ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Departamento de Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa

TESIS DOCTORAL

Paralelización hardware-software de técnicas de clasificación basadas en algoritmos de generación de conglomerados de datos

Autor: Javier Cano Montero

Directores: Javier Martínez Moguerza, Javier Castillo Villar y Jose Ignacio Martínez Torre

Don **Javier Martínez Moguerza** y **Jose Ignacio Martínez** profesores titulares y **Javier Castillo Villar** profesor contratado doctor del Departamento de Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa de la Universidad Rey Juan Carlos.

AUTORIZAN:

La presentación de la Tesis Doctoral titulada

Paralelización hardware-software de técnicas de clasificación basadas en algoritmos de generación de conglomerados de datos

Realizada por Don **Javier Cano Montero** bajo su dirección y supervisión.

En Móstoles, a 18 de Marzo de 2015,

D. Javier Martínez Moguerza D. Jose Ignacio Martínez Torre

D. Javier Castillo Villar

A mi abuela Concha.

Agradecimientos

Gracias a...

...mis padres y mis abuelos por el apoyo y sobre todo su paciencia.

...mis directores (y amigos) Javier M. Moguerza, Javier Castillo y Jose Ignacio Martínez por las ideas, las oportunidades, el apoyo y su guía, sin ellos nunca habría sido doctor.

...los antiguos "Departamento de Arquitectura de Computadores" y "Departamento de Estadística e Investigación Operativa" y al actual "Departamento de Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa"

...todos los grandes compañeros que he tenido a lo largo de los años y me han ayudado en la realización de esta tesis: Antonio Pérez, César Pedraza, Pablo Huerta, César Alfaro, Javier Gómez, Felipe Ortega e Ignacio San Román.

...Emilio Castillo y la Universidad de Cantabria por dejarme usar el Altamira.

...mi actual grupo "Face Recognition and Artificial Vision" y

en especial a Isaac Martín por permitirme quedarme en la universidad.

...mis amigos de "La Cueva" Sergio, Raúl, Kike, David y Dani por los buenos ratos y los buenos vicios.

...Sergio, María y Lucía por acompañarme en los momentos buenos y en los malos.

...Ana, Alon, Isa, Cris y Luis por las risas, los palos, las piedras, la locura, los desvaríos y la cerveza.

...Extremoduro porque son muy grandes y en especial por "La Ley Innata".

...la financiación de los proyectos de investigación "Democracy4All", "EducaLAB", "SECONOMICS" e "INVISUM"

Resumen

pSVM es un nuevo algoritmo que permite por primera vez el entrenamiento en paralelo de una SVM. Para conseguirlo el conjunto de entrenamiento se divide en distintas regiones de Voronoi utilizando el algoritmo k-medias, consiguiendo así SVMs más pequeñas y rápidas de entrenar con el objetivo de reducir la carga computacional. permitiendo la futura implementación de sistemas embebidos. Además, se ofrece una comparativa entre libSVM y el nuevo algoritmo pSVM, probando que el segundo es más rápido y ofrece resultados similares que libSVM.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción a técnicas de clasificación

Un clasificador es un proceso que utiliza un modelo para predecir valores desconocidos utilizando algunos valores conocidos. Un clasificador binario con aprendizaje es una función cuya tarea es clasificar los elementos dados de un conjunto en dos grupos diferentes de acuerdo a una regla de clasificación. Algunos ejemplos clásicos son:

- En medicina, determinar si un paciente tiene cierta enfermedad o no.
- En control de calidad en fábricas, decidir si un nuevo producto es bueno para ser vendido o debe descartarse.

Los clasificadores supervisados tienen una fase de entrenamiento que sirve para generalizar las reglas que se utilizarán a pos-

teriori en la clasificación, ya que crear una función dependiente de todas las variables sería muy costoso,

Algunos de estos métodos son vecino más próximo, redes bayesianas, redes neuronales, máquinas de vector soporte.

1.1. Vecino más próximo

La técnica de k-vecino más próximo (o k-NN del inglés k-Nearest Neighbors) es un método no paramétrico utilizado para clasificación y regresión [2]. k-NN usa observaciones en el conjunto de entrenamiento para buscar los vecinos más próximos en el espacio de entrada del nuevo dato a clasificar y clasificarlo por semejanza. El algoritmo está entre los más simples dentro de los algoritmos de aprendizaje computacional.

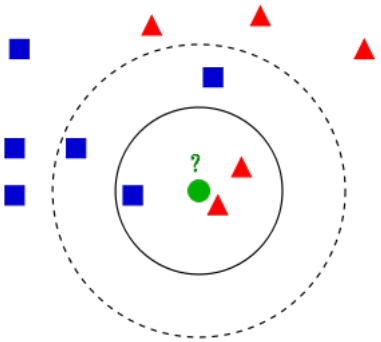

Figura 1.1: Ejemplo de clasificación k-NN

La salida en clasificación es la clase a la que pertenece el nuevo dato. Un objeto se clasifica por votación de la mayoría de sus vecinos, k (un entero positivo, normalmente pequeño) indica

cuántos vecinos votan, si *k=1* el objeto se asigna según la votación del vecino más próximo a él.

En el ejemplo de la figura 1.1 se quiere clasificar el círculo verde. Para *k=1* se clasifica como triángulo, ya que el vecino más próximo es un triángulo. Para *k=3* se clasifica también como triángulo, ya que entre los 3 vecinos más próximos hay 2 triángulos y 1 cuadrado. Para *k=5* se clasifica como cuadrado, ya que entre los 5 vecinos más próximos hay 3 cuadrados y 2 triángulos.

1.1.1. Ejemplo *k*-NN en clasificación de documentos

Dado un documento x, su distancia $d(x, x_i)$ a los documentos en C (siendo $C = \{x_1, \ldots, x_n\}$ un subconjunto de n documentos) induce un orden representado como $x_{(1)}...x_{(k)}...x_{(n)}$, donde $x_{(1)}$ es el documento más cercano a x en la muestra C y $x_{(n)}$ el más lejano. De este modo, la distancia del mensaje x a su k-ésimo vecino más próximo en C se define como $d_k(x, C) = d(x, x_{(k)})$, donde d puede ser cualquier distancia válida entre mensajes. Determinar el valor adecuado de k es un problema abierto. En el caso de información textual, la elección preferida es la distancia inducida por la similaridad del coseno, véase [5], que viene dada, para los mensajes x_i y x_j, por:

$$sim(x_i, x_j) = \frac{\vec{x_i} \bullet \vec{x_j}}{|\vec{x_i}| \times |x_j|} = \frac{\sum_{t=1}^{m} f_{t,i} \times f_{t,j}}{\sqrt{\sum_{t=1}^{m} f_{t,i}^2} \times \sqrt{\sum_{t=1}^{m} f_{t,j}^2}}$$

A partir de ella, la distancia entre dos mensajes se calcula como:

$$d(x_i, x_j) = 1 - sim(x_i, x_j)$$

Supongamos que $C_1, \ldots, C_l$ son un conjunto predefinido de categorías que contienen cada una de ellas una muestra de mensajes asignados a cada categoría por un experto. Para clasificar un nuevo mensaje x en una de tales categorías, todo lo que hay que hacer es asignar el mensaje a la categoría más cercana utilizando el método k-NN, donde, hemos seleccionado un valor de k igual a 1 y x se asigna a C_j si C_j es la categoría tal que $d(x, C_j) = \text{mín}\{d(x, C_1), \ldots, d(x, C_l)\}$.

Algoritmo

Las instancias de entrenamiento son vectores multidimensionales con una clase asignada. La fase de entrenamiento del algoritmo consiste únicamente en almacenar los vectores y sus clases. En la fase de clasificación, utilizando la constante definida k, se mide la distancia del vector sin clasificar (punto de prueba) a los puntos de entrenamiento y se seleccionan los k más próximos para realizar la votación.

La distancia más común para variables continuas es la distancia Euclídea, aunque para variables discretas (clasificación de texto por ejemplo) se suelen utilizar medidas de solapamiento (o distancia de Hamming).

Selección del parámetro k

La elección del valor del parámetro k determinará el correcto funcionamiento de la clasificación. El valor depende de los datos, normalmente los valores grandes de k reducen el "ruido" en la clasificación, pero hacen que las fronteras entre clases sean borrosas. Un buen valor de k puede elegirse utilizando heurísticas o con algoritmos evolutivos [36].

Reducción CNN

CNN (del inglés Condensed Nearest Neighbors) o algoritmo de Hart es un algoritmo diseñado para reducir el conjunto de datos de la clasificación k-NN [18]. Selecciona un conjunto de prototipos U del conjunto de entrenamiento, de forma que 1-NN con U puede clasificar los ejemplos con una precisión similar a 1-NN con el conjunto de datos completo.

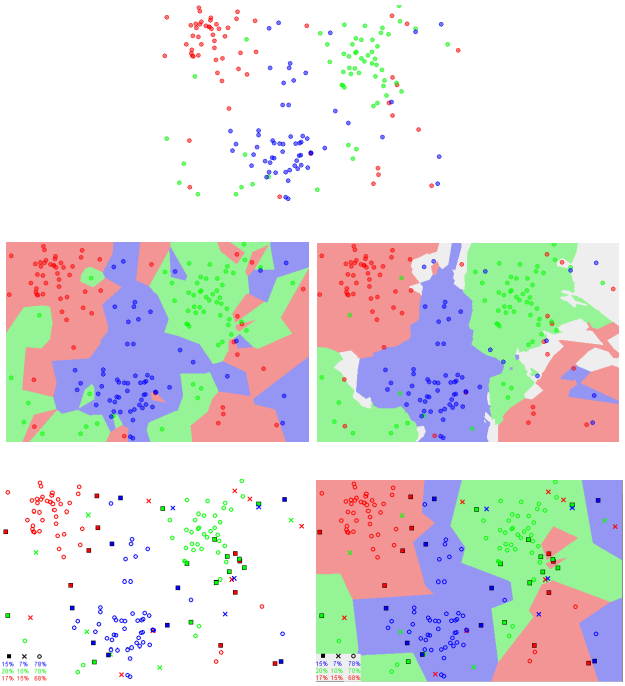

Figura 1.2: (1) Conjunto de datos (2) 1-NN (3) 5-NN (4) Conjunto reducido con CNN (5) 1-NN con prototipos extraídos del conjunto reducido CNN

Dado un conjunto X, CNN realiza la siguiente iteración:

1. Busca todos los elementos de X, buscando un elemento x cuyo prototipo más cercano en U tenga una clasificación distinta.

2. Elimina x de X y lo añade a U.

3. Repite la búsqueda hasta que no quedan más prototipos que añadir a U.

El algoritmo utiliza U en vez de X para realizar la clasificación. Los ejemplos que no son prototipos se denominan puntos "absorbidos".

1.2. Redes bayesianas

Las redes Bayesianas, red de Bayes, modelo bayesiano o modelo gráfico dirigido acíclico es un modelo gráfico probabilístico que representa un conjunto de variables aleatorias y sus dependencias condicionales vía un grafo acíclico dirigido o DAG (de sus siglas en Inglés: Directed Acyclic Graph). Por ejemplo, una red bayesiana puede representar la relaciones probabilísticas entre enfermedades y síntomas. Dados los síntomas, la red puede usarse para calcular las probabilidades de la presencia de varias enfermedades.

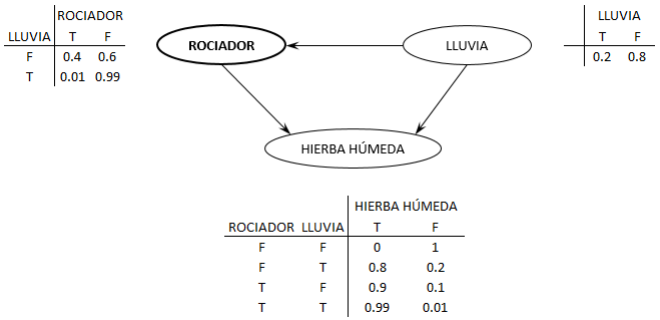

Figura 1.3: Red Bayesiana simple, que supone que el césped está húmedo por dos posibles eventos: riego o lluvia

Clasificador bayesiano ingenuo

Fundamentados en el teorema de Bayes se crearon los clasificadores bayesianos ingenuos, que reciben este nombre debido a las reglas de independencia entre variables utilizadas para simplificar el problema; es decir, que la presencia o ausencia de una característica no esta relacionada con la presencia o ausencia de otra característica. Por ejemplo, una fruta puede ser una naranja si es redonda, de color naranja y tiene una piel rugosa. El clasificador Bayes ingenuo considera estas características independientes a la probabilidad de que esta fruta es una naranja.

1.3. Redes neuronales

Las redes neuronales artificales o ANN (de sus siglas en Inglés: Artificial Neural Networks) son modelos computacionales inspirados en el sistema nervioso central de los animales y son capaces de aprender y reconocer patrones. Las ANN se representan generalmente como un sistema de "neuronas" interconectadas que pueden calcular valores desde sus entradas.

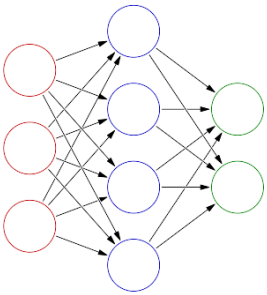

Figura 1.4: Ejemplo de red neuronal. En rojo las neuronas de entrada, en azul las neuronas ocultas y en verde las neuronas de salida

Por ejemplo, una ANN para reconocer escritura a mano se define como un conjunto de neuronas de entrada que pueden ser activadas por los píxeles de una imagen usada como entrada. La activación de estas neuronas se comunica, pesa y transforma por una función determinada por el diseñador de la red y se comunica al resto de la red. Este proceso se repite hasta que finalmente una neurona salida se activa, determinando así qué carácter se ha leído.

1.4. Máquinas de vector soporte

Las máquinas de vector soporte o SVMs (del sus siglas en Inglés: Support Vector Machines) aparecieron a principios de los noventa como un clasificador de margen óptimo en el contexto de la teoría de aprendizaje estadístico de Vapnik [44]. Desde entonces, las SVMs han sido aplicadas satisfactoriamente a problemas de análisis de datos del mundo real, a menudo

ofreciendo resultados mejores comparados con otras técnicas. Las SVMs operan en el ámbito de la teoría de regularización minimizando el riesgo empírico en un modo consistente. Una clara ventaja de los vectores soporte es que la solución dispersa a los problemas de clasificación y regresión son normalmente obtenidos: solo unas cuantas muestras hacen falta en la determinación de las funciones. Esto hace que sea más fácil aplicar SVMs a los problemas con gran cantidad de datos como el procesamiento de texto y las tareas bioinformáticas, convirtiéndolas en una metodología estándar. En el siguiente capítulo realizaremos un análisis sobre este tipo de clasificadores.

Capítulo 2

Máquinas de vector soporte (SVM)

Una SVM es una función que recoge un conjunto de entrada $\{(x_i, y_i) \in X \times Y\}_{i=1}^{n}$y su objetivo es "aprender" la relación existente entre los datos de entrenamiento x_i y las variables etiqueta y_i. El objetivo, en este contexto, es predecir que etiqueta tendrá un nuevo dato.

Ejemplos:

- X es un conjunto de matrices de 20 x 20 que representan las letras en mayúscula del abecedario. Y sería el conjunto de etiquetas {1, ..., 27}
- X es un conjunto $\mathbb{N}^{2000}$, es el espacio correspondiente al vocabulario utilizado en un foro. Y sería el conjunto de etiquetas {-1, 0, 1}, que indica si un mensaje es negativo, neutro o positivo.

- X es un conjunto $\mathbb{R}^{250}$, es el espacio correspondiente a la representación de los rostros almacenados en una base de datos. Y sería el conjunto de etiquetas {-1,+1}, que indica si esa persona puede acceder a una zona de seguridad o no.

El objetivo más común es predecir a qué clase pertenece cada dato y en la mayoría de los casos no es posible asumir una forma paramétrica para la distribución $p(x, y)$. Así, las soluciones para problemas de análisis de datos industriales ponen más énfasis en la aproximación algorítmica que en el modelo de los datos, lo que lleva a procedimientos de caja negra. Las redes neuronales constituyen un ejemplo de esta aproximación, ya que son lo suficiente potentes para aproximarse a funciones continuas con precisión arbitraria, sin embargo, los parámetros son muy difíciles de ajustar e interpretar y la inferencia estadística no es posible normalmente. Las SVMs alcanzan un compromiso entre ambas aproximaciones (paramétrica y no paramétrica): como clasificador lineal las SVMs estiman una función de decisión lineal, con la particularidad de que un mapeo previo de los datos en un espacio de dimensión superior puede ser necesario. Este mapeo se caracteriza por la elección de unas funciones conocidas como kernels.

A continuación veremos las bases de las SVMs para el problema de clasificación de dos clases (SVM biclase).

2.1. SVM biclase

Según [19] una SVM para reconocimiento de patrones es una función $f : \mathbb{R}^N \rightarrow \{\pm 1\}$ que utiliza un conjunto de datos de entrenamiento, es decir, patrones x_i N-dimensionales y unas clases etiquetadas como y_i,

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^N \times \{\pm 1\}$$

de forma que f clasificará correctamente nuevos ejemplos (x, y) si estos fueron generados bajo la misma distribución $P(x, y)$ que el conjunto de entrenamiento.

Consideremos un problema de clasificación como el de la figura 2.1(a). Supongamos que tenemos un mapeo Φ en un "espacio característico" tal que los datos que consideramos se vuelven linealmente separables como en la figura 2.1(b)

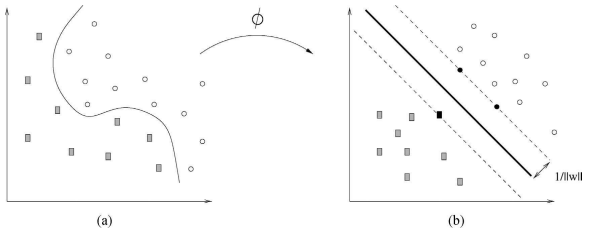

Figura 2.1: (a) Datos originales en espacio de entrada (b) Datos mapeados en espacio característico

Del número infinito de hiperplanos existentes que separan ambas clases, la SVM busca el que está más lejos de ambas clases (hiperplano de margen óptimo). Las clases típicamente se definen como {+1; -1}. Para ser más específicos, denota la posibi-

lidad de que una muestra mapeada por $\{(\Phi(x_i), y_i)_{i=1}^n\}$, donde $y_i \in \{-1, +1\}$indica las dos posibles clases. Denota como $w^T\Phi(x)+b = 0$ a cualquier hiperplano separador en el espacio mapeado de datos equidistante al punto más próximo de cada clase. Asumiendo separabilidad, podemos reescalar w y b de forma que $|w^T\Phi(x)+b| = 1$ para aquellos puntos de cada clase más cercanos al hiperplano, teniendo para cada $i \in \{1, ..., n\}$

$$w^T\Phi(x) + b \begin{cases} \geq +1, & si\ i = +1 \\ \leq -1, & si\ y_i = -1 \end{cases} \tag{2.1}$$

Tras el reescalado, la distancia al punto más cercano del hiperplano es $1/\|w\|$. Entonces, la distancia entre los dos grupos o margen es $2/\|w\|$. Para maximizar el margen, hay que solucionar el siguiente problema de optimización:

$$\begin{array}{cc} \text{mín}_{w,b} & \|w\|^2 \\ s.t. & y_i(w^T\Phi(x_i) + b \geq 1,\ i = 1, \dots, n \end{array} \tag{2.2}$$

donde el cuadrado de la norma de w se ha usado para hacer el problema cuadrático. Dada su convexidad, este problema no tiene un mínimo local. Consideremos la solución al problema (2) y denotémoslo como w^*y b^*. Esta solución determina el hiperplano en el espacio característico $D^*(x) = (w^*)^T\Phi(x) + b^* = 0$. Los puntos $\Phi(x_i)$ que satisfacen la igualdad $y_i((w^*)^T\Phi(x_i) + b^*) = 1$ se llaman vectores soporte (en la figura 2.1(b) están marcados en negro)

2.1.1. Mapeado kernel

En esta sección nos enfrentamos a uno de los problemas clave de la SVM: como usar $\Phi(x)$ para mapear los datos en un espacio dimensional superior. El procedimiento se justifica con el teorema de Cover [12], que garantiza que cualquier conjunto de datos se convierte en separable según la dimensión crece, aunque encontrar una transformación no lineal no es sencillo. Para conseguir esto, se utiliza un tipo de funciones llamadas kernels. Un kernel $K(x, y)$ es una función $K : X \times X \to \mathbb{R}$ para la que existe una función $\Phi : X \to Z$, donde Z es un espacio vector real, con la propiedad de que $K(x, y) = \Phi(x)^T \Phi(y)$. La función Φ es justo el mapeo de la figura 2.1. El kernel $K(x, y)$ actúa como producto escalar en el espacio Z. Típicamente en la literatura de SVM X y Z son llamados espacio de entrada y espacio característico respectivamente (ver figura 2.1)

$$
\begin{aligned}
(x \cdot y)^2 &= \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right)^2 \\
&= \left(\begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ \sqrt{2} y_1 y_2 \\ y_2^2 \end{pmatrix} \right) \\
&= (\Phi(x) \cdot \Phi(y))
\end{aligned}
$$

Cuadro 2.1: Ejemplo kernel

Los kernels más utilizados son:

- Kernel lineal $K(x, y) = x^T y$, corresponde con mapear la identidad.
- Kernel polinomial $K(x, y) = (c + x^T y)^d$, donde c y d son

constantes. Mapea los mensajes en un espacio dimensional finito.

- Kernel radial basis function (RBF) o Gaussiano $K(x,y) = e^{\frac{-\|x-y\|^2}{\sigma}}$ donde σ es una constante postitiva. Mapea los mensajes en un espacio dimensional infinito.

- Kernel sigmoide $K(x,y) = tanh(\kappa(x \cdot y) + \Theta)$ donde κ es una ganancia y Θ un desplazamiento.

Dado un Kernel K, podemos considerar que el set de funciones alcanzadas por una combinación lineal finita de la forma $f(x) = \sum_j \alpha_j K(x_j, x)$, donde $x_j \in X$ haciendo que la completitud del espacio vector es un kernel de espacio reproductivo de Hilbert o RKHS. Dado que $K(x_j, x) = \Phi(x_j)^T \Phi(x)$, las funciones $f(x)$ que pertenecen a un RKHS pueden expresarse como $f(x) = w^T \Phi(x)$, con $w = \sum_j \alpha_j \Phi(x_j)$. Si además $f(x) = 0$ describe un hiperplano en el espacio característico determinado por Φ(como el de la figura 2.1(b)). Sin la pérdida de generalidad, puede añadirse una constante b tomando la forma

$$f(x) = \sum_j \alpha_j K(x_j, x) + b \tag{2.3}$$

Esta ecuación permite usar Φ para mapear los datos en un espacio de dimensión superior: $f(x)$ puede evaluarse usando la expresión (3) en la que solo están involucrados los valores del kernel.

2.1.2. Método de regularización

Anteriormente, hemos explicado la situación ilustrada en la figura 2.1(b), dónde los datos están mapeados de forma linealmente separables, pero en general los datos no lo son. Esta situación se ilustra en la figura 2.2(a).

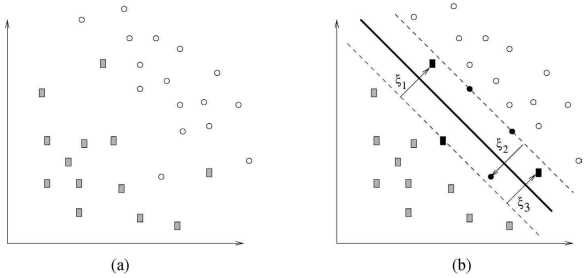

Figura 2.2: (a) Datos mapeados no separables en el espacio característico (b) Hiperplano normalizado para los datos en (a)

La SVM resuelve este problema encontrando una función f que minimiza el error empírico de la forma $\sum_{i=1}^{n} L(y_i, f(x_i))$, donde L es una función de pérdida particular y $(x_i, y_i)_{i=1}^{n}$ son los datos de muestra disponibles. Puede que existan infinitas soluciones, en cuyo caso el problema esta mal planteado. A continuación se muestra como la SVM soluciona este problema haciendo que la función de decisión que calcula la SVM sea única y la solución dependa de los datos de forma continua.

La función de pérdida específica L usada por la SVM es $L(y_i, f(x_i)) = (1 - y_i f(x_i))_+$, con $(x)_+ = max(x, 0)$. A esta función se la conoce como pérdida bisagra y está representada en la figura 2.2. Es cero para los puntos bien clasificados con $\mid f(x_i) \mid \geq 1$ y es

lineal en el resto de casos. Así, la función no penaliza los valores grandes de $f(x_i)$ con el mismo signo que y_i (entendemos por grande $\mid f(x_i) \mid \geq 1$).

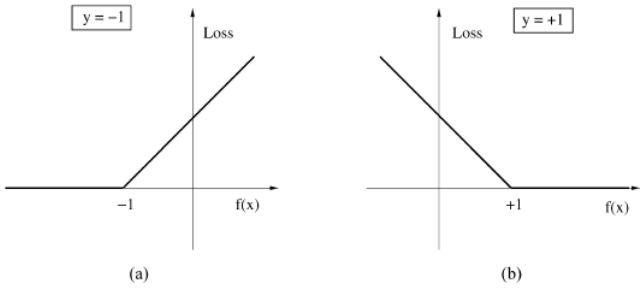

Figura 2.3: Función pérdida bisagra $L(y_i, f(x_i)) = (1 - y_i f(x_i))_+$ (a) $L(-1, f(x_i))$; (b) $L(+1, f(x_i))$

Este comportamiento satisface el hecho de que en problemas de clasificación solo se necesita una estimación del borde de la clasificación. Como consecuencia solo tomamos en cuenta los puntos que hacen $L(y_i, f(x_i)) > 0$ para determinar la función decisión.

Para alcanzar un buen planteamiento, las SVMs hacen uso de la teoría de la regularización, por la que se proponen varias aproximaciones similares [22][42][39]. La más utilizada es la regularización de Tikhonov [42], consistente en resolver el problema de optimización

$$\min_{f \in H_K} \frac{1}{n} \sum_{i=1}^{n} (1 - y_i f(x_i))_+ + \mu \parallel f \parallel_K^2 \tag{2.4}$$

donde $\mu > 0$, H_K es el RKHS asociado al kernel K, $\parallel f \parallel_K$ la

norma de f en el RKHS y x_i la muestra. Dado que f pertenece a H_K,toma la forma $f(\mathring{u})=\sum_j \alpha_j K(x_j, \mathring{u})$. Como se ha visto anteriormente, $f(x) = 0$ es un hiperplano en el espacio característico. Usando la propiedad reproductiva $\{K(x_j, \mathring{u}),\ K(x_l, \mathring{u})\}_K = K(x_j, x_l)$ (ver [3]), mantiene que $\|f\|_K^2 = \{f, f\}_K = \sum_j \sum_l \alpha_j \alpha_l K(x_j, x_l)$

En el problema (4) el escalar μ controla el compromiso de encontrar la solución f más ajustada a los datos (medido por L) y la capacidad de aproximación del espacio de la función a la que f pertenece (medido por $\|f\|_K$).

La solución al problema (4) tiene la forma $f(x)=\sum_{i=1}^n \alpha_i K(x_i, x)+ b$ donde x_i son los datos de muestra, un caso particular de (3). Es inmediato demostrar que $\|f\|_K^2 = \|w\|^2$, donde $w = \sum_i^n \alpha_i \Phi(x_i)$. Dado este último resultado, el problema (4) puede reescribirse como

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} (1 - y_i(w^T \Phi(x_i) + b)_+ + \mu \|w\|^2 \qquad (2.5)$$

El segundo termino de (5) coincide con el término de la función objetivo del problema (2). Los problemas (4) y (5) revisan algunos de los problemas principales de las SVMs tal y como están propuestas en 1.1: mediante el uso de kernels, el problema a priori de estimar una función de decisión no lineal en el espacio de entrada es transformado en un problema de estimación de pesos del hiperplano en el espacio característico a posteriori.

Debido a la función de pérdida bisagra, el problema (5) es no diferenciable, implicando dificultad para aplicar técnicas de optimización eficientes (ver [7, 34]. El problema (5) puede

suavizarse directamente formulándolo como (ver [28])

$$\begin{array}{ll} min & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ s.t. & y_i(w^T\Phi(x_i)+b) \geq 1-\xi_i \\ & i = 1,...,n \\ & \xi \geq 0 \end{array} \tag{2.6}$$

donde ξ_i son variables de holgura introducidas para evitar la no diferenciabilidad de la función de pérdida bisagra y $C = 1/(2\mu n)$. Esta es la formulación más utilizada de las SVM.

Las variables de holgura ξ_i permiten la violación de restricciones del problema (1), extendiendo el problema (2) a un caso no separable (el problema (2) no se puede solucionar para datos no separables). Las variables de holgura garantizan la existencia de una solución. La situación se muestra en la figura 2.2(b), que constituye una generalización de la figura 2.1(b). El problema (2) es un caso particular del problema (6). Para ser más específico, si los datos mapeados se vuelven separables, el problema (2) es equivalente al (6) cuando, en la solución, $\xi_i = 0$. Intuitivamente queremos resolver el problema (2) y, a la vez, minimizar el número de muestras no separables, es decir, $\sum_i \#(\xi_i > 0)$. La inclusión de este término otorga un problema combinacional no diferenciable, el termino de suavidad $\sum_{i=1}^{n}\xi_i$ aparece en sustitución.

Hemos deducido la formulación estándar de la SVM (6) utilizando la teoría de la regularización, garantizando que el error empírico para las SVMs converge según lo esperado cuando n tiende a infinito, y que las funciones de decisión obtenidas son estadísticamente consistentes, es decir, los hiperplanos de

decisión calculados no son ni arbitrarios ni inestables.

Usando la teoría de optimización estándar, se puede demostrar que (6) es equivalente a resolver

$$\begin{array}{ll} min_{\lambda} & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j y_i y_j K(x_i,x_j) - \sum_{i=1}^{n}\lambda_i \\ s.t. & \sum_{i=1}^{n} y_i\lambda_i = 0, \\ & 0 \leq \lambda_i \leq C,\ i = 1,...,n \end{array} \tag{2.7}$$

Las variables λ_i son multiplicadores de Lagrange asociados a las restricciones de (6). Este problema se conoce en la teoría de optimización como el problema dual de (6). Es convexo y cuadrático, teniendo asi que todo mínimo local es un mínimo global. En la práctica, este es el problema a resolver y para el que se han desarrollado métodos eficientes específicos.

Denotemos el vector λ^*como la solución al problema (7). Los puntos que satisfacen $\lambda^* > 0$ son el vector soporte. Puede demostrarse que la solución al problema 6 es $w^* = \sum_{i=1}^{n}\lambda_i^* y_i K(x_i, x^+)$ y

$$\begin{array}{c} b^* = -\frac{\sum_{i=1}^{n}\lambda_i^* y_i K(x_i,x^+)}{2} \\ +\frac{\sum_{i=1}^{n}\lambda_i^* y_i K(x_i,x^-)}{2} \end{array} \tag{2.8}$$

donde x^+ y x^- son respectivamente dos vectores soporte de las clases +1 y -1, asi como sus multiplicadores de Lagrange asociados λ^+ y λ^-, manteniendo que $0 < \lambda^+ < C$ y $0 < \lambda^- < C$.

La función de decisión que determina el hiperplano $(w^*)^T\phi(x)+ b^* = 0$ toma la forma

$$\begin{aligned} D^*(x) &= (w^*)^T\phi(x) + b^* \\ &= \sum_{i=1}^{n} \lambda_i^* y_i K(x_i, x) + b^* \end{aligned} \tag{2.9}$$

Las ecuaciones (8) y (9) demuestran que $D^*(x)$ está completamente determinada por la submuestra conformada por los vectores soporte, los únicos puntos de la muestra que cumplen $\lambda_i^* \neq 0$. Esta definición es coherente con la geométrica dada anteriormente.

2.1.3. Ejemplos

En el primer ejemplo consideremos un problema de clasificación de dos clases, donde cada clase tiene 1000 puntos generados por una distribución normal bivariada $N(\mu_i, I)$ con $\mu_1 = (0, 0)$ y $\mu_2 = (10, 10)$. Nuestro objetivo sera ilustrar el rendimiento de la SVM con este ejemplo y, en particular, el rendimiento del algoritmo para distintos valores del parámetro de regularización C en el problema (6). Se utiliza la función identidad de mapeo $\phi(x) = x$. La figura 2.4 (a) ilustra el resultado para C=1 (mismo resultado para $C > 1$). Hay tres vectores soporte y el hiperplano de margen óptimo de separación obtenido por la SVM es $1{,}05x + 1{,}00y - 10{,}4 = 0$. Para $C = 0{,}01$ se obtienen siete vectores soporte (figura 2.4 (b)) y la recta discriminante es $1{,}02x + 1{,}00y - 10{,}4 = 0$. Para $C = 0{,}00001$, se obtienen 1776 vectores soporte (88.8 % de la muestra, ver figura 2.4(c)) y el hiperplano separador es $1{,}00x + 1{,}00y - 10{,}0 = 0$. Cuanto más pequeño es C, más vectores soporte se obtienen. Esto se debe a que, en el problema (6),C penaliza el valor de las variables ξ_i, que determina el ancho de banda que contiene

los vectores soporte.

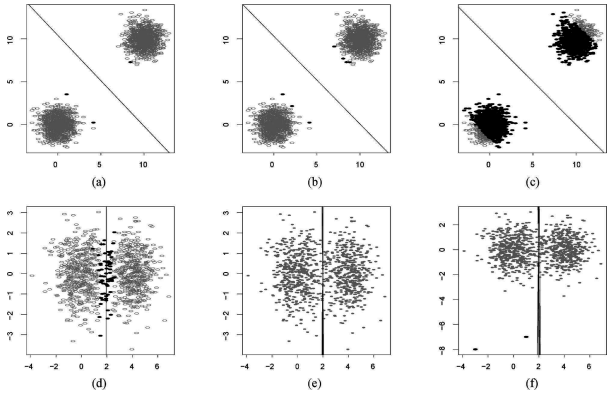

Figura 2.4: (a)-(c) Hiperplanos SVM para datos separables. Los vectores soporte son los puntos negros. (d)-(f) Hiperplanos SVM para datos no separables.

El segundo ejemplo es bastante similar al anterior, pero las muestras que corresponden a cada clase son no separables. Em este caso los vectores media de las dos nubes normales (500 puntos de datos en cada grupo) son $\mu_1 = (0, 0)$ y $\mu_2 = (4, 0)$, respectivamente. El error teórico de Bayes es 2.27 %. En teoría el hiperplano de separación normal (y óptimo) es $x = 2$, es decir, $0{,}5x + 0y - 1 = 0$. El hiperplano estimado por la SVM (con C=2) es $0{,}497x - 0{,}001y - 1 = 0$. El error en un conjunto de datos de 20000 puntos es 2.3 %. La figura 2.4(d) muestra el hiperplano estimado y los vectores soporte (puntos negros), que representan el 6.3 % de la muestra. Para mostrar el comportamiento del método cuando el parámetro C varía, la figura 2.4(e) muestra los hiperplanos de separación para 30 SVMs con C variando entre 0.001 hasta 10. Todos son bastante similares. Finalmente, la figura 2.4(f) muestra los

mismos 30 hiperplanos cuando los dos puntos subyacentes (en negro) añadidos a la nube izquierda. Dado que las funciones discriminantes estimadas por la SVM dependen solo de los vectores soporte, los hiperplanos no cambian.

2.2. SVM multiclase

Aunque por simplicidad se ha descrito una SVM biclase, en la práctica existen versiones multiclase que usan esquemas de votación para clasificar los nuevos datos en una categoría $C_1, \ldots, C_m$ (ver [25])

Existen tres tipos según [20]:

Uno contra todos (One-versus-all)

Es la primera implementación utilizada para clasificación SVM multiclase. Se construyen k clasificadores binarios, siendo k el número de clases. Construye k modelos (uno por cada clase). Cada clasificador i se entrena tomando los ejemplos de la clase i como positivos y el resto de ejemplos como negativos. Dado un dato de entrenamiento l, $(x_1, y_1), \ldots, (x_l, y_l)$, donde $x_i \epsilon \mathbb{R}^n$, $i = 1, \ldots, l$ y $y_i \epsilon \{1 \ldots k\}$ es la clase de x_i, la i-ésima SVM soluciona el siguiente problema:

$$\begin{array}{ll} \text{mín}_{w^i, b^i, \xi^i} & \frac{1}{2}(w^i)^T w^i + C \sum_{j=1}^{l} \xi_j^i \\ & (w^i)^T \phi(x_j) + b^i \geq 1 - \xi_j^i, \quad si\, y_i = i \\ & (w^i)^T \phi(x_j) + b^i \leq -1 + \xi_j^i, \quad si\, y_i \neq i \\ & \xi_j^i \geq 0,\, j = 1, \ldots, l, \end{array} \tag{2.10}$$

Tras resolver este problema, hay k funciones decisión:

$$\begin{gathered}(w^1)^T\phi(x)+b^1,\\ \vdots\\ (w^k)\phi(x)+b^k\end{gathered}$$

Podemos decir que x está en la clase con el mayor valor de la función decisión:

$$class\, of\, x \equiv argmax_{i=1,\ldots,k}((w^i)^T\phi(x)+b^i) \tag{2.11}$$

Uno contra uno (One-versus-one)

Este método construye $k(k-1)/2$ clasificadores donde cada uno se entrena con los datos de dos clases. Para los datos de entrenamiento de las clases i y j, resolvemos el siguiente problema de clasificación binario:

$$\begin{array}{ll}\text{mín}_{w^{ij},b^{ij},\xi^{ij}} & \frac{1}{2}(w^{ij})^Tw^{ij}+C\sum_t\xi_t^{ij}\\ & (w^{ij})^T\phi(x_t)+b^{ij}\geq 1-\xi_t^{ij}, \quad si\, y_t=i\\ & (w^{ij})^T\phi(x_t)+b^{ij}\leq -1+\xi_t^{ij}, \quad si\, y_t\neq i\\ & \xi_t^{ij}\geq 0,\end{array} \tag{2.12}$$

Hay diferentes formas de hacer las futuras pruebas de los $k(k-1)/2$ clasificadores construidos, aunque el más común es el sistema de votación sugerido en [15]: si $sign((w^{ij})^T\phi(x)+b^{ij})$ dice que x pertenece a la clase i, entonces el voto para esa clase se incrementa, si no, se incrementa la j. Entonces predecimos que

x pertenece a la clase con más votos. A esta estrategia se le conoce como "Max wins". Si dos clases tienen los mismos votos, se elige la que tiene menor índice.

Gráfico Acíclico Dirigido de Máquinas Vector Soporte (DAGSVM)

En la fase de entrenamiento, resolvemos los mismos $k(k-1)/2$ modelos que en "uno contra uno". En la fase de pruebas, en cambio, se usa un gráfico acíclico dirigido con raíz que tiene $k(k-1)/2$ nodos internos y k hojas. Cada nodo es una SVM binaria de las clases i y j. Dada una muestra x, comenzando en el nodo raíz, la función de decisión binaria se evalúa, moviéndose a izquierda o derecha según los resultados. Cuando alcanzamos una hoja, decidimos que la muestra pertenece a esa clase.

2.3. SVM de regresión

Hasta ahora hemos contemplado el algoritmo SVM como un algortimo de clasificación, pero también puede aplicarse a problemas de regresión [41]. El algoritmo tiene las mismas características que en clasificación: sigue habiendo una fase de entrenamiento que extrae un modelo de los datos y sigue utilizando kernels para calcular las distancias, pero utiliza una función de pérdida cuadrática (o aproximaciones a esta) igual que otras técnicas de mínimos cuadrados.

Supongamos un conjunto de datos $\{(x_1, y_1), \ldots, (x_\ell, y_\ell)\} \subset \chi \times \mathbb{R}$, donde χ denota el espacio de los patrones de entrada.

Por ejemplo, evolución de los precios de acciones, evolución de valor de cambio entre dos monedas... El objetivo es encontrar una función cuya desviación máxima de los objetivos y_i sea ε para los datos de entrenamiento y sea lo más suave posible.

Igual que con la SVM de clasificación, partimos de un problema de optimización convexo:

$$\begin{array}{ll} min & \frac{1}{2}\|w\|^2 \\ s.t. & \begin{cases} y_i - \langle w, x_i \rangle - b & \leq \varepsilon \\ \langle w, x_i \rangle + b - y_i & \leq \varepsilon \end{cases} \end{array} \tag{2.13}$$

Asumimos que existe una función que aproxima todos los pares (x_i, y_i) con precisión ε. Puede que no sea factible o queramos permitir algún error, por ello introducimos las variables de holgura ξ_i, ξ_i^*.

$$\begin{array}{ll} min & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ s.t. & \begin{cases} y_i - \langle w, x_i \rangle - b & \leq \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i & \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases} \end{array} \tag{2.14}$$

La constante $C > 0$ determina la penalización entre la suavidad de la función y la cantidad de errores (desviaciones mayores que ε). Esto corresponde con una función de pérdida.

La función de pérdida cuadrática tiene la forma $L(x) = C(t - x)^2$ siendo t el objetivo y C una constante que puede ser ignorada ($C = 1$).

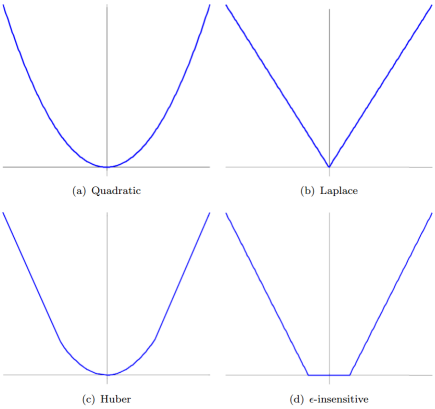

Figura 2.5: Funciones de pérdida

La función de la figura 2.5 (a) corresponde con la función de error convencional de mínimos cuadrados, (b) es la función de Laplace que es menos sensible a los valores atípicos, (c) es la función propuesta por Huber, una función robusta que tiene propiedades óptimas cuando se desconoce la distribución de los datos. Estas tres funciones no producen dispersión en los vectores soporte. Para solucionarlo, Smola propuso en [41] la función $\varepsilon - intensive$, representada en (d), como una aproximación a la función de Huber. Esta función es descrita por:

$$|\xi|_{\varepsilon} = \begin{cases} 0 & si\ |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & en\,otro\,caso \end{cases} \tag{2.15}$$

Con esta función de pérdida se penaliza a los valores que están a una distancia mayor que ϵ del valor medio de los puntos permitiendo obtener una SVM que realiza una regresión de datos en vez de una clasificación.

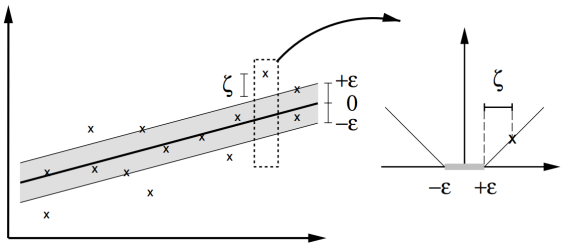

Figura 2.6: Margen suave de pérdida para SVM lineal

Capítulo 3

Paralelización de clasificadores con regiones de Voronoi

En los últimos años debido al incremento de los datos a analizar en los clasificadores, los algoritmos que se utilizan habitualmente están empezando a verse limitados debido a su falta de paralelismo. El algoritmo Sequential Minimal Optimization (o SMO) propuesto en [40] permite cierto grado de paralelismo y existen versiones paralelas implementadas en C y CUDA, pero no permite la distribución de datos en grandes centros de cálculo. Además es necesario que trabaje con el conjunto de entrenamiento completo para construir un modelo que permita una posterior clasificación, incrementando los requisitos de memoria del equipo que utilicemos para ejecutar el clasificador.

Teniendo en cuenta estas limitaciones se optó por buscar una alternativa que permitiera particionar el conjunto de entrenamiento usando regiones de Voronoi para permitir la computación paralela y distribuida del clasificador y disminuir la cantidad de memoria necesaria para clasificar datos en grandes conjuntos.

3.1. Generación de regiones de Voronoi y paralelización

Un diagrama de Voronoi es un método que divide el espacio en un número de regiones determinadas. La característica principal de las regiones es que cualquier punto dentro de ella está más cerca al mismo centroide que a otro que esté en otra región. Estas regiones son conocidas como regiones de Voronoi. Por definición, esta forma de particionar el espacio hace que al generar las regiones de Voronoi estemos creando una clasificación 1-NN de vecino más próximo.

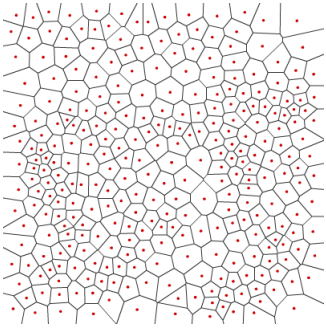

Figura 3.1: Ejemplo de espacio particionado en regiones de Voronoi

El procedimiento de paralelización de las SVM tiene su base en la división de cada una de las clases de entrenamiento en un número grande de regiones de Voronoi. En este trabajo se ha optado por fijar el número de regiones a calcular en cada clase, y encontrarlas utilizando un algoritmo de cluster k-medias.

3.1.1. El algoritmo *k*-medias

El algoritmo de k-medias nos permite particionar un conjunto grande de datos en k conjuntos más pequeños. Cada uno de estos conjuntos engloba a los datos que tienen una media más cercana entre ellos, buscando un centroide de Voronoi. Para este trabajo se implementó un programa en C que ejecuta el algoritmo k-medias en cada una de las clases y a continuación realiza una combinación entre los distintos pares de conjuntos de cada una de las clases para crear k_1 x k_2 combinaciones que servirán como conjuntos de entrenamiento para las SVMs en paralelo.

3.1.2. El algoritmo SMO y libSVM

El algoritmo utilizado para implementar el algoritmo de entrenamiento de las SVMs es SMO [40] implementado en libSVM [10]. El algoritmo SMO fue elegido para este trabajo porque las otras alternativas para entrenar SVMs (Chunking y Osuna) son más lentas y computacionalmente son más pesadas [40].

La versión del algoritmo SMO utilizada permite el uso de diferentes funciones kernel. Los detalles concretos sobre estas funciones se puenden consultar en 2.1.1.

3.2. El algoritmo pSVM

El algoritmo implementado en este trabajo como otros clasificadores tiene dos fases:

Entrenamiento

El entrenamiento en pSVM realiza una partición del espacio de entrada del conjunto de entrenamiento utilizando el algoritmo de k-medias. Hecho esto, se entrenan de manera paralela un número aleatorio de clasificadores. Cada clasificador toma como clases de entrada dos regiones de Voronoi (una de cada una de las clases). Cuando el algoritmo de entrenamiento finaliza, se realiza una clasificación de parte de los datos con los que ha sido entrenada la SVM para comprobar el correcto funcionamiento. El porcentaje de datos clasificados correctamente se usará para ponderar el voto de la SVM en el sistema de votaciones. Por ejemplo, si la SVM realiza correctamente un 75 % de las clasificaciones de la muestra de control, y al clasificar un nuevo dato lo coloca en la clase -1, el voto de esta SVM valdrá -0,75.

Clasificación

El proceso de clasificación de nuevos individuos se lleva a cabo mediante un proceso de votación, similar a los propuestos por [25] para los clasificadores multiclase. Así, cada clasificador emite su voto sobre la clase a que pertenece el individuo.

El voto de cada una de las SVM se pondera con un valor, entre -1 y 1 para el caso de dos clases, dependiendo del porcentaje de valores de entrenamiento que clasifique correctamente. Finalmente, el individuo es clasificado en la clase que recibe un mayor número de votos.

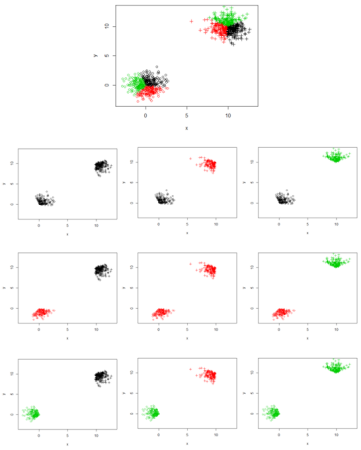

Figura 3.2: Ejemplo de particionado con k-medias en pSVM

3.2.1. Implementación en R

La primera implementación para estudiar la viabilidad del algoritmo se realizó en el entorno de programación estadística R. Este lenguaje ofrece una gran cantidad de librerías estadísticas incluyendo k-medias y libSVM [10] que permitió un desarrollo rápido de un primer prototipo del algoritmo pSVM. Este prototipo permitió demostrar que los resultados obtenidos con pSVM eran similares a los obtenidos con libSVM aunque no permitió realizar comparativas de tiempo, ya que R no inclu-

ye programación concurrente sin recurrir a ampliaciones no oficiales.

3.2.2. Implementación en C/C++

Una vez probado en R que el algoritmo era viable se decidió implementar el algoritmo en C/C++. Para ello se partió de la versión en C/C++ de la misma librería libSVM [10] utilizada en R y la librería de k-medias [24]. En esta versión del algoritmo se utilizaron threads para implementar el paralelismo por su sencillez para compartir datos y porque permitía aprovechar al máximo la arquitectura del ordenador en el que se realizarían las pruebas.

3.3. Comparativa entre libSVM y pSVM

A continuación se realiza una comparativa entre libSVM y pSVM implementado en C/C++ con 10 SVMs en paralelo utilizando los distintos kernels implementados en la libreria (lineal, polinomial, RBF y sigmoid), ya explicados en la subsección 2.1.1. Como la selección de los parámetros de cada kernel influye en la clasificación, se han utilizado los parámetros por defecto de cada kernel.

El experimento entrena y clasifica 10 veces un nube de datos fija, como la mostrada en la ilustración 3.3, formada por dos distribuciones de Poisson con 5000 puntos cada una. El objetivo de este experimento es identificar con cual de los kernels se obtienen valores más similares entre libSVM y pSVM, para

ello es necesario utilizar siempre la misma nube, ya que para libSVM la solución será única, pero como pSVM realiza un particionado del espacio de entrada con k-medias, que es no determinista, la solución obtenida puede variar, siendo necesario realizar un análisis estadístico de los datos obtenidos.

Para evaluar el rendimiento de los diferentes kernels, se han utilizado las métricas *precision*, *recall* y *accuracy* para la recuperación de información, véase [5] y [37]. Para ello, definimos *verdaderos positivos* (t_p), como el conjunto de puntos correctamente clasificados; *verdaderos negativos* (t_n), como el conjunto de puntos que correctamente no se han catalogado en cierta clase; *falsos positivos* (f_p) a los puntos que han sido incorrectamente clasificados en una categoría y *falsos negativos* (f_n) a aquellos puntos catalogados incorrectamente en una categoría que no era la suya. Como consecuencia, se definen las métricas:

- *Precision* para la clase C_i: Es la fracción de puntos asignados a la clase C_i que pertenecen a esa clase.

$$Precision = \frac{t_p}{t_p + f_p}$$

- *Recall* para la clase C_i: Es la fracción de puntos que deberían pertenecer a la clase C_i que se clasifican correctamente.

$$Recall = \frac{t_p}{t_p + f_n}$$

- *Accuracy*: Es la proporción de puntos correctamente catalogados sobre todos los mensajes disponibles.

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n}$$

El ordenador sobre el que se han realizado los experimentos tiene un procesador Intel i7 con 8GB de memoria RAM.

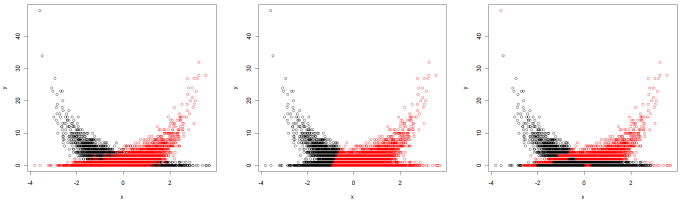

Figura 3.3: (1) Distribución Poisson 10k (2) Clasificación SMO (3) Clasificación pSVM

Kernel Lineal Como puede observarse en la figura 3.4 la fracción de puntos clasificados correctamente para libSVM es 56.68 %, clasificando correctamente un 55.6 % y 58.27 % de los puntos de la clase +1 y -1 respectivamente. Se puede ver en 3.5 cómo los resultados obtenidos encajarian con los resultados obtenidos por pSVM: 60.2 % de media de puntos clasificados correctamente, con un 59.43 % para la clase +1 y 63.45 % para la -1.

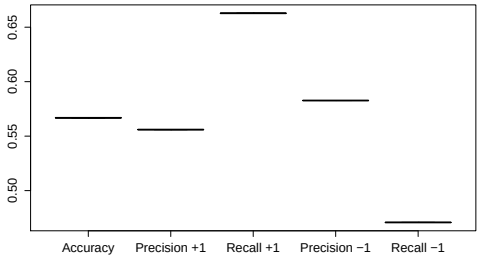

Figura 3.4: libSVM kernel Lineal

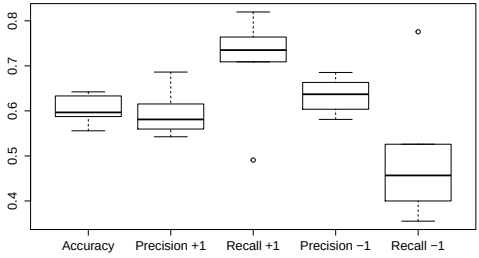

Figura 3.5: pSVM kernel Lineal

Kernel polinomial Como puede observarse en la figura 3.6 la fracción de puntos clasificados correctamente es 74.52 %, clasificando correctamente un 78.30 % y 71.63 % de los puntos de la clase +1 y -1 respectivamente. Se puede ver en 3.7 cómo los resultados obtenidos por libSVM son ligeramente superiores a los resultados obtenidos por pSVM: 63.42 % de media de puntos clasificados correctamente, con un 64.24 % para la clase +1 y 63.56 % para la -1.

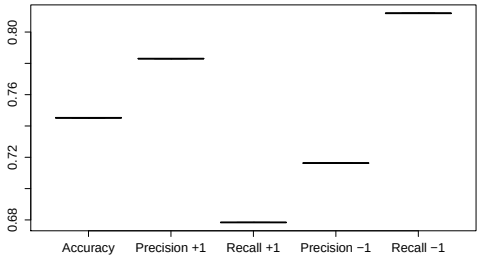

Figura 3.6: libSVM kernel Polinomial

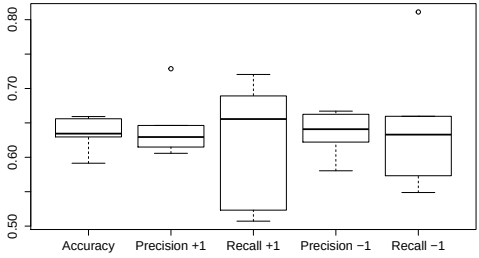

Figura 3.7: pSVM kernel Polinomial

Kernel RBF Como puede observarse en la figura 3.8 la fracción de puntos clasificados correctamente es 75.50 %, clasificando correctamente un 74.08 % y 77.09 % de los puntos de la clase +1 y -1 respectivamente. Se puede ver en 3.9 cómo los resultados obtenidos por libSVM son ligeramente superiores a los resultados obtenidos por pSVM: 59.47 % de media de puntos clasificados correctamente, con un 56.83 % para la clase +1 y 77.80 % para la -1.

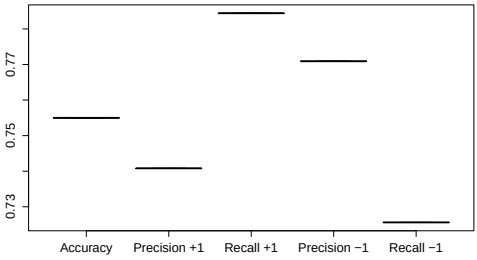

Figura 3.8: libSVM kernel RBF

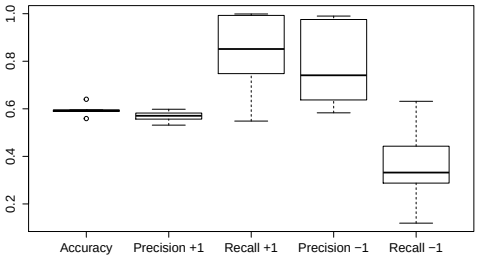

Figura 3.9: pSVM kernel RBF

Kernel sigmoid Como puede observarse en la figura 3.10 la fracción de puntos clasificados correctamente es 27.38 %, clasificando correctamente un 27.41 % y 27.35 % de los puntos de la clase +1 y -1 respectivamente. Se puede ver en 3.11 cómo los resultados obtenidos por libSVM son bastante inferiores a los resultados obtenidos por pSVM: 61.70 % de media de puntos clasificados correctamente, con un 57.95 % para la clase +1 y 73.57 % para la -1.

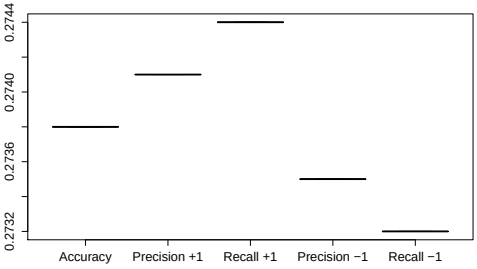

Figura 3.10: libSVM kernel Sigmoid

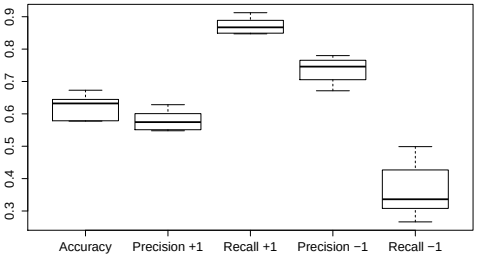

Figura 3.11: pSVM kernel Sigmoid

Como se puede observar en los experimentos, el único kernel que ofrece resultados similares entre las versiones secuencial y paralela es el kernel lineal, por lo que se utilizará a partir de este punto para realizar todas las comparativas de rendimiento entre ambas versiones de la libreria.

3.4. Otras técnicas

A continuación se presentan algunas alternativas a la metodología presentada en este capítulo:

3.4.1. Una mezcla paralela de SVMs para problemas de gran escala

En [11] se plantea un algoritmo de paralelización. En el artículo los autores detallan un algoritmo que divide el problema original en subproblemas aleatorios sin tener en cuenta la distribución de los datos.

El modelo propuesto por estos autores tiene dos pasos principales. En uno de ellos (entrenar cada SVM separadamente con subconjuntos calculados previamente) la paralelización es posible, pero en el otro paso (entrenar lo que ellos llaman "gater" para minimizar una función de pérdida en el conjunto de entrenamiento completo) la paralelización no es posible. Este paso no paralelizable es crucial en el buen rendimiento del método porque se usa para calcular los pesos (importancia dada a cada SVM previamente calculada). Dado que el artículo está basado en el juicio de expertos, el cálculo de estos pesos no puede ser evitado. De hecho, los autores explícitamente proponen ejecutar unas cuantas iteraciones en una red neuronal sobre el conjunto de entrenamiento completo para calcular una solución aproximada (explican esto después de la presentación de los pasos del algoritmo en la sección 3 del artículo).

Diferencias con pSVM:

1. El proceso de paralelización de este artículo es explícitamente paralelización software, mientras que pSVM propone una paralelización hardware (para usar en FPGAs). Es importante remarcar que el diseño de un algoritmo paralelo difiere dependiendo de la plataforma de implementación. La paralelización software no suele tener fuertes restricciones de almacenamiento de datos, mientras que en hardware se tiene que tener muy en cuenta. El problema con una posible implementación en FPGA del algoritmo propuesto en este artículo es el siguiente.

2. El algoritmo propuesto en este artículo no es apropiado para implementar en FPGA (hardware), por el número de datos que necesita almacenar (el conjunto de entrenamiento completo), haciendo obligatorio el uso de DRAM externa (memoria) disminuyendo críticamente la aceleración o speedup que se obtendría con esta implementación. Nuestra propuesta tiene este inconveniente en cuenta y solo necesita para la implementación hardware el almacenamiento de pequeñas partes del conjunto de entrenamiento.

3.4.2. Clasificación con kernels diádicos discriminantes de apoyo

En la patente [33] los autores presentan un nuevo algoritmo de clasificación no paralelizable. La patente no discute ni reivindica nada relativo a la posible paralelización del algoritmo que además no está basado en SVM, sino en las técnicas conoci-

das como “boosting”. Estas técnicas aunque de la familia de las técnicas basadas en “kernels de Mercer” al igual que las SVM, difieren en su funcionamiento pues asignan pesos a las muestras según su importancia en la clasificación que se va a realizar.

Diferencias con pSVM:

1. La metodología que se propone en esta tesis difiere claramente de la propuesta en la patente, ya que la nuestra permite la clasificación de más de dos clases y además es implementable en dispositivos electrónicos (hardware) de manera paralelizable. Sin embargo esta patente requiere calcular pesos para las muestras de entrenamiento, es decir, requiere almacenar todas las muestras con el correspondiente problema de almacenamiento ya discutido en el artículo anterior y algunos de los que se discuten a continuación.

2. Finalmente cabe reseñar que en la propia patente se indica que el “boosting lleva a cabo una distribución que evoluciona iterativamente” (“Boosting maintains an iteratively evolving distribution, i.e., weights D(i), over the training samples based on the difficulty of classification”). Es decir, el cálculo de los pesos se realiza mediante el uso de algoritmos iterativos basados en bucles, cuyo número de iteraciones no está predefinido, haciendo difícil su implementación eficiente en hardware.

3.4.3. Estrategia de votación de seguridad mayoritaria para SVMs modulares y paralelas

En [48] se presenta un algoritmo de paralelización. En el artículo los autores detallan un algoritmo que divide el problema original en subproblemas aleatorios sin tener en cuenta la distribución de los datos. El proceso de paralelización de este artículo es explícitamente paralelización software, mientras que nuestra patente propone una paralelización hardware (para usar en FPGAs). Es importante remarcar que el diseño de un algoritmo paralelo difiere dependiendo de la plataforma de implementación. La paralelización software no suele tener fuertes restricciones de almacenamiento de datos, mientras que en hardware se tiene que tener muy en cuenta.

El problema con una posible implementación en FPGA del algoritmo propuesto en este artículo es el siguiente:

1. En este artículo, como dicen los autores, "the limitation of this proposed CMV-SVMs lies in the necessity of storing all the training samples to evaluate the classification confidence for novel inputs" (citado de las conclusiones del artículo). Este es un inconveniente muy importante. Con la metodología que nosotros proponemos solo los vectores soporte correspondientes a cada par de regiones de Voronoi son necesarios y de hecho, son los únicos datos almacenados. En la práctica, normalmente los vectores soporte corresponden con menos del 10 % de conjunto de entrenamiento, siendo la mayoría de las veces un porcentaje por debajo del 5 %

2. El algoritmo propuesto en este artículo no es apropiado para implementar en FPGA (hardware), por el número de datos que necesita almacenar (el conjunto de entrenamiento entero), haciendo obligatorio el uso de DRAM externa (memoria) disminuyendo críticamente la aceleración o speedup que se obtendría con esta implementación. Nuestra propuesta tiene este inconveniente en cuenta y solo necesita para la implementación hardware el almacenamiento de pequeñas partes de las muestras.

3.4.4. Un algoritmos de entrenamiento paralelo de SVM en problemas de clasificación de gran escala

En [23] se presenta un algoritmo en cascada.

Diferencias con pSVM:

1. El proceso de paralelización en cascada que muestra este artículo es explícitamente software, mientras que el propuesto en nuestra patente es paralelización hardware (FPGA). Es importante señalar que el diseño de un algoritmo difiere dependiendo del entorno de implementación. La paralelización software no está limitada por la iteratividad derivada de los bucles, mientras que en hardware se necesita saber de antemano el número de iteraciones involucradas en el algoritmo para realizar un pipeline eficiente.

2. Los autores explican en el artículo que pudieran ser necesarias varias iteraciones del algoritmo para alcanzar el

óptimo global. Esto implica que la implementación hardware no sería posible pues en el caso hardware se necesita saber de antemano el número de iteraciones involucradas en el algoritmo para realizar un pipeline eficiente.

3. Desde un punto de vista puramente hardware, un algoritmo recursivo es imposible de implementar. De hecho, como se espera, el artículo propone una implementación puramente software. En el artículo se explica que el algoritmo es paralelizable en arquitecturas distribuidas tipo cluster (es decir paralelizable en software) y en ningún caso hablan de posible paralelización embebidas en dispositivos electrónicos como los propuestos en nuestra patente.

3.4.5. Un método de cascada para reducir el tiempo de entrenamiento y el número de vectores soporte

En [45] se presenta un algoritmo en cascada para reducir los tiempos de entrenamiento. Los autores dicen que el problema principal se divide recursivamente en subproblemas más pequeños hasta que el tamaño de estos subproblemas es manejable. La recursividad depende fuertemente del tamaño del conjunto de entrenamiento y consecuentemente el número de iteraciones en los pasos recursivos no puede ser determinado de antemano, haciendo imposible una implementación hardware eficiente.

Diferencias con pSVM:

1. El proceso de paralelización en cascada que muestra este artículo es explícitamente software, mientras que el propuesto en nuestra patente es paralelización hardware (FPGA). Es importante señalar que el diseño de un algoritmo difiere dependiendo del entorno de implementación. La paralelización software no está limitada por la iteratividad derivada de los bucles, mientras que en hardware se necesita saber de antemano el número de iteraciones involucradas en el algoritmo para realizar un pipeline eficiente.

2. Desde un punto de vista puramente hardware, un algoritmo recursivo es imposible de implementar. De hecho, como se espera, el artículo propone una implementación puramente software.

3.4.6. Comparación de métodos en cascada y paralelo para el entrenamiento de SVM en problemas de gran escala

En [6] presentan dos algoritmos: min-max-modular SVM (M3-SVM) y cascade SVM (C-SVM).

Diferencias con pSVM:

1. El artículo describe un método en cascada (C-SVM) para reducir los tiempos de entrenamiento. El autor indica que el problema principal tiene que dividirse recursivamente en subproblemas más pequeños hasta que el tamaño de estos subproblemas es manejable. La recursividad

depende fuertemente del tamaño del conjunto de entrenamiento y consecuentemente el número de iteraciones en los pasos recursivos no puede ser determinado de antemano, haciendo imposible una implementación hardware eficiente.

2. Respecto al algoritmo M3-SVM que también se compara, es conocido (ver [27]) que "para un conjunto de gran tamaño y alta dimensión, el rendimiento de M3-SVM será pobre y el coste de tiempo se incrementará, por las muchas redundancias o características irrelevantes que degradarían el rendimiento de base de los clasificadores tanto en velocidad como en precisión de predición". Esto hace imposible una implementación hardware eficiente.

3. Desde un punto de vista puramente hardware, un algoritmo recursivo es imposible de implementar. De hecho, como se espera, el artículo propone una implementación puramente software.

3.5. Conclusiones

El algoritmo pSVM diseñado con la metodología divide y vencerás consigue una aceleración cercana a la lineal y unos resultados similares a la versión del algoritmo estándar, además de permitir ejecutar problemas más grandes. Debido a que pueden limitarse los puntos de los clusters generados con k-medias, permite una implementación hardware en FPGA cuyo diseño será presentado en el siguiente capítulo.

También se han presentado algunos algoritmos paralelos para realizar entrenamiento de SVMs, aunque ninguno de ellos permite realizar el entrenamiento en paralelo en un dispositivo hardware.

Capítulo 4

Implementación hardware

4.1. FPGAs

Las FPGA (Field-Programmable Gate Array) son dispositivos semiconductores que incluyen bloques de lógica cuyas interconexiones y funcionalidad es programable. La lógica programable puede reproducir funciones tan sencillas como las llevadas a cabo por una puerta lógica, un sistema combinacional o complejos sistemas en un chip (SoC). Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan las FPGA incluyen a los DSP (procesamiento digital de señales), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de

visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, entre otras. Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

Existe código fuente disponible (bajo licencia GNU GPL) de sistemas como microprocesadores, microcontroladores, filtros, módulos de comunicaciones y memorias entre otros. Estos códigos se llaman cores.

4.1.1. Breve historia de las FPGAs

Las FPGAs son inventadas en el año 1984 por Ross Freeman, co-fundador de Xilinx, y surgen como una evolución de los CPLDs (Complex Programmable Logic Device).

Tanto los CPLDs como las FPGAs contienen un gran número de elementos lógicos programables. La densidad de los elementos lógicos programables en puertas lógicas equivalentes (numero de puertas NAND equivalentes que podríamos programar en un dispositivo) en un CPLD es del orden de decenas de miles de puertas lógicas equivalentes y en una FPGA del orden de cientos de miles hasta millones de ellas.

Aparte de las diferencias en densidad entre ambos tipos de dispositivos, la diferencia fundamental entre las FPGAs y los CPLDs es su arquitectura. La arquitectura de los CPLDs es más rígida y consiste en una o más sumas de productos programables cuyos resultados van a parar a un número reducido de biestables síncronos (también denominados flip-flops). La

arquitectura de las FPGAs, por otro lado, se basa en un gran número de pequeños bloques utilizados para reproducir sencillas operaciones lógicas, que cuentan a su vez con biestables síncronos. La enorme libertad disponible en la interconexión de dichos bloques confiere a las FPGAs una gran flexibilidad.

Otra diferencia importante entre FPGAs y CPLDs es que en la mayoría de las FPGAs se pueden encontrar funciones de alto nivel (como sumadores y multiplicadores) embebidas en la propia matriz de interconexiones, así como bloques de memoria.

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de las FPGA con microprocesadores y periféricos relacionados para formar un ńSistema programable en un chipż. Ejemplo de tales tecnologías híbridas pueden ser encontradas en los dispositivos Virtex-II PRO y Virtex-4 de Xilinx, los cuales incluyen uno o más procesadores PowerPC embebidos junto con la lógica del FPGA. Otra alternativa es hacer uso de núcleos de procesadores implementados haciendo uso de la lógica del FPGA. Esos núcleos incluyen los procesadores MicroBlaze y PicoBlaze de Xilinx, Nios y Nios II de Altera, y los procesadores de código abierto OpenRISC.

Las FPGA modernas soportan la reconfiguración parcial del sistema, permitiendo que una parte del diseño sea reprogramada, mientras las demás partes siguen funcionando. Este es el principio de la idea de la "computación reconfigurable", o los "sistemas reconfigurables".

4.1.2. Arquitectura de las FPGAs

La estructura típica de una FPGA mostrada en la figura 4.1 consiste en una matriz de dos dimensiones de recursos hardware dispuestos para la implementación de cualquier sistema digital. Se observan tres estructuras importantes:

- Bloques lógicos configurables (CLB): son los elementos más importantes de la FPGA, debido a que son los elementos que implementan las funciones lógicas. Se encuentran compuestos por las LookUp Table (LUTs) que sirven para implementar funciones lógicas mediante memoria, multiplexores y biestables para la implementación de sistemas secuenciales. La dimensión de un CLB depende del fabricante y la familia a la que pertenezca la FPGA.

- Bloques de entrada salida (IOB): son los bloques encargados de conectar la lógica interna de la FPGA con los pines exteriores de la misma. Se componen de multiplexores, biestables y otros elementos que en conjunto forman una estructura totalmente configurable mediante memoria, y determinan las características de los puertos de entrada/salida requeridos en un diseño digital.

- Bloques de interconexión programables (PIB): son elementos que se encargan de conectar las distintas líneas de comunicación dentro de la FPGA, permitiendo a los CLB e IOB conectarse unos con otros e implementar funciones lógicas más complejas. De igual forma que los

CLB e IOB, su configuración depende de una memoria RAM dedicada.

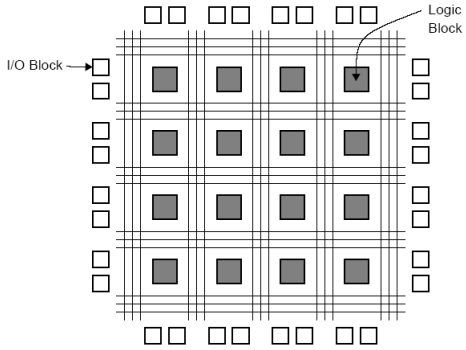

Figura 4.1: Arquitectura genérica de una FPGA

Adicionalmente, hoy en día podemos encontrar dentro de una FPGA componentes que facilitan la implementación de sistemas para múltiples aplicaciones, tales como memorias, multiplicadores, módulos para comunicaciones, DSP, procesadores... Todos estos permiten el diseño de sistemas en chip (SoC), en los que múltiples componentes de procesamiento, almacenamiento y comunicaciones son puestos en un mismo circuito integrado.

4.1.3. Bitstream de configuración

Las FPGAs al ser dispositivos programables mediante memoria requieren de un paquete de bits que establezcan su configuración. Dichos bits suelen almacenarse en un archivo de configuración *bitstrean* que determina el funcionamiento de un dispositivo programable y que es copiado directamente en

la memoria de configuración. Actualmente, los *bitstreams* para las FPGA pueden ser parciales o completos, dependiendo de si se requiere cambiar la configuración de una parte o de toda la FPGA respectivamente. Para que se pueda realizar la configuración parcial es necesario que el dispositivo soporte esta tecnología.

De igual forma, según el fabricante, encontramos múltiples interfaces de configuración de la FPGA. La interfaz serie, la paralela SelectMAP, el puerto interno de configuración (ICAP) son algunos ejemplos. Algunas soportan variaciones para configurar múltiples dispositivos, para realizar configuración desde microcontroladores, PROMs, etc.

4.1.4. Diseño electrónico de FPGAs

El diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL o Hardware Description Language (lenguajes de descripción de hardware). Los HDLs más utilizados son VHDL y Verilog.

Actualmente cualquier proceso de ingeniería dispone de un soporte software que asiste al ingeniero de aplicaciones o sistemas en el desarrollo de sistemas complejos. Los sistemas electrónicos reconfigurables del tipo FPGA son un buen ejemplo de la complejidad que se puede alcanzar, esta complejidad no sería abarcable sin la ayuda de un entorno con herramientas que

asistan en el proceso de diseño, simulación, síntesis del resultado y configuración del hardware. Un ejemplo de un entorno de este tipo es el software de la empresa Xilinx denominado ISE (Integrated Software Environment), con el cuál se puede seguir todo el flujo de desarrollo de un sistema FPGA.

Diseño

El diseño de una FPGA, al igual que el de un programa es la pieza clave para el funcionamiento correcto y eficiente del sistema. EL diseño de puede hacer de tres formas: mediante esquemas (representación gráfica de las puertas lógicas, útil para pequeños sistemas pero ineficiente para grandes sistemas), mediante grafos de estado. De cualquiera de estas dos formas es necesaria una traducción a un HDL. La tercera forma de realizar el diseño es realizarlo directamente en un HDL.

Síntesis

Una vez realizado el diseño, este se lleva al proceso de síntesis. En este proceso, el diseño en HDL se compila para crear las unidades funcionales que tienen que configurarse en la FPGA. Una vez terminada la compilación, las unidades funcionales se conectan en el proceso de enrutado siguiendo las instrucciones del diseño. Una vez el diseño ha sido enrutado se genera un fichero de configuración para la FPGA, que configura la lógica programable para que funcione según la especificación del diseño.

El proceso de síntesis es computacionalmente pesado y requiere de una gran cantidad de tiempo llevarlo a cabo, de forma que es necesario realizar verificaciones del diseño y comprobar su correcto funcionamiento antes de sintetizarlo, utilizando el proceso de simulación.

Simulación

Con la simulación se puede comprobar que el diseño realizado es correcto. Este proceso se utiliza para detectar y corregir errores más fácilmente que en la configuración final de la FPGA porque permite observar el comportamiento de cada señal y módulo funcional definidos en la fase de diseño.

Además de las herramientas de simulación tradicionales, Xilinx ha creado una herramienta de verificación de la configuración de la FPGA llamada ChipScope que permite visualizar las señales como se haría en los procesos de simulación, pero la diferencia es que estas señales son generadas realmente por la FPGA. Esta herramienta se puede utilizar para detectar errores que no son detectables en simulación.

Además del entorno de diseño ISE, existe el entorno de desarrollo para sistemas embebidos EDK o XPS, con el que se facilitan las labores del desarrollador, ya que incluye en su interfaz gráfica todas la herramientas necesarias para programar la lógica reconfigurable de las FPGAs. Con EDK se pueden realizar los procesos de diseño y síntesis de forma sencilla.

4.2. Xilinx University Program XUPV5-LX110T

Para realizar la implementación de los algoritmos SMO y pSVM en sistemas embebidos se ha utilizado la plataforma de prototipado Xilinx Virtex 5 XUPV5-LX110T. Esta placa tiene una FPGA XC5VLX110T, dos módulos de 32 MB de memoria PROM, 256 MB de memoria RAM DDR2, ethernet 10/100/1000, controlador USB e interfaz PCI Express x1 entre otras características.

4.3. Implementación de SMO clásico en FPGA

El diseño del sistema consiste en un Microblaze con memorias caché de 4KB, 256 MB de RAM DDR y un conector PCI-E (utilizado para comunicaciones con el ordenador), todo conectado a través de un bus PLB (Figura 4.2).

Entrenamiento

El ordenador envía los datos de entrenamiento al sistema embebido y este ejecuta el algoritmo estándar de entrenamiento. Una vez finalizado el entrenamiento, el sistema embebido devuelve el modelo obtenido para poder utilizarlo posteriormente en clasificación.

Clasificación

El ordenador carga un modelo de entrenamiento en el sistema embebido y a continuación comienza a enviar los datos a clasificar. Una vez clasificados los datos utilizando el modelo deseado, el sistema embebido devuelve a que clase pertenece cada uno de los puntos.

Conclusiones

La implementación de este algoritmo sobre un Microblaze para su utilización en sistemas embebidos demostró que es altamente ineficaz debido al alto consumo de recursos en la FPGA, funcionando únicamente en nubes de datos muy pequeñas y con mucha separación entre clases.

4.4. Implementación en la FPGA de la versión en serie de pSVM.

El sistema de la FPGA está compuesto por un procesador Microblaze con memorias caché de 4 KB, 256 MB de RAM DDR y un conector PCI-E (utilizado para comunicaciones con el ordenador), todo conectado a través de un bus PLB (Figura 4.2).

Entrenamiento

En el PC se ejecuta el programa que realiza la combinación de los conjuntos generados por k-medias y genera los conjun-

tos de entrenamiento. El algoritmo comienza escribiendo uno de los conjuntos de entrenamiento en la memoria DDR de la FPGA y se notifica al Microblaze para que comience la ejecución del SMO. Cuando el algoritmo de entrenamiento finaliza, se realiza una clasificación de parte de los datos con los que ha sido entrenada la SVM para comprobar el correcto funcionamiento. Como se ha indicado anteriormente, el porcentaje de datos clasificados correctamente se usará para ponderar el voto de la SVM en el sistema de votaciones. Por ejemplo, si la SVM realiza correctamente un 75 % de las clasificaciones de la muestra de control, y al clasificar un nuevo dato lo coloca en la clase -1, el voto de esta SVM valdrá -0,75. Cuando la SVM finaliza la prueba, el sistema devuelve al PC la SVM y su ponderación; y se comienza a entrenar la siguiente.

Cuando todas las SVMs (k_1 x k_2) han sido entrenadas, el sistema finaliza el entrenamiento y permite la clasificación de nuevos datos utilizando los modelos guardados durante el entrenamiento.

Clasificación

Para clasificar nuevos datos, una SVM se cargan en la memoria DDR, es restaurada por el Microblaze y clasifica los datos nuevos que se le envíe desde el PC. Una vez finaliza la SVM de clasificar los nuevos datos retorna la clasificación de cada uno de ellos ponderada y se procede a cargar la siguiente SVM para que comience la clasificación de los datos.

Cuando todas las SVMs han finalizado, se procede al proceso de votación, que consiste en sumar la clasificación ponderada

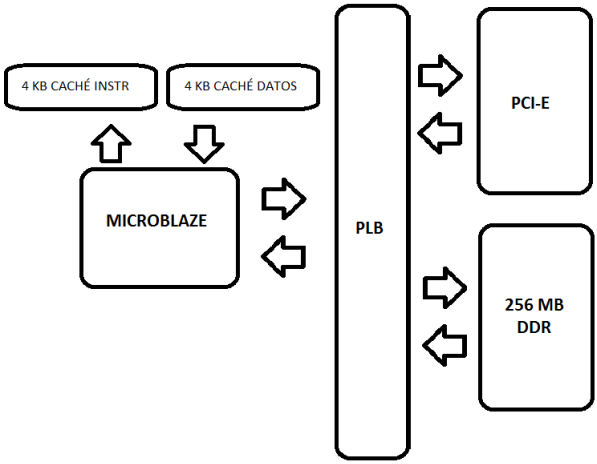

Figura 4.2: Diseño del sistema secuencial de la FPGA

de cada una de las SVMs. Si para un dato el resultado es mayor que cero, pertenecerá a la clase +1, si por el contrario es menor que cero, pertenecerá a la -1. Cuando el proceso de votación finaliza, se devuelve al PC la clasificación general de cada uno de los datos.

4.5. Implementación en la FPGA de la versión paralela de pSVM

Como el entrenamiento de distintas SVMs se realiza con datos distintos e independientes entre sí, el proceso de entrenamiento y clasificación de varias SVMs puede realizarse en paralelo, por lo que se construyó un sistema con multiples Microblaze.

Cada uno de los Microblazes tiene acceso a una zona de me-

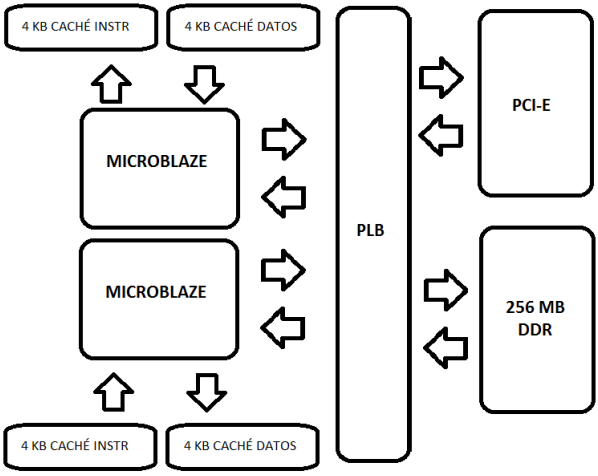

Figura 4.3: Diseño del sistema paralelo de la FPGA

moria exclusiva en la que recibirá los datos para realizar el entrenamiento y a una zona de memoria compartida en la que se recibirán los datos de clasificación.

Entrenamiento

En esta versión del sistema, el PC mantiene una estructura con la disponibilidad de los Microblazes. Cuando un Microblaze queda libre, el PC carga los datos de una SVM a la memoria asociada de ese Microblaze para que comience a realizar el entrenamiento al igual que en la versión secuencial. Mientras siga habiendo Microblazes disponibles, el PC seguirá cargando datos en sus memorias asociadas.

Una vez el entrenamiento de todas las SVMs ha finalizado y se ha ponderado su voto puede comenzar el proceso de clasifi-

cación.

Clasificación

La clasificación se realiza de modo similar a la versión secuencial. El PC carga los datos a clasificar en una zona de memoria compartida y se cargan las SVMs en los distintos Microblazes. Los Microblazes clasifican los datos en paralelo y votan para cada uno de los datos. Cuando todas las SVMs han finalizado el proceso de clasificación, se suman todos los votos ponderados, al igual que en la versión secuencial, y se clasifica cada dato.

4.6. Comparativas entre versiones FPGA del algoritmo pSVM

A continuación se realiza una comparativa entre las dos versiones de la FPGA del algoritmo pSVM. El experimento realizado para comparar las versiones ha sido el mismo que en la sección 3.3 del capítulo anterior .

El sistema secuencial de la FPGA consiguió un tiempo medio de entrenamiento de 79.58 segundos, la versión paralela con dos Microblazes tardó 43.27 segundos. Se observa que, al igual que en las comparativas entre el SMO y pSVM, debido a la independencia de los datos, el incluir más procesadores para realizar el entrenamiento de las SVMs hace que disminuya el tiempo de entrenamiento de forma casi lineal.

	Entrenamiento	Clasificación	Clasificados correctamente
FPGA x1	79.58	93.47	67.53 %
FPGA x2	43.27	57.11	67.38 %

Cuadro 4.1: Comparativa de tiempo (segundos) monoprocesador vs multiprocesador

Respecto al tiempo de clasificación, la versión secuencial de la FPGA tardó 93.47 segundos frente a los 57.11 segundos de la versión con dos Microblazes. Se observa que el tiempo de clasificación de los datos disminuye también de manera casi lineal debido a la independencia de los datos al ser clasificados.

4.7. Conclusiones

En este capítulo se ha diseñado e implementado un dispositivo capaz de paralelizar el entrenamiento de algoritmos SVM.

Aunque se sigue demostrando que el rendimiento mejora de manera lineal al añadir un segundo procesador, queda claro que el procesador Microblaze que se utilizó en los experimentos no es lo suficiente potente para ejecutar el entrenamiento sobre problemas reales, aunque abre las puertas a futuros experimentos con hardware específico que ejecute los algoritmos de entrenamiento y clasificación.

Capítulo 5

Implementación en computación distribuida (Altamira)

Como hemos visto, el algoritmo pSVM es altamente paralelizable y su implementación en plataformas masivamente paralelas se puede realizar de forma relativamente sencilla gracias a su estructura. Los resultados mostrados hasta ahora han sido clasificando conjuntos sintéticos obtenidos en un ordenador y una placa de prototipado. Realizar pruebas con conjuntos de datos orgánicos en estos sistemas es difícil debido al tamaño que tienen y a las limitaciones de memoria RAM y capacidad de cálculo que presentan estos sistemas, por lo que es necesario realizar pruebas en un sistema de alto rendimiento como Altamira.

Altamira es un cluster de la Universidad de Cantabria com-

puesto por 158 nodos de cómputo, 5 nodos de cómputo adicionales con GPU, un servidor de login y varios de servicio. Los nodos de cómputo principales tienen dos procesadores Intel Sandybridge E5-2670, cada uno con 8 cores operando a 2.6 GHz, 64 GB de RAM (disponibles 4 GB/core) y 500 GB de almacenamiento en disco duro local. Los nodos de cómputo corren Scietific Linux 6.2. La red interna de Altamira es una red Infiniband. En el año 2007 llegó a situarse en el puesto 412 de la lista top 500 de supercomputadores.

5.1. Modelo de programación

Según [14] un sistema paralelo debe seguir un modelo para la ejecución y comunicación de sus tareas que permita llevar a cabo de forma ordenada y coherente su objetivo o algoritmo en ejecución. Entre los más importantes encontramos:

- Modelo de tareas-canales. El sistema paralelo se compone de tareas, que son programas en ejecución con su propia memoria y sistemas de entrada-salida funcionando en una máquina Von Neumann "virtual". Así mismo, se encuentra compuesto por canales que consisten en una cola de mensajes que se encargan de llevar información desde un sistema de entrada-salida de una tarea a un sistema e entrada-salida de otra.

- Modelo de paso de mensajes. Es uno de los modelos más usados actualmente para la computación en sistemas paralelos como clusters, grids, etc. Consiste en una variación del modelo tareas-canales, dado que se lanzan una

serie de tareas en distintas máquinas, con su propia memoria y sistemas de entrada, pero en vez de hacer uso de canales para las comunicaciones, estas se hacen mediante mensajes para tareas específicas. Así, si se desea enviar un mensaje, este se envía a una tarea específica y no a un canal abierto.

- Paralelismo de datos. En este tipo de paralelismo se aprovecha la ejecución de un mismo código a un conjunto de datos, por ejemplo, cuando deseamos sumar un valor determinado a cada uno de los elementos de un vector. Debido a que las operaciones que se realizan son independientes, es posible que cada elemento de procesamiento en una plataforma paralela se encargue de realizar las operaciones en uno o más elementos del vector. En este caso, el compilador requiere información sobre la forma en que serán distribuidas las tareas en cada procesador y las tareas de comunicaciones podrán ser incluidas de forma automática.

- Memoria compartida. En este modelo las tareas se ejecutan en distintos elementos de procesamiento que comparten un espacio de memoria para intercambiar los datos asíncronamente. Para controlar el flujo de datos a la memoria se hace uso de semáforos y mutex entre otros. Entre las ventajas de este modelo se encuentra la facilidad relativa de programación y la velocidad de las comunicaciones entre tareas, dado que la latencia que presenta la memoria es menor que la presentada al pasar mensajes.

Debido a la metodología usada en diseño del algoritmo pSVM (divide y vencerás), el uso del modelo de memoria distribuida con paso de mensajes encaja de forma natural, porque cada uno de los subconjuntos de entrenamiento generados con k-medias es independiente entre sí y puede ser ejecutado en un espacio independiente de memoria. Una vez ejecutado k-medias en el nodo maestro, este enviará los subconjuntos de entrenamiento a los nodos esclavos para que ejecuten el entrenamiento y escriban los modelos en el directorio seleccionado. Además, el modelo de paso de mensajes ofrece la ventaja de ser compatible con la mayoría de las arquitecturas paralelas más potentes [26], permitiendo portar el código sin modificaciones.

5.2. Diseño del programa paralelo

A continuación se detalla el funcionamiento del algoritmo pSVM implementado utilizando la librería de paso de mensajes MVAPICH2 que ha demostrado ser hasta un 20 % más rápida funcionando sobre Infiniband que OpenMPI [21]. El patrón de comunicaciones utilizado en pSVM es el conocido como fan-out, en el que el proceso maestro se encarga de particionar el problema y repartir subproblemas a los procesos esclavo que realizarán la tarea y comunicarán de nuevo al proceso maestro la salida.

5.2.1. Particionamiento

El nodo maestro lee el problema completo y realiza k-medias sobre cada una de las clases, como en versiones anteriores,

para a continuación realizar una combinación de cada uno de los conglomerados generados obteniendo $k_1 \times k_2$ subconjuntos de entrenamiento. Para optimizar el rendimiento, el número de procesos de la aplicación debe ser $n_p = k_1 \times k_2$. En los experimentos realizados en esta tesis se ha optado por leer el número de procesos lanzados con MPI (fijados desde línea de comandos al lanzar el programa) y hacer que $k_1 = k_2 = \sqrt{n_p}$.

Una vez generados los conglomerados, utilizando las funciones de MPI enviaremos un subconjunto a cada proceso para que realice el entrenamiento y este escriba el modelo obtenido en el directorio de salida.

5.2.2. Paso de mensaje con MPI

En la programación paralela con paso de mensajes se ejecutan una serie de procesos en paralelo que se comunican con una librería de funciones que permiten enviar y recibir mensajes. Para las comunicaciones en el algoritmo de entrenamiento en el cluster Altamira se empleó la librería MVAPICH2, que consiste en una colección de funciones complejas multiplataforma que ayudan a simplificar la programación de las comunicaciones en los procesos.

La librería contiene 6 funciones básicas (mostradas en la tabla 5.1) que permiten construir un programa paralelo completamente funcional.

Además, MVAPICH2 cuenta con funciones colectivas que permiten realizar comunicaciones que involucren a un grupo o todos los procesos presentes en un determinado momento. La

Función	Descripción
MPI_Init	Inicia la librería MPI
MPI_Finalize	Termina la librería MPI
MPI_Comm_size	Determina el número de procesos lanzados
MPI_Comm_rank	Determina cuál es el proceso activo
MPI_Send	Envía un mensaje
MPI_Recv	Recibe un mensaje

Cuadro 5.1: Funciones básicas MVAPICH2

tabla 5.2 muestra las principales funciones colectivas que fueron estudiadas como posibles métodos de comunicación del programa. De todas ellas, solo se empleó MPI_Barrier, una función que permite sincronizar todas las tareas.

Función	Descripción
MPI_Bcast	Comunicación de difusión
MPI_Reduce	Comunicación con reducción o acumulación
MPI_Gather	Comunicación con recolección
MPI_Scatter	Comunicación con dispersión
MPI_Allgather	Comunicación con multidifusión
MPI_Alltoall	Comunicación con intercambio completo
MPI_Barrier	Función de sincronismo

Cuadro 5.2: Funciones colectivas de MVAPICH2

5.3. Resultados

Para esta versión del algoritmo se han realizado varios experimentos en los que medimos el tiempo medio de entrenamiento de distintas ejecuciones de distintos conjuntos de datos:

5.3.1. Normal

Este experimento presenta dos nubes de datos generados a partir de dos distribuciones normales: una de ellas centrada en el punto (0,0) y la otra centrada en el punto (10,10), ambas con una desviación tipica de 1.

En los capítulos anteriores se ha probado con éxito la mejora de rendimiento de pSVM con el experimento Poisson que presentaba bastante solapamiento, ahora se pretende determinar si el rendimiento también mejora en conjuntos muy separados. Para todos los casos, tanto libSVM como pSVM clasificaban correctamente el 100 % de los puntos. A continuación, se muestra en la tabla 5.3 del tiempo medio de 100 ejecuciones entre pSVM y libSVM en el que se modifica la población de ambas nubes de datos y el número de procesos para la versión paralela.

Se observa que debido a la gran separación entre ambas nubes libSVM encuentra rápidamente un hiperplano de separación independientemente del tamaño de las nubes, mientras que la versión paralela es más lenta debido a la sobrecarga de tener que realizar los conglomerados con k-medias.

5.3.2. Poisson

Es el mismo experimento expuesto en la sección 3.3. Se han realizado múltiples experimentos variando los tamaños de las nubes y el número de procesos lanzados en el cluster Altamira.

En la tabla 5.4 se puede observar una comparativa del tiempo medio de entrenamiento (en segundos) de 100 ejecuciones del

	libSVM	pSVM				
Num. Procesos	1	4	9	16	25	36
100k	0.03	0.91 (0.22)	1.43 (0.23)	3.91 (0.26)	4.61 (0.28)	5.31 (0.31)
1M	0.26	6.72 (2.57)	12.75 (2.67)	14.14 (2.95)	18.31 (3.15)	26.75 (3.19)
2M	0.42	13.75 (5.27)	22.85 (5.63)	27.21 (5.81)	29.99 (6.3)	41.82 (6.42)
10M	2.29	40.22 (30.47)	57.96 (30.55)	68.28 (33.14)	123.67 (33.78)	135.32 (35.19)
20M	3.02	89.79 (65.28)	121.08 (68.87)	136.55 (72.22)	259.41 (75.36)	280.68 (79.4)

Cuadro 5.3: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento Normal. Los tiempos entre paréntesis para pSVM son los tiempos (segundos) para crear las sub-svms.

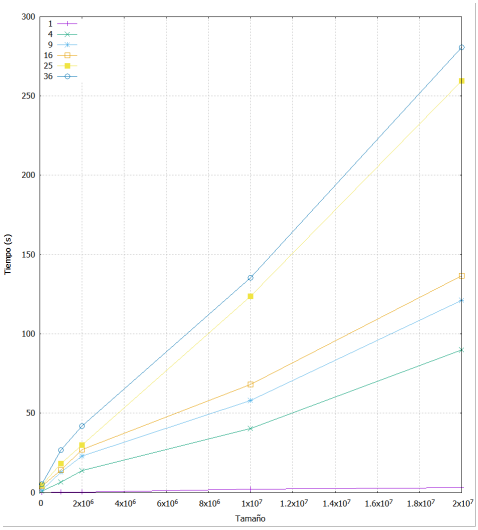

Figura 5.1: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento Normal

algoritmo estándar libSVM y de pSVM modificando el número de procesos concurrentes Se puede observar como las conclusiones alcanzadas en capítulos anteriores se siguen manteniendo.

A continuación, en el cuadro 5.5 se muestra el speedup de cada uno de los experimentos respecto a la versión secuencial.

Se puede observar en los experimentos que aumentando la población de individuos, aumenta el solapamiento de las dos clases, haciendo más difícil trazar un hiperplano, por lo que la versión secuencial del algoritmo se hace más lenta. En cambio, el algoritmo pSVM gracias al tratamiento previo de los datos con k-medias, consigue separar las subclases haciendo más fácil trazar un hiperplano.

También se puede observar que aumentar el número de proce-

	libSVM	pSVM						
Num. Procesos	1	4	9	16	25	36	49	64
Poisson 1k	0.02	0.01	0.01	0.01	0.01	0.01	0.3 (0.01)	0.38 (0.01)
Poisson 10k	1.62	0.72 (0.02)	0.42 (0.02)	0.24 (0.02)	0.28 (0.02)	0.45 (0.02)	0.89 (0.03)	1.97 (0.03)
Poisson 20k	6.65	1.77 (0.04)	1.75 (0.04)	1.05 (0.04)	1.12 (0.05)	1.19 (0.05)	1.29 (0.06)	2.15 (0.06)
Poisson 30k	15.14	5.15 (0.07)	4.71 (0.07)	3.31 (0.07)	4.59 (0.08)	4.26 (0.08)	4.45 (0.08)	4.49 (0.08)
Poisson 40k	26.70	5.53 (0.09)	4.87 (0.09)	3.50 (0.09)	4.01 (0.09)	4.59 (0.10)	4.72 (0.10)	4.91 (0.10)
Poisson 50k	73.54	6.44 (0.12)	5.85 (0.12)	5.32 (0.12)	5.45 (0.12)	6.12 (0.12)	6.45 (0.13)	7.79 (0.13)

Cuadro 5.4: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento Poisson. Los tiempos entre paréntesis para pSVM son los tiempos (segundos) para crear las sub-svms.

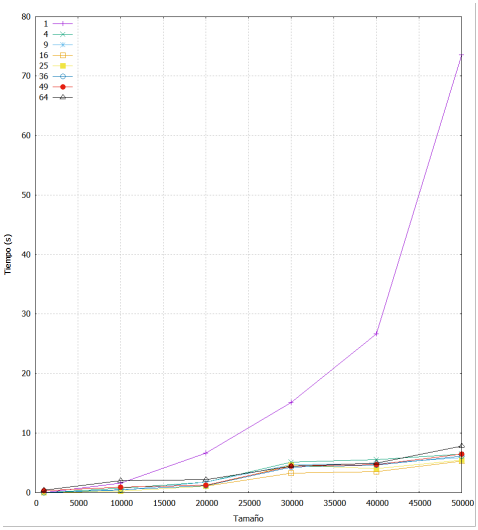

Figura 5.2: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento Poisson

	pSVM						
Num. Procesos	4	9	16	25	36	49	64
Poisson 1k	2	2	2	2	2	0.07	0.05
Poisson 10k	2.25	3.86	6.75	5.79	3.60	1.82	0.82
Poisson 20k	3.76	3.80	6.33	5.94	5.59	5.16	3.09
Poisson 30k	4.64	5.07	7.22	5.21	5.61	5.37	5.32
Poisson 40k	4.83	5.48	7.63	6.66	5.82	5.66	5.44
Poisson 50k	11.42	12.57	13.82	13.49	12.02	11.40	9.44

Cuadro 5.5: Speedup de pSVM respecto a libSVM en Altamira para el experimento Poisson

sos de la versión paralela influye en el tiempo de ejecución debido a la sobrecarga de generar demasiados conglomerados con k-medias, provocando como efecto colateral que unos pocos conglomerados concentren la mayoría de individuos haciendo

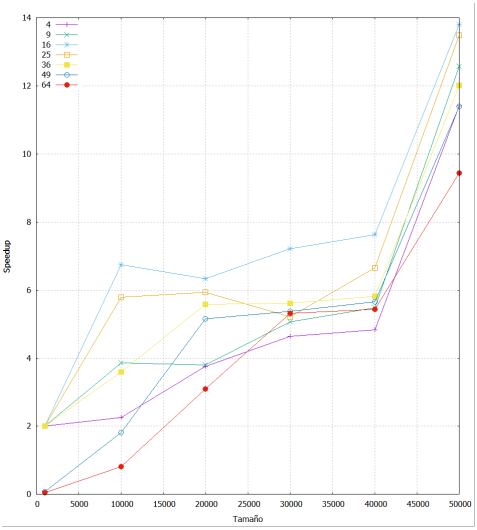

Figura 5.3: Speedup de pSVM respecto a libSVM en Altamira para el experimento Poisson

más difícil su entrenamiento.

5.3.3. UCI Adult

Este conjunto fue donado al UCI Machine Learning Repository ([4]) por Ronny Kohavy y Barry Becker y es usado en [40] para demostrar el correcto funcionamiento de SMO y hacer comparativas con otros algoritmos de la familia SVM.

Está formado por 32.561 entradas con 14 atributos de un censo de familias e intenta predecir si en esa familia los ingresos anuales son mayores de 50.000 dólares. De los 14 atributos, 8 son categóricos (clase de trabajo, educación, estado civil, ocupación, relación, raza, sexo y país de origen) y 6 continuos (edad, peso final otorgado por el experto, educación más alta,

ganancias de capital, pérdidas de capital y horas de trabajo a la semana). Los 6 continuos son discretizados en quintiles, haciendo un total de 123 atributos binarios usados para entrenar la SVM. El 84.99 % de los individuos fueron clasificados correctamente, un 73.05 % para la clase positiva y 87.92 % para la negativa.

En la tabla 5.6 se puede observar una comparativa del tiempo medio de entrenamiento (en segundos) de 100 ejecuciones del algoritmo estándar libSVM y de pSVM modificando el número de procesos concurrentes.

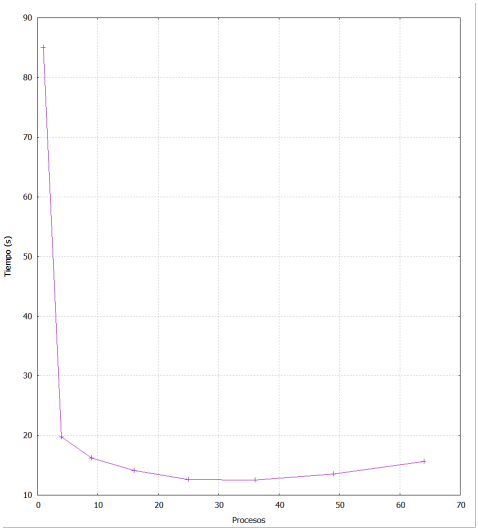

Figura 5.4: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento UCI Adult

En la tabla 5.7 se muestra el speedup obtenido con pSVM respecto a la versión secuencial libSVM.

Los experimentos realizados con el conjunto de datos Adult

	libSVM	pSVM						
Num. Procesos	1	4	9	16	25	36	49	64
UCI Adult	85.06	19.8 (2.01)	16.27 (2.48)	14.17 (2.85)	12.61 (3.29)	12.58 (3.54)	13.55 (3.89)	15.67 (4.44)

Cuadro 5.6: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento UCI Adult. Los tiempos entre paréntesis para pSVM son los tiempos (segundos) para crear las subsvms.

	pSVM						
Num. Procesos	4	9	16	25	36	49	64
UCI Adult	4.30	5.23	6.00	6.75	6.76	6.28	5.43

Cuadro 5.7: Speedup de pSVM respecto a libSVM en Altamira para el experimento UCI Adult

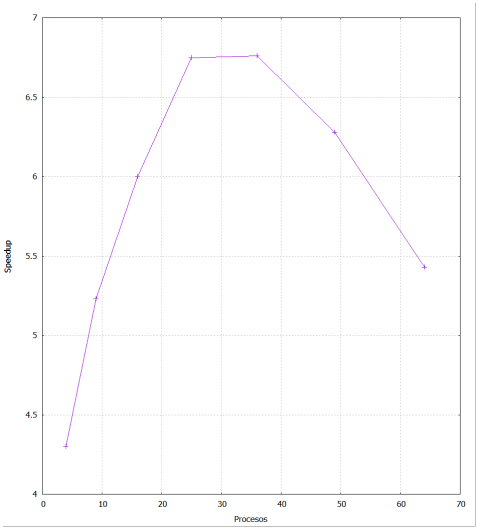

Figura 5.5: Speedup de pSVM respecto a libSVM en Altamira para el experimento UCI Adult

demuestran que el tiempo de entrenamiento con pSVM disminuye respecto a la versión secuencial de libSVM.

5.3.4. RNA no codificante

Los RNA no codificante (ncRNA) son moléculas de ARN funcionales que no se traducen en una proteína. El número de ncRNAs conocidos está creciendo rápidamente y su importan-

cia ha sido subestimada en los modelos clásicos de procesos celulares. Es deseable desarrollar métodos de alto rendimiento para clasificar nuevos ncRNAs para comprender mejor la biología y descubrir candidatos de nuevas medicinas.

Este conjunto de datos fue presentado en [43]. Es un conjunto de 59535 muestras con 8 atributos. El atributo 1 es el Dynalign computado total [31], el atributo 2 es la longitud de la secuencia más corta, los atributos 3, 4 y 5 son las frecuencias de dinucleótidos AA, AU y AC de la secuencia 1 (E. Coli); y los atributos 6, 7 y 8 son las frecuencias de dinucleótidos AA, AU y AC de la secuencia 2 (S.Typhi). Estas muestras están clasificadas como positivas (+1) sin contienen una secuencia RNA no codificante o negativa (-1) si no la contienen. El 90.04 % de los individuos se clasificaron correctamente, un 88.72 % de los positivos y un 90.61 % de los negativos.

En la tabla 5.8 se presenta una comparativa del tiempo medio de entrenamiento (en segundos) de 100 ejecuciones del algoritmo estándar libSVM y de pSVM modificando el número de procesos concurrentes.

En la tabla 5.9 se muestra el speedup obtenido respecto a la versión secuencial para los experimentos realizados.

Los experimentos realizados con el conjunto RNA no codificante desmuestran que pSVM mejora notablemente el tiempo de entrenamiento respecto a la versión secuencial libSVM.

	libSVM	pSVM		
Num. Procesos	1	4	9	16
Cod-RNA	5838.68	2859.63 (0.21)	2755.42 (0.24)	2699.82 (0.27)
	pSVM			
Num. Procesos	25	36	49	64
Cod-RNA	2545.14 (0.24)	2492.27 (0.27)	2591.68 (0.27)	2702.14 (0.27)

Cuadro 5.8: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento RNA no codificante. Los tiempos entre paréntesis para pSVM son los tiempos (segundos) para crear las sub-svms.

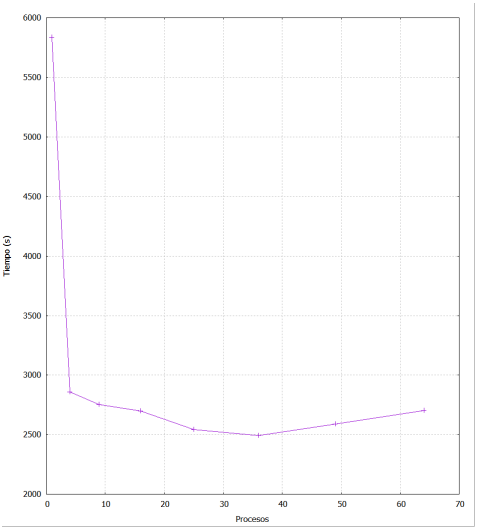

Figura 5.6: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento RNA no codificante

	pSVM						
Num. Procesos	4	9	16	25	36	49	64
Cod-RNA	2.04	2.12	2.16	2.29	2.34	2.25	2.16

Cuadro 5.9: Speedup de pSVM respecto a libSVM en Altamira para el experimento RNA no codificante

5.3.5. Cáncer de mama de Universidad de Wisconsin

El siguiente conjunto fue presentado en [47] y donado al repositorio UCI. El conjunto está compuesto por 699 individuos con 10 atributos (identificador, espesor de masa, uniformidad del tamaño de células, uniformidad del tamaño de células, adhesión marginal, tamaño individual de células epiteliales, nú-

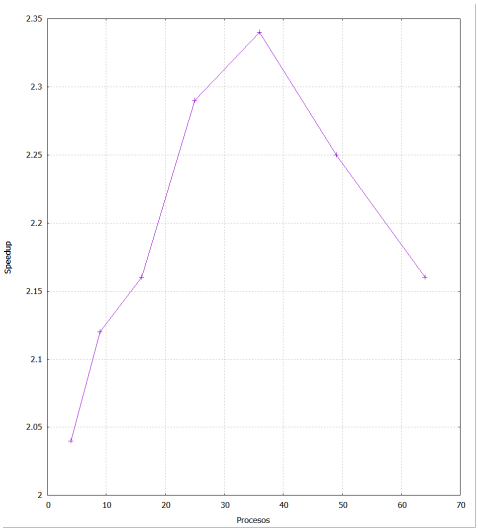

Figura 5.7: Speedup de pSVM respecto a libSVM en Altamira para el experimento RNA no codificante

cleos desnudos, cromatina blanda, nucleolos normales y mitosis). Los individuos se clasifican como +1 si padecen de cáncer de mama o -1 en caso contrario. Se clasificaron correctamente un 97.07 % de los individuos, un 95.06 % de los positivos y un 98.18 % de los negativos.

Se presenta una comparativa del tiempo medio de entrenamiento (en segundos) de 100 ejecuciones del algoritmo estándar libSVM y de pSVM modificando el número de procesos concurrentes en la tabla 5.10.

A continuación, se muestra en la tabla 5.11 el speedup obtenido por la versión paralela respecto a la versión secuencial en los experimentos anteriores.

	libSVM	pSVM						
Num. Procesos	1	4	9	16	25	36	49	64
Breast-cancer	51.80	11.23 (0.01)	18.46 (0.01)	24.11 (0.01)	24.18 (0.01)	26.03 (0.01)	28.1 (0.01)	28.85 (0.01)

Cuadro 5.10: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento cáncer de mama en Wisconsin. Los tiempos entre paréntesis para pSVM son los tiempos (segundos) para crear las sub-svms.

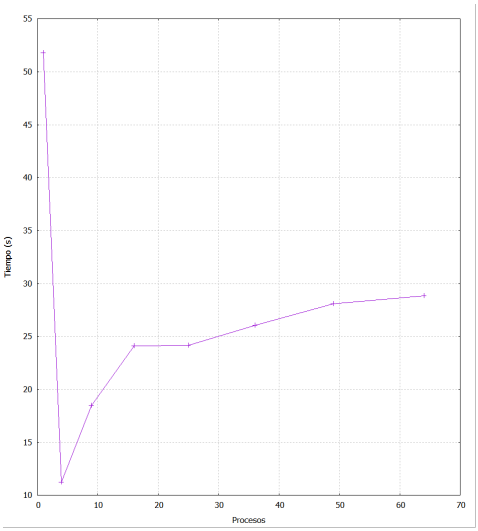

Figura 5.8: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento cáncer de mama en Wisconsin

	pSVM						
Num. Procesos	4	9	16	25	36	49	64
Breast-cancer	4.61	2.81	2.15	2.14	1.99	1.84	1.80

Cuadro 5.11: Speedup de pSVM respecto a libSVM en Altamira para el experimento cáncer de mama en Wisconsin

5.3.6. Enfermedades de hígado

Presentado y donado por [13] al repositorio UCI, este conjunto presenta 6 atributos (volumen medio del cuerpo, fosfatasa alcalina, alanina aminotransferasa, aspartato aminotransferasa, gamma-glutamil transpeptidasa, número de medias pintas que bebe la persona al dia) de 345 personas y clasificados como +1 si tienen alguna enfermedad de hígado y -1 en caso contrario.

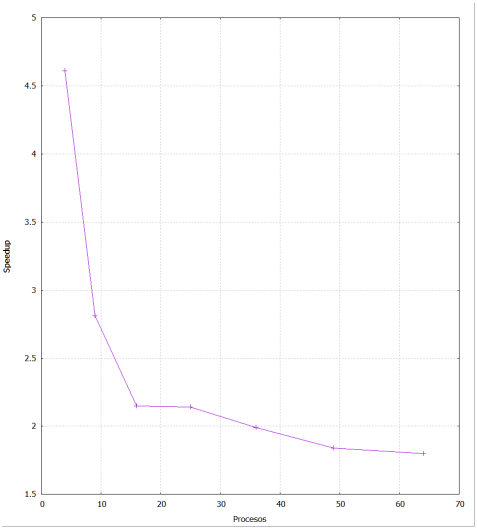

Figura 5.9: Speedup de pSVM respecto a libSVM en Altamira para el experimento cáncer de mama en Wisconsin

El 71.59 % de los individuos fue clasificado correctamente, un 71.98 % de los positivos y un 70.80 % de los negativos.

Como en los apartados anteriores, se presenta una comparativa del tiempo medio de entrenamiento en segundos de 100 ejecuciones de la versión secuencial libSVM y de la versión paralela pSVM modificando el número de procesos en la tabla 5.12.

El speedup obtenido en el entrenamiento por la versión paralela pSVM respecto a la versión secuencial libSVM se muestra en la tabla 5.13.

	libSVM	pSVM						
Num. Procesos	1	4	9	16	25	36	49	64
Liver	2.56	0.42 (0.01)	0.42 (0.01)	0.21 (0.01)	0.16 (0.01)	0.08 (0.01)	0.11 (0.01)	0.18 (0.01)

Cuadro 5.12: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento enfermedades de hígado. Los tiempos entre paréntesis para pSVM son los tiempos (segundos) para crear las sub-svms.

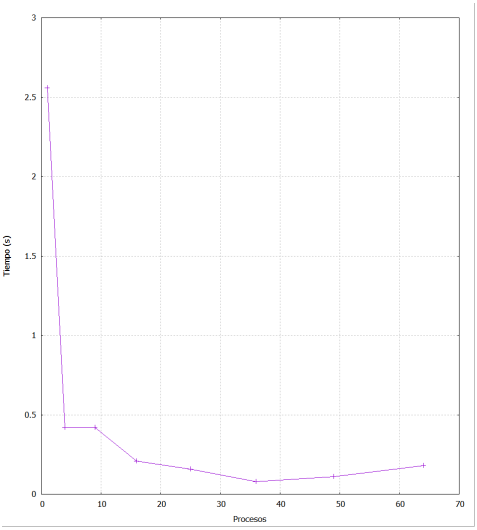

Figura 5.10: Comparativa de tiempo (segundos) entre libSVM y pSVM en Altamira para el experimento enfermedades de hígado

	pSVM						
Num. Procesos	4	9	16	25	36	49	64
Breast-cancer	6.10	6.10	12.19	16.00	32.00	23.27	14.22

Cuadro 5.13: Speedup de pSVM respecto a libSVM en Altamira para el experimento enfermedades de hígado

5.4. Conclusiones

Se ha implementado una versión concurrente para computación de alto rendimiento del algoritmo pSVM utilizando MVAPICH2. Esta librería de paso de mensaje permite ejecutar concurrentemente distintas sub-SVMs, explotando el paralelismo del algoritmo.

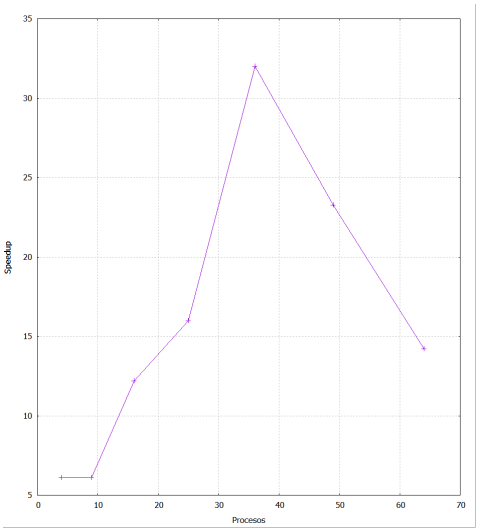

Figura 5.11: Speedup de pSVM respecto a libSVM en Altamira para el experimento enfermedades de hígado

Los experimentos con conjuntos sintéticos Normal y Poisson demuestran que la separación de las clases influye directamente en el rendimiento tanto de la versión secuencial como la versión paralela del algoritmo.

Además, se han realizado experimento sobre conjuntos orgánicos presentados en artículos conocidos de la bibliografía sobre aprendizaje máquina:

- Adult: con el conjunto presentado en [40] se intenta predecir si una familia va a tener ingresos superiores a 50.000 dólares según algunos parámetros relevantes.
- RNA no codificante: con el conjunto presentado en [43] se intenta predecir si las muestras contienen RNA no

codificante a partir de secuencias de dinucleótidos y otros parámetros.

- Cáncer de mama de la Universidad de Wisconsin: presentado en [47] se intenta predecir si una paciente tiene un cáncer de mama a partir de ciertos análisis.

- Enfermedades de hígado: presentado en [13] se intenta predecir si un paciente tiene un hígado enfermo a partir de parámetros obtenidos con análisis de sangre.

En estos experimentos se puede observar como el tiempo que tarda en ejecutar k-medias incrementa con el tamaño del problema, pero casi no varia independientemente del número de procesos que se lancen. También se observa que el speedup máximo se alcanza con un número de procesos de 16 o 25 y a partir de ese número el speedup decae a valores alcanzados con menos procesos. Esto se debe al aumento de carga por las comunicaciones entre el maestro y los esclavos.

Los experimentos realizados con pSVM sobre estos conjuntos en el computador de alto rendimiento Altamira han obtenido resultados que respaldan los resultados obtenidos en capítulos anteriores de la tesis.

Capítulo 6

Análisis de sentimientos en textos con clasificadores

En la sociedad actual, debido al gran auge que en los últimos años está experimentando lo que recibió el nombre de Web 2.0, véase [38], que incluye las redes sociales, micro-blogs, blogs personales o redes profesionales, se ha producido un incremento exponencial de la información subjetiva disponible en Internet. Así, cualquier persona o institución puede recuperar información sustancial de un tema de interés a partir del análisis de los datos disponibles en forma de mensajes y comentarios recibidos mediante los canales de redes sociales, blogs o wikis, véase [32]. Estos datos subjetivos tienen un gran potencial y pueden explotarse para, por ejemplo, conocer opininiones sobre personalidades públicas, elegir la propaganda idónea según las

preferencias u opiniones de la gente o, encontrar el producto mejor valorado por los usuarios, véase [29]. Para ello, ha surgido el análisis de sentimientos, también conocido como minería de opiniones, que es una tarea del procesamiento del lenguaje natural que identifica opiniones relacionadas con un objeto y trata de analizar las opiniones, sentimientos, evaluaciones o emociones en relación con cualquier tema de interés mencionado en los datos recopilados. Los temas pueden variar desde la opinión sobre una marca o producto, en caso de querer conocer la satisfacción de los clientes, a temas más generales como noticias de última hora, cuestiones políticas o asuntos económicos.

La minería de textos es una aplicación del procesamiento de textos que pretende facilitar la identificación y extracción de nuevo conocimiento a partir de colecciones de documentos [5], trabajando, por lo tanto, con información no estructurada. Inicialmente, la minería de textos debería facilitar el análisis de estos documentos que, a priori, resultarían inmanejables debido a su tamaño, de modo que se puedan obtener relaciones entre los documentos y extraer conclusiones sobre los mismos. En el caso de documentos escritos en diferentes idiomas, resulta interesante la utilización de diferentes recursos lingüísticos (glosarios, ontologías, ...) para representar los documentos mediante rasgos independientes del idioma.

Los problemas básicos que pueden abordarse con técnicas de minería de textos son la recuperación de información relevante, la categorización de documentos y la agrupación de documentos relacionados con un tema. Para conseguir esto, la minería textual adopta una serie de técnicas procedentes de la recupe-

ración de información y de la lingüística computacional como son:

- El pre-procesamiento de documentos, que se compone de la extracción de términos, la eliminación de palabras vacías y la normalización de los términos mediante la técnica de "stemming", que consiste en extraer la raíz de las palabras.
- La representación de los documentos mediante un modelo vectorial, con el que un documento se representa mediante un vector, siendo cada término del documento un componente del mismo.
- La categorización automática, que se utiliza para clasificar documentos en alguna de las categorías preestablecidas.

Por tanto, la minería de textos tiene como objetivo intermedio procesar y presentar la información disponible en un formato que facilite su comprensión y análisis. En este punto es donde entra en relación con los sistemas de explicación, en particular en sistemas de participación, como nos interesa en esta Tesis.

Originalmente, los sistemas de explicación surgen en el contexto de los sistemas expertos y la inteligencia artificial, incorporando técnicas de lenguaje natural. Los sistemas de explicación deben ser capaces de describir claramente el dominio del problema y los resultados obtenidos. Se ha demostrado que las explicaciones adecuadas contribuyen positivamente a mejorar los resultados y la satisfacción de los usuarios. Además, influyen en las percepciones del usuario como la confianza, la

confidencialidad y la satisfacción, e incrementan sus niveles de aceptación y aprendizaje. Cuando se construye un sistema o proceso, hay que tener cuidado con predecir las necesidades de los futuros usuarios y elaborar el material que satisfaga las necesidades de los mismos, puesto que si tales suposiciones no están probadas, pueden no constituir una base fiable para el diseño. Una falta de atención a la hora de diseñar el proceso sin tener en cuenta las necesidades de los usuarios, está destinada a limitar la funcionalidad del sistema, ya que es probable que no todos los usuarios conozcan los supuestos que se dan por sentados.

Un buen ejemplo son los módulos de explicación para justificar los resultados obtenidos mediante un sistema de toma de decisiones multicriterio de modo comprensible. Usualmente, generan dos informes en lenguaje natural, uno para evaluar los resultados y comparar alternativas, y otro, que explica la sensibilidad a cambios en, por ejemplo, los pesos asignados a criterios.

Los sistemas de explicación son especialmente importantes en democracia electrónica, donde muchas personas con diferentes percepciones o expectativas están involucradas en los procesos de toma de decisiones complejas.

6.1. Análisis de los mensajes

Durante el estudio de los diferentes instrumentos de participación utilizados globalmente, vimos cómo una de las tareas comunes en muchos de ellos era el debate a través de un foro

en el que los participantes intercambiaban información y comunicaban sus preferencias. También describimos el método de negociación POSTING, con el que los usuarios pueden escribir mensajes que apoyen las propuestas enviadas. Por ello, en la arquitectura presentada en el Capítulo 2 se incluye un módulo para implementar un foro que permita añadir dicha tarea de debate a un proceso participativo. Toda esta información puede ser fácilmente recuperada de los documentos XML creados a partir de los esquemas presentados en el Capítulo 3.

Nuestro objetivo en este capítulo es, mediante el análisis y clasificación de los mensajes escritos por los participantes, conocer cuáles son las tendencias de opinión de los mismos, las propuestas con mayor aceptación, las posibles soluciones al problema planteado que satisfagan a la mayoría de los participantes, qué reacciones generaría el hecho de tomar una decisión concreta, etc, con vistas a proporcionar una explicación semiautomática.

En esta sección explicamos brevemente cómo transformar mensajes en vectores y dos maneras de aprendizaje adaptables al problema del análisis de sentimientos en el contenido de textos como son k-NN, k vecino más próximo (del inglés k-Nearest Neighbour) y SVM, máquinas de vector soporte (del inglés Support Vector Machines).

6.1.1. Representación vectorial de los mensajes

Los mensajes y términos de, por ejemplo, un foro, pueden representarse conjuntamente a través de la denominada *matriz*

de términos-documentos [5], a la que nos referimos como matriz de términos-mensajes con el fin de que sea más representativa del ejemplo que mostraremos. Consideremos una muestra de n mensajes $C = \{x_1, \ldots, x_n\}$ y una muestra de m términos $T = \{t_1, \ldots, t_m\}$. En esta matriz, cada columna representa un mensaje y cada fila corresponde a un término. Entonces, la celda (i, j) corresponde a la frecuencia ij de aparición del término t_i en el mensaje x_j. En el Cuadro 6.1 podemos ver un ejemplo de una matriz de términos-mensajes con n mensajes y m términos.

Diccionario	Mensaje 1	Mensaje 2	...	Mensaje n
mejor	f_{11}	f_{12}	...	f_{1n}
grande	f_{21}	f_{22}	...	f_{2n}
malo	f_{31}	f_{32}	...	f_{3n}
bueno	f_{41}	f_{42}	...	f_{4n}
...	...	...	...	...
término-m	f_{m1}	f_{m2}	...	f_{mn}

Cuadro 6.1: Ejemplo de matriz de términos-mensajes.

La lista de términos T forma lo que se denomina el *diccionario* de términos. Como resultado, podemos considerar cada columna como una representación m-dimensional de los correspondientes mensajes permitiéndonos utilizar diferentes métodos de aprendizaje cuyo funcionamiento está basado en la representación vectorial de mensajes.

6.2. Metodología

En esta sección se presentan los detalles de la metodología e implementación que se han utilizado para extraer opiniones. El objetivo es:

- Describir un proceso multifase para la categorización de los mensajes y la extracción de opiniones, combinado con métodos supervisados y no supervisados de aprendizaje.
- Comparar el rendimiento de diferentes procedimientos para el análisis de sentimientos del contenido textual extraído de los comentarios del foro.
- Obtener conclusiones y recomendaciones para esta comparación, que puede utilizarse para mejorar la metodología e implementación del análisis de sentimientos en otros casos similares.

Para analizar los mensajes extraídos, es importante utilizar los métodos y diccionarios apropiados para llevar a cabo el análisis de sentimientos.

6.2.1. Evaluación de los mensajes

Nuestro propósito es comparar diferentes técnicas para la clasificación del contenido y el análisis de sentimientos de los mensajes expuestos en un foro, utilizando una combinación de métodos de aprendizaje supervisados y no supervisados, véase [46]. Para lograr esto, proponemos una clasificación multifase y un análisis de opiniones como el presentado en la Figura

6.1. En primer lugar, utilizamos métodos de aprendizaje supervisado para discriminar los mensajes categorizándolos en las temáticas 1, ..., N. Seguidamente, y utilizando de nuevo los métodos de aprendizaje supervisado, clasificamos los mensajes de cada una de las temáticas como comentario positivo, negativo o neutro. Por último, aplicamos el algoritmo de k-medias (aprendizaje no supervisado) para extraer las sub-temáticas que caracterizan las diferentes tendencias de opinión expresadas en los mensajes de cada temática. El esquema mostrado en la Figura 6.1 es adaptativo, pues permite la actualización de tendencias de opinión con la llegada de nuevos mensajes que llegan al sistema.

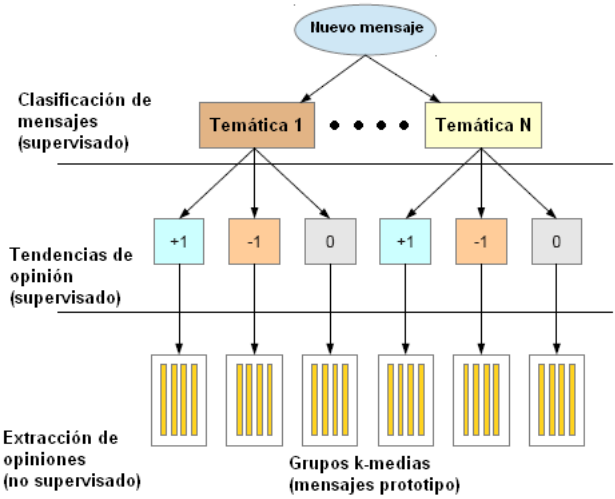

Figura 6.1: Método de clasificación multifase para categorización de mensajes y análisis de sentimientos.

La Figura 6.2 muestra una vista general del procedimiento llevado a cabo en cada fase, lo que implica un método de apren-

dizaje supervisado. Todos los mensajes se clasifican manualmente de acuerdo al objetivo correspondiente a cada nivel del proceso multifase mostrado en la Figura 6.1. En cada nivel, se lleva a cabo un proceso de votación simple para clasificar los mensajes. En el primer nivel, clasificación de contenido, los mensajes se clasifican agrupándolos en las temáticas 1, ..., N. En caso de que un mensaje mencione varias temáticas, se divide para agrupar cada texto en su temática correspondiente.

En el segundo nivel (tendencias de opinión) podríamos contar con tantas categorías como consideremos necesario. Para el caso que trataremos hemos clasificado los mensajes de cada temática en tres categorías diferentes:

- *Categoría +1* (C_{+1}): Mensajes cuyo contenido expresa una opinión positiva.
- *Categoría 0* (C_0): Mensajes sin un claro signo de opinión, ni positivo ni negativo.
- *Categoría -1* (C_{-1}): Mensaje cuyo contenido expresa una opinión negativa.

Estas clasificaciones, realizadas manualmente, se utilizarán como entrada a los métodos de aprendizaje supervisado aplicados en este estudio. En particular, se toma el 70 % de los mensajes para crear el *conjunto de entrenamiento*, utilizando el 30 % restante como *conjunto de prueba* para evaluar el rendimiento de los métodos (de aprendizaje supervisado). La misma operación se repite para la categorización del contenido. En el primer nivel, se identifican los mensajes que hablan de cada temática y, en el segundo, se identifican las tendencias de opinión.

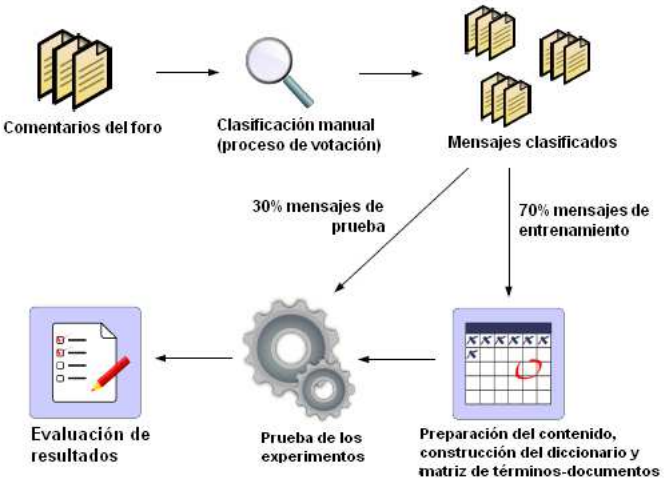

Figura 6.2: Descripción de la metodología.

Una vez hecho esto, los mensajes deben prepararse para el análisis. Para ello, cada uno de ellos se trata utilizando la librería de minería de textos *tm*, que permite eliminar palabras vacías (palabras comunes que normalmente no añaden información útil para el análisis), números, signos de puntuación y espacios en blanco. Después de esto, el texto de todos los mensajes se convierte a minúsculas y se utiliza la función *termDocumentMatrix* de *tm* para generar la matriz de términos-mensajes

Debemos indicar que sólo los mensajes utilizados para entrenamiento contribuyen a la construcción del diccionario de términos. Así, cuando un nuevo mensaje del conjunto de prueba se analiza con el clasificador, se ignora cualquier término que no aparece en los mensajes de entrenamiento a la hora de calcular la distancia entre mensajes. Es importante indicar qué sinónimos, abreviaciones o formas alternativas para un término dado

no han sido tenidas en cuenta en el trabajo realizado.

6.2.2. Clasificación de textos y análisis de sentimientos

En los dos primeros niveles del procedimiento multifase descrito anteriormente se han comparado los resultados de dos métodos de aprendizaje supervisado que pueden aplicarse a la clasificación de textos y al análisis de sentimientos en los mensajes:

- *k-NN*: Este método viene proporcionado directamente por la librería *tm*. Se basa en el contenido de todo el conjunto de mensajes que se pretenden analizar. Siguiendo este procedimiento, se obtiene la distancia de todos los mensajes de prueba a los mensajes de entrenamiento de cierta categoría C. En el primer nivel, se consideran dos categorías (que para nuestra discusión denominamos Candidato A o Candidato B), mientras que en el segundo nivel se comparan los mensajes de prueba con el conjunto de mensajes de entrenamiento en tres categorías diferentes (C_{+1}, C_0 y C_{-1}).

  De este modo, se calculan las distancias de la *matriz de mensajes-mensajes* para cada categoría C, como puede verse en el Cuadro 6.2. A continuación, se ordenan estas distancias para cada categoría, de modo que se pueda calcular el percentil p que maximiza el rendimiento del clasificador. Así, tomando este percentil como referencia de la distancia a cada clase, se asigna finalmente el mensaje de prueba a la categoría cuya distancia es menor.

Si el mensaje de entrenamiento se encuentra a la misma distancia de dos categorías, no se asigna a ninguna y se almacena para ser evaluado manualmente. Una descripción más extensa puede verse en [35].

- *Clasificación utilizando SVM*: En este caso, se ha seguido el método SVM para clasificar los mensajes de los Candidatos A o B (primer nivel), e identificar dichos mensajes siguiendo diferentes tendencias de opinión, pertenecientes a las categorías C_{+1}, C_0 o C_{-1} (segundo nivel). En este caso, hemos utilizado un núcleo lineal para esta tarea, habiéndose obtenido resultados similares utilizando el núcleo radial. Este método está disponible en varios paquetes del entorno estadístico R, del que hemos seleccionado la librería *e1071*.

Diccionario	Prueba 1	Prueba 2	...	Prueba t
Entrenamiento 1	d_{11}	d_{12}	...	d_{1t}
Entrenamiento 2	d_{21}	d_{22}	...	d_{2t}
Entrenamiento 3	d_{31}	d_{32}	...	d_{3t}
...	...	...	...	...
Entrenamiento a	d_{a1}	d_{a2}	...	d_{at}

Cuadro 6.2: Matriz mensaje-mensaje generada por el método k-NN para una categoría C. d_{at} representa la distancia entre el mensaje de entrenamiento a y el mensaje de prueba t.

Finalmente, procedemos a la extracción de opiniones. En este nivel, nuestro objetivo es obtener las palabras relevantes que caracterizan los comentarios asignados a las diferentes tendencias de opinión (positivo, negativo o neutro) catalogadas para el Candidato A o el Candidato B. Con este objetivo, parece ra-

zonable utilizar métodos de aprendizaje no supervisados, pues no tenemos categorías o temas de interés predefinidos que podamos utilizar. Para este propósito, uno de los algoritmos de agrupamiento comúnmente utilizado es el método de k-medias, véase [30].

En nuestro caso, aplicamos el método de k-medias (con $k = 4$) para identificar un número variable de grupos de mensajes en cada categoría (C_{+1}, C_0 y C_{-1}), dependiendo del número de mensajes asignados a dicha categoría. Seguidamente, se inspeccionan los mensajes prototipo que actúan como centroides de los grupos para obtener las palabras que caracterizan el contenido de los mismos. Así, obtenemos las palabras más relevantes definidas en los comentarios enviados por los participantes sobre cada candidato para elaborar las tendencias de opinión.

6.3. Presentación y resultados del experimento

En esta sección, se presentan los resultados del experimento comparando diferentes métodos para la clasificación y el análisis de sentimientos del procedimiento multifase explicado anteriormente. En este caso, se han simulado diez repeticiones para cada uno de los dos primeros niveles (clasificación de los mensajes y tendencias de opinión).

6.3.1. Datos del problema

Los mensajes utilizados para realizar el estudio de la metodología presentada pertenecen al foro de debate de un experimento propuesto con la arquitectura presentada en el Capítulo 2. En dicho experimento, se realizaron diversas cuestiones para conocer las opiniones de los participantes sobre dos posibles representantes (Candidato A y Candidato B) a unas futuras elecciones en una asociación de empresarios. En el Cuadro 6.3 se muestran algunas estadísticas generales sobre el número de mensajes analizados y el número medio de palabras de estos mensajes.

Estadísticos descriptivos	Valor
Número de mensajes analizados	483
Número medio de palabras por mensaje	39.83
Número medio de palabras por mensaje (sin palabras vacías)	16.14
Número total de mensajes del Candidato A	252
Número total de mensajes del Candidato B	231

Cuadro 6.3: Estadísticas generales del caso analizado.

Del mismo modo, en el Cuadro 6.4 podemos ver algunos ejemplos de mensajes clasificados en cada una de las diferentes categorías consideradas en el análisis.

6.3.2. Métricas de rendimiento para clasificación

Para evaluar el rendimiento de los diferentes métodos de clasificación de textos y de análisis de sentimientos, se han utilizado las métricas *precision*, *recall* y *accuracy* para la recuperación

Categoría opinión	Mensaje de muestra
C_{+1}	*Candidato A* te apoyamos, cuenta con mi voto para las próximas elecciones.
C_0	Me gustaría saber la opinión del *Candidato A* sobre la subida de impuestos a los empresarios de la zona.
C_{-1}	Pienso que el *Candidato A* es una mala persona, él solamente quiere mandar.
C_{+1}	De hecho, pienso que el *Candidato B* ofrece una buena imagen para el resto de empresarios.
C_0	El *Candidato B* ya nos representó en años anteriores.
C_{-1}	£Es cierto que algunas personas no ven que el *Candidato B* esta dañando al resto de empresarios?

Cuadro 6.4: Ejemplo de mensajes asignados a cada categoría considerada en el caso de uso.

de información, véase [5] y [37]. Para ello, definimos *verdaderos positivos* (t_p), como el conjunto de mensajes correctamente clasificados; *verdaderos negativos* (t_n), como el conjunto de mensajes que correctamente no se han catalogado en cierta clase; *falsos positivos* (f_p) a los mensajes que han sido incorrectamente clasificados en una categoría y *falsos negativos* (f_n) a aquellos mensajes catalogados incorrectamente en una categoría que no era la suya. Como consecuencia, se definen las métricas:

- *Precision* para la clase C_i: Es la fracción de mensajes asignados a la clase C_i que pertenecen a esa clase.

$$Precision = \frac{t_p}{t_p + f_p}$$

- *Recall* para la clase C_i: Es la fracción de mensajes que deberían pertenecer a la clase C_i que se clasifican correctamente.

$$Recall = \frac{t_p}{t_p + f_n}$$

- *Accuracy*: Es la proporción de mensajes correctamente catalogados sobre todos los mensajes disponibles.

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n}$$

6.3.3. Clasificación de mensajes

En el primer nivel, nuestro objetivo era comparar el rendimiento de los algoritmos k-NN y SVM para la clasificación de mensajes, diferenciando entre comentarios dirigidos al Candidato A y al Candidato B. La Figura 6.3 resume los resultados para este caso. Como se podría esperar de antemano, los resultados para el método SVM son mejores que los del método k-NN, aunque por pequeño margen. El hecho de tener unos temas bien definidos y precisos para la clasificación de mensajes facilita la tarea a ambos métodos de aprendizaje, ya que se encuentra una clara distinción entre los mensajes dirigidos al Candidato A o al Candidato B.

Otro resultado destacable es la baja variabilidad en cualquiera de las tres medidas de precisión en ambos métodos. Esto indica que la estimación de las métricas de rendimiento es bastante acertada y existe escasa influencia por la selección de diferentes muestras de mensajes en cada experimento.

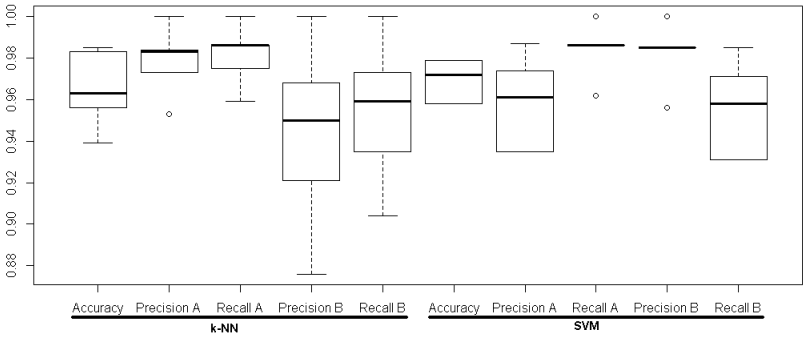

Figura 6.3: Rendimiento de algoritmos de clasificación de mensajes en el primer nivel.

6.3.4. Identificar tendencias de opinión

En el segundo nivel se compara el rendimiento de los métodos de aprendizaje supervisado para la clasificación de mensajes en cada una de las tres posibles categorías de opinión: C_{+1}, C_0 o C_{-1}. En la Figura 6.4 se muestran los resultados obtenidos en los experimentos para los mensajes clasificados como del Candidato A. Para el Candidato B se obtuvieron resultados similares. A pesar de tener tres categorías en esta etapa, se resumen los resultados mediante la comparación de dos categorías, C_{+1} y C_{-1}, puesto que son nuestro foco de atención para comparar tendencias de opinión positivas o negativas.

Como se observa, los resultados son bastante buenos, pero no tanto como en el primer nivel. La identificación de tendencias de opinión es más difícil que la categorización de texto, puesto que la distinción de los mensajes relacionados con alguna de las tres categorías consideradas puede depender sólo de unas pocas palabras. Además, la variabilidad en las estimaciones

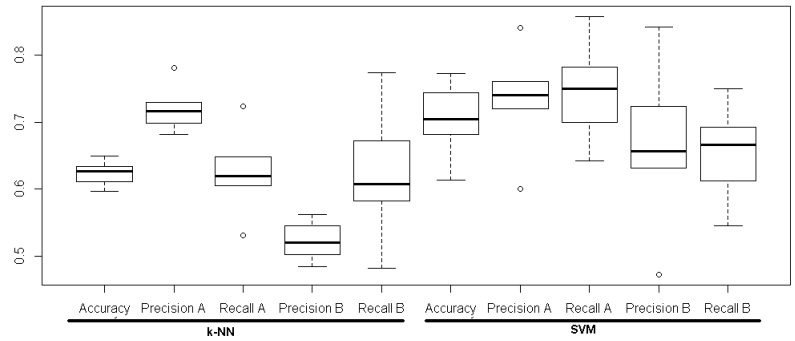

Figura 6.4: Métricas de rendimiento para el segundo nivel del método multifase (sin palabras vacías), para los mensajes asignados al Candidato A.

continúa siendo bastante buena, como indican los pequeños valores obtenidos en la desviación típica.

Por otra parte, los mensajes escritos en el foro son muy cortos cuando las palabras vacías u otra información textual superflua es eliminada. Debido a ello, la frecuencia de aparición de las palabras importantes para detectar tendencias de opinión es mucho más baja que en problemas típicos de categorización de textos donde los documentos son normalmente largos y los términos que caracterizan cada tema están más definidos. Por esta razón, es importante saber si podríamos mejorar nuestros resultados manteniendo las palabras vacías que son normalmente filtradas en los métodos tradicionales de clasificación de textos. Así, decidimos repetir el experimento con las palabras vacías en un intento de mantener términos que pudieran servir para discriminar diferentes tendencias de opinión y que pudieran mejorar nuestro experimento anterior.

En la Figura 6.5 se muestran los nuevos resultados mante-

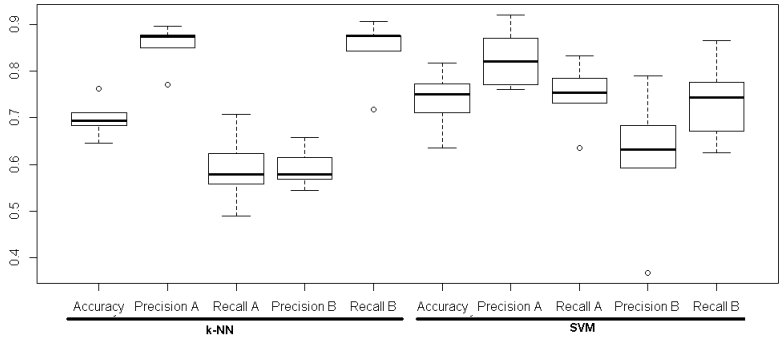

Figura 6.5: Métricas de rendimiento para el segundo nivel del método multifase (con palabras vacías) para los mensajes asignados al Candidato A.

niendo palabras vacías. En general, se observa que mejoran en ambas categorías aproximadamente un 10-15 %, excepto para el recall de la categoría C_{+1} y la precisión de la categoría C_{-1}. Esto sugiere que el hecho de eliminar palabras vacías influye en la detección de tendencias de opinión, especialmente en los casos en los que se trabaja con mensajes cortos y baja frecuencia de aparición de cada término en los documentos. Por ejemplo, se filtran los términos que claramente contienen significado positivo o negativo como *bueno*, *malo* o *mejor* en el caso del primer experimento. Esto muestra cómo eliminando palabras vacías que no tienen ninguna influencia en la categorización clásica de grandes documentos puede tener un impacto negativo en la detección de opiniones, ya que desaparecen términos que pueden ayudar a encontrar las mismas.

6.3.5. Extracción de opiniones y palabras relevantes

Para concluir el análisis, se aplica el método k-medias para identificar los grupos de mensajes clasificados en las categorías C_{+1}, C_0 y C_{-1} para los Candidatos A y B. Para simplificar, sólo se muestran los resultados de la versión SVM del esquema multifase, puesto que los resultados del método k-NN son similares. Los cuadros 6.6 y 6.5 muestran las listas de términos que caracterizan los diferentes grupos identificados para cada tendencia de opinión. En este caso, se han mantenido las palabras vacías para minimizar la probabilidad de eliminar términos destacables que pudieran definir cada tendencia. En el caso del Candidato A, el número de mensajes positivos era menor que en las otras dos clases y sólo se han identificado dos grupos. Para el Candidato B, no se ha identificado claramente ningún grupo para los mensajes positivos, y sólo se han identificado dos grupos para los mensajes de la categoría C_0. En el resto de casos, se han identificado 4 grupos diferentes de mensajes.

Examinando las palabras de cada grupo, resulta sencillo encontrar algunos términos relevantes que caracterizan los diferentes temas de opinión. Entre ellos, existen referencias a temas económicos y de enseñanza, procesos burocráticos y enlaces a un proceso electoral. En este sentido, se observa cómo podemos aplicar esta técnica para obtener más información acerca de los conceptos e ideas asociadas a cada corriente de opinión. La investigación en profundidad de estos resultados puede llevar a una lista más elaborada de términos importantes relacionados

con cada categoría. De hecho, esta lista puede utilizarse como entrada de datos para mejorar la toma de decisiones en política o encontrar los intereses que más preocupan a los usuarios que participan en el debate. Además, esta lista puede utilizarse para generar los informes que expliquen las decisiones tomadas para aumentar el grado de satisfacción de los usuarios.

	Grupo 1	Grupo 2	Grupo 3	Grupo 4
A/C_{+1}	felicidades, excelente, documento, elegido	árbol, bueno, socios, correcto	N/A	N/A
A/C_0	promesas, empresa, reunión, declaración	ataque, empresarios, arbitrario, confortable	empleados, calidad, sentencia, adherirse	elecciones, paz, amigos, brillante
A/C_{-1}	oficinas, artículo, católico, creyente	sede, carta, culto, disciplinado	advertir, barreras, corrupción, difamatorio	dinero, actividades, conservador gerente

Figura 6.6: Términos que caracterizan los grupos identificados por el algoritmo k-medias, manteniendo palabras vacías para el candidato A. *N/A* representa *No disponible*.

6.3.6. Comparación de experimentos

Las investigaciones previas en análisis de sentimientos y minería de opiniones se han centrado usualmente en la comparación de métodos para mejorar el rendimiento en la detección de corrientes de opinión o en la clasificación de mensajes. Sin

	Grupo 1	Grupo 2	Grupo 3	Grupo 4
B/C_{+1}	N/A	N/A	N/A	N/A
B/C_0	juez, amigos, comisión, mandato	asociado, divertido, piensa, por favor	N/A	N/A
B/C_{-1}	sentencia, elecciones, departamento justicia	admitido, contrario, denunciado, prohibir	reunión, consejo, compasión, responsable	explicación, fundación, grave, acoso

Cuadro 6.5: Términos que caracterizan los grupos identificados por el algoritmo k-medias, manteniendo palabras vacías para el candidato B. *N/A* representa *No disponible*.

embargo, las dependencias entre la categorización de temas y el análisis de sentimientos rara vez han sido exploradas.

Hemos mostrado un método multifase que puede adoptarse en aquellos casos en los que resulte necesaria la clasificación textual de documentos, de acuerdo a los temas relevantes de opinión y considerando el análisis de sentimientos para la clasificación. Este procedimiento aporta importantes ventajas:

- Modularidad en la clasificación de textos. En lugar de realizar al mismo tiempo la categorización de textos y la detección de tendencias de opinión, se pueden implementar diferentes procedimientos en cada nivel de acuerdo a las necesidades específicas. Siguiendo esta aproximación, se pueden aprovechar los métodos precisos para la categorización de contenidos, obteniendo un rendimiento superior en esta tarea y, a continuación, centrarnos en el

análisis de sentimientos para cada categoría identificada en el primer nivel.

- Requisitos de acierto para el análisis de sentimientos. Puesto que se ha obtenido un buen rendimiento en la clasificación de contenido en el primer nivel, la detección de tendencias de opinión puede ser satisfactoria con unos buenos resultados de rendimiento.

- Combinación de métodos supervisados y no supervisados. La combinación adecuada de ambas técnicas proporciona resultados más satisfactorios que la sola aplicación de métodos supervisados de aprendizaje. Con este enfoque, no sólo somos capaces de identificar diferentes corrientes de opinión, sino que conseguimos caracterizar grupos de mensajes relacionados por la tendencia de acuerdo a su contenido. Esto claramente proporciona más información que un mero análisis cuantitativo de la cantidad de mensajes asignados a cada categoría.

Del mismo modo, podemos ofrecer una serie de recomendaciones a partir de los resultados y la experiencia práctica obtenida a través de estos experimentos:

- Clasificar mensajes cortos es un reto: La clasificación de mensajes cortos no es comparable a otros tipos de documentos como libros, artículos, noticias, etc. Esto es especialmente relevante para la información textual extraída de páginas web o servicios de micro-blogging.

- Alto rendimiento de las SVM: En general, el uso de métodos de clasificación SVM ofrece mejores resultados de

rendimiento que los métodos k-NN, tanto en categorización de textos como en análisis de sentimientos. Sin embargo, esta diferencia no es muy notable. En cualquier caso, el escaso tiempo necesario para entrenar una SVM comparado con k-NN y su rapidez de computación, especialmente con grandes conjuntos de documentos, son ventajas importantes.

- Influencia de las palabras vacías: Los métodos tradicionales para la categorización de textos recomiendan la eliminación de palabras vacías como paso fundamental para mejorar el rendimiento de estas técnicas. Sin embargo, como hemos visto en estos experimentos, algunos términos identificados como palabras vacías por las librerías de análisis textual pueden contener información válida desde el punto de vista del análisis de sentimientos. Por lo tanto, las palabras vacías no deben eliminarse sin tener en cuenta qué términos podrían mejorar la tasa de acierto en la detección de tendencias de opinión.

6.4. Conclusiones

En este capítulo se ha presentado una metodología multifase de análisis de sentimientos y extracción de opiniones combinada con algoritmos y técnicas de aprendizaje supervisado y no supervisado. El objetivo final es la detección de tendencias de opinión positivas y/o negativas de los mensajes obtenidos del foro de discusión presentado en nuestra arquitectura. El esquema ha sido probado exitosamente con los mensajes de

un foro, aportando información valiosa al dueño del problema, facilitando la explicación de las decisiones tomadas. El diseño de la herramienta descrita en este capítulo es bastante flexible permitiendo la detección de cambios en las tendencias de opinión en un corto espacio de tiempo, de modo que, pueda adaptarse a actualizaciones en tiempo real de las tendencias cuando llegan al sistema nuevos mensajes.

Las tendencias de opinión se representan mediante vectores prototipo de palabras que pueden interpretarse por los responsables del problema. El esquema mostrado puede adaptarse a cualquier foro específico utilizando, por ejemplo, métodos alternativos tales como el análisis de sentimientos latente o mediante la construcción de diccionarios específicos para dicho análisis, además de utilizar términos derivados de los documentos bajo estudio.

La herramienta propuesta puede utilizarse para la generación de informes una vez que ha concluido el proceso que permita explicar a los participantes las conclusiones obtenidas. Por lo tanto, estaría enlazada con el módulo de explicación contenido en la arquitectura presentada en capítulos anteriores. También se puede usar directamente, por ejemplo, para la detección de tendencias de opinión populares en campañas electorales o para conocer el grado de satisfacción con nuevas leyes o políticas públicas. Del mismo modo, también puede adoptarse en otras aplicaciones tales como la categorización automática y detección de mensajes relacionados con campañas de publicidad, productos y servicios comerciales, así como las críticas u opiniones a páginas web.

Capítulo 7

Conclusiones y trabajo futuro

En el trabajo de esta tesis se ha estudiado una técnica divide y vencerás que divide el espacio de entrada de un clasificador en regiones de Voronoi y permite realizar el entrenamiento de subproblemas paralelamente, disminuyendo el tiempo de entrenamiento y permitiendo la ejecución en centros de computación de alto rendimiento, como Altamira, explotando al máximo los recursos que estos ofrecen.

Aunque los centros de computación de alto rendimiento son el entorno idóneo para la ejecución de algoritmos de clasificación sobre grandes volúmenes de datos (big data), muchas aplicaciones en sistemas embebidos o telefonía móvil [17] utilizan clasificadores y podrían beneficiarse de las técnicas expuestas en esta tesis usando arquitecturas similares a las expuestas en el capítulo 4 y en [8, 9].

Las técnicas bioinformáticas para identificar genes y clasificarlos como cancerígenos [16, 43] también pueden beneficiarse de las técnicas de paralelismo como se vio en el capítulo 5, reduciendo el tiempo de entrenamiento.

Gracias a la masificación del acceso a Internet y la aparición de las redes sociales, la minería de datos se ha convertido en una pieza clave para obtener datos estadísticos tales como el sentimiento que tiene la población sobre cierto tema, como por ejemplo leyes, marcas comerciales o candidatos a un puesto de responsabilidad pública sin necesidad de realizar encuestas [1]. Estas aplicaciones pueden implementarse utilizando las técnicas de paralelismo expuestas en esta tesis como se ha explicado en el capítulo 6.

7.1. Aportaciones de la tesis

1. Revisión del estado de los algoritmos de clasificación actuales, especialmente SVM.

2. Diseño y desarrollo de una técnica de paralelización aplicable a clasificadores basada en la división del espacio de entrada en regiones de Voronoi.

3. Revisión del estado de los algoritmos de clasificación paralelizables actuales.

4. Implementación de un sistema hardware de clasificación basada en SVM con paralelización usando conglomerados.

5. Implementación de una librería de clasificación basada en SVM con paralelización usando conglomerados para computadores de alto rendimiento (libpSVM).

6. Caso práctico: análisis de sentimiento de textos en foros.

7.2. Trabajo futuro

El trabajo de investigación mostrado en esta tesis ha permitido fijar algunas líneas de investigación y trabajo futuro, en algunas de ellas ya se está trabajando en la actualidad.

1. Implementación dispositivo hardware espacífico de pSVM. En el capítulo 4 se mostró una arquitectura harware basada en el procesador softcore Xilinx Microblaze. Esta plataforma es un prototipo que serviría como primer paso para el diseño y desarrollo de un dispositivo hardware que permitiese la ejecución del algoritmo SVM con particionado o pSVM. Esta línea de investigación está propuesta y protegida con la patente "Procedimiento y dispositivo de clasificación para grandes volúmenes de datos WO 2013186402 A1" [9]

2. Implementación en CUDA. Como se ha demostrado a lo largo de la tesis, el algoritmo explota la independencia de datos al dividir el espacio de entrada y permite el paralelismo. Actualmente, las tarjetas gráficas permiten utilizar su arquitectura para computación de propósito general, gracias a la tecnología CUDA, para ejecutar algoritmos con un grado de paralelismo sin precedente.

Además, los grandes centros de computación de alto rendimiento disponen de nodos de computación con tarjetas gráficas que permiten la implementación de algoritmos heterogéneos y distribuidos, lo que permitiría calcular las distancias utilizadas por el algoritmo SVM en paralelo, acelerando más todavía el tiempo de entrenamiento.

3. Diseño de una metodología similar a la planteada en esta tesis para el problema de SVM de regresión. Posiblemente el sistema de particionado por regiones de Voronoi sería válido para la SVM-R, pero sería necesario encontrar un nuevo esquema de votación o modificar el propuesto en esta tesis.

4. Selección de kernel en cada subSVM. Actualmente libpSVM utiliza el mismo kernel para todos los subproblemas. Una línea de investigación futura podría añadir alguna técnica y desarrollar un algoritmo que permitiera seleccionar el kernel más apropiado para cada subSVM y analizar el impacto de esta técnica sobre el tiempo de entrenamiento y los resultados de la clasificación.

5. Optimización de hiperparámetros en cada subSVM. Actualmente libSVM contiene funciones que permiten realizar búsquedas GRID para optimizar los parámetros del kernel de la SVM [10]. Se podría realizar la búsqueda de estos hiperparámetros para cada subproblema y analizar su impacto en el tiempo de entrenamiento y resultados de la clasificación.

Bibliografía

[1] César Alfaro, Javier Cano Montero, Javier Gómez, Javier M Moguerza, and Felipe Ortega. A multi-stage method for content classification and opinion mining on weblog comments. *Annals of Operations Research manuscript*, 2013. 7

[2] N S Altman. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3):175–185, 1992. 1.1

[3] N Aronszajn. Theory of Reproducing Kernels Author. *Transactions of the American Mathematical Society , Vol . 68 No . 3*, 68(3):337–404, 1950. 2.1.2

[4] K Bache and M Lichman. {UCI} Machine Learning Repository, 2013. 5.3.3

[5] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition) (ACM Press Books)*. Addison-Wesley Professional, 2011. 1.1.1, 3.3, 6, 6.1.1, 6.3.2

[6] Bao-Liang Lu, Kai-An Wang, and Yi-Min Wen. Comparison of parallel and cascade methods for training support vector machines on larege-scale problems. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 5, pages 3056–3061. IEEE, 2004. 3.4.6

[7] M. S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear programming: theory and algorithms*. 1993. 2.1.2

[8] J. Cano, J. Castillo, J. Moguerza, P Huerta, and J. I. Martinez. .Algoritmo de entrenamiento de SVM mediante clusters generados con K-Medias: KETSVM",. In *Jornadas de Computación Reconfigurable y Aplicaciones*, 2011. 7

[9] Javier Cano Montero, Javier Martínez Moguerza, Javier Castillo Villar, José Ignacio Martínez Torre, and David Ríos Ínsua. Procedimiento y dispositivo de clasificación para grandes volúmenes de datos, December 2013. 7, 1

[10] Chih-Chung Chang and Chih-Jen Lin. {LIBSVM}: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1—-27:27, 2011. 3.1.2, 3.2.1, 3.2.2, 5

[11] Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. *Neural computation*, 14(5):1105–14, May 2002. 3.4.1

[12] Thomas M. Cover. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications

in Pattern Recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, June 1965. 2.1.1

[13] P.A. Forsyth. *PC/BEAGLE UserâĂŹs Guide*. 1990. 5.3.6, 5.4

[14] Ian Foster. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. January 1995. 5.1

[15] Jerome H Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1996. 2.2

[16] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1-3):389–422, January 2002. 7

[17] Chris Harrison, Munehiko Sato, and Ivan Poupyrev. Capacitive fingerprinting. In *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, page 537, New York, New York, USA, October 2012. ACM Press. 7

[18] P. Hart. The condensed nearest neighbor rule (Corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, May 1968. 1.1.1

[19] Marti A. Hearst, ST Dumais, and E Osman. Support Vector Machines. *. . . Systems and their . . .*, 1998. 2.1

[20] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 13(2):415–25, January 2002. 2.2

[21] W. Huang, G. Santhanaraman, H.-W. Jin, Q. Gao, and D.K. Panda. Design of High Performance MVAPICH2: MPI2 over InfiniBand. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, volume 1, pages 43–48. IEEE, 2006. 5.2

[22] V V Ivanov. *The theory of approximate methods and their application to the numerical solution of singular integral equations*. 1976. 2.1.2

[23] Jian-pei Zhang, Zhong-Wei Li, and Jing Yang. A parallel SVM training algorithm on large-scale classification problems. In *2005 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1637–1641 Vol. 3. IEEE, 2005. 3.4.4

[24] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, July 2002. 3.2.2

[25] Ulrich H.-G. Kressel. Pairwise classification and support vector machines. pages 255–268, February 1999. 2.2, 3.2

[26] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. Introduction to parallel computing: design and analysis of algorithms. January 1994. 5.1

[27] Yun Li and Li-Li Feng. Integrating feature selection and Min-Max Modular SVM for powerful ensemble. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, June 2012. 2

[28] Yi Lin, Grace Wahba, Hao Zhang, and Yoonkyung Lee. Statistical Properties and Adaptive Tuning of Support Vector Machines. *Machine Learning*, 48(1-3):115–136, July 2002. 2.1.2

[29] Bing Liu. Sentiment Analysis and Opinion Mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, May 2012. 6

[30] K. V. Mardia, John T. Kent, and John M. Bibby. *Multivariate analysis*. 1979. 6.2.2

[31] David H Mathews and Douglas H Turner. Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. *Journal of molecular biology*, 317(2):191–203, March 2002. 5.3.4

[32] Ines A. Mergel, Charles M. Schweik, and Jane E. Fountain. The Transformational Effect of Web 2.0 Technologies on Government. *SSRN Electronic Journal*, June 2009. 6

[33] Baback Moghaddam. Classification with boosted dyadic kernel discriminants, October 2003. 3.4.2

[34] Javier M. Moguerza and Francisco J. Prieto. An augmented Lagrangian interior-point method using directions of negative curvature, March 2003. 2.1.2

[35] Alberto Muñoz and Javier M. Moguerza. Building Smooth Neighbourhood Kernels via Functional Data Analysis. pages 631–636, January 2005. 6.2.2

[36] Florian Nigsch, Andreas Bender, Bernd van Buuren, Jos Tissen, Eduard Nigsch, and John B O Mitchell. Melting point prediction employing k-nearest neighbor algorithms and genetic parameter optimization. *Journal of chemical information and modeling*, 46(6):2412–22, January 2006. 1.1.1

[37] David L. Olson and Dursun Delen. Advanced Data Mining Techniques. March 2008. 3.3, 6.3.2

[38] Tim O'Reilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. 6

[39] David L Phillips. A technique for the numerical solution of certain integral equations of the first kind. *Journal of the ACM*, 9:84–97, 1962. 2.1.2

[40] John C. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. 1998. 3, 3.1.2, 5.3.3, 5.4

[41] Alexander Smola, Chris Burges, Harris Drucker, Steve Golowich, Leo Van Hemmen, Klaus-Robert Müller, Bernhard Schölkopf, and Vladimir Vapnik. Regression Estimation with Support Vector Learning Machines. 2.3, 2.3

[42] Andrei Nikolaevich Tikhonov and Vasili Yakovlevich Arsenin. *Solutions of ill-posed problems*. Winston, 1977. 2.1.2

[43] Andrew V Uzilov, Joshua M Keegan, and David H Mathews. Detection of non-coding RNAs on the basis of predicted secondary structure formation free energy change. *BMC bioinformatics*, 7:173, January 2006. 5.3.4, 5.4, 7

[44] Vladimir N. Vapnik. The nature of statistical learning theory. June 1995. 1.4

[45] Yi-Min Wen and Bao-Liang Lu. *Advances in Neural Networks âĂŞ ISNN 2007*, volume 4493 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, June 2007. 3.4.5

[46] Ian H. Witten and Eibe Frank. Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems). June 2005. 6.2.1

[47] William H. Wolberg, Olvi L. Mangasarian, and Ph. D. Breast Cytology Diagnosis Via Digital Image Analysis. January 1999. 5.3.5, 5.4

[48] Bao-liang Lu Yi-min Wen. A Confident Majority Voting Strategy for Parallel and Modular Support Vector Machines. 3.4.3