Trabajo Práctico Final Integrador

41409 – Sistemas Distribuidos y Programación Paralela
11088 – Bases de Datos Masivas

Francisco Altoe
franciscoda@outlook.com
Legajo: 131776

11 de septiembre de 2017

Resumen

En este trabajo final se tratan los fundamentos teóricos y matemáticos de las máquinas de vectores de soporte, un conocido modelo de aprendizaje supervisado. Se presenta una implementación paralelizada para resolver el problema de optimización junto a una implementación secuencial para comparar el rendimiento respecto al tiempo de entrenamiento. El conjunto de datos utilizado para entrenar y probar es la base de datos MNIST, sobre la cual se alcanza una precisión aceptable en la clasificación.

Índice

1. Introducción

La propuesta de proyecto final consiste en desarrollar una implementación paralelizable de un algoritmo de entrenamiento para un clasificador SVM. Las SVMs (Support Vector Machines) son modelos de aprendizaje supervisado que se utilizan para problemas de clasificación y regresión.

Los parámetros para un modelo basado en SVMs son: un vector $\vec{\omega}$ que define un hiperplano centrado en el origen, y un escalar b que se define a partir del "sesgo" del conjunto de datos. El clasificador es de tipo binario y lineal, es decir:

- La clasificación *binaria*, significa que sólo se distinguen dos clases: positiva y negativa. Se les suele asignar los valores 1 y -1, respectivamente.
- La función $D(\vec{u}) = \vec{u} \cdot \vec{\omega} + b$ es una combinación lineal que permite conocer la distancia de un punto desonocido, $\vec{u}$, al límite de decisión.
- La clasificación resultante se define como una función de D: $sgn(D(\vec{u})) \in \{1, -1\}$

Se suelen distinguir dos tipos de entrenamiento para SVMs lineales según el problema que se busca resolver en cada caso:

- SVMs de margen rígido (*hard-margin*): se requiere que el conjunto de datos de entrenamiento sea linealmente separable para encontrar el hiperplano que separe ambas clases con el mayor margen posible. Veremos que el problema de maximizar el margen es equivalente a minimizar $\|\vec{\omega}\|$.
- SVMs de margen flexible (*soft-margin*): admite que el conjunto de datos de entrenamiento sea linealmente inseparable. Para esto se introduce una función de pérdida que cuantifica la distancia entre la prediccion del modelo y la verdadera clase de la observación. El problema de optimización a resolver para este caso será el de minimizar el *riesgo empírico* de los parámetros del modelo. La función de pérdida que se utiliza en SVMs es la función de pérdida de articulación (*hinge-loss*)

Para resolver el problema de optimización (independientemente del tipo de SVM) se suele utilizar el método de multiplicadores de Lagrange, lo que permite relajar las restricciones que sujetan el problema de optimización original. También veremos que, aplicando el método, podemos obtener dos problemas: uno llamado *primal* y otro *dual*. Podremos entrenar un clasificador resolviendo cualquiera de los tres problemas, sin embargo, la mayoría de los algoritmos buscan resolver el dual.

Un SVM de margen flexible no es el unico enfoque que se puede tomar para entrenar con datos que no son linealmente separables. Es posible definir una funcion $\Phi(\vec{x}_i)$ de modo que la función Φ transforme un vector $\vec{x}_i$ a un espacio de mayor dimensionalidad, posiblemente infinita, donde los datos sean separables. Sin embargo, el clasificador no será lineal en el espacio original. El uso de un clasificador no lineal no excluye la posibilidad de utilizar una SVM de margen flexible.

2. Descripción

Dado un conjunto de datos de entrenamiento definido de la siguiente manera:

$$S = \{(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)\} \tag{1}$$

Donde:

- $\vec{x}_i = \vec{x}_{i1}, \vec{x}_{i2}, ..., \vec{x}_d$ es un vector con los valores de las variables predictoras para esa observación
- y_i es la clase de esa observación. Dado que el clasificador es binario, se puede considerar que $y \in \{-1, 1\}$

Se busca obtener un hiperplano definido según la siguiente igualdad:

$$\vec{\omega} \cdot \vec{x} + b = 0 \tag{2}$$

Donde:

- $\vec{\omega}$ es un vector normal al hiperplano, que no necesariamente debe ser unitario
- $\vec{x}$ son los puntos del hiperplano que cumplen la igualdad
- b es un escalar que define el "sesgo" (bias) del conjunto de datos

En base a estos parámetros se define la función de decisión para un vector deconocido, $\vec{u}$ como una combinación lineal:

$$D(\vec{u}) = \vec{\omega} \cdot \vec{u} + b \tag{3}$$

En el caso de una SVM de margen rígido, se restringe a que el hiperlplano debe separar todas las observaciones de los datos de entrenamiento. Se puede formular esto de la siguiente forma:

$$\vec{x}_i \cdot \vec{\omega} + b > 0 \quad \forall i : y_i = 1 \tag{4a}$$
$$\vec{x}_i \cdot \vec{\omega} + b < 0 \quad \forall i : y_i = -1 \tag{4b}$$

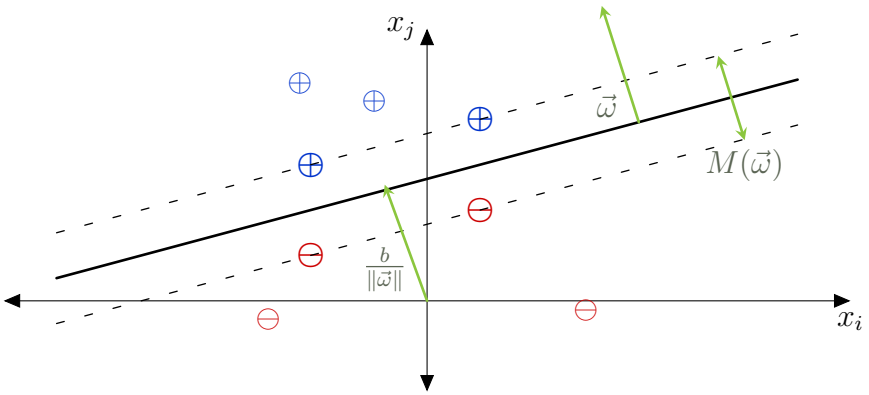


Figura 1: Solución con margen óptimo de un problema simple para un discriminador lineal.

Para que existan los parámetros $\vec{\omega}, b$ que definan el hiperplano de separación, se requiere que los datos de entrenamiento sean linealmente separables. Sin embargo, existe un número infinito de parámetros $\vec{\omega}, b$ que definen un hiperplano bajo tales circunstancias. Por lo tanto, se agrega la siguiente restricción:

$$\vec{\omega} \cdot \vec{x_i} + b \geq 1 \qquad \forall i : y_i = 1 \tag{5a}$$
$$\vec{\omega} \cdot \vec{x_i} + b \leq -1 \qquad \forall i : y_i = -1 \tag{5b}$$

Existirán puntos que *activen* estas restricciones, es decir, existirán observaciones en cada clase que cumplan la igualdad:

$$\exists \vec{x}_+ \in S : \quad \vec{\omega} \cdot \vec{x}_+ + b = 1 \tag{6a}$$
$$\exists \vec{x}_- \in S : \quad \vec{\omega} \cdot \vec{x}_- + b = -1 \tag{6b}$$

Estos puntos definen dos hiperplanos adicionales, equidistantes al original, formando un "margen" de separacion entre las observaciones de ambas clases. A las observaciones que activan las restricciones (y por lo tanto, yacen sobre estos hiperplanos) las llamaremos *vectores de soporte*. Siempre se podrá encontrar uno o más vectores de soporte por clase (Witten, Elbe, y Hall, 2011).

La mejor solución al problema será la que presenta la mayor distancia entre ambos márgenes puesto que se presume (aunque no se garantiza) que esta solución generaliza mejor (Boser, Guyon, y Vapnik, 1992). Por lo tanto, al hiperplano resultante también se lo conoce como *hiperplano de margen máximo* y al clasificador se lo puede llamar *clasificador de margen óptimo*

(Boser y cols., 1992). Esta característica es la principal diferencia entre un clasificador SVM y un perceptrón.

La distancia entre ambos márgenes se puede calcular a partir de la proyección de la distancia entre un par de vectores de soporte de cada clase sobre un vector unitario normal al hiperplano. Es decir, la función objetivo del problema de optimización será:

$$\begin{aligned} M &= (\vec{x}_+ - \vec{x}_-) \cdot \frac{\vec{\omega}}{\|\vec{\omega}\|} \\ &= \frac{\vec{\omega} \cdot \vec{x}_+ - \vec{\omega} \cdot \vec{x}_-}{\|\vec{\omega}\|} \\ &= \frac{(1-b) - (-1-b)}{\|\vec{\omega}\|} \\ &= \frac{2}{\|\vec{\omega}\|} \end{aligned} \tag{7}$$

Donde:

- x_+ es un vector de soporte positivo.
- x_- es un vector de soporte negativo.

Más aún, las restricciones en (5) se pueden generalizar para todo i multiplicando ambos lados de las desigualdades por y_i de la siguiente manera:

$$\begin{array}{rlrl} \vec{\omega} \cdot \vec{x_i} + b & \geq 1 & \vec{\omega} \cdot \vec{x_i} + b & \leq -1 \\ y_i(\vec{\omega} \cdot \vec{x_i} + b) & \geq y_i & y_i(\vec{\omega} \cdot \vec{x_i} + b) & \leq -y_i \\ y_i(\vec{\omega} \cdot \vec{x_i} + b) & \geq 1 & y_i(\vec{\omega} \cdot \vec{x_i} + b) & \geq -(-1) \end{array}$$

$$y_i(\vec{\omega} \cdot \vec{x_i} + b) \geq 1 \quad \forall i \tag{8}$$

Finalmente se obtiene el problema de optimización de maximizar M según (7) sujeto a las restricciones expresadas en (8):

$$\begin{aligned} &\text{maximize} && \frac{2}{\|\vec{\omega}\|} \\ &\text{subject to} && y_i(\vec{\omega} \cdot \vec{x}_i + b) \geq 1 \quad i = 1, \ldots, n \end{aligned} \tag{9}$$

Sin embargo, se suele optar por resolver el siguiente problema en su lugar:

$$\begin{aligned} &\text{minimize} && \|\vec{\omega}\| \\ &\text{subject to} && y_i(\vec{\omega} \cdot \vec{x}_i + b) \geq 1 \quad i = 1, \ldots, n \end{aligned} \tag{10}$$

Está demostrado (Boser y cols., 1992) que el problema (10) es equivalente al problema presentado en (9).

3. Condiciones de Karush-Kuhn-Tucker

Para facilitar la tarea de resolver un problema de optimización sujeto a restricciones de igualdad se suele utilizar el método de multiplicadores de Lagrange para relajar estas restricciones. Sin embargo, el problema (10) tiene restricciones de desigualdad, por lo que aplicaremos las condiciones de Karush-Kuhn-Tucker (KKT) que suponen una generalización del método de Lagrange y permiten tales restricciones.

Dado un problema de optimización en forma estándar (Boyd y Vandenberghe, 2004):

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_i(x) \leq 0 \quad i = 1, \ldots, m, \\ &&& h_i(x) = 0 \quad i = 1, \ldots, p \end{aligned}$$

Suponiendo que existe un conjunto de parámetros x^* que representan un óptimo local de la funcion objetivo, $f(x^*)$, y que x^* cumple con todas las restricciones $g_i(x^*) \leq 0$ y $h_i(x^*) = 0$, entonces se puede aseverar que existe un conjunto de constantes $\mu_i (i = 1, \ldots, m)$ y $\lambda_i (i = 1, \ldots, p)$ que necesariamente hacen cumplir las siguientes condiciones (Boyd y Vandenberghe, 2004):

- Estacionariedad: Los gradientes de la función objetivo y de las restricciones tendrán la misma dirección, mientras que sus magnitudes se deben igualar con los multiplicadores μ_i, λ_i. En el caso de que el problema sea de minimización, el sentido de las gradientes será opuesto.

$$\nabla f(x^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) \quad \text{para maximizar}$$

$$-\nabla f(x^*) = \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) \quad \text{para minimizar}$$

- Feasibilidad primal: La solución cumple con las restricciones originales.

$$\begin{aligned} g_i(x^*) &\leq 0 \quad i = 1, \ldots, m \\ h_i(x^*) &= 0 \quad i = 1, \ldots, p \end{aligned}$$

- Feasibilidad dual:

$$\mu_i \geq 0 \quad i = 1, \ldots, m$$

- Holgura complementaria: μ_i es no nulo si y sólo si $g_i(x)$ está activa.

$$\mu_i g_i(x^*) = 0 \quad i = 1, \ldots, m$$

Es posible relajar las restricciones del problema original. Reformulando la función objetivo, agregándole a f una suma ponderada de las restricciones, se obtiene una nuevo problema a optimizar, sin restricciones. Los pesos de las restricciones serán los multiplicadores μ_i y λ_i según corresponda. A esta reformulación del problema se lo conoce como *lagrangeano*:

$$\text{minimize} \quad \mathcal{L}_P(x, \mu, \lambda) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x) + \sum_{i=1}^{p} \lambda_i h_i(x)$$

Luego se puede proceder igualando las derivadas parciales de $\mathcal{L}_P$ a 0 y resolviendo el sistema de ecuaciones resultante para encontrar x^*:

$$\nabla \mathcal{L}_P = \left(\frac{\partial \mathcal{L}_P}{\partial x}, \frac{\partial \mathcal{L}_P}{\partial \mu}, \frac{\partial \mathcal{L}_P}{\partial \lambda} \right) = 0$$

4. Problema Primal

Está claro que en el caso de la SVM la función objetivo es el margen M. Sin embargo, para facilitar la tarea de derivar M, se suele reescribir la función de la siguiente forma:

$$\begin{aligned} M(\vec{\omega}) &= \|\vec{\omega}\| \\ M'(\vec{\omega}) &= \frac{1}{2} \|\vec{\omega}\|^2 \end{aligned} \tag{11}$$

Las funciones de restricción de desigualdad también se deben adaptar a la forma normal de la siguiente manera:

$$\begin{aligned} y_i(\vec{\omega} \cdot x_i + b) &\geq 1 \\ y_i \left(\vec{\omega} \cdot x_i + b\right) - 1 &\geq 0 \end{aligned} \tag{12}$$

Se obtiene el problema de optimización en forma normal según lo mostrado en las ecuaciones (11) y (12):

$$\begin{aligned} \text{minimize} \quad & M(\vec{\omega}) = \frac{1}{2} \|\vec{\omega}\|^2 \\ \text{subject to} \quad & y_i \left(\vec{\omega} \cdot x_i + b\right) - 1 \geq 0 \quad i = 1, \ldots, n \end{aligned} \tag{13}$$

Se procede aplicando el lagrangeano al problema. La convención en la literatura de SVM es que los multiplicadores asociados a las restricciones se llamen α_i:

$$\text{minimize} \quad \mathcal{L}_P(\vec{\omega}, b, \alpha) = \frac{1}{2} \|\vec{\omega}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1] \tag{14}$$

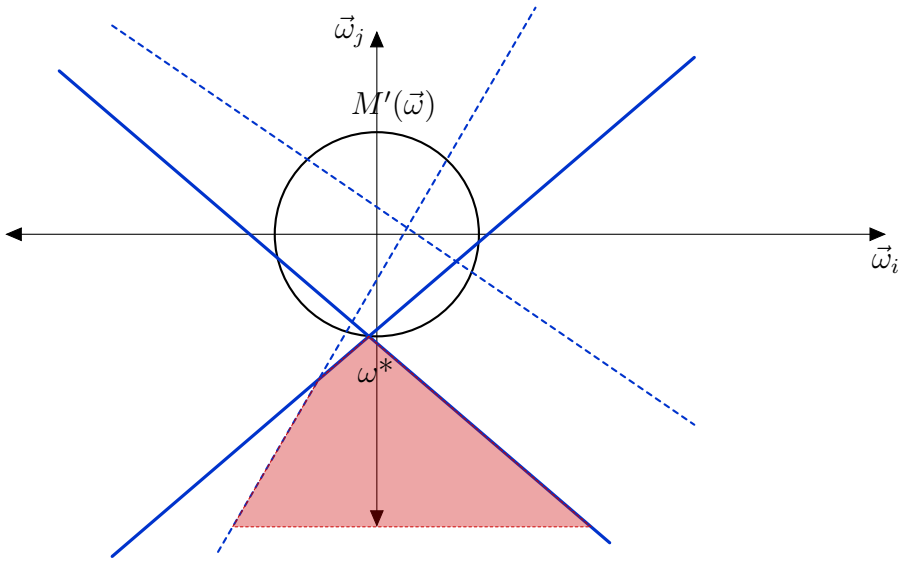


Figura 2: Vista de las curvas de nivel del problema junto a las restricciones para un b fijo. En rojo, la región feasible. Las rectas de las restricciones de los vectores de soporte están resaltadas.

Las condiciones necesarias de KKT aplicadas a este problema son las siguientes (Burges, 1998):

- Estacionariedad:

$$\begin{aligned} \frac{\partial \mathcal{L}_P}{\partial \vec{\omega}} &= \vec{\omega} - \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i = 0 \quad &\Longrightarrow \quad \vec{\omega} &= \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \\ \frac{\partial \mathcal{L}_P}{\partial b} &= -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad &\Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i &= 0 \end{aligned} \tag{15}$$

- Feasibilidad primal:

$$y_i(\vec{x}_i \cdot \vec{\omega} + b) - 1 \geq 0 \quad i = 1, \ldots, n \tag{16}$$

- Feasibilidad dual:

$$\alpha_i \geq 0 \quad i = 1, \ldots, n \tag{17}$$

- Holgura complementaria:

$$\alpha_i(y_i(\vec{x}_i \cdot \vec{\omega} + b) - 1) = 0 \quad i = 1, \ldots, n \tag{18}$$

Como se puede ver, el problema primal ya no estará sujeto a las restricciones de igualdad del problema original, pero estará influenciado por una serie de multiplicadores α_i. La condición de holgura complementaria implica que $\alpha_i = 0$ para los casos en que se cumple $y_i(\vec{x}_i \cdot \vec{\omega} + b) > 1$. Esto quiere decir que los vectores que no sean de soporte tendrán asociados, necesariamente, multiplicadores nulos. En el caso de los vectores de soporte, la unica condición que restringe a los multiplicadores es la de feasibilidad dual.

Es posible adaptar el problema primal a un SVM de margen flexible con kernels de transformación y minimizar ese problema con métodos como el de Newton (Chapelle, 2007). Sin embargo, generalmente se opta por resolver el problema dual en su lugar, para el que existen algoritmos específicos y que gozan de mayor popularidad.

5. Problema Dual

El problema dual que se suele resolver para la SVM es el dual de Wolfe: dado un problema de optimización con restricciones de desigualdad en forma normal:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \end{aligned}$$

Podemos formular un problema alternativo, que consiste en maximizar una función que tendrá un óptimo en el mismo punto que el problema original. El dual de Wolfe se obtiene de aplicar las condiciones de KKT y emplearlas como restricciones:

$$\begin{aligned} \underset{x,\, u}{\text{maximize}} \quad & f(x) + \sum_{i=1}^{m} u_i g_i(x) \\ \text{subject to} \quad & \nabla f(x) + \sum_{i=1}^{m} u_i \nabla g_i(x) = 0, \\ & u_i \geq 0, i = 1, \ldots, m \end{aligned}$$

En el caso de la SVM, el dual se obtiene de reemplazar con las condiciones

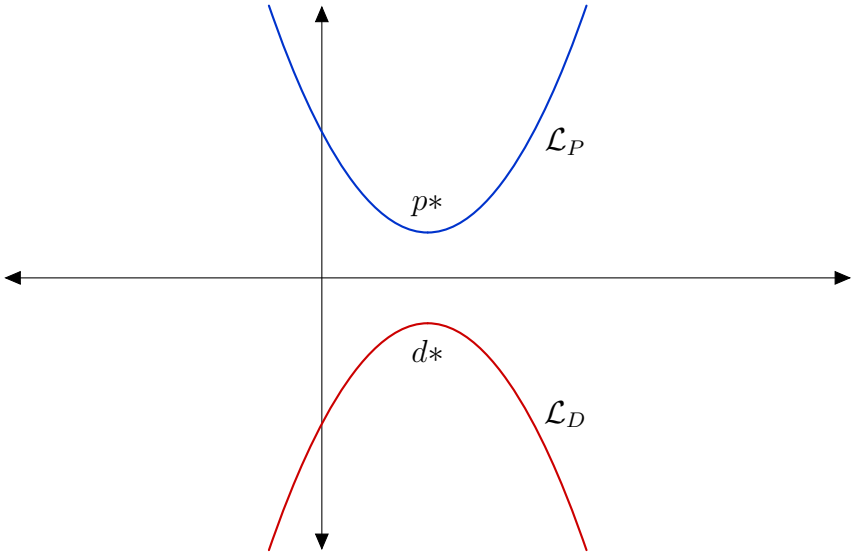


Figura 3: Para problemas de optimización convexa, el problema dual es cóncavo. También es posible que las soluciones de cada problema sean distintas. De serlo, se conoce a la diferencia $p^* - d^*$ como brecha de dualidad.

de KKT en el problema primal (14):

$$\begin{aligned} \text{maximize} \quad & \mathcal{L}_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \\ & \alpha_i \geq 0 \quad i = 1, \ldots, n \end{aligned} \tag{19}$$

El dual sólo requiere optimizar para los multiplicadores α_i. Sin embargo, no es necesario recuperar el parámetro $\vec{\omega}$. Basta con ajustar la función de clasificación a la igualdad dual hallada para $\vec{\omega}$. Nótese que sólo los vectores de soporte contribuyen al valor de $\vec{\omega}$, el resto de los vectores tienen asociado un multiplicador α_i igual a 0 por la condición de holgura complementaria.

$$D(\vec{u}) = \vec{\omega} \cdot \vec{u} + b = (\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \cdot \vec{u}) + b \tag{20}$$

Es posible recuperar el valor de b si se encuentra un vector de soporte de cualquier clase, $\vec{x}_\pm$, puesto que se conoce de antemano su valor en la función

de clasificación como se mostro en (6):

$$\begin{aligned} y_i(\vec{\omega} \cdot \vec{x}_{\pm} + b) &= 1 \\ \vec{\omega} \cdot \vec{x}_{\pm} + b &= \frac{1}{y_i} = y_i \\ b &= y_i - \vec{\omega} \cdot \vec{x}_{\pm} \\ b &= y_i - (\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \cdot \vec{x}_{\pm}) \end{aligned} \tag{21}$$

Está demostrado que el problema del clasificador SVM en el primal es un problema de *optimización cuadrática convexa*. Mientras tanto, el problema dual será cóncavo por definición.

6. Clasificador no lineal

Suponiendo un conjunto de datos S que no es separable linealmente en el espacio original $\mathbb{R}^{\mathrm{d}}$, es posible que los puntos $x_i \in S$ sean transformados a un nuevo espacio euclideano de mayor dimensionalidad, $\mathcal{H}$, a través de una funcion de transformación Φ:

$$\begin{aligned} &\Phi : \mathbb{R}^{\mathrm{d}} \mapsto \mathcal{H} \\ &\phi : \mathbb{R}^{\mathrm{d}} \mapsto \mathbb{R} \\ &\Phi(\vec{x}) = \langle \phi_1(\vec{x}), \phi_2(\vec{x}), ... \rangle \end{aligned} \tag{22}$$

La dimensionalidad del espacio $\mathcal{H}$ que permite la separacion de S puede llegar a ser infinita. Esto implica que el costo de entrenar y clasificar podría ser impredeciblemente alto dependiento de la dimensionalidad del espacio $\mathcal{H}$ resultante. La complejidad del modelo también crecerá porque la cantidad de funciones ϕ_i está ligada a $\mathcal{H}$ y también puede llegar a ser infinita.

La solución a esta dificultad parte de la observación de que el entrenamiento del clasificador depende del producto entre pares de vectores, por lo que se puede definir una función de kernel, K, tal que:

$$\begin{aligned} &K : \mathbb{R}^{\mathrm{d}} \times \mathbb{R}^{\mathrm{d}} \mapsto \mathbb{R} \\ &K(\vec{x}_1, \vec{x}_2) = \Phi(\vec{x}_1) \cdot \Phi(\vec{x}_2) \end{aligned} \tag{23}$$

Definir y calcular K puede resultar mucho más sencillo que hacer lo propio para el producto vectorial en términos de Φ para los propósitos de este problema. Dado que K debe cumplir las mismas propiedades que el producto entre un par de vectores en un espacio euclideano, no se debería definir esta

función de manera arbitraria. En otras palabras, debe existir el par $\{\mathcal{H}, \Phi\}$ para poder decir que K equivale al producto en un espacio vectorial. Esto se conoce como *condición de Mercer*. De no cumplirse esta condición, es posible que la matriz hessiana del problema no esté definida en algunos puntos, o que el mismo problema sea infeasible. En ambos casos, el algoritmo de solución no convergerá.

Algunas de las funciones de kernel más populares son:

- Polinómica (homogénea): $K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y})^p$. Donde p es un hiperparámetro.
- Polinómica (no homogénea): $K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + 1)^p$. Donde p es un hiperparámetro.
- Gaussiana de base radial: $K(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|^2/2\sigma^2}$. Donde $\sigma > 0$ es un hiperparámetro.
- Sigmoide: $K(\vec{x}, \vec{y}) = tanh(\kappa\vec{x} \cdot \vec{y} + c)$. Donde $\kappa > 0$ y $c < 0$ son hiperparámetros.

Los kernels de transformación permiten aprender reglas de clasificación no lineales, sin embargo, también introducen cierta facilidad para el sobreajuste en función de los hiperparámetros de cada kernel.

Se debe tener en cuenta que la función de decisión se ve influenciada por la función Φ, puesto que se debe llevar $\vec{\omega}$ a $\mathcal{H}$:

$$D(\vec{u}) = \vec{\omega} \cdot \Phi(\vec{u}) + b \tag{24}$$

Sin embargo, también es necesario actualizar la igualdad para $\vec{\omega}$ hallada en (15) para tener en cuenta el nuevo espacio:

$$\vec{\omega} = \sum_{i=0}^{n} \alpha_i y_i \Phi(\vec{x}_i) \tag{25}$$

Como se puede ver, $\vec{\omega}$ ahora yace en el espacio $\mathcal{H}$, por lo que no siempre será posible calcularlo fácilmente, dependiendo de la elección de la función de kernel. Para evitar este problema en la función de decisión, se puede distribuir el producto con $\Phi(\vec{x})$ dentro del cálculo de $\vec{\omega}$ de modo que aparece un producto entre $\vec{\omega}$ y $\vec{u}$:

$$\begin{aligned} D(\vec{u}) &= \sum_{i=0}^{n} \alpha_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{u}) + b \\ &= \sum_{i=0}^{n} \alpha_i y_i K(\vec{x}_i, \vec{u}) + b \end{aligned} \tag{26}$$

7. Minimización del Riesgo Empírico

Suponiendo que se tiene una función de clasificacion $h(\vec{x})$ para un modelo de aprendizaje supervisado. Llamaremos a la función h *hipótesis* y diremos que $h : X \mapsto Y$, tal que los atributos son $\vec{x} \in X$ y las clases son $y \in Y$. Si bien se busca que $h(\vec{x}_i) = y_i$ produzca una clasificación correcta, es posible que no todas las predicciones sean certeras, puesto que h modela una distribución de probabilidad condicionada con variable aleatoria, $P(y|x)$, y produce una clasificación sin posibilidad de predecir el ruido en los datos.

Sea $L(\hat{y}, y)$ una función que permite medir la disimiliridad entre la predicción $\hat{y}$ y la clase y tal que $L : Y \times Y \mapsto \mathbb{R} \geq 0$. Diremos que L es una *función de pérdida*.

Para comparar distintas funciones de hipótesis a fin de seleccionar la que se intuye que tendrá el mejor rendimiento, se define el *riesgo* de una hipótesis h como la esperanza de la función de pérdida:

$$R(h) = E[L(h(\vec{x}), y)] = \int L(h(\vec{x}), y) dP(X, Y) \tag{27}$$

Claramente, podemos plantear un problema de optimización para encontrar la hipótesis que minimiza el riesgo:

$$\underset{h}{\text{minimize}} \quad R(h)$$

Sin embargo, no siempre se conoce la funcion de distribución de probabilidad conjunta $P(x, y)$ de los datos para calcular el riesgo. Dada esta dificultad, en su lugar se utiliza una aproximación al riesgo. Esta aproximación se conoce como *riesgo empirico*, y se calcula como la media de las pérdidas:

$$R_{emp}(h) = \frac{1}{n} \sum_{i=0}^{n} L(h(\vec{x}_i), y_i) \tag{28}$$

8. Clasificador de Margen Flexible

Volviendo al caso de las SVMs, es necesario relajar las restricciones que fueron halladas para el clasificador de margen rígido para permitir que ciertos puntos se clasifiquen incorrectamente si es que el dataset es linealmente inseparable. Para lograr esto se introducen variables de holgura para cada restricción:

$$y_i(\vec{\omega} \cdot \vec{x}_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

El valor de cada ξ_i será 0 en caso de que la observación haya sido correctamente clasificada, de otro modo, su valor crecerá linealmente con la magnitud el error de clasificación en la función de decisión. Definimos, así, la función de *pérdida de articulación*:

$$\xi_i = L(D(\vec{x}_i), y_i) = max(0, 1 - y_i D(\vec{x}_i)) \tag{29}$$

Respecto a la función a minimizar, resulta intuitivo minimizar la norma de $\vec{\omega}$ junto al riesgo empírico que introducen las variables de holgura ξ_i:

$$\frac{1}{2}\|\vec{\omega}\|^2 + \frac{1}{n}\sum_{i=0}^{n}\xi_i$$

La función sigue siendo convexa, por lo que las propiedades mencionadas anteriormente se mantienen. Sin embargo, es deseable poder controlar el compromiso entre buscar el margen más amplio y minimizar el error del clasificador. Se introduce el hiperparámetro C para tales efectos:

$$\frac{1}{2}\|\vec{\omega}\|^2 + C\sum_{i=0}^{n}\xi_i$$

A medida que C crece, el clasificador se asemejará más a uno de margen rígido. Por el contrario, a medida que C se acerca a 0, el riesgo empírico incide menos en la función. Se restringe al hiperparámetro tal que $C > 0$, puesto que un valor nulo o negativo permitiría que la norma de $\vec{\omega}$ llegue a 0, el cual no es un resultado que tenga interpretación en términos del problema.

El problema primal resultante para el clasificador de margen flexible es el siguiente:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\vec{\omega}\|^2 + C\sum_{i=0}^{n}\xi_i \\ \text{subject to} \quad & y_i(\vec{\omega}\cdot\vec{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, n, \\ & \xi_i \geq 0 \qquad\qquad i = 1, \ldots, n \end{aligned} \tag{30}$$

De manera análoga al clasificador de margen rígido, se procede calculando el lagrangeano:

$$\text{minimize} \quad \mathcal{L}_P = \frac{1}{2}\|\vec{\omega}\|^2 + C\sum_{i=0}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(\vec{\omega}\cdot\vec{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{n}\mu_i\xi_i$$

Las condiciones de Karush-Kuhn-Tucker se vuelven las siguientes:

- Estacionariedad:

$$\begin{aligned}
\frac{\partial \mathcal{L}_P}{\partial \vec{\omega}} &= \vec{\omega} - \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i = 0 \quad &\implies \vec{\omega} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \\
\frac{\partial \mathcal{L}_P}{\partial b} &= -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad &\implies \sum_{i=1}^{n} \alpha_i y_i = 0 \\
\frac{\partial \mathcal{L}_P}{\partial \xi_i} &= C - \alpha_i - \mu_i = 0
\end{aligned}$$

- Feasibilidad primal:

$$\begin{aligned}
y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1 + \xi_i &\geq 0 \\
\xi_i &\geq 0
\end{aligned}$$

- Feasibilidad dual:

$$\begin{aligned}
\alpha_i &\geq 0 \\
\mu_i &\geq 0
\end{aligned}$$

- Holgura complementaria:

$$\begin{aligned}
\alpha_i[y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1 + \xi_i] &= 0 \\
\mu_i \xi_i &= 0
\end{aligned}$$

Afortunadamente, la tercer condición de estacionariedad permite cancelar todos los nuevos términos en el dual de Wolfe, por lo que el problema dual del clasificador de margen flexible es muy similar al de margen rígido:

$$\begin{aligned}
\text{maximize} \quad & \mathcal{L}_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\vec{x}_i \cdot \vec{x}_j) \\
\text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \\
& 0 \leq \alpha_i \leq C \quad i = 1, \ldots, n
\end{aligned}$$

Como se puede observar, sólo se agregó la condición de que $\alpha_i \leq C$, esto se obtiene de las condiciones de esacionariedad y de feasibilidad dual:

$$\begin{aligned}
C - \alpha_i - \mu_i &= 0 \\
C &> 0 \\
\alpha_i &\geq 0 \\
\mu_i &\geq 0
\end{aligned} \quad \iff \quad \alpha_i \leq C$$

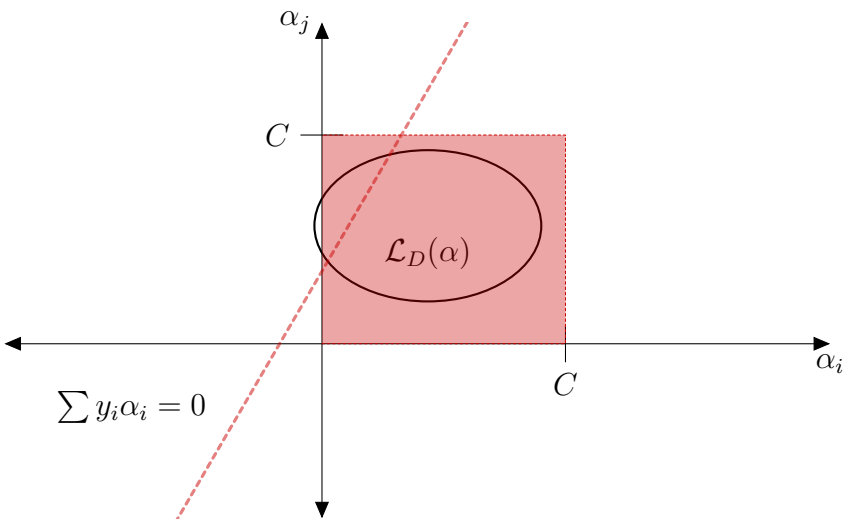


Figura 4: Vista de las curvas de nivel del problema dual para un par de variables α_i, α_j. La región feasible respecto a las restricciones de caja están sombreadas en rojo, mientras que la restricción de estacionariedad se representa con la recta punteada.

9. Paralelización de SMO con CUDA

Uno de los algoritmos más populares para resolver el problema dual es el método de optimización secuencial mínima (*Sequential Minimal Optimization*), propuesto por Platt (1998). El algoritmo, en su forma mas básica, se puede dividir en cuatro etapas iterativas:

- Selección del par de observaciones $\langle i, j \rangle$ a optimizar aplicando algún criterio.
- Cálculo del tamaño del paso óptimo λ, respecto a las restricciones de feasibilidad dual y el valor óptimo según el método de Newton.
- Actualización del vector gradiente $\vec{g}$ utilizando λ.
- Actualización de las variables duales $\langle \alpha_i, \alpha_j \rangle$ utilizando λ, tal que se mantenga la condición de estacionariedad.

El criterio de selección del par $\langle i, j \rangle$ es clave para mejorar la velocidad de convergencia del algoritmo. Existen diferentes heurísticas para seleccionarlo, en este caso se utilizó el criterio del par que presente los valores más extremos de gradiente, (*maximal violating pair*), esto representa una medida estimada

de la violación de las condiciones de optimalidad de KKT en el punto que se está evaluando. Si bien una buena selección de instancias a optimizar se traducirá en una menor cantidad de iteraciones necesarias, y por tanto, acelerará el ritmo de convergencia, también es importante que el costo de la selección no sea costoso en sí. El criterio de maximal violating pair permite realizar la selección en tiempo $O(n)$ respecto a la cantidad de instancias (Bottou y Lin, 2007).

Respecto al cálculo de λ, se puede decir que todas las variables duales del problema estan restringidas según lo siguiente: $A_i \leq y_i \alpha_i \leq B_i$

$$A_i = \begin{cases} -C, & \text{si } y_i < 0 \\ 0, & \text{si } y_i > 0 \end{cases} \qquad B_i = \begin{cases} 0, & \text{si } y_i < 0 \\ C, & \text{si } y_i > 0 \end{cases}$$

Esto supone una manera alternativa de escribir la restricción $0 \leq \alpha_i \leq C$. Al graficar estas restricciones sobre las curvas de nivel del problema, siendo cada eje un α_i, se puede observar que la región feasible esta delimitada por un cuadrado de lado C. Se utiliza el término *restricciones de caja* para referirse a esto. El lambda elegido no puede ser tal que α_i o α_j violen las restricciones de caja. La tercer y última restricción al valor de λ es que no puede superar el óptimo que se hallaría al dar un paso en la dirección óptima según el método de Newton. El paso según el método de Newton para dos variables resulta ser una solución directa al subproblema, por lo que se puede aseverar que SMO encuentra una solución analítica al problema. Otros métodos numéricos como el del gradiente conjugado convergen a través de aproximaciones y se necesita definir una constante ϵ para controlar la precisión con la que se busca cumplir las condiciones de optimalidad.

La actualización del vector $\vec{g}$ implica recalcular una fila completa de la matriz de kernels, mientras que la actualización de las variables duales consiste en dar un paso hacia la dirección de y_k según la magnitud λ hallada.

El algoritmo resultante es el siguiente (Bottou y Lin, 2007):

Algorithm 1 SMO con par en máxima violación

$\alpha_k \leftarrow 0, \quad k = 1, \ldots, n$
$g_k \leftarrow 1, \quad k = 1, \ldots, n$
while True **do**
    $i \leftarrow \arg\max_i \quad y_i g_i$ tal que $y_i \alpha_i < B_i$
    $j \leftarrow \arg\min_j \quad y_j g_j$ tal que $A_j < y_j \alpha_j$
    if $y_i g_i \leq y_j g_j$ **then**
        Break
    end if
    $\lambda \leftarrow \min\{B_i - y_i \alpha_i, \quad y_j \alpha_j - A_j, \quad \frac{y_i g_i - y_j g_j}{K_{ii} + K_{jj} - K_{ji} K_{ij}}\}$
    $g_k \leftarrow g_k + \lambda y_k (K_{jk} - K_{ik}), \quad k = 1, \ldots, n$
    $\alpha_i \leftarrow \alpha_i + \lambda y_i$
    $\alpha_j \leftarrow \alpha_j - \lambda y_j$
end while

La paralelización del algoritmo resulta sencilla y eficiente si se identifican las primitivas necesarias. La búsqueda de los índices $\langle i, j \rangle$ en las lineas 4 y 5 se puede traducir en un par de operaciones *map-reduce*, tal que el paso de mapeo asigne $\pm\infty$ a los valores que no califican para la búsqueda, mientras que la reducción se puede realizar segun las recomendaciones de NVIDIA para tales operaciones. Dadas las semanticas de los valores de infinito definidas según la IEEE754, los valores invalidos no calificarán para la reducción. Una optimización interesante respecto a realizar el mapeo y la reducción en el mismo kernel CUDA es que se puede realizar el mapeo directamente sobre la memoria compartida y reutilizarla para la reducción, sin necesidad de utilizar la memoria global en medio de ambos pasos. En la implementación se aprovecha la posibilidad de utilizar templates de C++11 con lambdas ejecutables del lado del *device* para volver al kernel más reutilizable.

La actualización del vector gradiente en la línea 10, que resulta ser el paso más costoso, se vuelve una operación de mapeo tradicional por lo que su paralelización no agrega complejidad a la implementación. Es recomendable realizar la actualización de las variables α_i, α_j en las líneas 11 y 12 del lado del host y del device por separado y en paralelo. De esta forma se evitan copias y barreras de sincronización innecesarias. Si se realiza la búsqueda del par $\langle i, j \rangle$ y se actualiza el vector $\vec{g}$ del lado del device, no será necesario mantener un buffer con los valores de $\vec{g}$ del lado del host.

La implementación realizada, *psvm*, se puede obtener del repositorio `https://github.com/franciscoda/psvm`. Los binarios que se pueden compilar son *svm* y *cusvm*. Ambos implementan un algoritmo de entrenamiento con SMO y clasificación binaria y multiclase (con 1AA) de los datos de prueba. Se puede obtener una lista de opciones de línea de comando ejecutando `$svm --help` o `$cusvm --help`. Evidentemente, la única diferencia entre ambos ejecutables es que `cusvm` implementa la versión paralelizable con CUDA del algoritmo de entrenamiento para aprovechar el hardware apropiado, mientras que `svm` utiliza una implementación completamente secuencial.

10. SVM Multiclase – SVC

Como se mencionó en secciones anteriores, el clasificador SVM es binario, por lo que existen varias extensiones para lograr la clasificación multiclase de un vector desconocido, $\vec{u}$ (Hsu y Lin, 2002):

- Uno contra todos (*1AA – one against all*): Consiste en entrenar k modelos, donde k es la cantidad de clases posibles. Cada modelo h_k es entrenado tal que sólo los puntos con clase k se consideran positivos, mientras que el resto del dataset se consideran muestras negativas. La

muestra $\vec{u}$ se clasificará como k tal que la hipótesis h_k produce el mayor valor en su función de decision. Esta claro que este método resulta contradictorio puesto que también se mencionó que un SVM no es un modelo probabilístico, por lo que la salida de la función de decisión no se debería tomar como un valor de probabilidad. Sin embargo, en la práctica produce resultados aceptables y tiene una implementación simple, por lo que es el que se utiliza en psvm.

- Uno contra uno (*1A1 – one against one*): Consiste en entrenar $\frac{k(k-1)}{2}$ modelos. Para esto se enfrentan pares de clases distintas y se entrena un h_k en cada caso tal que a una clase se la considera positiva y a la otra negativa. La muestra $\vec{u}$ se clasificará a través de una votación entre todas las hipótesis h_k: el k que resulte favorecido en la mayor cantidad de hipótesis dictará la decisión final. Este método es el que goza de mayor popularidad (siendo el único implementado en la ubicua LIBSVM: Chang y Lin (2011) y produce mejores resultados que 1AA. Se podría pensar que tiene un costo de entrenamiento mayor, puesto que se debe entrenar para una mayor cantidad de hipótesis, sin embargo, se debe considerar que los problemas que resuelven son más pequeños.

- Grafo acíclico dirigido (*DAGSVM*): Consiste en una fase de entrenamiento idéntica al método 1A1. Luego se construye un grafo acíclico dirigido binario con un nodo raíz. El grafo presenta $\frac{k(k-1)}{2}$ nodos internos; es decir, cada nodo interno se corresponde a una h_k, mientras que en las hojas se disponen las k clases. El vector $\vec{u}$ será evaluado por la hipótesis de la raíz y según el resultado de la clasificación, el proceso continua por el nodo de la izquierda o derecha. Eventualmente se llegará a una hoja que corresponde a la clasificación resultante. El tiempo para clasificar con el DAG resulta menor que en el caso 1A1.

11. Aplicación y resultados

La base de datos MNIST (*Modified National Institute for Standarization and Technology database*) es un dataset que presenta 60000 imágenes en escala de grises de 28x28 pixeles donde cada imágen corresponde a un dígito decimal manuscrito. Adicionalmente se proporciona un conjunto de 10000 imágenes de iguales características para evaluación. El dataset puede encontrarse en http://yann.lecun.com/exdb/mnist/. Los datos se encuentran en un formato binario denominado IDX. Además, los atributos se encuentran en un archivo aparte de sus clases correspondientes en otro archivo IDX.

Este es uno de los problemas más populares en cuanto a aprendizaje supervisado con imágenes.

Se suele aplicar una plétora de algoritmos de preprocesamiento, ya sean de naturaleza estadística (normalización, centrado de datos) o filtros de procesamiento digital de imágenes (deskewing, centrado de la imagen, blurring). Para los propósitos de este trabajo, basta con demostrar que el algoritmo implementado funciona en términos aceptables de precisión y tiempo de convergencia. Por lo tanto, sólo se optó por realizar una normalización min–max para llevar cada atributo al rango $[0; 1]$. Esto es necesario para que la aplicación de ciertos kernels, como el gaussiano, no resulte en un valor sesgado.

A continuación se detallan los resultados más relevantes obtenidos a través de un proceso de barrido similar a un GridSearch exhaustivo:

Hiperparámetros			Métricas		
$n_{training}$	C	Kernel	Accuracy	n_{SVs}	$t_{training}$
5000	0,2	linear	$1225/1500 \approx 0{,}816667$	3407	$155{,}024s$
5000	0,8		$1211/1500 \approx 0{,}807333$	3247	$454{,}835s$
5000	1		$1225/1500 \approx 0{,}816667$	3256	$523{,}022s$
2000	0,6	rbf($\gamma = 0{,}05$)	$1279/1500 \approx 0{,}852667$	9033	$16{,}9415s$
5000			$1396/1500 \approx 0{,}930667$	15475	$44{,}449s$
10000			$1222/1500 \approx 0{,}814667$	22740	$104{,}929s$
20000			$1284/1500 \approx 0{,}856$	33658	$706{,}929s$
2000	0,4	rbf($\gamma = 0{,}05$)	$1330/1500 \approx 0{,}886667$	8661	$15{,}2342s$
5000			$1291/1500 \approx 0{,}860667$	14942	39,9954
10000			$1404/1500 \approx 0{,}936$	22139	$92{,}3822s$
20000			$1323/1500 \approx 0{,}882$	32976	$614{,}402s$
20000	0,1	rbf($\gamma = 0{,}05$)	$1189/1500 \approx 0{,}792267$	35611	$473{,}592s$
20000	0,2		$1415/1500 \approx 0{,}943333$	33533	$515{,}678s$
30000	0,1		$1404/1500 \approx 0{,}936$	46188	$897{,}725s$
30000	0,2		$1322/1500 \approx 0{,}88133$	42282	$961{,}887s$
30000	0,2	rbf($\gamma = 0{,}01$)	$1316/1500 \approx 0{,}877333$	25710	$489{,}447s$
30000	0,5		$1339/1500 \approx 0{,}892667$	20085	$547{,}146s$
30000	0,8		$1387/1500 \approx 0{,}924667$	17956	$663{,}688s$
30000	1		$1355/1500 \approx 0{,}903333$	17131	$743{,}81s$
60000	1,5	rbf($\gamma = 0{,}01$)	$1407/1500 \approx 0{,}938$	24915	$2915{,}11s$
60000	1		$1396/1500 \approx 0{,}930667$	27035	$2329{,}51s$
60000	0,5		$1358/1500 \approx 0{,}905333$	31927	$1714{,}46s$
60000	0,2		$1408/1500 \approx 0{,}938667$	41171	$1540{,}21s$
60000	0,1		$1313/1500 \approx 0{,}875333$	50563	$1684{,}39s$

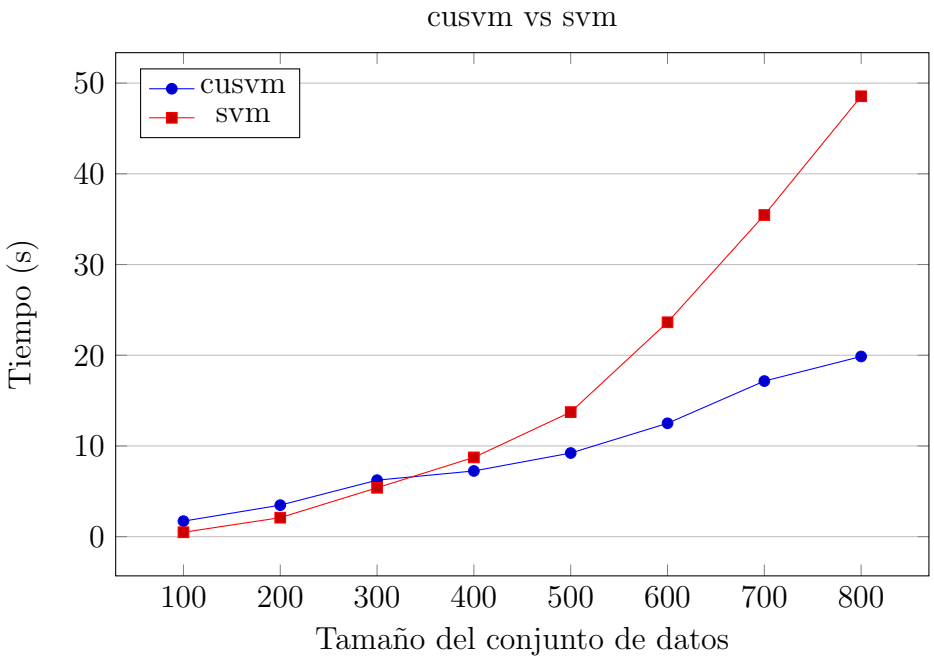
Se puede observar que el valor de C influye negativamente sobre tiempo

de entrenamiento, sin embargo, sus efectos sobre el accuracy o el número de vectores de soporte son impredecibles a priori.

A continuación se comparan los tiempos de entrenamiento[1] entre la implementación secuencial, `svm` y la implementación CUDA, `cusvm`. El subconjunto de entrenamiento siempre se selecciona de la misma manera, a fin de asegurar que el problema que resuelven ambas implementaciones sea el mismo.

Kernel lineal, $C = 0{,}5$

$n_{training}$	t_{cusvm}	t_{svm}	Speedup
100	1,7106s	0,4847s	0.283
200	3,4737s	2,0944s	0.602
300	6,2286s	5,3899s	0.865
400	7,2436s	8,7373s	1.206
500	9,2192s	13,7326s	1.489
600	12,4969s	23,6342s	1.891
700	17,1524s	35,4571s	2.067
800	19,8652s	48,5484s	2.443
900	24,2970s	67,3215s	2.770
1000	30,9150s	88,8882s	2.875



[1]El tiempo de entrenamiento no incluye overheads como la carga del dataset desde el disco ni la inicialización del runtime de CUDA

En la tabla se puede observar que el punto de inflexión se encuentra entre un n de 300 a 400. En este punto, la ganancia que se obtiene con el procesamiento paralelo logra compensar el overhead incurrido en tareas como sincronizar los lanzamientos de kernels y copias de memoria entre el host y el device. El método tiene una complejidad algorítmica de $O(n^3 \cdot d)$. Este crecimiento polinómico se ve reflejado en la implementación secuencial. Sin embargo, en la implementación CUDA, cada operación del algoritmo se puede resolver en una única invocación de kernel para los valores de n mostrados, por lo que el único factor que determina el crecimiento en el tiempo de entrenamiento es la cantidad de iteraciones necesarias para resolver el problema.

Referencias

Boser, B. E., Guyon, I. M., y Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. En *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–152). New York, NY, USA: ACM. Descargado de http://doi.acm.org/10.1145/130385.130401 doi: 10.1145/130385.130401

Bottou, L., y Lin, C.-J. (2007). Support vector machine solvers. *Large scale kernel machines*, *3*(1), 301–320.

Boyd, S., y Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167. Descargado de https://doi.org/10.1023/A:1009715923555 doi: 10.1023/A:1009715923555

Chang, C.-C., y Lin, C.-J. (2011). Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, *2*(3), 27.

Chapelle, O. (2007). Training a support vector machine in the primal. *Neural computation*, *19*(5), 1155–1178.

Hsu, C.-W., y Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, *13*(2), 415–425.

Platt, J. (1998, April). *Sequential minimal optimization: A fast algorithm for training support vector machines* (Inf. Téc.). Descargado de https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/

Witten, I., Elbe, F., y Hall, M. (2011). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.