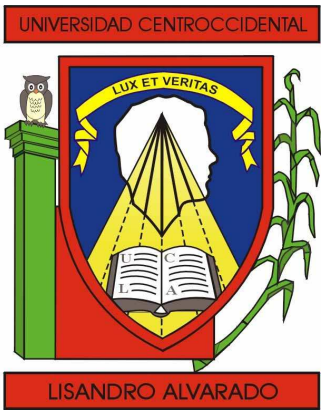
UNIVERSIDAD CENTROCCIDENTAL
"LISANDRO ALVARADO"
Decanato de Ciencias y Tecnología
Maestría en Ciencias
Mención: Optimización



MÁQUINAS VECTORIALES DE SOPORTE VÍA PROGRAMACIÓN SEMIDEFINIDA

Trabajo de Grado presentado por
Lcdo. Jacobo Alfonso Cortez Alejos.

Como requisito final para obtener el título de
Magister Scientiarum Mención Optimización

Área de Conocimiento: Matemática aplicada.
Tutor: Dr. Hugo Lara Urdaneta
Cotutor: Dr. Javier Hernández Benítez

Barquisimeto - Venezuela
Septiembre, 2014

MÁQUINAS VECTORIALES DE SOPORTE VÍA PROGRAMACIÓN SEMIDEFINIDA

Lcdo. Jacobo A. Cortez A.

RESUMEN

En este trabajo se relaciona la teoría de los núcleos en una máquina de vectores de soporte con la teoría de programación semidefinida. Se implementa un método alternante para averiguar la matriz kernel en una máquina de soporte vectorial, con la finalidad de solventar uno de los mayores inconvenientes para construir una SVM, el cual es la escogencia del kernel para un determinado problema. La matriz kernel es una matriz simétrica y definida positiva que codifica las posiciones relativas de todos los puntos de una muestra dada. En el trabajo se propone automatizar la escogencia de tal matriz, a través de un esquema alternante que haga uso de las bondades de la programación semidefinida.

A Dios, mis padres, hermanos y Anais

Índice general

Índice de figuras

Agradecimientos

Quiero expresar mi agradecimiento primeramente a Dios por darme fuerzas, por guiar mi camino, y darme sabiduría.

A mi tutor, el Dr. Hugo José Lara Urdaneta por su ayuda, sus conocimientos, ideas, consejos y sobre todo por la confianza que depositó en mí.

A mi amigo y profesor Javier Hernández, por sus consejos, palabras y conocimientos.

A mis padres Jacobo y Carmen por darme el don de la vida, por brindarme su apoyo, y por enseñarme lo importante del respeto y los valores. A mis hermanos Gerardo y Carmin, por sus comentarios, su apoyo y por sus buenos sentimientos.

A Anaís, por ser un pilar fundamental en este logro, por su ayuda, apoyo, comprensión; por estar presente para darme ánimos, fuerzas y esperanzas en la vida.

A los profesores Eibar Hernández, Maribel Perdomo, Rómulo Castillo, Jhonny Escalona, Freddy Torrealba y todos los que de una u otra forma han ayudado a mi formación académica y también por haber contribuido en mi formación como persona.

A mis amigos Ronald, Alexander, Erik, Mariana, Geraldine, Edgar y a todas las personas que de una u otra manera me han ayudado en el transcurso de mi carrera.

Introducción

Las máquinas vectoriales de soporte (SVM, por sus siglas en inglés) surgen como respuesta a una necesidad común en el aprendizaje de máquinas, como es la clasificación de datos según un criterio concreto. Debido a sus buenos fundamentos teóricos y su buena capacidad de generalización las SVM han llegado a ser en los últimos años uno de los métodos de clasificación más utilizados.

Una SVM es un modelo que representa los puntos de una muestra en el espacio, separándolos en clases con la mayor distancia posible entre las clases. Para ello generan separadores lineales o hiperplanos en espacios muy complejos de dimensión alta. Las SVM son aplicables en diversas áreas, como la medicina, genética, visión artificial, entre otras.

Un aspecto fundamental para la construcción de una SVM es la implementación de una función kernel, la cual está asociada a una matriz definida positiva llamada matriz kernel. Esta matriz kernel es una herramienta que permite llevar los datos originales a un espacio de dimensión mayor donde teóricamente exista una separación de los datos.

La implementación de una determinada función kernel en una SVM no es sencilla, puesto que no se cuenta con una regla predeterminada para saber cuál es el mejor kernel a usar en un determinado problema, es decir la escogencia de la función kernel está sujeta a un factor humano, y esto puede llevar a errores en la clasificación de los datos o que el kernel elegido no sea el óptimo.

Para intentar resolver este problema y tomando en cuenta que la función kernel está asociada a una matriz semidefinida positiva, se intenta relacionar la teoría

de las SVM y los kernels con la teoría de programación semidefinida. La programación semidefinida es la programación lineal sobre las matrices semidefinidas, y se ha convertido en una herramienta fundamental para la optimización y el modelado, ya que tiene una gran variedad de aplicaciones, y en años recientes se han generado algoritmos que resuelven de manera eficiente programas semidefinidos.

Combinar las bondades de la programación semidefinida con la teoría de SVM, es la idea para tratar de que la escogencia del kernel se realice de una forma más automática.

La estructura de este trabajo está diseñada de la siguiente forma: el capítulo 1, está dividido en dos secciones, la primera trata la teoría de las SVM, en la cual se exponen las ideas en detalle de diversos programas que permiten la construcción de una SVM, en su forma primal y en su forma dual. También se expone cuando los datos son linealmente separables y cuando no lo son. Además se trata el tema del kernel y su importancia en la construcción de una SVM. Mientras que la segunda trata sobre programación semidefinida; en ella se estudia la teoría de programación semidefinida, propiedades y las características principales de ella.

En el capítulo 2, se describe el esquema alternante SVM-SDP; en el que se enlazan ambas teorías y se explica en detalle el algoritmo desarrollado para resolver integralmente el problema de encontrar el kernel óptimo, tratado como un problema de programación semidefinida. Este problema se divide en dos etapas, en una etapa el algoritmo resuelve un problema de programación cuadrática convexa y en la otra etapa un problema de programación semidefinida mediante la implementación en el lenguaje de modelación YALMIP y el uso del resolvedor externo SEDUMI.

En el capítulo 3, se muestran resultados de la implementación del esquema alternante SVM-SDP. Se muestran corridas del algoritmo a datos linealmente separables y a datos que no son linealmente separables, también se muestran los vectores soporte arrojados por el algoritmo. Se establecen comparaciones entre el desempeño del algoritmo con otros algoritmos similares. Por último, se presenta el capítulo de conclusiones en el que se resumen las ventajas del algoritmo y también aspectos a ser mejorados de él.

Capítulo 1

Preliminares

La teoría de optimización es una rama de las matemáticas relativamente nueva y su estudio ha tomado una gran importancia en la actualidad. En la siguiente sección, describimos las ideas básicas de las máquinas de vectores de soporte desde el punto de vista de optimización.

1.1. Máquinas de soporte vectorial

Las máquinas de soporte vectorial o máquinas de vectores de soporte (Support Vector Machine, SVMs) son un conjunto de algoritmos de aprendizaje supervisado desarrollado por Vapnik (ver [15]) y su grupo de trabajo a principio de los años 80. La versatilidad de las SVM las hace aplicables en áreas muy variadas como: en visión artificial para el reconocimiento de caracteres, rostros, matrículas y objetos; en medicina, para la clasificación de exámenes radiológicos, TAC (Tomografía Axial Computarizada), y para el diagnóstico de tejido humano; en genética para predicción de genes; en la clasificación de documentos, entre muchas otras áreas.

Una SVM es un modelo que representa los puntos de una muestra en un es-

pacio, separando estos puntos en clases y separándolos entre sí con la mayor distancia posible, es decir, con un margen máximo. Se consideran dos conjuntos de puntos en $\mathbb{R}^n$ los cuales se etiquetan con P_+ (clase del +1) y P_- (clase del -1). Al denotar cualquiera de estos puntos por x, se puede construir una función f tal que $f(x) > 0$ si $x \in P_+$ y $f(x) < 0$ si $x \in P_-$. La función f es conocida como un clasificador. Así, dado un nuevo punto x, se puede usar f para clasificar x como perteneciente a P_+ (si $f(x) > 0$) o a P_- (si $f(x) < 0$).

Se inicia con la construcción de un clasificador lineal, el cual tiene la forma $f(x) = \omega^T x - \gamma$ donde $\omega \in \mathbb{R}^n$ y $\gamma \in \mathbb{R}$. La situación ideal es que el hiperplano definido por $f(x) = 0$ pudiese separar por completo los conjuntos P_+ y P_-, y de esta forma se obtendría que $f(x) > 0$ para $x \in P_+$ y $f(x) < 0$ para $x \in P_-$. Si tal (ω, γ) existe, se puede redefinir (ω, γ) como:

$$\frac{(\omega, \gamma)}{\min_{x \in P_+ \cup P_-} |\omega^T x - \gamma|}. \tag{1.1.1}$$

Así, por (1.1.1) se tiene que:

$$\begin{aligned} x \in P_+ \quad &\Rightarrow \quad f(x) = \omega^T x - \gamma \geq 1, \\ x \in P_- \quad &\Rightarrow \quad f(x) = \omega^T x - \gamma \leq -1. \end{aligned} \tag{1.1.2}$$

Ahora, para expresar estas condiciones como un sistema de desigualdades lineales, agrupamos cada uno de los puntos en una fila de la matriz A, la cual tiene $m = |P_+| + |P_-|$ filas, donde $|P_+|$ y $|P_-|$ denotan el número de puntos en P_+ y P_- respectivamente. Luego, se define una matriz diagonal D que etiqueta cada fila de A como:

$$D_{ii} = \begin{cases} 1, & \text{si el punto representado por la fila i de } A \text{ pertenece a } P_+; \\ -1, & \text{si el punto representado por la fila i de } A \text{ pertenece a } P_-. \end{cases}$$

Las condiciones (1.1.2) pueden ser escritas como:

$$D(A\omega - e\gamma) \geq e, \tag{1.1.3}$$

donde $e = (1, 1, \ldots, 1)^T$. La figura (1.1), muestra un hiperplano separador (ver [13]) $\omega^T x = \gamma$ para dos conjuntos de puntos, obtenidos hallando un par (ω, γ) que es factible para (1.1.3) junto a los hiperplanos soporte $\omega^T x - \gamma = \pm 1$.

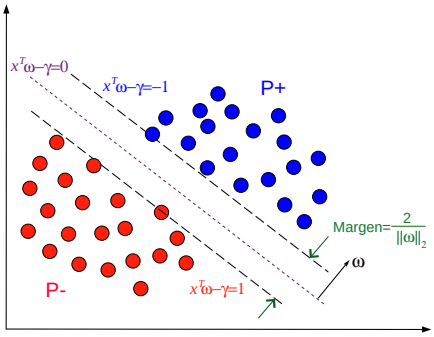


Figura 1.1: Caso linealmente separable

Ahora, si es posible separar las dos clases de puntos, entonces es deseable maximizar la distancia (margen) entre los hiperplanos soportes, los cuales en la Figura 1, son representados por las dos líneas segmentadas en color negro. Se puede mostrar que el margen de separación para cualquier norma $\|\cdot\|$ es

$$\frac{2}{\|\omega\|'},$$

donde $\|\omega\|'$ denota la norma dual (para detalles ver ([2])). Al utilizar la norma Euclídea la cual es su misma dual, entonces la maximización de $2/\|\omega\|'$ puede ser alcanzada por la minimización de $\|\omega\|$ o $\|\omega\|^2 = \omega^T\omega$. Por lo tanto, podemos resolver el siguiente programa para hallar el hiperplano separador con margen (Euclídeo) máximo:

$$\begin{array}{ll} \min\limits_{\omega,\gamma} & \frac{1}{2}\omega^T\omega \\ s.a. & D(A\omega - e\gamma) \geq \quad e. \end{array} \tag{1.1.4}$$

Los vectores soporte son los puntos x que están sobre los hiperplanos soporte que corresponden a los multiplicadores de Lagrange de las restricciones de (1.1.4) que son positivos. Estos corresponden a restricciones activas en (1.1.4). Es decir, tras obtener la solución se pueden determinar los vectores soporte mediante la condición de complementaridad de las condiciones KKT, ya que aquellos puntos en los que su multiplicador de Lagrange asociado cumple $\lambda_i \geq \epsilon > 0$, serán seleccionados como vectores soporte, mientras que los que tengan su multiplicador de Lagrange $\lambda_i \approx 0$ se descartan (ver [2]).

En la práctica, usualmente no es posible hallar un hiperplano que separe los dos conjuntos porque no existe tal hiperplano. En tales casos, el programa cuadrático (1.1.4) es infactible, pero se pueden definir otros problemas que identifiquen hiperplanos de separación "tan cerca de separar las clases como sea posible", en algún sentido. Para ello, se puede definir un vector y cuyas componentes indican la cantidad por la cual las restricciones son válidas como se sigue:

$$D(A\omega - e\gamma) + y \geq e, \quad y \geq 0. \tag{1.1.5}$$

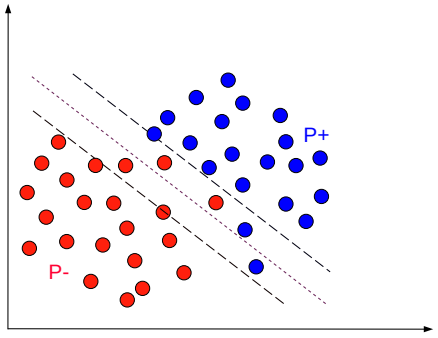


Figura 1.2: Caso linealmente no separable

Se puede medir la violación total sumando las componentes de y, y sumando

algunos múltiplos de esta cantidad del objetivo (1.1.4), para obtener

$$\begin{array}{ll} \min\limits_{\omega,\gamma,y} & \frac{1}{2}\omega^T\omega + \nu e^T y \\ s.a. & D(A\omega - e\gamma) + y \geq \quad e, \quad y \geq 0, \end{array} \tag{1.1.6}$$

donde ν es algún parámetro positivo. El problema (1.1.6) se refiere a una SVM (lineal).

La figura anterior, muestra dos conjuntos de puntos linealmente no separables y el hiperplano obtenido al resolver el problema (1.1.6). Los hiperplanos soportes son también mostrados. En esta formulación, los vectores soportes son los puntos desde cada conjunto P_- y P_+ que se encuentran en el lado erróneo de los hiperplanos delimitadores (o están en sus respectivos planos de delimitación y sus correspondientes multiplicadores de Lagrange son positivos). Nuevamente, estos puntos corresponden a restricciones activas del conjunto de restricciones $D(A\omega - e\gamma) + y \geq e$.

Usando u para denotar los multiplicadores de Lagrange para las restricciones $D(A\omega - e\gamma) + y \geq e$ en (1.1.6), las condiciones KKT (ver [16]) para este problema son:

$$\begin{aligned} 0 &= \omega - A^T D u, \\ 0 &= e^T D u, \\ 0 &\leq \nu e - u \perp y \geq 0, \\ 0 &\leq D(A\omega - e\gamma) + y - e \perp u \geq 0. \end{aligned}$$

Estas son las condiciones KKT para el siguiente problema el cual es el dual de (1.1.6)

$$\begin{array}{ll} \min\limits_{u} & \frac{1}{2}u^T DAA^T Du - e^T u \\ s.a. & 0 \leq u \leq \nu e \quad , \; e^T Du = 0. \end{array} \tag{1.1.7}$$

De hecho, a menudo es más conveniente resolver esta forma del problema dual

para u y entonces recuperar ω colocando $\omega = A^T Du$ (a partir de la primera condición KKT) y recuperando γ como el multiplicador de Lagrange para la restricción $e^T Du = 0$ en (1.1.7).

Se puede obtener un programa lineal alternativo a (1.1.6) reemplazando el término cuadrático $\frac{1}{2}\omega^T\omega$ por la norma-1 $\|\omega\|_1$ la cual corresponde a la medición del margen usando la norma-∞. Introduciendo un vector s cuyos elementos son valores absolutos de los correspondientes elementos de ω, obtenemos las siguientes formulaciones:

$$\begin{array}{ll} \underset{\omega,\gamma,s,y}{\text{mín}} & e^T s + \nu e^T y \\ s.a. & D(A\omega - e\gamma) + y \geq \quad e, \quad s \geq \omega \geq -s, \quad y \geq 0. \end{array} \tag{1.1.8}$$

Una formulación alternativa para (1.1.8) que parece ser más efectiva computacionalmente es el siguiente programa lineal, el cual podría ser resuelto de forma eficiente por el método simplex, o métodos de puntos interiores:

$$\begin{array}{lll} \underset{\omega^+,\omega^-,\gamma,y}{\text{mín}} & e^T(\omega^+ + \omega^-) + \nu e^T y & \\ s.a. & D(A\omega - e\gamma) + y \geq & e \\ & \omega = \omega^+ - \omega^- & \\ & \omega^+, \omega^-, y \geq 0, & \end{array} \tag{1.1.9}$$

note que $\omega_i^+ + \omega_i^- \geq |\omega_i|$, y en la solución se tendría que al menos un ω_i^+ o ω_i^- es nulo $\forall i$.

Las formulaciones anteriormente mencionadas se refieren a SVM lineales, ya que tratan de separar los puntos por hiperplanos (lineales). Un enfoque más general involucra un mapeo de cada punto en un espacio de dimensión mucho mayor vía una función Φ y así poder determinar el hiperplano que separa en este espacio de dimensión mayor $\mathbb{R}^h$. Cuando este hiperplano es mapeado de nuevo en el espacio original $\mathbb{R}^n$, él describe una superficie no lineal, dando mayor flexibilidad y un medio más potente para la clasificación.

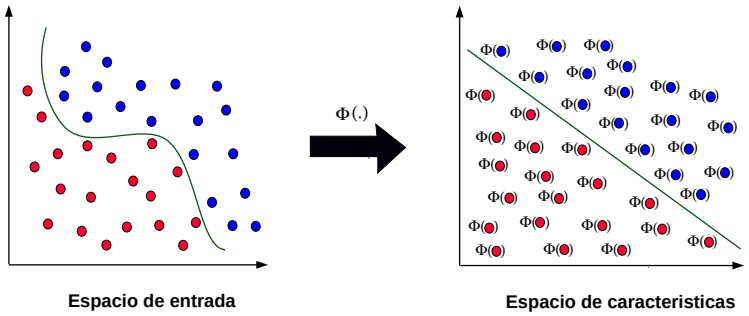


Figura 1.3: Función Φ

La dificultad en el cálculo de Φ y la posible alta dimensionalidad de $\mathbb{R}^h$ han sido importantes factores en el uso del kernel K como un generador de la superficie de separación no lineal en el espacio original de entradas $\mathbb{R}^n$ (ver [12]), pero la cual es lineal en el espacio de dimensión mayor $\mathbb{R}^h$.

Definición **1** *Sea* $A \in \mathbb{R}^{m\times n}$ *y* $B \in \mathbb{R}^{n\times l}$. *El kernel* $K(A,B)$ *mapea* $\mathbb{R}^{m\times n} \times \mathbb{R}^{n\times l}$ *en* $\mathbb{R}^{m\times l}$.

En particular si x e y son columnas en $\mathbb{R}^n$ entonces $K(x^T, A^T)$ es un vector fila en $\mathbb{R}^m$ ($x^T \in \mathbb{R}^{1\times n}, A^T \in \mathbb{R}^{n\times m}$), $K(x^T, y)$ es un número real y $K(A, A^T)$ es una matriz $m \times m$. Cuando se escribe $K(x, y)$ donde x, y son vectores nos referimos a K como una función, mientras que la expresión $K(A, A^T)$ es una matriz inducida por una función kernel, ya que sus entradas vienen dadas por la eveluación de la misma.

Ahora, involucremos el concepto de kernel en las SVM. Nuevamente consideremos un conjunto A de m puntos en el espacio n-dimensional $\mathbb{R}^n$, representados por la matriz $A \in \mathbb{R}^{m\times n}$. Cada punto A_i, $i = 1, \ldots, m$ pertenece a la clase del 1 o a la clase del -1 dependiendo si D_{ii} es 1 o -1. La meta es tratar de discriminar entre las clases 1 y -1 por una superficie no lineal, inducida por algún kernel $K(A, A^T)$, como sigue:

$$K(x^T, A^T)Du = \gamma,$$

donde $K(x^T, A^T) \in \mathbb{R}^m$ acorde a la Definición 1. Los parámetros $u \in \mathbb{R}^m$ y $\gamma \in \mathbb{R}$

son determinados resolviendo un programa matemático, típicamente cuadrático o lineal, como ya se han mencionado anteriormente. Un punto $x \in \mathbb{R}^n$ es clasificado en la clase 1 o -1 de acuerdo a la función de decisión

$$(K(x^T, A^T)Du - \gamma)_*,$$

la cual vale 1 o 0 respectivamente (para un vector $x \in \mathbb{R}^n$ la función paso x_* de x está definida como un vector de unos y ceros en $\mathbb{R}^n$, con unos que corresponden a las componentes positivas de x y ceros que corresponden a las componentes no positivas).

Para poder definir formalmente una función kernel, se introduce el concepto de Condición de Mercer, como sigue:

Definición **2 (Condición de Mercer (ver [2]))** *Para toda función g tal que* $\int g(x)^2 dx$ *es finita, entonces la función kernel K tiene que cumplir que* $\int K(x, y)g(x)g(y)dxdy \geq 0$.

La función kernel K implícitamente define una aplicación no lineal de $\mathbb{R}^n$ a algún espacio $\mathbb{R}^h$ donde h puede ser mucho más grande que n. En particular si el kernel K es un producto interno bajo la condición de Mercer entonces, para x e y en $\mathbb{R}^n$

$$K(x, y) = \Phi(x)^T \Phi(y),$$

y la superficie de separación viene dada por:

$$\Phi(x)^T \Phi(A^T)Du = \gamma,$$

donde Φ es una función, no fácilmente calculable, de $\mathbb{R}^n$ a $\mathbb{R}^h$, y $\Phi(A^T) \in \mathbb{R}^{k \times m}$ resulta de aplicar Φ a las m columnas de A^T.

Formalmente, una función kernel se define como:

Definición **3 (Función núcleo (ver [2]))** *Una función núcleo o kernel es un producto interno en el espacio de características que tiene su equivalente en el espacio de entrada:*

$$K(x, x^T) = \phi(x)^T \phi(x^T),$$

donde K es una función simétrica definida positiva y debe cumplir la condición de Mercer.

Cada función kernel está asociada a una matriz kernel, ella se define formalmente por:

Definición **4 (Matriz kernel (ver [8]))** *Dado un conjunto finito de entrada* $\mathcal{X} = \{x_i\}_{i=1}^n$. *Una matriz kernel es una matriz cuadrada* $\mathrm{K} \in \mathbb{R}^{n\times n}$ *tal que* $\mathrm{K}_{ij} = K(x_i, x_j)$ *con* $x_i, x_j \in \mathcal{X}$ *y K una función kernel.*

Cabe destacar que cada matriz $K(A, A^T)$ está asociada a una función kernel, puesto que sus entradas son generadas por una función kernel, por lo tanto $K(A, A^T)$ se trata de una matriz kernel. Ahora, como es difícil calcular Φ, y la dimensión de $\mathbb{R}^{\mathrm{h}}$ es alta, entonces se usa $K(A, A^T)$ para generar una superficie de separación no lineal en $\mathbb{R}^n$, la cual es lineal en $\mathbb{R}^h$. Ahora, la superficie de separación escrita en términos de funciones Kernel conserva esta ventaja y es lineal en términos de sus parámetros u, γ.

A continuación, se plantea un programa matemático que genera tal superficie para un kernel general, una máquina de soporte vectorial general vendría dada por:

$$\begin{array}{ll} \min\limits_{u,\gamma,y} & \nu e^T y + f(u) \\ s.a. & D(K(A, A^T)Du - e\gamma) + y \geq \quad e \\ & y \geq 0, \end{array} \tag{1.1.10}$$

aquí f es alguna función convexa sobre $\mathbb{R}^m$, típicamente una norma o seminorma, y como ya se ha visto ν es algún parámetro positivo de los pesos del error de separación $e^T y$ contra el parámetro u de la superficie de separación.

La minimización de u puede ser interpretada en una de dos maneras, puede ser vista como la minimización de los vectores soporte, es decir, las restricciones en (1.1.10) con multiplicadores positivos. Una interpretación más convencional es que se trata de la maximización de alguna medida de la distancia o margen entre los planos de delimitación paralelos en $\mathbb{R}^h$, bajo hipótesis apropiadas, tal f es una función cuadrática inducida por un kernel definido positivo.

Ahora, consideremos máquinas de vectores de soporte que incluyan una estándar y las cuales son obtenidas colocando f de (1.1.10) como una función cuadrática convexa $f(u) = \frac{1}{2}u^T Hu$, donde $H \in \mathbb{R}^{m\times m}$ es alguna matriz simétrica definida positiva. El programa matemático (1.1.10) viene a ser un programa cuadrático convexo:

$$\begin{array}{ll} \underset{u,\gamma,y}{\text{mín}} & \nu e^T y + \frac{1}{2}u^T Hu \\ s.a. & D(K(A, A^T)Du - e\gamma) + y \geq \quad e \\ & y \geq 0, \end{array} \tag{1.1.11}$$

el dual de Wolfe (ver [1]) de este programa cuadrático convexo es:

$$\begin{array}{ll} \underset{r\in\mathbb{R}^m}{\text{mín}} & \frac{1}{2}r^T DK(A, A^T)DH^{-1}DK(A, A^T)Dr - e^T r \\ s.a. & e^T Dr = 0 \\ & 0 \leq r \leq \nu e. \end{array} \tag{1.1.12}$$

Además, la variable primal u está relacionada con la variable dual r por:

$$u = H^{-1}DK(A, A^T)Dr. \tag{1.1.13}$$

Si se supone que el kernel $K(A, A^T)$ es simétrica definida positiva y se coloca $H = DK(A, A^T)D$, entonces el problema dual (1.1.12) genera el problema dual de

las máquinas de vectores de soporte estándar con $u = r$

$$\begin{array}{ll} \underset{u\in\mathbb{R}^m}{\text{mín}} & \frac{1}{2}u^T DK(A, A^T)Du - e^T u \\ s.a. & e^T Du = 0 \\ & 0 \leq u \leq \nu e. \end{array} \tag{1.1.14}$$

La suposición de ser definida positiva sobre $K(A, A^T)$ en (1.1.14) puede ser relajada a semidefinida positiva mientras se mantiene el programa cuadrático convexo (1.1.11), con $H = DK(A, A^T)D$, con el dual directo (1.1.14) sin utilizar (1.1.12) y (1.1.13). El hecho de que $r = u$ en la formulación dual (1.1.14), muestra que la variable u que aparece en la formulación original (1.1.11) es también el vector dual de multiplicadores del primer conjunto de restricciones de (1.1.11). Por tanto el término cuadrático en la función objetivo (1.1.11) puede pensarse como la supresión de muchos multiplicadores de vectores soporte y minimizar el número de tales vectores soporte. Como ya se ha mencionado, ésta es otra interpretación de máquinas de vectores de soporte estándar que es usualmente interpretada como la maximización del margen o distancia entre planos paralelos.

Esto lleva a la idea de usar otros valores para H que no sean $DK(A, A^T)D$. Una escogencia en particular es interesante, porque no coloca restricción sobre K: sin simetría, sin ser definido positivo o semidefinido, y no siempre continuo. Esta escogencia es $H = I$ en (1.1.11) la cual lleva el problema dual (1.1.12) con $H = I$ y $u = DK(A, A^T)Dr$ como sigue:

$$\begin{array}{ll} \underset{r\in\mathbb{R}^m}{\text{mín}} & \frac{1}{2}DK(A, A^T)K(A, A^T)^T Dr - e^T r \\ s.a. & e^T Dr = 0 \\ & 0 \leq r \leq \nu e. \end{array} \tag{1.1.15}$$

Es claro que $K(A, A^T)K(A, A^T)^T$ es semidefinido positivo sin suposiciones sobre $K(A, A^T)$, y por tanto el problema (1.1.15) es resoluble para cualquier kernel

$K(A, A^T)$. De hecho por la Proposición 1, el programa cuadrático (1.1.11) es resoluble para cualquier matriz simétrica definida positiva H, y por dualidad de programación cuadrática también lo es su problema dual (1.1.12), la solución r puede ser inmediatamente usada para generar una función de decisión. Así, que de cierta manera se tiene libertad de elegir cualquier H para generar una máquina de vectores de soporte. Aunque se necesitan experimentos para determinar cuáles son las opciones más adecuadas para la escogencia de H.

Para concluir esta sección se presentan las funciones kernel más comúnmente usadas:

- **Kernel perceptron**: El Perceptron es un muy importante modelo de aprendizaje relacionado con el aprendizaje en redes neuronales. Se ha demostrado que el trabajo con el kernel Perceptron equivale a construir una red neuronal con un número infinito de neuronas.

$$K(x_i, x_j) = \|x_i - x_j\|.$$

- **Núcleo Gaussiano**: A partir del kernel Perceptron obtenemos un equivalente del popular kernel exponencial.

$$K(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right), \text{donde } \sigma > 0.$$

- **Polinomial no homogéneo**: Una de las funciones kernel más usadas es el el polinomial homogéneo, este viene dado por:

$$K(x_i, x_j) = (\langle x_i, x_j\rangle + c)^d, \text{ con } d \in \mathbb{N},\ c \geq 0.$$

- **Sigmoide o kernel hiperbólico**: Consiste en el uso de la tangente hiperbólica en conjunto con los parámetros δ y θ.

$$K(x_i, x_j) = \tanh(\delta\langle x_i, x_j\rangle + \theta), \text{ donde } \delta > 0,\ \theta < 0.$$

1.2. Programación semidefinida

En esta sección se introduce el la teoría de programación semidefinida (SDP), que se ha convertido en una herramienta importante en el área de optimización, debido a sus múltiples aplicaciones en la teoría de control, la optimización de autovalores, optimización combinatoria, entre otras. La programación semidefinida es la programación lineal sobre matrices semidefinidas. Hoy en día es una de las herramientas de modelado y optimización básicas junto con la programación lineal y cuadrática, ya que se han desarrollado algoritmos eficientes para programas semidefinidos.

Brevemente establecemos alguna notación que usaremos en esta sección S^n denota el conjunto de matrices simétricas $\mathbb{R}^{n\times n}$. Para $X \in S^n$, $X \geq 0$ significa que X es semidefinida positiva positiva, y $S^n_{\geq 0}$ denota el cono de matrices semidefinidas. La expresión $\lambda_i(A)$ denota el i-ésimo autovalor asociado a la matriz A.

Recordemos que,

Definición **5 (Matriz simétrica semidefinida positiva (definida positiva) (ver [6]))**
$A \in S^n$ es semidefinida positiva si $x^T Ax \geq 0$, $\forall x \in \mathbb{R}^n$.
$A \in S^n$ es definida positiva si $x^T Ax > 0$, $\forall x \in \mathbb{R}^n \backslash \{0\}$.

Ahora, consideremos el producto interno:

$$\langle A, B\rangle = tr(A^T B) = \sum_{i,j=1}^{n} A_{ij}B_{ij} \,,$$

para $A, B \in \mathbb{R}^{n\times n}$. $tr(A) = \langle I, A\rangle$ denota la traza de A.

Como se puede ver, la traza $tr(\cdot)$ es la suma de los elementos de la diagonal de una matriz cuadrada. Ella es una función lineal. También tenemos que la norma asociada con el producto interno de matrices es la norma de Frobenius

$$\|A\|_F = \sqrt{\langle A, A\rangle}. \tag{1.2.1}$$

También tengamos en cuenta las siguientes caracterizaciones de las matrices semidefinidas positivas:

Proposición **1 (Caracterización de matrices semidefinidas positivas (ver [6]))** *Para $A \in S^n$ las siguientes declaraciones son equivalentes:*

1. *A es semidefinida positiva*
2. *$\lambda_i(A) \geq 0$, $i = 1, \ldots, n$*
3. *Existe $C \in M_{m,n}$ tal que $A = C^T C$. Con C tal que rango(C) = rango(A)*
4. *$\langle A, B \rangle \geq 0$ para todo $B \in S^n_{\geq 0}$.*

Una interpretación interesante de una factorización $A = C^T C \geq 0$ es obtenida al ver las columnas de C como vectores v_i. Los elementos a_{ij} son productos escalares $\langle v_i, v_j \rangle$ de los vectores v_i y v_j. Así, A puede ser vista como una matriz de Gram de los vectores $v_1, \ldots, v_n$.

Otro concepto de interés es el siguiente:

Definición **6 (Desigualdad lineal matricial)** *Una desigualdad lineal matricial, abreviada LMI, es una restricción de la forma:*

$$F(u) := F_0 + u_1 F_1 + \cdots + u_q F_q \preceq 0.$$

Aquí, u es el vector de variables de decisión, y $F_0, \ldots, F_q$ son matrices simétricas dadas.

Muchas desigualdades pueden ser representadas como LMIs, a continuación mostramos algunos ejemplos de ello.

Ejemplo 1.2.1 *La restricción lineal con variable x, $a_i^T x \leq b_i$, donde $i = 1, \ldots, k$*

puede ser representada como la LMI

$$diag(b_1 - a_1^T x, \ldots, b_k - a_k^T x) = \begin{pmatrix} b_1 - a_1^T x & 0 & \cdots & 0 \\ 0 & b_2 - a_2^T x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & b_k - a_k^T x \end{pmatrix} \geq 0$$

Si se trata de una restricción lineal de igualdad como $a_i^T x = b_i$, $i = 1, \ldots, k$, *entonces la escribir como:*

$$a_i^T x \leq b_i, \;\; a_i^T x \geq b_i \;\; i = 1, \ldots, k$$

lo cual es equivalente a la LMI:

$$diag(b_1 - a_1^T x, \ldots, b_k - a_k^T x, a_1^T x - b_1, \ldots, a_k^T x - b_k) \geq 0.$$

Ahora, entremos al contexto de programas semidefinidos. En comparación con la programación lineal estándar, el vector de variables $x \in \mathbb{R}^n_+$ (el ortante no negativo, cono poliedral) es remplazado por una matriz variable $X \in S^n_{\geq 0}$ (el cono no poliedral de matrices semidefinidas positivas). Los programas semidefinidos surgen de manera natural en problemas cuyos datos están dados en matrices.

La forma típica de un programa semidefinido (ver [9]) es:

$$\begin{aligned} p^* = \sup_X \quad & \langle C, X \rangle \\ s.a. \quad & \langle A_j, X \rangle = b_j \\ & X \geq 0; \;\; j \in \{1, \ldots, m\}. \end{aligned} \tag{P}$$

Aquí, $A_1, \ldots, A_m \in S^n$, $b \in \mathbb{R}^n$, $C \in \mathbb{R}^{n \times n}$, son los datos. El conjunto factible para (SDP) es $\mathfrak{F} = S^n_{\geq 0} \cap W$, donde W es el espacio afín

$$W = \{x \in S^n | \langle A_j, X \rangle = b_j, j = 1, \ldots, m\}.$$

Escribimos supremo en lugar de máximo en virtud de que este pudiera no ser

alcanzado en (P). Podemos escribir el dual de (P) de la forma:

$$\begin{aligned} d^* = \text{ínf} \quad & b^T y \\ s.a. \quad & \textstyle\sum_{j=1}^{m} y_j A_j - C \succeq 0. \end{aligned} \tag{D}$$

Este problema dual tiene una variable y_j para cada restricción primal. Nótese que en el problema (D) las restricciones duales son un sistema LMI. Los siguientes hechos relacionan los problemas primales y duales. Son simples pero muy importantes.

Lema **1** *Sea* (X, y) *un par de soluciones factibles primal-dual:*

1. *(Dualidad débil)* $\langle C, X\rangle \leq b^T \mathrm{y}$*, por tanto* $p^* \leq d^*$.
2. *(Holguras complementarias) Suponemos que el problema primal alcanza su supremo en X, que el dual alcanza su ínfimo en* y*, y que* $p^* = d^*$*. Entonces se verifica* $\langle X, \sum_{j=1}^{m} \mathrm{y}_j A_j - C\rangle = 0$*, y también que* $\langle C, X\rangle = b^T \mathrm{y}$.
3. *(Criterio de Optimalidad) Si se cumple* $\langle C, X\rangle = b^T \mathrm{y}$*, entonces el supremo de (P) se alcanza en X, el ínfimo de (D) se alcanza en* y*; además* $d^* = p^*$.

La diferencia $d^* - p^*$ es llamada brecha de dualidad. En general puede haber una brecha de dualidad positiva ($d^* > p^*$). Cuando no existe brecha de dualidad ($d^* = p^*$) decimos que se cumple la dualidad fuerte. A continuación enenunciamos el teorema de dualidad fuerte.

Proposición **2 (Dualidad Fuerte)** *Considere los problemas (P) y (D).*

1. *Supongamos que (D) es acotado inferiormente* ($d^* > -\infty$) *y que es estrictamente factible. Entonces (P) alcanza su supremo (i.e.* $\exists X \in \mathfrak{F} : p^* = \langle C, X\rangle$*) y no hay brecha de dualidad* ($p^* = d^*$).

2. *Supongamos que* (P) *es acotado superiormente* ($p^* < \infty$) *y que es estrictamente factible. Entonces el problema dual* (D) *alcanza su ínfimo. (i.e.* $d^* = b^T$y *para algún* y *factible) y no hay brecha de dualidad* ($p^* = d^*$).

Ahora, consideremos por simplicidad el caso de un SDP con una restricción LMI sencilla, y sin restricciones afines:

$$\begin{aligned} p^* = \min_x \quad & c^T x \\ s.a. \quad & F(x) := F_0 + \sum_{i=1}^{m} x_i F_i \geq 0 \end{aligned} \tag{1.2.2}$$

con $c \in \mathbb{R}^n$, $F_i \in S^n$, $i = 0, \ldots, m$. El Lagrangiano se define por:

$$\mathcal{L}(x, Y) = c^T x - tr(F(x)Y), \tag{1.2.3}$$

y ésta es una herramienta para el cálculo del problema dual, ya que

$$d^* = \max_{Y \geq 0} \min_x \mathcal{L}(x, Y)$$

lo cual da como resultado:

$$\begin{aligned} d^* = \max_Y \quad & -tr(F_0 Y) \\ s.a. \quad & tr(F_i Y) = c_i,\ i = 1, \ldots, n \\ & Y \geq 0. \end{aligned} \tag{1.2.4}$$

Como fue descrito anteriormente, si los problemas (1.2.2) y (1.2.4) son estrictamente factibles entonces la brecha de dualidad es cero, y ambos valores son alcanzados. Entonces, un par (x, Y) es primal-dual optimal si y sólo si las condiciones KKT:

1. Factibilidad primal: $F(X) \geq 0$.
2. Factibilidad dual: $tr(F_i Y) = c_i,\ i = 1, \ldots, n; \quad Y \geq 0$.
3. Holguras complementarias: $F(x)Y = O$, donde O denota la matriz nula,

se cumplen.

Parte del auge reciente del empleo de SDP es el uso de los algoritmos de puntos interiores, y su eficiencia. Los algoritmos de puntos interiores son algoritmos de tiempo polinomial que se han mostrado muy eficientes en problemas prácticos.

Recordemos de dualidad de programación lineal que para un problema de la forma $\text{mín}_x\{c^T x : Ax \leq b\}$ con variables duales y, una la las condiciones KKT eran $\forall i : y_i(b - Ax)_i = 0$. Esto puede ser escrito de manera compacta como $diag(y)diag(b - Ax) = 0$ lo cual es similar a las condiciones KKT para SDP (con $Y = diag(y)$). Esto no es de sorprender, pues los problemas SDP incluyen los problemas LP como casos particulares (ver [9]).

Además, la programación lineal puede ser resuelta por la programación cuadrática que a su vez, puede ser resuelta por la programación de cono de segundo orden, y esta puede ser resuelta por la programación semidefinida. Por lo tanto, en teoría, sólo es necesario un solucionador eficiente de programación semidefinida. En este trabajo, se empleó el lenguaje de modelado YALMIP junto con el solucionador externo SEDUMI (ver Apéndices para detalles).

Desde el punto de vista de los modelos, la programación semidefinida se ha mostrado como herramienta eficiente para construir relajaciones de problemas combinatorios, que son conocidos por ser NP-hard.

Nuestro interés en (SDP) pasa por la posibilidad de encontrar matrices SDP que son escogidas de forma óptima, de acuerdo a algún criterio descrito en relaciones lineales. Por ello, el problema de encontrar el Kernel óptimo en una máquina de soporte vectorial es una posibilidad para el uso de programación semidefinida.

Capítulo 2

Esquema alternante SVM-SDP

Los algoritmos de aprendizaje basados en kernel en general trabajan llevando datos a un espacio de Hilbert, y buscando relaciones lineales en tal espacio. El incrustamiento es desempeñado implícitamente, especificando el producto interno entre cada par de puntos en lugar de especificar sus coordenadas explícitamente, lo que en cierta literatura es conocido como el truco del kernel.

En ocasiones se refiere a la matriz kernel como una matriz de Gram, recordemos que:

Definición **7 (Matriz de Gram)** *La matriz de Gram de un conjunto de vectores* $v_1, \ldots, v_n$ *en un espacio prehilbertiano (un espacio vectorial dotado de un producto interno, no necesariamente debe ser completo) es la matriz que define el el producto escalar, cuyas entradas vienen dadas por* $A_{ij} = \langle v_i, v_j \rangle$.

Ahora, ya que las funciones kernels son generalizaciones de productos internos (ver [14]) y teniendo en cuenta el capítulo anterior tenemos las siguientes características fundamentales de la matriz kernel:

1. Ella es simétrica.

2. Es semidefinida positiva.

3. Ya que sus entradas están completamente dadas en términos de productos internos entre todos los pares de puntos $\{x_i\}_{i=1}^n$, ella completamente determina las posiciones relativas de estos puntos en el espacio de características.

Además, ya que estamos considerando conjuntos de entrada finitos $\mathcal{X} = \{x_i\}_{i=1}^n$, se pueden caracterizar de manera formal los kernels de la siguiente manera:

Proposición **3 ((ver [8]))** *Cada matriz simétrica y definida positiva es una matriz kernel. Recíprocamente, cada matriz kernel es simétrica y definida positiva.*

Con la finalidad de estudiar el comportamiento del kernel en una máquina de vectores de soporte mediante las bondades de la programación semidefinida, intentamos resolver una de las principales desventajas de una SVM, la cual es: la escogencia del mejor núcleo para ser implementado en un determinado problema, es decir, la elección de un kernel óptimo.

Una de las razones por la cual el problema de encontrar el kernel óptimo puede ser tratado como un problema de programación semidefinida, se debe a que la matriz kernel que se está considerando debe ser necesariamente una matriz definida positiva.

La escogencia del kernel, para ser usado en la construcción de una SVM es generalmente realizado por un usuario humano, y esto trae inconvenientes debido a que esta situación genera subjetividad, ya que en la práctica no se tiene una justificación a priori sobre que kernel usar en una determinada situación o que kernel no usar. Para tratar de solventar este inconveniente, se propone un método alternante que hace uso de la programación semidefinida para que la determinación del kernel sea de una manera más automática, y este basada en los datos.

En [4], Fung y colaboradores presentan un algoritmo que escoge de forma automática un kernel como combinación convexa de un conjunto potencialmente grande de diferentes kernel semidefinidos. Un enfoque de SDP fue presentado en [8], donde se calculan los parámetros de las SVM y se elige una combinación lineal de los kernels partiendo de un problema de programación semidefinida.

La propuesta que se introduce en el trabajo consiste en resolver integralmente el problema:

$$\begin{aligned} \min_{u,\gamma,y,\tilde{K}} \quad & \nu e^T y + \tfrac{1}{2} u^T H u + g(\tilde{K}) \\ s.a. \quad & D(\tilde{K} D u - e\gamma) + y \geq e \\ & y \geq 0, \\ & \tilde{K} \succeq 0, \end{aligned} \tag{2.0.1}$$

mediante la implementación de un esquema alternante al que llamaremos SDP-SVM; recordemos que $H \in \mathbb{R}^{m \times m}$ es alguna matriz simétrica definida positiva, ν es algún parámetro positivo. La función g es tal que preserve de alguna manera la esencia de los datos.

El esquema alternante consiste en primero, fijar (u^k, γ^k, y^k), y calcular la matriz $\tilde{K}$ mediante la resolución del siguiente programa

$$\begin{aligned} \min_{\tilde{K}} \quad & \nu e^T y^k + \tfrac{1}{2} (u^k)^T H u^k + g(\tilde{K}) \\ s.a. \quad & D(\tilde{K} D u^k - e\gamma^k) + y^k \geq e \\ & \tilde{K} \succeq 0, \end{aligned} \tag{2.0.2}$$

para obtener $\tilde{K}^k$, luego fijamos $\tilde{K}^k$ y se resuelve

$$\begin{aligned} \min_{u,\gamma,y} \quad & \nu e^T y + \tfrac{1}{2} u^T H u + g(\tilde{K}^k) \\ s.a. \quad & D(\tilde{K}^k D u - e\gamma) + y \geq e \\ & y \geq 0, \end{aligned} \tag{2.0.3}$$

para obtener (u^k, γ^k, y^k).

El criterio de parada usado es realizar el proceso hasta que la diferencia entre los valores objetivos de los problemas (2.0.2) y (2.0.3) sea menor que cierta tolerancia $\epsilon > 0$, esto debido a que al cumplirse esta condición, los valores objetivos serán muy cercanos entre sí, y por ende mediante el esquema alternante se está resolviendo de forma integral el problema (2.0.1).

Notemos, que una de las principales ideas detrás del esquema alternante SVM-

SDP es que al calcular de manera alterna las soluciones de los problemas, no necesitamos saber la función núcleo, ni la aplicación Φ, ni las coordenadas de los puntos $\Phi(x_i)$. De hecho, al correr el algoritmo los datos son llevados implícitamente a un espacio de Hilbert con sólo comprobar que la matriz kernel es simétrica y definida positiva.

Otro aspecto a tomar en cuenta, es que el problema que se está considerando inicialmente resolver de manera integral es (2.0.1), el cual correspondería (cuando la matriz kernel $\tilde{K}$ está fija) a un programa matemático primal del tipo (1.1.11), que viene a ser un programa cuadrático convexo. Al algoritmo que implementa el esquema alternante SVM-SDP a la resolución de (2.0.1) lo denominaremos esquema alternante SVM-SDP (primal), y lo resumimos a continuación.:

Algoritmo 1 Algoritmo esquema alternante SVM-SDP (primal)

DATOS DE ENTRADA: $x^0 = (u^0, \gamma^0, y^0)$, $\epsilon > 0$, ν, A
MIENTRAS: $|FO1 - FO2| < \epsilon$
Fijar (u^k, γ^k, y^k), y calcular $\tilde{K}^k$ mediante la solución de

$$\begin{array}{cc} \min\limits_{\tilde{K}} & FO2 = \nu e^T y^k + \frac{1}{2}(u^k)^T H u^k + g(\tilde{K}) \\ s.a. & D(\tilde{K} D u^k - e\gamma^k) + y^k \geq e \\ & \tilde{K} \succeq 0, \end{array}$$

para obtener la matriz $\tilde{K}^k$, luego fijamos $\tilde{K}^k$ y calculamos (u^k, γ^k, y^k) mediante la solución de

$$\begin{array}{cc} \min\limits_{u,\gamma,y} & FO1 = \nu e^T y + \frac{1}{2} u^T H u + g(\tilde{K}^k) \\ s.a. & D(\tilde{K}^k D u - e\gamma) + y \geq e \\ & y \geq 0, \end{array}$$

Observemos que podríamos permutar de posición en nuestro esquema alterante los problemas (2.0.2) y (2.0.3); y de esta manera obtener un esquema alternante similar, sólo que el algoritmo en lugar de comenzar con vector inicial $x^0 = (u^0, \gamma^0, y^0)$, el algoritmo comenzaría con matriz inicial K^0.

Así obtendríamos el siguiente proceso:

Algoritmo 2 Algoritmo esquema alternante SVM-SDP (primal)

DATOS DE ENTRADA: $\tilde{K}^0$, $\epsilon > 0$, ν, A
MIENTRAS: $|FO1 - FO2| < \epsilon$
Fijar $\tilde{K}^k$, y calcular (u^k, γ^k, y^k) mediante la solución de

$$\begin{array}{ll} \min\limits_{u,\gamma,y} & FO1 = \nu e^T y + \frac{1}{2} u^T H u + g(\tilde{K}^k) \\ s.a. & D(\tilde{K}^k D u - e\gamma) + y \geq e \\ & y \geq 0, \end{array}$$

para obtener (u^k, γ^k, y^k), luego fijamos (u^k, γ^k, y^k) y calculamos $\tilde{K}^k$ mediante la solución de

$$\begin{array}{ll} \min\limits_{\tilde{K}} & FO2 = \nu e^T y^k + \frac{1}{2} (u^k)^T H u^k + g(\tilde{K}) \\ s.a. & D(\tilde{K} D u^k - e\gamma^k) + y^k \geq e \\ & \tilde{K} \geq 0, \end{array}$$

Ahora, como se mencionó en el primer capítulo, en la practica el problema que usualmente se resuelve es el problema dual de (1.1.11), es decir el problemas de la forma (1.1.12), y en particular de la forma (1.1.14).

Por ello es conveniente implementar el algoritmo SVM-SDP a este tipo de problemas. Consideremos resolver de manera integral el problema:

$$\begin{array}{ll} \min\limits_{u,\tilde{K}} & \frac{1}{2} u^T D \tilde{K} D u - e^T u + g(\tilde{K}) \\ s.a. & e^T D u = 0, \\ & 0 \leq u \leq \nu e, \\ & \tilde{K} \geq 0. \end{array} \tag{2.0.4}$$

A este tipo de algoritmo lo denominademos esquema alternante SVM-SDP dual, y lo resumimos a continuación:

Algoritmo 3 Algoritmo esquema alternante SVM-SDP (dual)

DATOS DE ENTRADA: $\tilde{K}^0$, $\epsilon > 0$, ν, A
MIENTRAS: $|FO1 - FO2| < \epsilon$
Fijar $\tilde{K}^k$, y calcular u^k mediante la solución de

$$\begin{array}{ll} \min\limits_{u} & FO1 = \frac{1}{2} u^T D \tilde{K}^k D u - e^T u + g(\tilde{K}^k) \\ s.a. & e^T D u = 0, \\ & 0 \leq u \leq \nu e. \end{array}$$

para obtener el vector de multiplicadores de Lagrange u^k, luego fijamos u^k y calculamos $\tilde{K}^k$ mediante la solución del programa semidefinido

$$\begin{array}{ll} \min\limits_{\tilde{K}} & FO2 = \frac{1}{2} {u^k}^T D \tilde{K} D u^k - e^T u^k + g(\tilde{K}) \\ s.a. & \tilde{K} \succeq 0. \end{array}$$

Este enfoque tiene ventajas respecto al enfoque primal, puesto que los multiplicadores de Lagrange (como se vio en el primer capítulo) $u^* \in \mathbb{R}^n$ juegan un papel fundamental en la selección de los vectores de soporte por eliminación de un patrón x_i cuando $0 < u_i \leq \varepsilon$, donde $\varepsilon > 0$ (como veremos en el próximo capítulo) permite ajustar la cantidad de vectores de soporte a ser seleccionados.

La estrecha relación entre los vectores soporte y los multiplicadores de Lagrange, resulta de la condición de complementariedad, ya que aquellos puntos en los que su vector de multiplicador de Lagrange asociado sea muy cercano a cero son aquellos que se descartan, es decir los puntos que no hacen de la restricción correspondiente una restricción activa, y los que quedan son los llamados a ser vectores soporte, es decir los puntos que hacen activa la restricción.

Una ventaja que tiene el esquema alternante SVM-SDP dual con respecto al llamado esquema alternante SVM-SDP primal, es que si bien en esencia ambos usan el mismo principio, el tiempo de ejecución en el dual es más corto con respecto a la selección de los vectores de soporte, esto debido a que en éste se está calculando de manera alterna los multiplicadores de Lagrange y la matriz kernel hasta lograr la tolerancia requerida; mientras que en primal se está calculando los

parámetros del hiperplano y la matríz kernel de manera alterna, y luego de acanzar la tolerancia requerida, la matriz kernel óptima K^* debe ser insertada en un programa del tipo (1.1.14) para obtener el vector de multiplicadores de Lagrange u^* para luego proceder a seleccionar los vectores de soporte.

Al igual que en el caso primal, se puede permutar de posición los programas involucrados en el algoritmo anterior y obtener:

Algoritmo 4 Algoritmo esquema alternante SVM-SDP (dual)

DATOS DE ENTRADA: $u^0, \epsilon > 0, \nu, A$
MIENTRAS: $|FO1 - FO2| < \epsilon$
Fijar u^k, y calcular $\tilde{K}^k$ mediante la solución de

$$\begin{array}{ll} \min\limits_{\tilde{K}} & FO2 = \frac{1}{2}{u^k}^T D\tilde{K}Du^k - e^T u^k + g(\tilde{K}) \\ s.a. & \tilde{K} \succeq 0. \end{array}$$

para obtener la matriz $\tilde{K}^k$, luego fijamos $\tilde{K}^k$ y calculamos $\tilde{K}^k$ mediante la solución del programa semidefinido

$$\begin{array}{ll} \min\limits_{u} & FO1 = \frac{1}{2}u^T D\tilde{K}^k Du - e^T u + g(\tilde{K}^k) \\ s.a. & e^T Du = 0, \\ & 0 \leq u \leq \nu e. \end{array}$$

La selección de los vectores de soporte y construcción del clasificador en cada versión de esquema alternante presentada se realizó a través del programa *construccion_del_clasificador.m* el cual está incorporado en la parte final de cada uno de los algoritmos desarrollados (ver apéndices).

Ahora, una característica esencial del esquema alternante SVM-SDP, es la función g involucrada en la función objetivo de los programas. Como ya se mencionó la función g debe elegirse de tal manera que los datos aparezcan en ella de alguna manera en su definición; puesto que mediante ella tratamos de relacionar la matriz

kernel K con los datos de entrada almacenados en la matriz A.

En el presente trabajo proponemos alternativas para la escogencia de tal función, basadas en diferentes normas matriciales y vemos cual de ellas describe mejor, y es más eficiente para la impementación del esquema alternante SVM-SDP.

2.1. Posibilidades de normas matriciales

Consideremos el siguiente concepto,

Definición **8 (Norma matricial ([7]))** *Una norma matricial sobre el conjunto de todas las matrices de $n \times n$ es una función de valor real $\|\cdot\|$, definida en este conjunto y que satisface para todas las matrices B y C de $n \times n$ y todos los números reales α:*

- $\|B\| \geq 0$,
- $\|B\| = 0$, *si y sólo si B es O, la matriz con todas las entradas cero,*
- $\|\alpha B\| = |\alpha| \, \|B\|$,
- $\|B + C\| \leq \|B\| + \|C\|$,
- $\|BC\| \leq \|B\|\|C\|$.

La distancia entre las matrices B y C de $n \times n$ respecto a esta norma matricial es $\|B - C\|$.

Por lo general, las normas matriciales son extensiones naturales de las normas vectoriales. El estudio de las normas de matrices es importante por varias razones. Es necesario, por ejemplo, para definir de forma precisa conceptos tales como series de potencias de matrices; y desde luego es básico para precisar lo que se

entiende por proximidad o lejanía entre matrices, los cuales son aspectos fundamentales en el análisis de algoritmos en la computación numérica (ver [5]).

En este concepto de cercanía o proximidad existe especial interés, puesto que la idea es tomar una función con la matriz kernel K como variable en uno de los problemas de minimización del esquema alternante, de tal manera que a medida de que itere el algoritmo, la minimización de g esté de alguna manera preservando la esencia de los datos, en el sentido de que $K \approx AA^T$.

La anterior relación es importante, debido a que los datos en el espacio de entrada están almacenados en la matriz de datos A, y estos datos, es decir la información sobre muestra debe ser incluída en el esquema alternante.

Ahora, lo anterior se puede realizar considerando la función g definida por $g(K) = \|K - AA^T\|$, donde $\|\cdot\|$ es una norma matricial.

En este trabajo se utilizarán 4 alternativas para la norma matricial para la construcción de g. Ellas son:

Norma espectral

Sea $A \in \mathbb{R}^{n\times n}$ y denotemos los autovalores de la matriz $B = A^TA$ por λ_i, $i = 1, 2, \ldots, n$. Entonces la norma espectral de la matriz A viene dada por:

$$\|A\|_2 = \max_{1\leq i\leq n} \lambda_i^{\frac{1}{2}}, \tag{2.1.1}$$

esta es la norma inducida por la norma-ℓ_2 vectorial.

Norma infinito

Si $A = (a_{ij})$ es una matriz de $n \times n$ entonces, la norma infinito de A es dada por

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|, \tag{2.1.2}$$

esta es la norma inducida por la norma-ℓ_∞ vectorial.

Norma Frobenius

Es la norma proveniente del producto interno de matrices (1.2.1). No es una norma natural. También se define de la siguiente manera: dada A una matriz $n \times n$ su norma de Frobenius viene dada por:

$$\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{\frac{1}{2}}. \tag{2.1.3}$$

.

Norma 1

Si $A = (a_{ij})$ es una matriz de $n \times n$ entonces, la norma 1 de A es dada por

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|, \tag{2.1.4}$$

esta es la norma inducida por la norma-ℓ_1 vectorial.

Así, las funciones a ser objeto de nuestro estudio en la implementación del esquema alternante serán:

- $g_2(K) = \|K - AA^T\|_2$,
- $g_\infty(K) = \|K - AA^T\|_\infty$,
- $g_F(K) = \|K - AA^T\|_F$,
- $g_1(K) = \|K - AA^T\|_1$.

Capítulo 3

Pruebas numéricas

En este capítulo se exponen resultados de la implementación del algoritmo de esquema alternante SDP-SVM. Para ello fueron desarrollados los programas *algoritmo*1.*m*, *algoritmo*2.*m* los cuales corresponden a los enfoques primales del algoritmo, y los programas *algoritmo*3.*m* y *algoritmo*4.*m* que corresponden al enfoque dual del esquema alternante SVM-SDP visto en el capítulo anterior.

Para la construcción de los diversos programas se empleó la versión R2011a de la herramienta de software matemático MATLAB y el lenguaje de modelado YALMIP (ver [10]) y [11])), junto al resolvedor externo para YALMIP denominado SEDUMI (ver apéndices). A continuación mostraremos algunos resultados de la implementación del algoritmo esquema alternante SVM-SDP:

3.1. Implementación del esquema alternante

En esta sección mostramos distintas corridas del esquema alternante SVM-SDP, en algunos casos se implementa el enfoque primal y en otros el dual. También consideramos distintas posibilidades de la la función g, variaciones del parámetro de peso ν y de la tolerancia ϵ. También se presentan corridas en los que son mostrados cuadros comparativos en el que veremos parámetros tales como:

la cantidad de iteraciones realizadas por el algoritmo *iter*, cantidad de vectores de soporte seleccionados en la corrida VS y el tiempo de ejecución del algoritmo t en segundos.

La selección de los vectores de soporte es realizada mediante el programa *construccion_del_clasificador.m*, recordemos que los vectores de soporte juegan un papel fundamental para construir el clasificador. En este programa se incorpora un parámetro de tolerancia llamado *tol_vec_soport* (Ver Apéndice B) con el cual se puede controlar la cantidad de vectores de soporte a seleccionar.

Para los casos linealmente separables, se construyó el programa en MATLAB, *clasificador_lineal.m*, el cual se basa en los vectores de soporte para construir hiperplanos separadores.

La construcción del clasificador no lineal para los casos linealmente no separable, se basó en la interpolación de los vectores de soporte seleccionados por el esquema alternante; se utilizó el programa *clasificador_no_lineal.m*, el cual usa el comando de interpolación interp1. Cabe destacar que el comando interp1 permite seleccionar distintos métodos de interpolación; en nuestro caso combinamos este comando con el método nearest, el cual es una interpolación al vecino más cercano; y cubic que es basado en interpolación de Hermite seccionalmente cúbica (ver [3]).

En las diversas corridas que se presentarán a continuación, se muestra la data original y debajo de ella el resultado arrojado por el esquema alternante SVM-SDP. Se mostrará en color amarillo los vectores de soporte asociados con la data etiquetada con $+1$ (datos rojos) y de igual manera los vectores de soporte que corresponden a la data etiquetada con -1 (datos azules) se muestran en color verde; la superficie construida tanto por *clasificador_lineal.m* para muestras linealmente separables, como por *clasificador_no_lineal.m* para muestras linealmente no separables se presenta en color magenta.

Comenzamos estudiando casos en la que los datos son linealmente separable, y vemos el buen desempeño que tiene el esquema alternante SVM-SDP para este tipo de muestras.

3.1.1. Caso linealmente separable

Ejemplo 3.1.1 *La siguiente figura muestra el resultado arrojado por programa basado en el algoritmo 1. Luego de 2 iteraciones, usando parámetro de peso* $\nu = 10$*, función* g_2 *y valores iniciales* u^0*,* y^0 *vetores de orden* $m \times 1$*;* $\gamma^0 = 1$ *y una tolerancia* $\epsilon = 10^{-4}$*; el programa selecciona correctamente los vectores soporte correspondiente a la data etiquetada con* $+1$ *y de igual forma los vectores soporte que corresponden a la data etiquetada con* -1*.*

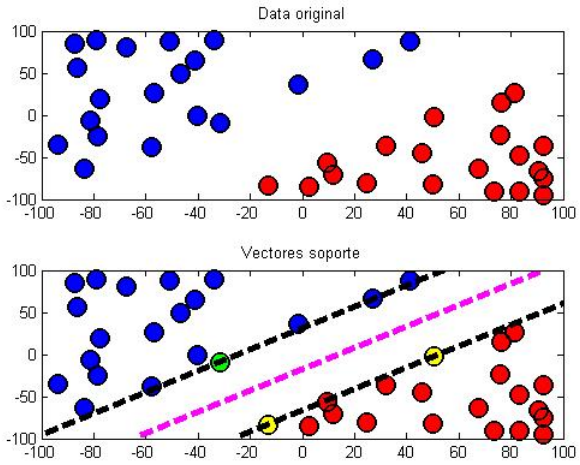


Figura 3.1: Datos 1

Ejemplo 3.1.2 *En esta corrida se muestra el resultado del programa basado en el algoritmo 2. Luego de 2 iteraciones, con parámetro de peso* $\nu = 10$*, función* g_2*, matriz inicial* $K^0 = I_{50}$*, y una tolerancia* $\epsilon = 10^{-4}$*, fueron seleccionados correctamente los vectores de soporte, y basados en ellos se construye un adecuado hiperplano separador.*

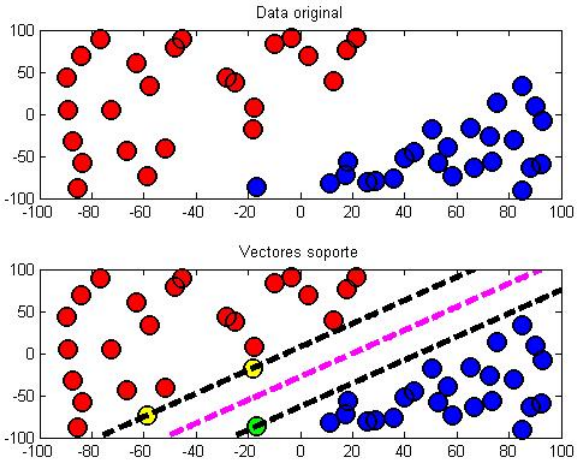


Figura 3.2: Datos 2

Ejemplo 3.1.3 *En esta corrida se utilizó un programa basado en el algoritmo 3. Luego de 2 iteraciones, con parámetro de peso* $\nu = 0.025$*, función* g_F*, matriz inicial* $K^0 = I_{61}$*, y una tolerancia* $\epsilon = 10^{-4}$*; se obtuvo un hiperplano separador correcto, que permite distinguir las clases* $+1$ *y* -1*, todo ello basado en la correcta selección de los vectores de soporte.*

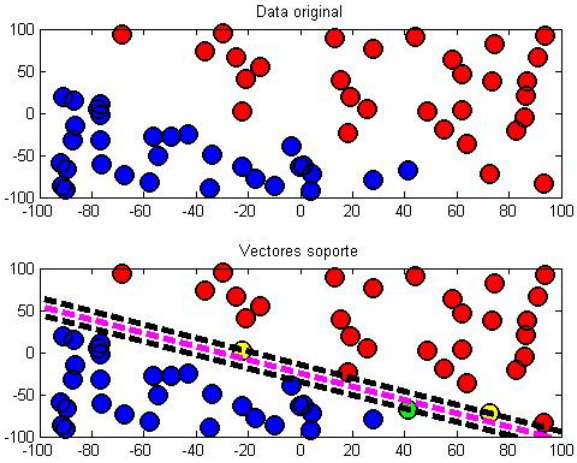


Figura 3.3: Datos 3

Ejemplo 3.1.4 *Para la siguiente muestra se utilizó, un enfoque basado en algoritmo 4. Luego de 3 iteraciones, con parámetro de peso* $\nu = 0.025$*, función* g_2 *y vector multiplicador de Lagrange inicial* u^0 *un vector de unos de orden* 40×1*,y una tolerancia de* $\epsilon = 10^{-4}$*, se puede apreciar que el programa selecciona correctamente los vectores soporte correspondiente a la data etiquetada con* $+1$ *y* -1*, también se consigue un hiperplano separador óptimo para los datos.*

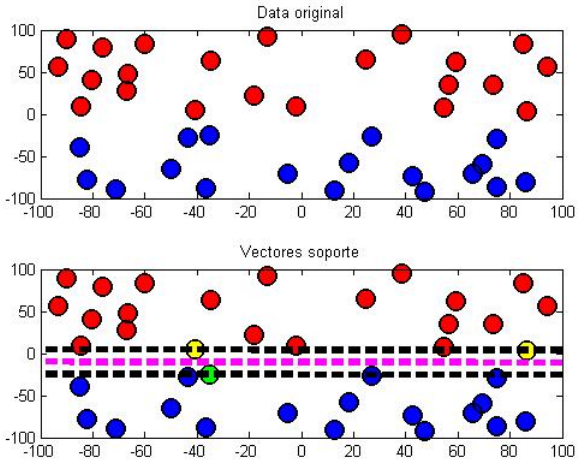


Figura 3.4: Datos 4

Ejemplo 3.1.5 *La corrida que se presenta fue realizada en el programa con enfoque del algoritmo 3, con parámetro* $\nu = 0.1$*, una tolerancia de* $\epsilon = 10^{-4}$ *y función* g_∞*. Luego de 3 iteraciones, el algoritmo selecciona adecuadamente los vectores de soporte, y mediante éstos construye un hiperplano separador para la muestra.*

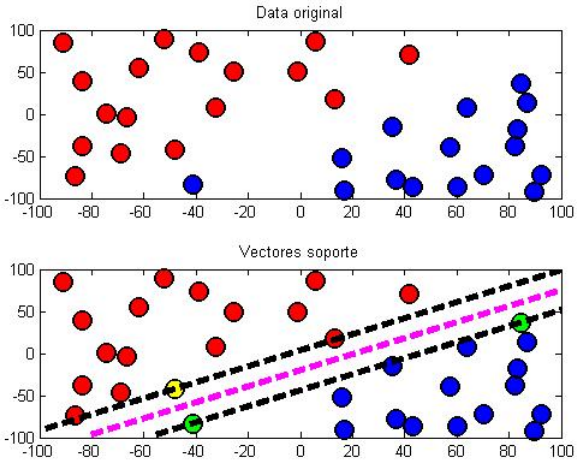


Figura 3.5: Datos 5

A continuación se presenta un cuadro en el que se presentan corridas con los 4 enfoques, combinadas con distintas elecciones de la función g y dos posibilidades del parámetro de peso v. Se puede observar que el algoritmo puede tener resultados muy buenos con distintas combinaciones de v con g. También vemos que en determinados casos cierta combinación del valor de v con alguna función g no da buenos resultados, pero el mismo parámetro con otra función g da resultados muy favorables, con un tiempo de ejecución pequeño.

Algoritmo	*punto inicial*	ν	t	*iter*	*VS*	g
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*10*	*5.186263*	*2*	*3*	g_2
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*0.1*	*4.802829*	*2*	*3*	g_2
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*10*	*4.24874*	*2*	*3*	g_F
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*0.1*	*4.218879*	*2*	*3*	g_F
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*10*	*4.300007*	*2*	*3*	g_∞
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*0.1*	*4.550517*	*2*	*3*	g_∞
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*10*	*4.559940*	*2*	*3*	g_1
Algoritmo 1	$(1_{m\times1}, 1_{m\times1}, 1)$	*0.1*	*4.725406*	*2*	*3*	g_1
Algoritmo 4	$1_{m\times1}$	*10*	*7.525395*	*3*	*3*	g_2
Algoritmo 4	$1_{m\times1}$	*0.1*	*7.723722*	*3*	*3*	g_2
Algoritmo 4	$1_{m\times1}$	*10*	*23.407098*	*8*	*todos*	g_F
Algoritmo 4	$1_{m\times1}$	*0.1*	*7.858499*	*3*	*3*	g_F
Algoritmo 4	$1_{m\times1}$	*10*	*7.778837*	*3*	*3*	g_∞
Algoritmo 4	$1_{m\times1}$	*0.1*	*7.792811*	*3*	*3*	g_∞
Algoritmo 4	$1_{m\times1}$	*10*	*10.933172*	*3*	*3*	g_1
Algoritmo 4	$1_{m\times1}$	*0.1*	*11.211618*	*3*	*3*	g_1
Algoritmo 3	I_{33}	*10*	*15.239083*	*3*	*3*	g_2
Algoritmo 3	I_{33}	*0.1*	*5.902278*	*2*	*3*	g_2
Algoritmo 3	I_{33}	*10*	*25.773941*	*8*	*todos*	g_F
Algoritmo 3	I_{33}	*0.1*	*5.305350*	*2*	*3*	g_F
Algoritmo 3	I_{33}	*10*	*29.887900*	*8*	*todos*	g_∞
Algoritmo 3	I_{33}	*0.1*	*4.947283*	*3*	*3*	g_∞
Algoritmo 3	I_{33}	*10*	*48.908801*	*8*	*todos*	g_1
Algoritmo 3	I_{33}	*0.1*	*6.599089*	*2*	*3*	g_1
Algoritmo 2	I_{33}	*10*	*7.910282*	*2*	*3*	g_2
Algoritmo 2	I_{33}	*0.1*	*19.657047*	*4*	*3*	g_2
Algoritmo 2	I_{33}	*10*	*9.669814*	*2*	*3*	g_1
Algoritmo 2	I_{33}	*0.1*	*51.210154*	*5*	*8*	g_1
Algoritmo 2	I_{33}	*10*	*9.993714*	*2*	*3*	g_F
Algoritmo 2	I_{33}	*0.1*	*13.093661*	*4*	*3*	g_F
Algoritmo 2	I_{33}	*0.1*	*24.407195*	*5*	*11*	g_∞
Algoritmo 2	I_{33}	*10*	*7.200936*	*2*	*3*	g_∞

Ejemplo 3.1.6 *En esta corrida se utilizó el algoritmo3.m, con parámetro de peso* $\nu = 0.025$*, con* g_F*. Luego de dos iteraciones, el algoritmo selecciona tres vectores de soporte, con los cuales es construido el hiperplano que permite clasificar la muestra etiquetada con* $+1$ *y* -1*, y separalas de manera adecuada.*

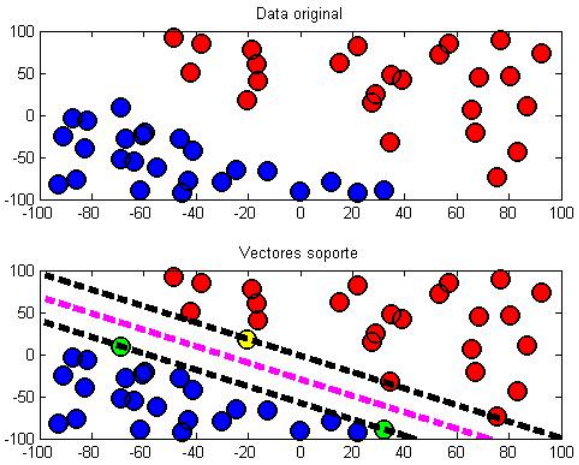


Figura 3.6: Datos 6

A continuación se muestra una tabla, en el que se exhiben datos de corridas sobre la muestra anterior, variando la matriz inicial K^0*, y distintas elecciones de la función g. Se puede observar que la matriz inicial no afecta significativamente el tiempo de ejecución del algoritmo. Pero si se puede observar que el tiempo de ejecución más pequeño se obtuvo al usar la función* g_∞ *independientemente de la matriz inicial empleada.*

g	K^0	ν	VS	iter	t
g_2	I_{50}	0.025	3	2	20.782719
g_∞	I_{50}	0.025	3	2	13.333038
g_F	I_{50}	0.025	3	2	18.03193
g_1	I_{50}	0.025	3	2	48.105354
g_2	Gaussiano	0.025	3	2	18.813354
g_∞	Gaussiano	0.025	3	2	13.734820
g_F	Gaussiano	0.025	3	2	19.879041
g_1	Gaussiano	0.025	3	2	48.192195
g_2	Polinomial	0.025	3	2	18.413558
g_∞	Polinomial	0.025	3	2	14.062766
g_F	Polinomial	0.025	3	2	23.735272
g_1	Polinomial	0.025	3	2	50.327015
g_2	O_{50}	0.025	3	2	20.087666
g_∞	O_{50}	0.025	3	2	13.889556
g_F	O_{50}	0.025	3	2	19.232550
g_1	O_{50}	0.025	3	2	49.783393

Tabla 3.1: Datos 6

3.1.2. Caso no separable linealmente

Si bien el esquema alternante SVM-SDP, tiene un buen desempeño para muestras linealmente separables, es de sumo interés ver su desempeño para muestras que no sean linealmente separables, y comprobar para este tipo de muestras que la matriz kernel que da como resultado el esquema alternante SVM-SDP, permite obtener un adecuado vector de multiplicadores de Lagrange, y por ende una buena selección de los vectores de soporte. Una vez obtenidos los vectores de soporte, se construye la superficie de clasificación y separación mediante la interpolación de estos.

A continuación se muestran una serie de corridas, en las que se implementa el esquema SVM-SDP, usamos en esta subsección el algoritmo que mejor resultados experimentales arrojó, el cual fue el *algoritmo*3.*m*.

Ejemplo 3.1.7 *En esta corrida se utilizó el algoritmo 3, con parámetro de peso*

$\nu = 0.025$, con distintas elecciones de la función g y matrices iniciales K^0. Se utilizó una tolerancia de 10^{-4} para el esquema alternante y de 3×10^{-5} para la selección de los vectores de soporte.

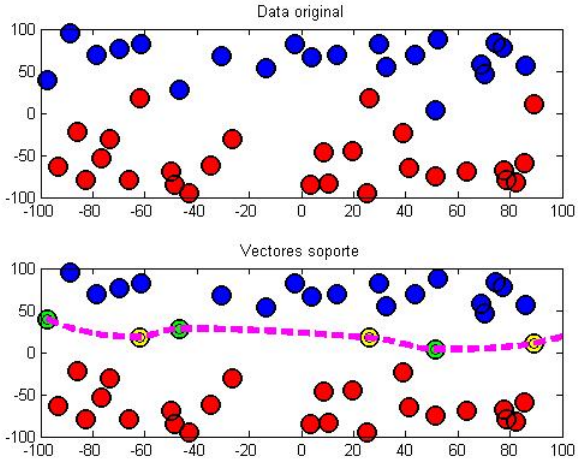


Figura 3.7: Datos 7

g	K^0	ν	VS	iter	t
g_2	I_{48}	0.025	6	2	12.351510
g_∞	I_{48}	0.025	6	2	7.612127
g_F	I_{48}	0.025	6	2	10.501611
g_1	I_{48}	0.025	6	2	32.107735
g_2	Gaussiano	0.025	6	2	12.159405
g_∞	Gaussiano	0.025	6	2	8.008951
g_F	Gaussiano	0.025	6	2	10.993410
g_1	Gaussiano	0.025	6	2	31.663940
g_2	Polinomial	0.025	6	2	12.557801
g_∞	Polinomial	0.025	6	2	8.603589
g_F	Polinomial	0.025	6	2	14.048727
g_1	Polinomial	0.025	6	2	35.352838

Tabla 3.2: Datos 7

Ejemplo 3.1.8 *La siguiente corrida fue realizada empleando el algoritmo 3, con*

la función g_F, parámetro de peso $\nu = 2.5$. Con tolerancia del esquema alternante de 10^{-4}. El algoritmo encuentra satisfactoriamente una superficie de separación, se muestra la data original, la superficie arrojada por el programa utilizando el método de interpolación nearest y también la superficie que arroja el programa cuando emplea el método de separación cubic.

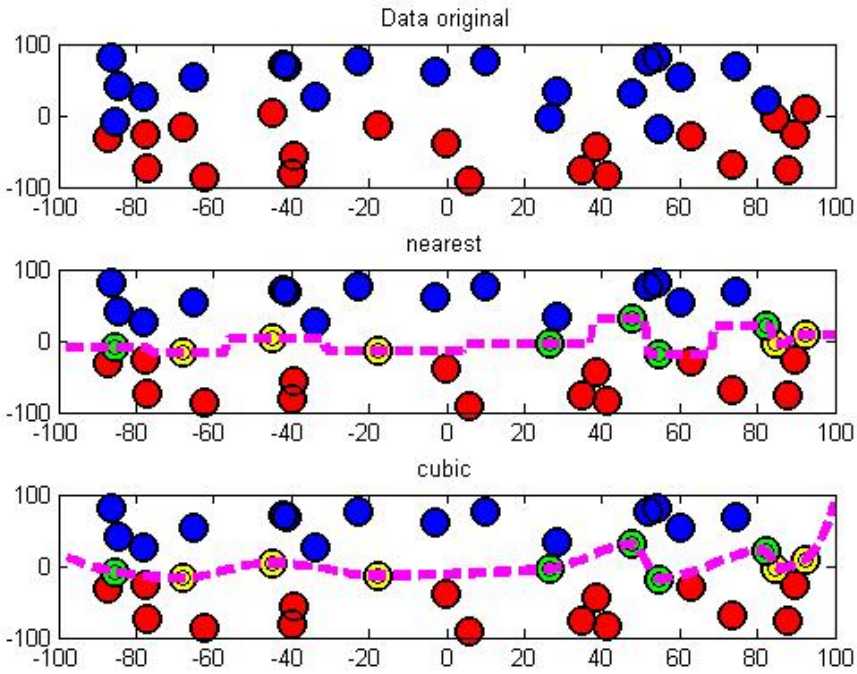


Figura 3.8: Datos 8

Ejemplo 3.1.9 *La siguiente corrida fue realizada con el algoritmo 3, usando la función g_∞, con parámetro de peso $\nu = 0.025$. Tolerancia para el esquema alternante de 10^{-4} y para la selección de los vectores de soporte una tolerancia de $3 \times 10^{-}5$.*

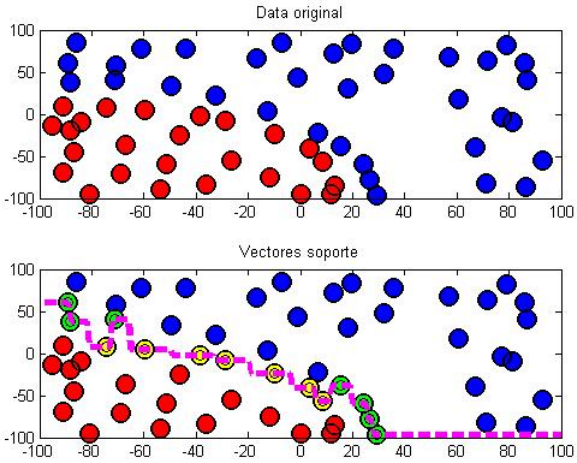


Figura 3.9: Datos 9

En la sigiguiente tabla se muestran algunos datos sobre corridas usando otras funciones g, si bien en todos los casos se logra la misma cantidad de vectores de soporte y la misma superficie de separación, el esquema alternante implementado con la función g_∞ dio el menor tiempo de ejecución.

g	K^0	ν	VS	iter	t
g_2	I_{60}	0.025	14	2	53.607858
g_∞	I_{60}	0.025	14	2	24.108756
g_F	I_{60}	0.025	14	2	45.356276
g_1	I_{60}	0.025	14	2	143.582804

Tabla 3.3: Datos 9

Ejemplo 3.1.10 *A continuación se muestra el resultado de una corrida del algoritmo 3. Se varía tanto el parámetro de peso como la función g.*

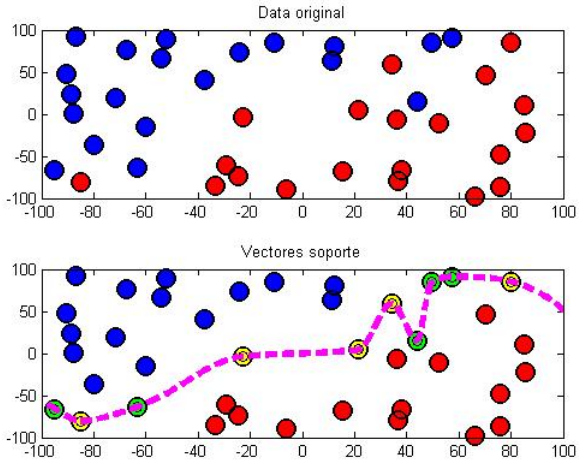


Figura 3.10: Datos 10

Si bien con todos los parámetros se obtuvo la misma cantidad de vectores de soporte y la misma superficie; se aprecia que el parámetro de peso ν influye tanto en la cantidad de iteraciones como en el tiempo de ejecución.

g	K^0	ν	VS	iter	t
g_2	I_{40}	0.4	10	3	10.365691
g_2	I_{40}	0.025	10	2	6.702604
g_2	I_{40}	0.25	10	3	10.596698
g_∞	I_{40}	0.4	10	3	11.180473
g_∞	I_{40}	0.025	10	2	5.422949
g_∞	I_{40}	0.25	10	2	7.612127
g_F	I_{40}	0.4	10	5	20.465253
g_F	I_{40}	0.025	10	2	5.903640
g_F	I_{40}	0.25	10	3	11.502374

Tabla 3.4: Datos 10

Ejemplo 3.1.11 *A la siguiente muestra se le aplicó el algoritmo 3, con tolerancia de* $\epsilon = 10^{-4}$, $\nu = 0.025$. *La superficie fue realizada mediante el algoritmo de interpolación nearest. También se muestra una tabla comparativa donde se emplean distintas elecciones de g, y de* K^0.

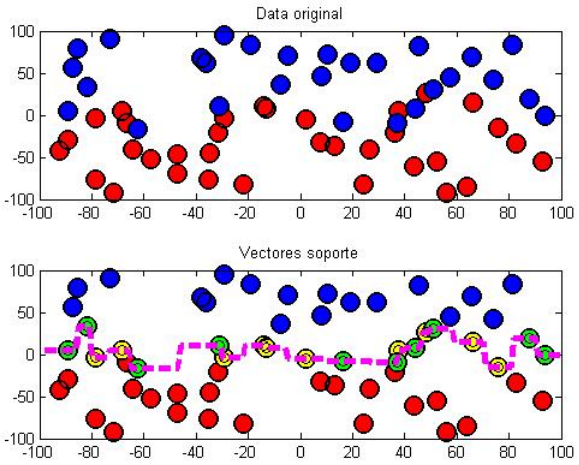


Figura 3.11: Datos 11

g	K^0	ν	VS	iter	t
g_2	I_{62}	0.025	20	2	69.578426
g_∞	I_{62}	0.025	20	2	25.489494
g_F	I_{62}	0.025	20	2	49.407682
g_1	I_{62}	0.025	20	2	168.040784
g_2	Gaussiano	0.025	20	2	62.065363
g_∞	Gaussiano	0.025	20	2	26.066550
g_F	Gaussiano	0.025	20	2	51.767382
g_1	Gaussiano	0.025	20	2	163.577240
g_2	Polinomial	0.025	20	2	72.003943
g_∞	Polinomial	0.025	20	2	26.007403
g_F	Polinomial	0.025	20	2	70.657126
g_1	Polinomial	0.025	20	2	163.769799

Tabla 3.5: Datos 11

De la tabla 3.5, concluimos que la función con mejor desempeño fue g_∞. La matriz K^0 inicial no influye de manera considerable en la ejecución del programa. Además, notamos que en todas las corridas se realizaron 2 iteraciones y se seleccionaron 20 vectores de soporte. Cabe destacar que en todas las corridas se obtuvo la misma superficie excepto con la función g^F partiendo de la matriz inicial polinomial.

Ejemplo 3.1.12 *La corrida que a continuación se presenta fue realizada con una tolerancia $\epsilon = 10^{-4}$ y un número máximo de iteraciones de 8. Se logra una adecuada superficie de clasificación, mediante la implementación de la función g_∞ con un parámetro de peso de $\nu = 0.025$.*

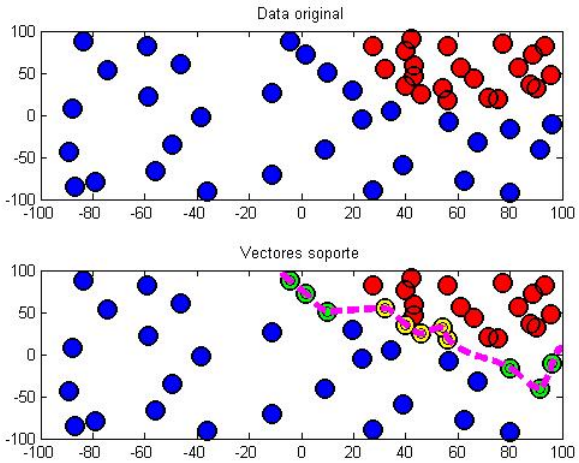


Figura 3.12: Datos 12

La siguiente tabla muestra distintas corridas del algoritmo con respecto a la muestra anterior, se estudian dos parámetros de pesos y tres elecciones distintas de la función g. Se observa que la combinación de la función g_∞ con el parámetro de peso $\nu = 0.025$ arrojo una buena clasificación y el tiempo de ejecución más pequeño, sin embargo cuando combinamos g_∞ con $\nu = 0.8$ no se obtuvo buen

resultado y el algoritmo se detine porque llega al número máximo de iteraciones que son 8, con este resultado podemos observar la influencia del parámetro ν en el programa.

g	K^0	ν	VS	iter	t
g_2	I_{54}	0.8	8	3	47.659440
g_2	I_{54}	0.025	11	2	25.404599
g_∞	I_{54}	0.8	43	8	234.817647
g_∞	I_{54}	0.025	11	2	15.948791
g_F	I_{54}	0.8	44	5	244.217342
g_F	I_{54}	0.025	11	2	25.754919

Tabla 3.6: Datos 12

Ejemplo 3.1.13 *La siguiente corrida fue realizada empleando el algoritmo 3, con la función g_∞, parámetro de peso $\nu = 0.025$. Con tolerancia para el esquema alternante SVM-SDP de 10^{-4}. Luego de 2 iteraciones, el programa encuentra satisfactoriamente una superficie de separación, se muestra la data original, la superficie arrojada por el programa utilizando el método de interpolación nearest. Se comparan también la selección de los vectores de soporte para tol_vs $= 0.00003$ y tol_vs $= 0.015$.*

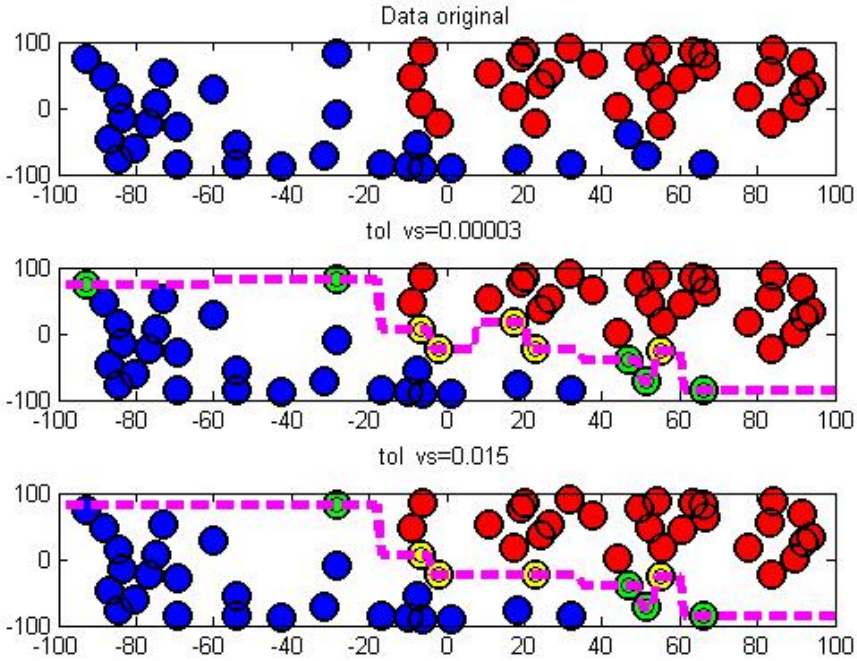


Figura 3.13: Datos 13

La siguiente tabla comparativa, exhibe información sobre las corridas:

g	K^0	ν	VS	iter	tol_vs	t
g_∞	I_{60}	0.025	10	2	0.00003	24.959234
g_∞	I_{60}	0.025	8	2	0.015	25.372437

Tabla 3.7: Datos 13

Como vemos en la tabla, con el parámetro $\nu = 0.015$ *se redujo en 2 la cantidad de vectores de soporte seleccionados en comparación a cuando se empleó* $\nu = 0.00003$*, esto se debe a que al aumentar tol_vs se están seleccionando una cantidad menor de multiplicadores de Lagrange que cumplan con ser mayor que la tolerancia requerida.*

Ejemplo 3.1.14 *En la siguiente figura se muestran dos corridas del algoritmo 3, en el que variamos el parámetro tol_vs. Fue empleada la función g_∞, junto con el parámetro de peso $\nu = 0.025$, además de una tolerancia $\epsilon = 10^{-4}$ para el esquema alternante.*

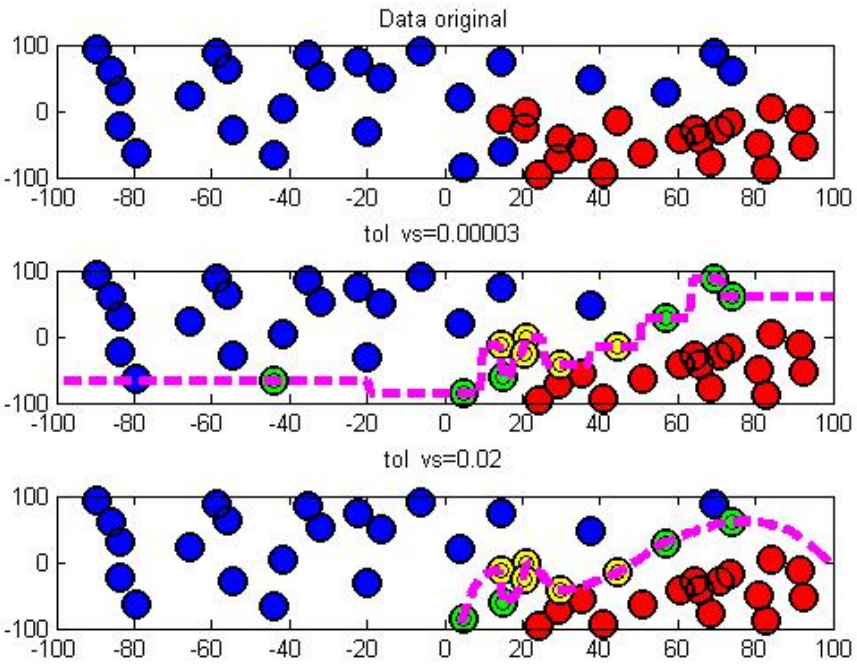


Figura 3.14: Datos 14

Información sobre las corridas está resumida en la siguiente tabla:

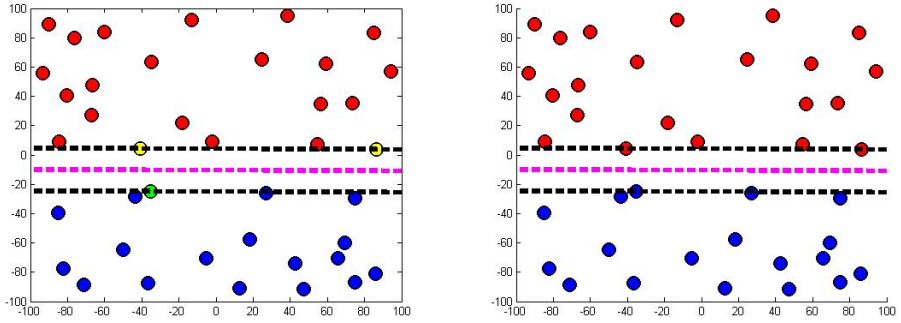
g	K^0	ν	VS	iter	tol_vs	t
g_∞	I_{46}	0.025	11	2	0.00003	7.469730
g_∞	I_{46}	0.025	9	2	0.02	7.350228

Tabla 3.8: Datos 14

3.2. Comparación con otros algoritmos

En esta sección se compara el desempeño del esquema alternante SVM-SDP con respecto a otros algoritmos, para ello se programó una svm estándar lineal y será implementada mediante el algoritmo desarrollado en MATLAB llamado *svm_lineal.m*, también se programó una svm no lineal, este algoritmo lo denominaremos *svm_no_lineal.m*. El algoritmo *svm_no_lineal.m* emplea funciones kernel comúnmente usadas para la construcción de una svm (ver Apéndices). Ahora mostraremos corridas donde se comparan los algoritmos:

Ejemplo 3.2.1 *A continuación se presentan dos corridas realizadas a una muestra linealmente separable, la primera es del esquema alernante basado en el algoritmo 3 con el uso de la función g_∞ y partiendo de la matriz identidad como matriz inicial; y la segunda corrida corresponde al programa svm_estandar.m.*



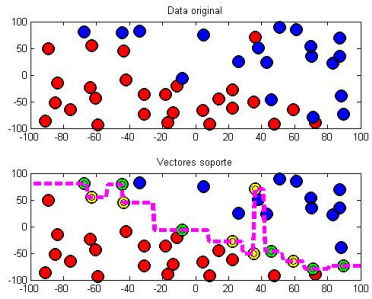
(a) Esquema alternante (b) SVM lineal

Figura 3.15: Comparación del esquema alternante con una svm lineal

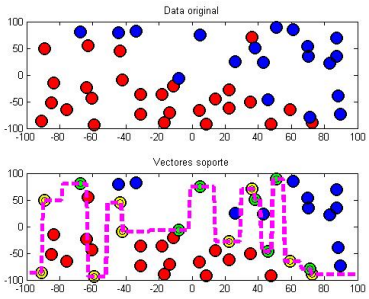
Como vemos en ambos casos se obtuvo la misma superficie de separación, lo cual valida nuestro algoritmo. Si bien svm_estandar.m es más rápido (1.315913 segundos) que algoritmo3.m (4.121549 segundos), nuestro algoritmo se basó en la selección de los vectores de soporte para la construcción del hiperplano, y lo realizó de manera correcta.

Además, recordemos que una svm lineal sólo arroja hiperplanos como superficies de separación en el espacio de entrada, sea la muestra linealmente separable o no, mientras que nuestro algoritmo es más versátil en ese sentido pues puede construir tanto superficies lineales (caso linealmente separable) como superficies no lineales (caso no separable linealmente) todo ello sólo basado en los vectores de soporte.

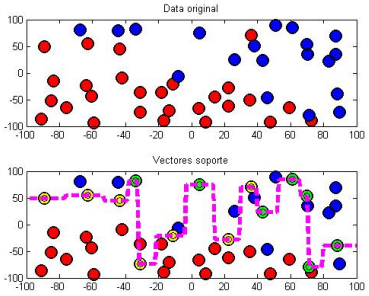
Ejemplo 3.2.2 *Ahora presentamos cuatro corridas a una muestra no separable linealmente, comparamos el desempeño de algoritmo3.m con función g_∞ y matriz inicial la identidad, con el programa de svm_no_lineal.m. En el programa svm_no_lineal.m fueron implementadas 3 funciones núcleo diferentes para de esta manera ver que vectores de soporte son seleccionados y que tan buena es la superficie que arroja nuestra algoritmo.*



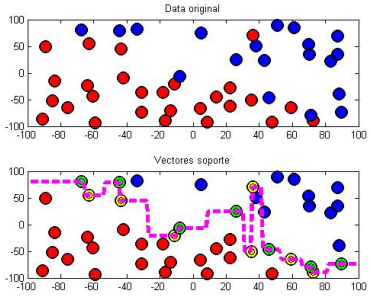
(a) Esquema alternante



(b) Gaussiano



(c) Sigmoide



(d) Polinomial no homogéneo

Figura 3.16: Comparación entre esquema alternante y funciones kernels

La siguiente tabla muestra información de la cantidad de vectores seleccionados y el tiempo de ejecución de los algoritmos en las 4 corridas.

Algoritmo	Función núcleo	VS	t
Algoritmo3	————	12	7.271798
svm_no_ineal	Gaussiano	16	2.603037
svm_no_ineal	Sigmoide	14	0.487182
svm_no_ineal	Polinomial	14	0.504794

Tabla 3.9: Datos 15

Se puede observar en tabla 3.9, que nuestro algoritmo posee mayor tiempo de ejecución en comparación con el programa svm_no_lineal.m, pero también es el que menor cantidad de vectores selecciona para la construcción del clasificador. Además, la superficie no lineal cosntruida por el esquema alternante SVM-SDP construye a una adecuada superficie de separación partiendo de la matriz identidad y no de una función núcleo en particular.

De este modo nuestro algoritmo, representa un paso hacia la automatización del proceso de construcción de una SVM, debido a que el esquema alternante SVM-SDP es capaz de producir superficies tanto lineales como no lineales sin la necesidad de estar probando funciones kernel, es capaz de partir de la matriz identidad e ir mejorando de manera iterada tanto la matriz kernel como el vector de multiplicadores de Lagrange en el proceso.

Además, nuestro algoritmo es capaz de construir tanto las superficies lineales como no lineales para una determinada muestra, basado únicamente en los vectores de soporte.

Conclusiones

A lo largo de esta investigación fueron desarrolladas las principales ideas de las máquinas de soporte vectorial y de la programación semidefinida, todo ello motivado a la creación e implementación exitosa de un algoritmo de esquema alternante que permitió el desarrollo de una SVM, que no dependiera de la función kernel a elegir para su construcción.

Si bien es un hecho que existen varios métodos desarrollados sobre clasificación binaria hasta la fecha, este trabajo es una paso hacia la automatización del proceso de construcción de una SVM, además con este trabajo se abren nuevos caminos como lo son involucrar la teoría de programación semidefinida en problemas ya existentes, así como también el uso del lenguaje de modelado YALMIP como herramienta de gran utilidad para el desarrollo de algoritmos de optimización.

Como investigación futura, se deben realizar estudios de convergencia del esquema alternante, así como también es de interés estudiar nuevas medidas de similaridad en el algoritmo.

Apéndices

YALMIP

YALMIP se utiliza como un complemento (toolbox) gratuito para MATLAB, y sirve para modelar problemas de optimización convexos (aunque también se utiliza para problemas no convexos). El lenguaje es consistente con la sintaxis de Matlab, por lo que es muy fácil de aprender para usuarios familiarizados con este entorno. Implementa una gran cantidad de recursos de modelado, permitiendo que el usuario se concentre en el modelo a alto nivel, mientras que YALMIP se ocupa del modelado a bajo nivel para obtener modelos eficientes y numéricamente satisfactorios.

Soporta varios tipos de problemas lineales, cuadráticos, cónicos de orden 2, semidefinidos, geométricos, y otros más como por ejemplo los problemas cónicos con mezcla de variables enteras. También se puede usar YALMIP para calcular soluciones explícitas mediante el interfaz "Multi-Parametric Toolbox" (MPT), este está orientado a controladores con restricciones lineales, no lineales y sistemas híbridos.

Una de las ideas centrales en YALMIP es confiar en solucionadores externos para la solución numérica de bajo nivel de problema de optimización. YALMIP concentra en algoritmos de modelado y de alto nivel eficientes. Una clasificación sencilla es la siguiente (las definiciones de libre y comercial depende un poco en el solucionador, vea los comentarios específicos en la descripción solucionador.

Mediante la explotación de la infraestructura de optimización en YALMIP, es bastante fácil de desarrollar algoritmos basados en los solucionadores externos. Esto ha motivado el desarrollo de solucionadores (internos) de enteros mixtos

cónicas (BNB, cutsdp), solucionadores globales para problemas no lineales generales (bmibnb), programación cuadrática indefinida (bmibnb, kktqp), suma de cuadrados módulos (solvesos), sólo por mencionar algunos de los la mayoría de las contribuciones importantes de YALMIP.

.1. Resolvedores externos de YALMIP para programación semidefinida

Entre los resolvedores externos libres de YALMIP se encuentran:

CSDP

CSDP se distribuye gratuitamente y está disponible en https://projects.coin-or.org/Csdp/. CSDP comunica desde MATLAB con un ejecutable a través de archivos de texto. Esto fácilmente puede causar grandes gastos para los grandes problemas densos. CSDP es invocado desde YALMIP con sdpsettings('solver','csdp').

DSDP

DSDP se distribuye gratuitamente en http://www-unix.mcs.anl.gov/DSDP/. La sentencia para invocarlo desde YALMIP es con sdpsettings('solver','dsdp').

PENLAB

PENLAB es un solucionador de SDP no lineal de código abierto disponible desde http://web.mat.bham.ac.uk/kocvara/penlab/. PENLAB se invoca con sdpsettings ('solver', 'penlab'). YALMIP actualmente sólo es compatible con el soporte de la desigualdad matricx bilineal de PENLAB, es decir, las no linealidades en general no son compatibles.

SDPA

SDPA se distribuye gratuitamente y está disponible en http://sdpa.sourceforge.net/. SDPA se invoca con sdpsettings ('solver', 'SDPA').

SDPT3

SDPT3 se distribuye gratuitamente y está disponible en http://www.math.nus.edu.sg/ mattohkc/sdpt3.html. Actualizaciones recientes (incluyendo correcciones para Octave) se pueden encontrar en https://github.com/sqlp/sdpt3. SDPT3 se invoca con sdpsettings ('solver', 'sdpt3').

LMIRANK

LMIRANK se distribuye gratuitamente en http://users.rsise.anu.edu.au/ robert / lmirank /. LMIRANK se invoca con sdpsettings ('solver', 'lmirank'). LMIRANK está especializado en problemas SDP con restricciones de rango, define utilizando el rango de comando. El solucionador sólo puede resolver los problemas de viabilidad, y se basa en un programa de solución SDP externa. El solucionador utiliza un enfoque local para hacer frente a las limitaciones de rango no convexas, por lo que no está garantizado para encontrar una solución.

KYPD

KYPD se invoca con sdpsettings ('solver', 'KYPD'). KYPD es un solucionador de problemas desarrollado para SDP a gran escala con las limitaciones KYP. Se basa en un solucionador de SDP externa y no está destinado a ser utilizado como un solucionador de propósito general. Está disponible en:
http://www.control.isy.liu.se/research/pub-db-supplemental/reports/2733/kypd.tar.gz.

CUTSDP

CUTSDP se envía con YALMIP. CUTSDP se invoca con sdpsettings ('solver', 'cutsdp'). CUTSDP es una implementación experimental interno con poco las pruebas realizadas, a pesar de que se ha aplicado con éxito en algunos de los problemas del mundo real. CUTSDP es un programa de solución simple para entera mixta cono de segundo orden y programas semidefinido, la implementación de un plano de corte (aproximación externa) enfoque.

SEDUMI

El sitio oficial de SeDuMi es http://sedumi.ie.lehigh.edu. SeDuMi se invoca con sdpsettings ('solver', 'sedumi'). SeDuMi es el solucionador más utilizado entre los usuarios YALMIP. Por favor, tenga en cuenta que hay problemas con las últimas versiones de MATLAB y 64 bits SeDuMi. Antes de utilizar SeDuMi con YALMIP, asegúrese de que la instalación SeDuMi realmente funciona, mediante la ejecución de algunos de los ejemplos incluidos en la distribución SeDuMi.

Códigos en MATLAB

Para estudiar el método de esquema alternante SVM-SDP se realizaron un conjunto de algoritmos desarrollados en MATLAB versión R2011a bajo el lenguaje de modelado YALMIP.

.2. genera_datos.m

Este algoritmo permite seleccionar puntos en el plano para generar muestras de prueba para el esquema alternante SVM-SDP, mediante el comando ginput, el cual permite crear matrices de dos clases de puntos A1 y A2. Los puntos son ingresados a gusto del usuario mediante el uso del cursor. Este programa también guarda las matrices generadas para de esta manera poder formar una batería de datos de prueba para los algoritmos.

```
1 format long
2 warning off %esta sentencia suprime los warning generados ...
     por el código
3 vv=[-100 100 -100 100];
4 axis(vv);
5 [x1 y1]=ginput();
6 [x2 y2]=ginput();
7 A1=[x1 y1];
8 A2=[x2 y2];
9 save A1
```

```
10 save A2
```

.3. construccion_del_clasificador.m

Este algoritmo selecciona los vectores de soportes de una máquina de vectores de soporte, basado en el vector de multiplicadores de Lagrange proveniente del esquema alternante SVM-SDP. Permite graficar los puntos en el plano, y colorea de color amarillo los vectores correspondientes a la clase del +1 y de verde a los correspondientes a la del −1. También identifica si el problema es linealmente separable o linealmente no separable. Si el problema es linealmente separable el programa llama al algoritmo *clasificador_lineal.m* y en caso contrario hace el llamado a *clasificador_no_lineal.m*.

```
1  %Seleccionador de los vectores de soporte
2  axis(vv);
3  subplot(2,1,2);
4  plot(A1(:,1),A1(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
      k','MarkerFaceColor','r','MarkerSize',13)
5  hold on
6  plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
      k','MarkerFaceColor','b','MarkerSize',13)
7  t=size(A);
8  sw=0;
9  CC=[0 0];
10 %tol_vec_soport=0.024;
11 tol_vec_soport=0.00003;
12 %tol_vec_soport=0.018;
13 for i=1:t;
14   if sol_u(i)> tol_vec_soport
15     if sw==0
16         AA=A(i,:);
17         BB=A(i,:);
```

```
18          ii=i;
19          sw=1;
20        else
21          AA=[AA;A(i,:)];
22          if i > nn
23              CC=[CC;A(i,:)];
24          end
25          ii=[ii,i];
26        end
27      %if sol_alfa(i)> 0.00005 %permite ajustar el epsilon
28      %if i < nn+1
29      i;
30      hold on
31      plot(A(i,1),A(i,2),'o','LineWidth',2,'MarkerEdgeColor',...
            'k','MarkerFaceColor','y','MarkerSize',13)
32      if i > nn
33      i;
34      A(i,:);
35      hold on
36      plot(A(i,1),A(i,2),'o','LineWidth',2,'MarkerEdgeColor',...
            'k','MarkerFaceColor','g','MarkerSize',13)
37      end
38    end
39  end
40  title('Vectores soporte');
41
42
43  %Resolución del primal para determinar si el problema es o...
        no separable
44  %linealmente
45  uu = sdpvar(m,1);
46  yy = sdpvar(m,1);
47  gamma = sdpvar(1,1);
```

```
48 %Definición de restricciones
49 J = set(D*(sol_K*D*uu-e*gamma)+yy>e);
50 J = J + set(yy > 0);
51 %el comando solvesdp es usado para todos los problemas de ...
      optimización
52 FO3 = v*e'*yy+(0.5)*uu'*H*uu;
53 valor_FO3=FO3;
54 solvesdp(J,FO3);
55 %La solución óptima puede ser extraída con el comando ...
      double(x)
56 uu1 = double(uu);
57 yy1 = double(yy);
58 gamma1 = double(gamma);
59
60
61 axis(vv);
62
63 if yy1 < 0.0000001
64
65   clasificador_lineal
66
67 else
68
69   clasificador_no_lineal
70
71 end
```

.4. clasificador_lineal.m

Este programa construye un hiperplano separador en el plano basado en los vectores de soporte seleccionados por el algoritmo *construccion_del_clasificador.m*, para un problema linealmente separable.

```
ccc=size(CC);
aaa=size(AA);
a=aaa(1);
c=ccc(1);
b=a-c+1;

if c >= 3 & b < 2

    m2=(CC(3,2)-CC(2,2))/(CC(3,1)-CC(2,1));
    x=[-100:0.1:100];
    yyy2=m2*(x-CC(2,1))+CC(2,2);
    yyy2l=m2*(x-AA(1,1))+AA(1,2);
    pm=[m2*(-CC(2,1))+CC(2,2)+m2*(-AA(1,1))+AA(1,2)]/2;
    yyy2c=m2*(x-0)+pm;
    axis(vv);
    plot(x,yyy2,'k--','LineWidth',4);
    plot(x,yyy2l,'k--','LineWidth',4);
    plot(x,yyy2c,'m--','LineWidth',4);

elseif b>=2 & c < 3
    m1=(AA(2,2)-AA(1,2))/(AA(2,1)-AA(1,1));
    x=[-100:0.1:100];
    yyy1=m1*(x-AA(1,1))+AA(1,2);
    yyy1l=m1*(x-CC(2,1))+CC(2,2);
    pm2=[m1*(-AA(1,1))+AA(1,2)+m1*(-CC(2,1))+CC(2,2)]/2;
    yyy1c=m1*(x-0)+pm2;
    axis(vv);
    plot(x,yyy1,'k--','LineWidth',4);
    plot(x,yyy1l,'k--','LineWidth',4);
    plot(x,yyy1c,'m--','LineWidth',4);

elseif b<2 & c <3
    m3=(CC(2,2)-AA(1,2))/(CC(2,1)-AA(1,1));
```

```
34        x=[-100:0.1:100];
35        pm31=[AA(1,1)+CC(2,1)]/2;
36        pm32=[AA(1,2)+CC(2,2)]/2;
37        yyy3=(-1/m3)*(x-AA(1,1))+AA(1,2);
38        yyy3l=(-1/m3)*(x-CC(2,1))+CC(2,2);
39        yyy3c=(-1/m3)*(x-pm31)+pm32;
40        plot(x,yyy3,'k--','LineWidth',4);
41        plot(x,yyy3l,'k--','LineWidth',4);
42        plot(x,yyy3c,'m--','LineWidth',4);
43
44    else
45
46      fprintf('El problema es separable linealmente pero se...
            han seleccionado más vectores de soporte de los ...
          necesarios, modifique el parámetro tol_vec_soport'...
          )
47
48    end
```

.5. clasificador_no_lineal.m

Este algoritmo construye una superficie de clasificación para un problema linealmente no separable, mediante la interpolación de los vectores de soporte arrojados por el esquema alternante SVM-SDP. Esto se logra mediante el uso del comando interp1 de MATLAB, el cual nos permite interpolar los vectores de soporte seleccionados por el algoritmo *construccion_del_clasificador.m* y graficar esa curva en el plano..

```
1   x = AA(:,1)'; y = AA(:,2)';
2   pp = interp1(x,y,'nearest','pp'); %nearest,pchip,cubic
3   xi = -100:1:100;
4   yi = ppval(pp,xi);
```

```
5   plot(x,y,'ko'), hold on, plot(xi,yi,'m--','LineWidth',4)...
      , hold off
```

.6. algoritmo3.m

Este código resuelve el problema

$$\begin{aligned} \min_{u,\tilde{K}} \quad & \tfrac{1}{2}u^T D\tilde{K}Du - e^T u + g(\tilde{K}) \\ s.a. \quad & e^T Du = 0, \\ & 0 \le u \le \nu e, \\ & \tilde{K} \succeq 0. \end{aligned} \tag{.6.1}$$

de manera integral, mediante el empleo del algoritmo 3 del esquema alternante SVM-SDP.

```
1  %ingreso de los datos
2  % warning off %esta sentencia suprime los warning ...
     generados por el código
3  vv=[-100 100 -100 100];
4  axis(vv);
5  %[x1 y1]=ginput();
6  %[x2 y2]=ginput();
7  %A1=[x1 y1];
8  %A2=[x2 y2];
9    load A140
10   load A240
11 A=[A1;A2];
12 subplot(2,1,1); plot(A1(:,1),A1(:,2),'o','LineWidth',2,'...
     MarkerEdgeColor','k','MarkerFaceColor','r','MarkerSize'...
     ,13)
13 hold on
14 plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
     k','MarkerFaceColor','b','MarkerSize',13)
```

```
15 title('Data original')
16 nn=size(A1);
17 n=nn(1);
18 nn1=size(A2);
19 n1=nn1(1);
20 m=n+n1;
21 d=[ones(n,1);-ones(n1,1)];
22 H=eye(m);
23 D=diag(d);
24 e=ones(m,1);
25 K0=eye(m);
26 O=zeros(m,1);
27 v=0.025;
28 epsilon=0.0001;
29 diff=2;
30 Nmax=0;
31
32 while (abs(diff) > epsilon) && (Nmax < 8)
33
34 %Inicio del primer problema (1)
35
36 %definición de variables de decisión
37 u = sdpvar(m,1);
38
39 %Definición de restricciones
40 F = set(e'*D*u == 0);
41 F = F + set(O < u);
42 F = F + set(u < v*e);
43
44 %el comando solvesdp es usado para todos los problemas de ...
     optimización
45 FO1 = (0.5)*u'*D*K0*D*u-e'*u+norm(K0-A*A',Inf);
46 solvesdp(F,FO1);
```

```
47 valor_FO1=FO1;
48
49 %La solución óptima puede ser extraída con el comando ...
      double(x)
50 u1 = double(u);
51
52 %Inicio del problema (2)
53
54 %Definición de variables de decisión
55 K = sdpvar(m,m);
56
57 %Definición de restricciones
58 G = set(K>0);
59
60 %el comando solvesdp es usado para todos los problemas de ...
      optimización
61 FO2 = (0.5)*(u1)'*D*K*D*(u1)-e'*(u1)+norm(K-A*A',Inf);
62 %ops = sdpsettings('solver','sdpa');
63 ops = sdpsettings('solver','sedumi');
64 solvesdp(G,FO2,ops);
65 %solvesdp(G,FO2);
66 valor_FO2=FO2;
67
68 %La solución óptima puede ser extraída con el comando ...
      double(x)
69 K0=double(K);
70
71 diff=abs(valor_FO1 - valor_FO2);
72 diff=double(diff);
73 Nmax=Nmax+1
74
75 end
76
```

```
77 sol_u = double(u);
78 sol_K = double(K);
79
80 construccion_del_clasificador
```

.7. svm_lineal.m

Este algoritmo consigue la solución de una SVM lineal (1.1.6). El algoritmo realiza una separación óptima de una muestra en el plano consiguiendo el hiperplano que las separa, hace uso del toolbox de optimización de MATLAB llamado fmincon, por ende todas los cálculos realizados dentro del algoritmo son hechos de manera tal, que que el programa que describe la svm estándar quede en la forma:

$$\begin{array}{lll} \text{mín} & f(x) \\ s.a. & Ax \leq b \\ g(x) & \leq & 0 \ (restriccines \ no \ lineales) \end{array}$$

```
1  hold on
2  format long
3  vv=[-100 100 -100 100];
4  tic;
5  load A14
6  load A24
7  A=[A1;A2];
8  plot(A1(:,1),A1(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
      k','MarkerFaceColor','r','MarkerSize',13)
9  hold on
10 plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
      k','MarkerFaceColor','b','MarkerSize',13)
11 %title('Data original')
```

```
12 nn=size(A1);
13 n=nn(1);
14 nn1=size(A2);
15 n1=nn1(1);
16 m=n+n1;
17
18 d=[ones(n,1);-ones(n1,1)];
19 D=diag(d);
20 m=n+n1;
21 DA=D*A;
22 DE=D*(ones(m,1));
23 I=-eye(m,m);
24 O1=zeros(m,2);
25 O2=zeros(m,1);
26 E=-ones(m,1);
27 O2=zeros(m,1);
28 B1=[-DA DE I];
29 C1=[O1 O2 I];
30 AA=[B1;C1]
31 C=[E;O2]
32 z=3+m;
33 w=zeros(1,z);
34 [x,fval,flag,output]=fmincon('fsvmestandar1',w,AA,C...
      ,[],[],[],[],[])
35 ww=[x(1) x(2)];
36 gg=x(3);
37 yy=x(4:z);
38 hold on
39 t=[-100:0.1:100];
40 xx1=(1+gg-ww(1)*t)/ww(2);
41 xx2=(-1+gg-ww(1)*t)/ww(2);
42 xx=(gg-ww(1)*t)/ww(2);
43 plot(t,xx1,'k--',t,xx2,'k--',t,xx,'m--','LineWidth',4)
```

.8. svm_no_lineal.m

Este algoritmo resuelve un problema de la forma (1.1.14) mediante el uso de funciones kernels conocidas.

```
vv=[-100 100 -100 100];
axis(vv);
% [x1 y1]=ginput();
% [x2 y2]=ginput();
% A1=[x1 y1];
% A2=[x2 y2];
format long
vv=[-100 100 -100 100];
% tic;
load A140
load A240

A=[A1;A2];
subplot(2,1,1); plot(A1(:,1),A1(:,2),'o','LineWidth',2,'...
   MarkerEdgeColor','k','MarkerFaceColor','r','MarkerSize'...
   ,13)
hold on
plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
   k','MarkerFaceColor','b','MarkerSize',13)
title('Data original')
nn=size(A1);
n=nn(1);
nn1=size(A2);
n1=nn1(1);
m=n+n1;
d=[ones(n,1);-ones(n1,1)];
H=eye(m);
D=diag(d);
```

```
26 e=ones(m,1);
27 %K0=(A*A'+e*e'); %se está inicializando K con el kernel ...
      polinomial
28 % K0=zeros(m,m);
29 % for i=1:m
30 % for j=1:m
31 % nor=norm(A(i,:)-A(j,:)); %empleo del núcleo
32 % K0(i,j)=exp((-(nor)^2)/2); %Gaussiano
33 % end
34 % end
35 % K0;
36  K0=zeros(m,m);
37    for i=1:m
38     for j=1:m
39      nor=norm(A(i,:)-A(j,:)); %empleo del núcleo
40      K0(i,j)=tanh(10*((nor)^2)+(-10)); %Hiperbólico
41     end
42    end
43  K0;
44
45 O=zeros(m,1);
46 v=10;
47
48 %definición de variables de decisión
49 u = sdpvar(m,1);
50 %Definición de restricciones
51 F = set(e'*D*u == 0);
52 F = F + set(O < u);
53 F = F + set(u < v*e);
54 %el comando solvesdp es usado para todos los problemas de ...
      optimización
55 FO1 = (0.5)*u'*D*K0*D*u-e'*u;
56 solvesdp(F,FO1);
```

```
valor_FO1=FO1;
%La solución óptima puede ser extraída con el comando ...
   double(x)
u1 = double(u);
```

.9. algoritmo4.m

Este código resuelve el problema

$$\begin{aligned} \min_{u,\tilde{K}} \quad & \tfrac{1}{2} u^T D\tilde{K}Du - e^T u + g(\tilde{K}) \\ s.a. \quad & e^T Du = 0, \\ & 0 \le u \le \nu e, \\ & \tilde{K} \succeq 0. \end{aligned} \tag{.9.1}$$

de manera integral, mediante el empleo del algoritmo 4 del esquema alternante SVM-SDP.

```
%ingreso de los datos
warning off %esta sentencia suprime los warning generados ...
   por el código
vv=[-100 100 -100 100];
axis(vv);
%[x1 y1]=ginput();
%[x2 y2]=ginput();
%A1=[x1 y1];
%A2=[x2 y2];
format long
%vv=[-100 100 -100 100];

load A150
load A250

A=[A1;A2];
```

```
16 subplot(2,1,1); plot(A1(:,1),A1(:,2),'o','LineWidth',2,'...
       MarkerEdgeColor','k','MarkerFaceColor','r','MarkerSize'...
       ,13)
17 hold on
18 plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
       k','MarkerFaceColor','b','MarkerSize',13)
19 title('Data original')
20 nn=size(A1);
21 n=nn(1);
22 nn1=size(A2);
23 n1=nn1(1);
24 m=n+n1;
25 d=[ones(n,1);-ones(n1,1)];
26 H=eye(m);
27 D=diag(d);
28 e=ones(m,1);
29 u1=ones(1,m)';
30 %u1=zeros(1,m)';
31 O=zeros(m,1);
32 %v=0.025;
33 v=0.1;
34 epsilon=0.0001;
35 diff=2;
36 Nmax=0;
37
38 while (abs(diff) > epsilon) && (Nmax < 8)
39
40 %Inicio del problema (1)
41
42 %Definición de variables de decisión
43 K = sdpvar(m,m);
44
45 %Definición de restricciones
```

```
G = set(K>0);

%el comando solvesdp es usado para todos los problemas de ...
   optimización
FO2 = (0.5)*(u1)'*D*K*D*(u1)-e'*(u1)+norm(K-A*A',1);
%ops = sdpsettings('solver','sdpa');
ops = sdpsettings('solver','sedumi');
solvesdp(G,FO2,ops);
%solvesdp(G,FO2);
valor_FO2=FO2;

%La solución óptima puede ser extraída con el comando ...
   double(x)
K0=double(K);

%Inicio del primer problema (2)

%definición de variables de decisión
u = sdpvar(m,1);

%Definición de restricciones
F = set(e'*D*u == 0);
F = F + set(0 < u);
F = F + set(u < v*e);

%el comando solvesdp es usado para todos los problemas de ...
   optimización
FO1 = (0.5)*u'*D*K0*D*u-e'*u+norm(K0-A*A',1)%-trace(K0'*(D...
   *D'));
solvesdp(F,FO1);
valor_FO1=FO1;
%La solución óptima puede ser extraída con el comando ...
   double(x)
```

```
74 u1 = double(u);
75
76 diff=abs(valor_FO1 - valor_FO2);
77 diff=double(diff);
78 Nmax=Nmax+1
79
80 end
81
82 sol_u = double(u);
83 sol_K = double(K);
84
85 construccion_del_clasificador
```

.10. fsvmestandar1.m

La función $fsvmestandar1.m$ representa la función objetivo de un programa de svm lineal estándar. En esta función $x = (\omega, \gamma, y)$; es decir ω, γ, y van implícitamente dentro del vector fila x. En este caso el valor de $\nu = 10000$.

```
1 function f = fsvmestandar1(x)
2 m=length(x);
3 t=0;
4 for i=4:m
5    t=t+x(i);
6 end
7 f=((1/2)*(x(1)^2+x(2)^2))+10000*t;
```

.11. algoritmo1.m

Este código resuelve el problema

$$\begin{aligned} \min_{u,\gamma,y,\tilde{K}} \quad & \nu e^T y + \tfrac{1}{2} u^T H u + g(\tilde{K}) \\ s.a. \quad & D(\tilde{K} D u - e\gamma) + y \geq e \\ & y \geq 0, \\ & \tilde{K} \geq 0, \end{aligned} \tag{.11.1}$$

de manera integral, mediante el empleo del algoritmo 1 del esquema alternante SVM-SDP.

```
1  warning off %esta sentencia suprime los warning generados ...
       por el código
2  vv=[-100 100 -100 100];
3  axis(vv);
4  % [x1 y1]=ginput();
5  % [x2 y2]=ginput();
6  % A1=[x1 y1];
7  % A2=[x2 y2];
8  load A135
9  load A235
10 A=[A1;A2];
11 subplot(2,1,1); plot(A1(:,1),A1(:,2),'o','LineWidth',2,'...
       MarkerEdgeColor','k','MarkerFaceColor','r','MarkerSize'...
       ,13)
12 hold on
13 plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
       k','MarkerFaceColor','b','MarkerSize',13)
14 title('Data original')
15 A=[A1;A2];
16 nn=size(A1);
17 n=nn(1);
18 nn1=size(A2);
```

```
19 n1=nn1(1);
20 m=n+n1;
21 d=[ones(n,1);-ones(n1,1)];
22 D=diag(d);
23 e=ones(m,1);
24
25 %Vector inicial
26 u1=ones(1,m)';
27 y1=ones(1,m)';
28 gamma1=1;
29
30 O=zeros(m,1);
31 H=eye(m);
32 I=eye(m);
33 v=0.025;
34 epsilon=0.0001;
35 diff_vo=5;
36 Nmax=0;
37 while (abs(diff_vo) > epsilon) && (Nmax < 5)
38 %Inicio del problema (2)
39 %Definición de variables de decisión
40 K = sdpvar(m,m);
41 %Definición de restricciones
42 G = set(D*(K*D*u1-e*gamma1)+y1-e > 0);
43 G = G + set(K>0);
44 %el comando solvesdp es usado para todos los problemas de ...
     optimización
45 FO2 = v*e'*y1+(0.5)*u1'*H*u1+norm(K-A*A',inf);
46 valor_FO2=FO2;
47 %solvesdp(G,FO2);
48 %ops = sdpsettings('solver','sedumi');
49 %ops = sdpsettings('solver','bnb');
50 %ops = sdpsettings('solver','cutsdp');
```

```
ops = sdpsettings('solver','sedumi');
solvesdp(G,FO2,ops);
%La solución óptima puede ser extraída con el comando ...
   double(x)
K0=double(K);
%Inicio del primer problema (1)
%definición de variables de decisión
u = sdpvar(m,1);
y = sdpvar(m,1);
gamma = sdpvar(1,1);
%Definición de restricciones
F = set(D*(K0*D*u-e*gamma)+y>e);
F = F + set(y > 0);
%el comando solvesdp es usado para todos los problemas de ...
   optimización
FO1 = v*e'*y+(0.5)*u'*H*u+norm(K0-A*A',inf);
valor_FO1=FO1;
solvesdp(F,FO1);
%La solución óptima puede ser extraída con el comando ...
   double(x)
u1 = double(u);
y1 = double(y);
gamma1 = double(gamma);
diff=abs(valor_FO1 - valor_FO2);
diff_vo=double(diff);
Nmax=Nmax+1
end
sol_u = double(u1);
sol_gamma=double(gamma1);
sol_y=double(y1);
sol_K = double(K);

```

```
80  %Resolución del dualprimal para determinar los ...
       multiplicadores de Lagrange
81  %definición de variables de decisión
82  u = sdpvar(m,1);
83  %Definición de restricciones
84  F = set(e'*D*u == 0);
85  F = F + set(0 < u);
86  F = F + set(u < v*e);
87  %el comando solvesdp es usado para todos los problemas de ...
       optimización
88  FO1 = (0.5)*u'*D*sol_K*D*u-e'*u
89  solvesdp(F,FO1);
90  valor_FO1=FO1;
91  %La solución óptima puede ser extraída con el comando ...
       double(x)
92  u1 = double(u);
93
94
95  %construcción del clasificador
96  %Seleccionador de los vectores de soporte
97  axis(vv);
98  subplot(2,1,2);
99  plot(A1(:,1),A1(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
       k','MarkerFaceColor','r','MarkerSize',13)
100 hold on
101 plot(A2(:,1),A2(:,2),'o','LineWidth',2,'MarkerEdgeColor','...
       k','MarkerFaceColor','b','MarkerSize',13)
102 t=size(A);
103 sw=0;
104 for i=1:t;
105   if u1(i)> 0.000001
106     if sw==0
107         AA=A(i,:);
```

```
108         ii=i;
109         sw=1;
110       else
111         AA=[AA;A(i,:)];
112         ii=[ii,i];
113       end
114   %if sol_alfa(i)> 0.00005 %permite ajustar el epsilon
115     %if i < nn+1
116     i;
117     A(i,:);
118     hold on
119     plot(A(i,1),A(i,2),'o','LineWidth',2,'MarkerEdgeColor',...
          'k','MarkerFaceColor','y','MarkerSize',13)
120     if i > nn
121     i;
122     A(i,:);
123     hold on
124     plot(A(i,1),A(i,2),'o','LineWidth',2,'MarkerEdgeColor',...
          'k','MarkerFaceColor','g','MarkerSize',13)
125     end
126   end
127 end
128 title('Vectores soporte');
129
130
131 axis(vv);
132 if sol_y < 0.0000001
133     svm_estandar_escalado_mo;
134 else
135    x = AA(:,1)'; y = AA(:,2)';
136    pp = interp1(x,y,'cubic','pp');%linear,spline,nearest,...
          pchip,cubic,v5cubic
137    xi = -100:1:100;
```

```
138   yi = ppval(pp,xi);
139   plot(x,y,'ko'), hold on, plot(xi,yi,'m--','LineWidth',4)...
          , hold off
140 end
```

REFERENCIAS

[1] D. Bertsekas, A. Nedić, and A. E. Ozdagar. *Convex Analysis and Optimization*. Athenas Scientific, Belmont, USA, 2003.

[2] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[3] P. J. Davis. *Interpolation & Approximation*. Dover Publications INC, New York, 1975.

[4] G. Fung, M. Dundar, J. Bi, and B. Rao. A fast iterative algorithm for fisher discriminant using heterogeneous kernels. Technical report, Computer Aided Diagnosis & Therapy Solutions, Siemens Medical Solutions, 51 Valley Stream Parkway, Malvern,.

[5] W. Gautschi. *Numerical Analysis: An introduction*. Birkhäuser, Boston, 1997.

[6] C. Helmberg. Semidefinite programming. *Konrad-Zuse-Zentrum für Informationstechnik*, October 1999.

[7] R. Horn and C. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, 1990.

[8] G. Lanckriet, N. Cristiani, P. Bartlett, L. El Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. *J. of Machine Learning Research 5*, pages 27–72, 2004.

[9] M. Laurent and F. Vallentin. *Semidefinite Optimization*, March 2012.

[10] J. Lofberg. Yalmip : A toolbox for modeling and optimization in matlab. *Automatic Control Laboratory, ETHZ.*

[11] J. Lofberg. Modeling and solving uncertain optimization problems in yalmip. *IFAC*, 2008.

[12] O. L. Mangasarian. Generalized support vector machines. In B. Schölkopf A. Smola, P. Bartlett and Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146. MIT Press, Cambridge, MA, 2000.

[13] R. T. Rockafellar. *Convex analysis*. Princeton University Press, Princeton, NJ, 1970.

[14] B. Schölkopf and A. Smola. *Learning with Kernels-Support Vector Machines, Regularization, Optimization and Beyond*. 2002.

[15] V. N. Vapnik. *The nature of Stadistical Learning Theory*. Springer, New York, 2^{nd} edition, 2000.

[16] S. Wright and J. Nocedal. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, 2^{nd} edition, 2006.