INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

ESTUDIO NUMÉRICO DE ALGORITMOS A GRAN ESCALA PARA MÁQUINAS DE SOPORTE VECTORIAL

TESIS

QUE PARA OBTENER EL GRADO DE

LICENCIADO EN MATEMÁTICAS APLICADAS

PRESENTA

PABLO DE JESÚS CAMPOS VIANA

ASESOR: DR. JOSÉ LUIS MORALES PÉREZ

MÉXICO, D.F. 2016

Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "ESTUDIO NUMÉRICO DE ALGORITMOS A GRAN ESCALA PARA MÁQUINAS DE SOPORTE VECTORIAL", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Bailléres Jr., autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación.

Pablo de Jesús Campos Viana

_____

Fecha

_____

Firma

A mis padres.

Agradecimientos

Agradezco a mis padres, Pablo y Clara, y a mi hermano Felipe, por el apoyo incondicional y por creer en mí.

A Andry Laura, por cada momento juntos y por ser mi otra parte.

A mi asesor José Luis Morales, por la revisión de mi trabajo y por las charlas sobre los métodos de optimización de gran escala.

A mis sinodales Zeferino Parada, Léon Berdichevsky y Luis Díaz, por sus comentarios para hacer de éste un mejor trabajo.

A mis profesores en el ITAM. Especialmente a Felipe González por su excepcional clase de aprendizaje estadístico y Adolfo de Unánue por su retadora clase de arquitectura de datos.

A mis compañeros y amigos en BBVA, por hacer del trabajo un lugar ameno e intelectualmente motivante.

Finalmente, agradezco al Dr. Jacek Gondzio y al Dr. Julian Hall por darme la oportunidad de seguir mis estudios en la disciplina que me apasiona.

Prefacio

El método de **Máquinas de Soporte Vectorial** (**MSV**) es un algoritmo de aprendizaje estadístico ampliamente utilizado para problemas de clasificación binaria. Por razones de eficiencia computacional, la gran mayoría de los métodos utilizados en aprendizaje estadístico se implementan con técnicas que divergen de las utilizadas en los métodos clásicos de la optimización numérica.

En años recientes se ha observado un crecimiento constante en la capacidad de almacenamiento de los datos, por lo que se requiere el desarrollo de algoritmos de aprendizaje que sean escalables. En el caso del método MSV, existen varias implementaciones con el objetivo de operar a gran escala para clasificación lineal. La mayoría de estas implementaciones se clasifican en dos categorías, de acuerdo a su forma de resolver el problema de optimización subyacente:

- Variantes de *programación cuadrática* en la formulación dual.
- Variantes del método *descenso del gradiente estocástico* en la formulación primal.

Por otra parte, el lenguaje de programación **Python** se ha convertido en años recientes en uno de los lenguajes predilectos de la Ciencia de Datos para procesos de desarrollo y de producción, por su amplia flexibilidad y por su integración con herramientas de Big Data. El propósito de este trabajo es comparar el desempeño de dos implementaciones que pertenecen a cada una de las categorías mencionadas sobre conjuntos grandes de datos y que poseen interfaces en **Python**:

- **LIBSVM**
- **SVM-SGD**

La estructura de la tesis es como se describe a continuación. El **capítulo 1** presenta brevemente conceptos importantes de la teoría del aprendizaje estadístico y su relación con la optimización. El **capítulo 2** presenta los aspectos teóricos del método **MSV** así como los elementos que componen su formulación actual. El **capítulo 3** describe las decisiones algorítmicas y detalles de implementación de la biblioteca **LIBSVM**, mientras que el **capítulo 4** describe los detalles de implementación del método **SVM-SGD**. El **capítulo 5** presenta y discute los experimentos numéricos para comparar ambos métodos y finalmente el **capítulo 6** presenta las conclusiones del trabajo.

Índice general

Capítulo 1

Introducción

1.1. Aprendizaje estadístico

Considérese un contexto de **aprendizaje supervisado**. Cada observación $\mathbf{z}$ representa una pareja $(\mathbf{x}, y)$ compuesta de un vector de características $\mathbf{x} \in \mathcal{X}$ y un escalar $y \in \mathcal{Y}$, en donde $\mathcal{X}$ es el espacio de características y $\mathcal{Y}$ el espacio de posibles respuestas.

La teoría del aprendizaje estadístico toma como perspectiva asumir que existe una distribución de probabilidad desconocida sobre el espacio $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, es decir, que existe una distribución de probabilidad $P(\mathbf{z}) = P(\mathbf{x}, y)$. El conjunto de entrenamiento $\mathcal{S}$ se compone de n observaciones de esta distribución:

$$\mathcal{S} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \tag{1.1}$$

El problema de aprendizaje consiste en encontrar una función $f \in \mathcal{F}$, $f : \mathcal{X} \to \mathcal{Y}$ tal que $f(\mathbf{x}) \approx y$. El espacio de funciones $\mathcal{F}$ es llamado el espacio de hipótesis. Por otra parte, considérese el *funcional de pérdida* $\ell(f(\mathbf{x}), y)$ que define una métrica para el costo de predecir $f(\mathbf{x})$ cuando la respuesta verdadera es y.

Con la finalidad de encontrar la mejor función de aproximación, se define el *riesgo esperado*:

$$E(f) = \int_{\mathcal{Z}} \ell(f(\mathbf{x}), y) dP(\mathbf{z}) \tag{1.2}$$

y dado que la distribución de probabilidad $P(\mathbf{z})$ es desconocida, debe

usarse una medida de aproximación al riesgo esperado. Esta medida está basada en el conjunto de entrenamiento $\mathcal{S}$, y se define como el *riesgo empírico*:

$$E_{\mathcal{S}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) \tag{1.3}$$

El riesgo empírico $E_{\mathcal{S}}(f)$ mide el desemepeño en el conjunto de entrenamiento, mientras que el riesgo esperado $E(f)$ mide el desempeño de generalización, es decir, el desempeño esperado sobre futuras observaciones.

Teóricamente, la mejor función de aproximación para el caso del riesgo esperado está dada por:

$$f^* = \inf_{f \in \mathcal{F}} \quad E(f) \tag{1.4}$$

mientras que para el caso del riesgo empírico:

$$f_S^* = \inf_{f \in \mathcal{F}} \quad E_{\mathcal{S}}(f) \tag{1.5}$$

El enfoque de elegir una función $f_{\mathcal{S}}^*$ que minimice el riesgo empírico es llamado *minimización del riesgo empírico*.

1.2. Regularización

Supóngase que la familia $\mathcal{F}$ de funciones está parametrizada por un vector de parámetros $\mathbf{w}$. El término *regularización* se refiere a modificar el problema de minimizar el riesgo empírico $E_{\mathcal{S}}(f)$ de modo que el nuevo problema sea minimizar la función:

$$Q(\mathbf{w}) = E_{\mathcal{S}}(f_{\mathbf{w}}) + \lambda R(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i; \mathbf{w}) + \lambda R(\mathbf{w}) \tag{1.6}$$

en donde λ es un valor fijo tal que $\lambda \geq 0$ y en donde $R(\mathbf{w})$ es una función que penaliza al vector de parámetros $\mathbf{w}$. Este enfoque es conocido como *minimización del riesgo empírico regularizado*. Elecciones típicas del término de regularización son las normas ℓ_1 y ℓ_2 del vector $\mathbf{w}$; y en donde esta última está íntimamente relacionada con el método **MSV**. En la práctica, la regularización se utiliza para evitar el *sobreajuste*.

1.3. El error de generalización

Históricamente, la literatura de la optimización numérica se ha concentrado en la *desempeño asintótico*; es decir, qué tan rápido la precisión de la solución aumenta con el tiempo de cómputo.

En el caso de los algoritmos de aprendizaje, el desempeño en la generalización está limitado por tres fuentes de error:

- *El error de aproximación*: mide qué tan bien puede ser aproximada la solución exacta por una función obtenida por el algoritmo de aprendizaje.
- *El error de estimación*: mide la precisión con la que se puede determinar la mejor función obtenida por el algoritmo de aprendizaje utilizando un conjunto finito de datos de entrenamiento.
- *El error de optimización*: mide la precisión con la que se puede calcular la función que mejor explote la información contenida en el conjunto de datos de entrenamiento.

El error de estimación está determinado por el tamaño del conjunto de datos de entrenamiento y por la capacidad de la familia de funciones elegida para el algoritmo de aprendizaje (Vapnik, 1982). Por ejemplo, familias más grandes de funciones suelen tener errores de aproximación más pequeños pero errores de estimación más grandes. Este compromiso ha sido estudiado ampliamente en la literatura (Steinwart and Scovel, 2004). Por otra parte, la literatura en la optimización numérica se ha concentrado en los algoritmos cuyo error decrece más rápido con el número de iteraciones. Usualmente, el tiempo de ejecución para cada iteración crece linealmente o cuadráticamente con el número de observaciones.

En el caso de los problemas de aprendizaje, errores de optimización decrecientes son irrelevantes en comparación con las otras fuentes de error. Por lo tanto, en la práctica usualmente es deseable utilizar algoritmos de optimización que sacrifiquen precisión asintótica por menor complejidad en cada iteración. Las implementaciones del método MSV han evolucionado históricamente hacia este objetivo.

1.4. Aprendizaje de pequeña y gran escala

En los problemas de aprendizaje existen restricciones dados por el tamaño del conjunto de datos o por el tiempo de cómputo. Estos dos factores permiten distinguir entre los problemas de aprendizaje de pequeña escala y de gran escala.

- *Problemas de pequeña escala*: Aquellos restringidos por el tamaño del conjunto de datos. El error de generalización suele estar dominado por los errores de estimación y de aproximación. El error de optimización puede ser reducido sustancialmente dado que no existe gran restricción sobre el tiempo de cómputo.

- *Problemas de gran escala*: Aquellos restringidos por el tiempo de cómputo. Además de elegir la complejidad de la familia de funciones, el tamaño del conjunto de datos de entrenamiento puede ser ajustado dados los recursos computacionales disponibles. Los algoritmos de optimización pueden lograr una mejor capacidad de generalización respecto de los problemas de pequeña escala ya que mayor cantidad de datos pueden ser procesados.

En la práctica, suelen preferirse utilizar algoritmos que reduzcan el error de optimización lo suficiente, por debajo de los errores esperados de aproximación y de estimación. Este ha sido el principal motor de la evolución de las implementaciones del método MSV.

Capítulo 2

Máquinas de soporte vectorial

El método MSV es ampliamente utilizado para resolver problemas de clasificación y de regresión. En su formulación más tradicional para problemas de clasificación, dado un conjunto de observaciones en donde cada una pertenece a una de dos clases, el método de MSV construye un modelo que asigna nuevas observaciones a alguna de las dos clases. Además de utilizarse para clasificación lineal, el método puede ser utilizado para realizar clasificación no lineal mediante la **técnica del Kernel**.

2.1. Transformación explícita de las observaciones

La complejidad del conjunto de datos de entrenamiento afecta directamente el desempeño de cualquier método de aprendizaje que pueda ser utilizado. En ocasiones, el uso de cierta clase de métodos no es apropiado para construir una función de predicción para un conjunto de datos dado. Tal puede ser el caso de métodos lineales cuando las variables a utilizar en la modelización de cierto fenómeno no guardan una relación lineal con el mismo. Por esta razón, realizar una transformación de las variables puede resultar de gran ayuda para la construcción de una función de predicción.

La razón más importante para transformar los datos es que el espacio al que se realiza la transformación puede estar dotado de cierta estructura que pueda ser explotado por el método de aprendizaje a utilizar; en particular dicha estructura puede ser lineal.

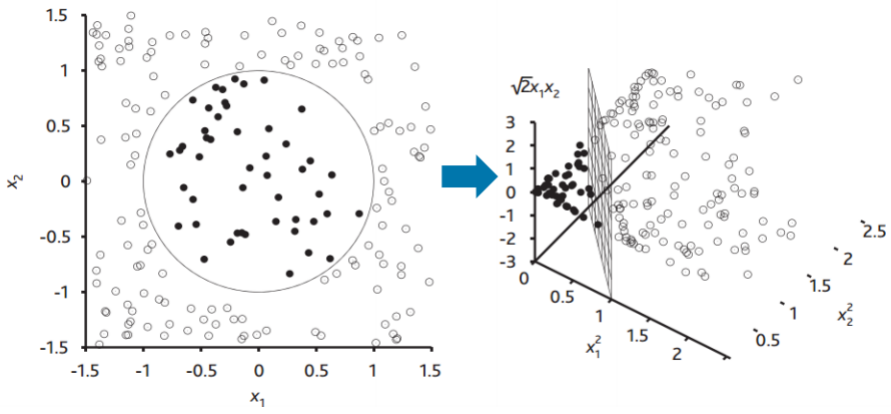

Figura 2.1: Mapeo explícito de las observaciones. En la figura de la izquierda, la función de decisión corresponde en $\mathbb{R}^2$ a $\mathbf{x}_1^2 + \mathbf{x}_2^2 = 1$. En la figura derecha, los mismos datos son transformados al espacio $\mathbb{R}^3$ mediante una transformación cuadrática: $\Phi(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^2, \mathbf{x}_2^2, \sqrt{2}\mathbf{x}_1\mathbf{x}_2)$.

De esta forma, el conjunto de datos de entrenamiento está dado por

$$\mathcal{S} = \{\mathbf{x}_i, y_i\}_{i=1}^n, \quad \mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y} \tag{2.1}$$

y su representación después de aplicar alguna función de transformación $\Phi : \mathcal{X} \to \mathcal{H}$ es

$$\mathcal{S} = \{\Phi(\mathbf{x}_i), y_i\}_{i=1}^n, \quad \Phi(\mathbf{x}_i) \in \mathcal{H}, y_i \in \mathcal{Y}. \tag{2.2}$$

en donde $\mathcal{X}$ es el espacio de las características que toman las observaciones, $\mathcal{Y} = \{-1, 1\}$ es el conjunto de ambas clases y $\mathcal{H}$ es el espacio al que son transformadas las observaciones.

2.2. Transformación implícita a través de Kernels

Realizar la transformación de cada $\mathbf{x}_i$ hacia $\Phi(\mathbf{x}_i)$ tiene la desventaja de ser costoso computacionalmente dado que la transformación Φ puede ser realizada hacia espacios de dimensión muy alta.

En lugar de realizar dicha transformación de forma explícita, el método MSV (y más generalmente, cualquier método basado en Kernels) utiliza alguna función

$$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \tag{2.3}$$

que realiza una evaluación entre cualquier pareja de observaciones $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}$. De esta forma, la función Kernel (o simplemente el *Kernel*) K se define sobre el espacio $\mathcal{X} \times \mathcal{X}$. Cuando se considera una muestra de observaciones $\mathcal{S}$, el Kernel finito puede ser representado como una matriz cuadrada de dimensión $n \times n$ en donde $K_{\mathcal{S}}(\mathbf{x}_i, \mathbf{x}_j) = k_{i,j}$.

Bajo este enfoque y asumiendo que las observaciones se encuentran definidas en un espacio $\mathcal{X}$ con producto interior, se puede construir una función Kernel K tomando el producto interior de modo que

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathcal{X}} \tag{2.4}$$

o bien, en espacios reales (i.e. $\mathcal{X} = \mathbb{R}^n$):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\intercal \mathbf{x}_j. \tag{2.5}$$

Debe considerarse que geométricamente, el producto punto entre dos vectores $\mathbf{x}_i$ y $\mathbf{x}_j$ se relaciona con el ángulo que existe entre ellos, dado que

$$\frac{\mathbf{x}_i^\intercal \mathbf{x}_j}{||\mathbf{x}_i|| ||\mathbf{x}_j||} = \cos\theta \tag{2.6}$$

de modo que puede interpretarse al Kernel como una medida de similitud entre sus argumentos.

Por otra parte, puede probarse que bajo condiciones muy generales sobre los Kernels, este enfoque y el enfoque basado en la transformación explícita de las observaciones a partir de una función $\Phi : \mathcal{X} \to \mathcal{H}$ son equivalentes. Es decir, que

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \tag{2.7}$$

en donde la correspondencia entre la función Kernel K y el espacio $\mathcal{H}$ es única, en virtud del Teorema de Moore-Aronszajn (1950), y la existencia de una función Φ para cierta función Kernel K está garantizada en virtud del Teorema de Mercer (1909). Una descripción muy completa del método MSV desde el punto de vista del análisis funcional puede ser hallada en [9].

2.2.1. Funciones Kernel

Más concretamente, una función Kernel K es una función $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ que es continua, simétrica y positiva definida. La noción de función Kernel generaliza la noción de matriz simétrica positiva definida; en particular para una realización finita de los datos (i.e. una muestra) $\mathcal{S}$, se tiene que la matriz Kernel $K_{\mathcal{S}} = [k_{i,j}]$ ha de ser simétrica y positiva definida ya que $k_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$.

2.2.2. Ejemplos de Kernels

De acuerdo a lo discutido con anterioridad, basta con proponer un Kernel (i.e. una función que satisfaga la condición de continuidad, simetría y positividad) de modo que se sabe que la ecuación 2.7 se satisface para alguna función Φ, para resolver el problema de clasificación.

Algunos de los Kernels más utilizados en aplicaciones son los siguientes:

- Polinomial homogéneo: $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\intercal \mathbf{x}_j)^d$ para $d \in \mathbb{N}$.
- Polinomial no homogéneo: $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\intercal \mathbf{x}_j + c)^d$, para $c \in \mathbb{R}$ y $d \in \mathbb{N}$.
- Tangente hiperbólica: $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i^\intercal \mathbf{x}_j + c)$, para $\kappa > 0$ y $c < 0$.
- Gaussiano de base radial (RBF): $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, para $\gamma > 0$.

Este último Kernel forma parte de una amplia familia de funciones llamadas *funciones de base radial*, cuyos valores dependen de una distancia entre dos puntos, i.e. $\phi(x, c) = \phi(\|x - c\|)$, y es uno de los más utilizados en aplicaciones (Bishop, 2010).

Además, tiene la ventaja de tener una interpretación inmediata: si dos observaciones $\mathbf{x}_i$ y $\mathbf{x}_j$ son muy parecidas, entonces $\|\mathbf{x}_i - \mathbf{x}_j\| \to 0$ y por lo tanto $\exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) = k(\mathbf{x}_i, \mathbf{x}_j) \to 1$; mientras que si son muy diferentes, entonces $\|\mathbf{x}_i - \mathbf{x}_j\| \to \infty$ y por lo tanto $\exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) = k(\mathbf{x}_i, \mathbf{x}_j) \to 0$. Es decir, valores del Kernel cercanos a uno corresponden a observaciones muy similares, mientras que valores cercanos a cero corresponden a observaciones muy diferentes.

2.3. Formulación del método

Históricamente, los primeros algoritmos para reconocimiento de patrones fueron clasificadores lineales. Dadas dos clases $y \in \{-1, 1\}$, una nueva observación $\boldsymbol{x}$ puede ser clasificada de acuerdo a una función lineal $f(\boldsymbol{x}) = \mathbf{w}^\intercal \boldsymbol{x} + b$ de modo que se tome el signo de esta función para realizar la clasificación. Además, se puede realizar una transformación Φ a las nuevas observaciones de modo que la función discriminante sea $f(\boldsymbol{x}) = \mathbf{w}^\intercal \Phi(\boldsymbol{x}) + b$, y en donde la transformación Φ puede ser realizada de forma explícita, tal como se menciona en la sección 2.1. El hiperplano obtenido mediante $f(\boldsymbol{x}) = 0$ define una frontera de decisión en el espacio transformado y los parámetros $\mathbf{w}$ y b son determinados a partir de la muestra disponible de los datos.

La formulación actual del método MSV se basa en estas ideas, junto con otros tres ingredientes escenciales:

- Hiperplanos óptimos.
- Margen suave.
- La técnica del Kernel.

2.3.1. Hiperplanos óptimos

El conjunto de datos es *linealmente separable* si existe una función lineal cuyo signo coincide con la clase a la que pertenecen los datos. Si el conjunto de datos es linealmente separable, entonces existe una infinidad de planos separadores.

Para elegir alguno de estos posibles hiperplanos, se puede utilizar la noción de *margen*. Intuitivamente, el margen representa la distancia del hiperplano hacia el punto más cercano de cada una de las dos clases. Por lo tanto, una elección razonable del hiperplano es aquel cuyo margen es máximo. Esta idea es ilustrada en la gráfica 2.2. El problema de encontrar la función $f(\mathbf{x}) = \mathbf{w}^\intercal \Phi(\mathbf{x}) + b$ que maximice el margen, y en donde $\mathbf{w}$ representa el vector normal al hiperplano, puede ser formulado de la siguiente forma.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \mathcal{P}(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{sujeto a:} \quad & y_i \left(\mathbf{w}^\intercal \Phi(\mathbf{x}_i) + b\right) \geq 1, \quad \forall i \end{aligned} \tag{2.8}$$

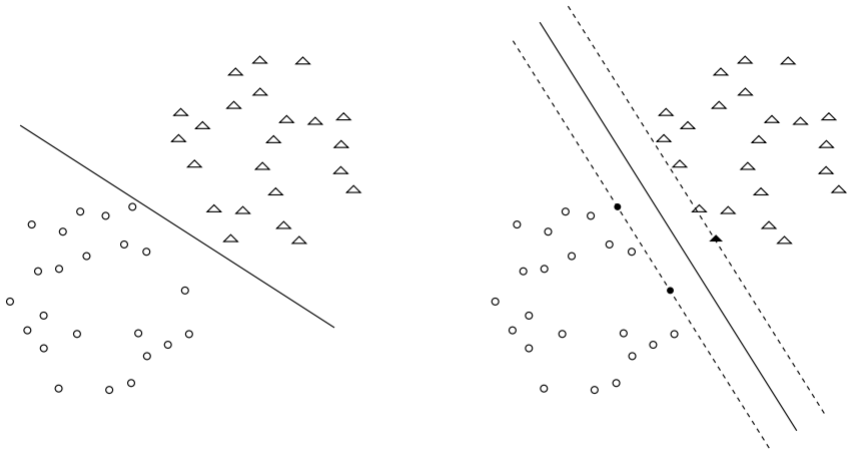

Figura 2.2: Un conjunto de datos con dos planos de diferente margen. En (a) se muestra un plano separador arbitrario y en (b) el plano es el de margen máximo, mientras que los puntos sombreados son aquellos que definen la función de separación. Este plano separador de margen máximo se define mediante la ecuación $f(\mathbf{x}) = \mathbf{w}^\intercal \Phi(\mathbf{x}) + b = 0$ a través del vector $\mathbf{w}$ que resuelve el problema 2.8.

Esta formulación es conocida como la **formulación primal** del problema. Por otra parte, utilizando Teoría de Dualidad, es posible establecer el siguiente problema equivalente, el cual es conocido como la **formulación dual** del problema.

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \mathcal{D}(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^\intercal \Phi(\mathbf{x}_j) \\ & \text{sujeto a:} \quad \alpha_i \geq 0 \quad \forall i \\ & \qquad\quad \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \forall i \end{aligned} \tag{2.9}$$

Esta formulación es más sencilla de resolver computacionalmente, ya que las restricciones son más simples. El vector de dirección $\mathbf{w}^*$ del hiperplano óptimo es recuperado a través del vector de solución $\boldsymbol{\alpha}^*$ del problema de

optimización de la formulación dual 2.9:

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \Phi(\mathbf{x}_i) \tag{2.10}$$

por lo que la función de decisión óptima puede expresarse como

$$f(\boldsymbol{x}) = \mathbf{w}^{*\intercal}\phi(\boldsymbol{x}) + b^* = \sum_{i=1}^{n} \alpha_i^* y_i \Phi(\mathbf{x}_i)^\intercal \Phi(\boldsymbol{x}) + b^* \tag{2.11}$$

2.3.2. Margen suave

Cuando el conjunto de datos no es linealmente separable, no se puede hablar de la noción de hiperplano óptimo. Sin embargo, este problema puede ser resuelto al permitir que algunos de los datos violen las restricciones del margen que aparecen en la formulación 2.8. Estas violaciones a las restricciones pueden ser representadas mediante variables de holgura $\xi_i, \forall i = 1, \ldots, n$, y adicionalmente un parámetro C controla el compromiso que existe entre el tamaño del margen y el número de violaciones a las restricciones del margen. Esta idea es ilustrada en la gráfica 2.3. El problema análogo a 2.8 que relaja las restricciones al margen puede ser formulado de la siguiente forma.

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \mathcal{P}(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{sujeto a:} \quad & y_i\left(\mathbf{w}^\intercal \Phi(\mathbf{x}_i) + b\right) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned} \tag{2.12}$$

De forma similar, la formulación dual del problema análoga a 2.9, está dada por

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^\intercal \Phi(\mathbf{x}_j) \\ & \text{sujeto a:} \quad 0 \leq \alpha_i \leq C \quad \forall i \\ & \qquad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \forall i \end{aligned} \tag{2.13}$$

en donde ahora aparece el parámetro C como una cota superior para los parámetros α_i. Este problema de optimización tiene como función objetivo

una función cuadrática con restricciones lineales, de modo que representa un problema de programación cuadrática.

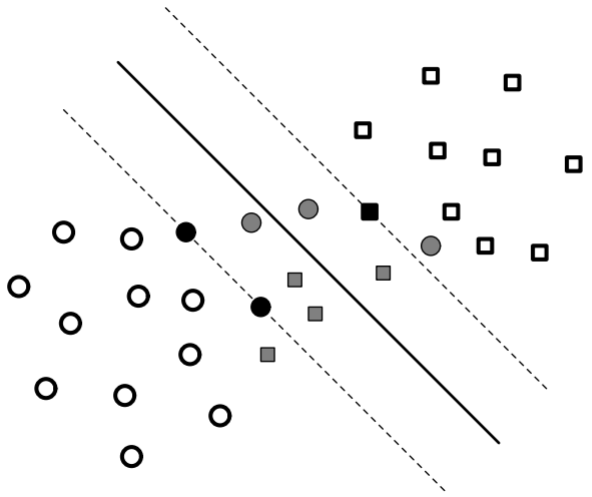

Figura 2.3: Un conjunto de datos que no es linealmente separable. El problema puede ser resuelto al permitir que algunos de los datos violen las restricciones del margen.

2.3.3. La técnica del Kernel

La **técnica del Kernel** simplifica el problema de optimización del método MSV. La técnica consiste en reemplazar el producto interno de los vectores de observaciones en el espacio transformado por evaluaciones mediante un Kernel de los vectores de observaciones en el espacio original. La función objetivo de la formulación dual del problema de optimización presentado anteriormente está dado, para datos sin transformar, por

$$\mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\intercal \mathbf{x}_j \tag{2.14}$$

de modo que se puede sustituir el producto entre vectores $\mathbf{x}_i^\intercal \mathbf{x}_j = \langle \mathbf{x}_i, \mathbf{x}_j \rangle_{\mathcal{X}}$ por el producto interno $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$ para cierta función $\Phi : \mathcal{X} \to \mathcal{H}$, es

decir

$$\mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \tag{2.15}$$

de forma que el problema sea resuelto en el espacio transformado; es decir, la separación lineal es llevada a cabo a través de las variables transformadas mediante la función Φ. Sin embargo, de acuerdo a lo discutido sobre la transformación implícita a través de Kernels, se puede utilizar una función Kernel K que satisface, como ya se ha mencionado, $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$ para cierta función Φ, por lo que la función objetivo utilizando un Kernel es

$$\mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.16}$$

Lo anterior implica que las evaluaciones $\Phi(\mathbf{x}_i)$ no necesitan realizarse, ni tampoco el producto interno entre las evaluaciones. Es decir, a través de las evaluaciones $K(\mathbf{x}_i, \mathbf{x}_j)$ mediante el Kernel, tal producto interno es calculado implícitamente. Para consideraciones prácticas, el hecho de que la matriz Kernel satisfaga la condición de simetría y positividad implica que el problema de optimización del método MSV es convexo, por lo que el problema siempre tiene solución.

Por otra parte, se ha visto que la solución del problema está dada por

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^* y_i \Phi(\mathbf{x}_i)^\intercal \Phi(\boldsymbol{x}) + b^* \tag{2.17}$$

y debido a que $\langle \Phi(\mathbf{x}_i), \Phi(\boldsymbol{x}) \rangle_{\mathcal{H}} = K(\mathbf{x}_i, \boldsymbol{x})$, la solución al utilizar un Kernel es

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^* y_i K(\mathbf{x}_i, \boldsymbol{x}) + b^* \tag{2.18}$$

por lo que la solución sólo depende de las evaluaciones a través del Kernel entre la nueva observación $\boldsymbol{x}$ y los datos de entrenamiento.

2.4. Dualidad

En esta sección se discuten las propiedades el problema de programación cuadrática del método MSV, así como otros detalles matemáticos de relevancia para la implementación del método.

Concretamente, se discuten detalles sobre la formulación dual con margen suave del método MSV:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j K_{ij} \\ \text{sujeto a:} \quad & 0 \leq \alpha_i \leq C \quad \forall i \\ & \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \forall i \end{aligned} \tag{2.19}$$

en donde $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.

El intercepto óptimo b^* puede ser determinado a partir del problema primal. La figura 2.4 ilustra la geometría del problema dual.

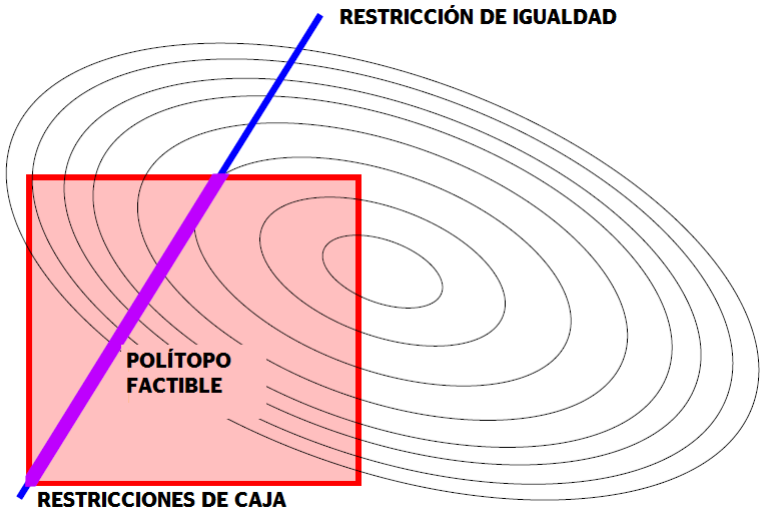

Figura 2.4: Geometría del problema dual 2.19. Las restricciones de caja $A_i \leq \alpha_i \leq B_i$ y la restricción de igualdad $\sum_i \alpha_i y_i = 0$ definen el polítopo factible, es decir, el dominio de los valores de $\boldsymbol{\alpha}$ que satisfacen las restricciones.

2.4.1. Construcción del problema dual

La dificultad del problema primal radica en la forma complicada de las restricciones de desigualdad que representan la condición del margen. Éstas pueden representarse mediante multiplicadores positivos de Lagrange $\alpha_i \geq 0$ de la siguiente forma:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i\left(w^\intercal\Phi(\mathbf{x}_i) + b\right) - 1 + \xi_i) \quad (2.20)$$

Formalmente, la función objetivo del problema dual $\underline{\mathcal{D}}(\boldsymbol{\alpha})$ se define como:

$$\underline{\mathcal{D}}(\boldsymbol{\alpha}) = \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) \quad \text{sujeto a:} \quad \xi_i \geq 1 \quad \forall i \qquad (2.21)$$

Este problema de minimización ya no contiene las restricciones complicadas expresadas por los multiplicadores de Lagrange, mientras que las restricciones dadas por $\xi_i \geq 1$ que aparecen en el problema dual son fáciles de manejar directamente. Por otra parte, argumentos diferenciales estándar llevan a la siguiente expresión analítica de la función objetivo dual:

$$\underline{\mathcal{D}}(\boldsymbol{\alpha}) = \begin{cases} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j K_{ij} \text{ si } \sum_i y_i\alpha_i = 0 \text{ y } \alpha_i \leq C \quad \forall i \\ -\infty \qquad\qquad\qquad\qquad\qquad\qquad \text{en otro caso} \end{cases} \qquad (2.22)$$

El problema dual 2.19 es la maximización de esta expresión sujeta a las restricciones $\alpha_i \geq 0$. Además, las condiciones $\sum_i y_i\alpha_i$ y $\alpha_i \leq C$ aparecen como restricciones en el problema dual ya que los casos en los que $\mathcal{D}(\boldsymbol{\alpha}) = -\infty$ no son útiles para la maximización.

La función diferenciable

$$\mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}^{n}\alpha_i\alpha_j y_i y_j K_{ij} \qquad (2.23)$$

coincide con la función dual $\underline{\mathcal{D}}(\boldsymbol{\alpha})$ cuando el vector $\boldsymbol{\alpha}$ satisface las restricciones del problema dual. Por abuso de lenguaje, en gran parte de la literatura se refiere a la función $\mathcal{D}(\boldsymbol{\alpha})$ como la función objetivo dual.

La definición formal de la función dual 2.21 asegura que la siguiente desigualdad es cierta para cualquier $(\mathbf{w}, b, \boldsymbol{\xi})$ que satisfagan las restricciones primales 2.12 y para cualquier $\boldsymbol{\alpha}$ que satisfaga las condiciones duales 2.13.

$$\mathcal{D}(\boldsymbol{\alpha}) = \underline{\mathcal{D}}(\boldsymbol{\alpha}) \leq \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) \leq \mathcal{P}(\mathbf{w}, b, \boldsymbol{\xi}) \tag{2.24}$$

Esta propiedad es la *dualidad débil*: el conjunto de valores que toma el problema primal están por arriba del conjunto de valores que toma el problema dual. Supóngase que se tiene el conjunto de valores $\boldsymbol{\alpha}^*$ y $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ tales que $\mathcal{D}(\boldsymbol{\alpha}^*) = \mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$. La desigualdad 2.24 implica que $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ y $\boldsymbol{\alpha}^*$ son soluciones de los problemas primal y dual; esta propiedad es la *dualidad fuerte*. Los problemas de optimización convexa con restricciones lineales se caracterizan por tener este tipo de soluciones.

2.4.2. Criterios de optimalidad

Sea $\boldsymbol{\alpha}^* = (\alpha_1^*, \ldots, \alpha_n^*)$ la solución del problema dual 2.19. Además, sea $\mathbf{g}^* = (g_1^*, \ldots, g_n^*)$ el vector de derivadas de la función objetivo en $\boldsymbol{\alpha}^*$, es decir:

$$g_i^* = \frac{\partial \mathcal{D}(\boldsymbol{\alpha}^*)}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{n} y_j \alpha_j^* K_{ij} \tag{2.25}$$

Por otra parte, debe notarse que que las restricciones $0 \leq \alpha_i \leq C$ pueden ser expresadas en función de las variables $\alpha_i y_i$ de la siguiente forma:

$$y_i \alpha_i \in [A_i, B_i] = \begin{cases} [0, +C] & \text{si } y_i = +1 \\ [-C, 0] & \text{si } y_i = -1 \end{cases} \tag{2.26}$$

Considérese un par de subíndidces i, j tales que se cumple $y_i \alpha_i^* < B_i$ y $A_j < y_j \alpha_j^*$. Ahora bien, sea $\boldsymbol{\alpha}^\epsilon = (\alpha_1^\epsilon, \ldots, \alpha_n^\epsilon) \in R^n$ de modo que

$$\alpha_k^\epsilon = \alpha_k^* + \begin{cases} +\epsilon y_k & \text{si } k = i \\ -\epsilon y_k & \text{si } k = j \\ 0 & \text{en otro caso} \end{cases} \tag{2.27}$$

Dado $\epsilon > 0$ suficientemente pequeño, el punto $\boldsymbol{\alpha}^\epsilon$ satisface las restricciones del problema dual. Por lo tanto $\mathcal{D}(\boldsymbol{\alpha}^\epsilon) \leq \mathcal{D}(\boldsymbol{\alpha}^*)$ ya que $\boldsymbol{\alpha}^*$ es solución. Por otra parte, considérese la expansión de primer orden:

$$\mathcal{D}(\boldsymbol{\alpha}^\epsilon) - \mathcal{D}(\boldsymbol{\alpha}^*) = \epsilon(y_i g_i^* - y_j g_j^*) + o(\epsilon) \tag{2.28}$$

Lo anterior implica que $y_i g_i^* - y_j g_j^* < 0$. Dado que esto ocurre para todas las parejas (i, j) tales que $y_i \alpha_i^* < B_i$ y $A_j < y_j \alpha_j^*$, se tiene la siguiente *condición necesaria de optimalidad*:

$$\exists \rho \in \mathbb{R} \text{ tal que } \max_{i \in I_{up}} \quad y_i g_i^* \leq \rho \leq \min_{j \in I_{down}} y_j g_j^* \tag{2.29}$$

en donde $I_{up} = \{i | y_i \alpha_i < B_i\}$ y $I_{down} = \{j | y_j \alpha_j > A_j\}$.

Usualmente, existen coeficientes α_k^* que se encuentran entre sus cotas inferior y superior de manera estricta. Además, dado que el valor $y_k g_k^*$ aparece en ambos lados de la desigualdad, esta situación deja un sólo posible valor para ρ. Por lo anterior, se puede reescribir 2.29 de la siguiente forma:

$$\exists \rho \in \mathbb{R} \text{ tal que } \forall k, \begin{cases} \text{si } y_k g_k^* > \rho \text{ entonces } y_k g_k^* = B_k \\ \text{si } y_k g_k^* < \rho \text{ entonces } y_k g_k^* = A_k \end{cases} \tag{2.30}$$

o equivalentemente:

$$\exists \rho \in \mathbb{R} \text{ tal que } \forall k, \begin{cases} \text{si } g_k^* > y_k \rho \text{ entonces } \alpha_k^* = C \\ \text{si } g_k^* < y_k \rho \text{ entonces } \alpha_k^* = 0 \end{cases} \tag{2.31}$$

Ahora bien, supóngase que elegimos los siguientes valores:

$$\mathbf{w}^* = \sum_k y_k \alpha_k^* \Phi(\boldsymbol{x}_k), \quad b^* = \rho, \quad \xi_k^* = \max\{0, g_k^* - y_k \rho\} \tag{2.32}$$

Estos valores satisfacen las restricciones del problema primal, y utilizando la relación 2.25 se tiene la siguiente igualdad:

$$\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) - \mathcal{D}(\boldsymbol{\alpha}^*) = C \sum_{k=1}^{n} \xi_k^* - \sum_{k=1}^{n} \alpha_k^* g_k^* = \sum_{k=1}^{n} (C \xi_k^* - \alpha_k^* g_k^*) \tag{2.33}$$

en donde el valor $(C\xi_k^* - \alpha_k^* g_k^*)$ puede calcularse utilizando 2.31 y 2.32. Por otra parte, la relación $(C\xi_k^* - \alpha_k^* g_k^*) = -y_k \alpha_k^* \rho$ se cumple independientemente de si g_k^* es menor, mayor, o igual a $y_k \rho$. Lo anterior implica:

$$\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) - \mathcal{D}(\boldsymbol{\alpha}^*) = -\rho \sum_{i=1}^{n} y_i \alpha_i^* = 0 \tag{2.34}$$

Esta *dualidad fuerte* tiene dos implicaciones:

- La elección hecha para $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ minimiza el problema primal ya que la desigualdad 2,24 establece que la función primal $\mathcal{P}(\mathbf{w}, b^*, \boldsymbol{\xi})$ no puede tomar valores más pequeños que $\mathcal{D}(\boldsymbol{\alpha}^*) = \mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$.

- Las condiciones de optimalidad 2.29, 2.30 y 2.31 son *necesarias y suficientes*. Para probar esto, supóngase que cualquiera de las condiciones mencionadas se cumple. La elección de $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ implica que por dualidad fuerte $\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) = \mathcal{D}(\boldsymbol{\alpha}^*)$. Lo anterior implica que $\boldsymbol{\alpha}^*$ maximiza el problema dual porque la desigualdad 2,24 asegura que la función dual $\mathcal{D}(\boldsymbol{\alpha})$ no puede tomar valores más grandes que $\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) = \mathcal{D}(\boldsymbol{\alpha}^*)$.

En particular, la condición necesaria y suficiente 2.29 es muy útil para la implementación del método MSV.

2.5. Solución rala del problema

El vector solución $\boldsymbol{\alpha}^*$ del problema dual 2.19 contiene muchos ceros. La función discriminante $f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^* y_i K(\mathbf{x}_i, \boldsymbol{x}) + b^*$ puede ser expresada utilizando solamente un subconjunto de los datos de entrenamiento, llamados *vectores de soporte*.

Desde el punto de vista computacional, esta propiedad se traduce en la oportunidad de reducir los requerimientos de memoria y de tiempo para las implementaciones del método MSV.

2.5.1. Vectores de soporte

La condición de optimalidad 2.31 caracteriza aquellas observaciones de entrenamiento que son vectores de soporte. Utilizando la ecuación para los gradientes 2.25 y los valores 2.32 se tiene que:

$$g_k - y_k \rho = 1 - y_k \sum_{i=1}^{n} y_i \alpha_i K_{ik} - y_k b^* = 1 - y_k f(\boldsymbol{x}_k) \tag{2.35}$$

que sustituyendo en la condición de optimalidad 2.31 se obtiene:

$$\begin{cases} \text{si } y_k f(\boldsymbol{x}_k) < 1 \text{ entonces } \alpha_k = C \\ \text{si } y_k f(\boldsymbol{x}_k) > 1 \text{ entonces } \alpha_k = 0 \end{cases} \tag{2.36}$$

Este resultado permite clasificar a los datos de entrenamiento en tres categorías:

- Datos $(\boldsymbol{x}_k, y_k)$ tales que $y_k f(\boldsymbol{x}_k) > 1$, los cuales no son vectores de soporte. Para estas observaciones se cumple que $\alpha_k = 0$.

- Datos $(\boldsymbol{x}_k, y_k)$ tales que $y_k f(\boldsymbol{x}_k) < 1$, los cuales son llamados *vectores de soporte acotados* ya que activan la desigualdad $\alpha_k \leq C$. Para estas observaciones se cumple que $\alpha_k = C$.

- Datos $(\boldsymbol{x}_k, y_k)$ tales que $y_k f(\boldsymbol{x}_k) = 1$, los cuales son llamados *vectores de soporte libres*. Para estas observaciones se cumple que $\alpha_k \in [0, C]$.

2.5.2. Complejidad computacional

Existen dos cotas inferiores intuitivas para el costo computaciones de cualquier algoritmo que resuelve el problema del método MSV para cualquier matriz Kernel arbitaria.

- Supóngase que se tiene una manera de saber cuáles datos no son vectores de soporte (i.e., $\alpha_i = 0$) y cuáles datos son vectores de soporte acotados. Los coeficientes de los restantes R vectores de soporte libres son determinados a través de un sistema lineal de ecuaciones que representa las derivadas de la función objetivo. Esto requiere un número de operaciones proporcional a R^3.

- Comprobar que un vector $\boldsymbol{\alpha}$ es solución del problema requiere el cálculo del gradiente $\boldsymbol{g}$ y verificar la condición de optimalidad dada por 2.29. Con n datos de entrenamiento y S vectores de soporte, esto requiere un número de operaciones proporcional a nS.

2.5.3. Cálculo de los valores Kernel

El cálculo de la matriz simétrica $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ no es un problema trivial. A continuación se presentan algunos de los problemas prácticos relacionados a su cálculo.

- *Calcular valores Kernel es costoso.* Calcular cada valor de Kernel involucra la manipulación de volúmenes de datos que pueden ser grandes.

Por ejemplo, en aplicaciones de *Computer Vision*, las imágenes contienen miles de pixeles; mientras que en *Text Mining*, los documentos contienen miles de palabras. En la práctica, el cálculo de los valores Kernel requiere más de la mitad del tiempo total de cómputo.

- *Calcular la matriz Kernel completa es ineficiente.* La expresión para los gradientes 2.25 sólo depende de los valores K_{ij} que involucran al menos un vector de soporte. Cada una de las condiciones de optimalidad 2.29, 2.30 y 2.31 puede ser verificada solamente con esos valores. El tiempo total de entrenamiento es usualmente más bajo que el tiempo requerido para calcular la matriz Kernel completa.

- *La matriz Kernel no cabe en memoria.* Cuando el número de datos de entrenamiento crece, la matriz Kernel se vuelve muy grande y no puede mantenerse guardada en memoria. Los valores Kernel necesitan ser calculados sobre la marcha o ser recuperados a través de la memoria caché.

En la práctica, estos problemas aparecen constantemente en cualquier implementación del método MSV.

Capítulo 3

LIBSVM

LIBSVM es una biblioteca de código abierto desarrollada por Chih-Chung Chang y Chih-Jen Lin de la Universidad Nacional de Taiwan, escritas en el lenguaje de programación **C++** para resolver problemas de clasificación y regresesión basados en el método MSV. La implementación está basada en el algoritmo *optimización mínima secuencial* (OMS) pero utiliza un esquema de elección de conjunto activo más avanzado, junto con otras heurísticas que hacen la implementación muy eficiente. Actualmente existen interfaces de esta biblioteca en otros lenguajes de programación; en particular, **Python**. Esta sección presenta los elementos algorítmicos y los detalles de implementación de la biblioteca **LIBSVM**.

3.1. Búsqueda de dirección

La optimización del problema dual es realizada a través de una *búsqueda de dirección* sobre direcciones sucesivas bien elegidas. Supóngase que se tiene un vector inicial $\boldsymbol{\alpha}$ que satisface las restricciones del problema dual 2.19. Por otra parte, decimos que una dirección $\mathbf{u} = (u_1, \ldots, u_n)$ es una *dirección factible* si podemos mover el vector $\boldsymbol{\alpha}$ sobre la dirección $\mathbf{u}$ sin violar las restricciones del problema dual.

Más formalmente, definimos el conjunto Λ como aquel que contiene a todos los coeficientes $\lambda \geq 0$ tales que el vector $\boldsymbol{\alpha}+\lambda\mathbf{u}$ satisface las restricciones del problema dual. Dado que el polítopo factible es convexo y acotado, el conjunto Λ es un intervalo acotado de la forma $\left[0, \lambda^{\text{máx}}\right]$.

Por lo tanto, se busca maximizar el problema dual 2.19 sujeto al subes-

pacio $\{\boldsymbol{\alpha} + \lambda\mathbf{u}, \lambda \in \Lambda\}$. Esto puede expresarse mediante el problema de optimización:

$$\lambda^* = \arg\max_{\lambda\in\Lambda} \quad \mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u}) \tag{3.1}$$

La figura 3.1 ilustra los valores de $\mathcal{D}(\boldsymbol{\alpha}+\lambda\mathbf{u})$ como función de λ. Además, el conjunto Λ es ilustrado por las regiones sombreadas. Por otra parte, dado que la función objetivo es cuadrática, $\mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u})$ es ilustrada como una parábola. El valor máximo λ^+ se calcula mediante el método de Newton:

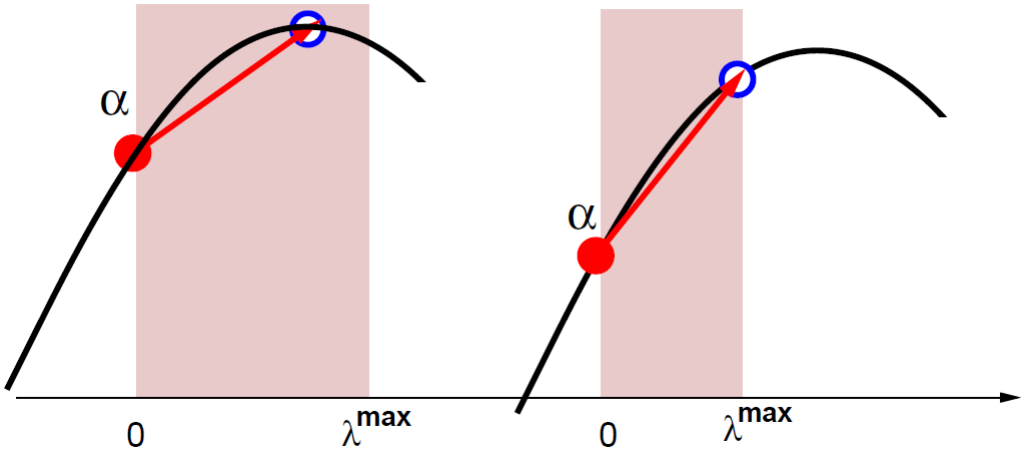

Figura 3.1: Dado un vector inicial $\boldsymbol{\alpha}$ y una dirección factible $\mathbf{u}$, la búsqueda de dirección maximiza la función $f(\lambda) = \mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u})$ para $\lambda \geq 0$ y $\boldsymbol{\alpha} + \lambda\mathbf{u}$ tal que se satisfacen las restricciones del problema dual 2.19.

$$\lambda^+ = \frac{\frac{\partial\mathcal{D}(\boldsymbol{\alpha}+\lambda\mathbf{u})}{\partial\lambda}}{\frac{\partial^2\mathcal{D}(\boldsymbol{\alpha}+\lambda\mathbf{u})}{\partial\lambda^2}} = \frac{\mathbf{g}^\intercal\mathbf{u}}{\mathbf{u}^\intercal\mathbf{H}\mathbf{u}} \tag{3.2}$$

en donde el vector $\mathbf{g}$ y la matriz $\mathbf{H}$ representan el gradiente y la Hessiana de la función objetivo del problema dual $\mathcal{D}(\boldsymbol{\alpha})$. Más explícitamente:

$$g_i = 1 - y_i \sum_j y_j\alpha_j K_{ij}, \quad H_{ij} = y_i y_j K_{ij} \tag{3.3}$$

Por lo tanto, la solución del problema es la proyección de λ^+ sobre el intervalo $\Lambda = \left[0, \lambda^{\text{máx}}\right]$, de modo que:

$$\lambda^* = \text{máx}\left\{0, \text{mín}\left\{\lambda^{\text{máx}}, \frac{\mathbf{g}^\intercal \mathbf{u}}{\mathbf{u}^\intercal \mathbf{H}\mathbf{u}}\right\}\right\} \tag{3.4}$$

A partir de un vector inicial factible, en cada iteración se selecciona una dirección factible y se aplica la fórmula de búsqueda de dirección 3.4 hasta alcanzar el máximo. Sin embargo, existen algunas dificultades con este enfoque:

- La restricción de igualdad en el problema 2.19 restringen a la dirección $\mathbf{u}$ al subespacio lineal $\sum_i y_i u_i = 0$.
- Las restricciones de caja en el problema 2.19 se convierten en restricciones sobre los signos en ciertos coeficientes del vector $\mathbf{u}$.
- El vector $\mathbf{u}$ debe ser una dirección de ascenso para asegurar que existe progreso hacia la solución.
- Puede existir convergencia más rápida cuando las direcciones sucesivas son conjugadas, es decir, cuando $\mathbf{u}^\intercal \mathbf{H}\mathbf{u}' = 0$ en donde $\mathbf{u}'$ es el último vector de dirección.

Encontrar una dirección que satisfaga simultáneamente todas estas restricciones es muy complicado. Los primeros algoritmos del método MSV solventaban estos problemas a través de reparametrizaciones del problema dual 2.19 que aumentaba sustancialmente el número de variables a optimizar. Como se discute más adelante, el algoritmo OMS utiliza un enfoque más eficiente.

3.2. El método de descomposición

Tal como se discute en secciones previas, el cálculo de la matriz Kernel completa es costoso e ineficiente. Los métodos de descomposición fueron diseñados para solventar este problema; éstos intentan resolver el problema dual completo 2.19 mediante la resolución de una secuencia de subproblemas cuadráticos más simples.

En lugar de actualizar todos los coeficientes del vector $\boldsymbol{\alpha}$, cada iteración del método de descomposición optimiza un subconjunto de coeficientes $\alpha_i, i \in$

$\mathcal{B}$ y mantiene el resto de coeficientes $\alpha_j, j \notin \mathcal{B}$ sin modificar. En este sentido, el método de descomposición es un caso particular de los métodos de conjunto activo.

Iniciando con un vector $\boldsymbol{\alpha}$ se puede calcular un nuevo vector $\boldsymbol{\alpha'}$ agregando una restricción adicional al problema dual 2.19 que representa los coeficientes sin modificar:

$$\begin{aligned} \max_{\boldsymbol{\alpha'}} \quad & \mathcal{D}(\boldsymbol{\alpha'}) = \textstyle\sum_{i=1}^{n} \alpha'_i - \frac{1}{2} \sum_{i,j}^{n} \alpha'_i \alpha'_j y_i y_j K_{ij} \\ \text{sujeto a:} \quad & 0 \leq \alpha'_i \leq C \quad \forall i \in \mathcal{B} \\ & \alpha'_i = \alpha_i \quad \forall i \notin \mathcal{B} \\ & \textstyle\sum_{i=1}^{n} \alpha'_i y_i = 0 \quad \forall i \end{aligned} \tag{3.5}$$

Éste último problema se puede reescribir como un problema de programación cuadrática en función de las variables $\alpha_i, i \in \mathcal{B}$ y eliminar los términos que no involucran las variables α'_i de la siguiente forma:

$$\begin{aligned} \max_{\boldsymbol{\alpha'}} \quad & \mathcal{D}(\boldsymbol{\alpha'}) = \textstyle\sum_{i \in \mathcal{B}} \alpha'_i \left(1 - y_i \sum_{j \notin \mathcal{B}} y_j \alpha_j K_{ij}\right) - \frac{1}{2} \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}} \alpha'_i \alpha'_j y_i y_j K_{ij} \\ \text{sujeto a:} \quad & 0 \leq \alpha'_i \leq C \quad \forall i \in \mathcal{B} \\ & \textstyle\sum_{i \in \mathcal{B}} y_i \alpha'_i = -\sum_{j \notin \mathcal{B}} y_j \alpha_j \end{aligned} \tag{3.6}$$

El algoritmo 1 ilustra a grandes rasgos el método de descomposición.

Algorithm 1 Método de Descomposición

$\forall k \in \{1, \ldots, n\}\ \alpha_k \leftarrow 0$
$\forall k \in \{1, \ldots, n\}\ g_k \leftarrow 1$
loop
  $G^{max} \leftarrow \max_i y_i g_i$ sujeto a $y_i \alpha_i < B_i$
  $G^{min} \leftarrow \min_j y_j g_j$ sujeto a $A_j < y_j \alpha_j$
  if $G^{max} \leq G^{min}$ **stop**
  Elegir un conjunto activo $\mathcal{B} \subset \{1, \ldots, n\}$
  $\boldsymbol{\alpha'} \leftarrow$ Solución del problema 3.6
  $\forall k \in \{1, \ldots, n\}\ g_k \leftarrow g_k - y_k \sum_{i \in \mathcal{B}} y_i (\alpha'_i - \alpha_i) K_{ik}$
  $\forall i \in \mathcal{B} \quad \alpha_i \leftarrow \alpha'_i$
end loop

Por construcción del problema 3.5, se garantiza que $\mathcal{D}(\boldsymbol{\alpha}') \geq \mathcal{D}(\boldsymbol{\alpha})$. La cuestión restante ahora es definir un esquema para la elección del conjunto activo que asegure que los valores crecientes del problema dual alcancen el máximo.

3.3. Optimización mínima secuencial

El método de *optimización mínima secuencial* fue propuesto por Platt (1999) y es un caso particular del método de descomposición. A continuación se enlistan algunos hechos sobre este método.

- Cada subproblema sucesivo de programación cuadrática tiene solamente dos variables. La restricción de igualdad hace a cada subproblema uno de optimización de una sola dimensión. Por otra parte, una sola búsqueda de dirección es suficiente para calcular la solución.
- El cálculo de paso de Newton es rápido ya que la dirección $\mathbf{u}$ solamente contiene dos coeficientes diferentes de cero.

El algoritmo 2 del método OMS realiza la elección del conjunto activo mediante el esquema de la pareja de máxima violación. Adicionalmente, cada subproblema es resuelto mediante una búsqueda sobre la dirección $\mathbf{u}$ que contiene solamente dos coeficientes distintos de cero $u_i = y_i$ y $u_j = -y_j$.

Algorithm 2 OMS con elección de conjunto activo por pareja de máxima violación

$\forall k \in \{1, \ldots, n\}\, \alpha_k \leftarrow 0$
$\forall k \in \{1, \ldots, n\}\, g_k \leftarrow 1$
loop
  $i \leftarrow \text{máx}_i\, y_i g_i$ sujeto a $y_i \alpha_i < B_i$
  $j \leftarrow \text{mín}_j\, y_j g_j$ sujeto a $A_j < y_j \alpha_j$
  if $y_i g_i \leq y_j g_j$ **stop**
  $\lambda \leftarrow \text{mín} \left\{ B_i - y_i \alpha_i, y_j \alpha_j - A_j, \frac{y_i g_i - y_j g_j}{K_{ii} + K_{jj} - 2K_{ij}} \right\}$
  $\forall k \in \{1, \ldots, n\}\, g_k \leftarrow g_k - \lambda y_k K_{ik} + \lambda y_k K_{jk}$
  $\alpha_i \leftarrow \alpha_i + y_i \lambda$
  $\alpha_j \leftarrow \alpha_j - y_j \lambda$
end loop

La parte más intensiva de cada iteración del algoritmo 2 es la actualización del gradiente, ya que se requiere el cálculo de dos renglones completos de la matriz Kernel. La técnica de *reducción* (*shrinking*) (Joachims, 1999) permite reducir este cálculo.

3.4. Criterio de paro

El algoritmo implementado en la biblioteca **LIBSVM** se detiene cuando la condición de optimalidad 2.29 se logra con cierta precisión ϵ, es decir, cuando se cumple:

$$\max_{i \in I_{up}} y_i g_i - \min_{j \in I_{down}} y_j g_j < \epsilon \tag{3.7}$$

en donde $I_{up} = \{i | y_i \alpha_i < B_i\}$ y $I_{down} = \{j | y_j \alpha_j > A_j\}$ como en la condición de optimalidad 2.29.

Existen resultados teóricos que establecen que el algoritmo OMS alcanzan la condición de paro 3.7 después de un tiempo finito. Estos resultados dependen en gran medida de la manera en la que se elije el conjunto activo.

3.5. Elección del conjunto activo

Existen varias posibilidades para elegir los índices (i, j) que forman el conjunto activo en cada iteración del algoritmo OMS. Supóngase que en alguna iteración se tiene el vector de coeficientes $\boldsymbol{\alpha}$. Solamente dos coeficientes de la solución $\boldsymbol{\alpha'}$ del algoritmo OMS difieren de los coeficientes del vector $\boldsymbol{\alpha}$, por lo cual $\boldsymbol{\alpha'} = \boldsymbol{\alpha} + \lambda \mathbf{u}$ en donde la dirección $\mathbf{u}$ solamente contiene dos coeficientes diferentes de cero. Por otra parte, la condición de igualdad implica $\sum_k y_k u_k = 0$. Por consiguiente, es suficiente considerar direcciones $\mathbf{u}^{ij} = (u_1^{ij}, \ldots, u_n^{ij})$ tales que:

$$u_k^{ij} = \begin{cases} +y_i \text{ si } k = i \\ -y_j \text{ si } k = j \\ 0 \quad \text{en otro caso} \end{cases} \tag{3.8}$$

El subproblema de optimización requiere una búsqueda de dirección sobre $\mathbf{u}^{ij}$ cuando $\lambda > 0$ o sobre $-\mathbf{u}^{ij}$ cuando $\lambda < 0$. Entonces la elección del conjunto activo en el algoritmo OMS se reduce a una búsqueda de un vector

de dirección de la forma 3.8. Además, como se necesita una dirección factible, se requiere también que $i \in I_{up}$ y $j \in I_{down}$.

3.5.1. Elección por máxima ganancia

Sea $\mathcal{U} = \{\mathbf{u}^{ij} | i \in I_{up}, j \in I_{down}\}$ el conjunto de todas las potenciales direcciones de búsqueda. La dirección más efectiva para cada iteración debería ser aquella que maximiza el incremento de la función objetivo dual, es decir:

$$\begin{aligned} \mathbf{u}^* = \arg \max_{\mathbf{u}^{ij} \in \mathcal{U}} \quad & \max_{0 \leq \lambda} \quad \mathcal{D}(\boldsymbol{\alpha} + \lambda \mathbf{u}^{ij}) - \mathcal{D}(\boldsymbol{\alpha}) \\ \text{sujeto a} \quad & y_i \alpha_i + \lambda \leq B_i, \quad y_j \alpha_j - \lambda \geq A_j \end{aligned} \tag{3.9}$$

Desfortunadamente, la búsqueda de la mejor dirección $\mathbf{u}^{ij}$ requiere iterar sobre las $n(n-1)$ posibles parejas de índices. La maximización de λ implica una búsqueda de dirección; ésta búsqueda repetitiva de direcciones implicaría acceder a la matriz Kernel completa durante cada iteración del algoritmo OMS. Esto no es viable para un algoritmo rápido ya que cada iteración puede ser muy lenta.

3.5.2. Elección por pareja de máxima violación

Una manera intuitiva de simplificar el problema 3.9 consiste en realizar una aproximación de primer orden de la función objetivo:

$$\mathcal{D}(\boldsymbol{\alpha} + \lambda \mathbf{u}^{ij}) - \mathcal{D}(\boldsymbol{\alpha}) \approx \lambda \mathbf{g}^\intercal \mathbf{u}^{ij} \tag{3.10}$$

y con la finalidad de hacer esta aproximación válida, reemplazar las restricciones por una restricción que garantice que λ es suficientemente pequeño, es decir:

$$\mathbf{u}^* = \arg \max_{\mathbf{u}^{ij} \in \mathcal{U}} \quad \max_{0 \leq \lambda \leq \epsilon} \quad \lambda \mathbf{g}^\intercal \mathbf{u}^{ij} \tag{3.11}$$

Por otra parte, se puede asumir que existe una dirección $\mathbf{u} \in \mathcal{U}$ tal que $\mathbf{g}^\intercal \mathbf{u} > 0$ ya que de otra manera, se habría alcanzado el óptimo. La maximización sobre λ lleva al problema:

$$\mathbf{u}^* = \arg \max_{\mathbf{u}^{ij} \in \mathcal{U}} \quad \mathbf{g}^\intercal \mathbf{u}^{ij} \tag{3.12}$$

Debe notarse que lo anterior puede escribirse de la siguiente forma:

$$\max_{\mathbf{u}^{ij}\in\mathcal{U}} \quad \mathbf{g}^\intercal \mathbf{u}^{ij} = \max_{i\in I_{up}, j\in I_{down}} (y_i g_i - y_j g_j) = \max_{i\in I_{up}} y_i g_i - \min_{j\in I_{down}} y_j g_j \tag{3.13}$$

De aquí puede reconocerse la condición de optimalidad 2.29. Por lo tanto, la solución de 3.12 es la *pareja de máxima violación* (Keerthi et al., 2001):

$$i = \arg\max_{k\in I_{up}} \quad y_k g_k, \quad j = \arg\min_{k\in I_{down}} \quad y_k g_k \tag{3.14}$$

El cálculo anterior requiere un tiempo proporcional a n. Los resultados de este cálculo pueden ser utilizados también para el criterio de paro 3.7.

3.5.3. Elección por criterio de segundo orden

Debe notarse que el cálculo de 3.14 no requiere valores Kernel adicionales; sin embargo, existen valores Kernel que eventualmente se necesitan para realizar una iteración del algoritmo OMS.

En vez de una aproximación lineal del problema 3.9, se puede considerar la ganancia cuadrática:

$$\mathcal{D}(\boldsymbol{\alpha} + \lambda \mathbf{u}^{ij}) - \mathcal{D}(\boldsymbol{\alpha}) \approx \lambda(y_i g_i - y_j g_j) - \frac{\lambda^2}{2}(K_{ii} + K_{jj} - 2K_{ij}) \tag{3.15}$$

y eliminar las restricciones. La solución para λ está dada por:

$$\lambda^* = \frac{y_i g_i - y_j g_j}{K_{ii} + K_{jj} - 2K_{ij}} \tag{3.16}$$

y su ganancia correspondiente es:

$$\frac{(y_i g_i - y_j g_j)^2}{2(K_{ii} + K_{jj} - 2K_{ij})} \tag{3.17}$$

Sin embargo, el problema resultante:

$$\mathbf{u}^* = \arg\max_{\mathbf{u}^{ij}\in\mathcal{U}} \quad \frac{(y_i g_i - y_j g_j)^2}{2(K_{ii} + K_{jj} - 2K_{ij})} \quad \text{sujeto a} \quad y_i g_i > y_j g_j \tag{3.18}$$

requiere una búsqueda exhaustiva sobre las $n(n-1)$ posibles parejas de índices. Por lo tanto, una implementación viable con un esquema de segundo orden debe restringir esta búsqueda de forma heurística.

El algoritmo implementado en la biblioteca **LIBSVM** utiliza el siguiente procedimiento (Fan et al., 2005):

$$\begin{aligned} i = & \arg \max_{k \in I_{up}} \quad y_k g_k \\ j = & \arg \max_{k \in I_{down}} \quad \frac{(y_i g_i - y_k g_k)^2}{2(K_{ii} + K_{kk} - 2K_{ik})} \text{ sujeto a } y_i g_i > y_k g_k \end{aligned} \tag{3.19}$$

Como se puede observar, el cálculo para el índice i es exacto de acuerdo al esquema de máxima pareja de violación. Por otra parte, el cálculo para el índice j puede ser realizado en un tiempo proporcional a n. Este último cálculo requiere la diagonal de la matriz Kernel, la cual es almacenada en memoria caché, así como el i-ésimo renglón de la matriz Kernel. Este renglón se necesita también para actualizar el vector gradiente. La convergencia de este esquema de elección de conjunto activo es probado en (Chen et al., 2006).

Debe notarse que los problemas de primer orden 3.12 y de segundo orden 3.18 solamente son utilizados para la elección del conjunto activo; éstos no tienen que mantener la restricción de factibilidad del problema de máxima ganancia 3.9, sin embargo, sí deben tomarse en cuenta para el cálculo de $\boldsymbol{\alpha}'$ después de la elección del conjunto activo.

3.6. La estrategia de *reducción*

La estrategia de *reducción* reduce el tamaño del problema al eliminar temporalmente algunas variables α_i que son improbables de ser seleccionadas como parte del conjunto activo en el algoritmo OMS ya que han alcanzado su cota inferior o superior (Joachims, 1999). De esta forma, las iteraciones en el algoritmo OMS continúan sobre el resto de variables. Adicionalmente, esta técnica reduce el número de valores Kernel que se necesitan para actualizar el vector del gradiente (ver el algoritmo 2).

En muchos problemas, el número de vectores de soporte libres es relativamente pequeño. Durante el proceso iterativo del algoritmo de la biblioteca **LIBSVM**, se identifica una partición de los datos de entrenamiento en: datos que no son vectores de soporte ($\alpha_i = 0$), vectores de soporte acotados

($\alpha_i = C$) y vectores de soporte libres. Los coeficientes asociados con datos que no son vectores de soporte y con vectores de soporte acotados son conocidos con alta precisión y son buenos candidatos para la *reducción*.

Considérense los siguientes conjuntos:

$$J_{up}(\boldsymbol{\alpha}) = \{k | y_k g_k > m(\boldsymbol{\alpha})\}$$
$$J_{down}(\boldsymbol{\alpha}) = \{k | y_k g_k < M(\boldsymbol{\alpha})\}$$

en donde $m(\boldsymbol{\alpha}) = \underset{i \in I_{up}}{\text{máx}} y_i g_i$ y $M(\boldsymbol{\alpha}) = \underset{j \in I_{down}}{\text{mín}}\ y_i g_i$ y en donde J_{up}, J_{down} y g_k dependen implícitamente del vector $\boldsymbol{\alpha}$. Con estas definiciones, se tiene que:

$$k \in J_{up}(\boldsymbol{\alpha}) \Rightarrow k \notin I_{up} \Rightarrow \alpha_k = y_k B_k$$
$$k \in J_{down}(\boldsymbol{\alpha}) \Rightarrow k \notin I_{down} \Rightarrow \alpha_k = y_k A_k$$

Es decir, estos conjuntos contienen los índices de las variables α_k que han alcanzado su cota inferior o superior con una derivada suficientemente fuerte, por lo que es probable que se mantengan así.

Dos hechos importantes, basados en (Chen et al., 2006,), son los siguientes:

- Si $\boldsymbol{\alpha^*}$ es una solución arbitraria del problema dual, entonces los valores $m(\boldsymbol{\alpha^*}), M(\boldsymbol{\alpha^*})$ y $y_k g_k(\boldsymbol{\alpha^*})$ no dependen de la solución elegida. Por consiguiente, también los conjuntos J^*_{up} y J^*_{down} son independientes de la solución elegida.

- Existe un número finito de conjuntos posibles $J_{up}(\boldsymbol{\alpha})$ y $J_{down}(\boldsymbol{\alpha})$. Ambos conjuntos alcanzan y mantienen sus valores finales J^*_{up} y J^*_{down} después de un número finito de iteraciones del algoritmo OMS.

Por lo anterior, las variables $\alpha_i, i \in J^*_{up} \cup J^*_{down}$ también alcanzan sus valores finales después de un numero finito de iteraciones y pueden ser eliminadas del problema de optimización. Sin embargo, en la práctica, no puede haber completa seguridad de que $J_{up}(\boldsymbol{\alpha})$ y $J_{down}(\boldsymbol{\alpha})$ han alcanzado sus valores finales J^*_{up} y J^*_{down}. Para esto, se pueden eliminar de forma tentativa las variables y después revisar si la deducción fue correcta. La biblioteca **LIBSVM** implementa dos estrategias:

- La rutina de *reducción* es invocada cada mín$(n, 1000)$ iteraciones. De forma dinámica se eliminan las variables α_i cuyos índices pertenecen a $J_{up}(\boldsymbol{\alpha}) \cup J_{down}(\boldsymbol{\alpha})$.

- Dado que esta estregia es relativamente agresiva, una rutina para revertir la *reducción* (*unshrinking*) es invocada cada vez que $m(\boldsymbol{\alpha}^k) - M(\boldsymbol{\alpha}^k) < 10\epsilon$ en donde ϵ es el valor de tolerancia que aparece en 3.7. Para esto, el vector gradiente $\mathbf{g}$ es reconstruido y todas las variables que no pertenecen a $J_{up}(\boldsymbol{\alpha})$ o $J_{down}(\boldsymbol{\alpha})$ son reactivadas.

Debe notarse que la reconstrucción del vector completo del gradiente $\mathbf{g}$ puede ser muy costosa. Para mitigar este costo, la biblioteca **LIBSVM** guarda un vector adicional $\bar{\mathbf{g}} = (\overline{g_1}, \ldots, \overline{g_n})$:

$$\overline{g_i} = y_i \sum_{\alpha_k = C} y_k \alpha_k K_{ik} = y_i C \sum_{\alpha_k = C} y_k K_{ik} \tag{3.20}$$

durante las iteraciones del algoritmo OMS. Este vector necesita actualizarse cada vez que en alguna iteración, un coeficiente α_k alcanza o abandona la cota superior C. El vector completo del gradiente es reconstruido utilizando la relación:

$$g_i = 1 - \overline{g_i} - y_i \sum_{0 < \alpha_k < C} y_k \alpha_k K_{ik} \tag{3.21}$$

La rutina de *reducción* nunca elimina las variables libres en donde $0 < \alpha_k < C$, por lo que la reconstrucción del gradiente solamente necesita ciertos renglones de la matriz Kernel.

3.7. Aspectos numéricos sobre la implementación

Dos aspectos sobre la implementación del algoritmo requieren especial atención: precisión numérica y el *caching*.

- **Precisión numérica**: Algunas partes del algoritmo distinguen entre aquellas variables α_i que han alcanzado sus cotas superior o inferior y el resto de variables. Cálculos inexactos podrían cambiar la partición de estas variables con consecuencias negativas. El algoritmo implementado

en la biblioteca **LIBSVM** se asegura de que cuando una búsqueda de dirección alcanza las restricciones de caja, al menos uno de los dos coeficientes α_i o α_j sea exactamente igual a su cota.

- ***Caching***: Cada entrada i de la memoria caché de la biblioteca **LIBSVM** representa el i-ésimo renglón de la matriz Kernel, pero almacena solamente los primeros $l_i \leq n$ coeficientes $K_{i1}, \ldots, K_{il_i}$. Las variables l_i son dinámicamente ajustadas para reflejar los coeficientes conocidos. Además, la estrategia de *reducción* es realizada permutando los datos para asignar los índices más bajos al conjunto activo. Las entradas del conjunto activo son agrupadas al inicio de cada renglón de la matriz Kernel. Para intercambiar las posiciones de dos datos de entrenamiento, se debe realizar el intercambio en los vectores $\boldsymbol{\alpha}, \mathbf{g}, \bar{\mathbf{g}}$ y sus entradas correspondientes en caché. Cuando el algoritmo OMS necesita los primeros l elementos de un renglón i en particular, el código de *caching* de la biblioteca **LIBSVM** recupera la entrada i y el número l_i de coeficientes en memoria caché. Además, el código de *caching* mantiene una lista circular sobre los renglones utilizados recientemente; cuando la memoria utilizada para los renglones en caché excede el máximo establecido, los coeficientes en caché para los renglones menos utilizados son desasignados.

Capítulo 4

SVM-SGD

SVM-SGD es una implementación del método *descenso del gradiente estocástico* (**SGD**) para el caso particular del método MSV. Una de las implementaciones más populares de este método es de (Bottou, 2010) y está escrita en **C++**, siguiendo un esquema de tasa de aprendizaje basado en (Shalev-Shwartz et al., 2007). Esta sección presenta los detalles de la implementación de este método en su interfaz para **Python**.

4.1. Descenso del gradiente

El método del *descenso del gradiente* es una técnica clásica para encontrar el mínimo de una función $f(\mathbf{w})$ continua y diferenciable. Este método usa el hecho de que el gradiente de una función ∇f apunta en la dirección de máximo crecimiento; lo anterior implica que $-\nabla f$ apunta en la dirección de máximo descenso. De esta forma, un algoritmo iterativo para encontrar el mínimo de la función $f(\mathbf{w})$ consiste en actualizar la estimación del parámetro $\mathbf{w}$ mediante la siguiente regla:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t) \tag{4.1}$$

en donde η_t es la *tasa de aprendizaje*, la cual determina qué tanto se debe mover la actualización del parámetro en la dirección del gradiente. Este valor debe elegirse de forma cuidadosa ya que valores muy pequeños pueden hacer la convergencia muy lenta, y valores muy grandes pueden hacer perder la convergencia. En el contexto de los problemas de aprendizaje, la función a

minimizar es $Q(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{x}_i, y_i; \mathbf{w}) + \lambda R(\mathbf{w})$ y su gradiente está dado por:

$$\nabla Q(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\mathbf{x}_i, y_i; \mathbf{w}) + \lambda \nabla R(\mathbf{w}) \tag{4.2}$$

y por lo tanto, la regla de actualización es:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left[\frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\mathbf{x}_i, y_i; \mathbf{w}_t) + \lambda \nabla R(\mathbf{w}_t) \right] \tag{4.3}$$

Por otra parte, el método MSV puede ser interpretado como un caso particular dentro del contexto de la teoría del aprendizaje estadístico. La función objetivo del problema primal 2.12 puede ser escrita de la siguiente forma:

$$\frac{1}{n}\sum_{i=1}^{n} \left[1 - f(\mathbf{x}_i; \mathbf{w}) y_i\right]_+ + \lambda \|\mathbf{w}\|^2 \tag{4.4}$$

en donde $\lambda = \frac{1}{2nC}$ y en donde la función $[^{\intercal}]_+$ denota la parte positiva de su argumento. De esta forma, se tiene que la función de pérdida es:

$$\ell(f(\mathbf{x}; \mathbf{w}), y) = \left[1 - f(\mathbf{x}; \mathbf{w}) y\right]_+ \tag{4.5}$$

la cual es conocida en la literatura como la función de pérdida *hinge loss*. La figura 4.1 ilustra la forma de esta función. Por otra parte, el término de regularización es:

$$R(\mathbf{w}) = \|\mathbf{w}\|^2 \tag{4.6}$$

De acuerdo a lo anterior, la regla de actualización para el método MSV toma la forma:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \begin{cases} \lambda \mathbf{w}_t & \text{si } f(\mathbf{x}_t; \mathbf{w}_t) y_t > 1 \\ \lambda \mathbf{w}_t - y_t \mathbf{x}_t & \text{en otro caso} \end{cases} \tag{4.7}$$

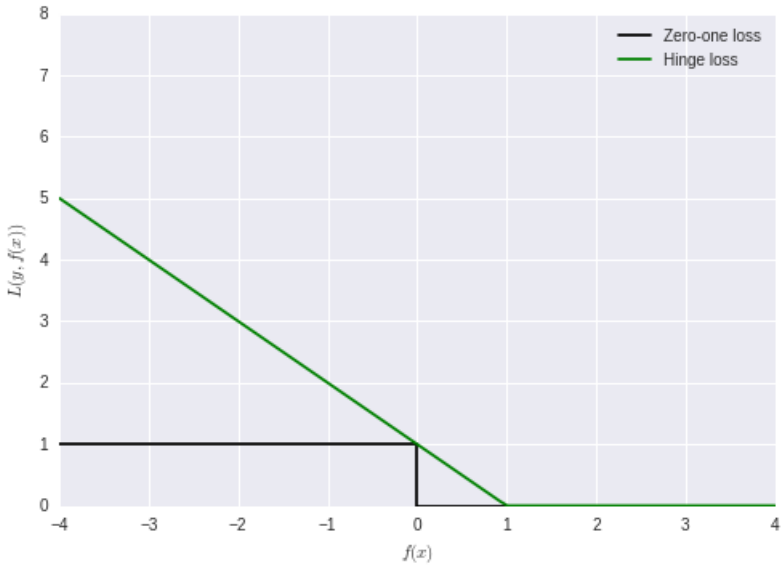

Figura 4.1: Función de pérdida *hinge* y función de pérdida *cero-uno*.

4.2. Descenso del gradiente estocástico

En la formulación estándar del método de descenso del gradiente, se utilizan todas las observaciones disponibles. Una modificación simple es calcular el gradiente respecto a una sola observación. Este método es llamado *descenso en gradiente estocástico*. Bajo este enfoque, la regla del método para $Q(\mathbf{w})$ está dada por:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left[\nabla \ell(\mathbf{x}_{i(t)}, y_{i(t)}; \mathbf{w}_t) + \lambda \nabla R(\mathbf{w}_t)\right] \tag{4.8}$$

en donde $i(t)$ es un índice obtenido seudoaleatoriamente del conjunto $\{1, \ldots, n\}$. Este proceso es repetido iterativamente pasando varias veces por todos las observaciones en el conjunto de entrenamiento; a cada una de estas veces se le denomina *período* (*epoch*). En valor esperado, esta actualización es la misma que en el método estándar ya que:

$$\mathbb{E}_{i(t)} \left[\ell(\mathbf{x}_{i(t)}, y_{i(t)}; \mathbf{w}_t)\right] = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{x}_i, y_i; \mathbf{w}_t) \tag{4.9}$$

4.3. Tasa de aprendizaje

La tasa de aprendizaje η_t puede mantenerse constante o disminuir conforme el algoritmo realiza las iteraciones. En el caso de la implementación del algoritmo **SVM-SGD** de la biblioteca **Scikit-Learn**, el esquema seguido para la tasa de aprendizaje es el propuesto por (Shalev-Shwartz et al., 2007):

$$\eta_t = \frac{1}{\lambda(t_0 + t)} \tag{4.10}$$

en donde t es un contador de las iteraciones, de modo que hay un total de iteraciones dado por $T =$ número de observaciones $\times$ número de períodos y t_0 es determinado mediante la siguiente heurística:

$$\begin{aligned} \rho &= \sqrt{\frac{1}{\sqrt{\lambda}}} \\ \eta_0 &= \frac{\rho}{\text{máx}(1, \nabla\ell(-\rho, 1))} \\ t_0 &= \frac{1}{\eta_0 \lambda} \end{aligned} \tag{4.11}$$

El algoritmo 3 ilustra a grandes rasgos el procedimiento implementado en la biblioteca **Scikit-Learn** para el método **SVM-SGD**.

Algorithm 3 Método de Descenso del gradiente estocástico para MSV

$\mathbf{w}_0 \leftarrow 0$
Calcular η_0 y t_0 de acuerdo a la heurística 4.11.
loop
  Seleccionar seudoaleatoriamente un índice $i(t)$ del conjunto $\{1, \ldots, n\}$.

  Calcular el gradiente $\nabla Q(\mathbf{x}_{i(t)}, y_{i(t)}; \mathbf{w}_t) = \nabla\ell(\mathbf{x}_{i(t)}, y_{i(t)}; \mathbf{w}_t) + \lambda\nabla R(\mathbf{w}_t)$.
  Calcular la tasa de aprendizaje: $\eta_t = \frac{1}{\lambda(t_0+t)}$
  Actualizar el vector de parámetros: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla Q(\mathbf{x}_{i(t)}, y_{i(t)}; \mathbf{w}_t)$.
  if $Q(\mathbf{w}_{t+1}) - Q(\mathbf{w}_t) \leq \epsilon$ **stop**
end loop

Capítulo 5

Experimentos numéricos

Los experimentos numéricos propuestos para comparar las implementaciones **LIBSVM** y **SVM-SGD** se clasifican en dos categorías:

- **Problemas simulados**: El propósito es medir el desempeño y tiempos de procesamiento de forma controlada variando el tamaño del conjunto de datos de entrenamiento.
- **Problemas reales**: El propósito es medir el desempeño y tiempos de procesamiento en conjuntos de datos inspirados en problemas reales, obtenidos de repositorios populares.

Los experimentos fueron llevados a cabo en una computadora corriendo de forma virtual el sistema operativo Linux Ubuntu 14.04, con memoria RAM de 8GB y un procesador Intel Core i7. El código utilizado y los resultados pueden ser consultados en `https://github.com/pjcv89/thesis_math`

El desempeño de los modelos se mide con la precisión (*accuracy*), que representa el porcentaje de casos correctamente clasificados, tanto en el conjunto de datos de entrenamiento (*train*) como en el conjunto de datos de prueba (*test*). El tiempo de entrenamiento se mide con el módulo **clock** de la biblioteca **time** de **Python**, el cual es el tiempo de CPU y se expresa en segundos como un número representado en punto flotante.

5.1. Problemas simulados

Con la finalidad de controlar los experimentos, es necesario establecer valores por defecto de los parámetros para ambos métodos.

Para **LIBSVM**:

- Parámetro de regularización: $C = 1$.
- Cantidad de caché: $m = 100$ megabytes.
- *Reducción*: $shrinking = 1$.
- Parámetro del Kernel RBF: $\gamma = 0{,}1$.

mientras que para **SVM-SGD**:

- Parámetro de regularización: $\lambda = 1$.
- Número de *períodos*: $n\_epochs = 10$.

El número de variables en ambos casos se fija en $n\_vars = 10$. Por otra parte, el tamaño del conjunto de datos, medido en cantidad de observaciones, fue generado siguiendo una escala logarítmica: $10^2, \ldots, 10^5$. Adicionalmente, se crean 20 % de observaciones como conjunto de datos de prueba para cada caso.

Simulación de datos

- Problemas linealmente separables: Los datos se crean a partir del módulo **datasets.make_classification** de la biblioteca **Scikit-Learn** de **Python**. La figura 5.1 muestra un ejemplo para el caso particular de 2 variables.
- Problemas no linealmente separables: Los datos se crean a partir del módulo **random.randn** que genera datos a partir de distribuciones normales estándar y las clases se crean a partir del módulo **logical_xor**, ambos de la biblioteca **numpy** de **Python**. La figura 5.2 muestra un ejemplo para el caso particular de 2 variables.

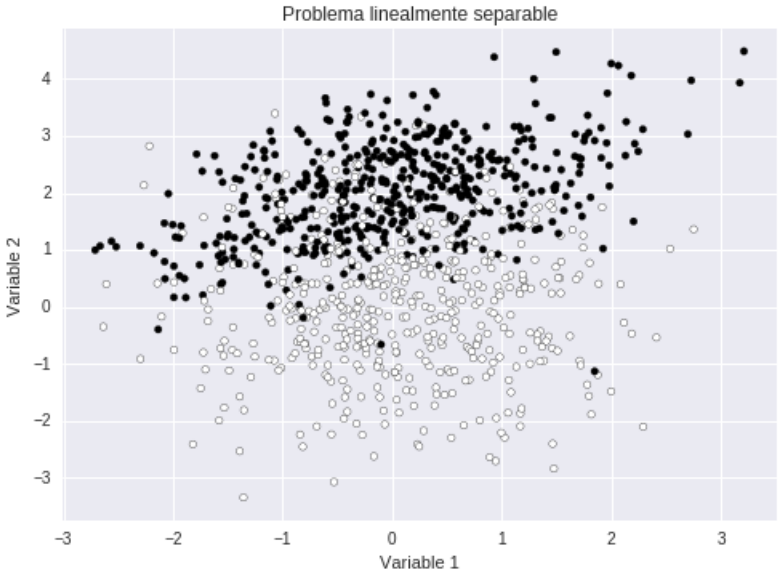

Figura 5.1: Ejemplo de datos simulados de un problema linealmente separable con 2 variables para una realización de $n = 1000$ observaciones.

5.1.1. Curvas de aprendizaje

En este experimento se construyen las curvas de aprendizaje para el problema linealmente separable comparando **SVM-SGD**, **LIBSVM** con Kernel Lineal y **LIBSVM** con Kernel Gaussiano de base radial (RBF), mientras que para el problema no linealmente separable se comparan **SVM-SGD** y **LIBSVM** con Kernel RBF. En las curvas de aprendizaje se observa el comportamiento de la precisión sobre el conjuntos de datos de entrenamiento y el conjunto de datos de prueba al variar el número de datos de entrenamiento. Los resultados en las siguientes tablas muestran la precisión en el conjunto de datos de entrenamiento, la precisión en el conjunto de datos de prueba y el tiempo de entrenamiento. Las figuras 5.3, 5.4 y 5.5 ilustran las curvas de aprendizaje para el caso del problema linealmente separable para **LIBSVM** con Kernel Lineal, el caso del problema linealmente separable para **LIBSVM** con Kernel RBF y el caso del problema no linealmente separable para **LIBSVM** con Kernel RBF.

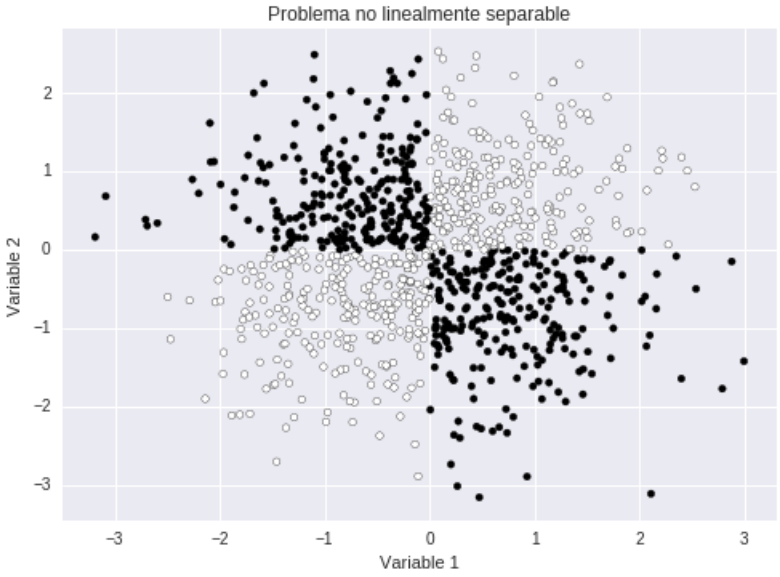

Figura 5.2: Ejemplo de datos simulados de un problema no linealmente separable con 2 variables para una realización de $n = 1000$ observaciones.

Problema lineal	10^2	10^3	10^4	10^5
SVM-SGD: Acc. Train	92.0	94.3	94.0	93.41
SVM-SGD: Acc. Test	93.34	93.95	93.74	93.26
SVM-SGD: Tiempo	0.0004	0.0019	0.0140	0.2325
LIBSVM (Lineal): Acc. Train	94.0	94.4	94.24	94.32
LIBSVM (Lineal): Acc. Test	90.99	93.98	94.18	94.19
LIBSVM (Lineal): Tiempo	0.0004	0.0183	0.2370	548.17
LIBSVM (RBF): Acc. Train	100.0	99.8	98.93	98.76
LIBSVM (RBF): Acc. Test	94.51	97.87	98.34	98.54
LIBSVM (RBF): Tiempo	0.0025	0.0051	0.1487	252.74

Problema no lineal	10^2	10^3	10^4	10^5
SVM-SGD: Acc. Train	50.0	50.3	50.12	50.27
SVM-SGD: Acc. Test	49.7	50.3	49.7	50.3
SVM-SGD: Tiempo	0.0004	0.0019	0.0079	0.1151
LIBSVM (RBF): Acc. Train	94.0	92.6	96.05	98.21
LIBSVM (RBF): Acc. Test	62.02	81.59	92.59	97.43
LIBSVM (RBF): Tiempo	0.0013	0.0456	0.3140	271.71

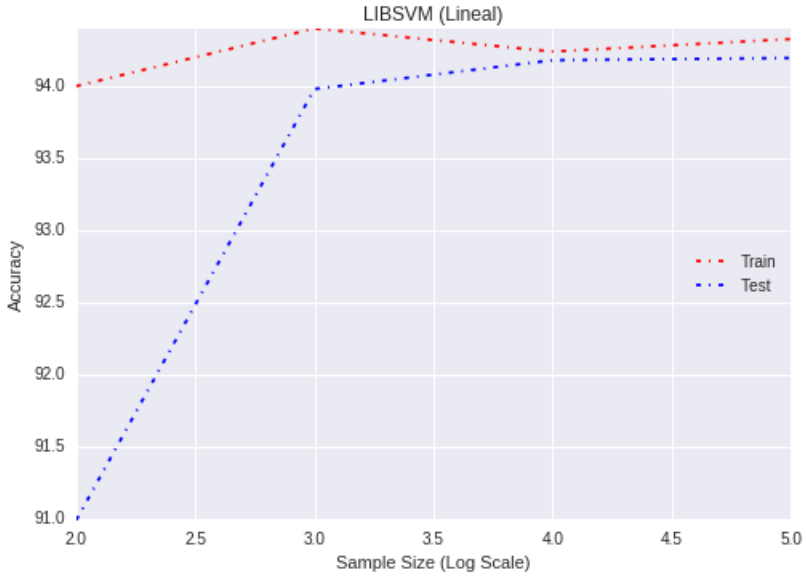

Figura 5.3: Problema linealmente separable: LIBSVM con Kernel Lineal

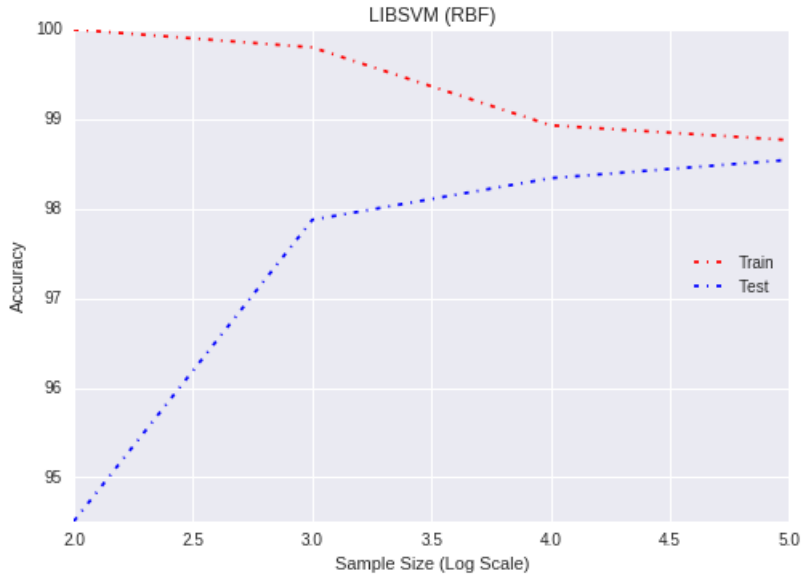

Figura 5.4: Problema linealmente separable: LIBSVM con Kernel RBF

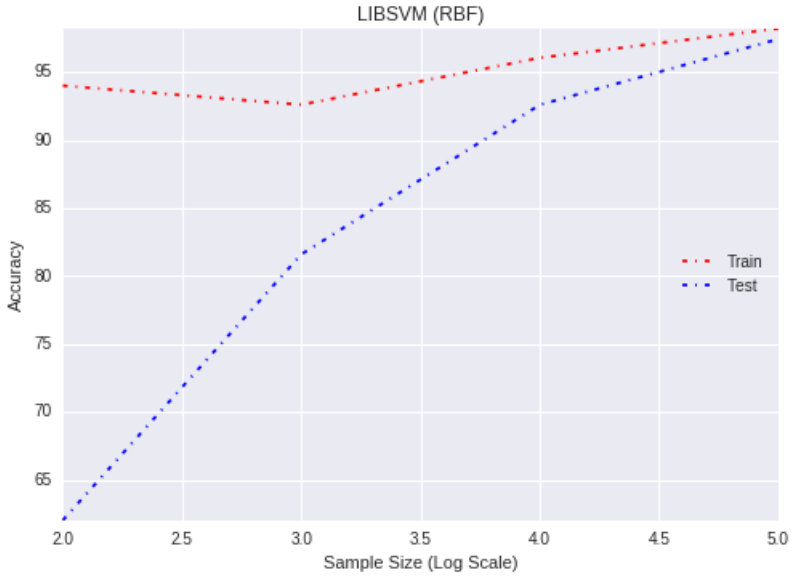

Figura 5.5: Problema no linealmente separable: LIBSVM con Kernel RBF

Los resultados muestran que cuando la estructura del problema es lineal, **SVM-SGD** y **LIBSVM** con Kernel lineal son capaces de obtener resultados similares, aunque **SVM-SGD** es mucho más eficiente computacionalmente. Por otra parte, **LIBSVM** con Kernel RBF es sustancialmente más rápido que **LIBSVM** con Kernel lineal, y con mejores resultados de predicción gracias a su mayor flexibilidad. Más aún, los resultados también muestran que cuando la estructura del problema no es lineal, **SVM-SGD** no es competitivo en predicción, mientras que **LIBSVM** con Kernel RBF aumenta su capacidad de predicción conforme el tamaño del conjunto de datos de entrenamiento crece.

5.1.2. Número de variables

En este experimento se varía el número de variables para el problema linealmente separable. Se consideran los casos $n\_vars = 10, 30, 50$. Los resultados en la siguiente tabla muestran la precisión en el conjunto de datos de entrenamiento, la precisión en el conjunto de datos de prueba y el tiempo de entrenamiento, para el caso $n = 10^5$. Por otra parte, las figuras 5.6, 5.7 y 5.8 ilustran el comportamiento del tiempo de entrenamiento para **SVM-SGD**, **LIBSVM** con Kernel Lineal y **LIBSVM** con Kernel RBF, respectivamente.

Problema lineal ($n = 10^5$)	10	30	50
SVM-SGD: Acc. Train	93.41	92.62	93.32
SVM-SGD: Acc. Test	93.26	92.43	93.81
SVM-SGD: Tiempo	0.2179	0.2661	0.3126
LIBSVM (Lineal): Acc. Train	94.32	94.32	94.32
LIBSVM (Lineal): Acc. Test	94.19	94.19	94.19
LIBSVM (Lineal): Tiempo	539.85	525.44	546.86
LIBSVM (RBF): Acc. Train	98.76	98.76	98.76
LIBSVM (RBF): Acc. Test	98.54	98.54	98.54
LIBSVM (RBF): Tiempo	258.40	248.30	253.72

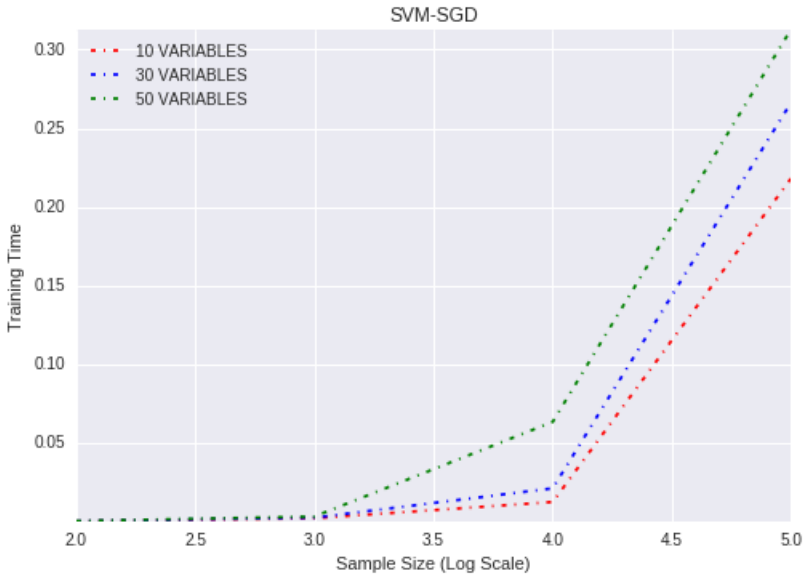

Figura 5.6: Problema linealmente separable con SVM-SGD variando el número de variables.

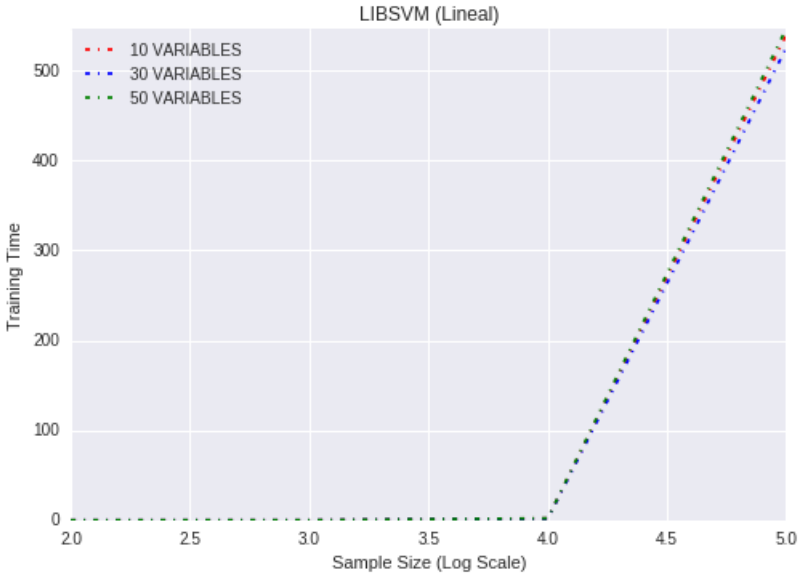

Figura 5.7: Problema linealmente separable con LIBSVM y Kernel Lineal variando el número de variables.

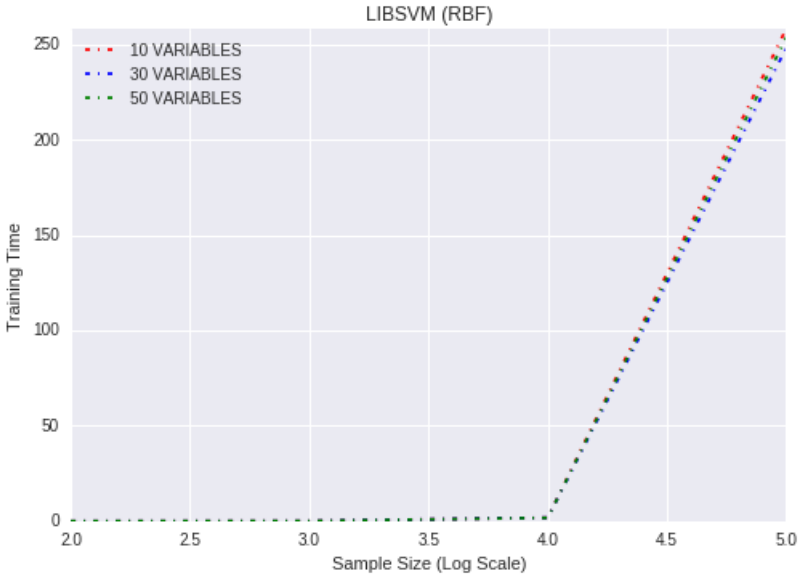

Figura 5.8: Problema linealmente separable con LIBSVM y Kernel RBF variando el número de variables.

De los resultados se concluye que en **SVM-SGD** aumenta el tiempo de entrenamiento con el número de variables pero no sustancialmente. Por otra parte, el tiempo de entrenamiento en **LIBSVM** no es muy dependiente del número de variables. Más aún, el tiempo de entrenamiento en **LIBSVM** con Kernel lineal es sustancialmente superior que con el Kernel RBF.

5.1.3. SVM-SGD: Número de *períodos*

En este experimento se varía el número de *períodos* para el problema linealmente separable. Se consideran los casos $n\_epochs = 10, 20, 30$. Los resultados en la siguiente tabla muestran la precisión en el conjunto de datos de entrenamiento, la precisión en el conjunto de datos de prueba y el tiempo de entrenamiento, para el caso $n = 10^5$. Por otra parte, las figura 5.9 ilustra el comportamiento del tiempo de entrenamiento.

Problema lineal ($n = 10^5$)	10	20	30
SVM-SGD: Acc. Train	93.41	93.92	93.89
SVM-SGD: Acc. Test	93.26	93.95	93.94
SVM-SGD: Tiempo	0.2438	0.3204	0.4390

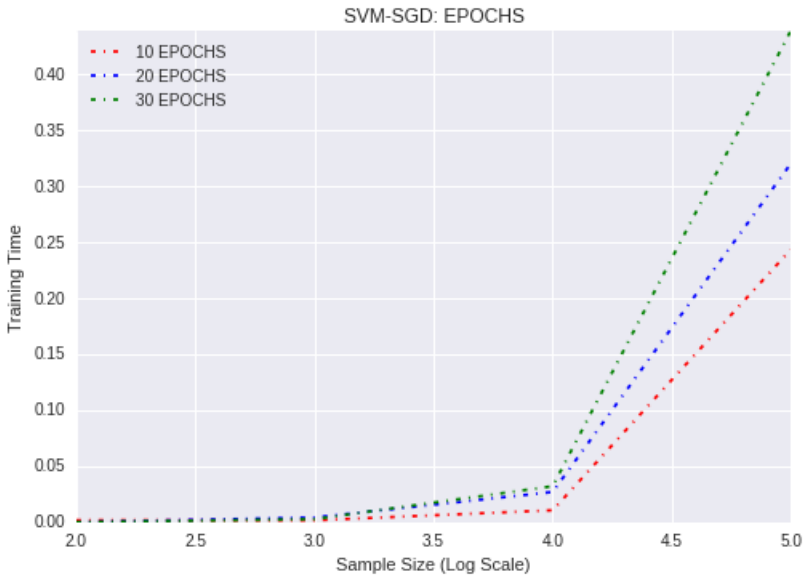

Figura 5.9: Problema linealmente separable con SVM-SGD variando el número de *períodos*.

De los resultados se concluye que la diferencia en tiempos de entrenamiento al aumentar el número de *períodos* es más grande conforme el número de datos aumenta. Sin embargo, el tiempo de entrenamiento no crece sustancialmente.

5.1.4. LIBSVM: Cantidad de caché

En este experimento se varía la cantidad de caché para el problema no linealmente separable. Se consideran los casos $m = 50, 100, 200$. Los resultados en la siguiente tabla muestran la precisión en el conjunto de datos de entrenamiento, la precisión en el conjunto de datos de prueba y el tiempo de entrenamiento, para el caso $n = 10^5$. Por otra parte, las figura 5.10 ilustra el comportamiento del tiempo de entrenamiento.

Problema no lineal ($n = 10^5$)	50	100	200
LIBSVM (RBF): Acc.Train	98.19	98.30	98.30
LIBSVM (RBF): Acc. Test	97.14	97.50	97.32
LIBSVM (RBF): Tiempo	293.22	304.85	263.71

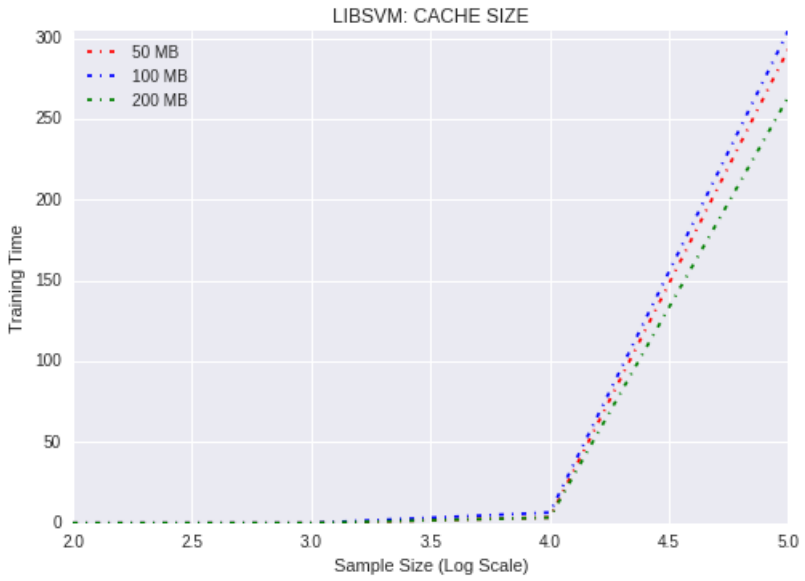

Figura 5.10: Problema no linealmente separable con LIBSVM y Kernel RBF variando la cantidad de caché.

Los resultados de este experimento no sugieren un patrón claro. Es probable que los resultados de la optimización no sean dependientes de la cantidad de caché cuando se utiliza al menos cierta cantidad suficiente.

5.1.5. LIBSVM: *Reducción*

En este experimento se varía la estrategia de *reducción*. Se consideran los casos en donde sí se utiliza y en donde no se utiliza dicha estrategia. Los resultados en la siguiente tabla muestran la precisión en el conjunto de datos de entrenamiento, la precisión en el conjunto de datos de prueba y el tiempo de entrenamiento, para el caso $n = 10^5$, para el caso del problea linealmente separable con Kernel Lineal y el problema no separable con Kernel RBF, respectivamente Por otra parte, las figuras 5.11 y 5.12 ilustran el comportamiento del tiempo de entrenamiento, de ambos casos, respectivamente.

Problema lineal ($n = 10^5$)	Con *reducción*	Sin *reducción*
LIBSVM (Lineal): Acc. Train	94.32	94.32
LIBSVM (Lineal): Acc. Test	94.19	94.19
LIBSVM (Lineal): Tiempo	555.16	1268.01

Problema no lineal ($n = 10^5$)	Con *reducción*	Sin *reducción*
LIBSVM (RBF): Acc. Train	98.21	98.22
LIBSVM (RBF): Acc. Test	97.30	97.15
LIBSVM (RBF): Tiempo	282.44	351.30

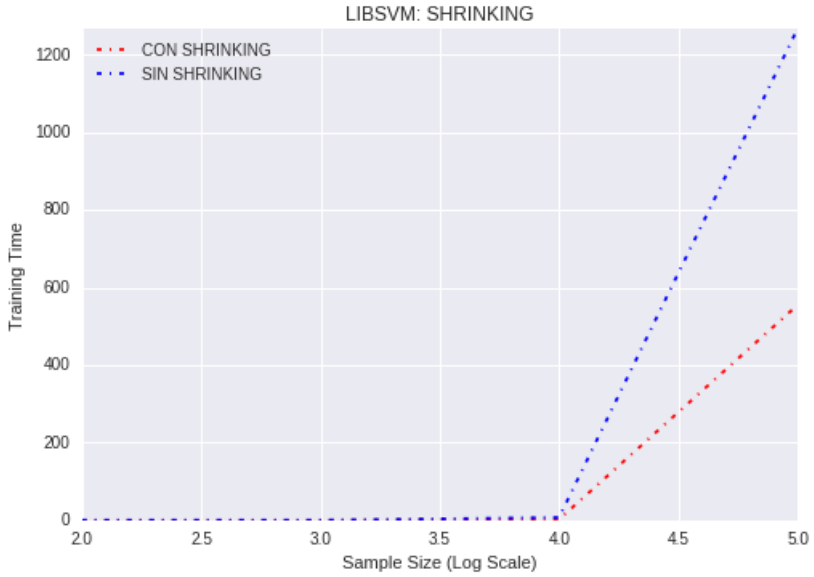

Figura 5.11: Problema linealmente separable con LIBSVM y Kernel RBF con y sin utilizar la estrategia de *reducción*.

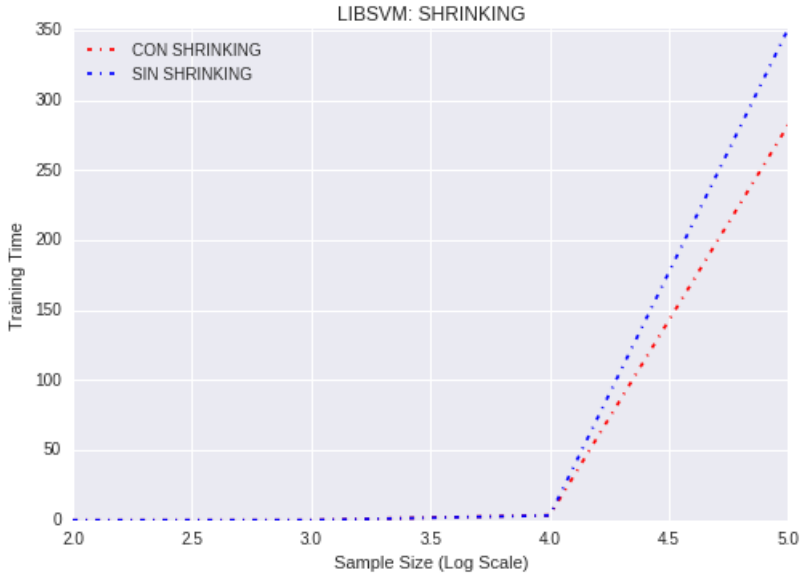

Figura 5.12: Problema no linealmente separable con LIBSVM y Kernel RBF con y sin utilizar la estrategia de *reducción*.

De los resultados se concluye que la reducción en el tiempo de entrenamiento al utilizar la estrategia de *reducción* se vuelve más grande conforme el número de datos aumenta. Por otra parte la diferencia en tiempos de entrenamiento al utilizar y no utilizar la estrategia es más pronunciada cuando se utiliza el Kernel lineal.

5.2. Problemas reales

Los conjuntos de datos utilizados para los experimentos se obtuvieron a partir del sitio oficial de la biblioteca **LIBSVM**: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` En todos los casos se tienen únicamente dos clases a predecir y se utilizó el 20 % de las observaciones como conjunto de prueba, elegido de forma seudoaleatoria a partir del módulo **random.RandomState** de la biblioteca **numpy** de **Python**. La siguiente tabla contiene información general sobre los conjuntos de datos.

Dataset	Referencia	Núm. observaciones	Núm. Variables
ijcnn1	[16]	49990	22
cod-rna	[17]	59535	8
a9a	[18]	32561	123
w8a	[18]	49749	300

Los resultados obtenidos en cada uno de estos conjuntos de datos, al comparar **SVM-SGD** y **LIBSVM** con Kernel RBF se presentan en la siguiente tabla.

	SVM-SGD	SVM-SGD	SVM-SGD	LIBSVM	LIBSVM	LIBSVM
Dataset	Acc. Train	Acc. Test	Tiempo	Acc. Train	Acc. Test	Tiempo
ijcnn1	90.24	90.46	0.06	94.11	94.19	29.32
cod-rna	83.47	83.42	0.05	89.79	89.09	145.31
a9a	76.05	75.35	0.03	86.72	83.89	57.78
w8a	97.05	96.90	0.05	99.06	98.71	68.65

A partir de estos resultados, la comparación de la precisión y de los tiempos de entrenamiento de ambos métodos a través de los ratios

$$\frac{\text{Acc. Train SVM-SGD}}{\text{Acc. Train LIBSVM}} \tag{5.1}$$

$$\frac{\text{Acc. Test SVM-SGD}}{\text{Acc. Test LIBSVM}} \tag{5.2}$$

$$\frac{\text{Tiempo SVM-SGD}}{\text{Tiempo LIBSVM}} \tag{5.3}$$

se muestra en la siguiente tabla.

Dataset	Ratio Acc. Train	Ratio Acc. Test	Ratio Tiempo
ijcnn1	.9588	.9603	.0020
cod-rna	.9296	.9363	.0003
a9a	.8769	.8982	.0005
w8a	.9797	.9816	.0007

Los resultados obtenidos muestran que en general, **LIBSVM** con Kernel RBF es superior en términos de predicción. Sin embargo, en la mayoría de los casos, **SVM-SGD** es capaz de obtener resultados competitivos con un tiempo de entrenamiento mucho menor.

Capítulo 6

Conclusiones

A continuación se presentan algunas conclusiones que se desprenden de este trabajo.

- El método **SVM-SGD** es muy eficiente computacionalmente. Sin embargo, está limitado a lograr una buena capacidad de predicción cuando la estructura del problema es lineal.
- Para problemas con estructura lineal, **SVM-SGD** es muy superior respecto a **LIBSVM** en cuanto a costo computacional, logrando resultados similares en predicción.
- Los resultados en los problemas simulados sugieren que **LIBSVM** es dependiente de la cantidad de memoria disponible, mientras que utiizar **SVM-SGD** puede utilizarse aún teniendo memoria limitada.
- Los resultados en los problemas reales sugieren que utilizar **LIBSVM** con un Kernel no lineal lleva a mejores resultados en predicción, con el costo de un mucho mayor tiempo de entrenamiento.
- Los resultados, en general, sugieren que para grandes volúmenes de datos es preferible utilizar **SVM-SGD** si el problema ha sido preprocesado y dotado de estructura lineal. En otro caso, si se privilegia la capacidad de predicción y se cuenta con la memoria suficiente para el volumen de datos, es preferible utilizar **LIBSVM** con un Kernel no lineal.

- Ambas implementaciones, **SVM-SGD** y **LIBSVM** no son paralelizables *per se* ya que los algoritmos de optimización que utilizan son secuenciales. Sin embargo, en la práctica deben elegirse los parámetros de ambos métodos de manera cuidadosa para lograr buenos resultados de predicción. La búsqueda de estos parámetros, en ambos casos, puede ser realizada en paralelo.

- Adicionalmente, pueden desarrollarse implementaciones del método MSV con el Kernel lineal y con Kernels no lineales, basadas en métodos de puntos interiores, que son altamente paralelizables. Ejemplos de este enfoque pueden ser consultados en [21] y en [22].

Bibliografía

[1] Bishop, C. (2010), *Pattern Recognition and Machine Learning*. Springer.

[2] Hastie, T. *et al.* (2008), *The elements of Statistical Learning*. Springer.

[3] Vapnik, Vladimir and Cortes, Corinna (1995), *Support Vector Networks*. Machine Learning, 20.

[4] Vapnik, Vladimir (1982), *Estimation of Dependences Based on Empirical Data*. Springer Series in Statistics.

[5] Joachims, Thorsten (1999), *Making Large Scale SVM Learning Practical*. Advances in Kernel Methods. MIT Press.

[6] Chen, Pai-Hsuen *et al.* (2006), *A Study on SMO-type Decomposition Methods For Support Vector Machines*. IEEE Transactions on Neural Networks.

[7] Steinwart, Ingo and Scovel, Clint (2004), *Fast rates for support vector machines using gaussian kernels*. Technical Report.

[8] Karatzoglou, A. *et al.* (2013), *kernlab: An S4 Package for Kernel Methods in R*. The Comprehensive R Archive Network.

[9] Shiloh, R. (2007), *Support Vector Machines for Classification and Regression*. M.Sc. Thesis, McGill University.

[10] Rifkin, R. (2002), *Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning*. Ph. D. Thesis, MIT.

[11] Bottou, Leon and Lin, Chih-Jen (2007), *Support Vector Machines Solvers*. MIT Press.

[12] Bottou, Leon (2010), *Large Scale Machine Learning With Stochastic Gradient Descent*. Documentación y software disponible en `http://leon.bottou.org/projects/sgd`

[13] Chang, Chih-Chung and Lin, Chih-Jen (2007), *LIBSVM: A library for Suppport Vector Machines*. Documentación y software disponible en `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[14] Shalev-Shwartz, Shai *et al.* (2013), *Pegasos: Primal Estimated Sub-Gradient Solver for SVM*. Proceedings of the 24th International Conference on Machine Learning.

[15] Krishna, Aditya, *Large Scale Support Vector Machines: Algorithms and Theory*.

[16] Prokhorov, Danil (2001), *Slide presentation in IJCNN 2001, Ford Research Laboratory* in IJCNN 2001 Neural Network Competition.

[17] Uzilov, A. *et al.* (2006), *Detection of Non-Coding RNAs on the Basis of Predicted Secondary Structure Formation Free Energy Change.*. BMC Bioinformatics.

[18] Platt, John (1998), *Fast Training of Support Vector Machines Using Sequential Minimal Optimization* in Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, Advances in Kernel Methods - Support Vector Learning. MIT Press.

[19] Mercer, J. (1909), *Functions of Positive and Negative Type and Their Connection With the Theory of Integral Equations*. Philosophical Transactions of the Royal Society.

[20] Aronszajn, N. (1950), *Theory of Reproducing Kernels*. Transactions of the American Mathematical Society.

[21] Gondzio, J. and Woodsend, K. (2009), *Exploiting Separability in Large-Scale Linear Support Vector Machine Training*. Technical Report. The University of Edinburgh.

[22] Gondzio, J. and Woodsend, K. (2007), *Parallel Support Vector Machine Training with Nonlinear Kernels*. Technical Report. The University of Edinburgh.