UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD #6

"Predicción de actividad biológica para el procesamiento masivo de datos utilizando Máquinas de Soporte Vectorial"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Yendry Álvarez Díaz
Julio César Cairo Vignier

Tutores: Dr. Ramón Carrasco Velar
Ing. Andy Machín González

Ciudad de la Habana, Cuba.
Junio ,2009

"Año del 50 Aniversario del triunfo de la Revolución."

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Julio César Cairo Vignier

_____

Firma del Autor

Yendry Alvarez Díaz

_____

Firma del Autor

Ing. Andy Machín González

_____

Firma del Tutor

Dr. Ramón Carrasco Velar

_____

Firma del Tutor

DATOS DE CONTACTO

Tutores:

Dr. Ramón Carrasco Velar

Centro de Química Farmacéutica, Ciudad de La Habana, Cuba.

ramon.carrasco@cqf.sld.cu

rcarrasco@uci.cu

Ing. Andy Machín González

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

amachin@uci.cu

"El día que el hombre se diese cuenta de sus profundas equivocaciones, habría terminado el progreso de la ciencia."

Marie Curie

AGRADECIMIENTOS

Corriendo el riesgo de alguien se nos quede en el olvido queremos agradecerle sinceramente a todas aquellas personas que de una forma u otra contribuyeron a la construcción de esta tesis. Contar a las personas que nos auxiliaron sería interminable, ya que por suerte contamos con mucha gente que siempre estuvo ahí sin pedir nada a cambio es por esto que:

Agradecemos a nuestros padres por el apoyo insustituible y mantenernos motivados para terminar el camino.

A nuestros tutores Carrasco y Machín por asesorarnos y compartir su conocimiento e inspirar en nosotros mucha admiración.

A Dismey gracias por dedicarnos todo ese valioso tiempo.

A Dachelys Valdés Y Darmalina aunque no formaron parte oficial de esta investigación fueron de vital importancia para su desarrollo a ustedes muchas gracias.

A nuestros amigos por brindarnos todo su cariño.

A todos los que de una manera u otra nos brindaron su apoyo.

Al pueblo de Cuba por existir.

DEDICATORIA

De Yendry:

Quisiera dedicar esta tesis a mis padres y a mi hermana por estar presente en cada momento de mi vida y por haberme brindado su amor, cariño y sobre todo su apoyo. Decirles que siempre serán la inspiración para alcanzar mis metas y que todo su esfuerzo no fue en vano, gracias a ustedes he podido superar todos los obstáculos y convertirme en la persona que soy hoy, a ustedes los quiero. Dedicársela también a René, pues se que me apoya y puedo contar con el en todo momento. A Jesús mi hermanito mayor que ha estado siempre alerta ante cualquier problema que se me pueda presentar, gracias por tu ayuda.

En fin a toda esa gente linda que me quiere como mis tíos y tías especialmente a Estela, Llilli y Zeida, a mis primos especialmente a Dayanis y Pedro Daniel que siempre los recuerdo, a mis compañeros que formaron parte de mi vida estos 5 años: Nigdel, Isbel, Yanoski, Jose Rolando, Jorge (Zeus), Yadir, Yoamel, Yexenia, a mi amigo de infancia Androvis que a pesar de la distancia siempre ha estado pendiente del desarrollo de este trabajo. También a mis abuelitos que se que me miran desde algún lado y que están orgullosos de mi. A todos ustedes gracias y esta tesis también es para ustedes.

De Julio César:

Dedico este sueño hecho realidad a mi madre que ha sido responsable de haber podido concluir mi carrera, por haberme dado ese toque de magia para seguir adelante, por su apoyo constante y amor infinito. A mi abuela del alma por malcriarme y a la vez ser tan recta y fuerte por enseñarme el verdadero valor de las cosas y a pensar positivamente. A mi tía Tere y a mi tío Tony sé que siempre puedo contar con ustedes. A mi padre, a mis tías Mercedita y Cachita por siempre estar al tanto y enviarme energía positiva. A mi abuelo Cuco que hoy no está, a mis abuelos paternos Julio y Elsa. A mis primos que son como mis hermanos, a mi hermana Akyana y a mi sobrinita Ziddalín Ziddane, a toda mi familia que siempre estará unida. A mi hermanita Yra la ex gorda, a mis brothers de la infancia Michel y Eydel, a mis hermanos de la 313 por las noches sin dormir cayéndonos a mentiras, a los de l3 y 8, al pikete party también que no me olvido de ustedes. A Darmalina por entenderme como nadie y soportar cada una de mis pesadeces que equivalen a cada trenza que me ha hecho. A todas esas personas que no aparecen aquí y forman parte de mi vida y mi trayectoria en los 5 años de carrera.

RESUMEN

La investigación realizada está dada por la necesidad que existe de dar solución al manejo de grandes volúmenes de datos, utilizando una de las técnicas de inteligencia artificial como son las Máquinas de Soporte Vectorial (MSV). Esta técnica se basa en métodos de aprendizaje estadístico y ha tenido un desarrollo vertiginoso en los últimos años en los campos de la clasificación de imágenes, procesamiento de textos e identificación de patrones entre otros, arrojando excelentes resultados.

Sin embargo, este procedimiento no ha sido capaz de dar el gran paso en el universo de la Bioinformática. Hasta el momento la cantidad de muestras a procesar se ha visto limitado por una amalgama de obstáculos y dificultades ocasionando así una interrupción en el desarrollo y evolución de las Máquinas de Soporte Vectorial.

Sin dudas esto constituye a simple vista una desventaja por lo que el equipo de trabajo se dio a la labor de investigar acerca de métodos que fueran capaces de neutralizar dichos problemas, como es el uso de aplicaciones y librerías ya existentes, las cuales han sido desarrolladas por talentosos equipos de trabajo y prestigiosas universidades y también nuevas tendencias como la programación paralela o concurrente, así como las Máquinas de Soporte Vectorial distribuidas, la cual no ha sido muy tratada en este aspecto.

Índice

Introducción

La predicción de la actividad biológica de compuestos químicos es hoy día un objetivo principal dentro de la Industria Médico Farmacéutica Mundial. El alto costo del proceso de investigación - desarrollo de nuevos fármacos, ha obligado a este sector económico a adoptar la estrategia del uso de técnicas de la computación y la informática para acelerar el proceso y disminuir los costos. Esta predicción de la actividad biológica es un importante eslabón en la producción de nuevos fármacos. Unido esto a los adelantos de las ciencias relacionadas con este campo como son la bioorgánica, la fisiología, la bioquímica, la medicina y las técnicas de computación han acelerado increíblemente el proceso de producción farmacológica a nivel internacional.

Muchos de los más exitosos enfoques que se le ha dado al diseño de fármacos asistidos computacionalmente están basados en la correlación entre estructura química y propiedades de las moléculas. Procesar correctamente los datos y describir la estructura química es el obstáculo a vencer para lograr una predicción efectiva desde el punto de vista informático. Dada su complejidad para identificar las regiones responsables de determinada respuesta, se acostumbra a dividir la molécula en partes.

Para optimizar el proceso anteriormente descrito juega un papel fundamental la búsqueda racional de entidades biológicamente activas candidatas a medicamentos. Sin el empleo de las técnicas de la computación este es un proceso complejo que suele llevar alrededor de una década de investigación y desarrollo con un altísimo gasto en recursos, tras lo cual tan solo un 5 o 10% de las moléculas que llegan a fase de ensayos clínicos terminan siendo comercializadas.

Debido a esto es una necesidad conocer cuáles son los resultados de la predicción de actividad biológica de compuestos orgánicos ante el procesamiento de grandes volúmenes de datos utilizando una técnica de IA.

Desde que se propusieron los primeros algoritmos de Máquinas de Soporte Vectorial actuales por Vladimir Vapnik en la década de los 90 se ha mostrado la capacidad de construir modelos precisos con una relevancia práctica para la clasificación y regresión. Se han reportado aplicaciones satisfactorias que utilizan este tipo de técnica para campos tan variados como el reconocimiento facial,

categorización de textos y Bioinformática. En particular, las Máquinas de Soporte Vectorial que utilizan la idea de sustitución del kernel han demostrado construir buenos modelos, y se han convertido en herramientas de clasificación con un incremento en popularidad.

Sin embargo, a pesar de sus propiedades deseadas, las Máquinas de Soporte Vectorial actuales, no son capaces de manejar fácilmente grandes volúmenes de datos. Un algoritmo estándar de MSV requiere la solución de programas cuadráticos o lineales, de manera que su costo computacional es al menos O (n^2), donde n es el número de puntos de entrenamiento en el conjunto de datos.

Con los recientes avances en la tecnología es posible generar y almacenar grandes volúmenes de datos, estos datos contienen información valiosa que pueden ayudar a muchas personas y son almacenados en bases de datos millonarias, como es el caso de la Farmacología. Por ello se plantea como problema científico de la presente investigación: **¿Cómo predecir con grandes volúmenes de datos la actividad biológica de compuestos orgánicos utilizando las Máquinas de Soporte Vectorial?**

Por otra parte los procesamientos paralelos y distribuidos se han convertido en áreas de gran importancia dentro de la Ciencia de la Computación, produciendo en muchos casos excelentes resultados. La programación paralela considera aspectos conceptuales y particularidades físicas de la computación paralela, su objetivo es mejorar las prestaciones mediante un buen aprovechamiento de la ejecución simultánea. La computación distribuida por su parte aunque es menos ostentosa también reduce el tiempo considerablemente y resuelve el problema del procesamiento masivo de datos por lo que también constituye una parte fundamental en esta investigación.

Con el fin de solucionar el problema planteado anteriormente se definió como objetivo general**: definir algoritmos capaces de predecir actividad biológica en grandes volúmenes de datos utilizando Máquinas de Soporte Vectorial.**

Para dar cumplimiento a dicho objetivo se establecieron como **objetivos específicos analizar los algoritmos de Máquinas de Soporte Vectorial para el cálculo masivo de datos, hacer uso de los algoritmos analizados y validarlos.**

Estructuración por capítulos

Este trabajo cuenta con 3 capítulos y está estructurada de la siguiente forma:

Capítulo 1: Revisión bibliográfica

Se plantea el concepto de Inteligencia Artificial, las Máquinas de Soporte Vectorial como una de sus ramas así como sus principales tipos y definiciones. También se exponen las condiciones de Karush-Kuhn-Tucker (KKT), el problema de la programación cuadrática y el algoritmo Sequential Minimal Optimization y su variante paralela como método para darle solución. Además se enfoca como alternativa la programación paralela y el estándar Message Passing Interface para la comunicación entre procesadores.

Capítulo 2: Programas y metodologías

Los principales programas y metodologías son abordados en este capítulo donde se puede encontrar el Entorno para análisis del conocimiento Weka como la primera solución que se investigó en este trabajo. Las librerías LIBLINEAR y LIBSVM como herramientas para en el ambiente local. El software πSVM y las Máquinas de Soporte Vectorial distribuidas. El lenguaje java y eclipse como entorno de desarrollo y por último las metodologías que se trazaron a lo largo de este trabajo.

Capítulo 3: Resultados y discusión

En este capítulo se plasmaron los resultados obtenidos principalmente de la comparación entre los procedimientos realizados, estos resultados están basados en primera instancia en el rendimiento y exactitud que son capaces de lograr las herramientas utilizadas.

Capítulo 1: Revisión bibliográfica

1.1 Introducción

Antes de comenzar el estudio referente a la predicción de grandes volúmenes de datos aplicando las Máquinas de Soporte Vectorial fue necesario realizar una revisión bibliográfica con el fin de conocer algunas investigaciones que se han hecho en el mismo ámbito. Dichas investigaciones permiten, de alguna manera, fundamentar el estudio que se realizó debido a la relación directa entre ellas.

La Bioinformática es una ciencia frontera en la que intervienen la informática, la biología, la bioquímica, la química, la física, etc. que tiene como objetivo el manejo de grandes volúmenes de información científico-técnica. En este marco se sitúa el trabajo, el procesamiento y la clasificación de grandes volúmenes de datos, aplicando la IA. Para ello se emplea una novedosa técnica denominada Máquinas de Soporte Vectorial. Al finalizar este capítulo se quiere dejar en claro una serie de conceptos que resultarán fundamentales en la solución final.

1.2 Inteligencia Artificial

Se denomina Inteligencia Artificial a la rama de la informática que desarrolla procesos que imitan a la inteligencia de los seres vivos. La principal aplicación de esta ciencia es la creación de máquinas para la automatización de tareas que requieran un comportamiento inteligente. Este es uno entre los más sencillos conceptos de Inteligencia Artificial enmarcado fundamentalmente al enfoque informático. Una de las más novedosas y efectivas técnicas de Inteligencia Artificial, aplicada fundamentalmente al procesamiento de grandes cantidades de información son, las Máquinas de Soporte Vectorial.

1.3 Máquinas de Soporte Vectorial.

Las Máquinas de Soporte Vectorial (MSV), también conocidas como Máquinas de Vectores Soporte, están referidas a una práctica de aprendizaje automático que ha estado ganando popularidad en los últimos años. Los fundamentos teóricos de las MSV empezaron a ser desarrollados por Vapnik y otros autores desde la década de los 70, pero no es hasta el año 1995 cuando Vapnik, en compañía de Boser, Guyon y otros, presentan el modelo formal de las MSV como se conoce hoy día y se pudo comenzar a aplicar en problemas reales de reconocimiento de patrones.

Las MSV pertenecen a la familia de clasificadores lineales puesto que inducen separadores lineales o hiperplanos en espacios de características de muy alta dimensionalidad (introducidas por funciones núcleo o kernel) con un sesgo inductivo muy particular (maximización del margen) (1). Sin embargo, la formulación matemática de las Máquinas de Soporte Vectorial varía dependiendo de la naturaleza de los datos; es decir, existe una formulación para los casos lineales y, por otro lado, una formulación para casos no lineales. Es importante tener claro que, de manera general para clasificación, las Máquinas de Vectores Soporte buscan encontrar un hiperplano óptimo que separe las clases.

En la actualidad, las Máquinas de Soporte Vectorial pueden ser utilizadas para resolver problemas tanto de clasificación como de regresión. Algunas de las aplicaciones de clasificación o reconocimiento de patrones son: reconocimiento de firmas, reconocimiento de imágenes como rostros y categorización de textos. Por otro lado, las aplicaciones de regresión incluyen predicción de series de tiempo y problemas de inversión en general.

1.3.1 Tipos de Máquinas de Soporte Vectorial

Para construir un hiperplano óptimo, las MSV emplean un algoritmo de entrenamiento iterativo el cual es usado para minimizar una función error. Acorde a la forma de la función error los prototipos de MSV pueden ser clasificados en cuatro grupos diferentes.
Clasificación MSV Tipo 1 (también conocido como clasificación C-SVC)
Clasificación MSV Tipo 2 (también conocido como clasificación nu-SVC)
Regresión MSV Tipo 1 (también conocido como regresión épsilon-SVR)
Regresión MSV Tipo 2 (también conocido como regresión nu-SVR)

1.3.1.1 Clasificación C-SVC

Para este tipo de MSV, el entrenamiento involucra la minimización de la función error:

$$\frac{1}{2}w^t w + C\sum_{i=1}^{N}\xi_i$$

Sujeto a las restricciones:

$$y(w^t\phi(x_i) + b) \geq 1 - \xi_i \text{ y } \xi_i \geq 0, i = 1, .., N$$

Donde C es la capacidad constante, w el vector de coeficientes, b es una constante y ξ_i son los parámetros para manejar de los datos de entrada no separables. El índice i etiqueta los casos de entrenamiento N. Tenga en cuenta que $y \in \pm 1$ es la clase etiquetada y x_i es la variable independiente. El kernel ϕ es usado para transformar los datos de entrada al espacio de característica. Cabe señalar que mientras mayor es C, mayor es el error penalizado. Por lo tanto, C deber ser escogido con cuidado para evitar el overfitting.

1.3.1.2 Clasificación nu-SVC

A diferencia de la clasificación C-SVC, el modelo de clasificación nu-SVC minimiza la función error.

$$\frac{1}{2}w^T w - \nu\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i$$

Sujeto a las restricciones:

$$y_i(w^T\phi(x_i) + b) \geq \rho - \xi_i, \xi_i \geq 0, i = 1, \ldots, N \text{ y } \rho \geq 0$$

Por otra parte en una regresión de MSV, se tiene que estimar la dependencia funcional de la variable dependiente “y” en un conjunto de variables independientes x. Se supone que, al igual que otros problemas de regresión, que la relación entre las variables independientes y dependientes está dada por una función f determinista más la adición de algunos ruidos aditivos:

y = f(x) + ruido

La tarea es entonces encontrar una forma funcional para f que pueda predecir correctamente los nuevos casos que las MSV no hayan presentado antes. Esto puede lograrse mediante el modelo de entrenamiento de las MSV establecido en un conjunto de muestras, es decir, un conjunto de entrenamiento, un proceso que implica, al igual que la clasificación (véase más arriba), la optimización secuencial de una función de error. Dependiendo de la definición de esta función de error, dos tipos de modelos de MSV pueden ser reconocidos:

1.3.1.3 Regresión épsilon-SVR

Para este tipo de MSV la función error es:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

la cual se minimiza sujeta a:

$$w^T\phi(x_i) + b - y_i \leq \varepsilon + \xi_i^*$$
$$y_i - w^T\phi(x_i) - b_i \leq \varepsilon + \xi_i$$
$$\xi_i, \xi_i^* \geq 0, i = 1, \ldots, N$$

1.3.1.4 Regresión nu-SVR

Para este tipo de MSV la función error es dada por:

$$\frac{1}{2}w^T w - C\left(\nu\varepsilon + \frac{1}{N}\sum_{i=1}^{N}(\xi_i + \xi_i^*)\right)$$

la cual se minimiza sujeta a:

$$(w^T\phi(x_i) + b) - y_i \leq \varepsilon + \xi_i$$
$$y_i - (w^T\phi(x_i) + b_i) \leq \varepsilon + \xi_i^*$$
$$\xi_i, \xi^* i \geq 0, i = 1, \ldots, N, \varepsilon \geq 0$$

Existen diversos conceptos previos que se deben tener claros antes de entender los fundamentos matemáticos de las MSV. Algunos de dichos conceptos, se presentan a continuación:

1.3.2 Dimensión de Vapnik- Chervonenkis (VC).

La Dimensión de Vapnik Chevonenkis es el número máximo h de vectores z1...zn que se pueden separar de todas las maneras posibles 2h por los hiperplanos, a partir de funciones que miden el mayor número de muestras que pueden ser explicadas por el sistema. Donde para un conjunto de

datos R, la familia de hiperplanos que se genera está dada por el término RD en donde D es al menos D+1 para valores de D mayor que cero. Teniendo como propiedades que:

i) La dimensión de VC *hk* de cada conjunto *Sk* de funciones es finita. Por lo tanto, *h*1≤*h*2...≤*hn*....

ii) Cualquier elemento *Sk* de la estructura contiene un conjunto de funciones.

La relación existente entre los conceptos anteriormente explicados viene dada por:

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{h(\ln(2N/h)+1) - \ln(n/4)}{N}}$$

Donde: N = Número de muestras entrenadas.

R (f)= Riesgo Esperado.

Remp (f)= Riesgo Empírico.

h= Dimensión de Vapnik-Chevonenkis.

Si se realiza un análisis de la misma se puede asegurar que: el modelo más adecuado se encuentra cuando se logra un balance entre el Riesgo Empírico (Remp (f)) y la dimensión de Vapnik-Chevonenkis, este problema se conoce como la minimización del riesgo estructural. Además, a medida que la relación N/h se hace mayor, la confianza Vapnik-Chevonenkis se hace menor y el Riesgo Esperado se acerca al Riesgo Empírico. Una MSV mapea los puntos de entrada en un espacio de características de una dimensión mayor y encuentra el hiperplano óptimo que los separe y maximice el margen m entre las clases, en este espacio.

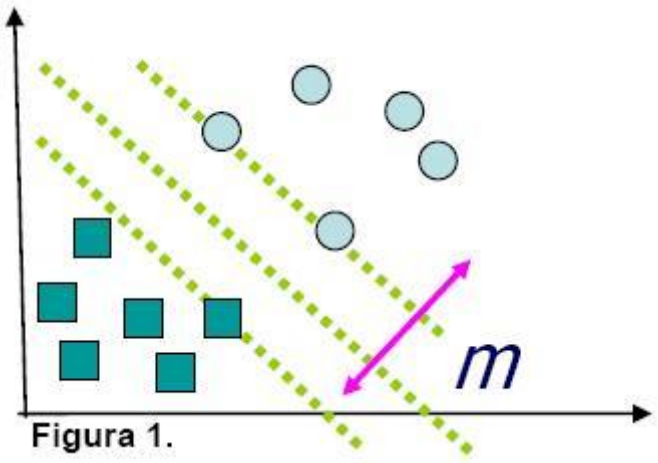

Figura 1.

A partir del conjunto de entrenamiento, donde el dominio de las imágenes se encuentran en el intervalo [-1; 1], las ecuaciones de los hiperplanos H1 y H2 viene dada por: H1= wxi+b=1, H2= wxi+b=-1 y el hiperplano óptimo por wxi+b=0, partiendo de que ||w|| es la norma del vector normal al hiperplano para

las clases que se representan. Por lo tanto el margen m se calcula por la expresión m=2/||w||. Ver Figura 2.

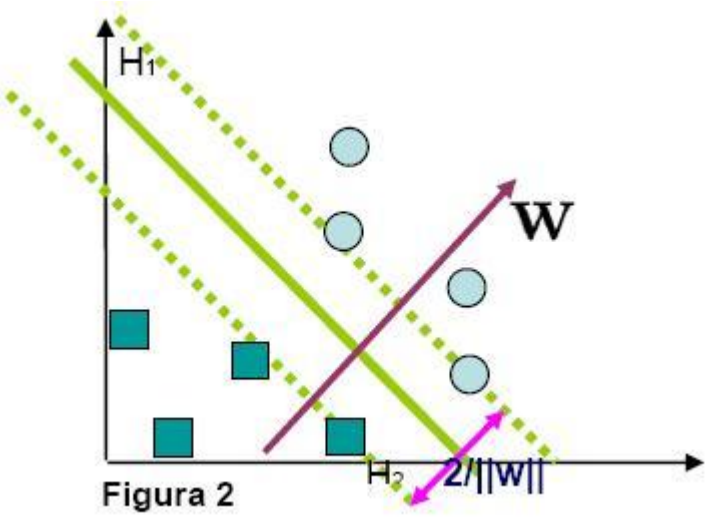

Figura 2

Maximizar dicho margen es un problema de programación cuadrática. Uno de los métodos para resolver el problema es a través de la teoría de LaGrange el cual fue extendido al problema de optimización con restricciones. Los conceptos principales de esta teoría son el concepto de multiplicadores de LaGrange (α) y la función del Lagrangiano (Lp (w, b, α)) relacionadas en la ecuación:

$$L_p(w, b, \alpha) = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^t x_i + b) - 1]$$

Dicha teoría es una generalización del resultado de Fermat en 1629, el cual plantea el problema de optimización sin restricciones. En 1951 Kuhn y Tucker generalizaron la teoría de LaGrange permitiéndose entonces la introducción de restricciones de desigualdad en el problema de optimización. Solo los puntos que estén situados en unos de los hiperplanos cumple que α>0 y se les denomina vectores de soporte, que son los que ayudan a definir el hiperplano óptimo, según la expresión:

$$\frac{\partial J(w,b,\alpha)}{\partial w} = 0 \Longrightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

En el que se utiliza el producto del punto, con funciones en el espacio de características que son llamadas Kernel. La minimización del riesgo empírico y la dimensión de Vapnik-Chervonenkis son fundamentales para lograr la búsqueda de la solución óptima mediante funciones que en un espacio numérico determinado, logre el hiperplano separador. En la representación de estos conjuntos de entrenamiento, los mismos pueden estar linealmente separados. De no ser así, se utilizan las funciones Kernels para llevar estas muestras a un plano de mayor dimensión donde puedan ser linealmente separables. Dentro de los más utilizados se encuentran: el Kernel Polinomial y el de Base Radial (RBF). Las Máquinas de Soporte Vectorial pueden trabajar con dos o más clases en dependencia del problema al que se apliquen.

1.3.3 Maximización del margen

La maximización del margen es la idea que corresponde con exactitud a la aplicación del principio de Minimización de Riesgo Estructural. La maximización del margen, definido como la distancia de las muestras de entrenamiento a la frontera de decisión, se refiere a la selección del hiperplano separador que está a la misma distancia de los ejemplos más cercanos de cada clase. Carreras, Márquez y Romero señalan que lo anterior es equivalente a decir que se debe encontrar el hiperplano separador que está a la misma distancia de los ejemplos más cercanos de cada clase, donde los ejemplos se refieren a los datos utilizados para el entrenamiento (1). Además, mencionan que dicho hiperplano sólo considera los vectores soporte, es decir, aquellos puntos que están en las fronteras de la región de decisión, que es la zona donde puede haber dudas sobre a qué clase pertenece una instancia. El modelo más simple de las MSV, conocido como clasificador de margen máximo, funciona sólo para datos linealmente separables en el espacio de características, por lo que no puede ser usado en muchas aplicaciones de la vida real. Sin embargo, tal como lo afirman Cristianini y Shawe-Taylor (2000), es el algoritmo más fácil de aprender y forma la base fundamental para MSVs más complejas (2). En la figura 3 se muestra geométricamente el hiperplano de margen máximo. La maximización del margen se encuentra dentro de la teoría de aprendizaje estadístico, específicamente dentro del principio de minimización de riesgo estructural.

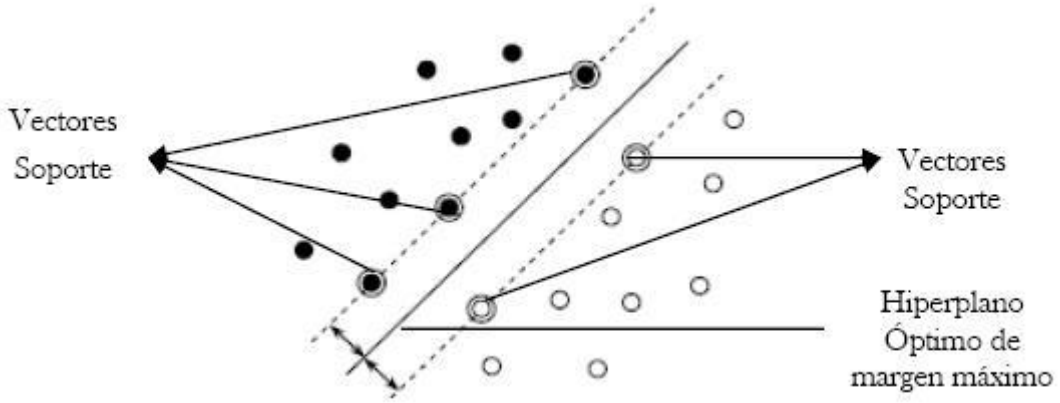

Figura 3 Hiperplano de margen máximo

1.3.4 Máquinas de Soporte Vectorial para el caso lineal

El modelo de MSV lineal con margen máximo es el caso más sencillo de clasificación de dicha técnica. Este caso se basa en el supuesto de que el conjunto de datos es linealmente separable en el espacio de entrada, es decir que, sin hacer ninguna transformación de los datos, los ejemplos pueden ser separados por un hiperplano de manera que en cada lado del mismo sólo queden ejemplos de una clase. En términos matemáticos, esto es equivalente a decir que existe un hiperplano $h: X \rightarrow \Re$ tal que $h(x) > 0$ para los ejemplos de la clase +1 y $(xh) < 0$ para los ejemplos de la clase -1. Un ejemplo de este caso se puede observar en la figura 4.

La distancia de un vector *x* a un hiperplano *h*, definido por (*b,* ω) como $h(x)=<\omega, b> + b$ viene dada por la fórmula $dist\ (h, x)=|h(x)|/||\omega||$, donde $||\omega||$ es la norma en $D\Re$ asociada al producto escalar. Así pues, el hiperplano equidistante a dos clases es el que maximiza el valor mínimo de *dist (h, x)* en el conjunto de datos. Como el conjunto es linealmente separable, se puede reescalar ω y b de manera que la distancia de los vectores más cercanos al hiperplano sea $1/||\omega||$. De esta manera, el problema de encontrar el hiperplano equidistante a dos clases se reduce a encontrar la solución al siguiente problema de optimización:

$$\max \ \frac{1}{||w||}$$

$$\text{sujeto a: } y_i\big((w, x_i) + \mathrm{b}\big) \geq 1$$
$$1 \leq i \leq N$$

Escrito de otra manera, el problema puede ser resuelto a través de la siguiente formulación:

$$\min \frac{1}{2}(w, w)$$
$$\text{sujeto a: } y_i\big((w, x_i) + \mathrm{b}\big) \geq 1$$
$$1 \leq i \leq N$$

Es importante destacar que cualquiera que sea el caso que se presente para la aplicación de las MSV, el problema a resolver es de optimización con restricciones y puede ser resuelto como un problema de programación cuadrática. Particularmente, el hiperplano separador de margen máximo ha demostrado una muy buena capacidad de generalización en numerosos problemas reales (1).

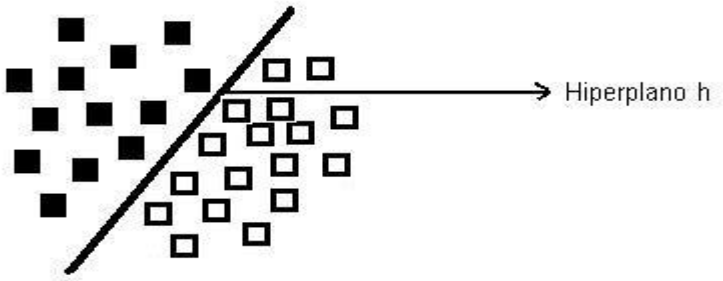

Figura 4: Clasificación de MSV para el caso lineal

1.3.5 Máquinas de Soporte Vectorial para el caso no lineal

Como se mencionó anteriormente, la mayoría de los casos de la vida real son de carácter no lineal y por lo general, incorporan una gran cantidad de información necesaria para ser resuelta. Una de las grandes ventajas de las MSV es que éstas pueden ser utilizadas para resolver problemas no lineales mediante la utilización de funciones kernel, las cuales permiten la solución sin necesidad de utilizar explícitamente algoritmos no lineales. Sin embargo, existen dos casos principales para la solución de datos que no son linealmente separables en el espacio de entradas; el primero, cuando los datos

pueden ser separables con margen máximo pero en un espacio de características y el segundo, cuando no es posible separar dichos datos linealmente incluso en un espacio de características. Ambos casos se explicarán en las subsecciones siguientes.

1.3.5.1 Máquinas de Soporte Vectorial con margen máximo en el espacio de características

Existen casos en los cuales los datos, debido a su naturaleza, no pueden ser separados linealmente a través de un hiperplano óptimo, que es la idea que hay detrás de margen máximo. Sin embargo, en muchas situaciones los datos, a través de una transformación no lineal del espacio de entradas, pueden ser separados linealmente pero en un espacio de características, en el cual se pueden aplicar los mismos razonamientos que para las Máquinas de Soporte Vectorial lineal con margen máximo.

Aunque la dimensión del espacio de características puede ser bastante grande, para ciertas transformaciones y ciertos espacios de características, se puede calcular el producto escalar usando las denominadas funciones kernel. En la figura 5 se puede observar un ejemplo donde se puede encontrar un hiperplano óptimo en el espacio de características, a través de un kernel.

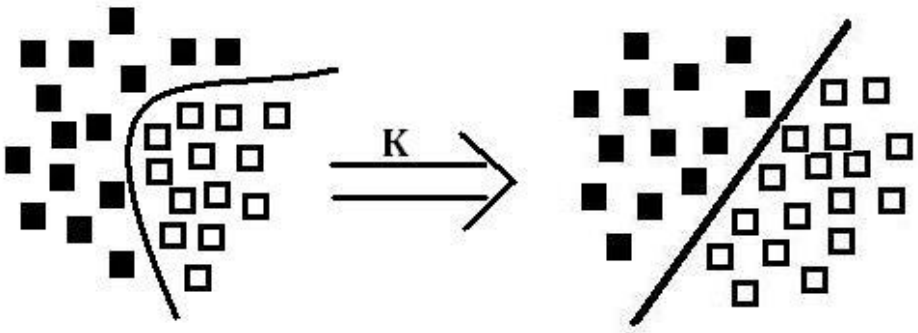

Figura 5: Clasificación de MSV para el caso no lineal

Una función kernel se puede definir como aquella que permite realizar la separación y el traslado de los datos al espacio de características. Existen diversos kernels predeterminados conocidos entre los cuales se destaca el lineal, el RBF (Función de Base Radial), el Polinomial, el Sigmoidal, entre otros más. Matemáticamente, un kernel se puede definir como una función $K: X \times X \rightarrow \Re$: tal que $K(y,x) = (\Phi(x), \Phi(y))$, donde Φ es una transformación de X en un cierto espacio de Hilbert $\Im$. La función kernel permite calcular el producto escalar $(\Phi(x), \Phi(y))$ en el espacio de características sin necesidad de usar ni conocer la transformación Φ (1).

Luego de conocer la idea de lo qué es una función kernel y qué se puede lograr con ella, se puede proceder a especificar el problema de optimización a resolver para las MSV con margen máximo en el espacio de características. El problema de programación cuadrática con restricciones a resolver es el siguiente:

$$\max \ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \, K(x_i, x_j)$$

Sujeto a: $\sum_{i=1}^{N} y_i \alpha_i = 0$

$$\alpha_i \geq 0, \ \ 1 \leq i \leq 0$$

Donde $K(x, y)$ es la función kernel.

1.3.5.2 Máquinas de Soporte Vectorial con margen blando

Las Máquinas de Soporte Vectorial con margen blando se refieren a aquellos casos en los cuales los datos no son linealmente separables, ni siquiera en el espacio de características. También es necesario este enfoque cuando los datos pueden ser clasificados linealmente en el espacio de características pero las soluciones son sobreajustadas a los ejemplos y con esto, se tiende a tener mala generalización. Este modelo es el que precisamente funciona bien con la inclusión de ruido pues resulta ser mucho más robusto que los anteriores modelos explicados.

Las Máquinas de Soporte Vectorial con margen blando incorporan variables de holgura al modelo permitiéndole a las máquinas seleccionar un clasificador con cierto margen de error pero que a su vez tienda a tener una mejor generalización. En otras palabras, estas variables, cuyo valor será siempre positivo o igual a cero, permitirán que con cierto margen no se cumplan las restricciones estrictamente.

Además, será necesario incluir tantas variables de holgura como datos se tengan, pues para cada instancia de datos se tendrá su respectiva variable de holgura.

La inclusión de variables de holgura a las restricciones en el modelo debe equilibrarse incluyendo en la función objetivo un término de regularización que depende de dichas variables. Este término es el que se conoce como el parámetro C, el cual determina la holgura del margen blando; es decir, el parámetro C es el que permite al clasificador un cierto margen error en el momento de clasificación. Con esto, el problema a optimizar resulta muy similar al de los modelos con margen máximo, con la diferencia de que en este modelo de margen blando se incluye un término en la función objetivo dependiente del parámetro C y las variables de holgura y, a su vez, a cada restricción se le añadirá las respectivas variables de holgura. Además, se incluirá una nueva restricción que es la que limita el valor del parámetro C.

1.3.6 Condiciones Karush-Kuhn-Tucker (KKT).

Para todos los algoritmos de optimización es importante decidir cuándo poner fin al proceso. Para decidir acerca de la optimización de la solución actual se deben verificar las llamadas condiciones Karush-Kuhn-Tucker (KKT).

Teorema 1 (Condiciones KKT)
Se tiene que f: $\mathbb{R}^m \to \mathbb{R}$ y c: $\mathbb{R}^m \to \mathbb{R}$ son funciones y $L(x, \alpha) = f(x) + \sum_{i=1}^{n} \alpha_i c_i(x), \alpha_i \geq 0$ la función LaGrange correspondiente. Si existe $(\bar{x}, \bar{\alpha})$ entonces

$$L(\bar{x}, \alpha) \leq L(\bar{x}, \bar{\alpha}) \leq L(x, \bar{\alpha})$$

Donde $\bar{x}$ es una solución al problema de optimización restringido:

$$\min f(x), s.t. c_i(x) \leq 0, \forall_i = 1, \dots, n.$$

La relación que figura en el teorema relativo a $L(\overline{x},\overline{\alpha})$ sólo establece que el lagrangiano L es $\min_{w,r,t}\ \overline{x}$ y $\max_{w,r,t}\ \overline{\alpha}$ en un punto de ensilladura. Para la función objetivo convexa y diferenciable f y las restricciones c_i, el anterior teorema puede ser reformulado.

Teorema 2 (condiciones KKT para problemas diferenciables convexos). : Se tiene
f: $\mathbb{R}^m \to \mathbb{R}$ y c: $\mathbb{R}^m \to \mathbb{R}$ son funciones diferenciables convexas. Entonces $\overline{x}$ es una solución al problema de optimización

$$min\ f(x), sujeto\ a\ .\ c_i\ (x) \leq 0, \forall_i\ =\ 1, \ldots., n.$$

si existe algún $\overline{\alpha} \geq 0$ de tal manera que se cumplan las siguientes condiciones:

$$\frac{\partial L(\overline{x},\overline{\alpha})}{\partial x} = \frac{\partial f(\overline{x})}{\partial x} + \sum_{i=1}^{n} \alpha_i \frac{\partial c_i(\overline{x})}{\partial x} = 0 \qquad \text{(punto de ensilladura en } \overline{x})$$

$$\frac{\partial L(\overline{x},\overline{\alpha})}{\partial \alpha} = c_i(\overline{x}) \leq 0 \qquad \text{(punto de ensilladura en } \overline{\alpha})$$

$$\sum_{i=1}^{n} \overline{\alpha}_i c_i(\overline{x}) = 0 \qquad \text{(KKT gap)}$$

1.3.7 Programación Cuadrática

La programación cuadrática (QP) es un tipo especial de problema para la optimización (maximizar o minimizar) de una función cuadrática de varias variables sujeto a restricciones lineales en esas variables. La importancia de la programación cuadrática radica no sólo en la variedad de problemas que encuentran en este modelo su marco apropiado, sino porque muchos de los algoritmos para la solución del programa general no lineal se basan en la solución de subproblemas cuadráticos. Lo que hace exactamente la programación cuadrática es tratar de optimizar una función objetivo cuadrática sujeta a restricciones lineales ya sea de igualdad o desigualdad. Por lo tanto un problema de programación no lineal, cuyas restricciones son lineales y cuya función objetivo es la suma de términos de la forma $X_1^{n_1}$, $X_2^{n_2}$,…,$X_n^{n_n}$ (en la cual cada término tiene un grado de 2, 1 o 0) es un problema de

programación cuadrática. Wolfe, Chunking, Osuna y Optimización de Secuencia Mínima son algunos de los algoritmos que resuelven este tipo de problema.

1.4 Secuential Minimal Optimization.

El entrenamiento de las máquinas de Soporte Vectorial requiere de una solución al enorme problema de optimización de la programación cuadrática. En este caso se eligió la Optimización de Secuencia Mínima (SMO), este clasificador resuelve los problemas de multiclasificación utilizando clasificación por pares, uno contra uno, con el cual se va realizando el modelo a partir de todas las posibles combinaciones de las diferentes clases. John C. Platt afirma que la SMO lo que hace es transformar este problema en una serie de problemas más pequeños que se resuelven de forma analítica mientras que los algoritmos anteriores de aprendizaje para las Máquinas de Soporte Vectorial utilizan programación cuadrática de forma numérica malgastando tiempo computacional (3). La cantidad de memoria requerida para SMO es lineal respecto al tamaño del conjunto de entrenamiento, lo cual le permite manejar conjuntos de datos muy grandes. Por la evasión que se hace a la computación matricial, SMO se clasifica entre lineal y cuadrática en el tamaño del conjunto de entrenamiento para diversos problemas de prueba, mientras que el algoritmo estándar para las MSV Chunking, lo hace entre lineal y cúbico en el tamaño del conjunto de entrenamiento por lo que SMO puede llegar a ser hasta 1000 veces más rápido que el algoritmo Chunking.

1.4.1 Parallel Secuential Minimal Optimization

A pesar de que SMO ha ganado popularidad en los últimos años para el entrenamiento de las MSV aún requiere de una gran cantidad de tiempo computacional para la solución de problemas de gran tamaño. Este trabajo propone una implementación paralela de SMO (PSMO) para el entrenamiento de las MSV el cual es desarrollado utilizando Message Passing Interface (MPI). Específicamente el PSMO primero divide el conjunto de datos de entrenamiento completo en subconjuntos más pequeños y luego los ejecuta simultáneamente en múltiples procesadores, cada uno de los datos particionados. Los experimentos han demostrados que ha habido una gran aceleración en conjuntos de datos mayores cuando se usan varios procesadores.

Actualmente existe una gran cantidad de trabajos de investigación que se han realizado en las Máquinas de Soporte Vectorial, debido principalmente a la generalización de su impresionante rendimiento en la solución de diversos problemas de aprendizaje de las máquinas. Dado un conjunto de puntos de datos $\{(X_i, y_i)\}_i^n$, $(X_i \in R^d$ es el vector de entrada de los i_{th} datos de entrenamiento; $y_i \in \{-1,1\}$ es la clase de etiqueta; n es el número total de datos de patrones de entrenamiento), entrenar una MSV en la clasificación es equivalente a la solución de un problema de programación cuadrática.

$$max: R(\alpha_i)=\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(X_i, X_j) \quad \text{(a)}$$

$$\text{Sujeto a: } \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \leq \overline{\alpha} \leq c, \qquad i = 1, \dots, n$$

Donde $k(X_i, X_j)$ es la función kernel. Una de las funciones kernel más ampliamente usada es la gaussiana $e^{\frac{-\|X_i - X_j\|^2}{\sigma^2}}$ donde σ^2es la amplitud del kernel gaussiano. α_i es el multiplicador de LaGrange a ser optimizado. Un α_i es asociado a cada modelo de datos de entrada. C la constante de regularización predeterminada por los usuarios. Después de resolver el problema de programación cuadrática (QP), la siguiente función de decisión es usada para determinar las clases para un nuevo modelo de datos.

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \, k(X_i, X) + b$$

Donde b es obtenida de la solución de (a)

Por lo tanto, el problema principal de las MSV se reduce a resolver el problema QP (a), donde el número de variables a ser optimizadas α_i es igual al número de modelo de datos de entrenamiento i. Para los problemas de pequeños tamaño, las técnicas QP estándar como el algoritmo del gradiente de proyección pueden aplicarse directamente. Sin embargo, para problemas de gran tamaño, las técnicas de QP estándar no son útiles porque requieren una gran cantidad de la memoria de la computadora para almacenar la matriz kernel K ya que el número de elementos de K es igual al cuadrado del número de modelos de datos de entrenamiento.

Para hacer más prácticas las MSV, se han desarrollado algoritmos especiales, tales como el Chunking de Vapnik, la descomposición de Osuna y el SVM^{light} de Joachims. Ellos hacen el entrenamiento de las MSV posible fragmentando el gran problema QP (a) en una serie de pequeños problemas QP y optimizando sólo un subconjunto de modelos de datos de entrenamiento en cada paso. Al subconjunto de modelos de datos de entrenamiento optimizado en cada paso, se le llama working set. Por lo que estos métodos se clasifican como métodos de *working set*.

Basado en la idea de los métodos de working set, Platt propuso el algoritmo de optimización de secuencia mínima (SMO) algoritmo que selecciona el tamaño del working set por pares y utiliza un simple enfoque analítico para resolver los pequeños problemas QP reducidos. Hay algunas heurísticas utilizadas para la elección de dos α_i para optimizar en cada paso. Como fue señalado por Platt, SMO equilibra solo cuadráticamente en el número de modelos de datos de entrenamiento, mientras que otros algoritmos lo hacen cúbicamente o más, en el número de modelos de datos de entrenamiento. Más tarde, Keerthi determina la ineficacia asociada al SMO de Platt y propone dos versiones modificadas del SMO que son mucho más eficientes que versiones originales de Platt. La segunda modificación es bien particular y se utilizan en los populares paquetes de las MSV como LIBSVM. Esta modificación se llamará algoritmo SMO modificado (4).

Actualmente existen unos cuantos trabajos desarrollando el entrenamiento de las MSV de forma paralela. Por ejemplo según R. Collobert, S. Bengio y Y. Bengio, una mezcla de MSVs es entrenada en paralelo usando subconjuntos de un conjunto de datos de entrenamiento (5). Los resultados de cada MSV se combinan mediante el entrenamiento de otra red neuronal. Este experimento ha demostrado que el algoritmo paralelo propuesto puede brindar mucha más eficiencia que usando una MSV simple. Otro algoritmo, éste propuesto por J. X. Dong, A. Krzyzak y C. Y. Suen, donde múltiples MSVs también son desarrolladas utilizando subconjuntos de un conjunto de datos de entrenamiento (6). Los vectores de soporte en cada una de las MSV se almacenarán para luego formar otra MSV nueva. Se ha visto mucha eficiencia en este algoritmo a través de los experimentos. Zanghirati y Zanni también propusieron una implementación paralela de SVMlight donde el problema de programación cuadrática se divide en subproblemas más pequeños (7). Los subproblemas son resueltos por un método de variable de proyección. Los resultados muestran que el enfoque es comparable con las máquinas de escalar una técnica ampliamente utilizada y puede alcanzar una buena eficiencia y escalabilidad en un sistema multiprocesador. B.H. Guang, K. Z. Mao, C.K. Siew y D.S. Huang, propusieron una implementación de red modular para las MSV (8). El resultado comprobó que la red modular puede

reducir significativamente el tiempo de aprendizaje de algoritmos de las MSV sin sacrificar mucho la generalización del rendimiento.

1.5 Programación Paralela

La programación paralela es una técnica de altas prestaciones en la que muchas instrucciones se ejecutan simultáneamente. Se puede aplicar bien sea en un único ordenador con varios elementos de procedimiento (multinúcleo, multiproceso) o en un conjunto de de ordenadores (Clúster, MPP). El objetivo principal es resolver grandes tareas dividiéndolas en pequeñas subtareas que pueden resolverse de forma concurrente para después combinarlas. Los programas de ordenador paralelos son más difíciles de escribir que los secuenciales porque la concurrencia introduce nuevos tipos de errores de software, siendo las condiciones de carrera o estado de carrera los más comunes. La comunicación y la sincronización entre las diferentes subtareas son típicamente las grandes barreras para conseguir un buen rendimiento de los programas paralelos. Para contrarrestar estos problemas afortunadamente existe una infraestructura de software para trabajar con múltiples procesadores: La interfaz de pases de mensaje (MPI), que no es más que una forma de comunicar procesos a través de diferentes estructuras lógicas.

1.5.1 Interfaz de Paso de Mensaje[1]

Interfaz de Paso de Mensajes ("Message Passing Interface") es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua, de manera similar a como se hace con los semáforos, monitores, entre otros.

Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación para sistemas paralelos. Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje.

Dependiendo de si el proceso que envía el mensaje espera a que sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono. En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar uno nuevo y enviarlo antes de que se haya recibido el anterior. Por este motivo se suelen emplear buzones, en los que se almacenan los mensajes a espera de que un proceso los reciba. Generalmente empleando este sistema, el proceso que envía mensajes solo se bloquea o para, cuando finaliza su ejecución, o si el buzón está lleno. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución. Por esto se suele llamar a esta técnica encuentro, o rendezvous. Dentro del paso de mensajes síncrono se engloba a la llamada a procedimiento remoto, muy popular en las arquitecturas cliente/servidor.

Ventajas

La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables (dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida), y rápidos, (porque cada implementación de la librería ha sido optimizada para el hardware en la cual se ejecuta).

Desventajas

El acceso remoto a memoria es lento

La programación puede ser complicada

[1]Mundialmente se conoce por sus siglas en inglés como MPI.

1.6 Computación Distribuida

Los sistemas distribuidos están definidos como una serie de computadores, separados físicamente y conectados entre sí por una red de comunicaciones distribuida, donde cada máquina posee sus propios elementos de hardware y software. Teniendo en cuenta que las computadoras casi nunca

utilizan casi toda su capacidad, la computación distribuida se encarga de aprovechar esta capacidad no utilizada por estos ordenadores, a través de una serie de programas que dividen cierto problema que necesite resolver un usuario en varias piezas que son enviadas por la red, siendo procesadas por ordenadores personales. Una vez concluido el proceso son enviadas de vuelta al ordenador del usuario.

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora, es capaz de reducir el tiempo de procesamiento a gran escala pues lo que se puede demorar horas en una PC independiente, un sistema distribuido lo ejecuta solo en minutos.

1.7 Conclusiones

Hasta aquí se han puntualizado algunos de los conceptos principales que permitirán una mejor comprensión de este trabajo. Como se ha hecho evidente en varios estudios las Máquinas de Soporte Vectorial no son capaces de manejar enormes cantidades de datos trayendo consigo la necesidad de búsqueda de nuevos métodos capaces de manipularlos. Partiendo de que el mayor inconveniente de las Máquinas de Soporte Vectorial es el problema de programación cuadrática se hizo una investigación acerca de algoritmos que pudieran darle solución. La mejor opción la brindó el algoritmo SMO en su versión paralela PSMO y las condiciones KKT para establecer la condición de parada. En este trabajo se buscó siempre la manera de explotar al máximo las MSV por lo que se hace uso de la computación paralela y distribuida como principal herramienta.

Capítulo 2: Programas y metodologías

2.1 Introducción

Para llevar a cabo esta investigación se decidió hacer uso de aplicaciones y tecnologías que servirán de herramienta para la solución del problema científico. Se investigaron varios métodos existentes desarrolladas por investigadores del mundo de las Máquinas de Soporte Vectorial y se proponen otros nuevos, lo que constituye una pieza fundamental en esta investigación. Asimismo se exponen las metodologías empleadas las cuales se irán explicando para alcanzar una mejor comprensión.

2.2 Weka 3.6

El Entorno para Análisis del Conocimiento de la Universidad de Waikato (Weka por sus siglas en inglés) es un conocido software para aprendizaje automático y minería de datos escrito en Java y desarrollado en dicha institución neozelandesa. WEKA es un software libre distribuido bajo licencia GNU-GPL estimulando así el constante desarrollo e intercambio de prestaciones. Este software se ha convertido en una potente herramienta a nivel mundial para la visualización, incluye algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades. Específicamente Weka 3.6 brinda un gran aporte a esta investigación al incluir LIBLINEAR como un nuevo servicio y mejorando potencialmente la LIBSVM, alcanzando excelentes resultados. Fue desarrollado en del 2007 por lo que cuenta con las más recientes innovaciones para el graficado y visualización de las muestras.

2.3 LIBLINEAR

LIBLINEAR es una biblioteca de código abierto de clasificación lineal a gran escala. Actualmente soporta L2-regularized logistic regression, L2-loss support vector machines, y L1-loss support vector machines. Una de sus principales características es que mantiene el mismo formato de datos de la LIBSVM. Es la solución a gran escala de los problemas de clasificación, fundamental en muchas aplicaciones, como clasificación de textos. La clasificación lineal se ha convertido en una de las más

prometedoras técnicas de aprendizaje para grandes cantidades datos con un gran número de instancias y características. Es válido señalar que aunque está investigación no se utilice también soporta clasificación multiclase. Los experimentos han demostrado que LIBLINEAR es muy eficiente con grandes conjuntos de datos. Por ejemplo, se tarda sólo unos segundos para capacitar a un problema de clasificación de texto de la agencia Reuters Corpus Volumen 1 (rcv1) que tiene como cifra más de 600.000 ejemplos. Por la misma tarea, un solucionador de MSV como LIBSVM puede tomarse varias horas. Además, LIBLINEAR es tan competitivo incluso más rápido que clasificadores lineales potentes como Pegasos de Shalev-Shwartz del 2007 y el SVMperf de Joachims del 2006. Sin utilizar los kernels, se puede capacitar a un conjunto mucho más amplio a través de un clasificador lineal. La validación cruzada reduce significativamente el tiempo mediante el uso de LIBLINEAR.

LIBLINEAR soporta los dos clasificadores binarios lineales más populares: LR y linear SVM. Determinando una serie de instancias pares $(x_i, y_i), i = 1, \dots, l, x_i \in \mathbb{R}^n, y_i \in \{-1,1\}$, ambos métodos resuelven el siguiente problema de optimización con diferentes funciones loss $\xi(w; x_i, y_i)$:

$$min_w \ \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi(w; x_i, y_i),$$

Donde C > o es un parámetro de penalidad. Para las MSV las funciones loss más comunes son $\max(1 - y_i w^T x_i, 0)$, y $\max(1 - y_i w^T x_i, 0)^2$. El primero es representado como L1-SVM mientras que el otro como L2-SVM. Para LR, la función loss es $\log(1 + e^{-y_i w^T x_i})$, la cual es derivada de un modelo probabilístico. En algunos casos, la función discriminante del clasificador incluye un término bias, b. LIBLINEAR maneja este término aumentando el vector w y cada instancia x_i con una dimensión adicional: $w^T \leftarrow [w^T, b], x_i^T \leftarrow [x_i^T, B]$, donde B es una constante especificada por el usuario. El enfoque para L1-SVM y L2-SVM, es un método de coordenada descendiente (9). Para LR y también L2-SVM LIBLINEAR implementa un método de región de confianza de Newton (10).

Los parámetros que se tienen en cuenta para el cálculo de estas predicciones a los cuales se enfocó esta investigación son:

Uso: train [options] training_set_file [model_file]

-s tipo: Modifica el tipo de solucionador (1: por defecto)

0 -- L2-regularized logistic regression

1 -- L2-loss support vector machines (dual)

2 -- L2-loss support vector machines (primal)

3 -- L1-loss support vector machines (dual)

4 – multi-clases support vector machines

-c costo: Modifica el parámetro C (1: por default)

-e épsilon: Modifica la tolerancia en el criterio de parada

-s 0 and 2

|f'(w) |_2 <= eps*min (pos, neg)/l*|f'(w0)|_2, donde f es la función primal (0.01: por defecto)

-s 1, 3, and 4

Máxima violación dual <= eps; similar a LIBSVM (0.1: por defecto)

-B bias: si bias >= 0, la instancia x se convierte en [x; bias]; si bias < 0, el término bias no es añadido (1: por defecto)

-wi peso: los pesos ajustan el parámetro C de las diferentes clases

-v n: n-segmento del modo de validación cruzada.

2.4 LIBSVM

LIBSVM es un software integrado para los vectores de soporte de clasificación, (C-SVC, nu-SVC), de regresión (épsilon-SVR, nu-SVR) y de estimación de la distribución (una clase MSV). Aunque no es tratada en este trabajo se debe señalar que también soporta clasificación multiclase. Para problemas de tamaño mediano la validación cruzada podría ser el modo más fiable para parámetros de selección. En primer lugar, el entrenamiento de datos es separado en varios segmentos. Secuencialmente un segmento está considerado como el conjunto de validación y el resto son para el entrenamiento. El promedio de exactitud en la predicción de los conjuntos validados es la exactitud de la validación cruzada.

LIBSVM proporciona una interfaz simple para clasificación y regresión, donde los usuarios se pueden conectar fácilmente con sus propios programas. Además de esto incluye importantes características que proporciona un mejor rendimiento. Entre las más notables se destaca el modelo de selección automática que puede generar con gran precisión el contorno de la validación cruzada, igualmente se puede encontrar su código fuente en java y en C++. Además LIBSVM brinda estimaciones de la probabilidad así como herramientas de ponderación para datos desbalanceados. Existe también la

posibilidad para los usuarios de pre compilar los valores del kernel e introducirlos como archivos de entrenamiento y prueba. Entonces LIBSVM no necesita obligatoriamente usar los conjuntos de entrenamiento/prueba originales.

El formato para los ficheros de entrenamiento y prueba es el siguiente:
<etiqueta> <indice1>:<valor1> <indice2>:<valor2>. . .
Cada línea contiene una instancia y es terminada por los caracteres `\n´. Para clasificación, "<etiqueta>" es un número entero que indica la etiqueta de la clase que determina si el fragmento es activo o inactivo. En regresión, <etiqueta> es un valor único que define al fragmento y está asociado a su actividad biológica, el cual puede ser cualquier número. Para la clase única de MSV aunque no es utilizada en esta investigación puede ser cualquier número real. El formato <índice>:<valor> muestra un valor de un campo (atributo), definido a partir de: el <índice>, que es un número entero a partir de 1 ordenado ascendentemente y el <valor> es un número real. Si estos son valores desconocidos, se completa la primera columna con números aleatorios.

Los parámetros que se tienen en cuenta para el cálculo de estas predicciones a los cuales se enfocó esta investigación son:

Uso: svm-train [options] training_set_file [model_file]

-s tipo de MSV: selecciona el tipo de MSV (0: por defecto)
- 0 -- C-SVC
- 1 -- nu-SVC
- 2 -- one-class SVM
- 3 -- épsilon-SVR
- 4 -- nu-SVR

-t tipo de kernel: selecciona la función kernel (2: por defecto)
- 0 -- Lineal: K(u, v) = u' * v.
- 1 --Polinomial: K(u, v) = (gamma * u' * v + coef0)^degree.
- 2 -- Función de Base Radial (RBF): K(u, v) = exp(-gamma * |u – v|^2).
- 3 -- Sigmoidal: K(u, v) = tanh (gamma * u * v + coef0).
- 4 – kernel pre compilado (valores kernel en el archivo de entrenamiento)

Los parámetros que se tienen cuenta para el cálculo de estas predicciones a los cuales se enfocó esta investigación son:

-d degree: Para modificar el parámetro del grado de la función polinómica (3: por defecto).
-g gamma: Para modificar este parámetro en la función kernel (1/k: por defecto, donde -k es el número de atributos en los datos de entrada).
-r coef0: Para modificar este parámetro en la función kernel (0: por defecto).
-c costo: Parámetro de penalización (por defecto: 100).
-n nu: Modifica el parámetro nu de nu-SVC, clase única y nu-SVR (0.5: por defecto)
-m tamaño del caché: Para modificar el tamaño de memoria del caché (100: por defecto).
-p épsilon: Modifica el valor de épsilon en la función loss de épsilon-SVR (0.1: por defecto).
-e épsilon: Modifica la tolerancia en el criterio de parada (0.00.1: por defecto).
-h heurística: Si se utilizan acortamientos heurísticos, 0 o 1 (1: por defecto).
-b estimación de probabilidades: estimar probabilidades tanto para modelos de entrenamiento de SVC como de SVR, 0 o 1 (1: por defecto).
-wi peso: Modifica el parámetro C de la clase i multiplicando peso*C, para C-SVC
(1: por defecto)
-v n: n-segmento del modo de validación cruzada.
Para la predicción solamente se necesita el parámetro b soportando solo el valor 0 para una sola clase. Se le debe especificar el archivo generado por el entrenamiento, el conjunto de datos que se quiere predecir y el fichero de salida que es el resultado final de la predicción.

A menudo las personas que nunca se han enfrentado a las MSV se pueden encontrar con problemas que tienen como resultado con una pobre clasificación o regresión. Algunos de estos problemas vienen dados por el desconocimiento o falta de información que puedan tener los usuarios acerca de de los conjuntos de datos tratados. Si bien no brindan una información completa los scripts grid.py y easy.py son de gran utilidad para quien se enfrenta a estas librerías. Estos segmentos de código hechos en lenguaje python vienen incluidos por defecto en ambas librerías a partir de las versiones más recientes los cuales proporcionan un mejor manejo a la hora de predecir. grid.py es una manera automática de encontrar los mejores parámetros costo (c) y gamma (g) que son claves en el entrenamiento de las MSV. Una correcta selección de c es muy importante y más cuando el conjunto datos de entrenamiento esta desbalanceado, esto quiere decir que existen mucho más muestras negativas que positivas o viceversa. Una vez terminada su ejecución se podrán obtener los valores de c y g así como

la exactitud de la validación cruzada del conjunto de datos de entrenamiento utilizado. Si el usuarios es inexperto en las MSV y el conjunto de datos no es muy grande es necesario el uso de easy.py. Este script es completamente automático y además de las funciones que realiza grid.py, escala los datos de entrenamiento, realiza la validación cruzada, hace el entrenamiento, escala los datos de prueba y hace la predicción. Las principales desventaja de esta herramienta es que para llevar a cabo este proceso puede tardar varias horas y requerir de mucha memoria en dependencia del tamaño del conjunto de datos.

Otra vía de obtener un indicador de los resultados es a través del script subset.py el cual hace una selección aleatoria del conjunto de datos a entrenar y lo almacena en un fichero creado previamente, luego este fichero se puede tratar como el conjunto de datos original brindando una guía para la búsqueda de los mejores parámetros.

2.5 πSVM 1.1

πSvM 1.1 es una implementación paralela de las Máquinas de Soporte Vectorial. Este software desarrollado en diciembre del año 2008 soporta igualmente C-SVC, nu-SVC, épsilon-SVR y nu-SVR, su interfaz de línea de comandos es semejante a la de LIBSVM pues solo incorpora los parámetros del tamaño máximo de working set, el número máximo de nuevas variables en el working set y el número de procesadores. Es capaz de funcionar en cualquier clúster de computadoras donde esté disponible MPI. Por razones técnicas esta investigación solamente hizo uso de un clúster de 8 procesadores donde la velocidad y la aceleración fueron protagonistas.

Uso : mpirun -np 1 ./pisvm-train Usage: svm-train [options] training_set_file [model_file]
-s tipo de MSV: selecciona el tipo de MSV (0: por defecto)

0 -- C-SVC

1 -- nu-SVC

2 -- one-class SVM

3 -- épsilon-SVR

4 -- nu-SVR

-t tipo de kernel: selecciona la función kernel (2: por defecto)

0 -- Lineal: K(u, v) = u' * v.

1 --Polinomial: K(u, v) = (gamma * u' * v + coef0)^degree.

2 -- Función de Base Radial (FBR): K(u, v) = exp(-gamma * |u – v|^2).

3 -- Sigmoidal: K (u, v) = tanh (gamma * u * v + coef0).

4 – kernel pre compilado (valores kernel en el archivo de entrenamiento)

-d degree: Para modificar el parámetro del grado de la función polinómica (3: por defecto).

-g gamma: Para modificar este parámetro en la función kernel (1/k: por defecto, donde -k es el número de atributos en los datos de entrada).

-r coef0: Para modificar este parámetro en la función kernel (0: por defecto).

-c costo: Parámetro de penalización (por defecto: 100).

-n nu: Modifica el parámetro nu de nu-SVC, clase única y nu-SVR (0.5: por defecto)

-m tamaño del caché: Para modificar el tamaño de memoria del caché (100: por defecto).

-p épsilon: Modifica el valor de épsilon en la función loss de épsilon-SVR (0.1: por defecto).

-e épsilon: Modifica la tolerancia en el criterio de parada (0.00.1: por defecto).

-h heurística: Si se utilizan acortamientos heurísticos, 0 o 1 (1: por defecto).

-b estimación de probabilidades: estimar probabilidades tanto para modelos de entrenamiento de SVC como de SVR, 0 o 1 (1: por defecto).

-wi peso: Modifica el parámetro C de la clase i multiplicando peso*C, para C-SVC

(1: por defecto)

-o: Tamaño máximo de working set

-q : Número máximo de nuevas variables en el working set

2.6 Plataforma de Tareas Distribuidas (T-arenal)

La plataforma de cálculos distribuidos brindó a esta investigación otra alternativa más para el tratamiento masivo de datos puesto que no se reportaron técnicas que utilizaran este tipo de ambiente. Está representada por una arquitectura Cliente-Servidor, donde las computadoras que se comunican durante el cómputo para procesar los datos representan a los clientes, y el computador donde está montada la plataforma constituye el servidor, el que cual se encarga de monitorear todo el proceso. La esencia de esta herramienta es funcionar como una supercomputadora virtual capaz de aunar y coordinar los esfuerzos entre los recursos disponibles y utilizar de esta forma, el poder computacional que ofrecen computadores personales generalmente de bajas prestaciones pero interconectados mediante una red local lo suficientemente rápida para entre todos lograr obtener resultados. Está dividido en tres componentes esenciales: servidor, cliente e interfaz de administración. El módulo servidor almacena el problema (datos del problema y el algoritmo que los procesará) y divide estos en

pequeños sub-problemas, llamados unidades de trabajo. El principal aspecto del servidor es repartir las diferentes unidades de trabajos entre los clientes (computadoras personales, clúster de computadoras, entre otros medios de cómputo), siempre y cuando estos estén autorizados a interactuar con el mismo; además de desarrollar una lógica de control de unidades pendientes de respuestas y otras que han expirado por algún error ocurrido.

El módulo cliente está instalado en cada una de las máquinas y conectadas al servidor mediante una red LAN. El cliente realiza una petición de una unidad de trabajo al servidor, hace el procesamiento de la misma y posteriormente retorna el resultado al servidor, y vuelve a pedir otra unidad de trabajo. Múltiples clientes pueden realizar peticiones de trabajo simultáneamente al servidor. El servidor colecciona el resultado de cada uno de los sub-problemas para finalmente construir el resultado del problema original.

Para poder definir un cálculo distribuido sobre esta plataforma solo se requiere heredar de dos clases de Java, la clase DataManager y la clase Task, el desarrollador debe redefinir diferentes métodos de estas clases para lograr establecer su aplicación distribuida.

2.7 Eclipse

Se utilizó como entorno de desarrollo Eclipse, debido a que este posee una serie de ventajas y aplicaciones que lo hacen el entorno de desarrollo ideal para implementar la solución propuesta, a continuación se exponen una serie de estas características:

- ✓ Integra diferentes aplicaciones.
- ✓ Utiliza JAVA como lenguaje de programación, también permite a través de la incorporación de plug-ins, programar para otros lenguajes.
- ✓ Soporta la programación orientada a objetos, la depuración y compilación de código.
- ✓ Con su desarrollo la implementación de aplicaciones resulta más sencilla que en entornos para Java anteriores.
- ✓ Es una herramienta de código abierto, corre en una gran cantidad de sistemas operativos.

- ✓ Es soportado por múltiples arquitecturas y posee capacidad de control de versiones con subversión.

2.8 JAVA como lenguaje de programación

Para realizar la distribución de las máquinas de soporte vectorial utilizando la Plataforma de Cálculo Distribuido T-arenal, se empleó el lenguaje de programación Java. Este lenguaje orientado a objetos, de plataforma independiente, fue desarrollado por la compañía Sun Microsystems con la idea original de usarlo para la creación de páginas web. Es un lenguaje de programación muy extendido en la actualidad y que cada día cobra más auge dentro del mundo de la programación por sus potencialidades y facilidad de aprendizaje. Dentro de algunas de sus características más relevantes se encuentran:

- ❖ Simple

Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje. Reduce en un 50% los errores más comunes de programación en lenguajes como C y C++, al eliminar muchas de las características de éstos.

- ❖ Orientado a Objetos

Implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

- ❖ Robusto

Realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores en el ciclo de desarrollo. Obliga

a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de esta.

- Seguro

El código Java pasa muchas pruebas antes de ejecutarse en una máquina. Este se pasa a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal. Si los byte-codes pasan la verificación sin generar ningún mensaje de error, entonces se conoce que: El código no produce desbordamiento de operandos en la pila. El tipo de los parámetros de todos los códigos de operación son conocidos y correctos.

- Distribuido

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como HTTP y FTP. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales. La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan serlos.

- Multihilo (Multithreaded)

Permite muchas actividades simultáneas en un programa. Los hilos de ejecución (threads, a veces llamados procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. El beneficio de ser multihilo consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real.

2.9 Metodología

Se diseñaron cuatro diferentes metodologías capaces de dar solución al problema de grandes volúmenes de datos. Desde el punto de vista local y distribuido se desarrollaron varias pruebas y experimentos para medir el rendimiento de las MSVs en conjuntos de datos a gran escala. También se procedió a realizar una comparación entre todas estas metodologías y las disímiles variantes que se pueden encontrar como la selección del kernel, la selección de parámetros y el uso adecuado o no de un determinado conjunto de datos.

2.9.1 Weka y LIBLINEAR

Se manejó la plataforma Weka con LIBLINEAR incorporado, en la que se utilizó el conjunto de datos leucemia.arff que archiva 38 instancias de entrenamiento, 34 de prueba y 7130 características para ambos ficheros mediante su interfaz gráfica. Los demás conjuntos de datos se procesaron utilizando la interfaz de líneas de comando (CLI) que trae Weka dentro de sus opciones. Se comparó el resultado arrojado por las librerías LIBLINEAR y LIBSVM que contiene esta plataforma confirmando así las potencialidades de esta nueva variante.

2.9.2 LIBSVM como herramienta independiente

En el último paquete de LIBSVM se ponen de manifiesto los avances obtenidos en comparación con las versiones anteriores desde el punto de vista rendimiento y velocidad, metas fundamentales en las MSV. Para su manejo se utilizaron los conjuntos de datos covtype.binary, rcv1_test.binary, mnist-576-rbf-8vr, mnist-784-poly-8vr, leukemia y covtype-2vr. En las pruebas realizadas esta metodología sirvió de base para trazar un patrón comparativo con respecto a las demás metodologías.

2.9.3 Máquinas de Soporte Vectorial paralelas

Los resultados alcanzados para el C-SVC y v-SVC, se basan en la aplicación πSVM. Se utilizaron 3 conjuntos de datos para la clasificación mnist-576-rbf-8vr y mnist-784-poly-8vr y covtype.binary. Todas las pruebas de rendimiento se ejecutaron en la máquina paralela Átropos, la cual constituye una red de PCs que funcionan bajo un ambiente GNU/Linux Debian Lenny 5.0 y consta de 8 nodos biprocesadores Intel(R) Core(TM)2 Duo CPU E4500 2.20GHz, interconectados mediante una red Fast-Ethernet con una topología Beowulf o de anillo. Cada nodo consta de 1 Gigabyte de memoria RAM y están disponibles para el cálculo científico.

2.9.4 Máquinas de Soporte Vectorial distribuidas

La plataforma de tareas distribuidas T-arenal se ejecuta de la siguiente forma:
En primer lugar el servidor almacena el problema (datos y el algoritmo que los procesará), y lo divide en pequeños subproblemas llamados unidades de trabajo. El principal aspecto del servidor es repartir las diferentes unidades de trabajos entre los clientes (computadoras personales, entre otros medios de cómputo), el módulo cliente está instalado en cada una de las máquinas y conectadas al servidor mediante una red LAN. El o los clientes realizan una petición de una unidad de trabajo al servidor, hace el procesamiento de la misma y posteriormente retorna el resultado al servidor, y vuelve a pedir otra unidad de trabajo. El servidor colecciona el resultado de cada uno de los subproblemas para finalmente construir el resultado del problema original y poder empaquetarlos para permitir su posterior descarga desde la interfaz de usuario. Las MSV distribuidas tienen esta misma filosofía de trabajo, primeramente se escala los datos con el objetivo de evitar dificultades numéricas durante el entrenamiento y principalmente balancear la muestra, además se seleccionan los parámetros adecuados para obtener buenos resultados. Se fragmenta la muestra en el servidor, creándose así las unidades de trabajo que son enviadas a los clientes a medida que estos se reporten, en estos clientes se lleva a cabo la ejecución del entrenamiento con el fragmento seleccionado y terminando esta acción van enviando al servidor los resultados del entrenamiento. A medida que estos llegan de los clientes al servidor, se realiza la predicción y al finalizar con todas estas se halla la media total. Finalmente estos resultados serán devueltos a la interfaz de la Plataforma de Tareas Distribuidas, para su uso posterior.

Capítulo 3: Resultados y discusión

3.1 Introducción

En el presente capítulo se muestra los resultados experimentales obtenidos después de realizar comparaciones entre LIBSVM y los diferentes ambiente y herramientas mencionados en el Capítulo 2, como es el caso de LIBLINEAR en una sola computadora personal, en un ambiente distribuido como es el uso de las máquinas de soporte vectorial distribuidas, utilizando la plataforma T-arenal y en un ambiente paralelo empleando πSVM, teniendo en cuenta esencialmente el tiempo de ejecución del entrenamiento y el porcentaje de error resultante al realizar o no, la validación cruzada.

3.2 Resultados

3.2.1 Explicación sobre como se debe realizar el entrenamiento de las Máquinas de Soporte Vectorial.

Las MSV son una herramienta muy útil para la técnica de clasificación. Incluso algunas personas piensan que es más fácil usarlas que las Redes Neuronales, sin embargo los usuarios que no están familiarizados con esta técnica, normalmente no obtienen resultados satisfactorios al inicio. Por esta razón se realizaron comparaciones entre el procedimiento que se recomienda a todos los usuarios principiantes y avanzados y la manera en que los usuarios principiantes generalmente realizan la clasificación. En la tabla 1 se representan algunos ejemplos del mundo real. Estos datos fueron reportados por algunos usuarios que no podían obtener una exactitud razonable al principio y usando el procedimiento que se ilustra a continuación lograron obtener un mejor rendimiento.

Muchos usuarios a menudo realizan los siguientes pasos:

* Transformar los datos al formato del software SVM.
* Tratar al azar unos pocos kernel y los parámetros.
* Prueba.

Se recomienda a todos los usuarios intentar primero el procedimiento siguiente:

*Transformar los datos al formato del software SVM.

*Escalar de los datos.

*Considerar el kernel RBF.

*Utilizar la validación cruzada para encontrar el mejor parámetro C y γ.

*Utilizar el mejor parámetro C y γ para entrenar a todo el conjunto de entrenamiento.

*Prueba.

Problemas	Nº datos de entrenamiento	Nº datos de prueba	Nº de características	Exactitud de los usuarios	Exactitud del procedimiento
Astro-partículas	3089	4000	4	75,2%	96,9%
Bioinformática	391	N/D	20	36%	85,2%
Vehículo	1243	41	21	4,88%	87,8%

Tabla 1: Resultados de los usuarios inexpertos contra los del procedimiento

A continuación se compara la exactitud que obtienen aquellos usuarios principiantes con el procedimiento recomendado, para cada uno de los problemas representados en la Tabla 1.Primeramente se lista la exactitud (Accuracy) realizando el entrenamiento y prueba de manera directa, en segundo lugar se evidencia la diferencia de exactitud escalando sin y con selección de parámetros. En tercer lugar se expone la exactitud del procedimiento (escalar con selección de parámetros). Finalmente se demuestra el uso de una de las herramientas de LIBSVM, llamada grid.py, que realiza todo el procedimiento de manera eficiente y automática.

Conjunto de datos original con los parámetros por defecto.

```
libsvm-2.89$ ./svm-train train.1
libsvm-2.89$ ./svm-predict test.1 train.1.model output.txt
Accuracy = 66.925% (2677/4000) (classification)
```

Escalada de los datos y entrenamiento con los parámetros por defecto.

```
libsvm-2.89$ ./svm-scale -l -1 -u 1 -s range1 train.1 > train.1.scale
libsvm-2.89$ ./svm-scale -r range1 test.1 > test.1.scale
libsvm-2.89$ ./svm-train train.1.scale
libsvm-2.89$ ./svm-predict test.1.scale train.1.scale.model salida.txt
Accuracy = 96.15% (3846/4000) (classification)
```

Selección de parámetros c y g con grid.py en el conjunto de datos.

```
libsvm-2.89/tools$ python grid.py train.1.scale
...
2.0 2.0 96.9893
libsvm-2.89$ ./svm-train -c 2 -g 2 train.1.scale
libsvm-2.89$ ./svm-predict test.1.scale train.1.scale.model output.txt
Accuracy = 96.875% (3875/4000) (classification)
```

Escalada, selección de parámetros y entrenamiento con easy.py en el conjunto de datos.

```
libsvm-2.89/tools$ python easy.py train.1 test.1
Scaling training data...
Cross validation...
Best c=2.0, g=2.0 CV rate=96.9893
Training...
Scaling testing data...
Testing...
Accuracy = 96.875% (3875/4000) (classification)
```

3.2.2 Primer resultado

Se realizaran estudios y comparaciones de los resultados de varias ejecuciones de las MSV en un procesador para diferentes tipos de datos. Para la comparación se tuvo en cuenta principalmente el tiempo de entrenamiento de cada ejecución y el porcentaje de validación cruzada.

Seguidamente se expresan en la tabla 2 los mejores parámetros *c* y *g* para el entrenamiento de los diferentes conjuntos de datos.

Conjuntos de datos	Mejor C	Mejor γ y d
mnist-576-rbf-8vr	C = 10.0	γ = 1.667
mnist-784-poly-8vr	C = 10.0	d = 7
covtype-2vr	C = 10.0	γ = 2e – 5
leukemia.arff	C = 8.0	N/D
leu	C = 8.0	γ = 0.000030518
covtype.libsvm.binary.scale	C = 4.0	γ = 2e – 5

Tabla 2: Mejores parámetros de cada conjunto de datos.

Los mejores parámetros pueden ser afectados por el tamaño de los datos, pero en la práctica, aquellos obtenidos a través de la validación cruzada son adecuados para todo el conjunto de entrenamiento.

Estudios realizados por diversas personalidades en el mundo de las Máquinas de Soporte Vectorial, han demostrado que para algunos tipos de datos de gran tamaño, utilizar kernels no lineales, no mejora el rendimiento, por lo que se puede utilizar el kernel lineal, es decir usando el clasificador lineal se pueden entrenar una mayor cantidad de datos. Para explicar este planteamiento se presentan tres situaciones diferentes:

3.2.1.1 Cuando el número de instancias << número de características

Muchos microarray en la Bioinformática presentan este tipo de situación. Se considera la leucemia (leu y leu.t). Los datos de entrenamiento y prueba cuentan con 38 y 34 instancias respectivamente. El número de características es de 7130, mucho más grande que el número de instancias. Se concatenaron los dos archivos y se compararon la exactitud de la validación cruzada usando los kernels RBF y lineal.

*Kernel RBF con selección de parámetros

```
libsvm-2.89/tools$ cat leu leu.t > leu.combined
libsvm-2.89/tools$ python grid.py leu.combined
[local] 5 -7 65.2778  (best c=32.0, g=0.0078125, rate=65.2778)
[local] -1 -7 65.2778  (best c=0.5, g=0.0078125, rate=65.2778)
[local] 5 -1 65.2778  (best c=0.5, g=0.0078125, rate=65.2778)
***
[local] 13 3 65.2778  (best c=8.0, g=3.0517578125e-05, rate=97.2222)
[local] 13 -9 65.2778  (best c=8.0, g=3.0517578125e-05, rate=97.2222)
[local] 13 -3 65.2778  (best c=8.0, g=3.0517578125e-05, rate=97.2222)
8.0 3.0517578125e-05 97.2222
```

(Mejor c =8,0, mejor gamma = 0.000030518 con un porcentaje de validación cruzada = 97.2222%).

*Kernel Lineal con selección de parámetros

```
libsvm-2.89/tools$ python grid.py -log2c -1,2,1 -log2g 1,1,1 -t 0 leu.combined
[local] 1.0 1.0 98.6111  (best c=2.0, g=2.0, rate=98.6111)
[local] 0.0 1.0 98.6111  (best c=1.0, g=2.0, rate=98.6111)
[local] 2.0 1.0 98.6111  (best c=1.0, g=2.0, rate=98.6111)
[local] -1.0 1.0 98.6111  (best c=0.5, g=2.0, rate=98.6111)
0.5 2.0 98.6111
```

(Mejor c = 0,5 con un porcentaje de validación cruzada = 98.6111%)

Aunque el grid.py fue diseñado para el kernel RBF, la forma anterior busca diferentes *C* usando el kernel lineal (-log2g 1, 1,1, hace que el valor de gamma sea estático).

El índice de la validación cruzada usando el kernel lineal es comparable al del RBF. Lo que da a entender que, cuando se tiene un número de características muy grande y un número pequeño de instancias, no hay necesidad de mapear los datos.

En adición a la LIBSVM, la LIBLINEAR una biblioteca que se señala a continuación es también efectiva para datos de este tipo. Las imágenes siguientes muestran la librería LIBLINEAR en la plataforma de aprendizaje Weka para la cual se utilizó el conjunto de datos leukemia.arff con C = 8, el cual arroja un 97,0588% de exactitud para las instancias clasificadas correctamente.

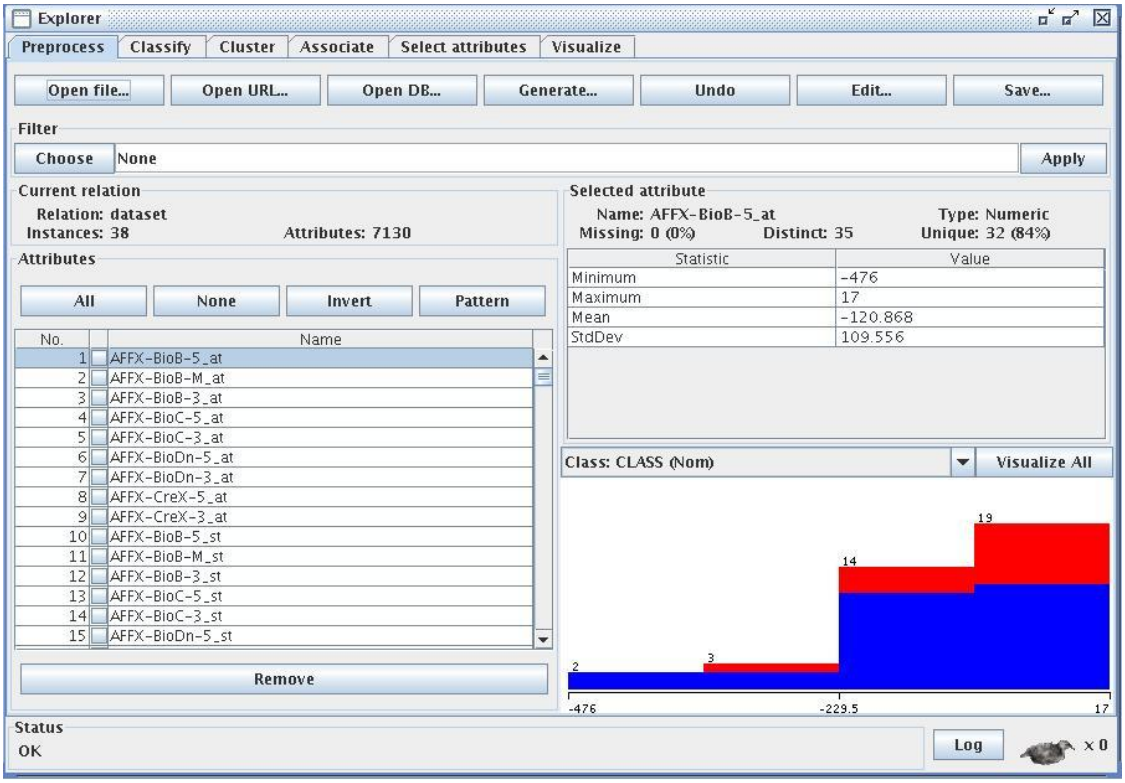

Figura 6: Interfaz de Weka una vez cargado el conjunto de datos leukemia.arff.

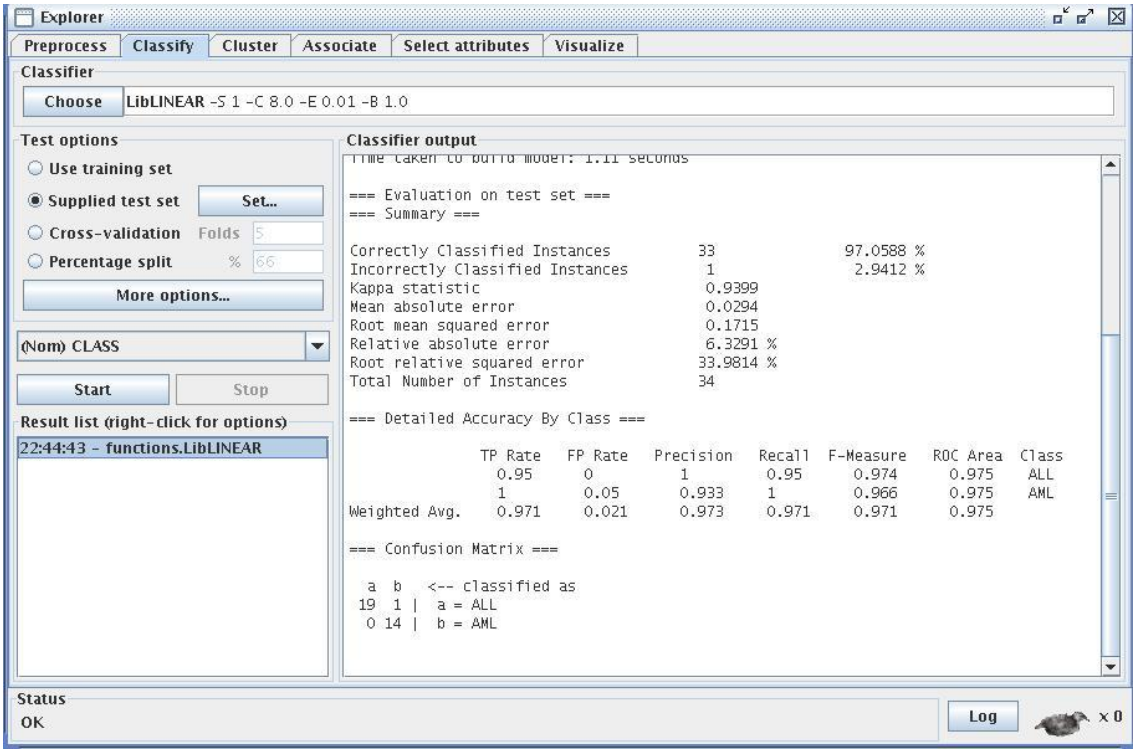

Figura 7: Resultados de la predicción del conjunto de datos leukemia.arff en Weka.

3.2.1.2 Cuando ambos, número de instancias y características son grandes

Estos tipos de datos se encuentran a menudo en la clasificación de documentos. LIBSVM particularmente no es muy buena para este tipo de problemas. Afortunadamente existe la librería LIBLINEAR, la cual es muy conveniente para estos datos. Para ilustrar la diferencia entre LIBSVM y LIBLINEAR se hará uso de un problema de clasificación de documentos como es el caso de rcv1_train.binary .El número de instancias y características son de 20242 y 47236, respectivamente.

```
Escritorio$ java -Xmx512m -classpath libsvm.jar svm_train -c 4 -t 0 -e 0.1 -m 800 -v 5 rcv1_train.binary
.
Warning: using -h 0 may be faster
....*
optimization finished, #iter = 5020
nu = 0.06122062627168584
obj = -2177.31485338656, rho = -0.52646404989259
nSV = 3648, nBSV = 139
Total nSV = 3648
***
optimization finished, #iter = 5029
nu = 0.05951609815918065
obj = -2102.360912517413, rho = -0.5351328027938939
nSV = 3595, nBSV = 126
Total nSV = 3595
.
Warning: using -h 0 may be faster
...*
optimization finished, #iter = 4944
nu = 0.059683389961314
obj = -2125.0237677400887, rho = -0.5142000862485214
nSV = 3616, nBSV = 133
Total nSV = 3616
Cross Validation Accuracy = 96.9370615551823%
Time: 1021
```

```
liblinear-1.33$ ./train -c 4 -v 5 -e 0.1 rcv1_train.binary
....**.
optimization finished, #iter = 50
Objective value = -1794.695003
nSV = 4567
.....*
optimization finished, #iter = 51
Objective value = -1830.662980
nSV = 4676
....*
optimization finished, #iter = 49
Objective value = -1796.901133
nSV = 4576
Cross Validation Accuracy = 97.026%
Tiempo = 4.86
```

Con 5 folds de validación cruzada, LIBSVM tarda alrededor de 1021 segundos, pero LIBLINEAR demora sólo 4.86. Además, LIBSVM consume mucho más memoria para asignar algunos espacios, como almacenar los elementos del kernel utilizado (véase el –m 800). Evidentemente, LIBLINEAR es mucho más rápido que LIBSVM para obtener un modelo con una precisión comparable.

3.2.1.3 Cuando número de instancias >> número de características

Cuando el número de características es pequeño, se suele mapear los datos a un espacio de mayor dimensión (usando kernels no lineales). Sin embargo, si realmente se desea usar el kernel lineal, se puede utilizar la LIBLINEAR con la opción *-s* 2. Como el número de características es pequeño muchas veces es más rápido que la opción por defecto *-s* 1. Se consideró el covtype.libsvm.binary.scale donde el número de instancias es de 581012 es mucho más grande que el número de características 54.

Se ejecuta la LIBLINEAR con -*s* 1(defecto) y -*s* 2

```
liblinear-1.33$ ./train -c 4 -v 5 -s 1 covtype.libsvm.binary.scale
.......................*
optimization finished, #iter = 236
Objective value = -1288796.888465
nSV = 424523
.......................**
optimization finished, #iter = 237
Objective value = -1291862.322404
nSV = 425253
.......................*
optimization finished, #iter = 232
Objective value = -1289877.610817
nSV = 424761
.......................**
optimization finished, #iter = 237
Objective value = -1291387.145272
nSV = 424827
.......................**
optimization finished, #iter = 237
Objective value = -1292003.298328
nSV = 425133
Cross Validation Accuracy = 75.6652%
Tiempo = 571.28
```

```
liblinear-1.33$ ./train -c 4 -v 5 -s 2 covtype.libsvm.binary.scale
iter  1 act 5.396e+05 pre 5.314e+05 delta 4.400e+00 f 1.859e+06 |g| 5.035e+05 CG   8
iter  2 act 2.828e+04 pre 2.796e+04 delta 4.400e+00 f 1.320e+06 |g| 8.852e+04 CG  14
iter  3 act 2.107e+03 pre 1.911e+03 delta 4.400e+00 f 1.291e+06 |g| 1.985e+04 CG  15
iter  4 act 3.192e+02 pre 3.069e+02 delta 4.400e+00 f 1.289e+06 |g| 4.617e+03 CG  26
iter  1 act 5.367e+05 pre 5.287e+05 delta 4.398e+00 f 1.859e+06 |g| 4.997e+05 CG   8
iter  2 act 2.812e+04 pre 2.785e+04 delta 4.398e+00 f 1.323e+06 |g| 8.738e+04 CG  14
iter  3 act 2.273e+03 pre 2.063e+03 delta 4.398e+00 f 1.294e+06 |g| 1.975e+04 CG  20
iter  4 act 2.024e+02 pre 1.883e+02 delta 4.398e+00 f 1.292e+06 |g| 4.548e+03 CG  25
iter  1 act 5.385e+05 pre 5.306e+05 delta 4.391e+00 f 1.859e+06 |g| 5.000e+05 CG   8
iter  2 act 2.817e+04 pre 2.793e+04 delta 4.391e+00 f 1.321e+06 |g| 8.854e+04 CG  14
iter  3 act 2.188e+03 pre 1.997e+03 delta 4.391e+00 f 1.293e+06 |g| 2.063e+04 CG  15
iter  4 act 3.054e+02 pre 2.926e+02 delta 4.391e+00 f 1.290e+06 |g| 4.807e+03 CG  27
iter  1 act 5.369e+05 pre 5.289e+05 delta 4.398e+00 f 1.859e+06 |g| 5.040e+05 CG   8
iter  2 act 2.767e+04 pre 2.709e+04 delta 4.398e+00 f 1.322e+06 |g| 8.980e+04 CG  12
iter  3 act 2.780e+03 pre 2.513e+03 delta 4.398e+00 f 1.295e+06 |g| 1.901e+04 CG  20
iter  4 act 3.286e+02 pre 3.282e+02 delta 4.398e+00 f 1.292e+06 |g| 4.705e+03 CG  26
iter  1 act 5.364e+05 pre 5.285e+05 delta 4.399e+00 f 1.859e+06 |g| 5.023e+05 CG   8
iter  2 act 2.823e+04 pre 2.804e+04 delta 4.399e+00 f 1.323e+06 |g| 8.715e+04 CG  14
iter  3 act 2.204e+03 pre 2.013e+03 delta 4.399e+00 f 1.295e+06 |g| 2.045e+04 CG  15
iter  4 act 3.013e+02 pre 2.853e+02 delta 4.399e+00 f 1.292e+06 |g| 4.744e+03 CG  22
Cross Validation Accuracy = 75.6702%
Tiempo = 95.10
```

El tiempo total usando la función –s 2 fue de 95.10 segundos mucho menor que utilizando la opción –s 1 que fue de 571.28 segundos. Por lo que se puede apreciar claramente, que usando -s 2 permite minimizar el tiempo de entrenamiento.

Resumido los casos anteriores se puede concluir que cuando el número de instancias es pequeño como en el primer caso, el usuario puede utilizar cualquiera de las dos librerías o kernels mencionados pues se obtendrá buenos resultados con ambas, sin embargo en los casos dos y tres cuando el número de instancias es mediano o grande, LIBLINEAR es una buena opción porque se reduce considerablemente el tiempo y se obtiene con ella modelos de gran precisión.

Por tanto cuando se trata de problemas de gran tamaño como los que se muestran a continuación, se puede apreciar a simple vista las potencialidades de LIBLINEAR respecto a LIBSVM.

Conjunto de datos	Tiempo de ejecución	
	LIBLINEAR	LIBSVM
rcv1_train.binary	4,86 s	0.17 h
covtype-2vr	20,15 s	30,19 h
covtype.libsvm.binary.scale	95,10s	40.35 h

Tabla 3: Comparación de tiempos de ejecución entre LIBLINEAR y LIBSVM

.

3.2.3 Segundo resultado

En este epígrafe se mostrará y analizará las Máquinas de Soporte Vectorial en un ambiente distribuido, haciendo uso de la plataforma de cálculo distribuido T-arenal. Para llevar a cabo este experimento se modificó el tamaño de la muestra, con el objetivo de validar la factibilidad del uso de las técnicas de distribución y se tendrá en cuenta el tiempo de ejecución y el porcentaje de validación cruzada.

Para llevar a cabo este experimento se utilizó el covtype-2vr que cuenta con 300000 instancias y 54 características. Se realizaron diferentes pruebas variando el tamaño de los archivos con el objetivo de buscar un mayor porcentaje de validación cruzada, aunque se sacrifique un poco la velocidad de ejecución y garantizar una mayor calidad en los resultados. Como se puede apreciar en la tabla 4, mientras menor sea el tamaño de los fragmentos mayor será la velocidad y ligeramente mayor la exactitud.

Cantidad de simulaciones	Número de instancias	Número de características	Cantidad de archivos	Tiempo total (seg)	Validación Cruzada (%)
Prueba #1	30000	54	10	15951	63.05
Prueba #2	15000	54	20	3908	63.49
Prueba #3	6000	54	50	191	66.44

Tabla 4: Tiempos de CPU y exactitud de la validación cruzada.

Se puede apreciar mediante una gráfica la disminución del tiempo a medida que se entrena con un número menor de muestras.

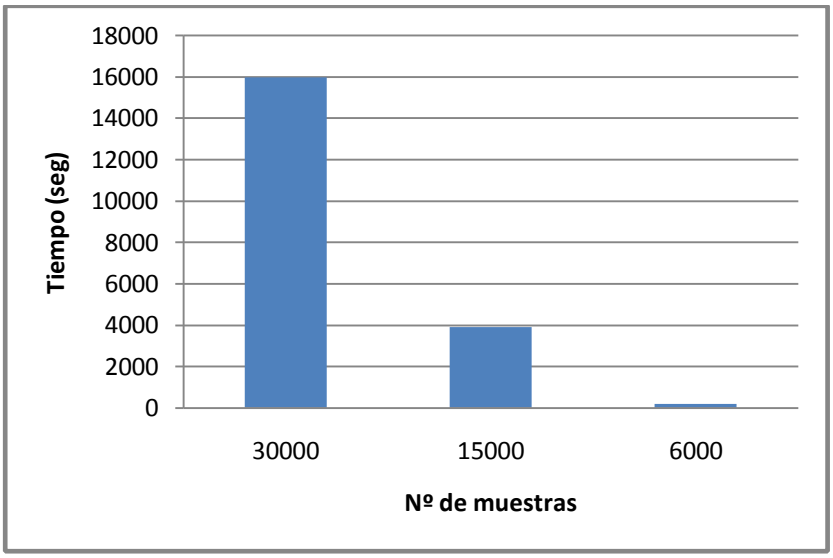

Figura 8: Tiempo de cada prueba.

En la siguiente tabla se muestra una comparación de los tiempos de ejecución y porcentaje de validación cruzada entre la LIBSVM y las MSV distribuidas, como se puede apreciar, LIBSVM es considerablemente lenta en cuanto a tiempo, pero un poco mejor en exactitud. Aunque los resultados obtenidos en un ambiente distribuidos son un poco menores se pueden comparar con la LIBSVM, garantizando que sea una buena opción, pues dependiendo del tamaño de la muestra se puede ganar mucho en velocidad, sacrificando un poco la exactitud.

Archivos	Número de instancias	Tiempo (seg)		% Aceptación
		MSV Distribuidas	LIBSVM	
covtype-2vr	300000	-	108684	71.42
Simulación #1	30000	15951	-	63.05
Simulación #2	15000	3908	-	63.49
Simulación #3	6000	191	-	66.44

Tabla 5: Tiempos de CPU y % de aceptación de las MSV Distribuidas y LIBSVM

Como se puede ver la distribución de las Máquinas de Soporte Vectorial es realmente rápida, pues brinda la posibilidad de aumentar la cantidad de clientes y así disminuir considerablemente el tiempo de aprendizaje, aunque se impone en cuanto a rapidez no es capaz de mejorar el por ciento de aceptación (predicción) con respecto a la LIBSVM, sin embargo logra resolver de una manera eficiente el problema de la clasificación con grandes volúmenes de datos.

Con los experimentos anteriores, queda demostrado que a medida que aumenta la cantidad de clientes en el proceso de aprendizaje, el tiempo de entrenamiento disminuye en gran medida y aunque las pruebas se realizaron con muestras de tamaños diferentes y los mismos parámetros no varió mucho el resultado de las mismas.

3.2.4 Tercer resultado

En este epígrafe se exponen las Máquinas de Soporte Vectorial en un ambiente paralelo utilizando el software πSVM, para ello se utilizará la máquina paralela Átropos con la que cuenta la Universidad de las Ciencias Informáticas. Se realizará una comparación entre la LIBSVM con πSVM teniendo en cuenta el tiempo y el porcentaje de aceptación.

Para realizar las pruebas concurrentes se utilizó el conjunto de datos mnist-576-rbf-8vr y mnist_ 784_poly_8vr que poseen 60000 muestras y 576 características y el covtype.libsvm.binary.scale que

cuenta con 581012 muestras y 54 características. A continuación se muestran algunas imágenes del trabajo realizado con el software πSVM:

Con 1 procesador

```
pisvm-1.1$ mpirun -np 1 ./pisvm-train -s 0 -t 2 -g 1.667 -c 10 mnist_train_576_rbf_8vr.dat
Initializing gradient...done.
  it  | setup time | solver it | solver time | gradient time | kkt violation
    1 |       0.00 |         0 |        0.00 |          0.10 | 2
    2 |       0.00 |         1 |        0.00 |          0.30 | 2.35026
    3 |       0.00 |         1 |        0.00 |          0.29 | 2.55858
 *****
 8433 |       0.00 |         1 |        0.00 |          0.25 | 0.00100251
 8434 |       0.00 |         1 |        0.00 |          0.25 | 0.00100695
 8435 |       0.00 |         1 |        0.00 |          0.25 | 0.000990696

Total opt. time = 2404.68
Problem setup time = 0.25 (0.01%)
Inner solver time = 0.13 (0.01%)
Gradient updating time = 2150.88 (89.45%)

optimization finished, #iter = 8435
nu = 0.002356
obj = -706.795149, rho = -0.946952
nSV = 3292, nBSV = 0
Total nSV = 3292
I/O time = 81.36
```

Con 2 procesadores

```
Initializing gradient...Initializing gradient...done.
done.
  it  | setup time | solver it | solver time | gradient time | kkt violation
    1 |       0.00 |         0 |        0.00 |          0.01 | 2
    2 |       0.00 |         1 |        0.00 |          0.10 | 2.35026
    3 |       0.00 |         1 |        0.00 |          0.09 | 2.55858
 ***
 8433 |       0.00 |         1 |        0.00 |          0.09 | 0.00100251
 8434 |       0.00 |         1 |        0.00 |          0.06 | 0.00100695
 8435 |       0.00 |         1 |        0.00 |          0.01 | 0.000990696

Total opt. time = 555.56
Problem setup time = 1.14 (0.21%)
Inner solver time = 0.07 (0.01%)
Gradient updating time = 443.24 (79.78%)

optimization finished, #iter = 8435
nu = 0.002356
obj = -706.795149, rho = -0.946952
nSV = 3292, nBSV = 0
Total nSV = 3292
I/O time = 25.55
I/O Tiempo = 179.89

optimization finished, #iter = 8435
nu = 0.002356
obj = -706.795149, rho = -0.946952
nSV = 3292, nBSV = 0
Total nSV = 3292
I/O time = 25.76
```

Con 4 procesadores

```
Initializing gradient...Initializing gradient...Initializing gradient...Initializing gradient...done.
done.
done.
  it  | setup time | solver it | solver time | gradient time | kkt violation
    1 |       0.00 |         0 |        0.00 |          0.11 | 2
    2 |       0.00 |         1 |        0.00 |          0.12 | 2.35026
    3 |       0.00 |         1 |        0.00 |          0.11 | 2.55858
***
 8433 |       0.00 |         1 |        0.00 |          0.20 | 0.00100251
 8434 |       0.02 |         1 |        0.00 |          0.35 | 0.00100695
 8435 |       0.09 |         1 |        0.00 |          0.15 | 0.000990696

Total opt. time = 1215.79
Problem setup time = 80.80 (6.65%)
Inner solver time = 0.07 (0.01%)
Gradient updating time = 1000.41 (82.28%)

optimization finished, #iter = 8435
nu = 0.002356
obj = -706.795149, rho = -0.946952
nSV = 3292, nBSV = 0
Total nSV = 3292
I/O time = 45.86
I/O Tiempo = 215.71
***
optimization finished, #iter = 8435
nu = 0.002356
obj = -706.795149, rho = -0.946952
nSV = 3292, nBSV = 0
Total nSV = 3292
I/O time = 45.27
```

Con 8 procesadores

```
pisvm-1.1$ mpirun -np 8 ./pisvm-train -s 0 -t 2 -g 1.667 -c 10 mnist_train_576_rbf_8vr.dat
Initializing gradient...Initializing gradient...Initializing gradient...Initializing
gradient...Initializing gradient...Initializing gradient...Initializing gradient...Initializing
gradient...done.
done.
done.
done.
done.
done.
done.
  it  | setup time | solver it | solver time | gradient time | kkt violation
    1 |       0.55 |        37 |        0.00 |          2.03 | 2
    2 |       0.50 |       128 |        0.00 |          3.17 | 2
    3 |       0.77 |       128 |        0.00 |          3.17 | 2
*****
   76 |       0.73 |        61 |        0.00 |          2.18 | 2
   77 |       0.54 |        61 |        0.00 |          1.93 | 2
   78 |       0.76 |        50 |        0.00 |          1.75 | 0

Total opt. time = 232.64
Problem setup time = 46.13 (19.83%)
Inner solver time = 0.11 (0.05%)
Gradient updating time = 179.14 (77.00%)

optimization finished, #iter = 78
nu = 0.195033
obj = -117020.000000, rho = -1.000000
nSV = 11702, nBSV = 11702
Total nSV = 3292
I/O time = 31.90
```

La solución paralela del problema C-SVC para la clasificación de los datos mnist-576-rbf-8vr y mnist_784_poly_8vr muestran una aceleración casi lineal en un máximo de 4 procesadores mientras que el covtype.libsvm.binary.scale la alcanza con 8. Los parámetros utilizados para el entrenamiento de πSVM y LIBSVM se muestran en la Tabla 6. En todos los datos el tiempo de CPU de LIBSVM es mejor que πSVM con un procesador, aunque este tiempo podría mejorarse mediante la elección de un working set diferente, es importante señalar que con 4 procesadores πSVM es mucho más rápido que LIBSVM y en cuando a la exactitud no importa que se aumente el número de procesadores, pues seguirá siendo la misma en ambos casos.

Los parámetros obtenidos para C-SVC son los siguientes:

Nombre	Parámetros	Kernel	Tamaño del working set
mnist-576-rbf-8vr	C = 10 γ = 1.667	RBF	512/256
Mnist- 784-poly-8vr	C = 10 d = 7	Polinomial	512/256
covtype.libsvm.binary.scale	C = 4 γ = 2e-5	RBF	1024/512

Tabla 6: Parámetros para el problema C-SVC.

Muestras	Tiempo de CPU		Prueba de error (%)	
	πSVM	LIBSVM	πSVM	LIBSVM
mnist-576-rbf-8vr	25.31(min)	25.06(min)	99.82	99.82
mnist- 784-poly-8vr	34.03(min)	30.15(min)	99.21	99.21
covtype.libsvm.binary.scale	41.57(h)	40.39(h)	75.65	75.65

Tabla 7: Tiempo de CPU para πSVM con un procesador y porcentaje de error en comparación con la LIBSVM.

En las siguientes las gráficas en función del tiempo de CPU y la aceleración para los datos mencionados anteriormente.

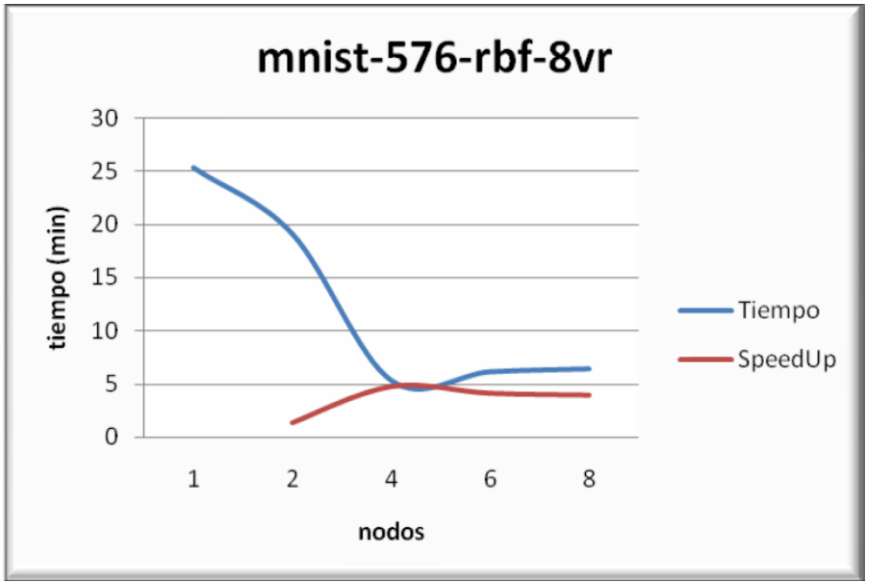

Figura 9: Speedup y tiempo de CPU para mnist-576-rbf-8vr

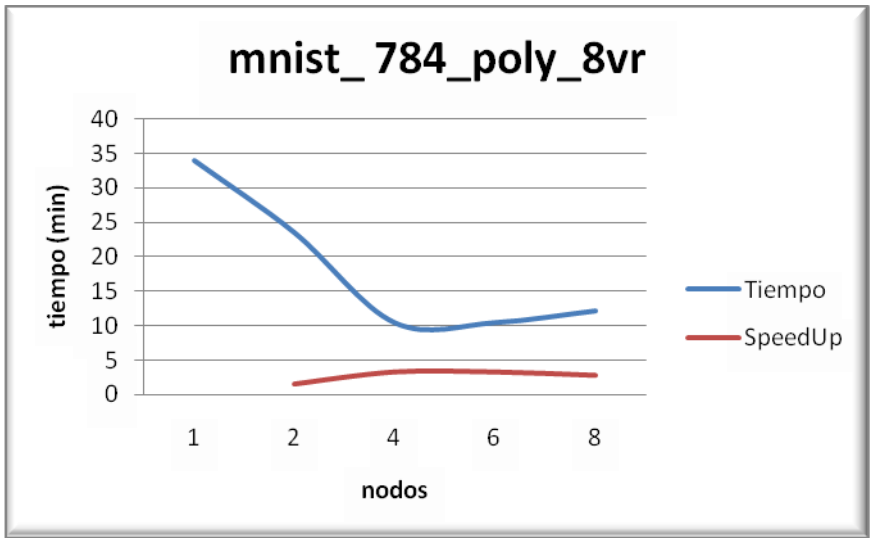

Figura 10: Speedup y tiempo de CPU para mnist_ 784_poly_8vr

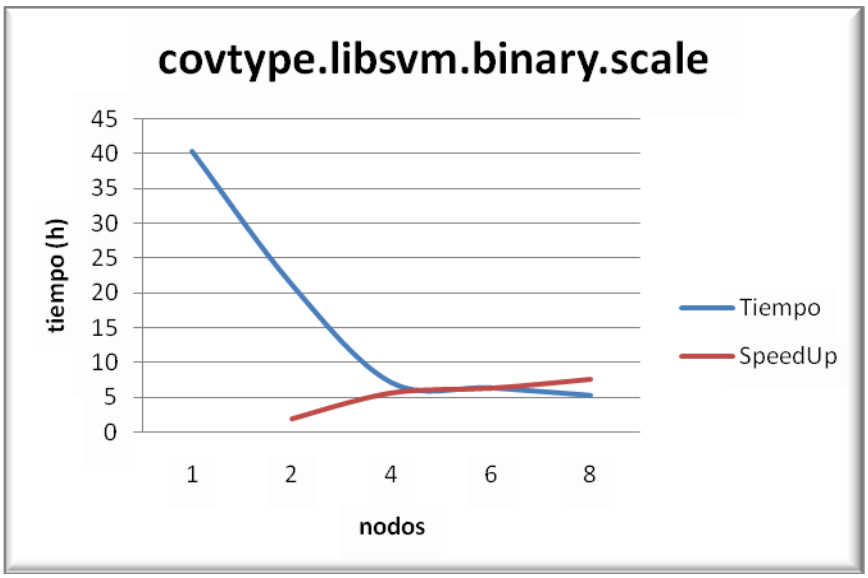

Figura 11: Speedup y tiempo de CPU para covtype.libsvm.binary.scale

3.3 Conclusiones

En el presente capítulo se realizó una descripción de los tres tipos de soluciones para el problema planteado. En las mismas se comparó a la biblioteca LIBSVM con LIBLINEAR, MSV distribuidas y πSVM. Estas tres últimas resultaron ser mucho más rápidas para problemas de gran tamaño y de eficiencia comparable que LIBSVM. Con la diferencia de que LIBLINEAR es solamente para los clasificadores lineales, lo que es una desventaja si se quiere utilizar otro tipo de clasificador. En el caso de πSVM, ésta es la mejor en cuanto a exactitud pero tiene la desventaja de que trabaja solamente sobre Clústeres de computadoras, los mismos al ser poco baratos no se encuentran con mucha frecuencia. Por último en el caso de las MSV distribuidas, aunque es menos exacta, el tipo de sistema sobre el que se ejecuta es más barato y por lo tanto fácil de encontrar, por este motivo se concluye que las MSV distribuidas es la solución más aceptable.

Conclusiones

Luego de todo lo antes expuesto en el presente trabajo es posible concluir que:

- ✓ Se analizaron algoritmos capaces de manejar las Máquinas de Soporte Vectorial para el procesamiento masivo de datos.
- ✓ Se emplearon los algoritmos analizados en diferentes ambientes: locales, distribuidos y paralelos.
- ✓ Se logró comprobar la eficiencia de los algoritmos a través de pruebas que se realizaron utilizando los programas existentes para ello.

Se obtuvieron resultados alentadores en cuanto a la reducción del tiempo de entrenamiento de las Máquinas de Soporte Vectorial para el procesamiento masivo de datos, alcanzando soluciones satisfactorias para esta investigación de acuerdo con los objetivos específicos.

Recomendaciones

- Implementar una aplicación que reúna todos los aspectos de esta investigación y centrarla en la Bioinformática.
- Implementar los nuevos kernels en el software paralelo y compararlos con los ya existentes.
- Profundizar acerca de los scripts grid.py y easy.py para una mejor selección de parámetros.
- Hacer un estudio más profundo acerca de la clasificación multiclase en un ambiente distribuido.

Anexos

Anexo 1

Conjuntos de datos utilizados

Conjunto de datos	Nº de datos de Entrenamiento	Nº de datos de prueba	Nº de características
mnist-576-rbf-8vr	60000	10000	576
mnist-784-poly-8vr	60000	10000	784
covtype.libsvm.binary.scale	581012	N/D	54
covtype-2vr	300000	281012	44
rcv1_train.binary	20242	N/D	47236
Leukemia	38	34	7129

Tabla 8: Tamaño de los conjuntos de datos y de sus características.

mnist-576-rbf-8vr y mnist-784-poly-8vr

Los conjuntos de datos mnist-576-rbf-8vr y mnist-784-poly-8vr proceden de la base de datos MNIST para el reconocimiento de patrones de escritura y solamente difieren en el tipo de procesamiento que se hace. Para mnist-576-rbf-8vr que se utiliza en conjunto con el kernel RBF y posee 576 dimensiones extraídas de los datos originales. El conjunto de datos mnist-784-poly-8vr está preparado para centrar cada dígito de la imagen en un cuadro de 28 × 28. Para SVC los parámetros son C = 10 para las dos bases de datos, γ=1,667 para mnist-576-rbf-8vr y d=7 para mnist-784-poly-8vr y se determinaron mediante la validación cruzada en un subconjunto de datos de entrenamiento. Finalmente en el conjunto original de 10 clases se separaron 2 y se obviaron las 8 restantes.

covtype-2vr y covtype.libsvm.binary.scale

La tarea de distinguir entre las 8 diferentes clases de bosques covertype está representada por el conjunto de datos covtype. Para la conversión a binario se separaron 2 de las otras clases. La preparación de la base de datos y la elección de los parámetros de SVM se realiza como describe en el anexo 1. El kernel RBF es utilizado con el parámetro γ = 2e - 5, el parámetro costo de SVC es C = 10 y la condición de parada es =0,01.

rcv1_train.binary

Clasificación o categorización de Documentos es un problema en las ciencias de la información. La tarea consiste en asignar un documento electrónico a una o más categorías, basadas en su contenido. Las tareas de clasificación de documentos se pueden dividir en dos tipos: la clasificación supervisada de documento donde algunos mecanismos externos (como la información) ofrece información sobre la correcta clasificación de documentos, y sin supervisión, el documento de clasificación, donde la clasificación debe hacerse por completo sin hacer referencia a información externa.

Bibliografía

Carreras, X., Márquez L. y Romero E. (2004), "Máquinas de Vectores Soporte", en Hernández, J., Ramírez, M. y Ferri, C., Introducción a la Minería de Datos, Editorial Pearson, España.

Cristianini, N. y Shawe-Taylor, J. (2000), "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods", Cambridge University Press, United Kingdom.

Statnikov A., Harding D., Guyon I. and Aliferis C. F. (2008), "Support Vector Machines Without Tears", Vanderbilt University.

Goic F.M., "Clase Auxiliar Programación Cuadrática", Departamento de Ingeniería Industrial, Universidad de Chile.

Gunn S.R., "Support Vector Machines for Classification and Regression", Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, University of Southampton.

Chang, C.-C. and C.-J. Lin (2001). LIBSVM: a library for support vector machines. Software disponible en http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Fan R.E., Chang K.W., Lin C.J., "LIBLINEAR: A Library for Large Linear Classification", Department of Computer Science, National Taiwan University, Taiwan.

Brugger D, "Parallel Support Vector Machines", Departamento de técnicas de la informática, Universidad de Tübingen, Dinamarca.

L.J. Cao, J.Q. Zhang, "Parallel Sequential Minimal Optimization for the Training of Support Vector Machines", Financial Studies of Fudan University, P.R. China.

Mantas Ruiz J.M., Introducción a la Interfaz de paso de mensajes (MPI), Depto. de Lenguajes y Sistemas Informáticos, Universidad de Granada.

Platt J.C., "Fast training of support vector machines using sequential minimal optimization," In Advances in Kernel Methods.

Keerthi S.S., Shevade S.K., Bhattaacharyya C. and Murthy K.R.K., "Improvements to Platt's SMO algorithm for SVM classifier design" *Neural Computation*,

Collobert R., Bengio S. and Bengio Y., "A parallel mixture of SVMs for very large scale problems," Neural Computation.

Dong J.X., Krzyzak A. y Suen C.Y., "A fast Parallel Optimization for Training Support Vector Machine".

Zanghirati G., Zanni L., “A parallel solver for large quadratic programs in training support vector machines”.

Guang B.H., Mao K.Z., Siew C.K. and Huang D.S., “Fast Modular Network Implementation for Support Vector Machines”.

Hsieh, C.-J. y otros (2008) “A dual coordinate descent method for large-scale linear SVM”.

C.-J. Lin, R. C. Weng, and S. S. Keerthi (2008), “Trust Region Newton Methods for Large-Scale Logistic Regression”.

Referencias bibliográficas

1. **Carreras, X. y Romero, E. Márquez L.** Máquinas de Vectores Soporte. [aut. libro] J., Ramírez, M. y Ferri, C Hernández. *Introducción a la Minería de Datos.* España : Pearson, 2004.

2. **Cristianini, N y Shawe-Taylor, J.** *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge : s.n., 2000.

3. **Platt J.C.** Fast training of support vector machines using Sequential Minimal Optimization. *Advances in Kernel Methods.*

4. **Keerthi S.S., Shevade S.K., Bhattaacharyya C. and Murthy K.R.K.,.** Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation.*

5. **R., Collobert, S., Bengio y Y., Bengio.** A parallel mixture of SVMs for very large scale problems. *Neural Computation.*

6. **J.X., Dong, A., Krzyzak y Suen C.Y.** *A fast Parallel Optimization for Training Support Vector Machine.*

7. **G., Zanghirati y L., Zanni.** A parallel solver for large quadratic programs in training support vector machines.

8. **B.H., Guang, y otros.** Fast Modular Network Implementation for Support Vector Machines.

9. **Hsieh, C.-J., y otros.** A dual coordinate descent method for large-scale linear SVM. 2008.

10. **C.-J. Lin, R. C. Weng, and S. S. Keerthi.** Trust Region Newton Methods for Large-Scale Logistic Regression. 2008.

11. **Chang, C.-C. y Lin, C.-J.** *LIBSVM: a library for support vector machines.* 2001.

Glosario de términos

Actividad biológica: Actividad que caracteriza el comportamiento biológico en compuestos químicos (Molécula o Fragmento).

Bioinformática: Es la aplicación de los ordenadores y los métodos informáticos en el análisis de datos experimentales y simulación de los sistemas biológicos.

Características: Atributos que posee cada una de las instancias.

Entrenamiento: Acción que se realiza para el aprendizaje de ciertas muestras.

Escalar: Equilibrar o balancear los conjuntos de datos cuando estos se encuentran desproporcionados.

Folds: Número de segmentos divididos en la validación cruzada.

Funciones kernel: Son funciones matemáticas que se emplean en las Máquinas de Soporte Vectorial.

Instancias: Cada uno de los casos a clasificar.

Máquinas de Soporte Vectorial: Técnica de inteligencia artificial para la clasificación o regresión.

Overfitting: Sobreajuste o sobreaprendizaje de la muestra lo que conlleva a una pobre generalización.

Predicción: Acción que el sistema realiza para emitir un resultado.

Ruido: Instancias del conjunto de datos que no brindan ninguna información y que obstaculizan el proceso de entrenamiento.

Validación Cruzada: Método de estimación aleatorio que indica el por ciento de exactitud de los datos que serán correctamente clasificadas en una muestra.