, utilizando bases datos más complejas y con una penalización alta, el algoritmo no converge. Es por eso que no se utilizaron los algoritmos Simplex Genético ni Simples Annealing para resolver el problema de optimización lineal.

Existe una modificación a este algortimo propuesta por Topkins [23], esta modificación asegura la convergencia del algoritmo, sin embargo, la forma para calcular el paso de aprendizaje propuesta no puede ser utilizada para caso especial de las MSV debido a que únicamente cuentan con una restricción de igualdad a cero y esto provoca que el valor del tamaño del paso de aprendizaje sea siempre cero. Una aportación importante sería encontrar una manera alterna de encontrar el tamaño del paso de aprendizaje que se adapte al problema.

4.3. MÉTODO HÍBRIDO: ZOUTENDIJK Y QUADPROG

4.3.1. RESULTADOS

Se realizaron pruebas del método con la base de datos Iris, en algunas ocasiones la solución obtenida por el algoritmo de Zoutendijk al momento de comenzar a oscilar no era factible para ser utilizada como solución inicial por el algortimo Quadprog, es por eso que se hicieron pruebas corriendo el algoritmo de Zoutendijk sólo cierto número de iteraciones con la finalidad de determinar si hasta cierta etapa del proceso el algoritmo obtiene soluciones factibles pra Quadprog..
Los resultados obtenidos con la base de datos Iris se muestran en la tabla 4.2.

Número de iteraciones del algoritmo de Zoutendijk	Tiempo total de entrenamiento (s)	Calidad de la solución obtenida por Zoutendijk
1	6.69	√
2	51	X
3	8.93	√
5	8.21	√
6	52	X
7	8.18	√
10	5.608	√
11	7.25	√
13	7.82	√
15	51	X
20	51.46	X
30	54.6	X
40	50.34	X
50	52	X

Tabla 4.2: Resultados IRIS kernel polinomial 2 .

4.3.2. ANÁLISIS DE RESULTADOS

Como se mencionó anteriormente, el algoritmo de Zoutendijk realiza aproximaciones lineales para llegar a la solución óptima, es por eso que la optimización con este algoritmo dentro del método propuesto puede traer consigo los siguientes problemas:

- Las soluciones obtenidas en diferentes iteraciones del algoritmo de Zoutendijk pueden o no ser soluciones factibles para el algoritmo de Quadprog.
- Cuando una solución encontrada por el algoritmo de Zoutendijk es no factible, el tiempo que ocupa Quadprog para resolver el problema es muy grande y el resultado final no es útil ya que no guarda ninguna relación con el resultado óptimo.
- Cuando la solución obtenida por el algoritmo de Zoutendijk es factible para se utilizada como solución inicial por Quadprog, el tiempo total utilizado por el algoritmo híbrido fue mayor que si sólo Quadprog es utilizado.

Con los resultados obtenidos se concluyó que con la implementación del algoritmo híbrido de esta manera no se obtuvieron resultados satisfactorios.

4.4. PRETRATAMIENTO DE LOS DATOS CON ALGORITMOS GENÉTICOS

4.4.1. RESULTADOS.

En el artículo [27], con el cual se realizó la implemetación del Algoritmo Genético descrita en la sección 3.3, no se menciona el tiempo necesario para obtener ese superconjunto de los vectores de soporte ni el número de iteraciones que se necesitaron para hacer este pretratamiento. Para encontrar esos valores se programó el Algoritmo Genético tal como se especifica en [27]. Se encontraron los siguientes problemas para su uso:

- **Tiempo**. El algoritmo funciona bien con una base de datos muy pequeña, en el artículo utilizan un máximo de 48 de los 150 datos de la base de datos Iris, pero al emplear una base de datos más grande, en este caso la base de datos IRIS completa, el Algoritmo Genético tarda demasiado tiempo en encontrar una solución aceptable.

- **Condición de paro**. Debido a que la solución que deseamos no es exacta, no podemos usar las condiciones de Karush-Khun-Tucker para determinar que estamos en la solución óptima y así parar el algortimo, por otro lado se podría detener el algoritmo cuando ya se haya reducido la cantidad de vectores de soporte una cierta cantidad o un porcentaje, sin embargo no es posible determinar antes del entrenamiento la cantidad de vectores de soporte resultantes.

La opción restante (y la utilizada por los artículos) es hacer el algoritmo trabaje sólo un cierto número de iteraciones, pero el inconveniente de utilizar esta condición de paro es que si el número de iteraciones ejecutadas es demasiado grande, el algoritmo tardará mucho más de lo que es necesario o, del lado contario, si el número de iteraciones es demasiado pequeño, el subconjunto de datos encontrados será muy diferente al conjunto de vectores de soporte.

4.4.2. ANÁLISIS DE LOS RESULTADOS.

La utilización de un algoritmo genético para resolver el problema de programación cuadrática resulta ineficiente debido a que el espacio de búsqueda es muy grande y existen algoritmos de programación cuadrática clásicos que nos permiten obtener una solución más rápida y precisa.
El uso de algoritmos genéticos, así como de recocido simulado, han demostrado que pueden dar buenos resultado si son utilizados dentro de un algoritmo de optimización, se pueden ver algunos de

estos casos en [22] Y [21], donde los algortimos genéticos y el recocido simulado, respectivamente, son utilizados para acelerar el proceso de optimización del algoritmo simplex. Técnicas similares podrían buscarse para incorporar los algoritmos genéticos o el recocido simulado en algoritmos de programación cuadrática.

4.5. MÉTODO ZQP.

4.5.1. RESULTADOS

Las siguientes tablas muestran los resultados obtenidos del entrenamiento con las bases de datos de prueba. Para el análisis del comportamiento del método con diferentes constantes de penalización para el algoritmo de Zoutendijk (CZ), la constante de penalización utilizada por Quadprog en todas las pruebas es $C = 1000$. Se midió el tiempo ocupado en cada etapa del proceso con la finalidad de analizar como afecta a cada etapa la modificación de la CZ. En las tablas mostradas a continuación se utilizan las siguientes abreviaturas:

- **CZ:** Valor de la constante de optimización utilizada únicamente en el algoritmo de Zoutendijk.
- **Tiempo H:** Tiempo necesario para calcular la matriz Hessiana con el conjunto de datos completo.
- **Tiempo Zou:** Tiempo ocupado para realizar el pretratamiento de los datos con Zoutendijk.
- **Tiempo H2:** Tiempo necesario para recalcular la matriz Hessiana con el conjnto de datos reducido obtenido en el pretratamiento con Zoutendijk.
- **Tiempo QP:** Tiempo que ocupa Quadprog para realizar la optimización con el conjunto de datos reducido.
- **Tiempo total ZQP:** Es el tiempo que se ocupó durante todo el proceso de optimización del método ZQP: Tiempo H + Tiempo Zoutendijk + Tiempo H2 + Tiempo QP.
- **Datos encontrados:** Número de datos en el conjnto de datos reducido obtenido en el pretratamiento con Zoutendijk.
- **S.V. omitidos:** Número de datos que son vectores de soporte y que durante el pretratamiento fueron eliminados.

4.5.1.1. Pruebas con la base de datos Iris

El tiempo de entrenamiento de la base de datos Iris utilizando un kernel polinomial de grado 2 con Quadprog es de **4.97 s** y el número de vectores de soporte es de **11**. La tabla 4.3 muestra el tiempo utilizado para el entrenamiento con diferentes CZ.

CZ	Tiempo H	Tiempo Zoutendijk	Tiempo H2	Tiempo QP	Tiempo total ZQP	Datos encontrados	S.V. omitidos
0.1	1.1	6	0	0.015	7.23	12	3
0.1	1.1	4.68	0	0.094	6.02	19	1
0.001	1.1	3.67	0.03	0.025	5.21	42	0
0.0009	1.1	3.7	0.016	0.026	5.26	43	0
0.0005	1.1	3.29	0	3.5	4.95	50	0
0.0004	1.1	3.07	0.016	0.39	4.76	52	0
0.0003	1.1	2.97	0.015	0.45	4.71	56	0
0.0002	1.1	2.8	0.016	0.58	4.7	63	0
0.0001	1.1	2	0.03	0.84	4.14	72	0
0.00009	1.1	2	0.03	0.9	4.21	74	0
0.00007	1.1	1.78	0.03	1.25	4.18	79	0
0.00005	1.1	1.51	0.03	1.65	4.46	87	0

Tabla 4.3: Resultados del entrenamiento, con el método ZQP, de la base de datos IRIS con un kernel polinomial de grado 2 .

El tiempo de entrenamiento de la base de datos Iris utilizando un kernel RBF con $\sigma = 2$ con Quadprog es de **5.95** s y el número de vectores de soporte es de **10**. La tabla 4.4 muestra el tiempo utilizado para el entrenamiento con diferentes CZ.

El tiempo de entrenamiento de la base de datos Iris utilizando un kernel RBF con $\sigma = 0{,}5$ con Quadprog es de **5.76** s y el número de vectores de soporte es de **66**. La tabla 4.5 muestra el tiempo utilizado para el entrenamiento con diferentes CZ.

Conclusiones de las pruebas con la base de datos Iris.

De las tablas 4.3, 4.4 y 4.5 se puede observar que es necesario utilizar una CZ de un valor pequeño para que el algoritmo de Zoutendijk pueda realizar una optimización rápida ya que entre

CZ	tiempo H	Tiempo Zoutendijk	Tiempo H2	Tiempo QP	Tiempo total ZQP	Datos encontrados	S.V. omitidos
0.1	1.3	1.56	0.03	2.31	5.29	93	0
0.01	1.3	0.51	0.046	3.43	5.42	101	0
0.001	1.3	0.45	0.04	3.34	5.29	101	0
0.0001	1.3	0.29	0.06	3.07	4.87	103	0
0.00005	1.3	0.25	0.06	2.45	4.7	105	0
0.00001	1.3	0.15	0.04	3.4	5	105	0

Tabla 4.4: Resultados del entrenamiento, con el método ZQP, de la base de datos IRIS con un kernel RBF con $\sigma = 2$.

CZ	tiempo H	Tiempo Zoutendijk	Tiempo H2	Tiempo QP	Tiempo total ZQP	Datos encontrados	S.V. omitidos
1	1.32	39.5	0.03	0.15	41.12	76	5
0.5	1.32	7.75	0.01	0.25	9.45	81	5
0.1	1.32	1.7	0.06	1.68	4.87	105	3
0.05	1.32	1.75	0.04	1.68	4.92	105	3
0.01	1.32	1.2	0.06	1.7	4.4	105	3
0.001	1.32	0.89	0.04	2.3	4.71	107	2
0.0001	1.32	0.6	0.07	2.78	4.9	123	1
0.00005	1.32	0.29	0.011	3.5	5.32	136	0
0.00001	1.32	0.12	0.09	3.51	5.18	134	1

Tabla 4.5: Resultados del entrenamiento, con el método ZQP, de la base de datos IRIS con un kernel RBF con $\sigma = 0.5$.

mayor sea la CZ el algoritmo de Zoutendijk buscará una mejor solución y es ahí donde puede empezar a oscilar y tardar demasiado tiempo. Por el contrario, si la constante de penalización CZ es demasiado pequeña, el subconjunto de datos encontrado por Zoutendijk será de un tamaño muy parecido al conjunto completo (o posiblemente del mismo tamaño) lo que ocasionará que el tiempo reducido en la optimización con Quadprog sea muy pequeño (o en el peor de los casos no se reduzca).

4.5.1.2. Pruebas con la base de datos Sonar

El tiempo de entrenamiento de la base de datos Sonar utilizando un kernel polinomial de grado 2 con Quadprog es de **9.81 s** y el número de vectores de soporte es de **79**. La tabla 4.6 muestra el tiempo utilizado para el entrenamiento con diferentes CZ.

C	tiempo H	Tiempo Zoutendijk	Tiempo H2	Tiempo QP	Tiempo total ZQP	Datos encontrados	S.V. omitidos
0.01	2.3	6.95	0.12	2.9	12.42	161	5
0.001	2.3	0.26	0.26	6.55	9.39	196	2
0.0001	2.3	0.18	0.26	6.34	9.23	196	2

Tabla 4.6: Resultados del entrenamiento, con el método ZQP, de la base de datos SONAR con un kernel polinomial de grado 2.

El tiempo de entrenamiento de la base de datos Sonar utilizando un kernel RBF con $\sigma = 2$ con Quadprog es de **9.87** s y el número de vectores de soporte es de **81**. La tabla 4.7 muestra el tiempo utilizado para el entrenamiento con diferentes CZ.

C	tiempo H	Tiempo Zoutendijk	Tiempo H2	Tiempo QP	Tiempo total ZQP	Datos encontrados	S.V. omitidos
0.1	2.8	0.36	0.26	5.62	9.2	196	2
0.01	2.8	0.25	0.23	5.6	9.07	196	2
0.001	2.8	0.2	0.23	5.75	9.14	196	2

Tabla 4.7: Resultados del entrenamiento, con el método ZQP, de la base de datos SONAR con un kernel RBF con $\sigma = 2$

.

El tiempo de entrenamiento de la base de datos Sonar utilizando un kernel RBF con $\sigma = 0{,}5$ con Quadprog es de **3.51** s y el número de vectores de soporte es de **201**. La tabla 4.8 muestra el tiempo utilizado para el entrenamiento con diferentes CZ.

C	tiempo H	Tiempo Zoutendijk	Tiempo H2	Tiempo QP	Tiempo total ZQP	Datos encontrados	S.V. omitidos
0.1	2.76	5.04	0.32	0.5	8.84	207	0
1000	2.76	1.04	0.29	0.5	4.82	207	0
800	2.76	0.35	0.31	0.5	4.1	207	0
600	2.76	0.14	0.29	0.56	3.98	208	0

Tabla 4.8: Resultados del entrenamiento, con el método ZQP, de la base de datos SONAR con un kernel RBF con $\sigma = 2$.

Conclusiones de las pruebas con la base de datos Sonar.

Con el pretratamiento de datos se obtuvieron el tiempo de entrenamiento con las funciones kernel polinomial de grado 2 y RBF con $\sigma = 2$ (tablas 4.6 y 4.7) fue menor que el entrenamiento realizado directamente con Quadprog.

La realización del pretratamiento con el kernel RBF con σ = 0.5 (tabla 4.8) provocó que el entrenamiento tardara más que si se utilizara directamente Quadprog.

Este resultado se debe a que el kernel RBF con σ = 0.5 hace que la función de decisión tenga mucha capacidad lo que provoca que la mayoría de los datos de entrenamiento sean vectores de soporte y por consiguiente al hacer el pretratamiento los datos eliminados son muy pocos por lo que el tiempo ocupado para el pretratamiento y el recálculo de la matriz Hessiana resulta ser tiempo perdido.

4.5.1.3. Pruebas con la base de datos Pima Indians Diabetes

No fue posible realizar el entrenamiento de la base de datos Pima Indians Diabetes utilizando un kernel polinomial de grado 2. Para llegar a esta conclusión se esperaron aproximadamente 12 horas y no se llegó a la solución óptima realizando el entrenamiento directamente con Quadprog ni con el pretratameinto. Dadas estas circunstancias se dividió la lase en partes de 100 cada una y se realizó el entrenamiento con el kernel polinolial de grado 2 para poder observar el número de vectores de soporte en cada parte con la finalidad de tener una idea de la complejidad de la base de datos.

En la tabla 4.9 se muestra el tiempo total obtenido para la parte de la base de datos que esté indicada en la primera columna y se compara con el tiempo utilizado por el entrenamiento tradicional con Quadprog. Se utilizó una CZ = 0.0001 y como se indicó anteriormente, el valor de C en estas pruebas es de 1000.

La tabla 4.10 muestra, para cada parte de la base entrenada, el número de datos encontrados en el pretratamiento, el número de vectores de soporte y el número de vectores de soporte que fueron incorrectamente excluidos del conjunto de datos encontrados en el pretratamiento.

Parte base	tiempo H	Tiempo Zou	Tiempo H2	Tiempo QP	Tiempo total con ZQP	Tiempo total con el alg. tradicional QP
1-100	0.5	2.98	0.03	7.42	10.95	
100-200	0.5	3.28	0.047	5.25	9.15	7.9
200-300	0.5	3.15	0.04	9.46	13.68	12.15
300-400	0.5	3.07	0.032	4.609	8.14	10.48
400-500	0.5	3.17	0.016	3.76	7.5	8.93
500-600	0.5	2.51	0.03	2.2	5.3	5.41
600-700	0.5	3.37	0.031	5.23	9.2	9.37

Tabla 4.9: Resultados del tiempo de entrenamiento, con el método ZQP, de la base de datos Pima Indians Diabetes con un kernel polinomial de grado 2.

Parte de la base	Datos encontrados con Zoutendijk	Número de vectores de soporte	Vectores de soporte omitidos por Zoutendijk
1-100	87	51	3
100-200	72	34	2
200-300	93	39	2
300-400	76	41	3
400-500	69	32	3
500-600	68	27	4
600-700	79	40	5

Tabla 4.10: Resultados de la calidad de la solución del entrenamiento, con el método ZQP, de la base de datos Pima Indians Diabetes con un kernel polinomial de grado 2.

No se muestran los datos obtenidos con los kernels empleados en las bases de datos anteriores (kernel RBF con $\sigma = 2$ y con σ = 0.5). Esto debido a que el número de datos encontrados en el pretratamiento fue similar al número total de datos de entrenamieto y por lo tanto el tiempo de entrenamiento del método ZQP fue mayor que el del método tradicional con Quadprog.

Conclusiones de las pruebas con la base de datos Pima Indians Diabetes.

Con esta base de datos es fácil notar que el tiempo de entrenamiento con el método ZQP es menor que con el algoritmo Quadprog entre menor sea la relación (Número de vectores de soporte / Subconjunto de datos encontrado por Zoutendijk).

Por lo anterior es muy importante seleccionar el kernel adecuado para obtener en el entrenamiento únicamente el número de vectores de soporte necesarios para clasificar correctamente los datos.

4.5.1.4. Pruebas con la base de datos Phonemes

Al igual que con las pruebas realizadas en la sección anterior con la base de datos Pima Indians Diabetes, el entrenamiento de esta base de datos se realizó por partes para poder analizar como varía la dificultad para entrenar las diferentes partes de la base de datos. En esta base de datos, algunas partes no pudieron ser entrenadas con el kernel y la penalización especificada, ya que en poco más de una hora el optimizador no pudo encontrar una solución. Los celdas de las tablas con NA indican que esa parte de la base no pudo ser entrenada.

La tabla 4.11 muestra el tiempo requerido para el entrenamiento de la parte de la base de datos indicada en la primer columna por el método ZQP y por el método tradicional con Quadprog.

La tabla 4.12 muestra el número de datos encontrados en el pretratamiento, el número de vectores de soporte y el número de vectores de soporte que fueron incorrectamente excluidos del conjunto de datos encontrados en el pretratamiento.
En las pruebas realizadas en estas dos tablas se utilizó un kernel polinomial de grado 2, un valor de C = 1000 y CZ = 0.0001.

Parte base	tiempo H	Tiempo Zou	Tiempo H2	Tiempo QP	Tiempo total con ZQP	Tiempo total con el alg. tradicional QP
1-100	0.48	0.54	0.03	9	2.03	2.51
100-200	NA					
200-300	0.5	0.46	0	0.98	2.01	2.78
300-400	0.5	2.87	0.016	145.81	149.5	1
400-500	NA					
500-600	0.51	0.47	0.016	4.75	5.8	6.82
600-700	NA					
700-800	0.58	0.18	0.016	0.43	1.28	1.79
800-900	NA					
900-1000	NA					

Tabla 4.11: Resultados del tiempo de entrenamiento, con el método ZQP, de la partes de la base de datos PHONEMES con un kernel polinomial de grado 2.

Parte de la base	Datos encontrados con Zoutendijk	Número de vectores de soporte	Vectores de soporte omitidos por Zoutendijk
1-100	56	21	2
100-200	NA		
200-300	54	23	1
300-400	66	37	1
400-500	NA		
500-600	66	33	2
600-700	NA		
400-500	73	39	0
500-600	78	47	3
600-700	NA		
700-800	73	39	0
800-900	NA		
900-10000	NA		

Tabla 4.12: Resultados de la calidad de la solción del entrenamiento, con el método ZQP, de partes de la base de datos PHONEMES con un kernel polinomial de grado 2.

La tabla 4.13 muestra el tiempo requerido para el entrenamiento de la parte de la base de datos indicada en la primer columna por el método ZQP y por el método tradicional con Quadprog.

La tabla 4.14 muestra el número de datos encontrados en el pretratamiento, el número de vectores de soporte y el número de vectores de soporte que fueron incorrectamente excluidos del conjunto de datos encontrados en el pretratamiento.

En las pruebas realizadas en estas dos tablas se utilizó un kernel RBF con σ = 2, un valor de C = 1000 y CZ = 0.0001.

Parte base	Tiempo H	Tiempo Zou	Tiempo H2	Tiempo QP	Tiempo total con ZQP	Tiempo total con el alg. tradicional QP
1-100	0.59	0.32	0.03	0.28	1.29	1.5
100-200	0.59	0.15	0.03	0.59	1.46	1.9
200-300	0.59	0.14	0.016	0.29	1.09	1.64
300-400	0.59	0.25	0.016	0.031	1.21	1.93
400-500	0.59	0.078	0.016	0.75	1.47	2.3
500-600	0.59	0.2	0.031	0.48	1.34	1.85
600-700	0.59	0.25	0.016	0.82	1.71	2.15
700-800	0.59	0.17	0.031	0.51	1.35	1.82
800-900	0.59	0.2	0.03	1.03	1.9	2.51
900-1000	0.59	0.34	0.016	0.45	1.43	2.1

Tabla 4.13: Resultados del tiempo de entrenamiento, con el método ZQP, de la base de datos IRIS con un kernel RBF con σ = 2.

Parte de la base	Datos encontrados con Zoutendijk	Número de vectores de soporte	Vectores de soporte omitidos por Zoutendijk
1-100	69	27	2
100-200	73	29	3
200-300	55	29	2
300-400	62	5	2
400-500	73	42	4
500-600	63	36	3
600-700	74	39	2
400-500	73	39	0
500-600	78	47	3
600-700	74	39	2
700-800	73	39	0
800-900	78	47	3
900-1000	69	37	3

Tabla 4.14: Resultados de la calidad de la solución del entrenamiento, con el método ZQP, de la base de datos PHONEMES con un kernel RBF con $\sigma = 2$.

La tabla 4.15 muestra el tiempo requerido para el entrenamiento de la parte de la base de datos indicada en la primer columna por el método ZQP y por el método tradicional con Quadprog.

La tabla 4.16 muestra el número de datos encontrados en el pretratamiento, el número de vectores de soporte y el número de vectores de soporte que fueron incorrectamente excluidos del conjunto de datos encontrados en el pretratamiento.

En las pruebas realizadas en estas dos tablas se utilizó un kernel RBF con σ = 0.05, un valor de C = 1000 y CZ = 0.0001.

Parte base	Tiempo H	Tiempo Zou	Tiempo H2	Tiempo QP	Tiempo total con ZQP	Tiempo total con el alg. tradicional QP
1-100	0.59	0.25	0.03	0.094	1	0.71
100-200	0.59	0.25	0.06	0.06	0.96	0.75
200-300	0.59	0.25	0.047	0.047	1	0.73
300-400	0.59	0.23	0.047	0.047	1	0.73
400-500	0.59	0.23	0.078	0.078	0.98	0.76
500-600	0.59	0.21	0.047	0.094	1.01	0.73

Tabla 4.15: Resultados del tiempo de entrenamiento, con el método ZQP, de la base de datos PHONEMES con un kernel RBF con σ = 0.5.

Parte de la base	Datos encontrados con Zoutendijk	Número de vectores de soporte	Vectores de soporte omitidos por Zoutendijk
1-100	96	97	3
100-200	98	97	0
200-300	100	98	0
300-400	98	97	2
400-500	99	98	0
500-600	101	101	0

Tabla 4.16: Resultados de la calidad de la solución del entrenamiento, con el método ZQP, de la base de datos PHONEMES con un kernel RBF con σ = 0.5.

4.5.2. ANÁLISIS DE LOS RESULTADOS OBTENIDOS CON EL MÉTODO ZQP

En las tablas 4.3, 4.4, 4.5, 4.6, 4.7 y 4.8, se realizaron pruebas con diferentes constantes de penalización C del algoritmo de Zoutendijk (CZ), en todas las pruebas se nota que con una CZ del mismo valor que C o muy cercano, se obtiene una solución muy aproximada a la real pero para llegar a esa solución el algoritmo de Zoutendijk consume más tiempo que el consumido por Quadprog. A medida que el valor de CZ disminuye, se obtiene una solución menos exacta pero la velocidad con que es encontrada esta solución es más rápida y, aunque el tiempo de Quadprog aumenta a comparación del caso anterior, tiene una disminución muy notoria comparada al entrenamiento con todo el conjunto de datos. Por el contrario, si CZ es demasiado pequeña, el algoritmo producirá una solución que puede regresar el conjunto completo de datos de entrenamiento.

En las pruebas realizadas encontramos que en general en todas las bases de datos utilizadas como prueba un buen valor para CZ es de 10^{-5}, ya que con este valor la optimización con Zoutendijk se realiza rápidamente y el tiempo total empleado por los dos algoritmos (Zoutendijk y Quadprog) es menor que el utilizado sólo por Quadprog.

Para que el método trabaje de manera más rápida que si es utilizado sólo el algoritmo de Quadprog es muy importante que el número de vectores de soportes sea mucho más pequeño que el número de datos de entrenamiento. Para que esta condición se cumpla es necesario utilizar una función kernel adecuada, si se utiliza una función con una capacidad muy alta para el problema que estamos resolviendo, el número de vectores de entrenamiento será muy cercano al número de datos de entrenamiento y el tiempo invertido para encontrar el superconjunto con los datos de entrenamiento será tiempo perdido ya que encontrará la mayoría de ellos.

4.6. *CHUNKING* CON ZQP

4.6.1. RESULTADOS

El algoritmo de Chunking, por ser aleatorio, produce diferentes tiempos de aprendizaje. Para medir la eficiencia del algoritmo ZQP dentro del algoritmo *Chunking* se realizaron 1000 corridas para después calcular el tiempo promedio y la desviación estándard y así poder comparar el funcionamiento del algoritmo de Chunking tradicional contra Chunking con ZQP. Los resultados mostrados en las tablas 4.17, 4.18 y 4.19 se realizaron utilizando el algoritmo ZQP únicamente en las tres primeras iteraciones del Chunking, esto debido a que a medida que aumenta el número de iteraciones, el tamaño del conjunto de trabajo del Chunking se incrementa y a la vez este conjunto tiene un mayor número de vectores de soporte y eso provoca que después de un número determinado de iteraciones sea más conveniente utilizar el algoritmo QP. En las pruebas se eligió utilizar el método ZQP en las tres primeras iteracioens de manera arbitraria, sabiendo que el número óptimo puede variar dependiendo de la complejidad de la base de datos.

4.6.1.1. Pruebas con la base de datos Iris

La base de datos Iris tiene 150 elementos y con el kernel polinomial de grado 2 se obtienen 11 vectores de soporte para encontrar el hiperplano de separación óptimo.

Porcentaje de Chunk	Chunking tradicional Chunking + QP		Chunking con inicialización (Chunking con ZQP)	
	Tiempo prom.	Desv. estándard	Tiempo prom.	Desv. estándard
10 %	0.4111	0.1047	0.4167	0.1072
30 %	0.6176	0.1077	0.5547	0.1108
50 %	0.9589	0.1007	0.9272	0.1022

Tabla 4.17: Resultados del entrenamiento del método Chunking con ZQP, de la base de datos IRIS con un kernel polinomial de grado 2

4.6.1.2. Pruebas con la base de datos Sonar

La base de datos Sonar tiene 208 elementos y con el kernel polinomial de grado 2 se obtienen 75 vectores de soporte para encontrar el hiperplano de separación óptimo.

Porcentaje de Chunk	Chunking tradicional Chunking + QP		Chunking con inicialización (Chunking con ZQP)	
	Tiempo prom.	Desv. estándard	Tiempo prom.	Desv. estándard
10 %	1.5614	0.1990	1.6910	0.2005
30 %	1.77	0.2021	1.9203	0.2392
50 %	2.5301	0.2740	2.7639	0.2811

Tabla 4.18: Resultados SONAR kernel polinomial 2

4.6.1.3. Pruebas con la base de datos Pima Indians Diabetes

La base de datos Pima Indians Diabetes tiene 768 elementos y con el kernel RBF = 0.5 se obtienen 608 vectores de soporte para encontrar el hiperplano de separación óptimo.

Porcentaje de Chunk	Chunking tradicional Chunking + QP		Chunking con inicialización Chunking con ZQP)	
	Tiempo prom.	Desv. estándard	Tiempo prom.	Desv. estándard
5 %	193.79	18.0774	197.6215	21.3137
8 %	195.8	18.12	195.1466	19.11

Tabla 4.19: Resultados Pima Indians Diabetes kernel polinomial 2

4.6.1.4. Pruebas con la base de datos Phonemes

La base de datos Iris tiene 1027 elementos y con el kernel RBF = 2 se obtienen 329 vectores de soporte para encontrar el hiperplano de separación óptimo.

Porcentaje de Chunk	Chunking tradicional Chunking + QP		Chunking con inicialización Chunking con ZQP)	
	Tiempo prom.	Desv. estándard	Tiempo prom.	Desv. estándard
5 %	244.5669	15.9645	241.6707	17.27
8 %	271.33	41.8011	235.5911	12.6499

Tabla 4.20: Resultados PHONEMES kernel RBF 2

4.6.2. ANÁLISIS DE RESULTADOS DEL ALGORITMO *CHUNKING CON ZQP*

Los tiempos de entrenamiento del algoritmo *Chunking* con ZQP fueron menores los del *Chunking* con QP en las bases de datos Iris (tabla 4.17), Sonar (tabla 4.18) y PHONEMES (tabla 4.20). Esto se debe a que el número de vectores de soporte es significativamente menor al número de datos de entrenamiento.

El tiempo de entrenamiento con la base de datos Pima Indians Diabetes (tabla 4.19) fue mayor con el algoritmo *Chunking* con ZQP que con el algoritmo tradicional con QP. Esto se debe a que el número de vectores de soporte es similar al número de elementos de la base de datos de entrenamiento.

Concluyendo, es muy importante que, para que el algoritmo ZQP sea más rápido que el algoritmo QP, el número de vectores de soporte sea significativamente menor que el número de datos de entrenamiento, lo cual es normalmete ciertopara muchas bases de datos con una función kernel adecuada.

5. CONCLUSIONES Y TRABAJO FUTURO

5.1. CONCLUSIONES.

El objetivo de este trabajo de tesis fue implementar un método para reducir el tiempo de entrenamiento de las Máquinas de Soporte Vectorial que, como se presentó, siendo parte del aprendizaje estadístico, resuelven un sinnúmero de problemas siendo su principal desventaja justamente el tiempo de aprendizaje. El proyecto que enmarca este trabajo de tesis está enfocado a reducir el tiempo de aprendizaje por dos diferentes tipos de métodos: i) por métodos heurísticos y ii) por métodos matemáticos. En esta tesis se presentan los métodos matemáticos. Para cumplir con este objetivo se implementaron y se realizaron pruebas con diferentes métodos. Los siguientes puntos resumen las conclusiones a las cuales se llegaron con cada uno de estos métodos.

1. **Aceleración del algoritmo de Zoutendijk.** Con las modificaciones realizadas al algoritmo Simplex, el cual se ocupa en cada iteración del algoritmo de Zoutendijk, se logró reducir el tiempo de optimización comparado con el algoritmo Simplex de Linprog aproximadamente a la mitad. Sin embargo, como se muestra en las tablas, el tiempo utilizado por Quadprog es mucho menor, habiendo mayor diferencia entre mayor sea la constante de penalización C. Los resultados obtenidos son importantes porque muestran que adaptando el algoritmo a los requerimientos específicos del

problema (en este caso al generado por las SVM) se puede acelerar de manera significativa el optimizador. Adaptaciones similares se pueden realizar a los optimizadores cuadráticos como Quadprog.

2. **Método híbrido.** Con este método los resultados obtenidos no fueron satisfactorios, esto debido a que el tiempo de optimización con este método es mayor que el utilizado con el algoritmo Zoutendijk únicamente, además de que no es posible asegurar que la solución obtenida sea la óptima.

3. **Pretratamiento de los datos con Algoritmos Genéticos.** La idea de encontrar un subconjunto de los datos de entrenamiento el cual contenga los vectores de soporte y realizar el entrenamiento únicamente con ese subconjunto es en principio una muy buena idea ya que normalmente el número de vectores de soporte es mucho menor que el número de vectores de soporte. En las pruebas realizadas con esta implementación se encontraron principalmente los dos problemas siguientes:

a) La búsqueda mediante los Algoritmos Genéticos resultó ser muy lenta, incluso la simple búsqueda tarda mucho más que realizando el entrenamiento con el conjunto de datos completo directamente con Quadprog.
b) Un problema muy importante es que para realizar la búsqueda del subconjunto de datos es necesario construir la matriz Hessiana con el conjunto de datos completo.

Estos dos problemas traen como resultado que no sea posible resolver ninguna de las dos principales dificultades a las que se enfrentan las MSV (tiempo de entrenamiento y memoria computacional).

4. **Método ZQP.** La idea de buscar un subconjunto de datos de entrenamiento que contenga los vectores de soporte de una manera mucho más rápida trajo como resultado el método ZQP. Los resultados de las pruebas realizadas a este método muestran que es más rápido que el algoritmo Quadprog si el número de vectores de soporte es significativamente menor al número de datos de entrenamiento. Este resultado era el esperado ya que el tiempo invertido para buscar los vectores de soporte tiene que ser recompensado encontrando pocos vectores de soporte para que con estos la MSV sea entrenada en un tiempo muy corto. Normalmente, en las bases de datos obtenidas de problemas reales, al seleccionar una función kernel adecuada, el conjunto de vectores de soporte es mucho menor que el número de datos de entrenamiento, es por eso que los resultados obtenidos

con el método ZQP son satisfactorios. La principal desventaja de este método es que, al igual que con los Algoritmo Genéticos, es necesario el cálculo de la matriz Hessiana

5. **Chunking con ZQP.** Como se mencionó en el punto anterior, para utilizar el algoritmo ZQP es necesario calcular la matriz Hessiana completa. El problema principal de este cálculo es que si la base de datos es muy grande, la memoria puede ser insuficiente para almacenar dicha matriz. Para tratar con este problema se utilizó el algoritmo Chunking, el cual crea subproblemas del problema original y los resuelve de manera iterativa hasta llegar a una solución. Se utilizó el método ZQP para resolver los subproblemas y se comparó cuando estos problemas son resueltos con Quadprog. La utilización del método ZQP para resolver estos problemas tiene buenos resultados si es utilizado únicamente en las primeras iteraciones del algoritmo (en las siguientes se utiliza Quadprog) y es nuevamente muy importante que el número de vectores de soporte sea significativamente menor que el número de datos de entrenamiento para poder reducir el tiempo de optimización.

5.2. TRABAJO FUTURO.

- Se puede observar que los resultados obtenidos con el método ZQP, con el cual se obtuvieron los mejores resultados, muestran que la utilización del algoritmo de direcciones factibles Zoutendijk para encontrar el superconjunto de datos de entrenamiento, ayudó a reducir el tiempo de entrenamiento total. La importancia de este método es que se logró comprobar que si se encuentra un primero un subconjunto de los datos de entremamiento de manera muy rápida, que tengan mayor probabilidad de ser vectores de soporte, se puede reducir el tiempo de entrenamiento.

- Las pruebas realizadas del método ZQP con un algoritmo de descomposición fueron implementadas con el algoritmo *Chunking*. Como trabajo futuro se pueden realizar pruebas con el algoritmo de Osuna, donde se esperan mejores resultados, ya con el algoritmo de *Chunking* el conjunto de trabajo crece muy rápidamente, guardando los vectores de soporte, lo que provoca que se puedan eliminar muy pocos datos a medida que transcurren las iteraciones, esto en contraste con el algoritmo de Osuna, donde el conjunto de trabajo no crece y en cada iteración es muy diferente al anterior.

- Buscar un método que realice la búsqueda de los vectores de soporte de una manera más rápida y más precisa que el algoritmo de Zoutendijk, de esta forma los resultados que se pueden

obtener serían de gran importancia ya que reducirían muy significativamente el tiempo de entrenamiento.

- También sería importante realizar un desarrollo similar al de este trabajo pero con un algoritmo de programación cuadrático como Quadprog. Se podría analizar su funcionamiento, como el análisis realizado con Zoutendijk, para encontrar partes del algoritmo donde sea posible incorporar técnicas estocásticas como Algoritmos Genéticos o Recocido Simulado para acelerar el proceso de optimización.

6. BIBLIOGRAFÍA

BIBLIOGRAFÍA

[1] VAPNIK, V. Statistical Learning Theory. *Journal of the Association for Computing Machinery*, 40:741–764, 1993.

[2] OSUNA E. et al. ”Suppot Vector Machines: Training and applications”. In *Massachusetts Institute of Technology*, Marzo de 1997.

[3] PLATT, JOHN. Fast Training of Support Vector Machines using Sequential Minimal Optimization. Microsoft Research. MIT Press, 1998.

[4] McCULLOCH, W. S., PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133, 1943.

[5] HEBB, D. The Organization of Behavior. *Wiley, New York.* , 1949.

[6] ROSENBLATT, F. ”The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” *In Psychological Review, Volume 65, November, 1958, pp. 386-408.*

[7] WIDROW, BERNARD. ”Generalization and Information Storage in Networks of Adaline ’Neurons’” Self-Organizing Systems, pp.435-461, Washington, DC: Spartan Books, 1962.

[8] MINSKY AND PAPERT, 1969. Perceptrons: Introduction to Computational Geometry. *WMIT Press, Cambridge.* , 1969.

[9] RUMELHART,D.E., HINTON G.E., AND WILLIAMS, R.J. Learning representations by back-propagating errors. In *Parallel distributed procesing, vol 1, pp. 318-362, The MIT Press, 1986.*

[10] VECKMAN, VOJISLAV. Learning and Soft Computing. In MIT Press, Cambidge Massachusetts*, 2001.*

[11] BISHOP, CHRISTOPHER. Neural Networks for Pattern Recognition. Oxford University Press*, reprinted 2002.*

[12] CRISTIANINI, N., SHAWE, J. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2002.

[13] RAJESH PAREKH, et al. Constructive Neural-Network Learning Algorithms for Pattern Classification. TRANSACTIONS ON NEURAL NETWORKS, VOL. 11, NO. 2, MARCH 2000.

[14] ARAN, O., ALPAYDIN, E An Incremental Neural Network Construction. Algorithm for Training Multilayer Perceptrons", ICANN'03, Istanbul, June 2003.

[15] REED R. Pruning algorithms, a survey. IEEE Transactions on Neural Networks, 1993.

[16] FAHLMAN, S. E. The recurrent cascade-correlation architecture. Advances in Neural Information Processing Systems 3, pages 190-196. Morgan Kaufmann, Denver, Co.

[17] VAPNIK, V. An Overview of Statistical Learning Theory. *IRRR Transactions on Neural Networks, Vol. 10, no.5, september 1999.*

[18] VAPNIK, V. and CHERVONENKIS, A. J. One the uniform convergence of relative frequencies of events to their probabilities. Theory of probabilities and its applications, vol. 16, pp. 264-280, 1971.

[19] VAPNIK, V., Minimization of expected risk based on empirical data. Proc. of the 1st World Congress of the Bernoulli Society VNUSCIENCEPRESS, vol. 2, Utrecht, the Netherlands, pp. 821-832, 1987.

[20] VAPNIK, V. and CHERVONENKIS, A. J. The necessary and sufficient conditions for consistency in the empirical risk minimization method Pattern Recognition and Image Analysis, vol. 1, no. 3, pp. 283-305, 1991.

[21] FRAUSTO, JUAN, RIVERA, RAFAEL et al. Fast hard Linear Problem resolution using Simulatead Annealing and Datzing's Rule. *Proc. of IASTED Int. Conf. in Artificial Intelligence and Soft Computing (ASC 2001)*, May 2001.

[22] FRAUSTO, JUAN, RIVERA, RAFAEL. A simplex Genetic Method for Solving the Klee Minty Cube. *Transactions on Systems, WSEAS, 2(1) pp 232-237.ISSN 1109-2777*, April 2002.

[23] BAZARAA, M., SHETTY, C.M. Nonlinear Programing, Theory and Algorithms. *School of Industrial and Systems Enginnering, Georgia Institute of Technology, Atlanta, Georgia. Jhon Wiley & Sons*, 1979.

[24] VAŠEK CHVÁTAL. Linear programming. *W.H. freeman and Company, New York*, 2000.

[25] DUDA, RICHARD et al. Pattern classification. *Wiley-Interscience, second edition*, 2001.

[26] GOLDBERG, DAVID E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley Publishing Conpany, 1989.

[27] FLORES R, HERNÁNDEZ A. Uso de algoritmos genéticos para detección de vectores de soporte en el entrenamiento de Máquinas de Soporte Vectorial. Memorias del Primer Congreso Mexicano de Computación Evolutiva, México 2003.

[28] VAŠIANGRONG, ZHANG, FANG, LIU. A Pattern Classification Method Based on GA and SVM. *Signal Processing, 2002 6th International Conference on , Volume: 1 , 26-30 Pages:110 - 113 vol.1*, Aug. 2002.

[29] BLAKE, C.L. and MERS. UCI Repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html, Department of Information and Computer Science, 1998.