Chunking Suave: Un Método de Descomposición para el Entrenamiento de Máquinas de Soporte Vectorial

M.C. Ricardo del Angel Pérez Flores
Departamento de Cómputo e Informática
Universidad Autónoma del Carmen
e-mail: rperez@pampano.unacar.mx

Dr. Arturo Hernández Aguirre
Area de Ciencias de la Computación
Centro de Investigación en Matemáticas, A.C.
e-mail: artha@cimat.mx

Resumen

Este artículo propone el método de Chunking Suave para el rápido entrenamiento de una Máquinas de Soporte Vectorial. Este método, basado en Conjuntos de Trabajo, es una modificación del método de Chunking propuesto originalmente por Vapnik. Las ventajas del Chunking Suave son su velocidad y una menor cantidad de vectores de soporte. Describimos nuestro método, se reportan experimentos con la base de datos MNIST, y explicamos porqué un SVM de Chunking Suave generaliza de manera comparable a los algoritmos tradicionales.

Palabras clave: Máquinas de Soporte Vectorial, Chunking, Chunking Suave, Teoría Estadística del Aprendizaje, Reconocimiento de Patrones.

1. Introducción.

Las Maquinas de Soporte Vectorial (Support Vector Machines o SVM) constituyen una de las técnicas mas reciente para reconocimiento de patrones. En 1995, Vapnik [8] y su equipo de colaboradores establecieron las bases teóricas de la Teoría Estadística del Aprendizaje (Statistical Learning Theory o SLT), dando como resultado práctico las SVMs. Estas, afirma Burges [1], son un paradigma aparte de las Redes Neuronales que, aunque tienen muchas similitudes, están mejor fundamentadas en la teoría. Tienen mejor capacidad de generalización; es decir, capacidad para clasificar correctamente datos distintos al conjunto de entrenamiento. La diferencia entre los métodos es notable; el entrenamiento de la Red Neuronal no controla la generalización, sino que asume que ésta será correcta si el error de entrenamiento es pequeño sobre un conjunto de entrenamiento lo suficientemente grande. El entrenamiento de una SVM, en cambio, controla simultáneamente ambos parámetros: error de entrenamiento y generalización.

Una SVM efectúa la clasificación de objetos puntuales en dos clases por medio de una superficie de decisión determinada por ciertos puntos del conjunto de entrenamiento,

conocidos como vectores de soporte. Dicha superficie se obtiene al resolver un problema de Programación Cuadrática Convexa con restricciones lineales, cuyo número de variables es igual al número de datos de entrenamiento. La solución de este problema es el objetivo de este artículo.

2. Formulación del Problema de Clasificación con SVM.

Tenemos l muestras independientes e idénticamente distribuidas, provenientes de una distribución de probabilidad $P(x,y)$ desconocida, consistentes en pares conformados por un vector y una etiqueta de clase $(x_1,y_1),(x_2,y_2),...,(x_l,y_l)$, donde $x_i \in R^n$ y $y_i \in \{1,-1\}$. Deseamos construir una SVM capaz de mapear los valores de los vectores x_i a los valores de las etiquetas y_i correspondientes por medio de una función de decisión $y = f : R^n \to \{1,-1\}$.

2.1. Caso Linealmente Separable.

Es el caso más simple, Figura 2.1, donde los datos pueden ser clasificados por un hiperplano orientado, conocido como hiperplano de separación, con ecuación $w \cdot x + b = 0$, donde w es un vector normal al hiperplano llamado vector de pesos y b conoce como sesgo.

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

$$w \cdot x_i + b \geq 1 \text{ para } y_i = 1 \tag{2.1}$$

$$w \cdot x_i + b \leq -1 \text{ para } y_i = -1 \tag{2.2}$$

La distancia entre H_1 y H_2 nos da el margen de separación para las dos clases, el cual es $M = \frac{2}{\|w\|}$. Por lo tanto, para encontrar el par de hiperplanos que dan el máximo margen, y al mismo tiempo el hiperplano de separación, es necesario maximizar distancia M, lo que equivale a minimizar el valor de $\|w\|$, sujeto a las restricciones dadas.

De esta forma, llegamos al siguiente problema de Programación Cuadrática Convexa:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 = \frac{1}{2} w \cdot w \tag{2.3}$$

sujeto a las restricciones

$$-y_i (w \cdot x_i + b) + 1 \leq 0 \text{ para } i = 1,2,...,l. \tag{2.4}$$

La solución de este problema de optimización se encuentra en el Punto Silla Lagrangiano primal del mismo, dado por

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

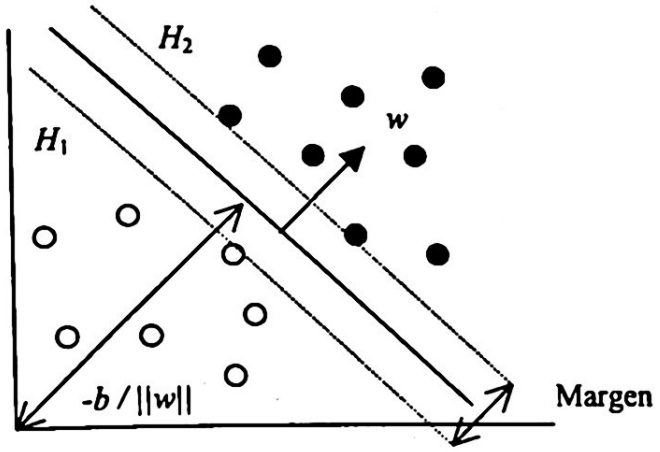

Figura 2.1. El hiperplano de separación para el caso linealmente separable en R^2.

donde cada α_i para $i = 1,2,\ldots,l$, es un multiplicador de Lagrange. Dicha solución es la misma que para el problema dual. Para encontrarlo debemos minimizar el Lagrangiano respecto a w y b, al tiempo que es maximizado con respecto a $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_l)$.

Así, podemos replantear el problema como

$$\max_{\alpha} \quad L_D(\alpha) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j x_i \cdot x_j + \sum_{i=1}^{l}\alpha_i \tag{2.6}$$

sujeto a las restricciones

$$\sum_{i=1}^{l}\alpha_i y_i = 0 \tag{2.7}$$

$$\alpha_i \geq 0 \text{ para } i = 1,2,\ldots,l. \tag{2.8}$$

Este problema cumple con la forma estándar del problema de Programación Cuadrática Convexa con una restricción de igualdad (2.7) y l de desigualdad (2.8). Puede manejado con mayor facilidad, pues las restricciones originales son sustituidas por restricciones que incluyen a los multiplicadores de Lagrange, además de que los datos de entrenamiento aparecen como simples productos punto de vectores. Dicha situación resulta ventajosa en el caso no lineal, que se revisará en la siguiente sección.

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

Para las SVM, los vectores de soporte son los elementos críticos del conjunto de entrenamiento. Ellos definen la región de decisión, cumplen con la igualdad en las restricciones (2.8) y si los eliminaramos cambiaría la solución encontrada, contrario a lo que sucedería, si todos los demás puntos de entrenamiento fueran removidos o movidos de lugar sin cruzar H_1 o H_2.

Una vez solucionado el problema, tendremos un vector α^* de multiplicadores de Lagrange, con los cuales podemos calcular el vector de pesos óptimo y el sesgo óptimo

$$w^* = \sum_{i=1}^{l} \alpha^*_i x_i y_i \tag{2.9}$$

$$b^* = \frac{1}{k}\left[\sum_{s=1}^{k}\left(y_s - w^* \cdot x_s\right)\right] \tag{2.10}$$

donde k es el número de vectores de soporte.

Así, obtenemos la ecuación del hiperplano de decisión $u(x) = w^* \cdot x + b$.

2.2. Caso No Lineal.

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

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

Cabe señalar que tendría un altísimo costo computacional efectuar explícitamente estos mapeos y luego calcular los productos punto. Sin embargo, cuando un kernel cumple las Condiciones de Mercer (ver [6]), el mapeo es implícito. Esto es, trabajamos en R^n aunque la solución se encuentre en R^m.

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

$$\max_{\alpha} \quad L_D(\alpha) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i, x_j) + \sum_{i=1}^{l}\alpha_i \tag{2.11}$$

sujeto a las restricciones

$$\sum_{i=1}^{l}\alpha_i y_i = 0 \tag{2.12}$$

$$\alpha_i \geq 0 \text{ para } i = 1,2,...,l. \tag{2.13}$$

La ventaja del problema dual sobre el problema primal, por depender de los multiplicadores de Lagrange y de productos punto de vectores de entrenamiento se hace evidente aquí: si el problema dependiera del vector de pesos, tendríamos tantas incógnitas como la dimensión del espacio de rasgos, que se ha visto que tiene una dimensión demasiado grande.

El sesgo óptimo se obtiene como

$$b^* = \frac{1}{k}\left[\sum_{s=1}^{k}\left(y_s - \sum_{i=1}^{l}\alpha_i y_i \cdot k(x_s, x_i)\right)\right] \tag{2.14}$$

donde k es el número de vectores de soporte.

La clasificación se efectúa mediante la misma función que en el caso linealmente separable, pero en este caso no es posible calcular el vector de pesos explícitamente debido a que necesitaríamos conocer φ, por lo tanto tenemos

$$u(x) = \sum_{i=1}^{l}\alpha_i y_i k(x_i, x) + b^* \tag{2.15}$$

Así, la extensión de la teoría se traduce en encontrar kerneles que identifiquen a ciertas familias de regiones de decisión que puedan escribirse como productos punto en algún espacio de rasgos. En la práctica, será necesario emplear el enfoque no separable para el caso no lineal y escoger el kernel que combine la mejor generalización con un mapeo al espacio de rasgos de la menor dimensión posible. Los kerneles mas conocidos y utilizados, según describe Gunn [3], son:

Tipo de Kernel	Función
Polinomiales de Grado d	$k(u,v) = (u \cdot v)^d \quad k(u,v) = (u \cdot v + 1)^d$
Funciones de Base Radial (RBF) Gaussianas	$k(u,v) = e^{\frac{\|u-v\|^2}{2\sigma^2}}$
Perceptrones Multicapas	$k(u,v) = \tanh(ku \cdot v - \delta)$

Existen muchos otros tipos de kerneles, tales como RBF exponenciales, series de Fourier, splines y splines B. Además, es posible construir kerneles más complicados mediante la suma o el producto de varios kerneles [3].

Métodos de Conjunto de Trabajo.

Aquí podemos distinguir dos clases de problemas: los problemas "pequeños", que se puede resolver utilizando cualquier paquete de optimización de propósito general que resuelva problemas de Programación Cuadrática Convexa con restricciones lineales. Estas técnicas requieren que los datos sean mantenidos en memoria en forma de una matriz, llamada Hessiano. La complejidad del problema de entrenamiento crece con el tamaño de esta matriz, limitando los enfoques a conjuntos de pocos miles de datos.

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

Se ha reportado que estos métodos trabajan muy bien en la práctica y hacen posible repartir con conjuntos de datos de varias decenas de miles de puntos [5].

3.1. El Método de Chunking.

El método de Chunking [8] (por pedazos) se inicia entrenando una SVM a partir de un subconjunto pequeño de muestras elegidas arbitrariamente, llamado "chunk" ("pedazo"). Las demás muestras se prueban con el clasificador resultante y se eligen los N_V puntos que más violen las condiciones de Karush-Kuhn-Tucker (KKT), junto con los N_S vectores de soporte del pedazo entrenado, para formar un nuevo pedazo de tamaño $M = N_S + N_V$. Este proceso se realiza iterativamente, inicializando el vector α para cada nuevo subproblema con los valores de salida del anterior, hasta que todos los puntos satisfacen las condiciones KKT (explicadas en la Sección 4).

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

3.2 Los Métodos de Descomposición y SMO.

El método anterior requiere que el número de vectores de soporte N_S sea lo suficientemente pequeño para que un Hessiano de tamaño $(N_S + N_V) \times (N_S + N_V)$ quepa en la memoria. Un algoritmo de conjunto de trabajo alternativo que supera esta limitación fue propuesto por Osuna, Freund y Girosi [5]. En este algoritmo, como en el anterior, se entrena una pequeña porción de los datos de entrenamiento en cada iteración, con la diferencia de que el tamaño del conjunto de trabajo permanece fijo y pueden haber más vectores de soporte que datos en dicho conjunto.
Consiste en particionar el conjunto de entrenamiento en dos conjuntos B y N, donde B será el conjunto de trabajo de tamaño M. El vector α también se divide en α_B y α_N, de modo que α_N permanecerá fijo y α_B cambiará en cada iteración. En cada iteración, se realiza un intercambio de datos entre los dos conjuntos, reemplazándose algunos valores de B por otros de N, de acuerdo a alguna heurística determinada.

El problema es independiente del número de vectores de soporte, y puede probarse que la función objetivo mejora en cada iteración, por lo que este algoritmo debe converger hacia el óptimo global en un número finito de iteraciones.

El algoritmo de Optimización Secuencial Mínima (Sequential Minimal Optimization o SMO) [7] se deriva tomando la idea del método de descomposición hasta su extremo optimizando un subconjunto mínimo de dos puntos en cada iteración. Fue propuesto por Platt [7]. El poder de esta técnica reside en el hecho de que el problema de optimización para dos puntos permite una solución analítica. En cada paso, SMO escoge dos elementos α_i y α_j, con $i \neq j$, para optimizarlos, encuentra los valores óptimos para estos dos parámetros dado que todos los otros están fijos, y actualiza el vector α de acuerdo a esto. La opción de los dos puntos es determinada por una heurística, mientras que la optimización de los dos multiplicadores es realizada analíticamente.

4. El Método de Chunking Suave.

Una buena manera de revisar que algún algoritmo trabaja es comprobando que la solución satisface todas las condiciones KKT para el problema primal, pues son condiciones necesarias y suficientes para que la solución sea óptima. En los métodos de conjuntos de trabajo se realizan iteraciones en cada una de las cuales se construyen SVM, con el fin de probar si todos los datos cumplen con las condiciones KKT, las cuales están dadas por las siguientes ecuaciones:

$$\frac{\partial f}{\partial w} = -\sum_{i=1}^{l} \alpha_i^* \frac{\partial g_i}{\partial w} \Rightarrow w = -\sum_{i=1}^{l} \alpha_i^* (-y_i x_i) \Rightarrow w = \sum_{i=1}^{l} \alpha_i^* y_i x_i \tag{4.1}$$

$$\frac{\partial f}{\partial b} = -\sum_{i=1}^{l} \alpha_i^* \frac{\partial g_i}{\partial b} \Rightarrow 0 = -\sum_{i=1}^{l} \alpha_i^* y_i \Rightarrow \sum_{i=1}^{l} \alpha_i^* y_i = 0 \tag{4.2}$$

$$\alpha_i^* \geq 0 \text{ para } i = 1,2,...,l, \tag{4.3}$$

$$\alpha_i^* g_i(w^*, b^*) = \alpha_i^* \left[-y_i \left(w^* \cdot x_i + b^* \right) + 1 \right] = 0 \text{ para } i = 1,2,...,l, \tag{4.4}$$

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

Así, para verificar si una SVM entrenada cumple con las condiciones KKT, es suficiente con verificar que todas las α_i satisfacen las ecuaciones (4.2) a (4.5).

Cuando termina un entrenamiento tenemos valores de α^* y, por consiguiente, de w^* y b^*. Generalmente los valores del vector α^* encontrado cumplen con las condiciones establecidas por (4.2) y (4.3), pues son las condiciones del problema de Programación Cuadrática ya resuelto. Por lo tanto, nos resta verificar si para cada punto de entrenamiento los multiplicadores cumplen con (4.4) y (4.5). Para ello, hagamos $u(x_i) = w^* \cdot x_i + b^*$ y veamos a las ecuaciones (4.4) y (4.5) como

$$\alpha_i^* \left[-y_i u(x_i) + 1 \right] = 0 \text{ para } i = 1,2,...,l, \tag{4.6}$$

$$-y_i u(x_i) + 1 \leq 0 \text{ para } i = 1,2,...,l. \tag{4.7}$$

De acuerdo a (4.4), hay dos factores que pueden hacer 0 dicha igualdad: α_i y $-y_i u(x_i) + 1$; y de acuerdo a (4.3), tenemos dos casos posibles para cada x_i de un conjunto de entrenamiento: $\alpha_i^* = 0$ y $\alpha_i^* > 0$.

Caso 1. Cuando $\alpha_i^* = 0$, entonces $-y_i u(x_i) + 1$ puede o no ser distinto de 0. Pero como ese factor debe cumplir con (4.5), no puede ser positivo y, por lo tanto, al despejar adecuadamente tenemos que $y_i u(x_i) \geq 1$. El cero en el multiplicador nos dice que el vector de entrenamiento x_i no es un vector de soporte y por lo tanto no cae en ninguno de los hiperplanos que acotan a éste; por lo tanto su valor no puede ser 1. Así, para toda $\alpha_i^* = 0$ debe cumplirse que $y_i u(x_i) > 1$.

Caso 2. Cuando $\alpha_i^* > 0$, entonces necesariamente $-y_i u(x_i) + 1 = 0$. De esa manera, el factor cumple con (4.5) y, por lo tanto, al despejar adecuadamente tenemos que $y_i u(x_i) = 1$. En este caso, el vector de entrenamiento x_i sí es un vector de soporte. Así, para toda $\alpha_i^* > 0$ debe cumplirse que $y_i u(x_i) = 1$.

Es claro que cualesquiera vectores que no hayan tomado parte de un conjunto de entrenamiento no tienen un valor de α determinado, por lo cual para ellos no se toman en cuenta las condiciones (4.1) a (4.4). Sin embargo, cuando al utilizar el método de Chunking queremos probar si un vector de entrenamiento cumple con las condiciones KKT, asumimos valores de $\alpha = 0$ para los vectores que no fueron parte del pedazo.

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

4.1. El Algoritmo de Chunking Suave y su Implementación.

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

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

Selección del Conjunto de Trabajo Inicial. Se construye el conjunto de trabajo de manera aleatoria, pero estableciendo un balance en el numero de vectores de cada clase; es decir, se van eligiendo uno de cada clase de manera intercalada hasta completar el conjunto de trabajo. Si acaso la cantidad de vectores de una clase es menor a la mitad del tamaño del conjunto de trabajo inicial, entonces todos ellos serán incluidos en él.

Primera Fase (Suave). En esta fase, lo que sigue se repetirá hasta que no existan vectores fuera del conjunto de trabajo que estén mal clasificados. Se resuelve el problema de Programación Cuadrática para los elementos del conjunto de trabajo, se extraen los vectores de soporte y se obtiene una región de decisión.

Se calculan los valores de $y_i u(x_i)$ para todos los vectores del conjunto de entrenamiento y se hacen dos listas, ordenadas de acuerdo a los valores recién calculados: una para los vectores que violan la condición de no estar bien clasificados (candidatos a entrar al conjunto de trabajo), y otra para los que no fueron vectores de soporte en el conjunto de trabajo (candidatos a salir).

Se determina un número de intercambios entre los dos conjuntos, de acuerdo a la menor de las cantidades de vectores entrantes y salientes. En caso de haber cambios posibles, estos se efectúan intercambiando al vector de afuera que mas viola la condición con el no vector de soporte de adentro que este mas alejado del margen, y así sucesivamente con los segundos, terceros, etc., de cada lista.

En caso de no haber candidatos a salir pero si a entrar, o no haber obtenido vectores de soporte, se incrementa el tamaño del conjunto de trabajo de acuerdo al parámetro ya

establecido, y esos lugares se ocupan con los vectores candidatos a entrar con mayor violación a la condición.

Si hubieron cambios o un incremento para el conjunto de trabajo, se pasa a la siguiente iteración con el nuevo conjunto de trabajo. En caso de no haber candidatos a entrar, no importa si hay o no candidatos a salir, la fase iterativa termina.

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

5. Resultados Experimentales.

Se ha probado este nuevo método utilizando la base de datos de caracteres manuscritos MNIST [4] que consta de imágenes de 28×28, lo que resulta en vectores de 784 elementos enteros de 0 a 255; es decir, nuestro espacio de entrada es R^{784}. Los experimentos aquí mostrados fueron hechos para el problema de separar el dígito 0 de los demás. El Experimento 1 corresponde a 250 datos de entrenamiento, y el Experimento 2 500 datos. Se ha calculado la generalización para 4500 datos de prueba en ambos casos.

5.1 Experimento 1.

Primeramente veremos el caso del entrenamiento de SVM utilizando 25 muestras de cada dígito, lo que nos da 250 patrones de entrenamiento. Se encontraron las regiones de decisión para cada enfoque y se tomaron algunos datos resultantes. Las tablas 5.1 a 5.3 nos muestran en forma comparativa dichos resultados. Cabe señalar que el tiempo que aparece en el renglón de la iteración adicional del método de Chunking Suave es exclusivo de la iteración, efectuada al final de la primera parte del método.

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

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

No solamente la velocidad de entrenamiento y el número de vectores de soporte encontrados son factores a favor del Chunking Suave, sino la capacidad de generalización que se obtiene resulta ser casi la misma. Incluso al efectuar la iteración adicional lo que gana no es significativo. Puede decirse que la región de decisión del Chunking Suave

es tan buena como las de las SVM convencionales, con la gran ventaja de la rapidez con que se obtienen.

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

Tabla 5.1. Resultados para un kernel polinomial de grado 4 con 250 muestras.

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

Tabla 5.2. Resultados para un kernel polinomial de grado 5 con 250 muestras.

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

Tabla 5.3. Resultados para un kernel polinomial de grado 6 con 250 muestras

5.2 Experimento 2.

Ahora nos ocuparemos del caso del entrenamiento de SVM utilizando 50 muestras de cada dígito, lo que nos da 500 patrones de entrenamiento. Como en el ejemplo anterior, se encontraron las regiones de decisión para cada enfoque y se tomaron algunos datos resultantes. Las tablas 5.4 a 5.6 nos muestran en forma comparativa dichos resultados.

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

Tabla 5.4. Resultados para un kernel polinomial de grado 4 con 500 muestras.

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

Tabla 5.5. Resultados para un kernel polinomial de grado 5 con 500 muestras.

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

Tabla 5.6. Resultados para un kernel polinomial de grado 6 con 500 muestras.

Ahora la diferencia en velocidades se ha acentuado pero el porcentaje de ahorro sigue siendo superior al 50%. Aún si sumamos el tiempo de la iteración adicional, la ventaja sigue siendo del Chunking Suave. Las diferencias con el total, aún con la iteración adicional son mucho menores en porcentaje y en ocasiones gana un método y en otras el otro. La capacidad de generalización, como era de esperarse, ha mejorado debido a la cantidad de vectores de entrenamiento.

Conclusiones.

Para terminar explicaremos porqué el Chunking Suave es tan "bueno" para generalizar como el Chunking original. En esta clase de problemas el tamaño del margen ínfimo y ciertamente despreciable, como puede verse en las Tabla 6.1.

Concluimos que la simple región de decisión (correctamente ubicada como en nuestro caso) es suficientemente buena para darnos una generalización adecuada, de la misma forma en que lo haría una SVM entrenada con el método de Chunking o con la función QuadProg. La gran diferencia es que el Chunking Suave requiere de un menor esfuerzo computacional para hacer válidas todas las restricciones del problema, que los otros dos métodos de entrenamiento. La consecuencia de esto es una mayor velocidad a la hora de encontrar la región de decisión. Por lo tanto, si podemos obtener una región de decisión similarmente buena y en un tiempo menor, tenemos un método mucho mejor, sobre todo se utiliza para construir SVMs con grandes cantidades de datos de entrenamiento.

Kernel	Grado 4	Grado 5	Grado 6
Margen con 250 Patrones	5.0501×10^{-26}	1.3834×10^{-32}	3.9758×10^{-39}
Margen con 500 Patrones	1.485×10^{-25}	4.2029×10^{-32}	1.2878×10^{-38}

Tabla 6.1. Anchos de los márgenes de seis SVM.

Hemos descrito el Chunking Suave como un método muy veloz para entrenar SVMs. Otra aportación importante de este trabajo es el reconocimiento de que en muchos problemas que requieren de kerneles, el tamaño del márgen es ínfimo. Aunque no hemos probado con otros problemas, por analogía la gran mayoría de los SVMs que clasifican esta clase de datos a través de kerneles van a tener un ancho de márgen despreciable. De esta forma, una región de decisión que se encuentra dentro de un márgen óptimo tan angosto es equivalente al mismo óptimo. Con la ventaja de menor tiempo de cálculo, menor número de vectores de soporte y similar generalización que otorga, el Chunking Suave supera a los algoritmos con los que fue comparado.

Agradecimientos.

El segundo autor reconoce ayuda parcial para este trabajo a través del proyecto CONACyT No. I-39324-A.

Referencias.

[1] Christopher C. Burges (1998). *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, Kluwer Academis Publishers.

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

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

[4] Yann Le Cun (1999). *The MNIST Database of Handwritten Digits*, Dirección de Internet: http://yann.lecun.com/exdb/mnist/.

[5] Edgar Osuna, Robert Freund, Federico Girosi (1997). *An Improved Training Algorithm for Support Vector Machines*, Proceedings of IEEE Workshop on Neural Networks for Signal Procesing.

[6] Ricardo del A. Pérez Flores (2002). *Entrenamiento de Máquinas de Soporte Vectorial y su Aplicación en el Reconocimiento de Patrones*, Tesis de Maestría, CIMAT, A.C.

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

[8] Vladimir Vapnik (1995). *The Nature of Statistical Learning Theory*, Springer-Verlag.