UNIVERSIDAD NACIONAL DE
EDUCACIÓN A DISTANCIA

Departamento de Informática y Automática
Escuela Técnica Superior de Ingeniería Informática

MÁQUINAS DE VECTORES SOPORTE EN ENTORNOS DE SUPERCOMPUTACIÓN: APLICACIÓN A FUSIÓN NUCLEAR

TESIS DOCTORAL

D. Jesús Manuel Ramírez Pérez
Licenciado en Informática

2014

UNIVERSIDAD NACIONAL DE
EDUCACIÓN A DISTANCIA

Departamento de Informática y Automática
Escuela Técnica Superior de Ingeniería Informática

MÁQUINAS DE VECTORES SOPORTE EN ENTORNOS DE SUPERCOMPUTACIÓN: APLICACIÓN A FUSIÓN NUCLEAR

TESIS DOCTORAL
2014

Autor: **D. Jesús Manuel Ramírez Pérez**

Directores: **Dr. D. Sebastián Dormido Canto**
Dr. D. Jesús Antonio Vega Sánchez

A Julián, José Manuel y Ana

Agradecimientos

He podido llevar adelante la presente Tesis Doctoral gracias a la ayuda de muchas personas, a quienes quiero agradecer su apoyo y sobre todo, sus muestras de amistad durante todo el tiempo que ha durado este camino.

En primer lugar, quiero dar las gracias al Dr. Jesús Antonio Vega Sánchez, por todas sus orientaciones, soporte, correcciones y horas de dedicación a mi trabajo en esta Tesis. Su capacidad de trabajo y su dedicación a la investigación han sido un gran referente para mí durante estos años.

A mi director de Tesis, Dr. Sebastián Dormido Canto, sin cuyo apoyo incondicional y colaboraciones hubiese sido imposible terminar este trabajo. Y sobre todo, sus palabras de ánimo, y su confianza en mi trabajo, han sido las que han permitido que esta Tesis viera la luz.

A mis compañeros del Centro de Supercomputación de Galicia, que en los inicios de este trabajo me prestaron su ayuda para poder realizar las primeras simulaciones cuando aún no tenía a mi disposición un gran entorno de supercomputación.

A mis dos compañeros y amigos, Agustín Corrales y Agustín Rodríguez, que han permitido con sus correcciones lingüísticas, que esta Tesis, y los trabajos en inglés asociados a ella, hayan sido más comprensibles.

A mis compañeros, Augusto Pereira y Sergio González, que me han hecho sentir como en casa cada vez que he visitado el CIEMAT, en las múltiples reuniones de seguimiento de la investigación realizada. Asimismo, mi agradecimiento al "*Programa de acceso a investigadores externos en el TJ-II*", financiado por la Secretaría de Estado de Investigación del Ministerio de Ciencia e Innovación, que en sus convocatorias de 2009, 2010 y 2011 me han concedido ayudas para los viajes y las estancias en el CIEMAT.

También quiero agradecer de forma especial a mis amigos Francisco Gallardo y Juan Manuel Mora su apoyo y su amistad. Habéis contribuido de forma muy especial en este trabajo. Es un lujo saber que siempre puedo contar con vosotros.

Por último, agradecer el apoyo de toda mi familia y, en especial, de mi esposa. Pepi, gracias por la paciencia, comprensión, horas de dedicación y cariño incondicional que me has demostrado durante todos estos años.

COMENTARIOS INICIALES Y RESUMEN

El objetivo de este proyecto de Tesis es la aplicación de técnicas de minería de datos para la predicción y clasificación de eventos físicos en plasmas termonucleares.

La estructura del trabajo descrito en esta memoria incluye una introducción sobre los principios de la fusión nuclear y una descripción de los objetivos planteados cuando se comenzó el trabajo, que están detallados en los Capítulos 1 y 2 respectivamente.

En el Capítulo 3, se describe de forma amplia la técnica de creación de modelos de aprendizaje, máquinas de vectores soporte (SVM, Support Vector Machines).

En el Capítulo 4 se exponen las arquitecturas y técnicas generales de paralelización para, finalmente, describir la implementación paralela de SVM realizada.

El Capítulo 5 comienza con una revisión del estado del arte sobre un fenómeno físico de gran importancia para la fusión nuclear: las disrupciones. Posteriormente se detalla el sistema de detección de disrupciones que hace uso del software descrito en el capítulo anterior.

En el Capítulo 6 se hace un planteamiento diferente al realizado en el Capítulo 5, ya que se parte de la hipótesis de ausencia de datos. El objetivo es crear modelos que aprendan de manera continua desde las primeras descargas de un nuevo dispositivo, tal y como tendrá que suceder en ITER.

En el Capítulo 7 se hace uso de nuevas herramientas de minería de datos, como son los algoritmos genéticos, para hacer un estudio de la información que aportan los diferentes grupos de señales en la detección de disrupciones.

Finalmente, en el Capítulo 8 se resumen y valoran los objetivos alcanzados y se proponen nuevas líneas de trabajo futuro.

Tabla de contenidos

Índice de Figuras

Índice de Tablas

CAPÍTULO 1

Fusión nuclear

1.1 Introducción

La producción y la utilización de la energía se han convertido en los últimos tiempos en objeto de discusión y preocupación por parte de la comunidad mundial. Además de que el consumo se ha disparado, muchos de los recursos energéticos llamados no renovables tienen fecha de caducidad.

Para empezar, si sigue el ritmo actual de crecimiento de la población, se estima que esta podrá estar cerca de doblarse pasada la mitad del siglo XXI, lo que por sí mismo representaría un grave problema de abastecimiento. Pero además, está probado que el consumo energético per cápita está directamente relacionado con el nivel de vida y el desarrollo tecnológico del individuo. Por tanto, teniendo en cuenta que la mayor parte de la población mundial está incluida dentro de los países en vías de desarrollo, el dato es aún más preocupante. Si en los próximos años el desarrollo de países como China o India les lleva a alcanzar una cifra de consumo cercana a la de los países occidentales, el impacto sobre el nivel de producción conllevaría una rápida disminución de las reservas mundiales de combustibles y dispararía la producción de contaminantes.

De hecho, con las reservas actuales y manteniendo estable el consumo, el petróleo y el gas natural se agotarán en el plazo de 50 años. De cualquier manera, lo que parece claro es que los recursos actuales de energías no

renovables, es decir aquellas energías cuyos recursos están cuantificados y no se reponen a corto plazo, no garantizan el consumo futuro del planeta.

Una posible solución a toda esta problemática consistiría en explotar el recurso de la fisión nuclear. Esta solución posee muchas ventajas con respecto a los combustibles tradiciones, la principal es que no emite gases contaminantes. Sin embargo, esta solución plantea también los siguientes problemas:

- Posibilidades de accidente, hacen necesarias unas medidas de seguridad muy altas, con el consiguiente coste de las mismas.
- Financiación costosa y largo plazo de construcción.
- Acumulación de residuos radiactivos de alta actividad durante al menos 100.000 años.

Todo ello, junto con el malestar social que llevan asociadas, hace que no esté nada claro su viabilidad como solución definitiva al problema energético mundial.

Las energías renovables constituyen una alternativa a corto plazo a las energías tradicionales, gracias a que la materia prima en la que se basan procede de una fuente inagotable. Es el caso del sol, del viento o del calor del magma del manto terrestre. Además, su baja producción de contaminantes, restringida en la mayoría de los casos al proceso de conversión a formas de energía aprovechables por el ser humano, las hace muy atractivas. Sin embargo, también tienen desventajas importantes. Por ejemplo, su baja densidad, lo que dificulta su captación y concentración, su baja disponibilidad, su variabilidad asociada a los distintos periodos naturales, el elevado coste de implantación y explotación, el impacto ambiental visual para el caso de la energía eólica y solar o la destrucción de ecosistemas, en el caso de la energía hidráulica.

Como hemos visto, es un hecho insoslayable que el agotamiento de los combustibles fósiles está cercano, y que se hace imprescindible la investigación e innovación del campo energético, dada la inmadurez y la baja rentabilidad de las energías renovables. Una de las opciones más prometedoras la proporciona

la fusión termonuclear controlada. Esta opción, propuesta ya desde los años 40, parece la solución perfecta a todos los problemas planteados.

Principalmente, genera energía mediante la fusión de núcleos atómicos ligeros, que dan lugar a otros más pesados. La energía que se genera puede entonces convertirse a alguna forma útil para su explotación humana. En cierta forma, el ejemplo más claro de su factibilidad y eficacia nos lo dan las estrellas, que funden núcleos de hidrógeno para dar helio. La energía resultante se emite en forma de luz que es la responsable de la aparición y el mantenimiento de la vida en la Tierra. La producción de energía por fusión utilizaría como combustible los dos isótopos pesados del hidrógeno: el deuterio y el tritio. El primero se encuentra en relativa abundancia en la naturaleza, como en el agua del mar, de donde podría extraerse de manera virtualmente ilimitada. En cuanto al tritio, es un elemento radiactivo de muy corta vida media, que puede producirse artificialmente a partir del litio.

La energía de fusión nuclear no tiene el riesgo de accidentes nucleares incontrolados, que sí afecta al caso de la fisión. Además, tampoco genera residuos radioactivos de alta actividad, ya que el período de semidesintegración del tritio es de 12 años.

Por otro lado, en el caso de la contaminación atmosférica, la fusión comparte la gran ventaja de las centrales de fisión. Es decir, al no existir combustión, el único gas que se vierte a la atmósfera es vapor de agua. El desarrollo exitoso de esta tecnología acabaría con la dependencia del mundo occidental de los combustibles fósiles, y también evitaría los problemas sociales que generan las centrales nucleares convencionales, dando a la sociedad una fuente de energía limpia, segura, de alta potencia y prácticamente inagotable.

1.2 La energía de fusión

La fusión es el proceso en el que dos núcleos de átomos ligeros (como el deuterio y el tritio) se unen formando elementos más pesados. La gran fuerza gravitacional que aparece en el interior de las estrellas (como en nuestro sol)

comprime enormemente el hidrógeno presente y logra que se produzca la fusión de los núcleos. La energía generada es el resultado de la conversión de dos núcleos atómicos que se unen para formar uno de mayor masa atómica. Sin embargo, las presiones y las temperaturas en el núcleo de las estrellas distan de las condiciones normales en la Tierra.

En la década de los 50 se formalizaron los primeros intentos para la producción de energía mediante fusión [Braams, 2002]. Para que los núcleos cargados positivamente sean capaces de superar las fuerzas electrostáticas de repulsión que existen entre ellos y se acerquen lo suficiente como para producir reacciones de fusión a un ritmo adecuado, hay que alcanzar temperaturas del orden de centenares de millones de grados (de aquí procede la denominación de fusión TERMO-nuclear). A estas temperaturas, el gas se encuentra en estado de plasma, al que se le puede definir como un conjunto de partículas cargadas positiva y negativamente sin ninguna estructura atómica que las ligue. En otras palabras, un plasma es un gas altamente ionizado.

Las reacciones de fusión. Las reacciones de fusión más sencillas de realizar en la tierra son las que se consiguen a partir de isótopos del hidrógeno:

$$
\begin{aligned}
D_1^2 + T_1^3 &\rightarrow He_2^4\,(3.5\ MeV) + n\ (14.1\ MeV)\\
D_1^2 + D_1^2 &\rightarrow He_2^3\,(0.8\ MeV) + n\ (2.4\ MeV)\\
D_1^2 + D_1^2 &\rightarrow T_1^3\,(1.0\ MeV) + p\ (3.0\ MeV)\\
D_1^2 + He_2^3 &\rightarrow He_2^4\,(3.6\ MeV) + p\ (14.7\ MeV)
\end{aligned}
$$

Las secciones eficaces para estas reacciones se indican en la *Figura 1.1*, donde se observa que por debajo de 1 keV éstas tienen unos valores muy bajos, aunque crecen muy deprisa. En torno a temperaturas del orden de 10 keV (alrededor de 100 millones de grados), la reacción entre el *D* y *T* es varios órdenes de magnitud más probable que las restantes [Sheffield, 1994]. Por ello se ha elegido esta reacción para ser utilizada en la primera generación de reactores de fusión.

El deuterio es un isótopo estable del hidrógeno presente en la naturaleza en una proporción de 1 por cada 6670 átomos de hidrógeno y su separación isotópica es relativamente fácil. A pesar de esta pequeña proporción, la cantidad de deuterio contenida en los océanos sería prácticamente inagotable. El tritio es un isótopo inestable del hidrógeno, decae radiactivamente con una vida media de 12.36 años y no existe en la naturaleza, por lo tanto debe ser obtenido.

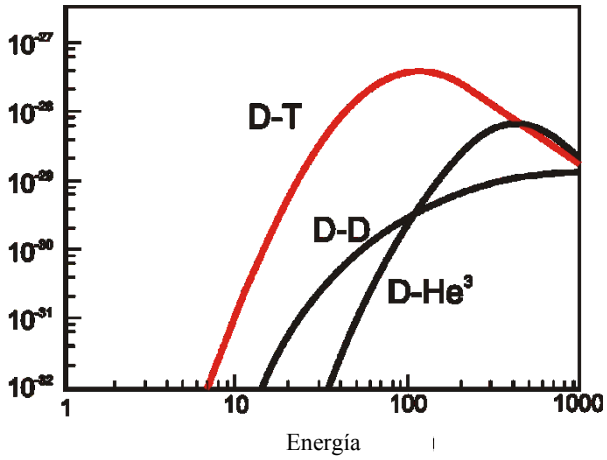


Figura 1.1: Secciones eficaces de las reacciones D-T, D-D y D-He^3.

Ciclo del tritio. En la reacción *D-T*, el 80% de la energía la transporta el neutrón resultante, este neutrón puede servir para producir tritio al reaccionar con el litio según la ecuación:

$$Li + n \rightarrow He + T + 4.8\,MeV$$

Por lo tanto, si se rodea al recinto donde ocurren las reacciones de fusión de una envoltura de litio (llamado manto fértil), el reactor sólo necesitará una carga inicial de tritio, ya que posteriormente él mismo generará el tritio necesario.

La fusión entre el *D-T* tiene, además de la complicación del manejo del tritio, otra dificultad añadida, que es la producción de un neutrón altamente energético capaz de modificar las propiedades mecánicas y radiactivas de los materiales estructurales del reactor. Desde este punto de vista, la fusión entre el *D-*He^3, es mucho más atractiva, al ser una reacción que no produce neutrones, pero requiere temperaturas más altas. Además, el He^3 no existe en nuestro

planeta, por lo que esta reacción no se considera actualmente en ningún escenario bajo estudio.

Balance energético. La condición para un balance energético neutro (en el que la potencia generada de las reacciones de fusión es igual a la potencia suministrada al combustible, Q=1), viene definida por el criterio de Lawson [Lawson, 1957], en el que el producto de la densidad iónica por el tiempo de confinamiento debe ser ≥ 10^{21} m^{-3}·s

Para obtener la condición de ignición, en la que el sistema es capaz de producir sus propias reacciones, deberá cumplirse

$$n_i \tau_e T_i \geq 5x10^{21} \text{ keV} \cdot \text{m}^{-3} \cdot \text{s}$$

donde τ_e es el tiempo de confinamiento, y T_i la temperatura iónica.

1.3 Métodos de confinamiento

Para poder calentar el combustible a la temperatura necesaria para que se produzcan suficientes reacciones de fusión, es necesario mantenerlo aislado, confinado en un cierto volumen y alejado del contacto con otros materiales. Hoy en día, los dos métodos investigados son el confinamiento inercial y el magnético.

a) **Confinamiento inercial**. En la fusión por confinamiento inercial, se concentran uniformemente haces láser sobre la superficie de una microesfera de pocos mm de diámetro de deuterio y tritio. El material en la superficie se evapora, y por el principio de la conservación de la cantidad de movimiento, se provoca que el resto de materia se comprima, llevando el combustible a densidades del orden de 1000 a 10000 veces la del estado sólido. Al mismo tiempo, la temperatura en el centro de la esfera alcanza los 100 millones de grados necesarios para producir las primeras reacciones de fusión, que se propagan por toda la micro-bola hasta que el combustible es consumido totalmente. Todo este proceso se realiza en tiempos de nanosegundos.

b) **Confinamiento magnético**. En este método, el plasma se confina mediante campos magnéticos que actúan creando una trampa magnética para las partículas cargadas del plasma. En las máquinas toroidales (que han sido las más desarrolladas), el campo magnético final se crea superponiendo a un campo magnético, según el eje del toro (toroidal), otro campo en dirección azimutal (poloidal). Las dos principales familias de dispositivos de fusión por confinamiento magnético, son el Tokamak y el estellarator.

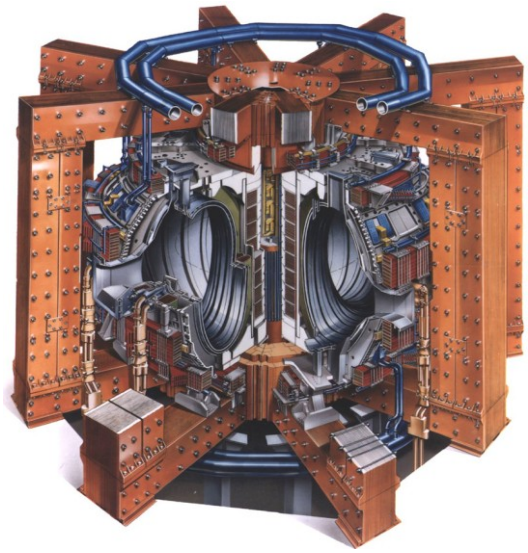


Figura 1.2: Esquema constructivo del tokamak JET (Reino Unido).

En un tokamak el campo magnético toroidal se genera mediante un conjunto de bobinas equiespaciadas alrededor de la cámara de vacío, y el campo magnético poloidal se crea mediante una corriente inducida en el plasma. En la *Figura 1.2* se puede apreciar el esquema constructivo de un dispositivo tokamak.

En los estellarators [McCracken, 2005], tanto el campo magnético toroidal como el poloidal se generan por bobinas externas y no se inducen corrientes en el plasma. Con respecto a los tokamaks, los estellarators tienen como ventajas la capacidad de operación en estado estacionario y la ausencia de disrupciones y como principal desventaja su complejidad de diseño y construcción.

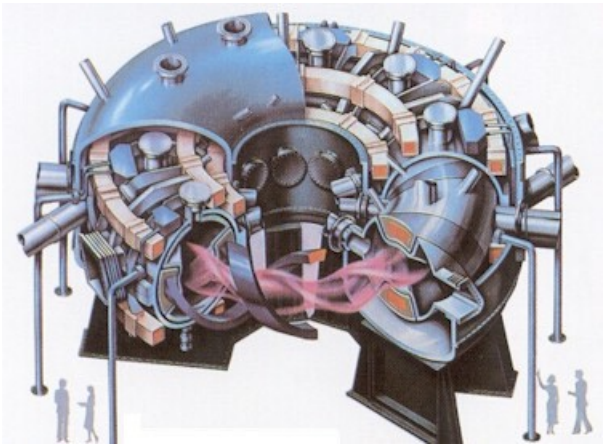


Figura 1.3: El prototipo LHD Torsatron (Japón).

En Madrid se encuentra en funcionamiento el estellarator más grande de Europa, el TJ-II [Alejaldre, 1999], [Sánchez, 2011]. Sus objetivos principales son el estudio del confinamiento y del transporte (de energía y de partículas) en plasmas calientes. El diseño de sus bobinas permite una gran variedad de configuraciones magnéticas, y un amplio rango de valores de la transformada rotacional. Ésta puede variar, en el centro del TJ-II, entre 0.9 y 2. Esta característica lo distingue de otros estellarators al ser capaz de generar una gran variedad de geometrías magnéticas en el plasma. Su eje magnético es helicoidal, confiriéndole una extremada tridimensionalidad.

1.4 Planta de fusión nuclear productora de electricidad

Independientemente de que un futuro reactor sea un tokamak o un estellarator, se sabe que una planta de fusión basada en reacciones D-T producirá partículas alfa ($^4_2$He) que depositarán su energía en el plasma, calentándolo. También se obtendrá un neutrón de alta energía que atraviesa sin problemas el campo magnético y deposita su energía en el manto fértil formado por litio, calentándolo por un lado y forzando la reacción en la que se obtiene tritio y helio.

Tanto el tritio del manto fértil como las partículas alfa del plasma han de extraerse. El primero, para volver a ser introducido en la cámara y el segundo como ceniza no utilizable. La energía almacenada en el manto fértil se extrae a

través de un intercambiador, para posteriormente producir el vapor que mueva la turbina de un generador eléctrico.

Por lo tanto, excepto por la carga inicial, los combustibles básicos de este reactor son el deuterio y el litio, y el residuo es el gas inerte helio. De forma intrínseca, el proceso de fusión no produce ningún tipo de residuo radiactivo. Hay que considerar que el neutrón rápido que se obtiene de la fusión *D-T*, puede alcanzar los elementos estructurales del reactor activándolos, por lo que hay importantes programas de investigación cuyo objetivo es el desarrollo de materiales cuya activación sea mínima y decrezca rápidamente.

CAPÍTULO 2

Definición del problema

2.1 Fenómenos físicos

En un dispositivo de fusión nuclear pueden aparecer fenómenos físicos de diversa naturaleza que es necesario predecir y detectar. Este trabajo se ha centrado en el desarrollo de software tanto para el reconocimiento de transiciones de confinamiento como para la predicción de disrupciones.

2.1.1 Transiciones de confinamiento

Mediante el extenso estudio de las propiedades del plasma, se han identificado a través de los años diferentes regímenes de confinamiento. Conocidos como modos de confinamiento, estos regímenes usualmente ofrecen características únicas en cuanto a los procesos físicos experimentados por el plasma. Los modos más observados y estudiados son el modo óhmico (exclusivo de tokamaks), el modo L, el modo H y las configuraciones que incluyen barreras internas de transporte, las cuales mejoran sustancialmente el confinamiento [Wesson, 2004]. Estos regímenes han sido no sólo observados sino además reproducidos en un gran número de dispositivos de fusión.

Inicialmente, la energía aplicada al plasma en un tokamak estaba predominada por el calentamiento óhmico (modo óhmico). Aunque los tiempos de confinamiento en este tipo de operación aparentaban ser prometedores, se descubrió que el simple efecto Joule era insuficiente para alcanzar los altos parámetros requeridos por un dispositivo termonuclear. Mientras la temperatura del plasma aumenta, su resistencia disminuye, reduciendo consecuentemente el

nivel de calentamiento que puede ser obtenido. El uso de técnicas adicionales para el calentamiento (generalmente inyección de partículas neutras y microondas) incrementaron las temperaturas alcanzadas. Este tipo de régimen con calentamiento adicional se denomina modo L ("Low Confinement Mode"), o modo de bajo confinamiento.

Por otro lado, también se sabe que por encima de un cierto umbral de potencia de calentamiento, se observa un incremento en el tiempo de confinamiento de la energía de más de un factor 2. A este régimen se le llamó modo de alto confinamiento o modo H (del inglés "High Confinement Mode").

La determinación del instante en el que el plasma cambia de un régimen de bajo a alto confinamiento tiene un marcado interés científico, por ello existen numerosos trabajos que han desarrollado sistemas automáticos para su detección, [Vega, 2009 (1)], [González, 2012 (1)] y [González, 2012 (2)].

2.1.2 Disrupciones

La configuración tokamak se presenta como una seria opción para el desarrollo de un futuro reactor. Sin embargo, se ve sometida con frecuencia a pérdidas abruptas del confinamiento del plasma llamadas disrupciones [Schuller, 1995], [ITER, 1999]. Estas repentinas inestabilidades que causan el fin de la descarga, además de afectar a la continuidad de la operación del dispositivo, pueden constituir un serio factor de riesgo para su integridad. Durante una disrupción, en períodos de tiempo del orden de milisegundos, el plasma excede sus límites operacionales con la resultante pérdida de energía y corriente. Durante la primera fase de la disrupción es posible que se transfieran altísimas cargas térmicas a los componentes de la primera pared de la máquina. Posteriormente, grandes corrientes son inducidas en la cámara de vacío y en las estructuras aledañas, provocando fuerzas capaces de causar daños considerables. Actualmente, su aparición durante la operación es inevitable, especialmente en configuraciones de alto rendimiento. La necesidad de mitigar las consecuencias de este evento físico mediante su temprana detección, será esencial en el próximo Tokamak ITER [Boozer, 2012].

La caracterización física de las disrupciones para su posible predicción y control es extremadamente compleja, debido a la gran cantidad de variables involucradas en el fenómeno y la relación altamente no lineal entre ellas. Desafortunadamente, solamente existen modelos teóricos parciales que no son capaces de tratar las disrupciones de manera fiable. Debido a esta falta de teoría desde primeros principios, habitualmente se utilizan modelos estadísticos de aprendizaje para su predicción.

2.2 Naturaleza de los datos y técnicas de clasificación

Los datos de cada descarga de un dispositivo de fusión son obtenidos a través de diagnósticos y almacenados en bases de datos para su tratamiento diferido. Normalmente es imposible su análisis en tiempo real debido a la gran cantidad de datos adquiridos y la corta duración de la descarga (500 ms en el estellarator TJ-II, algunos segundos en el Tokamak alemán Asdex Upgrade y alrededor de 30 s en el Tokamak JET, el dispositivo de fusión más grande del mundo). Cada descarga adquiere el mayor número de datos posible para tratar de conocer las propiedades físicas del plasma con la mayor precisión posible. El Tokamak JET ha llegado a adquirir hasta 10GB de datos en una descarga y su base de datos ocupa unos 100 Tbytes. El dispositivo ITER adquirirá del orden de 1 Tbyte de datos por descarga y del orden de Pbytes al año. Teniendo en cuenta esta gran cantidad de datos almacenada, resulta imposible a efectos prácticos analizar toda la información. Para facilitar estas labores, es necesario utilizar técnicas inteligentes de acceso a datos [Vega, 2008], [Dormido-Canto, 2006], [Vega, 2009 (2)], de creación automática de bases de datos, [Vega, 2010 (1)], [Vega 2010 (2)] y de generación de modelos basados en aprendizaje automático, [Dormido-Canto, 2008], [Duro, 2009]. Como parte del aprendizaje automático, existen técnicas de clasificación que permiten agrupar muestras de acuerdo a ciertos criterios. Dependiendo de si se conocen o no las clases donde se van a ir clasificando las muestras, en estos casos se habla de clasificación supervisada o no supervisada respectivamente.

El objetivo de la clasificación dentro del aprendizaje automático consiste en la asignación de un objeto (que puede representar un fenómeno físico) a una de las diversas categorías o clases especificadas.

2.2.1 Clasificación supervisada

Este tipo de clasificación cuenta con un conocimiento a priori, es decir, para la tarea de clasificar un objeto dentro de una categoría o clase se parte de modelos ya clasificados (objetos agrupados que tienen características comunes). Se pueden diferenciar dos fases dentro de este tipo de clasificación:

- En la primera fase se tiene un conjunto de entrenamiento o de aprendizaje (para el diseño del clasificador) y otro llamado de test o de validación (para clasificación), ambos conjuntos servirán para construir un modelo o regla general para la clasificación.

- La segunda fase constituye el proceso en sí de clasificar los objetos o muestras de las que se desconoce la clase a la que pertenecen.

2.2.2 Clasificación no supervisada

A diferencia de la clasificación supervisada no se posee un conocimiento a priori de las categorías involucradas en el proceso de clasificación. A la clasificación no supervisada se la suele llamar también *clustering*.

En este tipo de clasificación, se dispone de "objetos" o muestras que tienen un conjunto de características, de las que no se sabe a qué clase o categoría pertenecen [Liu, 1998], siendo la finalidad el descubrimiento de grupos de "objetos" cuyas características afines permitan separar las diferentes clases.

2.3 Necesidad de la paralelización

En este trabajo se ha optado por la clasificación supervisada ya que existe conocimiento a priori de los datos con los que se va a trabajar. Además, es la técnica más adecuada cuando se pretenden obtener modelos de predicción de eventos. Dentro de los métodos supervisados, se han utilizado las Máquinas de Vectores Soporte (Support Vector Machines, SVM) [Vapnik, 2000], [Cherkassky, 1998] que permiten clasificar muestras con una alta dimensionalidad, como ya se comentó anteriormente.

Las SVM tienen ciertas características que las han puesto en ventaja, comparadas con otras técnicas de clasificación:

- Usar un principio inductivo, que busca la minimización del riesgo estructural usando una función de transformación llamada kernel, les da una gran capacidad de generalización, incluso cuando el conjunto de entrenamiento es pequeño y de alta dimensión.

- El proceso de entrenamiento (o aprendizaje) no depende necesariamente del número de atributos, por lo que se comportan muy bien en problemas de alta dimensionalidad.

- Existen pocos parámetros a ajustar; el modelo generado sólo depende de los datos con mayor información.

- El modelo final puede ser escrito como una combinación de un número muy pequeño de vectores de entrada, llamados *vectores soporte*.

- El compromiso entre la complejidad del clasificador y el error puede ser controlado explícitamente.

Pero las SVM también tienen desventajas:

- Cuando los parámetros están mal seleccionados se puede presentar un problema de sobre-entrenamiento, lo cual ocurre cuando se han aprendido muy bien los datos de entrenamiento pero no se pueden clasificar bien las muestras futuras. Es decir, no existe suficiente capacidad de generalización. En la *Figura 2.1* se observa claramente cómo el clasificador representado por la línea verde se ajusta a la perfección a los datos de entrenamiento. Sin embargo, el clasificador de la línea negra, mucho más sencillo, probablemente tendrá mejor capacidad de generalización.

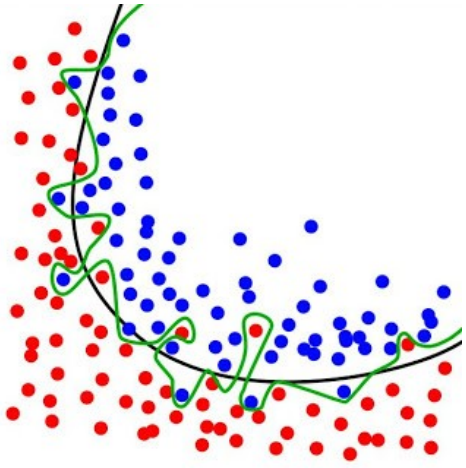


Figura 2.1: Problema de sobre-entrenamiento.

- En gran medida, la solución al problema, así como su generalización, depende del kernel que se use y de los parámetros del mismo.

- Gran esfuerzo computacional para resolver el problema de minimización de SVM con gran cantidad de datos (> 10^5) de alta dimensión (> 15).

La solución a esta última desventaja es el principal objetivo de esta Tesis. Se va a abordar la paralelización de aplicaciones basadas en SVM, permitiendo así una ejecución más rápida que ayude en la búsqueda de los parámetros óptimos y que permita obtener el mejor modelo en la menor cantidad de tiempo.

CAPÍTULO 3

Máquinas de Vectores Soporte

3.1 Introducción

Durante los últimos 30 años, y de la mano del rápido crecimiento del poder de cálculo de los computadores, una parte importante de la teoría de inducción de modelos sobre la base de patrones, desarrollada desde los años 50, ha encontrado la plataforma física necesaria para ser aplicada a problemas reales. La mayoría de estos modelos de aprendizaje abordan dos tipos de problemas: el *Reconocimiento de Patrones o Clasificación* y la *Aproximación de Funciones o Regresión,* [Marqués de Sá, 2001]. Sin embargo, aunque han dado muy buenos resultados, la mayoría de los modelos puestos en práctica utilizan métodos de síntesis (entrenamiento) que desembocan frecuentemente en estimaciones locales o en la utilización de métodos empíricos para encontrar un modelo general.

Las *Redes Neuronales Artificiales* [Hilera, 1995], [Chen, 1996] y [Patterson, 1996], forman parte de los modelos que tienen como marco una teoría conocida con el nombre de Máquinas de Aprendizaje (Learning Machines, LM), [Haykin, 2009]. Esta teoría describe las condiciones necesarias y suficientes para la consistencia, al minimizar la magnitud del error de entrenamiento. La consistencia significa que se alcanza un nivel aceptable de generalización (una buena descripción para datos no conocidos). El progreso de esta teoría fue además la base para el desarrollo de importantes conceptos que

desembocarían en un nuevo principio, aún más general llamado *Minimización del Riesgo Estructural*.

Gracias al principio de Minimización del Riesgo Estructural se desarrolló un método conocido con el nombre: *Máquinas de Vectores Soporte* o en ingles Support Vector Machines (SVM) [Christianini, 2000]. En los últimos años ha crecido considerablemente la cantidad de aplicaciones e implementaciones de este método, que en su forma original tiene la fundamentación matemática que garantiza que se alcanzaría la solución global en un tiempo finito y con una mínima parametrización. Como ejemplos de campos en los que se ha hecho uso de la técnica SVM cabe destacar la Biología [Dror, 2005], la Medicina [Yuh-Jye, 2000], la Economía [Tay, 2001], etc. Un listado más exhaustivo de ejemplos de aplicación de SVM puede encontrarse en [Guyon, 2013].

El algoritmo de las SVM fue desarrollado por Vapnik [Vapnik, 2000] y está basado en la teoría del Aprendizaje Estadístico. Las SVM en el caso de clasificación tienen como objetivo encontrar un hiperplano óptimo que separe dos clases, ver *Figura 3.1*. Para encontrar el hiperplano óptimo hay que minimizar la norma de un vector (w) que define el hiperplano de separación. Esto es equivalente a maximizar el margen entre dos clases.

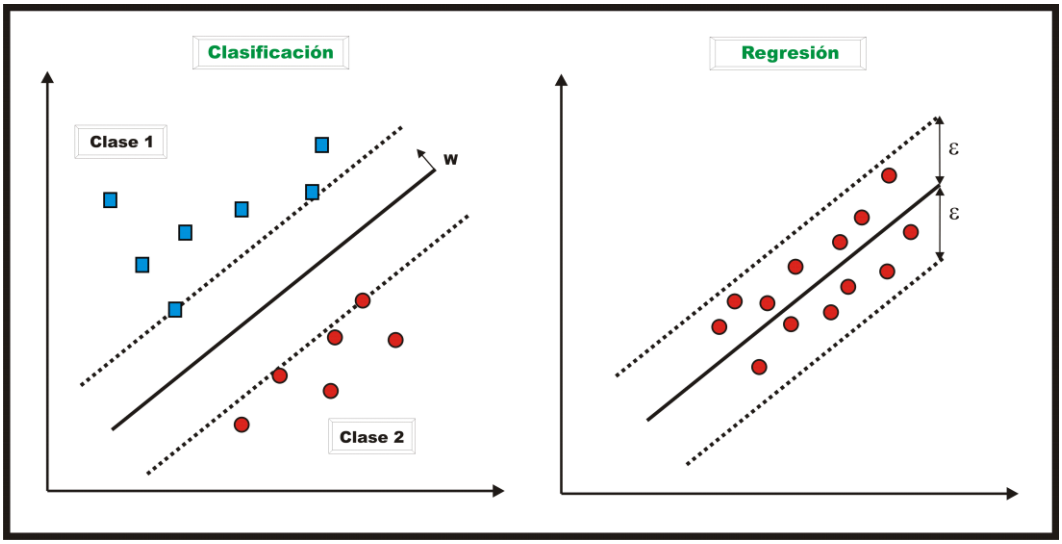


Figura 3.1: Esquemas de clasificación y regresión en SVM.

En el caso de la regresión, el objetivo es la estimación de una función, en lugar de predecir una etiqueta y_i = $\{\pm 1\}$ como era el caso de la clasificación. Para el sistema basado en regresión, y_i será considerado como una función dependiente de x_i, esta función podrá tomar cualquier valor real, a diferencia de lo que sucedía en clasificación donde sólo podía tomar los valores correspondientes a las etiquetas de las clases.

Tanto en clasificación como en regresión se obtiene un problema de Programación Cuadrática.

3.2 Espacios inducidos por la función kernel

Debido a las limitaciones computacionales de las máquinas de aprendizaje lineal, éstas no pueden ser utilizadas en la mayoría de las aplicaciones del mundo real. La representación por medio de una función Kernel ofrece una solución alternativa a este problema, proyectando la información a un espacio de características de mayor dimensión, el cual aumenta la capacidad computacional de las máquinas de aprendizaje lineal. La forma más común en que las máquinas de aprendizaje lineales aprenden una función objetivo es cambiando la representación de la función. Esto es similar a mapear el espacio de entradas X a un nuevo espacio de características $F = \{\varphi(x) \mid x \in X\}$. Esto es:

$$x = \{x_1, x_2, \ldots, x_n\} \rightarrow \varphi(x) = \{\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_n)\}$$

Los valores introducidos para describir la información original o atributos son conocidos como características, mientras que la selección de la mejor representación se conoce como selección de características.

En la *Figura 3.2* se muestra un mapeo de un espacio de entradas de dos dimensiones a un espacio de características de dos dimensiones. La información original no puede ser separada por una máquina lineal en el espacio de entradas, mientras que en el espacio de características esto resulta muy sencillo.

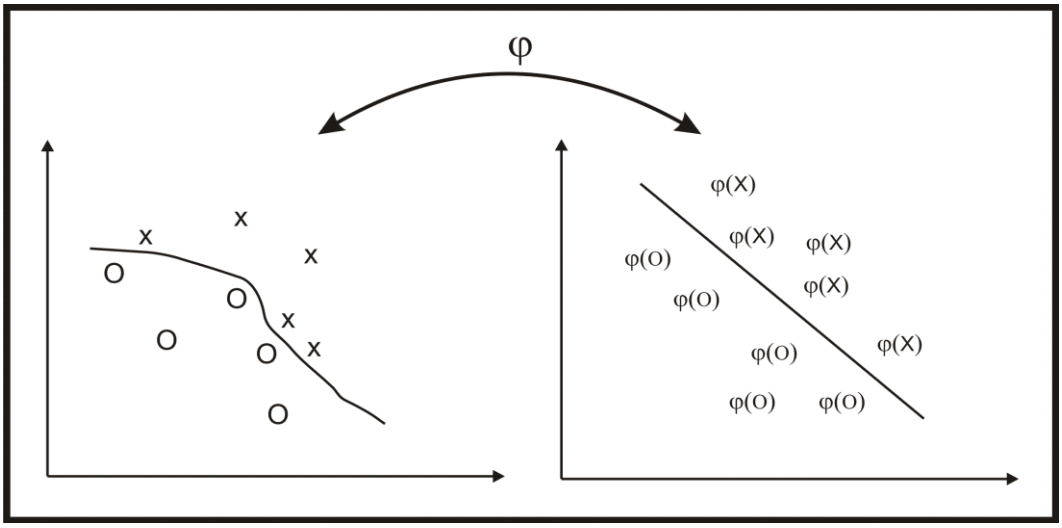


Figura 3.2: Mapeo del espacio de entrada a un espacio de características de mayor dimensión.

Las máquinas de aprendizaje lineales son funciones reales $f : X \in \Re^n \to Y \in \Re$. La función f se considera como una función lineal de $x \in X$, tal que se puede escribir como:

$$\begin{aligned} f(x) &= \langle w \cdot x \rangle + b \\ &= wx^T + b \\ &= \sum_{i=1}^{n} w_i x_i + b \end{aligned}$$

donde w es el vector de pesos y b es el bias, términos tomados de la literatura de redes neuronales. Las máquinas de este tipo admiten una representación dual, esto es, si se define $w = \sum \alpha_i y_i x_i$ se tiene que la función lineal se puede escribir en su forma dual como:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x_i \cdot x_i \rangle + b$$

donde $\langle \cdot \rangle$ es el producto interno. Una propiedad importante de la representación dual es que la información de entrenamiento entra a la función a través de las entradas de la matriz $G = \langle x_i \cdot x_i \rangle$. A fin de aprender relaciones no lineales con máquinas lineales, es necesario seleccionar un conjunto de características no lineales, con las cuales poder reescribir la información original en una nueva

representación. De ahí que el conjunto de hipótesis que se consideran son del tipo:

$$f(x) = \sum_{i=1}^{n} w_i \varphi_i(x) + b$$

donde $\varphi : X \rightarrow F$ es un mapeo no lineal que va del espacio de entradas a algún espacio de características. Debido a que las máquinas de aprendizaje lineal admiten una representación dual se pueden escribir las hipótesis como una combinación lineal de la información de entrada (x_i, y_i), de la siguiente manera:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \varphi_i^T(x) \cdot \varphi_i(x) \rangle + b$$

Por lo que sí se puede calcular el producto interno en el espacio de características como una función de las entradas, se estará realizando un mapeo del espacio de entradas al espacio de características, donde se realizará el aprendizaje con una máquina lineal.

Un Kernel K es una función, tal que para todo $x, z \in X$

$$K(x,z) = \langle \varphi(x) \cdot \varphi(z) \rangle = \sum_{i=1}^{l} \varphi_i^T(x) \varphi_i(z)$$

donde φ es un mapeo del espacio de entradas X al espacio de características F.

El uso de la función kernel hace posible realizar el mapeo de la información de entrada (x, z) al espacio de características ($\varphi(x)$, $\varphi_i(z)$) de forma implícita y entrenar a la máquina lineal en dicho espacio. La única información necesaria para el entrenamiento es la *matriz de Gram*. Dicha matriz también es conocida como la *matriz kernel,* la cual se denota con la letra K

$$K(x,z) = \left\langle \varphi(x)_i \cdot \varphi(z)_j \right\rangle_{i,j=1}^{l}$$

Una vez definida la matriz K, la hipótesis se puede calcular evaluando, al menos l veces, la matriz K de la siguiente manera:

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b$$

Utilizando la función kernel no es necesario calcular explícitamente el mapeo $\varphi : X \to F$ para aprender en el espacio de características.

Algunas de las propiedades que debe cumplir la función kernel para definir un espacio de características son las siguientes:

1. Simétrica

$$K(x,z) = \langle \varphi(x) \cdot \varphi(z) \rangle = \langle \varphi(z) \cdot \varphi(x) \rangle = \sum_{i=1}^{l} \varphi_i(x) \varphi_i(z) = K(z,x)$$

2. Desigualdad de Cauchy-Schwarz

$$K(x,z)^2 = \langle \varphi(x) \cdot \varphi(z) \rangle^2 \leq \|\varphi(x)\|^2 \|\varphi(z)\|^2 =$$
$$= \langle \varphi(x) \cdot \varphi(x) \rangle \langle \varphi(z) \cdot \varphi(z) \rangle =$$
$$= K(x,x) K(z,z)$$

Estas condiciones, no son suficientes para la existencia de un espacio de características. En [Bottou, 2007] se demuestra cómo el teorema de Mercer provee las condiciones que debe cumplir una función K para ser un kernel, el cual induzca un espacio de características.

Sean K_1 y K_2 kernels sobre XxX, $X \subseteq \Re^n$, $a \subset \Re^+$, con f una función real sobre *X*

$$\varphi : X \to \Re^m$$

con K_3 un kernel sobre $\Re^m x \Re^m$, B^{nxn} una matriz semidefinida positiva, entonces las siguientes funciones son kernels:

1. $K(x, z) = K_1(x, z) + K_2(x, z)$
2. $K(x, z) = aK_1(x, z),\ a \in \Re$
3. $K(x, z) = K_1(x, z)K_2(x, z)$
4. $K(x, z) = f(x)f(z)$
5. $K(x, z) = K_3(\varphi(x), \varphi(z))$
6. $K(x, z) = x^T Bz$

A continuación se dan algunas funciones que satisfacen las condiciones de Mercer:

- ***Lineal***

  Este kernel es una transformación lineal del tipo

  $$k(x,z) = \langle Ax \cdot Az \rangle = x^T A^T Az = x^T Bz$$

  Donde $B = A^T A$ es una matriz semidefinida positiva.

- ***Polinomial***

  El mapeo polinomial es un método muy popular para modelar funciones no lineales,

  $$K(x,x) = \langle x,x \rangle^d$$
  $$K(x,x) = (\langle x,x \rangle + c)^d$$

  Con $c \in \Re$, en la práctica se prefiere utilizar el segundo kernel.

- ***Funciones de Base Radial***

  $$K(x,z) = \exp(-\|x-z\|^2 / \sigma^2)$$

  Las funciones de base radial (RBF) son también conocidas como funciones Gaussianas.

3.3 Hiperplano de separación óptimo

Un hiperplano de separación es una función lineal que es capaz de separar los datos de entrenamiento sin error. Supongamos que los datos de entrenamiento consisten en n puntos $(x_1, y_1),...,(x_n, y_n)$, $x \in \Re^d, y \in \{+1,-1\}$ que pueden ser separados por la siguiente función:

$$D(x) = (w \cdot x) + w_0 \qquad \textbf{(Ecuación 3.1)}$$

con valores apropiados para w y w_0.

El hiperplano de separación debe cumplir las siguientes restricciones que definen la separación de los puntos del ejemplo:

$$(w \cdot x_i) + w_0 \geq +\Delta \qquad si \quad y_i = +1$$

$$(w \cdot x_i) + w_0 \leq -\Delta \qquad si \quad y_i = -1, i = 1,...,n$$

De la unión de ambas ecuaciones:

$$y_i[(w \cdot x_i) + w_0] \geq \Delta, \quad i = 1,...,n \qquad \textbf{(Ecuación 3.2)}$$

De esta última ecuación se deduce que todos los hiperplanos posibles de separación pueden expresarse en función de los datos de entrenamiento.

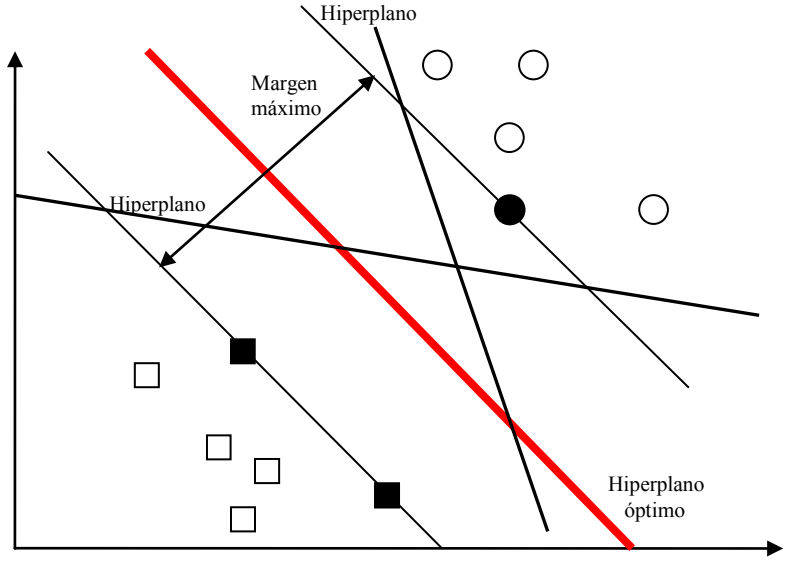


Figura 3.3: Hiperplano óptimo de separación.

La distancia desde el hiperplano de separación hasta el punto más cercano se denomina margen y será denotada por Δ. Un hiperplano se denomina óptimo si el margen Δ es máximo. La distancia entre el hiperplano de separación y un punto *x'* es $\left|D(x')\right|/\|w\|$, como se muestra en la *Figura 3.4*.

Si se supone que existe un margen Δ, todos los puntos de entrenamiento deben cumplir que:

$$\frac{y_k[(w\cdot x_k)+b]}{\|w\|} \geq \Delta, \qquad k = 1,...,n$$

donde $y_k \in \{-1,+1\}$

El problema de encontrar un hiperplano óptimo es igual que el de encontrar el w que maximice el margen Δ. Pero podría haber infinitas soluciones que pueden diferir sólo en la escala de w. Para limitar el número de soluciones se fija la escala con la siguiente ecuación:

$$\Delta\|w\| = 1$$

De esta ecuación se deduce que maximizar el margen Δ es equivalente a minimizar la norma de w.

Un hiperplano óptimo es aquel que cumple la *ecuación 3.2* y además minimiza:

$$\eta(w) = \|w\|^2$$

con respecto a w y w_0. Los puntos para los cuales la *ecuación 3.2* cumple la igualdad son los que están justo en el borde del margen y son llamados vectores soporte. En la *Figura 3.4* se muestra un ejemplo de distancias al hiperplano de separación de cada punto.

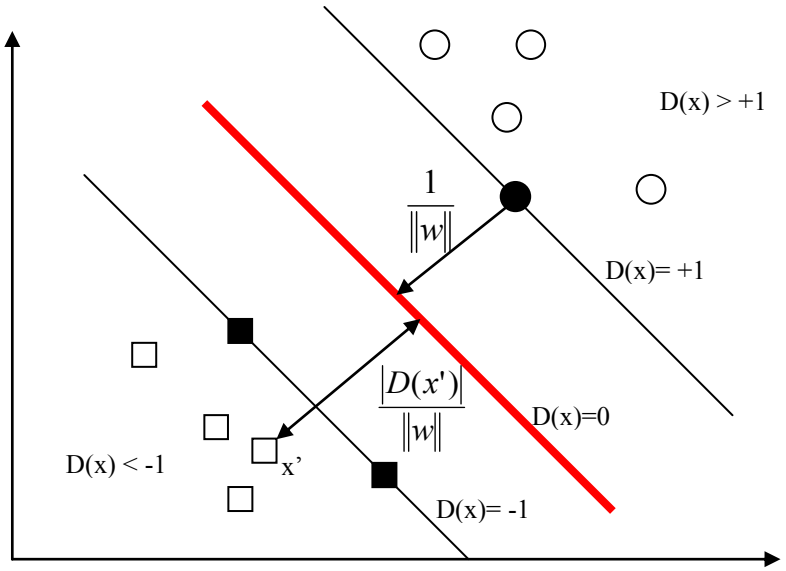


Figura 3.4: Distancia al hiperplano óptimo de separación.

Como los vectores soporte son los puntos que están más cercanos a la superficie de decisión, también serán los puntos más difíciles de clasificar y por tanto definen la superficie de clasificación. Más tarde se verá que la superficie de decisión se expresa en función de los vectores soporte.

La capacidad de generalización del hiperplano de separación óptimo se puede relacionar directamente con el número de vectores soporte. Según Vapnik el número de vectores soporte proporciona un límite para la expectativa de la tasa de error de una muestra

$$E_n[\%error] \leq \frac{E_n[numero\ de\ vectores\ soporte]}{n}$$

El operador E_n representa la expectativa sobre todos los conjuntos de tamaño n. Este límite es independiente de la dimensionalidad de los datos. Asumiendo que un hiperplano óptimo puede ser construido a partir de un número pequeño de puntos (si se compara con el número total de puntos de la muestra de entrenamiento), éste será una buena generalización incluso para dimensiones de los datos mucho más altas.

A continuación se va a calcular la dimensión VC (del inglés *Vapnik-Chervonenkis dimension*) que es una medida de la capacidad de los algoritmos

de clasificación estadística, definida como la cardinalidad del mayor conjunto de puntos que el algoritmo puede separar. Es un concepto fundamental en la teoría Vapnik-Chervonenkis, y fue originalmente definido por Vladimir Vapnik y Alexey Chervonenkis [Vapnik, 1971]. Vapnik proporciona un límite para la dimensión VC de un conjunto de puntos. Para las funciones que cumplen la *ecuación 3.2*, la dimensión VC está limitada por

$$h \leq \min(\frac{r^2}{\Delta^2}, d) + 1$$

donde *r* es el radio de la esfera más pequeña que contiene a todos los puntos de entrenamiento $(x_1, ..., x_n)$. De esto se deduce que es posible controlar la complejidad del hiperplano, independientemente de la dimensionalidad de los puntos. Encontrar un hiperplano óptimo para un conjunto de puntos separables es un problema de optimización cuadrática con restricciones lineales, como se verá a continuación. Determinar los w y b que minimicen la función:

$$\eta(w) = \frac{1}{2}\|w\|^2$$

sujeto a la restricción

$$y_i[(w \cdot x_i) + b] \geq 1, \quad i = 1, ..., n$$

dada por los puntos de entrenamiento (x_i, y_i), $i=1,...,n$ $x \in \Re^d$. La solución a este problema está formada por d+1 parámetros. Para datos de dimensión pequeña este problema puede ser resuelto con programación cuadrática (QP). Sin embargo, para datos de dimensión elevada no es práctico resolverlo de esa forma, no obstante, es posible traducir el problema a la representación dual para así obtener una solución.

La teoría de optimización afirma que un problema de optimización tiene una representación dual, si las funciones de coste y las restricciones asociadas tienen forma estrictamente convexa. En este caso resolver el problema dual es equivalente a resolver el problema original. El problema de optimización satisface estos criterios y por tanto posee una representación dual. En este caso el teorema de Karush-Kuhn-Tucker (KKT) se utiliza para traducir el problema original a su forma dual. Este problema dual posee también una interpretación

geométrica. Primero se determina una malla convexa para cada clase, esta malla será la más pequeña posible y que contenga todos los puntos de cada clase. Entonces el hiperplano óptimo es aquel que corta en dos la distancia mínima entre los dos puntos más cercanos de cada malla.

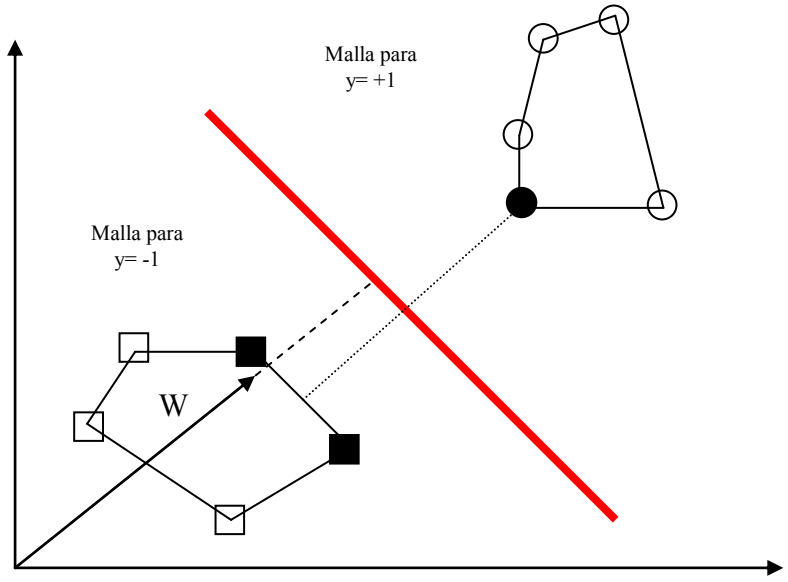


Figura 3.5: Malla convexa de las clases +1 y -1.

Para obtener la representación dual se deben realizar dos pasos, primero construir el problema de optimización sin restricciones usando los multiplicadores de Lagrange:

$$L(w,b,\alpha) = \frac{1}{2}(w \cdot w) - \sum_{i=1}^{n} \alpha_i \{y_i[(w \cdot x_i) + b] - 1\} \qquad \textit{(Ecuación 3.3)}$$

donde los α_i son los multiplicadores de Lagrange. El punto de silla de esta ecuación proporciona una solución al problema de optimización. La función debería ser minimizada con respecto a w y b, y maximizada con respecto a $\alpha_i \geq 0$.

El segundo paso es utilizar las condiciones *KKT* para expresar los parámetros w y b sólo en función de los parámetros α_i. Entonces solo se necesitaría maximizar la ecuación con respecto a los multiplicadores de

Lagrange. Según el teorema de Karush-Kuhn-Tucker las soluciones w^*, b^* y α^*, deberían satisfacer las siguientes condiciones:

$$\frac{\partial L(w^*,\alpha^*,b^*)}{\partial w} = 0$$
$$\frac{\partial L(w^*,\alpha^*,b^*)}{\partial b} = 0$$

Resolviendo las derivadas parciales se obtienen las siguientes propiedades para los hiperplanos óptimos:

1. Los coeficientes $\alpha_i^*, \quad i = 1,...,n$ deben cumplir que

$$\sum_{i=1}^{n} \alpha_i^* y_i = 0 \qquad \alpha_i^* \geq 0, \qquad i = 1,....,n \qquad \textbf{(Ecuación 3.4)}$$

2. El vector w es una combinación lineal de los vectores del conjunto de entrenamiento.

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i \qquad \alpha_i^* \geq 0, \qquad i = 1,....,n \qquad \textbf{(Ecuación 3.5)}$$

Los datos que cumplen la *ecuación 3.2* para el caso de la igualdad son los vectores soporte. Para construir el problema dual se van a sustituir los parámetros w y b en la *ecuación 3.3* y en la función de decisión que aparece en la *ecuación 3.1*. Para hacer más clara esta sustitución, primero se reescribe la *ecuación 3.3* de la siguiente forma:

$$L(w,b,\alpha) = \frac{1}{2}(w \cdot w) - \sum_{i=1}^{n} \alpha_i y_i x_i \cdot w - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i$$

Teniendo en cuenta la condición que refleja la *ecuación 3.4,* el tercer término será 0, por lo que haciendo las sustituciones oportunas se obtiene la función dual correspondiente al problema de optimización:

$$L(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_{i\cdot} \cdot x_j) + \sum_{i=1}^{n} \alpha_i$$

Esta ecuación debe ser maximizada con respecto a los multiplicadores α_i.

De esta forma la representación de la función del hiperplano óptimo tendría el siguiente aspecto:

$$D(x) = \sum_{i=1}^{n} \alpha_i^* y_i (x \cdot x_i) + b^*$$

El parámetro b^* se calcula teniendo en cuenta las propiedades de los vectores soporte. Sea (x_s, y_s) un vector soporte, entonces se cumple que:

$$y_s[(w \cdot x_s) + b^*] = 1$$

Y sustituyendo en la *ecuación 3.5* se obtiene:

$$b^* = y_s - \sum_{i=1}^{n} \alpha_i^* y_i (x_i \cdot x_s)$$

Con esto se ha completado la construcción del problema dual utilizando la formulación Langrangiana y las condiciones *KKT*. La definición de la forma dual del problema de optimización consiste en encontrar los parámetros α_i, i=1,….,n que maximicen la función:

$$L(\alpha) = -\frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_{i\cdot} \cdot x_j) + \sum_{i=1}^{n} \alpha_i$$

teniendo en cuenta las siguientes restricciones

$$\sum_{i=1}^{n} \alpha_i^* y_i = 0, \quad \alpha_i^* \geq 0, \qquad i = 1,...., n$$

Se puede observar que esta última formulación requiere sólo calcular los productos interiores de los puntos de entrenamiento. Esto se verá más adelante con un ejemplo numérico. En la práctica este problema de optimización puede ser resuelto usando métodos de programación cuadrática.

La formulación del hiperplano óptimo de separación parte de un supuesto en el que los datos son divisibles linealmente. Sin embargo, en muchos casos es

imposible separar linealmente los datos. En estos casos es necesario hacer un balance entre el objetivo de minimizar el riesgo (estrechando el margen) y obtener datos falsos (maximización del margen).

Para poder clasificar los datos no separables linealmente se debe permitir que algunos de los puntos de entrenamiento se encuentren dentro del margen y cuantificar el riesgo empírico usando la función de pérdida basada en el margen.

$$R_{emp}(w, Z_n) = \frac{1}{n}\sum_{i=1}^{n} L_{\Delta}(y_i, f(x_i, w))$$

El error basado en el margen indica la desviación con respecto a los límites del margen, la cual es representada con las variables de holgura:

$$\xi_i = \max(1 - y_i f(x_i, w)), \qquad i = 1,....,n\,.$$

En la *Figura 3.6* aparecen tres datos x_1, x_2 y x_3, cada uno de ellos no separable y que están dentro del margen. Además, los puntos x_2 y x_3 están mal clasificados porque están en el lado equivocado del margen. El punto x_1 representa un punto no separable pero que está bien clasificado.

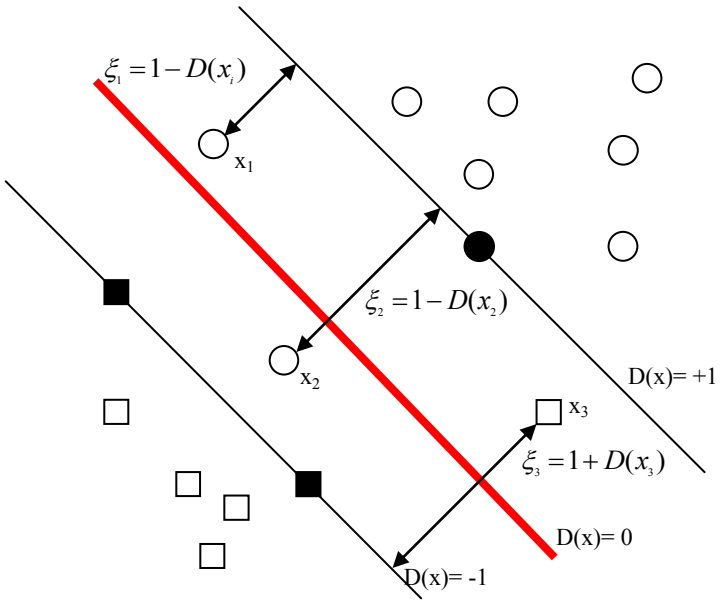


Figura 3.6: Variables holgura en función del margen de separación.

Al igual que todos los otros métodos de clasificación prácticos, SVM intenta obtener un valor del error de aproximación de una clasificación errónea. Teniendo en cuenta las variables de holgura ξ_i , ahora se debe encontrar un w y un b que cumplan lo siguiente:

$$\frac{C}{n}\sum_{i=1}^{n}\xi_i + \frac{1}{2}\|w\|^2 \qquad \textbf{(Ecuación 3.6)}$$

sujeto a la restricción

$$y_i[(w\cdot x_i)+b] \geq 1-\xi_i, \quad i=1,...,n$$

y dado un C suficientemente grande. De esta forma, el parámetro C controla el compromiso entre la complejidad y la proporción de los puntos no separables y debe ser elegido por el usuario. El valor de C especifica implícitamente el tamaño del margen, ya que el hiperplano de margen óptimo que satisface la *ecuación 3.6*, es un hiperplano con $\Delta = 1/\|w^*\|$

Este problema de optimización también puede ser traducido a su notación dual para ser resuelto con datos de gran dimensionalidad. Si se realiza el mismo procedimiento de traducción al dual que se ha usado anteriormente, la expresión dual sería:

$$L(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j (x_{i\cdot}\cdot x_j) + \sum_{i=1}^{n}\alpha_i$$

Buscando los parámetros α_i i=1,….,n que maximicen la función anterior teniendo en cuenta que:

$$\sum_{i=1}^{n}\alpha_i\, y_i = 0, \quad 0 \leq \alpha_i \leq C/n \qquad i=1,....,n$$

Como se puede ver, el problema de optimización difiere del caso separable sólo por la inclusión del límite máximo C/n en las restricciones.

3.4 Máquina de vectores soporte para clasificación

La máquina de vectores soporte para clasificación es construida teniendo en cuenta todo lo visto en las secciones anteriores. El kernel con producto interior es utilizado para mapear los datos a una dimensión mayor donde se encontrará un hiperplano de separación óptimo. Esto se corresponde con la sustitución de los productos interiores en las ecuaciones de optimización con los productos interiores de los kernels. Para la clasificación de datos no separables linealmente la función de decisión vendrá definida por:

$$f(x) = \sum_{i=1}^{n} \alpha_i^* y_i K(x_i, x) + b$$

Los parámetros $\alpha_i^*,\ i = 1,...,n$ son la solución del siguiente problema de optimización cuadrático. Maximizar la función:

$$L(\alpha) = -\frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_{i\cdot} \cdot x_j) + \sum_{i=1}^{n} \alpha_i$$

sujeto a las restricciones

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \le \alpha_i \le C/n \qquad i = 1,....,n$$

siendo $(x_i, y_i),\ i = 1,...,n$ los datos de entrenamiento, K un kernel y C el parámetro de regularización.

La formulación básica de SVM puede ser adaptada para problemas de múltiples clases, situaciones donde la distribución de datos futura se sabe que es diferente de los datos de entrenamiento y problemas con datos desiguales en las tasas de puntos no clasificados. Estas situaciones son comúnmente conocidas como casos de la vida real.

- Múltiples clases: Para aplicar SVM a un problema con más de dos clases lo que se suele hacer es descomponer el problema en sub-problemas de dos clases. Para un problema con N clases, se puede descomponer el problema en N sub-problemas de dos clases, el sub-problema binario J_i, siendo

i=1,….,N consistirá en tomar los puntos de la clase i, contra todos los demás puntos que no son de la clase i.

- Distribuciones desequilibradas y costes desiguales: Se puede modificar el coste funcional del SVM para tener en cuenta distribuciones desequilibradas y costes desiguales. Siguiendo [Lin, 2000] se introducen dos nuevos parámetros C (uno por cada clase) para tener en cuenta las diferentes probabilidades ante errores de clasificación y así

$$C^{+}\sum_{i\in+clase}\xi_i + C^{-}\sum_{j\in-clase}\xi_j + \frac{1}{2}\|w\|^2$$

donde C^{+} y C^{-} pueden ser calculadas utilizando la información sobre los costes de mala clasificación de los datos, de esta forma:

$$C^{+} = Coste\ \ (falsos\ \ negativos)\ \ p^{+}p_s^{-}$$

$$C^{-} = Coste\ \ (falsos\ \ positivos)\ \ p^{-}p_s^{+}$$

donde p^{+} y p^{-} son las probabilidades de los datos futuros, p_s^{+} y p_s^{-} son las probabilidades de los datos de entrenamiento. Teniendo en cuenta estas probabilidades, la formulación dual no cambia pero sí las restricciones, ya que debemos incorporar C^{+} y C^{-}

$$0 \le \alpha_i \le C^{+}, \quad i \in +clase$$
$$0 \le \alpha_j \le C^{-}, \quad j \in -clase$$

3.5 Ejemplo de utilización de SVM para el XOR (Or-exclusivo)

Este ejemplo demuestra el mecanismo de cálculo de SVM para un problema binario de separación lineal utilizando un kernel. Como se ve en la *Figura 3.7*, el conjunto de datos no es linealmente separable en dimensión dos, que es la dimensión de los puntos de entrenamiento. La función XOR tiene la siguiente representación en dos dimensiones:

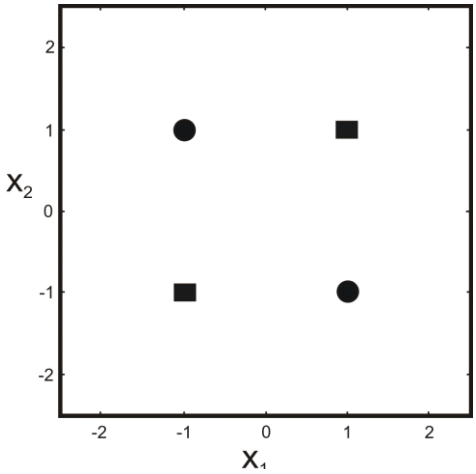


Figura 3.7: Representación de la función XOR**.**

Se utiliza una función kernel polinomial de grado dos para transformar los datos de entrada y así facilitar el cálculo del hiperplano óptimo de separación en un espacio de dimensión mayor. El kernel utilizado será el siguiente:

$$K(x,x') = [(x \cdot x') + 1]^2$$

que se corresponde con el grupo de funciones $z = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$, donde x_1 y x_2 representan las dos dimensiones de entrada. El vector z es un punto en el espacio de características de dimensión seis. Para determinar los límites de decisión dentro de este espacio se debe resolver la formulación dual del problema de optimización para $C = \infty$.

Maximizar la función

$$L(\alpha) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}\sum_{i,j=1}^{4} \alpha_i \alpha_j y_i y_j K_{ij}$$

Sujeta a las restricciones

$$\sum_{i=1}^{4} y_i \alpha_i = \alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 = 0,$$
$$0 \le \alpha_1, \quad 0 \le \alpha_2, \quad 0 \le \alpha_3, \quad 0 \le \alpha_4$$

El kernel se representa por una matriz de 4x4 elementos k_{ij} .

$$K = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$$

La solución al problema de optimización se obtiene con los valores $\alpha_1^* = \alpha_2^* = \alpha_3^* = \alpha_4^* = 0.125$, teniendo en cuenta que los cuatro puntos serán vectores soporte. La función L alcanza un máximo de 0.25 como solución. El margen se puede calcular sabiendo el máximo de la función L ya que:

$$2L(\alpha^*) = \|w\|^2 = 0.5 \text{ y como } \Delta = 1/\|w\| = \sqrt{2}$$

La *ecuación 3.5* se utiliza para resolver los parámetros $w_1^*, w_2^*, w_3^*, w_4^*, w_5^*$ y w_6^* en términos de α_i^* .

$$w^* = \sum_{i=1}^{4} \alpha_i^* y_i x_i = [0 \quad 0 \quad 0 \quad 1/2 \quad 0 \quad 0]$$

Resultando la función de decisión

$$D(x_1, x_2) = x_1 x_2$$

En la *Figura 3.8 (a)* se representan los datos de entrada para la función XOR y en la *Figura 3.8 (b)* el hiperplano de separación, así como los márgenes asociados en el espacio de características.

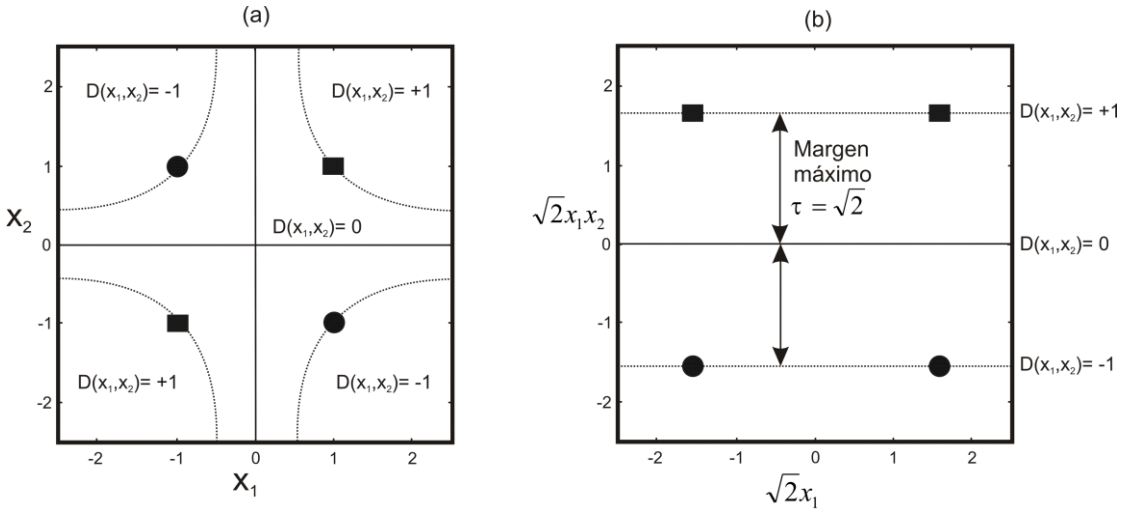


Figura 3.8: (a) Datos de entrada y clases asociadas. **(b)** Hiperplano de separación y márgenes asociados en el espacio de características.

CAPÍTULO 4

Implementación paralela de SVM

4.1 Introducción al Procesamiento Paralelo

Un computador paralelo puede ser definido como un conjunto de dos o más procesadores que, entre otras cualidades, ofrecen la posibilidad de trabajar cooperativamente para resolver diferentes tareas y problemas [Blazewicz, 2000]. Esta definición incluye desde redes de estaciones de trabajo hasta supercomputadores con cientos de procesadores, pasando por clústeres de computadores.

En la actualidad, la tendencia apunta hacia los clústeres de computadores, ya que dichas plataformas ofrecen una buena relación prestaciones/coste, además de interesantes características de re-configurabilidad, tolerancia a fallos, etc.

A la vez que se han diseñado nuevas arquitecturas paralelas, se han desarrollado interfaces de programación que facilitan las tareas de implementación en las mismas, como por ejemplo, las librerías para programación basadas en paso de mensajes o las herramientas para la programación multiproceso en arquitecturas de memoria compartida, [Grama, 2003]. La paralelización con estas herramientas es generada de forma casi íntegra por las decisiones del programador.

4.2 Arquitecturas Paralelas más utilizadas

En esta sección se ofrecen los conceptos básicos del procesamiento paralelo en computadores de altas prestaciones. En concreto se describen brevemente algunas de las arquitecturas paralelas más utilizadas, así como las métricas básicas utilizadas para evaluar el rendimiento de programas paralelos. En la *Figura 4.1* aparecen las dos arquitecturas paralelas más importantes, la arquitectura de memoria compartida y la arquitectura de memoria distribuida, aunque, como se verá más adelante, existen variantes que mezclan aspectos de estas dos arquitecturas básicas.

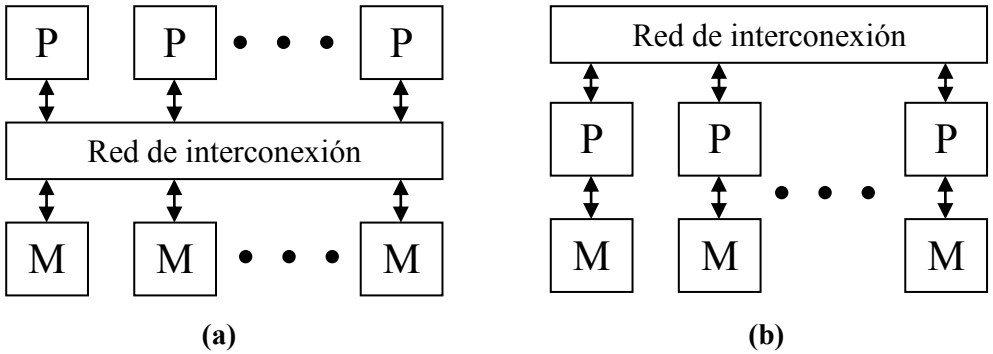


Figura 4.1: Arquitecturas paralelas: (a) Memoria compartida, (b) Memoria distribuida.

4.2.1 Multiprocesadores de Memoria Compartida

Los multiprocesadores de memoria compartida proporcionan un espacio de memoria único para todos los procesadores, simplificando la tarea de intercambiar datos entre los mismos. El acceso a memoria compartida ha sido implementado tradicionalmente usando una red de interconexión entre los procesadores y la memoria. A esta arquitectura se le conoce tradicionalmente como arquitectura UMA (Uniform Memory Access) [Hager, 2010].

La arquitectura paralela prevalente es el multiprocesador de pequeña o moderada escala que proporciona un espacio de direccionamiento global y acceso uniforme a toda la memoria principal desde cualquier procesador. A esta arquitectura se le denomina Multiprocesador Simétrico o SMP [Henessy, 2002], [Rey, 2011]. Cada procesador tiene su propia caché y todos los procesadores y

módulos de memoria están conectados a la misma red de interconexión, usualmente mediante un bus compartido.

Los SMP dominan el mercado de los servidores y son los más comunes en la actualidad. La compartición eficiente de los recursos, como la memoria y los procesadores, así como la posibilidad de acceder a todos los datos compartidos de forma eficiente desde cualquier procesador, usando operaciones de lectura y almacenamiento, hacen a este tipo de máquinas atractivas tanto para entornos multi-tarea como para la programación paralela.

El elemento de diseño clave de esta arquitectura es la organización de una jerarquía de memoria extendida. Sin embargo, las arquitecturas paralelas de este tipo requieren una adecuada gestión de la memoria, con el objetivo de evitar problemas de coherencia de caché. Este problema consiste en que cuando un bloque de memoria se encuentra en uno o más procesadores y otro distinto lo modifica, hay que evitar que el resto de procesadores accedan al contenido no actualizado existente en su memoria local. Esto conlleva que sea necesario aplicar técnicas de sincronización para controlar el acceso a las variables compartidas [Culler, 1998].

La programación paralela en este tipo de arquitectura se basa en el modelo *fork-join*, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (*fork*) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (*join)*. La interfaz de programación más utilizada en este caso es OpenMP, [OpenMP, 2013].

4.2.2 Multiprocesadores de Memoria Distribuida

Una forma de evitar los problemas de los multiprocesadores de memoria compartida consiste en hacer uso de multicomputadores. Los multicomputadores [Hwang, 1998] constan esencialmente de un conjunto de procesadores, cada uno conectado a su memoria local. Los procesadores se comunican mediante el paso de mensajes a través de una red de interconexión.

La programación de multicomputadores se realiza mediante librerías de paso de mensajes, en este paradigma de programación existen dos interfaces principales de programación de aplicaciones: MPI (Message Passing Interface) [Snir, 1996], y PVM (Parallel Virtual Machine) [Geist, 1994], siendo MPI el interface más utilizado en la actualidad. La complejidad de este modelo de programación radica en que el programador debe planificar la distribución de código y datos entre los diferentes procesadores de manera eficiente, estableciendo la secuencia correcta de envío de mensajes que necesiten otros procesadores. El código para cada procesador se carga en la memoria local al igual que cualquier dato que sea necesario. Los programas están divididos en diferentes partes, como en el sistema de memoria compartida, y dichas partes se ejecutan concurrentemente por procesadores individuales. Cuando los procesadores necesitan acceder a la información de otro procesador, o enviar información a otro procesador, en lugar de acceder a su memoria local, realizan las operaciones de lectura o escritura de la información mediante el envío de mensajes. La principal ventaja de esta arquitectura es que es directamente escalable y presenta un bajo coste para sistemas grandes.

4.2.3 Multiprocesadores de Memoria Compartida-Distribuida

Recientemente, los multiprocesadores de memoria compartida han seguido algunas de las tendencias establecidas previamente por los multicomputadores. En particular, la memoria ha sido distribuida físicamente entre los procesadores, reduciendo el tiempo de acceso a la memoria e incrementando la escalabilidad. A estos computadores paralelos se les denominan multiprocesadores de memoria compartida-distribuida (DSM) [Tanenbaum, 2009]. Los accesos a la memoria remota se realizan a través de una red de interconexión, de una manera similar al caso de los multicomputadores.

La diferencia principal entre los DSMs y los multicomputadores es que los mensajes se inician en respuesta a accesos a la memoria en vez de llamadas al sistema. Para reducir la latencia de la memoria, cada procesador tiene varios niveles de memoria caché. Esta arquitectura proporciona un tiempo no uniforme de acceso a la memoria (NUMA) [Hagersten, 1990], el cual se debe

principalmente a la diferencia entre los tiempos de acceso a las cachés y la memoria principal, más que al diferente tiempo de acceso entre las memorias locales y remotas.

.

4.3 Medidas de Rendimiento en Algoritmos Paralelos

En esta sección se describen brevemente las métricas más utilizadas a la hora de determinar el rendimiento de un algoritmo paralelo [Hwang, 1998]: la ganancia de velocidad, la eficiencia y la escalabilidad.

4.3.1 Ganancia de Velocidad o Speedup

La ganancia de velocidad (Speedup) se define como:

$$Speedup\ (I,P) = \frac{T(I,1)}{T(I,P)}$$

donde P denota el número de procesadores elementales (homogéneos) de la máquina paralela, $T(I,1)$ el tiempo de ejecución del algoritmo en un solo procesador, y $T(I,P)$ el tiempo de ejecución para un algoritmo que utiliza P procesadores.

El objetivo del procesamiento paralelo es reducir el tiempo para generar la solución de un problema determinado. Se denomina *ganancia de velocidad lineal* al caso ideal en el que P procesadores obtengan la solución secuencial en un tiempo P veces inferior al requerido por un solo procesador, es decir:

$$Speedup\ (I,P) = \frac{T(I,1)}{T(I,P)} = \mathrm{P} > 1$$

Sin embargo, en la práctica resulta difícil obtener dicha ganancia de velocidad lineal, y en la mayoría de los casos el valor de $T(I,P)*P$ suele ser superior al valor de $T(I,1)$, debido a las penalizaciones relacionadas con la sincronización y las comunicaciones. No obstante, existen múltiples referencias en las que se muestran valores de la aceleración para algunos programas,

[Mechoso, 1994], [Nagashima, 1995] por encima del número de procesadores, lo que se denomina ganancia de velocidad super-lineal, es decir:

$$Speedup\ (I,P) = \frac{T(I,1)}{T(I,P)} > P > 1$$

La razón de esta super-linealidad puede venir motivada por diversos factores, entre los que destacan:

- Incremento de los recursos en los sistemas paralelos. Los programas paralelos que exploten el aumento de los recursos de las máquinas paralelas pueden obtener valores de aceleración por encima de los lineales gracias, no sólo a la mayor capacidad de procesamiento en términos de número de procesadores, sino además, por el incremento de otros recursos, tales como la capacidad de la memoria principal y de la memoria caché.

- Características propias de los algoritmos paralelos. Algunos algoritmos paralelos evolucionan de forma distinta dependiendo del número de procesadores. Principalmente, esto ocurre en situaciones donde interviene de alguna forma la aleatoriedad.

No obstante, también es conveniente indicar que en muchos casos el uso del paralelismo no ofrece una compensación satisfactoria. Este es el caso en el que el programa paralelizado sea inherentemente secuencial, lo cual provoca que las porciones de código paralelizable sean tan reducidas que su paralelización resulta ineficiente. Este problema viene determinado por la conocida Ley de Amdahl [Amdahl, 1967], que dice que todo programa paralelo tiene una parte secuencial que eventualmente limita la aceleración que se puede alcanzar en una plataforma paralela. Por ejemplo, si el componente secuencial de un algoritmo es $1/s$ de su tiempo de ejecución, entonces la aceleración máxima posible que se puede alcanzar en el computador paralelo es s.

El tiempo para realizar el cómputo con P procesadores es:

$$T_P = s * t_s + \frac{(1-s)t_s}{P}$$

donde t_s es el tiempo de ejecución secuencial.

La aceleración (Speedup) puede ser redefinida de la siguiente forma:

$$Aceleración = \frac{1}{s + \frac{(1-s)}{P}} = \frac{P}{P * s + 1 - s} = \frac{P}{1 + (P-1) * s}$$

$$\frac{P}{1 + (P-1) * s} \to \frac{1}{s} \text{ cuando } P \to \infty$$

Según la ley de Amdahl, si un programa tiene un 5% de componente secuencial entonces la aceleración máxima que se puede alcanzar es de 20. La ley de Amdahl tiene varias implicaciones:

- Para una carga dada, la aceleración máxima tiene una cota superior de $1/s$. Al aumentar s, la aceleración decrecerá proporcionalmente.

- Para alcanzar buenas aceleraciones, es importante reducir s.

4.3.2 Eficiencia del sistema

Otra medida de rendimiento habitualmente utilizada es la *eficiencia del sistema* [Lee, 1980], que representa la ganancia de velocidad por procesador. Así, para un sistema con *P* procesadores viene definida por:

$$E(P) = \frac{S(P)}{P} = \frac{T(1)}{P * T(P)}$$

La eficiencia es una comparación del grado de *Speedup* conseguido frente al valor máximo. Dado que $1 \leq S(P) \leq P$, tenemos $1/P \leq E(P) \leq 1$. La eficiencia más baja $(E(P) \to 0)$ corresponde al caso en que todo el programa se ejecuta en un único procesador de forma serie. La eficiencia máxima $(E(P) = 1)$ se obtiene cuando todos los procesadores están siendo completamente utilizados durante todo el periodo de ejecución.

4.3.3 Escalabilidad

Por último, se define la *escalabilidad:* una aplicación paralela se dice que es *escalable* para un determinado rango de procesadores [1..P], si la eficiencia $E(P)$ del sistema se mantiene constante y cercana a la unidad en todo ese rango. Normalmente todas las aplicaciones paralelas tienen un determinado número de procesadores a partir del cual la eficiencia empieza a disminuir de forma más o menos brusca. Una aplicación es más escalable que otra si este número de procesadores, a partir del cual la eficiencia disminuye, es mayor.

4.4 Implementación paralela de LIBSVM

Implementar un algoritmo de aprendizaje SVM requiere resolver un problema de programación cuadrática (QP). Existen algoritmos de propósito general que se pueden utilizar para resolver problemas QP, en esta sección se verán algunos de ellos.

4.4.1 Implementaciones de máquinas de vectores soporte

En función del volumen de datos de entrada disponibles existen diferentes herramientas a la hora de crear modelos de aprendizaje con máquinas de vectores soporte. Por ejemplo, para muestras pequeñas (alrededor de 1000 puntos) un método quasi-Newton, como es el MINOS [Murtagh, 1978], o métodos como LOQO [Vanderbei, 1999] son aplicables. La ventaja de estos métodos es que poseen gran precisión, pero su inconveniente es que no se pueden utilizar si la matriz kernel no puede ser alojada en la memoria. Para resolver problemas de gran tamaño existen otros tipos de implementaciones especialmente adaptadas a las características de un problema SVM. Estas aplicaciones se pueden clasificar en las siguientes tres categorías:

- *Métodos de selección de subconjuntos*: Estos métodos sacrifican algo de precisión en la solución para dividir el problema de optimización en trozos que sean más “manejables”. Una optimización de este tipo es la llamada optimización de `chunking` [Osuna, 1997], basada en que sólo los vectores soporte contribuyen a la solución final y el resto de puntos no son importantes. Así que, en este método se elige un subconjunto

arbitrario de puntos para generar una solución SVM utilizando un algoritmo QP de propósito general. Una vez hecho esto, sólo se utilizan los vectores soporte y el resto se descartan. A continuación, se vuelven a añadir puntos hasta completar el subconjunto y se vuelve a aplicar el algoritmo QP de propósito general.

Este proceso se repite hasta que los vectores soporte no cambian de un cálculo a otro, en caso de que no exista variación, entonces los vectores soporte de esa última solución son los vectores soporte del conjunto de datos completo.

Una alternativa a este método es la llamada `descomposición`. En esta variante del método los datos se dividen en subconjuntos de tamaño fijo llamados "conjuntos de trabajo". La optimización se realiza en cada subconjunto, manteniendo los parámetros fijos en el resto. Los paquetes que realizan este tipo de implementación son SVMLight [Joachims, 2008] y SVMTorch [Collobert, 2001]. El algoritmo de optimización mínima secuencial (SMO, Sequential Minimal Optimization) desarrollado por Platt [Platt, 1998], es una forma extrema de descomposición donde los conjuntos de trabajo están formados por sólo dos puntos. La ventaja de SMO sobre el resto es que este algoritmo posee una gran escalabilidad y reduce la cantidad de memoria necesaria comparado con el método de `chunking`. La aplicación que está basada en SMO es LIBSVM [Rong-En, 2005], que implementa una variante de SMO para clasificación y regresión.

- *Métodos iterativos*: El método del Gradiente Descendiente puede ser aplicado a un problema básico de optimización SVM obteniendo un algoritmo iterativo. La principal ventaja de los métodos iterativos es que utilizan muy pocos pasos y son fáciles de implementar. La desventaja es que son más lentos que los algoritmos de resolución de problemas QP.

- *Formulaciones alternativas de SVM*: modificando el problema de aprendizaje es posible simplificar el resultado del problema de optimización. Esto implica simplificar o reducir el número de restricciones

modificando el error funcional o penalización. Por ejemplo, una aproximación llamada Lagrangian SVM (LSVM) [Mangasarian, 2001] utiliza esta formulación de aprendizaje, donde los resultados en un problema de optimización dependen de la resolución de un sistema lineal de inecuaciones. La ventaja de estas transformaciones es que aumenta considerablemente la velocidad de resolución del problema, incluso con una gran cantidad de datos de entrenamiento. El inconveniente es que aún no está demostrado que estas modificaciones afecten a la generalización del algoritmo de optimización.

4.4.2 Elección de LIBSVM

LIBSVM es una librería de desarrollo de Support Vector Machines desarrollada por Chih-Chung Chang y Chih-Jen Lin en 2001, [Chih-Chung, 2012]. Las principales razones de la elección del software LIBSVM es que está abierto a nuevos desarrollos y por otro lado, que el algoritmo utilizado para resolver la formulación matemática de SVM es el Sequential Minimal Optimization (SMO), al que se le ha añadido el método de segundo orden que aparece en la *Figura 4.2* para la fase de selección del conjunto de trabajo, consiguiendo una convergencia más rápida en la búsqueda del elemento j.

Primer orden:
$$i := \arg \max_{k=lo...hi} \begin{cases} -G_k : y_k > 0 \,\&\, \alpha_k < C \\ +G_k : y_k < 0 \,\&\, \alpha_k > 0 \end{cases}$$

Primer orden:
$$j := \arg \max_{k=lo...hi} \begin{cases} +G_k : y_k > 0 \,\&\, \alpha_k > 0 \\ -G_k : y_k < 0 \,\&\, \alpha_k < C \end{cases}$$

Segundo orden:
$$j := \arg \max_{k=lo...hi} \begin{cases} b^2/\eta : y_k > 0 \,\&\, \alpha_k > 0 \,\&\, b > 0 \\ b^2/\eta : y_k < 0 \,\&\, \alpha_k < C \,\&\, b > 0 \end{cases}$$

$$donde: \quad \begin{matrix} b = y_i G_i + y_k G_k \\ \eta = K_{i,i} + K_{k,k} - 2K_{i,k} \end{matrix}$$

Figura 4.2: Algoritmos para la selección del conjunto de trabajo.

El algoritmo comienza con una solución $\alpha = 0, G = -1$ y realiza los pasos de búsqueda de una solución utilizando la técnica del gradiente descendente. En

cada paso se elige un conjunto de trabajo, en concreto un par de puntos (*i, j*) que violan de forma más elevada las condiciones *KKT* (Karush-Kuhn-Tucker) que aparecen en la *Figura 4.2*. Para cada pareja de puntos se actualiza $\Delta\alpha$ y también el gradiente del vector. El algoritmo se repite hasta la convergencia. El proceso iterativo se detiene cuando las condiciones *KKT* son violadas con un error menor que una determinada cantidad fijada por el usuario.

En la parte derecha de la *Figura 4.3* se presentan esquemáticamente los principales bloques de datos: datos de entrenamiento, matriz kernel, valores del gradiente y valores de alpha. También se indica el patrón de acceso para un paso de la optimización: se seleccionan dos vectores de entrenamiento x_i, x_j y se calculan dos columnas de la matriz kernel que son necesarias para actualizar el vector de gradiente, siendo ésta la operación principal del algoritmo.

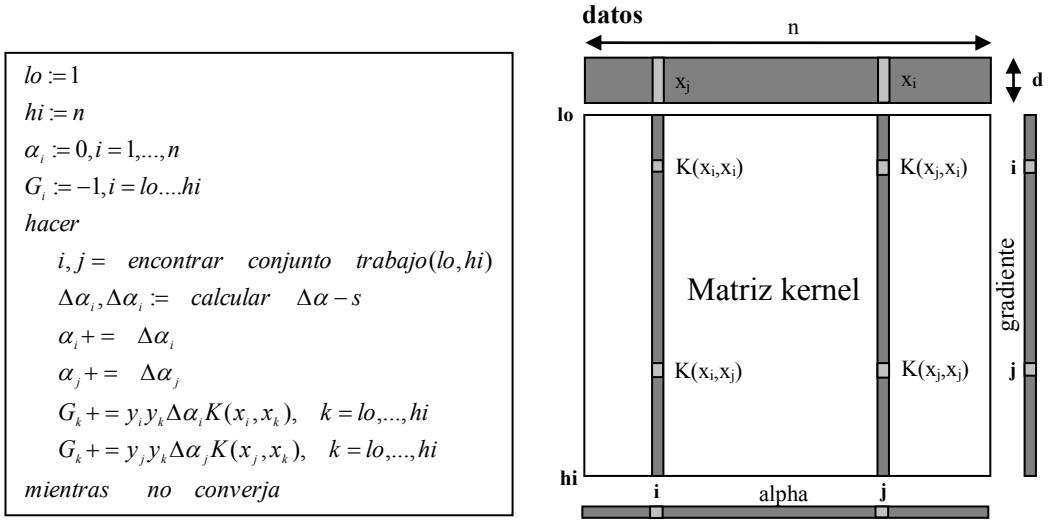


Figura 4.3: Pseudocódigo de optimización de SVM y esquema de bloques de datos utilizados.

El algoritmo ha sido expresado en función de (*lo, hi*) para hacer luego más fácil su conversión a la versión paralela.

Para calcular una columna del kernel hay que realizar una multiplicación de vectores $K(x_i,*)=x_i*x$, para lo que se necesitan $(hi-lo)*d$ operaciones, siendo d la dimensión de los datos de entrada. Aquí se supone que es posible calcular el producto escalar de dos vectores, éste es el caso de los kernel utilizados por LIBSVM: lineal, polinomial, o RBF (Radial Basis Function). Un análisis del algoritmo revela un alto grado de independencia en los datos. Todos

los bucles pueden ser paralelizados dividiéndolos entre múltiples máquinas y ajustando los valores de *hi* y *lo* adecuadamente. La principal dependencia de datos que contiene el algoritmo se produce en el cálculo del conjunto de trabajo. Para una buena convergencia es necesario obtener el conjunto de trabajo que más se aleja de las condiciones *KKT*, teniendo en cuenta todos los datos de entrenamiento.

4.4.3 Optimizaciones de LIBSVM secuencial

Antes de ver detalladamente la versión paralela del código de LIBSVM, se van a describir algunas modificaciones previas que se han realizado en el código secuencial original.

La primera de ellas es la reordenación de los datos, debido a que normalmente los datos de entrenamiento se reciben desordenados, para evitar continuos saltos dentro de la memoria durante la búsqueda del conjunto de trabajo, se han ordenado los datos según la clase a la que pertenezcan, tal y como se puede apreciar en la *Figura 4.4*. Esto también, como se verá más adelante, simplifica el código, ya que evita realizar saltos innecesarios cuando se recorre la matriz kernel.

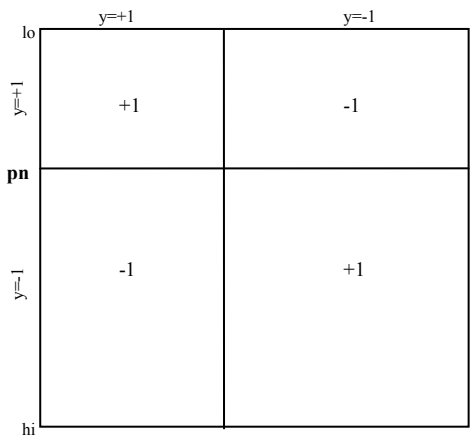


Figura 4.4: Organización de la matriz kernel.

Otra modificación que se ha realizado en el código original de LIBSVM es la anulación de la técnica `shrink`, que intenta reducir el tamaño de los datos almacenados en memoria.

La técnica de `shrink` [Chih-Chung, 2012] consiste básicamente en identificar en la fase inicial del algoritmo de SVM los puntos que nunca serán seleccionados como vectores soporte y eliminarlos del conjunto de datos de entrada para reducir el tamaño del problema a resolver. Con el fin de no complicar la estructura de la caché se ha desactivado esta opción, así todos los bloques de la caché tendrán el mismo tamaño.

Respecto a la caché que proporciona LIBSVM, se ha dejado la posibilidad de elegir el tamaño de la caché mediante una opción de entrada a la aplicación, pero se han realizado modificaciones que posibiliten que cada vez que se accede a un elemento de la matriz kernel Q que no está en la caché, no se realice el cálculo de todos los elementos correspondientes a esa fila. En el apartado de código de la aplicación se detallará esta modificación, que reduce drásticamente las operaciones en cada paso de la búsqueda del hiperplano óptimo de separación.

4.4.4 SVM en cascada

Utilizando este método se intenta paralelizar la herramienta LIBSVM mediante un refinamiento progresivo del conjunto de vectores de entrada. Como se mencionó anteriormente cuando se hacía referencia a la técnica de `shrink`, una forma de acelerar el rendimiento de una implementación de SVM es eliminar en una fase temprana de la implementación, los puntos dentro del conjunto de entrenamiento que nunca serán vectores soporte de la solución buscada. La técnica en cascada, cuya estructura se muestra en la *Figura 4.5,* realiza este tipo de filtrado mediante distintos niveles de SVM.

La técnica consiste, básicamente, en dividir los datos de entrenamiento entre un número *n* de máquinas, donde cada máquina evalúa individualmente el trozo del conjunto de datos de entrada que le ha sido asignado. A la salida de cada máquina se obtienen un conjunto de vectores soporte que son combinados dos a dos e introducidos en una nueva máquina SVM, así sucesivamente hasta que en el nivel más alto se obtiene una primera solución. En el esquema de la *Figura 4.5* la primera solución sería la obtenida en *SV15*. Esta solución es

realimentada a la entrada del primer nivel y combinada de nuevo con los trozos del conjunto de datos de entrada.

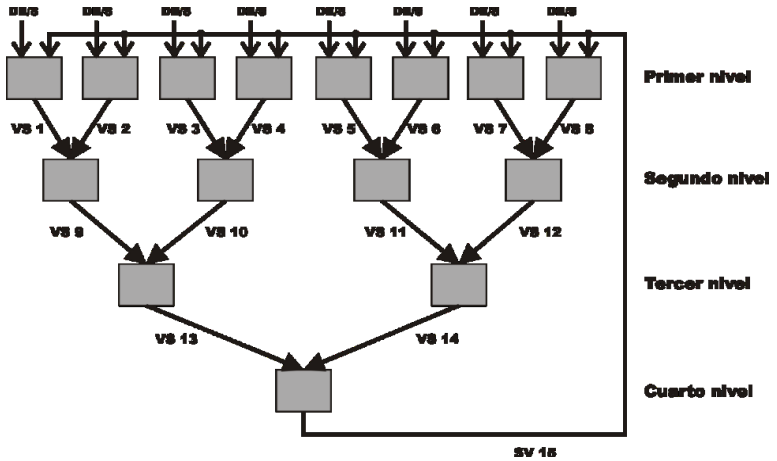


Figura 4.5: Esquema del algoritmo de SVM en cascada.

El proceso continúa iterativamente hasta que los vectores soporte obtenidos en *SV15* son los mismos que los obtenidos tras la salida del primer nivel una vez combinados con los trozos asignados a cada máquina. En el caso de que las salidas *VS1, VS2,…, VS8* sean iguales a *SV15* significará que se tiene un conjunto de vectores soporte que clasifican perfectamente los datos de entrada.

En [Graf, 2005] se demuestra que existe un número de pasos finito para obtener una solución siguiendo el método en cascada. La idea fundamental del algoritmo en cascada es que de cada trozo del conjunto de datos que se pasa en el primer nivel a las máquinas SVM, se obtienen un grupo de vectores soporte y un conjunto de puntos que son descartados. Los vectores soporte obtenidos son una solución válida para ese trozo del conjunto de datos de entrada, luego algunos de ellos formarán parte de la solución global. Siguiendo ese mismo razonamiento, el grupo de puntos descartados, son puntos interiores que no son vectores soporte de la solución parcial de esa máquina, luego tampoco lo serán de la solución global que agrupa a todos los puntos de entrada.

En la *Figura 4.6* se hace una evolución intuitiva del proceso de filtrado al que se someten los datos de entrada en cada nivel del algoritmo en cascada. Se parte de dos subconjuntos disjuntos de datos que son optimizados cada uno de

ellos de forma individual. En la gráfica de la izquierda aparecen con relleno los elementos del primer subconjunto, además de los vectores soporte resultantes que se muestran con un marco exterior. En la gráfica central aparecen, siguiendo la misma notación, los elementos del segundo subconjunto. Finalmente, en la gráfica de la izquierda se muestra la solución del conjunto completo (línea continua) una vez optimizados los vectores soporte de los dos subconjuntos, se aprecia que es muy similar a la solución obtenida si se hubiera optimizado el conjunto de datos completo sin realizar la división de subconjuntos (línea punteada).

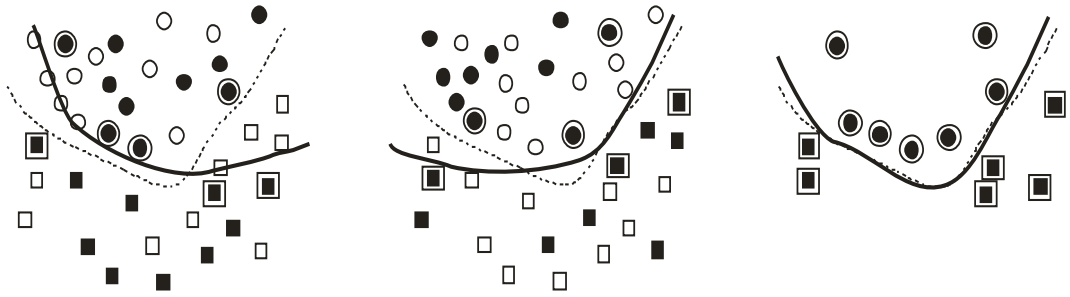
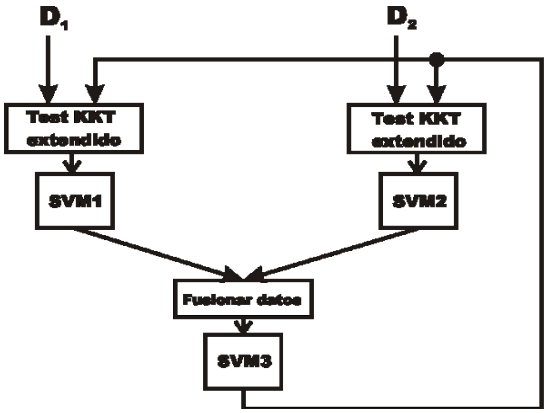


Figura 4.6: Proceso de filtrado del SVM en cascada.

En la *Figura 4.7* se representa de una forma más detallada el esquema del algoritmo en cascada para el caso de dos máquinas SVM. Se observa en la entrada de los datos a *SVM1* y *SVM2* un proceso previo de test *KKT* que se realiza cuando se ha obtenido la primera solución en *SVM3*, en el caso de que los vectores soporte obtenidos de *SVM3* combinados con los datos de entrada D_1 y D_2 estén dentro de las condiciones *KKT* el algoritmo parará, en caso contrario se continuará una iteración más.



$$W_i = -\frac{1}{2}\vec{\alpha}_i^T Q_i \alpha_i + \vec{e}_i^T \vec{\alpha}_i$$

$$\vec{G}_i = -\vec{\alpha}_i^T Q_i + \vec{e}_i$$

Figura 4.7: Algoritmo de SVM en cascada para dos conjuntos de datos de entrada.

Las fórmulas que describen las operaciones que se realizan con la matriz kernel y la fórmula que define la función objetivo aparecen en la parte derecha de la *Figura 4.7,* donde G_i es la función gradiente, W_i es la función objetivo, Q_i es la matriz kernel y e_i es un vector con todos los valores a 1. Asimismo, cuando se combinan los datos de salida de *SVM1* y *SVM2,* se deben seguir las fórmulas que aparecen en la *ecuación 4.1*

$$W_3 = -\frac{1}{2}\begin{bmatrix}\vec{\alpha}_1\\ \vec{\alpha}_2\end{bmatrix}^T\begin{bmatrix}Q_1 & Q_{12}\\ Q_{21} & Q_2\end{bmatrix}\begin{bmatrix}\vec{\alpha}_1\\ \vec{\alpha}_2\end{bmatrix}+\begin{bmatrix}\vec{e}_1\\ \vec{e}_2\end{bmatrix}^T\begin{bmatrix}\vec{\alpha}_1\\ \vec{\alpha}_2\end{bmatrix}$$

$$\vec{G}_3 = -\begin{bmatrix}\vec{\alpha}_1\\ \vec{\alpha}_2\end{bmatrix}^T\begin{bmatrix}Q_1 & Q_{12}\\ Q_{21} & Q_2\end{bmatrix}+\begin{bmatrix}\vec{e}_1\\ \vec{e}_2\end{bmatrix} \qquad \textbf{(Ecuación 4.1)}$$

Estas ecuaciones plantean la combinación de dos conjuntos de vectores soporte a la entrada de una nueva máquina SVM. Como resultado se obtiene la unión de las matrices kernel correspondientes a los vectores soporte, y además, se deben utilizar los vectores $\vec{\alpha}$ y $\vec{G}$ obtenidos de *SVM1* y *SVM2* como entrada para *SVM3,* lo cual ahorra bastante tiempo de cálculo en la convergencia hacia la solución final.

En [Graf, 2005] además se profundiza en las posibles formas de inicializar los valores de $\vec{\alpha}$ según los datos de la matriz kernel *Q*, aunque en la implementación realizada no se han tenido en cuenta estos parámetros de inicialización, ya que no influyen en la velocidad de obtención de la solución.

4.4.5 Esquema general de optimización paralela SPREAD-KERNEL

Como se señaló en la *Sección 4.4.2*, el algoritmo descrito en la *Figura 4.3,* presenta un alto grado de independencia en los datos, es decir, se puede ejecutar una etapa sin saber cuál es el resultado de la etapa anterior. Esta independencia puede ser aprovechada para obtener un diseño paralelo del algoritmo. Todos los bucles secuenciales en el algoritmo pueden ser ejecutados de forma paralela en distintas máquinas para rangos $(lo..hi)$ no superpuestos.

En la versión que se ha implementado, cada proceso tiene asignado el cálculo de una parte de la matriz kernel, desde lo hasta hi más los gradientes correspondientes dentro del mismo rango. Cada proceso debe contener todos los datos más todos los valores de alpha.

El único cálculo que no es independiente es la búsqueda global del conjunto de trabajo óptimo. Cada proceso busca localmente dentro de su rango $(lo,\ hi)$ el conjunto de trabajo (i_q, j_q), pero todos estos conjuntos de trabajo locales deben combinarse para obtener el conjunto de trabajo global. Esto se consigue con la comunicación `half-duplex` y *`multicast`* de todas las máquinas, tal y como se explica más adelante en el apartado tráfico de red.

La *Figura 4.8* muestra el pseudocódigo del algoritmo implementado siguiendo [Bottou, 2007]. La única diferencia con su versión secuencial es la selección del conjunto de trabajo.

En la versión paralela, cada proceso (indexado por q) trabaja sólo con su propia porción $(lo(q)....hi(q))$ del vector gradiente $\vec{G}$, para el cual es necesario calcular el trozo de la matriz kernel asociada $(lo(q),....hi(q))$. El coste de calcular una columna del kernel es proporcional a la dimensión de los datos y al tamaño del trozo seleccionado d*($hi(q)$-$lo(q)$).

1. $lo := 1 + q * [n / p]$
2. $hi := \min(n, lo + \lceil n / p \rceil$
3. $\alpha_i := 0, i = 1,...,n$
4. $G_i := -1, i = lo....hi$
5. *hacer*
6. $\quad i_q, j_q = encontrar\ \ conjunto\ \ trabajo(lo, hi)$
7. $\quad i\ , j\ = \max\ \ conjunto\ \ trabajo(lo, hi)$
8. $\quad \Delta\alpha_i, \Delta\alpha_j := \ \ calcular\ \ \Delta\alpha - s$
9. $\quad \alpha_i += \ \ \Delta\alpha_i$
10. $\quad \alpha_j += \ \ \Delta\alpha_j$
11. $\quad G_k += y_i y_k \Delta\alpha_i K(x_i, x_k), \quad k = lo,...,hi$
12. $\quad G_k += y_j y_k \Delta\alpha_j K(x_j, x_k), \quad k = lo,...,hi$
13. *mientras no converja*

Figura 4.8: Pseudocódigo de optimización de SVM con spread-kernel.

También se debe recordar que cada proceso contiene el vector $\vec{\alpha}$ completo, pero actualizar este vector es algo trivial que se hace en cada máquina y aunque se repite este cálculo, esto no afecta al rendimiento.

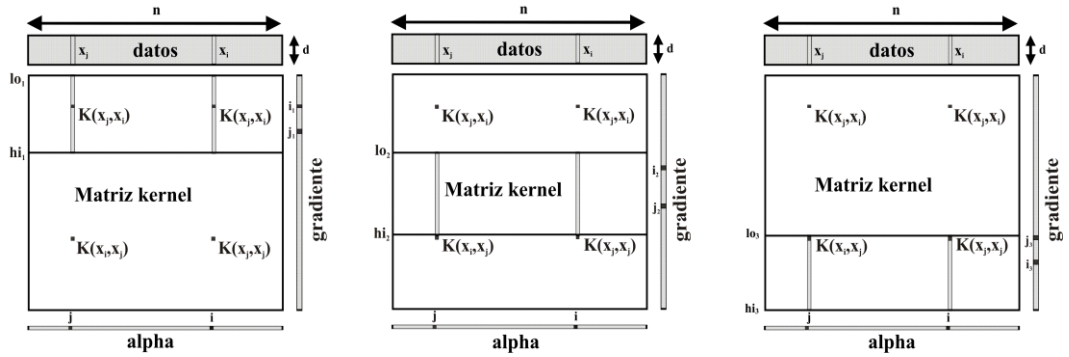


Figura 4.9: Esquema del reparto de datos entre los distintos procesos**.**

Tráfico de red

Para encontrar el conjunto de trabajo óptimo global es necesario mezclar todos los conjuntos de trabajo que ha calculado cada máquina. El propósito es encontrar el conjunto de trabajo con el mayor gradiente de segundo orden. El algoritmo utilizado en este caso ha sido `half-duplex` más *`multicast`*, que organiza las máquinas con una estructura de árbol y los máximos de cada máquina van ascendiendo después de ser comparados con las máquinas situadas más a la derecha de ellas. Como se puede ver en la *Figura 4.10*, una vez que se ha calculado el máximo global se hace un envío por *`multicast`* a todos los nodos para comenzar una nueva iteración.

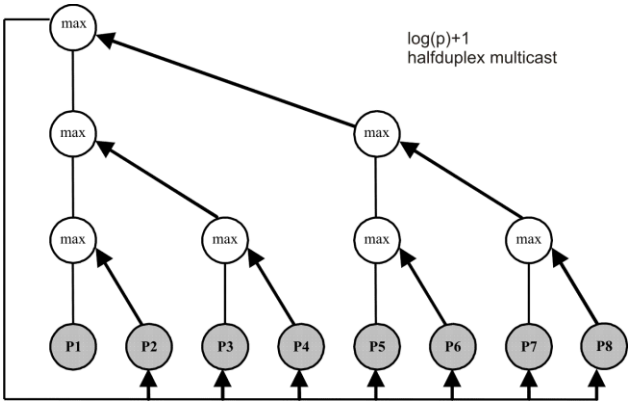


Figura 4.10: Red de comunicaciones entre procesos para 8 procesadores.

Con este esquema de comunicación entre los procesadores, el coste de I/O de este algoritmo es $1+[\log_2(p)*lat]$, siendo p el número de procesadores utilizados, y lat la latencia correspondiente a cada comunicación. Se define latencia lat como el tiempo que transcurre desde que un proceso envía un mensaje, hasta que dicho mensaje es recibido por el proceso destino. El tiempo resultante de la fórmula anterior se multiplica por dos, ya que se repite tanto para el envío de i como para el envío de j.

4.4.6 MPI (Message Passing Interface)

Para implementar el desarrollo paralelo del algoritmo detallado anteriormente se ha utilizado la librería MPI (*Message Passing Interface*), dado que es un estándar en este tipo de desarrollos y está presente en todas las instalaciones clúster de un centro de supercomputación.

MPI (*Message Passing Interface*) [Karniadakis, 2003] es un interfaz estandarizado para la realización de aplicaciones paralelas basadas en paso de mensajes. El modelo de programación que subyace tras MPI es MIMD (*Multiple Instruction streams, Multiple Data streams*), aunque se dan especiales facilidades para la utilización del modelo SPMD (*Single Program Multiple Data*), un caso particular de MIMD en el que todos los procesos ejecutan el mismo programa, aunque no necesariamente la misma instrucción al mismo tiempo.

MPI es, como su nombre indica, un interfaz, lo que quiere decir que el estándar no exige una determinada implementación del mismo. Lo importante es dar al programador una colección de funciones para que éste diseñe su aplicación, sin que tenga necesariamente que conocer el hardware concreto sobre el que se va a ejecutar, ni la forma en la que se han implementado las funciones a utilizar.

Esta librería de paso de mensajes permite que el programador utilice varios procesadores con su propio espacio de memoria, y escriba un programa que se ejecute en cada uno de ellos. La programación paralela por definición requiere cooperación entre los procesadores, lo cual requerirá algún tipo de comunicación entre ellos. Esta comunicación es el objetivo principal del

paradigma de MPI, y se realiza a través del intercambio de mensajes entre procesos. El modelo de paso de mensajes no posee el concepto de memoria compartida, ni el acceso directo a la memoria de otro procesador, por lo que los programas se ejecutan en los procesadores individuales involucrados y las operaciones de paso de mensajes son sólo llamadas a subrutinas.

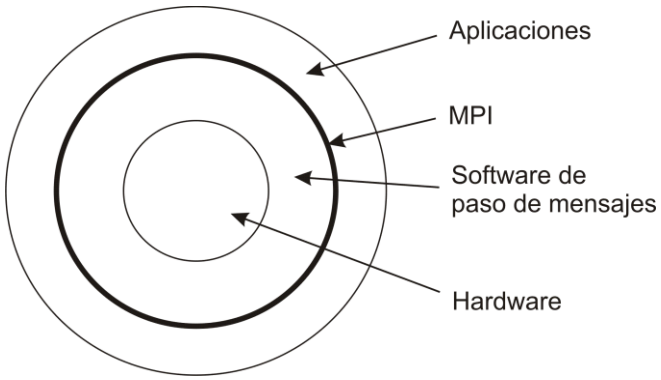


Figura 4.11: Ubicación de MPI dentro de un sistema informático.

Las llamadas de MPI se dividen en cuatro clases:

a) Llamadas utilizadas para inicializar, administrar y finalizar las comunicaciones.
b) Llamadas utilizadas para transferir datos entre un par de procesos.
c) Llamadas para transferir datos entre varios procesos.
d) Llamadas utilizadas para crear tipos de datos definidos por el usuario.

Las llamadas de la primera clase permiten inicializar la biblioteca de paso de mensajes, identificar el número de procesos (`size`) y el rango de los procesos (`rank`). La segunda clase de llamadas incluye operaciones de comunicación punto a punto, para diferentes tipos de actividades de envío y recepción. Las llamadas de la tercera clase son conocidas como operaciones grupales, y hacen posibles las comunicaciones entre grupos de procesos. Las llamadas de la última clase proporcionan flexibilidad en la construcción de estructuras de datos complejos.

a) Llamadas para inicializar, administrar y finalizar las comunicaciones

MPI dispone de 4 funciones básicas que se utilizan en todo programa con MPI. Estas funciones son `MPI_Init`, `MPI_Comm_size`, `MPI_Comm_rank` y `MPI_Finalize`.

- `MPI_Init` permite inicializar una sesión MPI. Esta función debe ser utilizada antes de llamar a cualquier otra función de MPI.

- `MPI_Finalize` permite terminar una sesión MPI. Esta función debe ser la última llamada a MPI que un programa realice. Permite liberar la memoria usada por MPI.

- `MPI_Comm_size` permite determinar el número total de procesos que pertenecen a un comunicador.

- `MPI_Comm_rank` permite determinar el identificador (`rank`) del proceso actual.

b) Llamadas para transferir datos entre dos procesos

La transferencia de datos entre dos procesos se consigue mediante las llamadas `MPI_Send` y `MPI_Recv`. Estas llamadas devuelven un código que indica su éxito o fracaso.

- `MPI_Send` permite enviar información desde un proceso a otro.
- `MPI_Recv` permite recibir información desde otro proceso. Ambas funciones son bloqueantes, es decir, que el proceso que realiza la llamada se bloquea hasta que la operación de comunicación se complete.
- `MPI_Isend` y `MPI_Irecv`, son las versiones no bloqueantes de `MPI_Send` y `MPI_Recv`, respectivamente. Estas llamadas inician la operación de transferencia, pero su finalización debe ser realizada de forma explícita mediante llamadas como `MPI_Test` y `MPI_Wait`.

- `MPI_Wait` es una llamada bloqueante y retorna cuando la operación de envío o recepción se completa. `MPI_Test` permite verificar si la

operación de envío o recepción ha finalizado, esta función primero chequea el estado de la operación de envío o recepción y luego retorna.

c) Llamadas para transferir datos entre varios procesos

MPI posee llamadas para comunicaciones grupales que incluyen operaciones tipo difusión (broadcast), recolección (gather), distribución (scatter) y reducción (reduce). Algunas de las funciones que permiten realizar transferencia entre varios procesos se presentan a continuación.

- `MPI_Barrier` permite realizar operaciones de sincronización. En estas operaciones no existe ninguna clase de intercambio de información. Suelen emplearse para dar por finalizada una etapa del programa, asegurándose de que todos los procesos han terminado antes de dar comienzo a la siguiente.

- `MPI_Bcast` permite a un proceso enviar una copia de sus datos a otros procesos dentro de un grupo definido por un comunicador.

- `MPI_Scatter` establece una operación de distribución, en la cual un dato (array de algún tipo de datos) se distribuye en diferentes procesos.

- `MPI_Gather` establece una operación de recolección, en la cual los datos son recolectados en un solo proceso.

- `MPI_Reduce` permite que el proceso raíz recolecte datos desde otros procesos en un grupo, y los combine en un solo ítem de datos. Por ejemplo, se podría utilizar una operación reducción, para calcular la suma de los elementos de un array que se distribuyó en algunos procesos.

d) Llamadas utilizadas para crear tipos definidos por el usuario

Para definir nuevos tipos de datos se puede utilizar la llamada `MPI_Type_struct` para crear un nuevo tipo o se puede utilizar la llamada `MPI_Pack` para empaquetar los datos. De este último tipo de llamadas no se ha hecho ningún uso en el código. Para una mayor información sobre las funciones

existentes en MPI, el tipo y número de sus parámetros pueden consultarse [Gropp, 1999].

4.4.7 Detalle del código desarrollado

En este apartado se hace un repaso rápido de los aspectos más importantes de la implementación realizada sobre las dos aproximaciones paralelas escogidas: *SVM en cascada y SVM Spread-Kernel*.

4.4.7.1 SVM en cascada

En el desarrollo en cascada no se han realizado grandes modificaciones sobre el código original de LIBSVM, pero sí se han añadido varias funciones que permitan realizar las operaciones descritas en el esquema de la *Figura 4.5*, así como el código necesario para implementar las operaciones de la *ecuación 4.1*, y la comprobación de si se ha alcanzado la condición de finalización mediante el test *KKT*, necesario antes del comienzo de cada iteración.

Las principales funciones creadas son:

- `partir`: Recibe como parámetro de entrada el nombre del fichero con los datos de entrenamiento y genera n ficheros según el número de procesos solicitados en los parámetros de entrada.

- `une_modelos`: Recibe como entrada los dos ficheros con los vectores soporte del nivel anterior, más las matrices $\vec{\alpha}$ y Q correspondientes, necesarias para generar un único fichero de entrada al nuevo nivel de SVM, aplicando las operaciones indicadas en la *ecuación 4.1.*

- `test_kkt`: Recibe como entrada los resultados de una iteración completa del sistema en cascada y finaliza la ejecución de la aplicación en el caso de que todos los vectores soporte cumplan las condiciones *KKT* reflejadas en la *Figura 4.2*. En caso contrario la ejecución continúa.

En la *Figura 4.12* se presenta la estructura de la función principal de LIBSVM modificada para incluir las funciones desarrolladas que permiten

generar el algoritmo en cascada. En las líneas 3, 4 y 5 se realizan las declaraciones de variables, inicializaciones, inicio de MPI y se comprueban los parámetros de entrada mediante la función `parse_command_line` original de LIBSVM. Posteriormente, en las líneas de la 9 a la 14 se inicializan dos variables: *nivel* y *paso*. La primera almacena el nivel en el que se encuentra la ejecución dentro del esquema en cascada que refleja la *Figura 4.5*. La segunda variable, *paso,* indica el salto dentro del grupo de procesos implicados en la ejecución. Inicialmente irá de uno en uno y se incrementará en una potencia de dos, cuyo valor vendrá dado por la variable *nivel*. El resultado de *paso* se irá almacenando para cada nivel en el array *procesos* con el fin de controlar mediante llamadas MPI qué proceso entra en funcionamiento en cada nivel, y de dónde debe recoger los datos de entrada del nivel anterior.

Una vez anotados los procesos responsables de la ejecución en cada nivel, se pasa al bucle principal, que va desde la línea 16 a la 43. En este fragmento de código se llama a la función `partir` para generar los distintos puntos de entrenamiento que se pasan como entrada en el primer nivel, después entre la línea 23 y la 41 se genera un nuevo bucle con el que se irá avanzando por los diferentes niveles del desarrollo en cascada. Antes de entrar en cada nivel se genera el fichero de entrada como unión de los dos ficheros del nivel anterior, se llama a la función `unir_modelos`, que fusiona dichos ficheros, así como las matrices Q y $\vec{\alpha}$. Todo esto se realiza entre las líneas 28 y 31.

Tras generar los ficheros de entrada sólo queda llamar a la función original de LIBSVM `read_problem` que carga los datos en las variables correspondientes para luego llamar a `svm_train`, que genera el modelo de salida para los datos de entrenamiento colocados a la entrada de cada máquina de vectores soporte.

En caso de que se esté en el nivel 0 y la variable *iter* sea mayor que 0, se interpretará que el sistema ha ejecutado una iteración completa, por lo que en la línea 33 se llama a la función `test_kkt` para ver si se ha llegado al final de la ejecución.

```
1. int main(int argc, char **argv)
2.  {
3.    /* Declaraciones e inicializaciones */
4.    /* Inicializar MPI */
5.    /* Comprobar parámetros de entrada */
6.
7.    procesos=(int *)malloc(sizeof(int)*(numproc*numproc/2)+1);
8.    for (i=0;i<((numproc*numproc/2)+1);i++) procesos[i]=0;
9.    nivel=0;paso=pow(2,nivel);
10.   while (paso<=numproc)
11.    {
12.      for (i=0;i<numproc;i+=paso) procesos[(numproc*nivel)+i]=1;
13.      nivel++; paso=pow(2,nivel);
14.    }
15.   iter=0;
16.   do
17.    {
18.     nivel=0;
19.     MPI_Barrier(MPI_COMM_WORLD);
20.     partir(input_file_name,model_file_name,iter);
21.     paso=pow(2,nivel);
22.     /* Construir nombre fichero con datos de entrada */
23.     while(paso<=numproc)
24.      {
25.        MPI_Barrier(MPI_COMM_WORLD);
26.        if(procesos[(nivel*numproc)+miproc]==1)
27.        {
28.         /* Generar nombre fichero de entrada */
29.         if (nivel!=0) une_modelos(input_file_name,model_file_name,
30.                                   fich,nivel-1,miproc);
31.         /* Generar en model1 el nombre del fichero de salida */
32.         read_problem(fich);
33.         if ((iter>0) && (nivel==0)) test_kkt(&prob, &param)
34.         prob.nivel=nivel;
35.         strcpy(prob.fich,fich);
36.        model = svm_train(&prob,&param);
37.        svm_save_model(model1,model,0);
38.       }
40.      nivel++;paso=pow(2,nivel);
41.      }
42.    iter++;
43.   }while (diff>LIMITE);
44.    MPI_Barrier(MPI_COMM_WORLD);
45.    MPI_Finalize (); /* Finalizar MPI */
46.  }
```

Figura 4.12: Función principal de SVM en cascada.

4.4.7.2 SVM con SPREAD-KERNEL

A continuación se detallan todas y cada una de las modificaciones realizadas para implementar la solución SPREAD-KERNEL sobre la aplicación original de LIBSVM. El código presentado es una modificación del método `solve` de la clase `Solver` dentro de la aplicación LIBSVM, y de momento sólo se aplica a la parte de clasificación, aunque en el futuro se podría adaptar para que también pueda ser utilizado en regresión.

```
1. Qfloat *get_Q(int i, int uno,int ini, int len) const
2. {
3.   Qfloat *data;
4.   int start, j, inicio;
5.
6.   if (uno==0)
7.      {
8.        inicio=ini;len-=ini;
9.        if( (start = cache->get_data(i,&data,ini,len))< len)
10.            for(j=0;j<len;j++)
11.         data[j] = (Qfloat)(y[i]*y[j+ini]*
12.                    (this->*kernel_function)(i,j+ini));
13.     }
14.  else
15.     {
16.      data= (Qfloat *)malloc(sizeof(Qfloat));
17.          data[0]= (Qfloat )(y[i]*y[ini]*
18.                      (this->*kernel_function)(i,ini));
19.     }
```

Figura 4.13: Función para obtener una fila del kernel de la caché.

Mejora del rendimiento de la caché

Antes de comenzar a detallar la división de los datos de entrada y la búsqueda del conjunto de trabajo en cada iteración, se expone la modificación realizada en el manejo original de la caché de LIBSVM.

El código original, busca en la caché una fila que contenga los valores de una columna determinada k, en el caso de no estar esa columna en la caché, vuelve a calcularla y sobrescribe el bloque más antiguo en la caché. La modificación realizada no altera el cambio de un bloque en la caché pero sí el cálculo de los valores a introducir, ya que ahora se realizan las operaciones del kernel correspondiente sólo entre las posiciones lo e hi de la fila solicitada.

Además, se ha introducido un caso especial mediante el parámetro *uno,* que indica que sólo interesa extraer un valor de la fila i del kernel. Este valor será obtenido igualando las columnas lo e hi al valor solicitado. De nuevo esta modificación ahorra muchas operaciones, ya que originalmente cuando sólo se necesitaba un único valor de una fila, si la fila no estaba en la caché se tenían que calcular todos los elementos de esa fila para poder cargar el bloque en la caché. El código correspondiente a esta modificación se encuentra detallado en la *Figura 4.13.*

Búsqueda de la fila más alejada de las condiciones KKT

En el desarrollo realizado sólo se han utilizado las llamadas bloqueantes (`MPI_Send` y `MPI_Recv`) de transferencia de datos entre dos procesos y la llamada de comunicación entre varios procesos `MPI_Bcast` para enviar mensajes desde el proceso 0 al resto de procesos.

Hay dos puntos principales en los que se reparten las instrucciones entre los diferentes procesadores. El primer punto es la búsqueda de la coordenada i que incumple las condiciones *KKT* de forma más holgada. En la *Figura 4.14* se puede ver cómo, tras un recorrido por la matriz de gradientes, la variable i contiene el índice buscado, mientras que la variable gi contiene el valor de la coordenada i.

El recorrido dentro de la matriz de gradientes va desde el límite menor que le corresponde a ese procesador, hasta la variable *lzz1,* que marca el mínimo entre el último valor de la clase 0 (valor positivo) dentro de los valores de entrenamiento y el máximo índice correspondiente al proceso. La posición del último valor de la clase 0 de los datos de entrenamiento se obtiene al leer los datos del fichero de entrada y ordenarlos según se ha descrito en la *Sección 4.4.3,* referente a optimizaciones secuenciales del código. El mismo caso ocurre para el segundo bucle, que es el dedicado a los valores negativos y que va desde $zz2$ a hi, siendo $lzz2$ el mínimo entre lo y el primer valor de la clase 1 de los datos de entrenamiento.

Como se observa en el código de la *Figura 4.14,* cada proceso le envía los datos a un solo proceso que viene dado por el valor de la variable $envi$. Asimismo, un proceso puede recibir datos de 0 ó varios procesos que vienen indicados por la posición k del array a_rec.

Para cada valor $gmax$ recibido se ve si es mayor que el máximo anterior. En ese caso, el nuevo valor pasa a ser $gmax$ y se actualizan las variables i y gi. Por último, se realiza una llamada `Bcast` para comunicar el valor de i a todos los procesos que comenzarán a calcular el valor j.

```
1.  for(int t=lo;t<lzz1;t++)
2.    if(!is_upper_bound(t))
3.    if(-G[t] >= gmax)
4.      {
5.       gmax = -G[t];
6.       i = t;
7.      }
8.  for(int t=lzz2;t<hi;t++)
9.  if(!is_lower_bound(t))
10.   if(G[t] >= gmax)
11.    {
12.      gmax = G[t];
13.      i = t;
14.    }
15. gi=G[i];
16.
17. for (k = 0; k< rec; k++)
18.    {
19.      MPI_Recv (gg, 3, MPI_DOUBLE, a_rec[k], 79, MPI_COMM_WORLD,
          &status);
20.      if (gmax<=gg[0]) {i=(int)gg[1];gmax=gg[0];gi=gg[2];}
21.    }
22. if (miproc!=0)
23.    {
24.      gg[0]=gmax;gg[1]=(double)i;gg[2]=gi;
25.      MPI_Send(gg, 3, MPI_DOUBLE, envi, 79, MPI_COMM_WORLD);
26.    }
27. else
28.    {
29.      gg[0]=gmax;gg[1]=(double)i;gg[2]=gi;
30.    }
31.
32. if ((numproc>1)&& (miproc==0))
33.    {
34.      MPI_Bcast(gg, 3, MPI_DOUBLE, 0,  MPI_COMM_WORLD);
35.      gmax=gg[0];i=(int)gg[1];gi=gg[2];
36.    }
```

Figura 4.14: Búsqueda de *i* y comunicación entre procesos.

Obtención de los números de proceso de envío y recepción

Antes de comentar la obtención del valor *j* se va a presentar el código necesario para construir el array *a_rec* y obtener el valor de la variable *envi*.

El código necesario para obtener dichos valores aparece en la *Figura 4.15*. Este código ha sido diseñado para obtener en cada nivel los procesos que envían y reciben siguiendo el esquema `full-duplex` mostrado en la *Figura 4.10* dentro del apartado de tráfico de red.

```
1. a_rec=(int *) malloc((numproc/2)*sizeof(int));
2.
3. nivel1=1;nivel2=2;rec=0;
4. while (nivel1<numproc)
5.  {
6.   for (vv=0;vv<numproc;vv+=nivel2)
7.    {
8.      li=min(numproc,vv+nivel2);
9.      for (ff=vv+nivel1;ff<li;ff+=nivel1)
10.        {
11.         if (miproc==ff) envi=vv;
12.         if (miproc==vv)
13.           {
14.            a_rec[rec]=ff;rec++;
15.           }
16.        }
17.    }
18.  nivel1=nivel2;
19.  nivel2*=2;
20. }
```

Figura 4.15: Obtención de los números de procesos de recepción y del proceso de envío.

Selección del conjunto de trabajo

En este apartado se detalla la función `select_working_set`, a la que se llama una vez obtenido el valor de la fila *i,* que se encuentra más alejada de las condiciones *KKT* mostradas en la *Figura 4.2*, en concreto se realiza una selección de segundo orden en la columna *j* y para ello se hace una llamada a la función `get_Q` para obtener el valor de la fila *i*. Como se puede ver en las líneas 11 y 12, se hace uso de la función modificada para recuperar un único valor de la caché.

En este fragmento de código de nuevo se hace uso de las variables *lzz1* y *lzz2,* que indican el final de la clase 0 y el principio de la clase 1 respectivamente. Asimismo, de las líneas 15 a la 49 se busca el valor más alejado de las condiciones *KKT,* para posteriormente retornar su valor en las líneas 55 y 56 a través de los parámetros de salida *out_gmax2* y *out_gmin*.

```
1.  int Solver::select_working_set(int i, int &out_j, double Gmax,
    double &out_gmin, int lo, int hi, double &out_gj, double &out_gmax2,
    int lzz1, int lzz2)
2.  {
3.   Qfloat qi;
4.   double Gmax2 = -INF;
5.   register int Gmin_idx = -1, zz1, zz2,j;
6.   double obj_diff_min = INF;
7.
8.   const Qfloat *Q_i = NULL;
9.   if(i != -1) // NULL Q_i not accessed: Gmax=-INF if i=-1
10.    {
11.     Q_i=Q->get_Q(i,0,lo,hi);
12.     qi=*(Q->get_Q(i,1,i,i));
13.    }
14.  for (j=lo;j<lzz1;j++)
15.     if (!is_lower_bound(j))
16.     {
17.      double grad_diff=Gmax+G[j];
18.      if (G[j] > Gmax2) Gmax2 = G[j];
19.       if (grad_diff > 0)
20.        {
21.      double obj_diff;
22.      double quad_coef=qi+QD[j]-2.0*y[i]*Q_i[j-lo];
23.      if (quad_coef > 0) obj_diff = -(grad_diff*grad_diff)/quad_coef;
24.                    else obj_diff = -(grad_diff*grad_diff)/TAU;
25.      if (obj_diff <= obj_diff_min)
26.           {
27.             Gmin_idx=j;
28.             obj_diff_min = obj_diff;
29.           }
30.     }
31.  }
32.  for (j=lzz2;j<hi;j++)
33.   if (!is_upper_bound(j))
34.   {
35.    double grad_diff= Gmax-G[j];
36.    if (-G[j] > Gmax2) Gmax2 = -G[j];
37.    if (grad_diff > 0)
38.     {
39.       double obj_diff;
40.       double quad_coef=qi+QD[j]+2.0*y[i]*Q_i[j-lo];
41.       if (quad_coef > 0) obj_diff = -(grad_diff*grad_diff)/quad_coef;
42.                     else obj_diff = -(grad_diff*grad_diff)/TAU;
43.       if (obj_diff <= obj_diff_min)
44.           {
45.              Gmin_idx=j;
46.              obj_diff_min = obj_diff;
47.           }
48.     }
49.   }
50. if (Gmin_idx!=-1)
51.  {
52.   out_j = Gmin_idx;
53.   out_gj = G[Gmin_idx];
54.  }
55. out_gmax2=Gmax2;
56. out_gmin= obj_diff_min;
57. return 0;
58. }
```

Figura 4.16: Búsqueda de *j* para completar la selección del conjunto de trabajo.

Código de envío de la columna más alejada de las condiciones KKT

En la sección de código mostrada en la *Figura 4.17*, se completa el envío del conjunto de trabajo, ya que anteriormente se ha enviado el valor de la fila *i* a todos los procesos y ahora, una vez ejecutada la función `select_working_set`, se conoce el valor de la columna más alejada de las condiciones *KKT*. Para saber a qué proceso hay que realizar el envío se hace uso de la variable *envi*. Igualmente, para conocer los procesos desde los cuales hay que recibir los datos se utiliza el array *a_rec*.

Como cada proceso envía su columna más alejada y lo que interesa es la más alejada del conjunto de todos los procesos, entonces se deben hacer comparaciones de la variable *gmax2,* que contiene el valor que marca la distancia en la que sobrepasan las condiciones *KKT*.

```
1.  fin=select_working_set(i,j,gmax,gmin,lo,hi,gj,gmax2,lzz1,lzz2);
2.  for (kk = 0; kk < rec; kk++)
3.   {
4.    MPI_Recv (gg, 4, MPI_DOUBLE, a_rec[kk], 69, MPI_COMM_WORLD,
        &status);
5.    if (gmax2<gg[2]) gmax2=gg[2];
6.    if (gmin>=gg[0])
7.     {
8.       j=(int)gg[3];gmin=gg[0];gj=gg[1];
9.     }
10.  }
11. if (miproc!=0)
12.  {
13.   gg[0]=gmin;gg[1]=gj;gg[2]=gmax2;gg[3]=(int)j;
14.   MPI_Send(gg, 4, MPI_DOUBLE, envi, 69, MPI_COMM_WORLD);
15.  }
16. else { gg[0]=gj;gg[1]=gmax2;gg[2]=(double)j;}
17. if (numproc>1)
18.  {
19.   MPI_Bcast(gg, 3, MPI_DOUBLE, 0,  MPI_COMM_WORLD);
20.   gj=gg[0];gmax2=gg[1];j=(int)gg[2];
21.  }
```

Figura 4.17: Comunicación entre procesos para el envío del valor j.

Finalmente, en la *Figura 4.17* se muestra el código en el que una vez obtenida la columna más alejada entre todos los procesos, se efectúa un `Bcast` desde el proceso 0, que es el que contiene la columna buscada, hacia todos los procesos involucrados en el cálculo siguiendo el esquema `half-duplex`, comentado en la *Sección 4.4.5.*

4.5 Pruebas de escalabilidad

Los datos de entrada utilizados para realizar las pruebas de ejecución son 581012 vectores de entrenamiento con 54 características cada uno. El conjunto de prueba que se llamará `covtype` se ha obtenido de la página web de LIBSVM, de esta forma se ha evitado hacer la transformación de los datos iniciales al formato LIBSVM. Los datos originales se encuentran en el repositorio de datos para máquinas de aprendizaje, cuya dirección Web es: http://archive.ics.uci.edu/ml/, concretamente los datos de estas pruebas se corresponden con una muestra de tipos de suelo obtenidos por el servicio forestal de Estados Unidos. Las máquinas sobre las que se han realizado las medidas de rendimiento han sido: el cluster Lince del CIEMAT, y el superordenador Finisterrae del Centro de Supercomputación de Galicia (CESGA). Las características de ambos sistemas aparecen detalladas en el Anexo II.

4.5.1 SVM en cascada

En la *Figura 4.18* se muestran los tiempos de ejecución en segundos de la implementación SVM en cascada. Se observa que la aceleración conseguida, según se aumenta el número de procesadores, es muy pobre, e incluso para un número de procesadores superior a 16 la curva de aceleración decae, por lo que para un número de procesadores igual a 16 se alcanza el límite de Amdahl, que como se vio anteriormente, es el máximo número de procesadores necesarios para ejecutar un problema de tamaño fijo en el menor tiempo posible.

Número de CPU's	Tiempo de ejecución	Aceleración
1	6780	1
2	4347	1,56
4	2774	2,44
8	2052	3,30
16	1716	3,95
32	1850	3,66

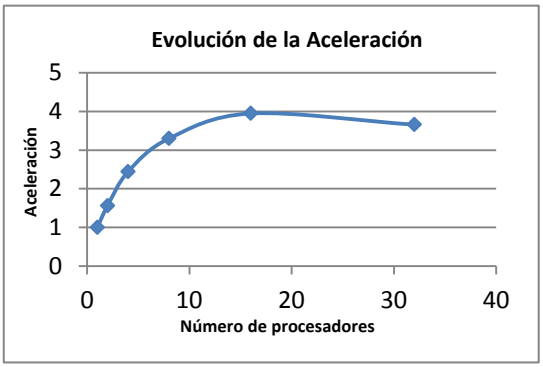


Figura 4.18: Tiempos de ejecución en CIEMAT de SVM en cascada.

Estos resultados tan pobres coinciden con los reflejados en [Graf, 2005], donde se observa que para un número de procesos superior a 8 el rendimiento disminuye bruscamente. Esto se debe a que la estructura en cascada obliga a que todos los vectores soporte pasen por una última fase en la que sólo se tiene un único proceso en funcionamiento, por lo que si el número de vectores soporte final es elevado en dicha fase el sistema se ralentiza.

Los tiempos anotados en la tabla de la *Figura 4.18* se han obtenido tras una única pasada del algoritmo SVM en cascada, ya que tras numerosas pruebas con distintos tamaños de entrenamiento de los datos de `covtype`, además de con otros datos obtenidos de la Web de LIBSVM, en ningún caso se mejoraba en más de 1% el porcentaje de aciertos tras varias pasadas, y sí que aumentaba considerablemente el tiempo de ejecución.

4.5.2 Spread-Kernel

A continuación se presentan los resultados obtenidos tras la ejecución de varias muestras de datos sobre los distintos entornos de proceso.

En la *Tabla 4.1*, se observan los tiempos de ejecución de la aplicación paralela en función del número de procesadores para el conjunto de datos `covtype`.

	Tiempo de ejecución en segundos			
Número de CPUS	**CIEMAT**	**log2(t_{CIEMAT})**	**CESGA**	**log2(t_{CESGA})**
1	36586	15,16	66033	16,01
2	19721	14,27	32384	14,98
4	10213	13,32	16376	14,00
8	5270	12,36	8216	13,00
16	2847	11,48	2920	11,51
24	1870	10,87	1520	10,57
32	1449	10,50	885	9,79
48	1053	10,04	527	9,04
64	863	9,75	411	8,68
96	-	-	299	8,22
128	-	-	245	7,94
160	-	-	225	7,81

Tabla 4.1: Comparativa de tiempos de ejecución en CESGA y CIEMAT.

En la gráfica de la *Figura 4.19* se muestra la evolución de tiempos en función del número de procesos. En este caso se ha colocado el eje X en escala logarítmica, ya que los tiempos van desde 67823 segundos para un solo procesador en CESGA hasta 225 segundos para 160 procesadores, lo que hace difícil observar los datos sin hacer esta transformación. En la gráfica se observa cómo a partir de 16 procesadores los tiempos mejoran ostensiblemente, ya que al dividir los datos por encima de 16 el tamaño en memoria ocupado por el trozo de la matriz kernel, en el que cada proceso hace la búsqueda del conjunto de trabajo, es cada vez menor, llegando a poder almacenarse directamente en la caché del procesador, que en el caso del CESGA se trata de un Itanium con 18MB de caché L2.

Sin embargo, para la máquina del CIEMAT la curva de tiempos es mucho más estable y se necesitaría dividir mucho más el conjunto de datos para obtener este incremento en la aceleración, ya que la caché de cada procesador Xeon es sólo de 2MB.

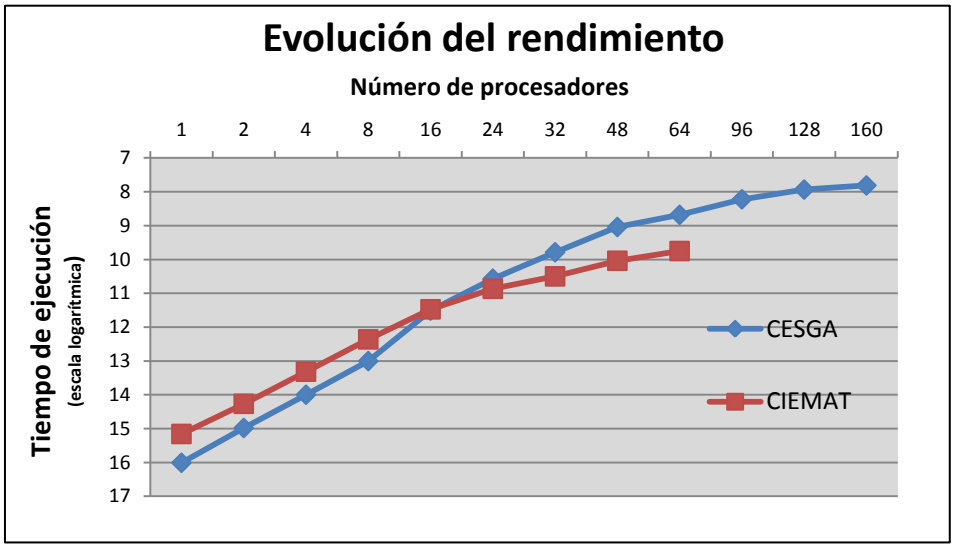


Figura 4.19: Evolución del rendimiento para los datos de entrada covtype.

En la *Figura 4.20* se muestra la gráfica de escalabilidad de la aplicación, de nuevo para los datos de entrada `covtype`. En este caso se observa que el sistema posee una gran escalabilidad, recuérdese que un sistema escala perfectamente si al presentar la gráfica de tiempos de ejecución en escala

logarítmica frente al número de procesos, también en escala logarítmica, lo que se obtiene es una recta perfecta.

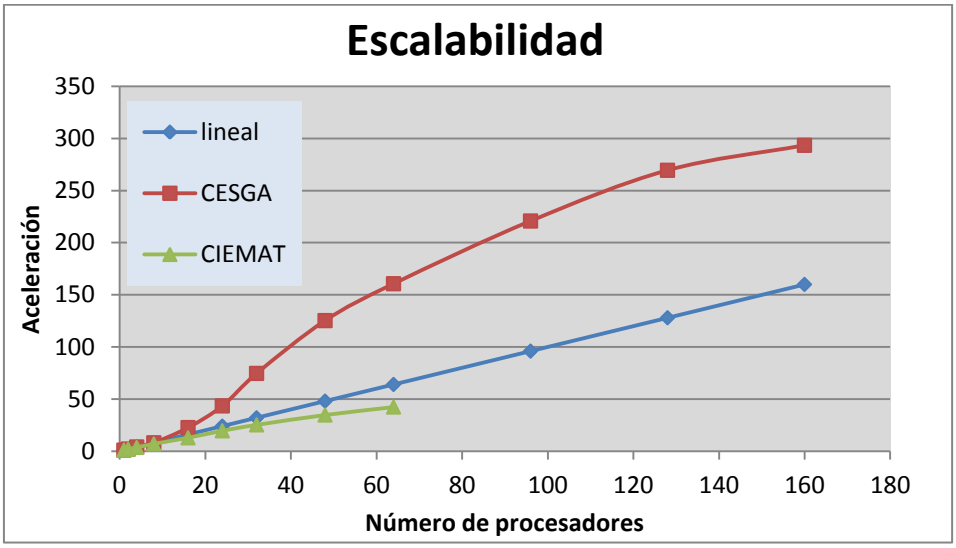


Figura 4.20: Escalabilidad de la implementación paralela.

En este caso la recta es perfecta en la máquina del CIEMAT porque la configuración de colas batch permite ejecutar los trabajos sin tener que compartir ni la red, ni la máquina con ningún otro trabajo. Sin embargo en el CESGA las ejecuciones en un nodo se han compartido con el trabajo de otro usuario.

Nº de CPUS	CIEMAT (segundos)	ACELERACION	CESGA (segundos)	ACELERACION
1	36586	1,00	66033	1,00
2	19721	1,86	32384	2,04
4	10213	3,58	16376	4,03
8	5270	6,94	8216	8,04
16	2847	12,85	2920	22,61
24	1870	19,56	1520	43,44
32	1449	25,25	885	74,61
48	1053	34,74	527	125,30
64	863	42,39	411	160,66
96	-	-	299	220,85
128	-	-	245	269,52
160	-	-	225	293,48

Tabla 4.2: Tiempos de ejecución en CESGA y CIEMAT.

Por último, destacar que para el caso del CESGA se observan tres partes bien diferenciadas. En la primera parte, hasta 8 procesadores, el tiempo de ejecución viene determinado por el número de procesos que intervienen. En la segunda parte, entre 8 y 32 procesadores, los tiempos disminuyen en función de

la cantidad de datos que quedan fuera de la caché. Finalmente, en la tercera parte de la gráfica, a partir de 32 procesadores, todos los datos están dentro de la caché y el tiempo de ejecución se ve condicionado más por la velocidad de las comunicaciones que por la velocidad del procesador. Todos los tiempos de ejecución de la *Figura 4.20* aparecen reflejados en la *Tabla 4.2*.

4.5.3 Comparativa de los dos algoritmos paralelos

En la *Figura 4.21* se muestra una comparativa de la aceleración conseguida por las dos técnicas en función del número de procesadores, para los datos de entrenamiento de la muestra `covtype`, obtenidos en el cluster Lince del CIEMAT. En el caso de la técnica en cascada la aceleración se aleja bastante de la lineal e incluso a partir de 16 procesadores su rendimiento decae, por este motivo se ha optado por usar la técnica Spread-SVM para tratar los datos de las muestras procedentes de los diagnósticos en los experimentos de fusión.

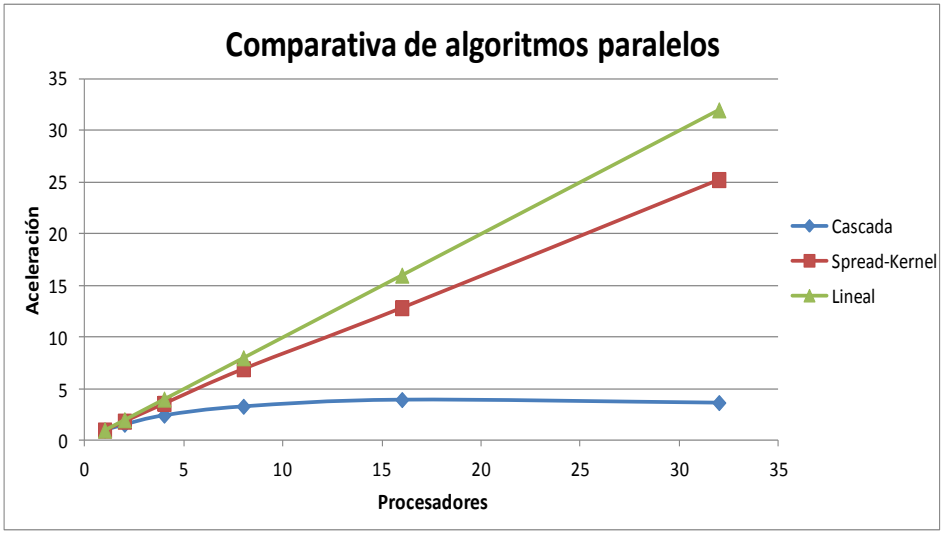


Figura 4.21: Comparativa de aceleración de los algoritmos paralelos.

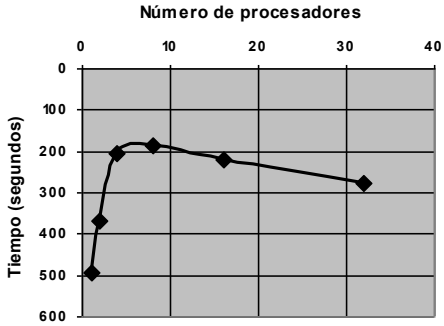
4.6 Resultados sobre datos de experimentos de Fusión

Como se mencionó en el Capítulo 2 de esta tesis, lo que ha motivado la realización de este trabajo ha sido el análisis de la enorme cantidad de datos que se obtienen en cada experimento de fusión. Los datos de los experimentos

de fusión con los que se ha trabajado se han extraído de las bases de datos del Centro de Investigación de Culham en el Reino Unido. Este centro alberga el Joint European Torus (JET) [EFDA, 2013], que es el dispositivo de fusión tokamak más grande del mundo, utilizado por investigadores de más de 20 países.

En los experimentos de fusión realizados en JET se somete al plasma a unas determinadas condiciones de temperatura, campos magnéticos, radiación, etc. y se observan los cambios que se producen en dicho plasma mediante sensores de temperatura, presión, etc. Una característica física importante en este tipo de experimentos es el estado del plasma, que puede encontrarse confinado en modo-L o en modo-H, así como el instante exacto en el que se produce esa transición de estados. Averiguar el estado en el que se encuentra el plasma durante un experimento de fusión se puede conseguir estudiando el valor de cinco señales obtenidas por los detectores de JET.

Las señales se han extraído de 56 descargas (experimentos de fusión), 50 de las cuales, seleccionadas al azar, se han utilizado como datos de entrenamiento. El resto de las descargas se han utilizado para datos de test. Como se ha mencionado anteriormente, se necesitan cinco señales para encontrar el estado del plasma, en concreto, los nombres de estas señales son los siguientes: Bndiam (Beta normalizada con respecto a la energía diamagnética), Ptot (Poder calorífico total), XPrl y XPzl (coordenadas R y Z de XP) y por último, Wmhd (Energía magnetohidrodinámica).



Procesadores	Tiempo (segundos)
1	494
2	368
4	204
8	187
6	219
32	277

Figura 4.22: Resultados de ejecución de Spread-SVM en el cluster Lince con datos de experimentos de fusión de JET.

De cada señal se han tomado datos cada dos milisegundos, 2.7 segundos antes y después del instante de transición entre el modo-L y el modo-H. Este instante de transición se ha obtenido a partir del estudio gráfico de las variables Ptot, Dalphaln (S3AD/AD36) y S3AD/AD35. Teniendo en cuenta los datos anteriores, se ha obtenido un fichero de entrada formado por 135000 vectores de entrenamiento con 5 características para cada vector. Los resultados obtenidos se muestran en la *Figura 4.22* y han sido presentados en el trabajo [Ramírez, 2010].

Como se observa claramente en la gráfica, el rendimiento decae a partir de 4 procesadores. Esto es debido a que los datos utilizados (135000) en estas ejecuciones son escasos para el método spread-SVM, ya que como se demuestra en [Bottou, 2007], se necesitan al menos 500000 datos de entrada para obtener una buena escalabilidad, por lo menos hasta 32 procesadores.

Las tasas de acierto para los datos de test han sido las que aparecen en la *Tabla 4.3.* Como se puede ver, se consiguen unas tasas de acierto superiores al 99% usando un Kernel RBF y ajustando adecuadamente los valores de C y *gamma*, por lo que se puede deducir que la máquina se ha entrenado perfectamente para este conjunto de datos.

$K(x,z) = \exp(-\gamma \|x-z\|^2)$		
C	**Gamma(γ)**	**Porcentaje de aciertos**
30000	0,5	89,46
60000	0,5	98,19
90000	1	99,79
90000	0,5	98,09
120000	1	98,05

Tabla 4.3: Tasa de aciertos para distintos valores de los parámetros *gamma* y *C* del Kernel RBF.

El algoritmo Spread-SVM proporciona una gran escalabilidad y ha sido aplicado con éxito en el entrenamiento de un modelo que permite clasificar satisfactoriamente transiciones de modo-L a modo-H. Antes de poseer esta herramienta para generar un modelo que permitiese la clasificación de datos como los mostrados, se necesitaban 90 horas en un ordenador personal

utilizando un toolbox de Matlab [Rattá, 2010], mientras que ahora sólo se necesitan tres minutos de ejecución en el clúster Lince usando 16 procesadores.

Señalar también que el modelo anterior obtenido a partir de 50 descargas se ha probado con datos de descargas obtenidas por otro grupo de investigadores y de las que se conoce, con una fiabilidad del cien por cien, el instante de la transición L-H. Este conjunto de datos de test estaba formado por 8600 vectores de cinco características cada uno, obteniendo para este grupo de test un 93.38% de aciertos, por lo que se puede deducir que el modelo generado detecta con gran precisión las transiciones L-H.

Finalmente, y como ha quedado de manifiesto en la *Figura 4.22,* es necesario obtener un mayor número de descargas para recoger una mayor variabilidad en el tipo de transiciones a detectar por el modelo, lo cual mejorará aún más su fiabilidad, ya que el algoritmo Spread-SVM permite un mayor crecimiento, tanto en el número de procesadores, como en la cantidad de vectores de entrenamiento.

CAPÍTULO 5

Predicción de disrupciones

5.1 Introducción y estado del arte

En este capítulo se aplica la herramienta de máquinas de vectores soporte, implementada y probada con éxito en el capítulo anterior, para crear modelos que permitan reconocer si durante la ejecución de una nueva descarga aparecen síntomas de una disrupción inminente.

Existen muy pocos trabajos basados en técnicas inteligentes que hayan sido desarrollados previamente como alternativa al sistema de detección en tiempo real de JET, llamado "Jet Protection System" (JPS) [Santagiustina, 1993]. Entre los más destacados deben mencionarse los de Cannas, publicados en los años 2004 y 2007. En el primero de ellos [Cannas, 2004] se utiliza una base de datos de 274 descargas. Mediante mapas auto-organizados se realizan agrupamientos en 86 descargas disruptivas para determinar las muestras que se utilizarán en el entrenamiento del sistema de aprendizaje. Este sistema, basado en redes neuronales, detecta hasta un 68% de las disrupciones. En otro estudio posterior [Cannas, 2007] se menciona que, según la experiencia del equipo investigador, los modelos son capaces de alcanzar altas tasas de acierto únicamente si son probados con descargas pertenecientes al mismo período de las usadas en el entrenamiento. Si en el test se utilizan descargas posteriores, las tasas de acierto caen drásticamente. Proponen como solución un sistema (basado en SVM y acoplado al de predicción) destinado a detectar nuevos

comportamientos, de manera que estos puedan ser utilizados para un reentrenamiento y se evita el "envejecimiento" del predictor. En el siguiente capítulo se analizará ese objetivo, intentando mediante la paralelización del código, que el tiempo de reentrenamiento de los modelos sea el menor posible y de esta forma conseguir altos porcentajes de acierto.

Fuera de JET, en el Tokamak ASDEX-Upgrade, un estudio de referencia para la predicción del fenómeno es el desarrollado por Pautasso [Pautasso 2002]. Se basa en redes neuronales que analizan simultáneamente la evolución de 8 parámetros del plasma y sus derivadas temporales en 99 experimentos. Una vez entrenado, el sistema se prueba en un entorno simulado de tiempo real con 500 nuevos pulsos, obteniendo un 85% de predicciones correctas con un 1% de alarmas perdidas. En una segunda etapa, las pruebas son realizadas en tiempo real, obteniendo un 79% de identificaciones correctas.

En el Tokamak indio ADITYA [Sengupta, 2001], un trabajo destinado a detectar disrupciones, se utiliza una base de datos de 23 descargas. Se analizaban únicamente descargas disruptivas y no era aplicable en tiempo real. También con redes neuronales, dos estudios de Yoshino fueron desarrollados en JT-60U. En el primer trabajo [Yoshino, 2003] el sistema es entrenado en 2 pasos: primero con 12 descargas disruptivas y 6 no disruptivas (paso 1). La información de salida de la red neuronal entrenada se valida y modifica (paso 2) de acuerdo con el análisis de un grupo independiente de 12 descargas disruptivas. El modelo final fue probado con 300 descargas disruptivas y 1008 no disruptivas, obteniendo tasas de acierto mayores del 80% a 50 ms antes de la disrupción. El segundo trabajo [Yoshino, 2005] emplea una base de datos de 525 descargas y prueba que, a partir de un entrenamiento con descargas no disruptivas y un adecuado ajuste del paso 2, se pueden obtener buenos resultados en la detección de disrupciones de mayor dificultad, con una tasa de aciertos del 76%.

Es necesario mencionar los inconvenientes y limitaciones de los trabajos mencionados anteriormente:

- Una de las principales limitaciones es la escasa robustez de los sistemas predictores generados. El problema reside en que una vez entrenados, no mantienen su rendimiento en descargas de campañas posteriores a las del entrenamiento. Por consiguiente, la utilidad y funcionalidad de los sistemas comentados anteriormente quedan reducidas, al no asegurar su capacidad de generalización mediante una tasa de aciertos que permanezca razonablemente estable.

- En todos estos trabajos la única solución propuesta consiste en el continuo reentrenamiento de los sistemas creados, descartando la posibilidad de desarrollar un modelo con la suficiente capacidad de generalización, como para funcionar correctamente en campañas posteriores a las de entrenamiento. El continuo reentrenamiento de los sistemas no surge como una solución viable, al ser un proceso lento, aunque gracias a la paralelización se consigue que el reentrenamiento sea más rápido.

Finalmente, en el trabajo presentado por G. Rattá [Rattá, 2010] se genera un sistema de predicción, ***A**dvanced **P**redictor **O**f **DIS**ruptions (APODIS)*, basado en una amplia base de datos y realizado con máquinas vectores soporte. Este trabajo se desarrolló con un sistema de predicción cuya arquitectura consiste en dos capas. En la primera capa se entrenan una serie de sistemas de clasificación para analizar secuencialmente los experimentos simulando el proceso de tiempo real. Como resultado, cada uno de los clasificadores provee una predicción de la inminencia o no de una disrupción. La determinación de activar o no una alarma depende de la combinación de los valores de salida de los clasificadores de primera capa. Esta combinación se realiza por otro clasificador en una segunda capa (llamado función de decisión) que también está basado en SVM. Esta segunda capa del sistema predictor es un aspecto completamente original y una de las causas de la mejora de los resultados obtenidos con respecto a otros modelos anteriores.

El sistema que se ha implementado en este trabajo [Vega, 2013] es una variante del realizado por Rattá. En su caso el sistema está desarrollado en

Matlab y en este trabajo se ha hecho uso de programación en lenguaje C, con extensiones MPI para la paralelización de la aplicación. Además, se han realizado una serie de modificaciones del esquema original con el objetivo de mejorar las tasas de acierto y el rendimiento global del sistema.

5.2 Base de datos y arquitectura del predictor

5.2.1 Base de datos

Para una mayor fiabilidad de los resultados, se recopiló una extensa base de datos de experimentos de JET (desde la descarga 69806 a la 79853), que incluye tanto descargas disruptivas como no disruptivas. Hasta la fecha, es la mayor base de datos de JET destinada al estudio de disrupciones mediante sistemas de aprendizaje en tiempo real.

	Campaña	Año	Rangos de Descargas	No disruptivas	Disruptivas no intencionadas	Disruptivas intencionadas	Total descargas
TRAIN	**C19**	2007	[69806, 70750]	585	47	41	673
	C20	2008	[72150, 73129]	703	28	12	743
	C21	2008	[73130, 73854]	573	16	3	592
	C22	2008	[73855, 74463]	451	34	3	488
TEST	**C23**	2008	[74464, 75115]	490	24	8	522
	C24	2008	[75116, 75599]	362	14	12	388
	C25	2008	[75600, 76329]	570	19	22	611
	C26	2009	[76395, 78157]	1323	58	49	1430
	C27a	2009	[78296, 78884]	320	43	10	373
	C27b	2009	[78995, 79853]	513	70	59	642
	Total			***5890***	***353***	***219***	***6462***

Tabla 5.1: Descargas utilizadas para entrenamiento y test del predictor desarrollado.

Un subgrupo de la base de datos (parte sombreada de la *Tabla 5.1*) fue usado para el entrenamiento y el resto para el test del sistema de predicción. El subgrupo de entrenamiento contiene descargas desde las campañas C19 a la C22. Para las pruebas de robustez del predictor, se destinó el resto de descargas, de las campañas C23 a C27b para el test, detalladas también en la tabla.

5.2.2 Arquitectura del predictor de tiempo real

Para la extracción de los datos se utilizaron las herramientas y métodos descritos en [Rattá, 2008], [Murari, 2008], y por lo tanto, los vectores de

características quedan definidos por la desviación estándar del módulo de la transformada de Fourier (eliminando la componente continua), tomando ventanas de 32 ms de cada señal. Se eligió un tamaño de ventanas de 32 ms porque a la hora de calcular la transformada de Fourier el tiempo de cálculo es mucho menor para números que sean potencia de 2. Además, se considera que 30 ms es el tiempo mínimo de respuesta de los actuadores de JET. El sistema de predicción se diseñó de manera que analice simultáneamente la evolución conjunta de las señales seleccionadas, por lo tanto, es necesario que todas y cada una de ellas hayan sido adquiridas durante la ejecución del pulso. En el caso de que alguna de ellas tuviese alguna alteración grave, la descarga no es analizada. Para descartar las descargas que no son adecuadas para la fase de entrenamiento se ha realizado un estudio exhaustivo de todas ellas, comenzando por un análisis visual de la gráfica correspondiente a la señal de corriente del plasma, descartándose las descargas que tuviesen alguna anomalía en dicha señal. A continuación, se ha realizado un procedimiento automático de análisis de las señales: *Amplitud del modo bloqueado*, *Inductancia del Plasma* y *Densidad del Plasma*, descartando nuevamente las descargas que tuviesen alguna anomalía en su rango normal de valores.

Sin embargo, si las señales necesarias para este estudio se encuentran disponibles, los experimentos se incluyen en la base de datos para su análisis, incluso si alguna o varias de las mediciones de alguna señal contienen datos igual a cero. Esto se debe a que el sistema también debe ser capaz de trabajar con algunas mediciones ruidosas o poco fiables. Por motivos de seguridad, en el caso de ser aplicado en tiempo real, el sistema activaría una alarma ante la ausencia de alguna de las señales para detener la descarga.

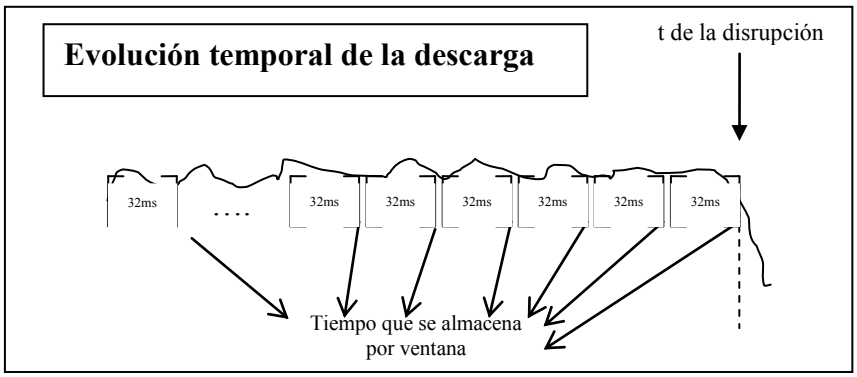


Figura 5.1: Evolución temporal de una descarga.

El objetivo primordial de los estudios sobre el fenómeno de la disrupción es el de evitarlo mediante una detección temprana. Para ello se deben analizar posibles anomalías en las magnitudes del plasma seleccionadas (inestabilidades MHD, incrementos abruptos en la radiación total, cambios inesperados en la corriente o inductancia del plasma, etc.) que puedan indicar comportamientos disruptivos. Además, los sistemas deben ser aplicables en tiempo real. La metodología de las investigaciones anteriores más relevantes sobre el tema (revisadas al inicio de este capítulo), se basa en el análisis secuencial de las ventanas temporales, tal como se obtendrían durante la ejecución de un experimento.

Para aclarar el concepto, en la *Figura 5.1* se puede observar cómo la total evolución temporal de un pulso, puede representarse como la concatenación de vectores de características, cada uno de ellos perteneciente a distintas ventanas temporales, representados por rectángulos. Con un único sistema de clasificación, se inspeccionan consecutivamente las ventanas de 32 ms de cada descarga en busca de posibles precursores disruptivos. Después de cada análisis, el predictor debe tomar una decisión: disparar una alarma o seguir analizando los siguientes 32 milisegundos. Si desde el comienzo hasta el final del experimento no se decide activar ninguna alarma, esto significa que el sistema no ha reconocido la descarga como disruptiva. En el caso contrario, si un evento disruptivo es identificado y la alarma es disparada, entonces el sistema ha detectado un comportamiento desencadenante de una disrupción.

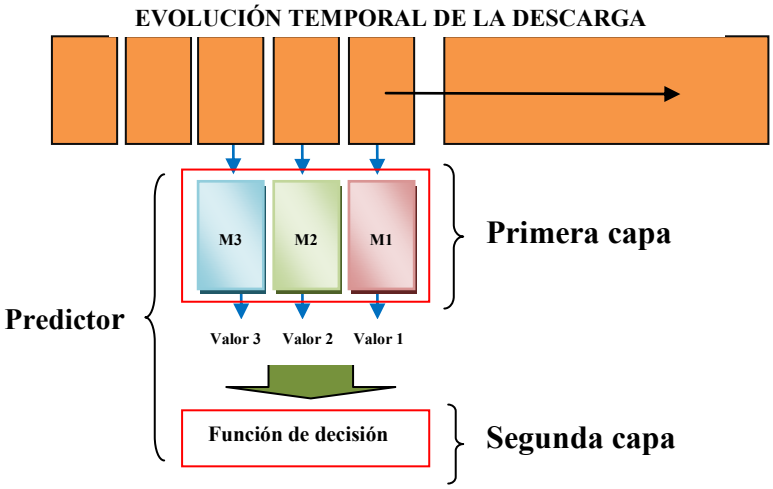


Figura 5.2: Estructura del sistema APODIS.

La falta de robustez de los clasificadores obtenidos en estudios anteriores, donde las tasas de acierto decrecen enormemente para campañas de medida diferentes a las de entrenamiento, puede interpretarse de la siguiente manera: los mecanismos que generan el fenómeno disruptivo son la consecuencia de una evolución de inestabilidades demasiado complejas y altamente no lineales como para ser detectadas satisfactoriamente con un único clasificador, independientemente de su longitud temporal. El sistema de predicción APODIS utiliza las tres últimas ventanas en el instante t de cada descarga como entrada a un segundo nivel, tal como se muestra en la *Figura 5.2*.

Para cada uno de los clasificadores que los modelos de una secuencia ejecutan paralelamente sobre ventanas de 32 ms consecutivos, se obtienen tres valores de salida. No siempre estos valores coinciden en el carácter disruptivo/no disruptivo de la predicción, ya que algunos pueden detectar un comportamiento anómalo y al mismo tiempo otros no. En consecuencia, es necesario implementar una función que evalúe los resultados y que mediante ellos decida si disparar o no una alarma. El desarrollo de esta función de decisión (FD) es crucial para alcanzar la mayor tasa de reconocimiento posible.

A continuación se describen los pasos seguidos por el sistema APODIS en el entrenamiento de cada una de sus capas:

1ª Capa: Cálculo de los modelos M1, M2 y M3

Se toma la primera de las descargas y se comienza desde la ventana número 23 (las 22 primeras ventanas se eliminan para evitar valores fuera de rango en las señales al comienzo de la descarga). La primera ventana obtenida corresponde con M1, la segunda con M2 y la tercera con M3, véase la *Figura 5.2*.

Se evalúan esas tres ventanas con los modelos de la primera capa (cada ventana con su modelo correspondiente) y se obtienen tres salidas con signo (distancias), denominadas respectivamente, V1, V2 y V3. El kernel utilizado en esta primera capa es el RBF.

Se utiliza la siguiente ***regla de decisión inicial*** como condición:

Si (V3 > -0.8 y V2 > -0.4 y V1 > 0)

Se dispara una alarma $\Rightarrow$ se tiene que escribir una fila en la tabla de Alarmas, equivale a una entrada disruptiva en la segunda capa (ver más adelante el formato de la tabla Alarmas).

Si no se cumple la condición:

Se escribe en la tabla de Alarmas cada 64 ms, equivale a una entrada no disruptiva en la segunda capa.

Los valores con los que se comparan las distancias V1, V2 y V3 (-0.8, -0.4 y 0) se han conservado del sistema APODIS original. Dependiendo del resultado de la ***regla de decisión inicial*** se pueden obtener los siguientes casos:

Caso 1) Alarma buena (a partir de una descarga disruptiva): El tiempo de alarma es anterior al tiempo de la disrupción sin exceder de 1 segundo. En este caso la etiqueta que se debe colocar en la tabla de Alarmas es un +1.

Caso 2) Alarma prematura (a partir de una descarga disruptiva): El tiempo de alarma es anterior al de la disrupción y excede al segundo. En este caso la etiqueta que se debe colocar en la tabla de Alarmas es un -1.

Caso 3) Falsa alarma (a partir de una descarga no disruptiva): se dispara una alarma en una descarga no disruptiva. En este caso la etiqueta que se debe colocar en la tabla de Alarmas es un -1.

Caso 4) No alarma: Cada 64 ms, es decir, cada dos ventanas se anota que no hay ningún tipo de alarma, -1. El objetivo de este apunte es reproducir la realidad desbalanceada de las descargas, ya que hay muchas más descargas no disruptivas que disruptivas. Esta anotación es una novedad respecto a la versión original del predictor.

Cuando se han analizado las tres primeras ventanas de una descarga, en el caso de que no se haya apuntado nada en la tabla de Alarmas, se avanza a la siguiente ventana y se repite el proceso anterior, véase la *Figura 5.3*.

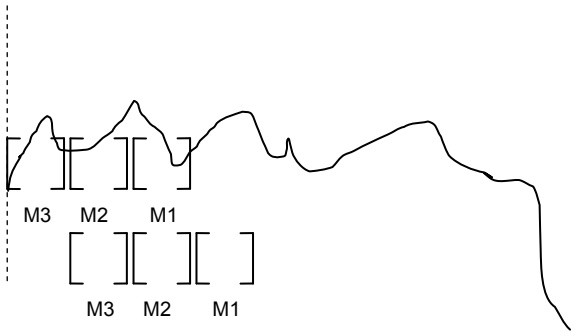


Figura 5.3: Evolución del análisis de una señal en la primera capa de APODIS.

Se continúa así por todas las ventanas de la descarga (hasta que se escriba una disrupción en la tabla de Alarmas o hasta que se llegue al final de la descarga analizada) y luego por todas las descargas. Cuando se terminan de analizar todas las descargas correspondientes al entrenamiento, se finaliza el primer nivel de entrenamiento. Esta parada tras el análisis de todas las descargas es una modificación del APODIS original, en el cual se analizaban las descargas continuamente hasta que no se mejoraba la tasa de test. En este trabajo, y con el fin de mejorar su rendimiento, se decidió, tras múltiples análisis con orígenes de datos diferentes, que volver a analizar todas las descargas una y otra vez, no mejora las tasas de acierto de los modelos generados, y sin embargo, ralentiza enormemente la generación de los modelos de predicción.

2ª Capa: Cálculo del modelo M

En la *Tabla 5.2* se muestran cuatro ejemplos correspondientes a los 4 casos en los que se tiene que escribir en la tabla de alarmas. Las celdas sombreadas son las que se utilizan para construir el clasificador lineal de la segunda capa. Con estos valores de entrada se calcula un modelo lineal que nos indicará si se produce disrupción o no en una descarga. De esta segunda capa se obtienen los coeficientes del hiperplano del modelo M que permiten decir si una descarga es disruptiva o no.

Descarga	Tiempo de Alarma (en seg)	Tiempo de la disrupción (en seg)	Margen (en mseg)	Salida de M1	Salida de M2	Salida de M3	Clase
56658	63.911	64.021	110	-0.910	0.448	0.0852	+1
53740	45.881	50.22	4339	-0.896	-1	0.445	-1
52461	40.916	0	-40916	-1.176	-1.202	-1.008	-1
74494	53.778	-	-	-2,526	-2,443	-3,698	-1

Tabla 5.2: Tabla de Alarmas.

5.3 Resultados de los clasificadores

En esta sección se detallan las pruebas realizadas a los sistemas entrenados. Éstas pueden ser divididas en tres etapas diferentes. En la primera, se analizan todas las descargas pertenecientes a las campañas de test diferentes, aunque consecutivas a las que se utilizaron para entrenar los modelos. Este tipo de test es muy diferente al que ha sido implementado en trabajos previos ya que casi todos los estudios testean con las propias campañas de entrenamiento. La segunda etapa de pruebas consiste en probar el predictor con descargas pertenecientes a las campañas C28 a C30 (C2830). Estos test son muy importantes, puesto que en esta campaña se ha modificado completamente la pared del dispositivo de fusión para hacerla similar a la que funcionará en el futuro dispositivo de ITER [Matthews, 2011]. En concreto, se ha pasado de una pared de carbono a una pared metálica.

Con estas dos primeras etapas de pruebas se pretende demostrar la robustez del sistema, pues se utilizan los modelos de campañas muy diferentes a las utilizadas en el entrenamiento. La tercera y última etapa consiste en comparar los resultados con los del sistema actualmente instalado en JET, el JPS (Jet Protection System).

5.3.1 Señales utilizadas en la detección de disrupciones

Las señales utilizadas para entrenar los modelos se han dividido en tres grupos A, B y C con el fin de poder comparar cuál es el mejor grupo de señales que pueden detectar una disrupción. El grupo A está formado por 7 señales, con un total de 14 componentes, ya que de cada señal se calcula la media y la desviación típica de la transformada de Fourier (excluyendo la componente continua) de los 32 datos de la ventana. Siguiendo este proceso el grupo B tiene

8 señales, con un total de 16 componentes y el grupo C tiene 12 señales, con un total de 24 componentes.

Todos los valores han sido normalizados entre 0 y 1 utilizando la siguiente expresión:

$$Señal\ normalizada = \frac{Entrada_{señal} - min}{max - min}$$

donde *min* y *max* representan, respectivamente, los valores mínimos y máximos de las señales con las que se está entrenando.

Modelo A **7 señales de tiempo real**	**Modelo B** **7 señales de tiempo real +** **1 señal calculada**	**Modelo C** **9 señales de tiempo real +** **3 señales calculadas**
Corriente del plasma Amplitud de modo bloqueado Inductancia del plasma Densidad del plasma Derivada temporal de la energía diamagnética Potencia radiada Potencia total de entrada	Corriente del plasma Amplitud de modo bloqueado Inductancia del plasma Densidad del plasma Derivada temporal de la energía diamagnética Potencia radiada Potencia total de entrada *Derivada temporal de la inductancia del plasma*	Corriente del plasma Amplitud de modo bloqueado Inductancia del plasma Densidad del plasma Derivada temporal de la energía diamagnética Potencia radiada Potencia total de entrada Beta poloidal Posición vertical del centroide del plasma *Derivada temporal de la inductancia del plasma* *Derivada temporal de la beta poloidal* *Derivada temporal de la posición vertical del centroide del plasma*
14 características (media y desviación)	**16 características (media y desviación)**	**24 características (media y desviación)**

Tabla 5.3: Grupos de señales utilizados para entrenar los modelos.

Para obtener los datos de entrada que permitan calcular los distintos modelos se han utilizado 125 descargas disruptivas y 100 descargas no disruptivas, escogidas aleatoriamente de las campañas de entrenamiento C19,

C20, C21 y C22. Teniendo en cuenta lo anterior, se han creado 50 grupos de entrenamiento de 225 descargas, en los que sólo difieren de un grupo a otro las descargas no disruptivas. Con estos conjuntos de entrenamiento se han creado los 50 modelos correspondientes, uno para grupo de descargas de entrenamiento. Cada modelo consta de tres modelos con kernel RBF: M1, M2 y M3 en la primera capa de APODIS, y un modelo M con kernel lineal en la segunda capa de APODIS.

Una vez obtenidos los 50 modelos, se han probado contra todas las descargas de test correspondientes a las campañas C23, C24, C25, C26, C27a y C27b. Tras comprobar los resultados, se ha extraído el mejor modelo de cada grupo de señales (Modelo A, Modelo B y Modelo C), cuyos resúmenes se muestran en el apartado siguiente.

Obtener todos estos cálculos ha sido posible gracias a la paralelización realizada de LIBSVM y a las modificaciones introducidas en el APODIS original. Éstas han permitido no sólo calcular los 50 modelos a partir de las 50 agrupaciones de 225 descargas obtenidas, sino que también se han realizado miles de ejecuciones con diferentes parámetros de entrada para los kernels utilizados en las dos capas que componen APODIS. En concreto, 1800 ejecuciones para cada una de las 50 agrupaciones, tal y como se puede apreciar en la Tabla 5.4.

APODIS	Variable	Mínimo	Intervalo	Máximo	TOTAL
Capa 1	RBF (gamma)	0.1	0.2	10	50
	C	100	10	10000000	6
Capa 2	C	100	10	10000000	6

Tabla 5.4: Búsqueda de parámetros óptimos para la obtención de los 50 modelos.

5.3.2 Resultados de test de las campañas C23 a C27b

En la *Tabla 5.4* se pueden ver las tasas de aciertos y de falsas alarmas que se han obtenido para los mejores modelos de los grupos de variables A, B y C. Analizando los resultados de la tabla se observa que el modelo A es el que mejores tasas de acierto obtiene en casi todas las campañas de test, sólo mejoradas por el modelo C en las campañas C24, C27a y C27b. El modelo B no

obtiene grandes tasas de acierto, pero sí unas muy bajas tasas de falsas alarmas. Este hecho hace pensar que la señal de la derivada de la inductancia es muy significativa a la hora de no detectar falsas alarmas, situación muy deseable.

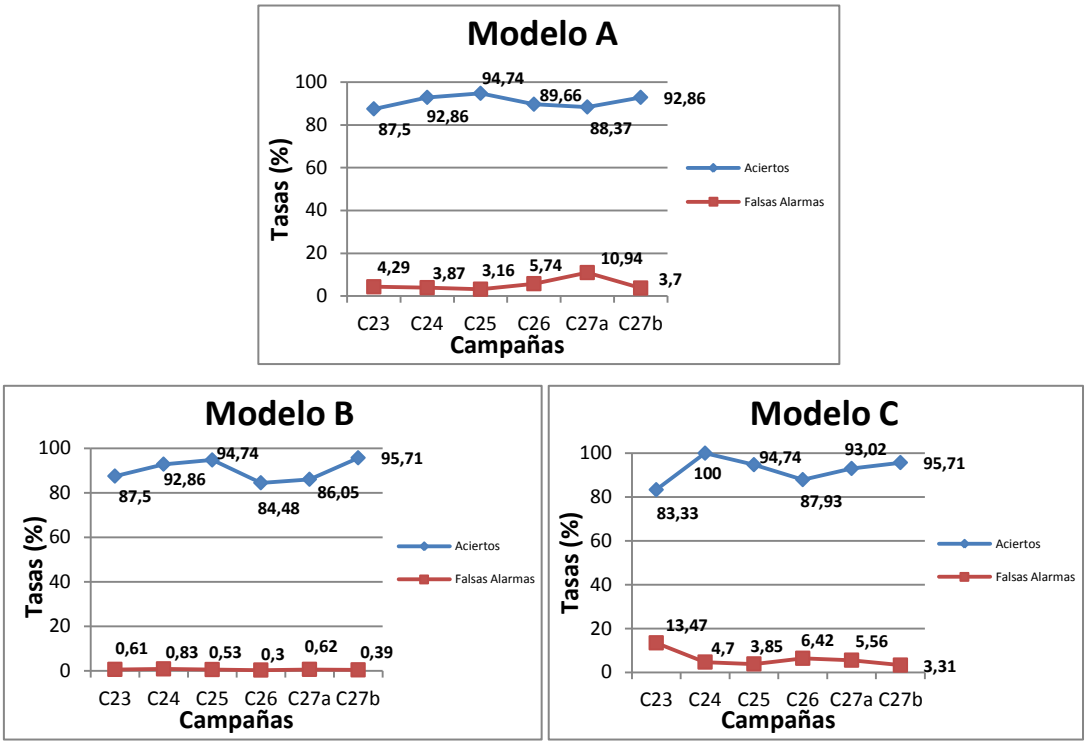


Figura 5.4: Resultados de test de los modelos A, B y C.

	Modelo A		Modelo B		Modelo C	
Campañas	**Aciertos**	**Falsas**	**Aciertos**	**Falsas**	**Aciertos**	**Falsas**
C23	87,50	4,29	87,50	0,61	83,33	13,47
C24	92,86	3,87	92,86	0,83	100,00	4,70
C25	94,74	3,16	94,74	0,53	94,74	3,85
C26	89,66	5,74	84,48	0,30	87,93	6,42
C27a	88,37	10,94	86,05	0,62	93,02	5,56
C27b	92,86	3,70	95,71	0,39	95,71	3,31

Tabla 5.5: Tasa de aciertos (%) y de falsas alarmas (%) de los modelos A, B y C.

Tras analizar los resultados se selecciona el modelo A como el mejor, ya que sólo utiliza 7 señales, frente a las 12 del modelo C. Y es que desde el punto de vista de la implementación de un sistema de detección en tiempo real, cuanto

menor sea el número de señales a utilizar, mucho más rápido será el sistema en sus respuestas ante posibles alarmas de disrupción.

5.3.3 Resultados de test de la campaña C2830

Combinando el modelo A, que es en el que mejores resultados se han obtenido, con las 100 primeras descargas disruptivas de la campaña C2830, en concreto desde la descarga 81852 hasta la 83730, y con 80 descargas no disruptivas escogidas aleatoriamente dentro del rango de descargas citado, se ha creado un nuevo modelo A' que posteriormente se ha testeado con las 101 siguientes descargas disruptivas de la campaña C2830, obteniendo como resultado un 96,04% de aciertos y un 2,23% de falsas alarmas, resultados que se presentan en la *Tabla 5.6*.

Estos datos vienen a reforzar la robustez del modelo A, ya que recuérdese que fue obtenido con datos de entrenamiento de las campañas C19 a C22, a las que ahora se han añadido datos de la campaña C2830. En C2830 se ha modificado completamente la pared de carbono del dispositivo JET, pasando a ser una pared metálica.

Entrenamiento	Test	Tasas de acierto	Tasas de falsas alarmas
225 descargas de C19 a C22 + 180 descargas (100 primeras disruptivas) de la C2830	101 descargas disruptivas + 358 no disruptivas Todas de la campaña C2830	***96,04%***	***2,23%***

Tabla 5.6: Resultados del modelo A' en la campaña C2830.

En [López, 2013] también se hace una prueba de test sobre algunas descargas de la campaña C2830 utilizando el modelo A. En este caso no se utilizan descargas de la citada campaña para entrenar, sino sólo para test. En concreto, las descargas utilizadas en este trabajo se encuentran entre la 80128 y la 81051 e incluyen 633 descargas no disruptivas, 33 disruptivas no intencionadas y 12 disruptivas intencionadas. Como resultado del test se obtiene un 96,55% de tasa de acierto y un 2,05% como tasa de falsas alarmas. Este dato nuevamente refuerza la robustez del modelo creado.

5.4 Comparación con otros sistemas de predicción

En el sistema APODIS original descrito por Rattá se presenta una gráfica de las tasas de acierto y falsas alarmas entre las distintas campañas de test, de la C8 a la C19. Allí se observa cómo a medida que uno se aleja de las campañas de entrenamiento los resultados empeoran, pasando de tasas de acierto del 90-95% a poco más del 70% de acierto. Algo parecido ocurre con las falsas alarmas, que también aumentan, pasando de ser menores del 5% a llegar hasta al 20% en la campaña C19. Todo esto parece lógico, ya que los datos de entrenamiento cada vez tienen menos relación con los datos de test, debido a que en cada campaña se hacen numerosos ajustes que hacen que los valores de las señales utilizadas se vean altamente modificados.

El rendimiento del predictor desarrollado también ha sido comparado con el sistema JPS (Jet Protection System). Para contrastar apropiadamente los resultados obtenidos por este sistema, es necesario tener en cuenta ciertas consideraciones preliminares. El sistema JPS interviene durante la ejecución de un experimento cada vez que identifica algún comportamiento disruptivo, desencadenando una serie de acciones de mitigación o apagado rápido del dispositivo de fusión. Tanto si el sistema está detectando correctamente un comportamiento disruptivo, como si el JPS erróneamente ha disparado una falsa alarma, se interviene inmediatamente sobre la descarga. No existen, por lo tanto, estadísticas sobre falsas alarmas, ya que cada alarma activada conlleva al apagado del experimento. Es más, en algunos casos las acciones de atenuación y de finalización rápida del pulso provocadas por alarmas erróneas pueden llevar a que descargas no disruptivas terminen en una disrupción [De Vries, 2009].

En el caso del JPS las tasas de aciertos máximas no llegan al 80%. Para el modelo generado en el sistema APODIS diseñado por Rattá los mejores porcentajes alcanzados son del 94% en las campañas más próximas a las utilizadas como entrenamiento, para luego caer hasta un 70% en la campaña final. Sin embargo, en el modelo creado en este trabajo se llega a un 94% de tasa de acierto, pero su principal cualidad es que mantiene las tasas de acierto estables, oscilando entre el 94% y el 87% en la peor campaña.

Finalmente, la gráfica de la *Figura 5.5* representa los porcentajes acumulados de las disrupciones detectadas (eje de ordenadas) para diferentes "tiempos de alarma" (abscisas en escala logarítmica). Estos tiempos de alarma (instante de la disrupción menos el instante en el que el sistema ejecutó la alarma) son de gran importancia, ya que definen el margen temporal de que dispone el sistema para realizar acciones de apagado o mitigación. Sólo se presentan las disrupciones no intencionadas (las que ocurren durante condiciones normales de operación). Se observa que el 90% de las disrupciones se detectan con un margen superior a 32 ms. Este dato es muy importante, ya que el tiempo mínimo de parada del dispositivo JET, tras la detección de una disrupción, es de 30 ms.

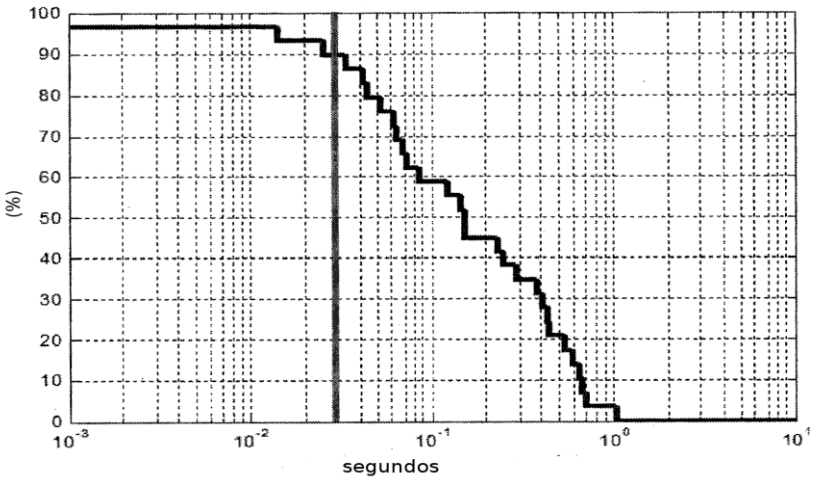


Figura 5.5: Tasa acumulada de aciertos frente a tiempo de detección en la C2830.

5.5 Consumo de recursos computacionales

Para obtener los resultados de los modelos presentados en este capítulo se crearon 50 conjuntos de entrenamiento en los que la única diferencia entre ellos son las descargas no disruptivas utilizadas para entrenar y, por lo tanto, cada conjunto siempre tiene las mismas descargas disruptivas. El motivo de crear estos conjuntos de entrenamiento es que las descargas no disruptivas son tan importantes como las disruptivas, ya que condicionan el resultado final. Un mal conjunto de descargas no disruptivas eleva considerablemente la tasa de falsas alarmas y hace que la tasa de aciertos baje, al inducir a error al proceso de clasificación.

Los resultados de rendimiento presentados en la gráfica de la *Figura 5.6* se han obtenido de modelos creados con una mezcla de descargas de las campañas C19 a C22 y descargas de la C2830. En total, las descargas utilizadas han sido 405 para la fase de entrenamiento, de ellas 225 disruptivas (125 de las campañas C19 a C22 y 100 de la campaña C2830) y 180 no disruptivas (100 de las campañas C19 a C22 y 80 de la campaña C2830). Para el test se han utilizado 457 descargas, todas de la campaña C2830, de ellas 101 son disruptivas no intencionadas y 356 no disruptivas diferentes de las utilizadas para entrenar.

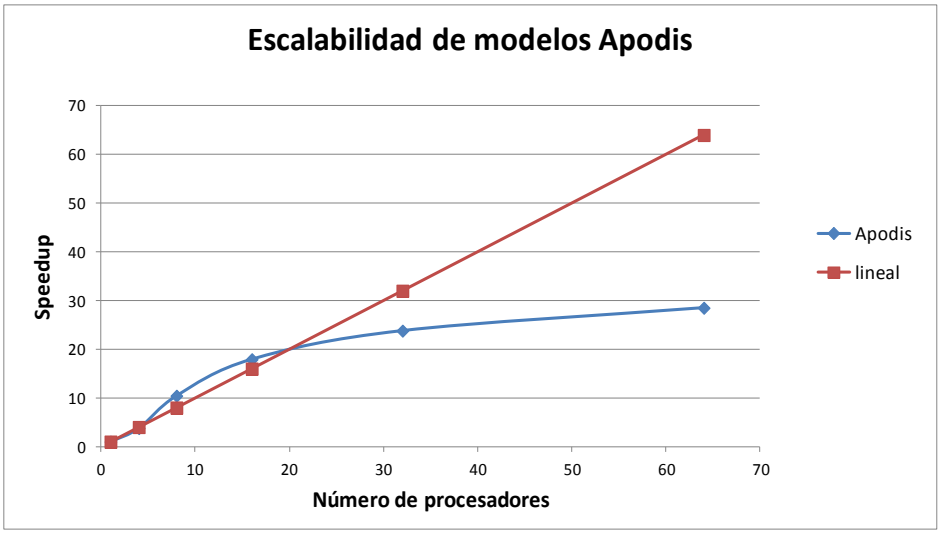


Figura 5.6: Aceleración de la aplicación Apodis paralelizada.

La gráfica muestra una evolución positiva para los casos de 4 procesadores y sobre todo para 8 y 16 procesadores. Este rendimiento tan positivo es debido a que los datos de entrenamiento y test han sido generados antes de iniciar la ejecución de dichos procesos. De nuevo, en los casos de 8 y 16 procesadores se produce una aceleración por encima de la lineal, al igual que ocurría con el SVM paralelo, ya que la división del conjunto de datos hace que el espacio de datos de entrada pueda alojarse completamente en la memoria cache, acelerando de esta forma la ejecución del proceso. A partir de 32 procesadores se aprecia un deterioro en la aceleración a causa del retardo que provocan las comunicaciones entre procesos, haciendo que aumente el tiempo de ejecución.

En la *Tabla 5.6* se detallan los tiempos de ejecución de la aplicación Apodis paralelizada. Del análisis de éstos se puede afirmar que la utilización de la programación paralela ha sido muy útil, permitiendo pasar de 14 días de ejecución para un solo procesador, hasta 11 horas de ejecución en 64 procesadores. La ventaja de esta ganancia de velocidad de proceso se ha utilizado para probar diferentes valores de los parámetros de entrada en la creación de los modelos de entrenamiento, y su posterior análisis con el conjunto de descargas de test. Se han realizado cientos de ejecuciones con diferentes valores del parámetro G del kernel RBF del primer nivel de Apodis y de diferentes valores de la variable C, tanto para el primer nivel, como para el segundo nivel de Apodis con el objetivo de encontrar los parámetros óptimos, los valores utilizados en la búsqueda son los indicados en la Tabla 5.4. En total, se han consumido más de 100000 horas de cálculo para obtener los datos presentados en este capítulo.

Procesadores	Tiempo (segundos)	Aceleración APODIS
1	1207899	1,00
4	319382	3,78
8	115787	10,43
16	67375	17,93
32	50764	23,79
64	42385	28,50

Tabla 5.7: Tiempos de ejecución de la aplicación Apodis paralelizada.

5.6 Conclusiones

En primer lugar, se debe destacar la alta tasa de aciertos de los modelos generados en todas las campañas utilizadas como test (C23, C24, C25, C26, C27a y C27), que han sido diferentes a las utilizadas como entrenamiento (C19, C20, C21 y C22). Asimismo, la tasa de falsas alarmas ha sido baja, menor del 5% en casi todas las campañas, incluso en el modelo B esta tasa ha sido casi nula, menor del 1% en todas las campañas de test.

En segundo lugar, se comprobó la robustez del sistema con un gran número de descargas correspondientes a la campaña más reciente de JET, la

C2830 (campaña con pared metálica en el dispositivo de fusión). Este tipo de pruebas tiene una gran importancia, ya que la falta de robustez es una de las mayores debilidades de los modelos desarrollados anteriormente en otros trabajos. Los resultados muestran claramente que las tasas de acierto se mantienen altas a pesar de los cambios estructurales de importancia que han sido efectuados sobre el dispositivo de fusión. Después de dichos cambios las tasas de acierto continúan siendo satisfactorias.

Finalmente, el rendimiento del sistema es comparado con el sistema implementado en JET desde hace muchos años, el JPS. Las tasas de reconocimiento, para tiempos de alarma de hasta ~200 ms, son considerablemente más altas para el método basado en la arquitectura bicapa de APODIS. Estos resultados son de gran interés, ya que el tiempo de respuesta del sistema de mitigación que controla la forma del plasma tiene un tiempo de ejecución típico de entre 30 ms y 200 ms.

CAPÍTULO 6

Predicción incremental

6.1 Introducción

En el capítulo anterior se realizó un estudio offline de las disrupciones producidas entre las campañas C23 y C27b, tomando como datos de entrenamiento los de las campañas desde la C19 a la C22, sin embargo, teniendo en cuenta que el dispositivo de fusión sufre constantes cambios de configuración en cada campaña, es recomendable que los modelos de previsión de disrupciones generados estén lo más adaptados posible a los datos que se producen en ese instante, ya que conforme se avanza en las diferentes campañas, los datos de entrenamiento pueden ser muy diferentes a los que se están obteniendo en ese instante, debido a cambios en la configuración del dispositivo, modificación de los sensores para la obtención de las señales, etc. También, con vistas al futuro dispositivo de fusión ITER, es necesario diseñar un sistema adaptable que permita calcular un nuevo modelo de predicción con el mínimo número de descargas, y siempre intentando obtener el máximo porcentaje de aciertos en la predicción de disrupciones y una baja tasa de falsas alarmas.

Por todo ello, se plantea la necesidad de crear un modelo de predicción que utilice datos de la campaña que se esté desarrollando en ese momento. Para ello, la primera incógnita que hay que resolver es determinar el número de descargas que son necesarias para obtener un modelo fiable, es decir, si es necesario tener en cuenta todas las descargas producidas desde el inicio de la campaña, o se podría utilizar una combinación de ellas, por ejemplo, grupos de descargas en las que aparezcan tantas descargas disruptivas como no

disruptivas, o bien grupos de descargas con un mayor número de descargas disruptivas que de no disruptivas. Como se ha estudiado en numerosos trabajos, el número de datos de entrenamiento es muy importante para obtener unos resultados que proporcionen márgenes de error reducidos [Raudys, 1991], [Wu, 2011], [Mingmin, 2008].

Con este objetivo se plantean una serie de experimentos que se denominarán "análisis partiendo de cero", los cuales se detallan a lo largo del capítulo y se presentan en [Dormido-Canto, 2013]. En primer lugar, conviene recalcar que en todos estos análisis se aplica la metodología APODIS, utilizada en el resto de experimentos y validaciones del capítulo anterior. Recuérdese que las hipótesis de APODIS, para el caso general, son las siguientes:

- Siempre se entrena con descargas no disruptivas y descargas disruptivas no intencionadas. Las descargas disruptivas intencionadas no se utilizan nunca para entrenar, debido a que se considera que no tienen precursores de comportamiento disruptivo durante la evolución de la misma.

- Una vez fijado el número de descargas para utilizar en el entrenamiento, se escogen de manera aleatoria las descargas no disruptivas.

- Como características no disruptivas (para el entrenamiento de M1, M2 y M3) se cogen todas las ventanas posibles, excluyendo las 22 primeras y aquéllas que estén dentro del último segundo antes del instante de la disrupción, para las descargas de comportamiento disruptivo. En el caso de las descargas no disruptivas sólo se eliminan las 22 primeras ventanas para evitar valores fuera de rango.

- Se detiene el algoritmo en la primera iteración.

- Para el entrenamiento del modelo M de segunda capa, se anotan en la tabla de alarmas, como característica de no alarma, las salidas de M1, M2 y M3 que no disparen alarma cada 64 ms. Este procedimiento se continúa hasta que se dispare una alarma o hasta que se esté dentro del último segundo antes de la disrupción.

6.2 Metodología de creación de modelos en ausencia de datos. Análisis offline

En una primera aproximación los "análisis partiendo de cero" se van a realizar teniendo en cuenta los grupos de señales de los modelos A y C de la *Tabla 5.3*. Por lo tanto, se crearán modelos con 14 y 24 variables, con el fin de comprobar cuál de ellos representará mejor el fenómeno de las disrupciones con pocos datos.

6.2.1 Descripción de la base de datos

El rango de descargas utilizado corresponde a las nuevas campañas experimentales de JET de la C28 a la C30 (C2830) donde se está haciendo uso de la nueva pared metálica del dispositivo de fusión. En la *Tabla 6.1* se muestra el rango de descargas, así como el número de disruptivas y no disruptivas seleccionadas después de un proceso bastante exhaustivo ya detallado en la Sección *5.2.2*.

Campañas	Rango	No disruptivas	Disruptivas No intencionadas	Disruptivas Intencionadas	Total
C28 a C30	[81852, 83793]	1036	201	56	1293

Tabla 6.1: Descargas de las campañas C2830 utilizadas en el análisis partiendo de cero**.**

Una vez seleccionadas las variables y el rango de descargas, se han diseñado numerosos experimentos con las distintas combinaciones de las descargas. Por una parte se ha considerado una aproximación desbalanceada, en el que las descargas no disruptivas de entrenamiento superan claramente en número a las disruptivas, tal y como ocurre en la realidad. Por lo general, en campañas anteriores a la C2830 la proporción entre descargas disruptivas y no disruptivas es de una descarga disruptiva, por cada 10 ó 20 no disruptivas. Sin embargo, tras la modificación del dispositivo de fusión en la campaña C2830 se han incrementado las descargas disruptivas, por lo que la proporción ha bajado a 1 descarga disruptiva por cada 5 no disruptivas, [De Vries, 2011], [De Vries, 2012].

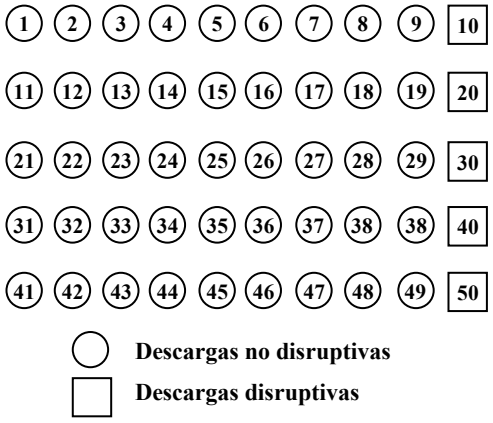
A continuación, se han creado modelos a partir de conjuntos de entrenamiento con un número equilibrado de descargas disruptivas y descargas

no disruptivas (aproximación balanceada). Finalmente, se ha optado por crear modelos con un método que se denominará híbrido, en el cual se fija el número de descargas no disruptivas y sólo se van considerando las nuevas descargas disruptivas que van apareciendo. En esta primera parte del análisis partiendo de cero se va a utilizar una metodología offline, ya que el conjunto de descargas de test, utilizadas para estudiar la viabilidad de los modelos creados, está formado por datos que se producen con posterioridad a las descargas utilizadas como entrenamiento.

6.2.2 Procedimiento desbalanceado

Para explicar el procedimiento que se ha implementado, se considera un ejemplo simplificado con el siguiente conjunto de 50 descargas: 45 no disruptivas y 5 disruptivas no intencionadas. Las cinco descargas disruptivas son la 10, la 20, la 30, la 40 y la 50.

Figura 6.1: Conjunto de descargas para el ejemplo simplificado.



En el procedimiento desbalanceado se intenta reproducir la frecuencia real de apariciones de descargas disruptivas/no disruptivas, teniendo en cuenta que estas últimas son las más numerosas. Por todo ello, los pasos seguidos han sido los siguientes:

- Cuando se llega a la descarga 10 se obtiene el primer modelo (A1). Para construir este modelo se utilizan 9 descargas no

disruptivas + 1 descarga disruptiva. Se evalúa el modelo con las descargas que van de la 11 a la 50.

- Cuando se llega a la descarga 20 se obtiene el segundo modelo (A2). Para este modelo se utilizan 18 descargas no disruptivas + 2 descargas disruptivas. Se evalúa el modelo A2 con las descargas que van de la 21 a la 50.

- Cuando se llega a la descarga 30 se obtiene el tercer modelo (A3). Para este modelo se utilizan 27 descargas no disruptivas + 3 descargas disruptivas. Se evalúa el modelo A3 con las descargas que van de la 31 a la 50.

- Se continúa el procedimiento hasta que se terminan las descargas disruptivas.

De una forma más detallada se puede resumir la selección de descargas para construir los diferentes modelos con el siguiente pseudocódigo:

*d(*0*)*=0*;*
*nd(*0*)*=1*;*
repetir *i*=1*:d*

$$M(i) \subset \left(\begin{array}{c} DD: \bigcup_{k=1}^{i} d(k) \\ NDD: \bigcup_{k=1}^{i} \{[nd(d(k-1)+1), nd(d(k)-i)]\} \end{array} \right)$$

fin repetir

donde

- *d* es el número de descargas disruptivas.
- *d*(*i*) es la posición de la descarga disruptiva *i* en la secuencia de todas las descargas (en el ejemplo simplificado de la *Figura 6.1* para *i*=1→*d*(1)=*10*, para *i*=2→*d*(*2*)=*20, …, i*=5→*d*(5)=50).
- *M*(*i*) es el modelo entrenado hasta la descarga disruptiva *i* (en el ejemplo simplificado *i*=1, 2, …, 5).

- *nd*(*i*) es la posición de la descarga no disruptiva *i* en la secuencia de todas las descargas *(en el ejemplo simplificado para i*=1→*nd*(1)=1, para *i*=2→*nd*(2)=2*, …, i*=45→*nd*(45)=49).
- *NDD*(*i*) es el número de descargas no disruptivas para el modelo *M*(*i*). Por ejemplo, en el modelo simplificado *M*(4), que se corresponde con un modelo formado por cuatro descargas disruptivas (10, 20, 30 y 40), *NDD*(4) tendrá las 36 descargas no disruptivas que se han producido hasta ese momento.

En la *Tabla 6.2* se muestran los cinco modelos de entrenamiento desbalanceados correspondientes al ejemplo simplificado.

Modelo de entrenamiento	***Descargas***	***i***	***NDD(i)***
M(1)	*DD:* {10} *NDD:* {[1, 9]}	1	9
M(2)	*DD:* {10, 20} *NDD:* {[1, 9] *U* [11, 19] }	2	18
M(3)	*DD:* {10, 20, 30} *NDD:* {[1, 9] *U* [11, 19] *U* [21, 29] }	3	27
M(4)	*DD:* {10, 20, 30, 40} *NDD:* {[1, 9] *U* [11, 19] *U* [21, 29] *U* [31, 39] }	4	36
M(5)	*DD:* {10, 20, 30, 40, 50} *NDD:* {[1, 9] *U* [11, 19] *U* [21, 29] *U* [31, 39] *U* [41, 49]}	5	45

Tabla 6.2: Modelos de entrenamiento desbalanceados para el ejemplo simplificado.

Como se señaló anteriormente, se va a realizar un análisis offline de los modelos creados, ya que las descargas utilizadas como test incluyen descargas que se producen con posterioridad a las utilizadas para crear los modelos. En el siguiente pseudocódigo se muestra de manera resumida la composición de los conjuntos de test:

```
repetir i = 1:d
        M(i) es testeado con [d(i)+1,end]
fin repetir
```

donde *end* representa el número total de descargas.

Para el caso del conjunto simplificado de test mostrado en la *Figura 6.1* los conjuntos de test serían los siguientes:

Modelos de entrenamiento	*Conjuntos de test*
M(1)	{[11,50]}
M(2)	{[21,50]}
M(3)	{[31,50]}
M(4)	{[41,50]}
M(5)	----------

Tabla 6.3: Conjuntos de test para el ejemplo simplificado.

Los modelos desbalanceados creados presentan como principal problema el desequilibrio entre los datos disruptivos y no disruptivos dentro del conjunto de entrenamiento, tal y como se demuestra empíricamente en [Nguyen, 2010]. Esto hace que sea muy difícil crear modelos con valores idénticos en los parámetros utilizados en el kernel de transformación, así como en la variable *C* que controla la separabilidad de los datos. Por otro lado, en [Japkowicz, 2002] se muestra cómo los conjuntos de datos que son linealmente separables no admiten ningún tipo de desbalanceo.

Por todo lo anterior, en la creación de los modelos utilizados en este método desbalanceado ha sido necesario hacer uso de diferentes valores de la variable *C,* para calcular el modelo lineal de la segunda capa de Apodis, en concreto los valores de *C* utilizados han sido *C*= {1000000, 5000000, 10000000, 100000000}. Esto, evidentemente, ha obligado a realizar cuatro cálculos diferentes de cada modelo en el segundo nivel de Apodis, lo cual ha requerido mucho más tiempo de proceso. Otro aspecto importante a la hora de trabajar con datos de entrada muy desbalanceados es que la métrica a utilizar para interpretar los resultados sea la adecuada [Weiss, 2003].

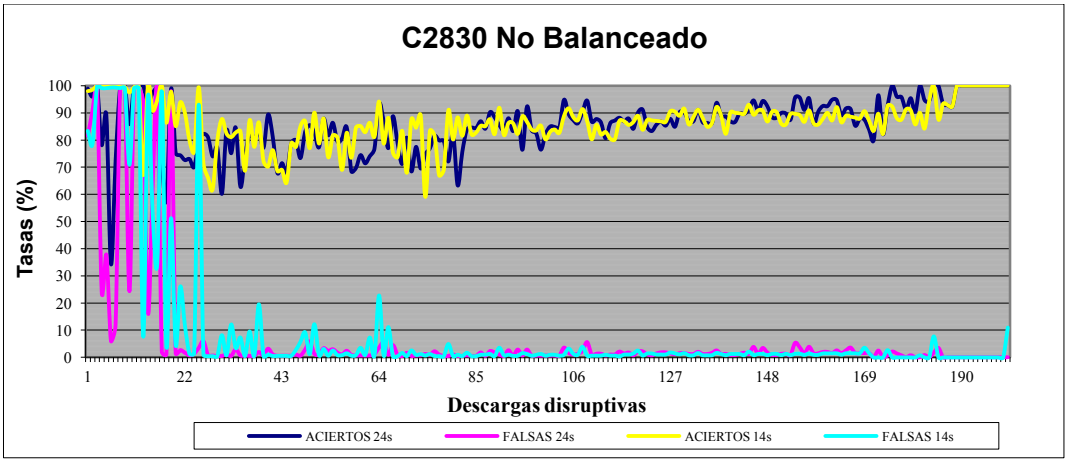


Figura 6.2: Resultados de los modelos desbalanceados.

La *Figura 6.2* muestra la predicción de disrupciones para la aproximación desbalanceada, tanto para 24 señales, como para 14 señales. El eje X representa el número de descargas disruptivas en cada modelo y el eje Y la tasa de aciertos y falsas alarmas. Por ejemplo, 106 significa que el modelo 106 se compone de las primeras 106 descargas disruptivas (en orden cronológico) y todas las descargas no disruptivas que se han producido hasta ese momento (en el presente caso, exactamente 678 descargas no disruptivas). Cada modelo se evalúa con el resto de las descargas producidas después de que se genera. Para el modelo 106, el conjunto de test está por tanto formado por 95 descargas disruptivas y 358 descargas no disruptivas.

Como se puede apreciar en la *Figura 6.2*, aparecen tres zonas muy diferenciadas: la primera de la descarga 1 a la 40, la tasa de falsas alarmas va decreciendo y estabilizándose, la de aciertos no crece pero sí se estabiliza. Entre la 40 y la 175 la tasa de falsas alarmas se mantiene fluctuando levemente, mientras que la de aciertos inicia una tendencia al alza aunque con fluctuaciones más elevadas. En los últimos modelos, a partir del 175, comienza a incrementarse la tasa de aciertos, debido al efecto frontera, ya que quedan muy pocas descargas disruptivas por analizar.

Los datos presentados en la *Tabla 6.4* han sido calculados a partir del modelo de 24 descargas disruptivas, con el objetivo de excluir la primera fase de alteraciones provocadas por la falta de suficientes descargas para crear un modelo estable. Los resultados muestran una alta tasa de aciertos y una baja tasa de falsas alarmas. Destacar también que los modelos de 24 señales proporcionan una tasa de falsas alarmas mucho menor que los modelos de 14 señales y una tasa de aciertos ligeramente superior.

	Tasa de aciertos (%)	**Tasa de falsas alarmas (%)**
Modelos de 24 señales	85.99±8.43	1.37±1.19
Modelos de 14 señales	85.66±7.78	2.27±3.06

Tabla 6.4: Media de aciertos y falsas alarmas junto con la desviación estándar de los modelos desbalanceados.

6.2.3 Procedimiento balanceado

En este apartado se muestran los resultados de los modelos creados con un número equilibrado de descargas disruptivas y descargas no disruptivas. El procedimiento se resume en los siguientes pasos:

- Cuando se llega a la primera descarga disruptiva no intencionada (en nuestro ejemplo la 10). En ese instante ya se puede obtener el primer modelo (llamémosle modelo A1). El modelo A1 estará compuesto por dos descargas (una no disruptiva y otra disruptiva no intencionada), es decir, la 10 como disruptiva no intencionada y una escogida aleatoriamente entre las 9 primeras como descarga no disruptiva. Para validar el modelo (A1) se utilizan las descargas que van de la 11 a la 50.

- Cuando se llega a la segunda descarga disruptiva no intencionada (en nuestro ejemplo la 20). En ese momento ya se puede obtener el segundo modelo (llamémosle modelo A2). Este modelo A2 estará compuesto por 4 descargas (2 no disruptivas y 2 disruptivas no intencionadas), es decir, como disruptivas no intencionadas serían la 10 y la 20 y como descargas no disruptivas se cogen 2 de forma aleatoria entre las 18 restantes. El modelo A2 se evalúa sólo con las descargas que van de la 21 a la 50.

- Cuando se llega a la tercera descarga disruptiva no intencionada (en nuestro ejemplo la 30). En ese momento ya se puede obtener el tercer modelo (llamémosle modelo A3). El modelo A3 estará compuesto por 6 descargas (3 no disruptivas y 3 disruptivas no intencionadas), es decir, como disruptivas no intencionadas serían la 10, la 20 y la 30, y como descargas no disruptivas se cogen 3 de forma aleatoria entre las 27 restantes. El modelo A3 se valida sólo con el grupo de descargas que van de la 31 a la 50.

Este procedimiento continúa hasta que se han terminado todas las descargas disruptivas no intencionadas. El número de descargas disruptivas no intencionadas disponibles indica el número de modelos (clasificadores)

entrenados. Al final, por cada modelo se dispondrá de tantas tasas como modelos se hayan entrenado.

Siguiendo la misma notación de los modelos no balaceados el pseudocódigo para la creación de los modelos desbalanceados se presenta a continuación:

d(*0*)=*0;*
nd(*0*)=1*;*
repetir *i*=1:*d*

$$M(i) \subset \begin{pmatrix} DD: \bigcup_{k=1}^{i} d(k) \\ i\ aleatorio\ NDD\ desde\ \bigcup_{k=1}^{i}\{[nd(d(k-1)+1), nd(d(k)-i)]\} \end{pmatrix}$$

fin repetir

En la *Tabla 6.5* se muestran los cinco modelos de entrenamiento balanceados correspondientes al modelo simplificado.

Modelo entrenado	***Descargas***	***i***	***NDD*(*i*)**
M(*1*)	*DD:* {10} *NDD:* 1 aleatoria entre {[1, 9]}	1	1
M(*2*)	*DD:* {10, 20} *NDD:* 2 aleatorias entre {[1, 9] *U* [11, 19] }	2	2
M(*3*)	*DD:* {10, 20, 30} *NDD:* 3 aleatorias entre {[1, 9] *U* [11, 19] *U* [21, 29] }	3	3
M(*4*)	*DD:* {10, 20, 30, 40} *NDD:* 4 aleatorias entre {[1, 9] *U* [11, 19] *U* [21, 29] *U* [31, 39] }	4	4
M(*5*)	*DD:* {10, 20, 30, 40, 50} *NDD:* 5 aleatorias entre {[1, 9] *U* [11, 19] *U* [21, 29] *U* [31, 39] *U* [41, 49]}	5	5

Tabla 6.5: Modelos de entrenamiento balanceados para el ejemplo simplificado.

Para este método, todos los modelos calculados han utilizado los mismos valores para los parámetros de creación de los modelos de primer y segundo nivel de Apodis, así como el mismo valor para la variable *C*.

La *Figura 6.3* muestra la predicción de disrupciones para la aproximación balanceada del conjunto de datos, tanto para 24 señales como para 14 señales. El eje X indica el número de descargas disruptivas en cada modelo. Por ejemplo, 106 representa el modelo 106, que se compone de las primeras 106 descargas disruptivas (en orden cronológico) y 106 descargas no disruptivas escogidas al azar a partir del conjunto de las descargas no disruptivas que se han producido

hasta el momento (en este caso, 106 descargas sin interrupciones han sido seleccionados de un total de 678). Como en el caso desbalanceado, cada clasificador se evalúa con el resto de las descargas producidas después de que se genera. En concreto, para el clasificador 106, el conjunto de test contiene un total de 95 descargas disruptivas y 358 descargas no disruptivas.

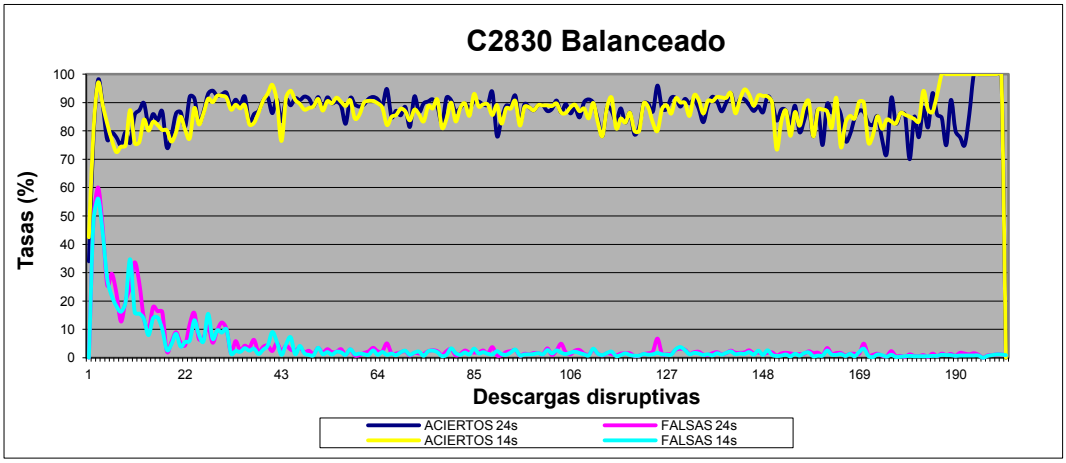


Figura 6.3: Resultados de los modelos balanceados.

Como se puede apreciar en la gráfica, al igual que en el caso desbalanceado, aparecen tres partes bien diferenciadas: la primera de la descarga 1 a la 45, aquí la tasa de falsas alarmas va decreciendo y estabilizándose y la de aciertos crece y también se estabiliza. Entre la 45 y la 175 las dos tasas permanecen fluctuando levemente, para a partir de la 175 comenzar a variar ampliamente la tasa de aciertos. Comparando ambos grupos de señales parece que los modelos de 14 señales se comportan mejor en la tercera fase de la gráfica, siendo mucho más estables.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	87.63±5.27	1.76±1.01
Modelos de 14 señales	88.80±5.41	1.48±0.98

Tabla 6.6: Media de aciertos y falsas alarmas junto con la desviación estándar de los modelos balanceados.

En la *Tabla 6.6* se muestran los datos medios de aciertos y falsas alarmas de los modelos balanceados, junto con la desviación estándar asociada, los datos han sido calculados desde el modelo 42, que es donde aproximadamente comienzan a estabilizarse las tasas de aciertos y errores. Comparándolos con

los del método desbalanceado se aprecia que son mejores, hasta casi un 3% mejora la tasa de aciertos para los modelos de 14 señales, disminuyendo a su vez en casi un punto el porcentaje de falsas alarmas.

6.2.4 Procedimiento híbrido

Como se vio en la sección anterior, los modelos balanceados estabilizan la tasa de falsas alarmas a partir de 40 descargas no disruptivas. De esta observación y persiguiendo el objetivo de utilizar el menor número de descargas posibles a la hora de generar los modelos, surgió la idea de generar modelos que se denominarán híbridos.

Los modelos analizados en esta sección se han creado, por tanto, fijando un número máximo de 40 descargas no disruptivas. A partir de ese número de descargas no disruptivas sólo se suman descargas disruptivas a los datos de entrada del conjunto de entrenamiento, es decir, los modelos tendrán un número variable de descargas disruptivas y un número fijo de no disruptivas. Por ejemplo, el modelo 106 en este caso se compone de las primeras 106 descargas disruptivas (en orden cronológico) y 40 descargas no disruptivas, escogidas al azar, a partir del conjunto de las descargas no disruptivas que se han producido hasta el momento (en este caso, 40 descargas no disruptivas han sido seleccionadas de un total de 678). Por otro lado, el conjunto de test resultante está formado por el resto de las descargas producidas después de que se genera la descarga 106. En concreto, para el clasificador 106, el conjunto de test contiene un total de 95 descargas disruptivas y 358 descargas no disruptivas.

Mantener invariable el número de descargas no disruptivas provoca que se fije el foco de interés de cada modelo en la detección de nuevas disrupciones, al estar las características no disruptivas claramente cubiertas con 40 descargas de dicho tipo.

Para este método híbrido, al igual que ocurría en el método balanceado, todos los modelos calculados han utilizado los mismos valores para los parámetros de creación de los modelos de primer nivel de Apodis, así como el

mismo valor para la variable *C* en los modelos de primer y segundo nivel de Apodis.

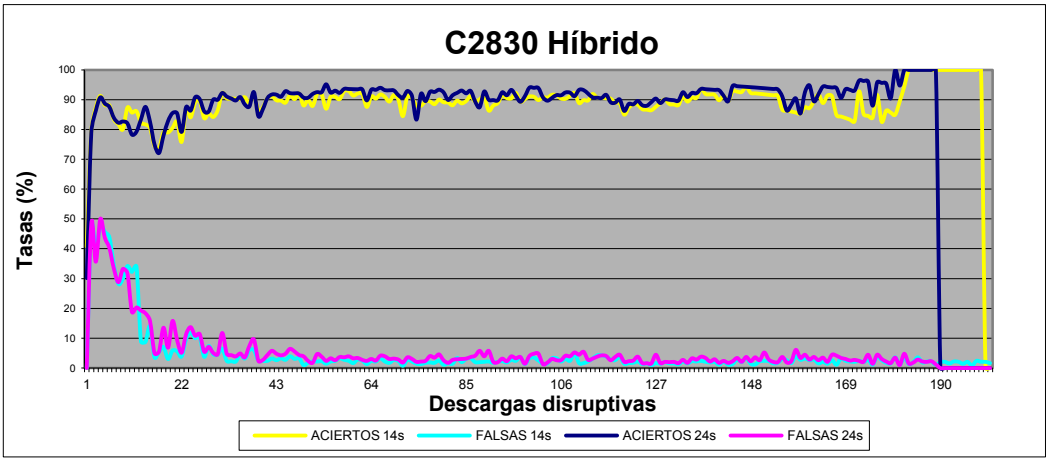


Figura 6.4: Resultados de los modelos híbridos.

En la *Figura 6.4* se muestran los resultados de test de los modelos híbridos creados. Se aprecian menos fluctuaciones, y por tanto una mayor estabilidad, tanto en la tasa de aciertos como en la de falsas alarmas. Una vez más se puede observar cómo aparecen tres partes diferenciadas en la gráfica: la primera de la descarga 1 a la 40, la tasa de falsas alarmas va decreciendo y estabilizándose y la de aciertos crece y también se estabiliza. Entre la 40 y la 180 las dos tasas permanecen fluctuando mínimamente, para a partir del modelo 180 comenzar el efecto frontera que hace, en los modelos de 24 señales, que la tasa de aciertos caiga a cero debido a que quedan muy pocas descargas disruptivas.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	92.40±2.91	3.14±1.14
Modelos de 14 señales	91.27±3.99	2.25±0.80

Tabla 6.7: Media de aciertos y falsas alarmas junto con la desviación estándar de los modelos híbridos.

De nuevo, al igual que para el caso balanceado, los datos presentados en la *Tabla 6.7* se han calculado desde el modelo 42 para evitar las fluctuaciones iniciales hasta que se estabilizan las tasas de acierto y falsas alarmas. Comparándolos con los del método balanceado se aprecia que son mejores, hasta casi un 5% mejora la tasa de aciertos para los modelos de 14 señales y un 2,5% para 24 señales, por el contrario, aumentan las tasas de falsas alarmas en

casi un punto porcentual, aunque globalmente se puede concluir que el método híbrido proporciona mejores resultados que el balanceado y al tener menos descargas de entrenamiento es más rápido de calcular.

6.2.5 Procedimiento híbrido mejorado

En este apartado se introduce una modificación en la aproximación híbrida con el fin de conseguir unas tasas de acierto más elevadas. Esta modificación tiene cierta analogía con el funcionamiento de las memorias caché de los ordenadores. En estos dispositivos de memoria lo que se intenta siempre es que estén en caché los datos que se han utilizado más recientemente porque suelen ser los que se van a utilizar en el futuro más próximo. Lo mismo ocurre con las disrupciones: a medida que van apareciendo las diferentes descargas se va teniendo conocimiento sobre los distintos tipos de disrupciones existentes. Algo similar ocurrirá con las descargas no disruptivas ya que el modelo será entrenado con descargas no disruptivas que serán similares a las que se puedan producir en un futuro cercano.

El procedimiento de cálculo de los modelos híbridos mejorados se resume en los siguientes pasos:

- Hasta la descarga 40 el procedimiento es exactamente el mismo que para la aproximación híbrida comentada en la sección anterior, ya que hasta que no se completan las 40 primeras descargas no disruptivas no se comienza a calcular el híbrido mejorado.
- A partir de la descarga 40 se calculan siempre dos modelos, uno con 40 descargas no disruptivas aleatorias extraídas de las que se hayan producido hasta ese momento, al que se llamará *Híbrido1*, y otro modelo con las 40 últimas que mejores resultados hayan proporcionado, al que se denominará *Híbrido2*.
- La función utilizada para decidir cuál es el mejor modelo es realizar la validación mediante el conjunto de test y quedarse con el que obtenga un valor más elevado en la siguiente expresión: **TASA ACIERTOS - TASA FALSAS**.

En la *Tabla 6.8* aparecen los modelos representados en la *Figura 6.5*, es decir, los que dan las diferencias de *tasas de aciertos-tasas de falsas alarmas* más altas.

	Hibrido 1	***Hibrido 2***
Modelos	*41,43:45,51,54,65,71,72,75, 78,80,82:84,112,116,117, 121,123,124,126,130,189*	*42,46:50,52,53,55:64,66:70,73,74,76,77, 79,81,85:111,113:115,118:120,122,125, 127:129,131:188,190:201*
Total utilizados	24	137

Tabla 6.8: Modelos utilizados en el procedimiento híbrido mejorado.

Como consecuencia de esto, las tasas de acierto serán algo más altas y sobre todo, mucho más estables, como se puede observar en la *Figura 6.5*. Sólo se representan los resultados correspondientes a los modelos de 24 señales para simplificar la gráfica y poder compararlos con los modelos del método híbrido normal. De nuevo aparecen las tres fases de los métodos anteriores, una inicial de estabilización hasta el modelo 40, otra de resultados estables hasta el modelo 180, y por último, una de frontera hasta el modelo 200.

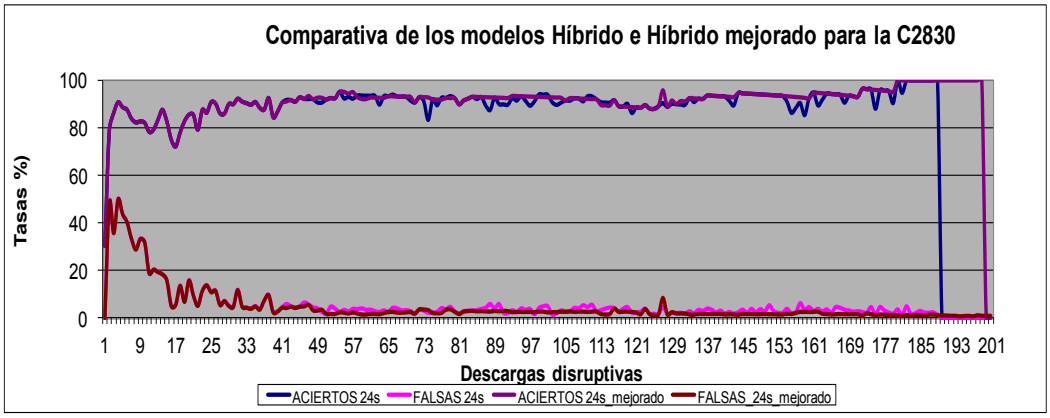


Figura 6.5: Comparativa entre el modelo híbrido y el híbrido mejorado.

Los datos presentados en la *Tabla 6.9* han sido calculados a partir del modelo 42 para evitar las alteraciones iniciales y también con el objetivo de poder compararlos con los del método híbrido normal. De los resultados obtenidos se puede concluir que el método híbrido mejorado es mejor que el híbrido normal, en concreto, la tasa de aciertos aumenta más de un punto porcentual mientras que la tasa de falsas alarmas baja en casi otro punto. El único inconveniente de este método es que se necesitan doblar los cálculos, ya

que por cada nueva descarga disruptiva se entrenan dos modelos y se realizan dos validaciones para quedarse finalmente con el mejor.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	93.64±2.48	2.21±0.98

Tabla 6.9: Media de aciertos y falsas alarmas junto con la desviación estándar de los modelos híbridos de 24 señales.

6.2.6 Modelos con tasas de acierto del 100%

Un funcionamiento ideal del sistema de detección de disrupciones sería que siempre se acertara al hacer una predicción de un evento disruptivo. Por este motivo es necesario dar respuesta a la siguiente cuestión: ¿Cuántos modelos sería necesario calcular para intentar llegar a esa situación ideal?

Para dar respuesta a esta pregunta se han analizado los modelos híbridos de 14 señales y se ha contabilizado el número de modelos necesarios, así como cuántas descargas disruptivas y no disruptivas es capaz de analizar con éxito cada modelo. En la *Figura 6.6* se puede ver cómo serían necesarios, al menos, 14 modelos diferentes durante las campañas C2830 para intentar mantener siempre el 100% de aciertos, evidentemente, con cada cambio se ha perdido esa tasa de aciertos, ya que la condición para calcular un nuevo modelo es que previamente exista un fallo en el modelo que está en uso en ese momento.

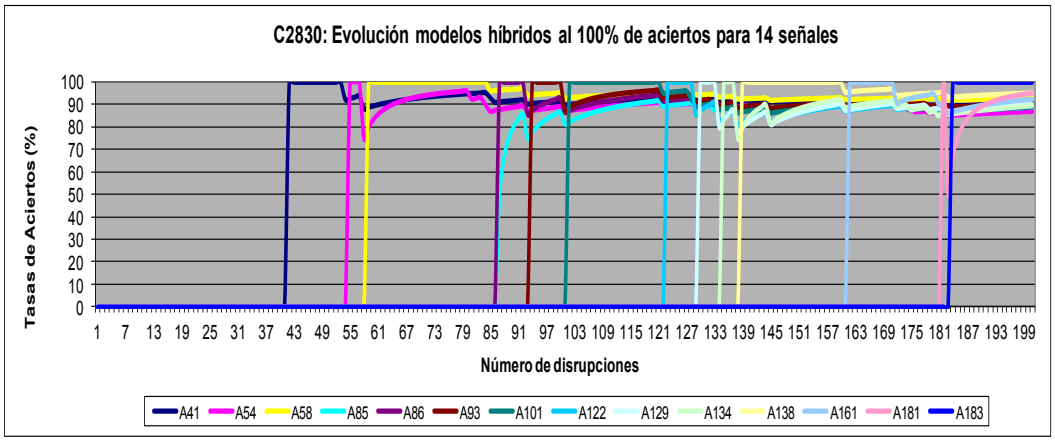


Figura 6.6: Evolución de la tasa de aciertos de modelos para conseguir el 100%.

En la *Tabla 6.10* se presentan los resultados estadísticos de los 14 modelos generados. Como puede verse, en el peor de los casos se ha tenido que cambiar de modelo después de una sola descarga, y como máximo se ha llegado a mantener un modelo durante 27 descargas disruptivas. De media cada modelo ha analizado correctamente 10,7 descargas disruptivas y 37,8 no disruptivas.

Modelo	41	54	58	85	86	93	101	122	129	134	138	161	172	181	183	PEOR	MEDIO	MEJOR
Disruptivas	13	4	27	1	7	8	21	7	5	4	23	11	9	2	19	1	10,73	27
No disruptivas	95	15	52	10	51	36	31	15	16	8	36	42	30	14	116	8	37,80	116

Tabla 6.10: Estadísticas de los modelos al 100%.

En esta sección se ha calculado un nuevo modelo cada vez que se pierde una descarga disruptiva, pero se podría tomar el criterio de calcular un nuevo modelo cuando se produce un fallo en la detección de una descarga disruptiva o cuando aparece una falsa alarma.

6.3 Resultados con datos de campañas con pared de carbono

Con el objetivo de analizar la robustez de los diferentes métodos de creación de modelos partiendo de cero, en este apartado, se muestran las tasas de aciertos y falsas alarmas de los modelos creados con campañas anteriores a la C28 para ver si la metodología es aplicable en campañas con pared de carbono.

Se han utilizado dos grupos de descargas, en el primer grupo, que se denominará CA1*,* se han utilizado descargas de las campañas de la C15 a la C21 para crear los modelos, en el segundo grupo, denominado CA2*,* se han utilizado descargas de las campañas C19 a la C22. En las *Tablas 6.11* y *6.12* se muestra el total de descargas utilizadas de cada campaña, así como sus rangos numéricos.

El grupo CA1 se ha escogido porque tiene un número de descargas disruptivas similar a las utilizadas en la campaña C2830 de la *Sección 6.2*, allí se hacía uso de 201 descargas disruptivas, mientras que en el CA1 las descargas disruptivas son 212. Sin embargo, existe una gran diferencia en las descargas no

disruptivas, siendo 1036 descargas no disruptivas para el rango de la campaña C2830 frente a las 3619 del conjunto CA1.

GRUPO CA1					
Campaña	**Rango**	**No disruptivas**	**Disruptivas No intencionadas**	**Disruptivas Intencionadas**	**Total**
C15a	[65985, 66355]	256	21	6	283
C15b	[66361, 66572]	158	10	0	168
C16	[66954, 66976]	10	3	0	13
C1617	[67641, 69148]	1032	73	0	1105
C18	[69227, 69795]	302	14	13	329
C19	[69806, 70750]	585	47	41	673
C20	[72150, 73129]	703	28	12	743
C21	[73130, 73854]	573	16	3	592
	TOTAL	**3619**	**212**	**75**	**3906**

Tabla 6.11: Grupo de campañas con pared de carbono CA1.

El conjunto CA2 se ha escogido por ser el grupo de descargas con las que se ha trabajado en el Capítulo 5, dentro del estudio de un modelo offline de predicción de disrupciones, y del cual se obtuvieron unos resultados excelentes. En este caso también existen grandes diferencias en el número de descargas disruptivas y no disruptivas con respecto a la campaña C2830.

GRUPO CA2					
Campaña	**Rango**	**No disruptivas**	**Disruptivas No intencionadas**	**Disruptivas Intencionadas**	**Total**
C19	[69806, 70750]	585	47	41	673
C20	[72150, 73129]	703	28	12	743
C21	[73130, 73854]	573	16	3	592
C22	[73855, 74463]	451	34	3	488
	TOTAL	**2312**	**125**	**59**	**2496**

Tabla 6.12: Grupo de campañas con pared de carbono CA2.

6.3.1 Modelos desbalanceados

Tanto en el grupo de descargas CA1, como en el CA2, se ha seguido el mismo procedimiento que en el grupo de descargas de la C2830. Se ha realizado un calibrado previo cada 20 descargas con valores del parámetro *gamma* del kernel RBF del primer nivel, entre 0.01 y 5, y con valores de la variable *C* entre 100000 y 10000000, tanto del primer nivel, como del segundo nivel de Apodis, no siendo posible obtener un conjunto de modelos con una buena tasa de aciertos y de falsas alarmas. En la *Figura 6.7* se muestran los resultados obtenidos para el grupo CA1 utilizando 14 variables de entrada.

Esta situación se produce por el gran desequilibrio existente entre las descargas disruptivas y las no disruptivas. Esta desproporción no era tan elevada para el caso de la campaña C2830, donde la proporción era de 1 disruptiva por cada cinco descargas no disruptivas. Sin embargo, en los conjuntos de descargas utilizados ahora la proporción es de una descarga disruptiva por cada 17 no disruptivas para CA1 y una descarga disruptiva por cada 18 no disruptivas para CA2.

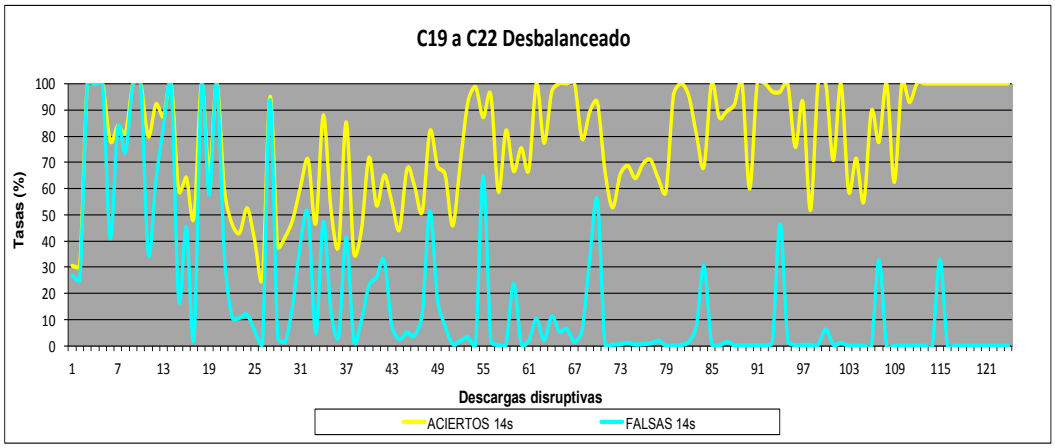


Figura 6.7: Modelos desbalanceados para la agrupación de descargas CA2.

6.3.2 Modelos balanceados

En la *Tabla 6.13* se presentan las tasas medias de los modelos calculados. En este caso son menores que las obtenidas con las descargas de la campaña C2830, lo cual es lógico ya que el grupo de descargas CA1 comprende ocho campañas diferentes a lo largo de las cuales el dispositivo de fusión ha sufrido cambios y ajustes considerables. Se debe recordar que el procedimiento partiendo de cero busca analizar cuántas descargas son necesarias para crear un modelo fiable una vez iniciada una nueva campaña, sin embargo, aquí se realiza un testeo a través de 8 campañas diferentes. Aun con estos condicionantes los resultados medios se mantienen por encima del 80%, lo cual pone de manifiesto una gran robustez en el método.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	83.50±6.71	5.25±4.54
Modelos de 14 señales	83.47±7.07	6.18±3.85

Tabla 6.13: Resultados medios y desviación estándar de los modelos balanceados del grupo de descargas CA1.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	81.24±11.91	5.61±3.57
Modelos de 14 señales	82.01±12.02	6.09±3.67

Tabla 6.14: Resultados medios y desviación estándar de los modelos balanceados del grupo de descargas CA2.

En los modelos balanceados representados en la *Figura 6.8* se observa una gran fluctuación en las tasas de aciertos y falsas alarmas, en este caso la primera fase, hasta el modelo 50, es mucho más estable que la fase central, del modelo 50 hasta el 170, que presenta altibajos importantes, sobre todo en la tasa de aciertos. En cuanto a la tasa de falsas alarmas, se produce una bajada alrededor del modelo 100.

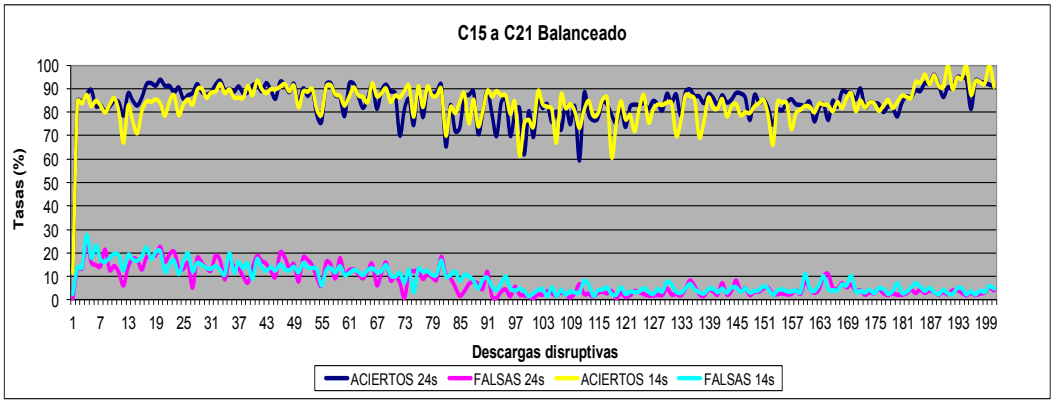


Figura 6.8: Modelos balanceados para el grupo de descargas CA1.

En la *Figura 6.9* se presentan los resultados de los modelos balanceados para el grupo de descargas CA2. De nuevo los resultados de los modelos presentan grandes fluctuaciones y los datos de la *Tabla 6.14*, aun siendo altos, muestran peores resultados que los obtenidos para la campaña C2830. Esto se debe a que se utilizan descargas de muchas campañas diferentes.

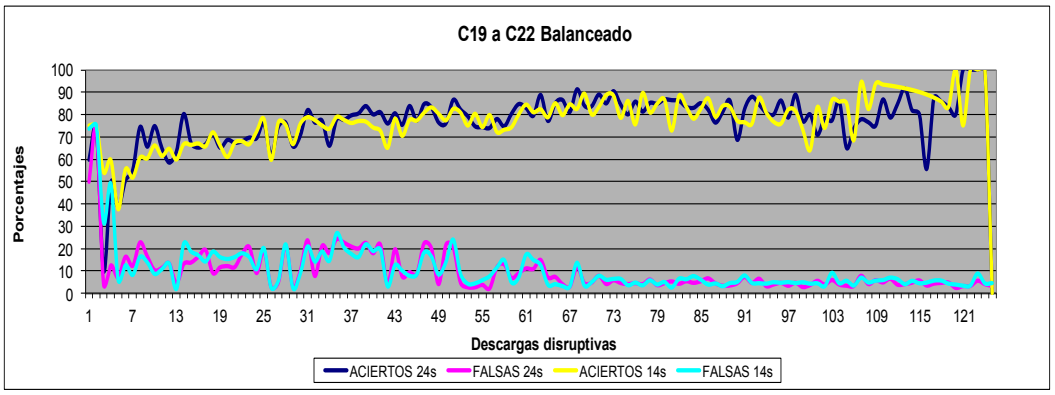


Figura 6.9: Modelos balanceados para el grupo de descargas CA2.

6.3.3 Modelos híbridos

En la *Figura 6.10* se presentan los datos correspondientes a los modelos híbridos del conjunto CA1. Las fluctuaciones son menores que para el método balanceado y las medias de acierto son muy superiores, al igual que ocurría con los modelos de la campaña C2830. Un dato a destacar es que aumentan las tasas de falsas alarmas, pasando de un 5% ó 6%, hasta un 9%, esto se debe a que el número de descargas no disruptivas en los modelos híbridos permanece fijo. En concreto, en este caso se han utilizado 50 descargas no disruptivas. Al utilizar pocas descargas disruptivas y ser tan amplio el número de descargas existentes de ese tipo a analizar, más de 3600, se produce un aumento de la tasa de falsas alarmas.

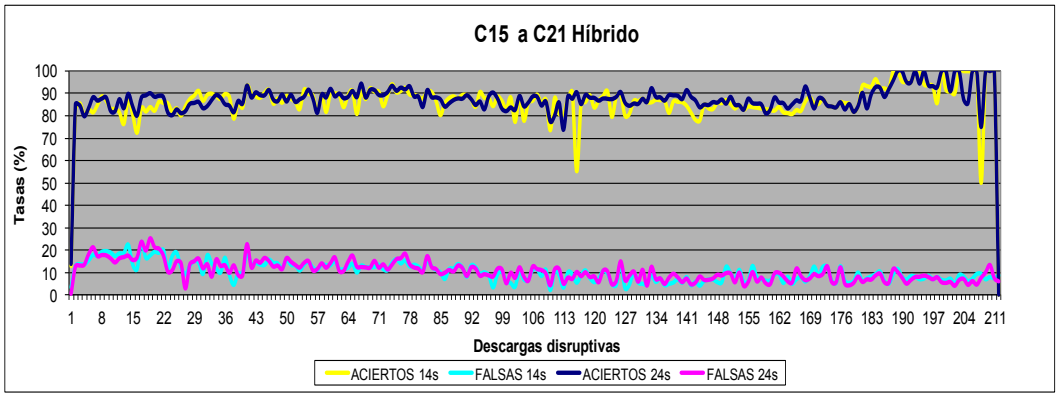


Figura 6.10: Modelos híbridos para las campañas C15 a C21.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	86.89±4.76	9.00±3.34
Modelos de 14 señales	87.96±6.49	9.43±3.25

Tabla 6.15: Resultados medios y desviación estándar de los modelos híbridos del grupo de descargas CA1.

En la *Tabla 6.15* se muestran los datos medios calculados para los modelos a partir del 50. Los resultados pueden considerarse buenos, ya que se están calculando modelos a lo largo de ocho campañas diferentes.

En la *Figura 6.11* y en la *Tabla 6.16* aparecen los resultados de los modelos correspondientes al conjunto CA2. Al igual que ocurre con el conjunto CA1, las medias de acierto son buenas, pero también aumentan las tasas de

falsas alarmas, pasando de un 5% ó 6%, hasta un 8%, debido al gran número de descargas no disruptivas que forman los conjuntos de test, frente a las 50 que poseen los modelos generados.

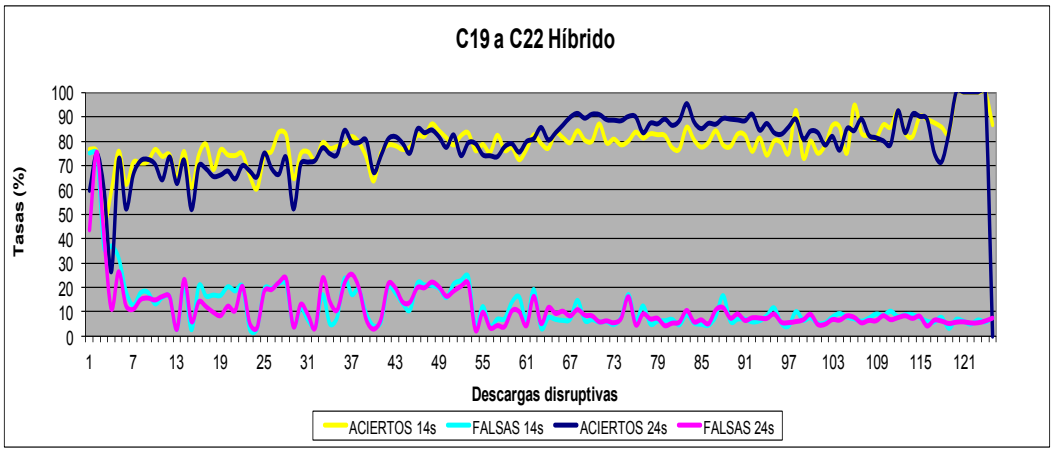


Figura 6.11: Modelos híbridos para las campañas C19 a C22.

	Tasa de aciertos (%)	Tasa de falsas alarmas (%)
Modelos de 24 señales	82.54±6.47	8.44±4.45
Modelos de 14 señales	84.20±6.56	8.06±3.70

Tabla 6.16: Resultados medios y desviación estándar de los modelos híbridos del grupo de descargas CA2.

6.4 Modelos en tiempo real

En este apartado se pretende analizar la viabilidad de los modelos creados para el caso offline, mediante una aproximación a lo que supondría su implantación en un sistema de tiempo real. Para ello se utilizan como conjuntos de test descargas que no se hayan producido con posterioridad a las utilizadas en los conjuntos de entrenamiento. Como los modelos creados utilizan todas las descargas disruptivas que se han producido hasta ese momento, sólo se pueden utilizar en los conjuntos de test las descargas no disruptivas que no han sido utilizadas para crear los modelos.

En la gráfica de la *Figura 6.9* se muestran los resultados obtenidos para la campaña C2830. Se observa cómo de nuevo en los modelos iniciales, hasta el modelo 20, las tasas de falsas alarmas presentan altas fluctuaciones, y a partir de ese momento se estabilizan en el entorno del 5%. Mediante la línea verde se

indica el porcentaje de descargas no disruptivas analizadas por cada modelo. La cantidad de descargas no disruptivas varía entre las 14 para el modelo 1 (1.5% de falsas alarmas) y las 944 (100% de falsas alarmas) del modelo 201.

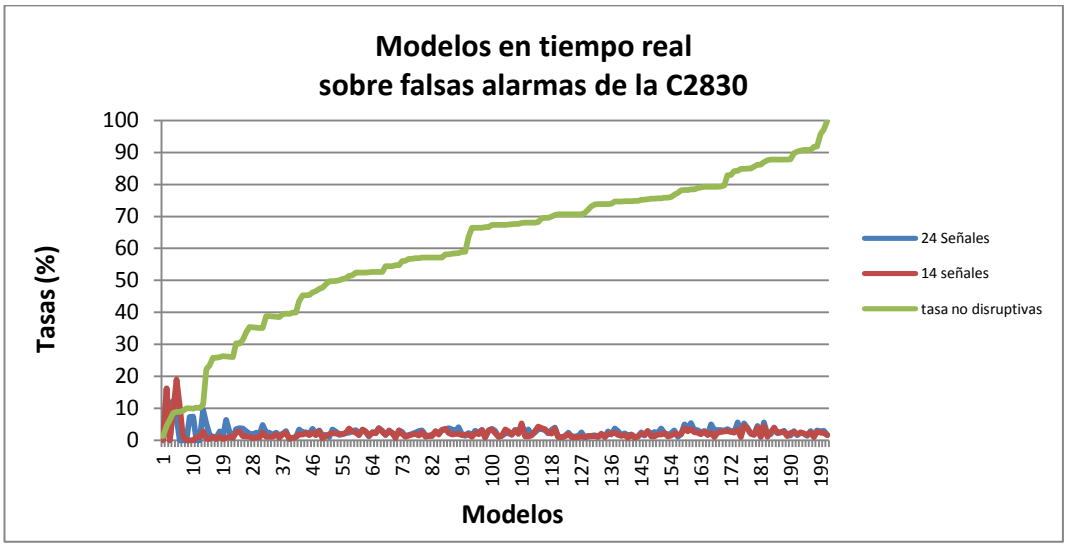


Figura 6.12: Modelos en tiempo real.

Como se ha mostrado a lo largo de la sección anterior, en todos los tipos de modelos creados, desbalanceado, balanceado y las dos versiones de híbridos, siempre se observa que una tasa baja de falsas alarmas se corresponde con una alta tasa de aciertos. Partiendo de este hecho y viendo los resultados de la *Tabla 6.17,* se puede concluir que los modelos pueden ofrecer grandes resultados aplicados en un entorno de tiempo real.

	Tasa de falsas alarmas (%)
Modelos de 24 señales	3.14±1.00
Modelos de 14 señales	2.25±0.89

Tabla 6.17: Tasa media de falsas alarmas y desviación estándar de los modelos híbridos en análisis de tiempo real.

6.5 Consumo de recursos computacionales

La creación de los diferentes modelos presentados en este capítulo ha requerido el uso de gran cantidad de recursos computacionales. La obtención de los modelos óptimos utilizando la técnica de máquinas de vectores soporte, requiere realizar un calibrado previo que permita conocer cuáles son los

parámetros más adecuados para construir los modelos. Este calibrado supone la ejecución de numerosos entrenamientos con diferentes valores en el parámetro *gamma* del kernel RBF para el primer nivel de Apodis, y de la variable *C* de separabilidad en los dos niveles de Apodis. Esto, unido a su ejecución con diferente número de descargas disruptivas, ha supuesto el consumo de miles de horas de proceso sólo en la búsqueda de los parámetros adecuados.

Una vez obtenidos los parámetros óptimos, se ha procedido a entrenar los modelos para todos los conjuntos de entrenamiento seleccionados. Como ejemplo de tiempo de proceso, en la *Figura 6.13* se representa la escalabilidad de la paralelización para los modelos balanceados, tanto de las campañas con pared de carbono C19 a C22 como de la campaña C2830 con pared metálica. Se puede observar que la aceleración obtenida es muy baja, esto es debido a que existe una gran parte secuencial en la aplicación, imposible de paralelizar tanto en el proceso de entrenamiento, como en el proceso de test. Esta parte secuencial se corresponde con los ficheros de entrada para construir cada uno de los modelos, cuyo proceso consiste en buscar los datos de cada una de las señales para cada descarga incluida en el conjunto de entrenamiento o de test, según sea el caso. Para el proceso asociado a la fase de entrenamiento se trata de la creación del conjunto de entrenamiento en función de las descargas disruptivas correspondientes de cada modelo, más las descargas no disruptivas que son extraídas al azar. En el caso del test, por la naturaleza intrínseca del proceso que obliga a pasar uno a uno todos los datos por el modelo. La pobre escalabilidad del proceso se acentúa a partir de 16 procesadores, observándose un casi nulo crecimiento en la aceleración, siendo por tanto 16 el número aconsejado de procesadores para realizar la ejecución de la aplicación.

	C19 a C22		C2830	
Procesadores	**Tiempo (segundos)**	**Speedup**	**Tiempo (segundos)**	**Speedup**
1	264415	1,00	125012	1,00
4	74957	3,53	41694	3,00
8	51636	5,12	29090	4,30
16	35841	7,38	23533	5,31
32	29113	9,08	28871	4,33

Tabla 6.18: Tiempos de ejecución de los modelos balanceados partiendo de cero.

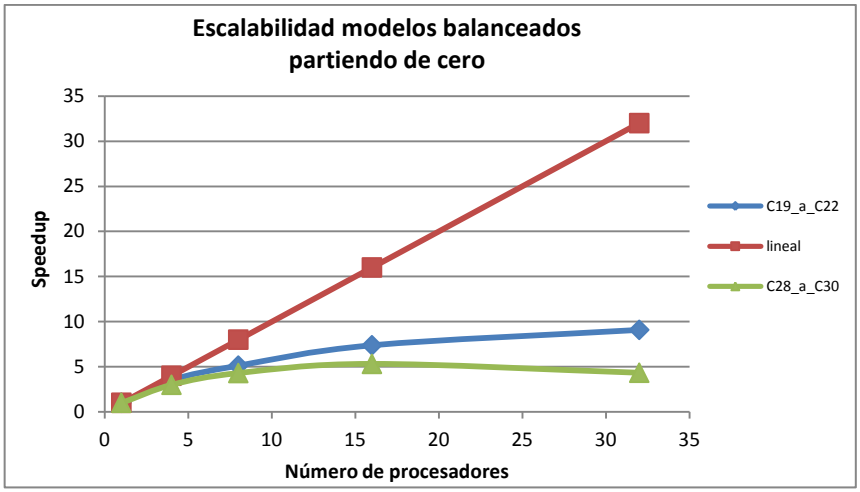


Figura 6.13: Escalabilidad de modelos balanceados partiendo de cero.

Con respecto al resto de métodos utilizados, comentar que el método que más recursos consume es el desbalanceado, debido a que necesita construir los modelos con todas las descargas producidas hasta ese momento. Esto hace que cuando se llega al cálculo de los modelos finales, el número de vectores de entrada para entrenar los modelos de primer nivel de Apodis puede llegar hasta los dos millones de vectores de entrenamiento, en el caso del conjunto CA2, y en el caso de CA1 hasta los tres millones.

Modelos desbalanceados conjunto CA2			
	Procesadores	**Tiempo (segundos)**	**Tiempo (días)**
Real	64	307087	3,55
Real	32	482933	5,59
Estimado	1	28556000	330

Tabla 6.19: Tiempos de ejecución de modelos desbalanceados partiendo de cero para CA2.

Como ejemplo de tiempo de ejecución, en la *Tabla 6.19* se muestran los tiempos de proceso para el método desbalanceado del conjunto CA2. Se presentan tiempos de ejecución para 32 y 64 procesadores. Asimismo, se ha realizado una estimación del tiempo que se tardaría en ejecutar dicho método en un solo procesador. El procedimiento de estimación ha consistido en lanzar el cálculo de los modelos con 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110 y 120 descargas disruptivas haciendo uso de un solo procesador y multiplicando el

resultado total de dichas ejecuciones por 10, ya que el conjunto CA2 está compuesto por 125 disrupciones.

Como se puede observar en la *Tabla 6.19*, habrían sido necesarios 330 días de cálculo ininterrumpido para obtener los modelos con un solo procesador, dato que justifica nuevamente la necesidad de hacer uso de la paralelización.

6.6 Conclusiones

Se comienza el capítulo planteando la posibilidad de obtener modelos a medida que se van produciendo descargas en las distintas campañas, y con el objetivo de conocer cuántas descargas son necesarias para crear un modelo fiable.

De los resultados de los modelos balanceados, desbalanceados e híbridos se puede concluir que, al menos, son necesarias 40 descargas disruptivas para obtener resultados en la tasa de aciertos superiores al 90%. También se puede afirmar que los modelos híbridos, en los que se fija el número máximo de descargas no disruptivas y a los que se les van añadiendo las descargas disruptivas que se van produciendo, son mejores que los desbalanceados. Siguiendo con ese tipo de modelos se ha creado una variante que duplica los cálculos a realizar, y evidentemente, el tiempo de proceso, pero que consigue una mayor estabilidad y unos mejores resultados medios. Es lo que se ha denominado *método híbrido mejorado*.

Seguidamente, se ha realizado una estadística de cuántos modelos sería necesario calcular para intentar mantener el 100% de acierto en las predicciones. Como se puede observar, no han sido necesarios más de 15 modelos por lo que su implementación en tiempo real es posible, ya que el tiempo medio de cálculo de cada modelo híbrido no supera los 300 segundos en un entorno de supercomputación con, al menos, 32 procesadores disponibles.

Por último, con el objetivo de simular la posibilidad de tiempo real, se ha realizado la fase de test de los modelos offline construidos para la campaña C2830 con descargas no disruptivas producidas con anterioridad a las descargas utilizadas para entrenar dichos modelos, obteniendo unos resultados excelentes

en las tasas de falsas alarmas, lo que hace esperar unos grandes resultados en la tasa de aciertos.

CAPÍTULO 7

Extracción de características

7.1 Introducción

En un sistema de aprendizaje supervisado no todas las características extraídas son necesarias o correctas a la hora de crear un modelo de predicción. Una selección de las mismas puede permitir:

- *Reducir el coste temporal/espacial*. Minimizando el número de características se necesitará menos memoria y menos tiempo de proceso.
- *Reducir el coste económico*. En este caso no se buscará el conjunto mínimo de características, sino aquel que minimice el coste económico total (por ejemplo, varias pruebas médicas baratas pueden interesar más que una única muy cara).
- *Facilitar y mejorar el proceso de clasificación*. Eliminando características ruidosas o redundantes, la tasa de aciertos o cualquier otro criterio de bondad del clasificador puede mejorar.

El proceso de selección se basa en encontrar un subconjunto de características que optimice un único criterio: minimizar el número de características, minimizar el coste económico, hacer máxima la tasa de aciertos del clasificador final o varios de ellos. En este último caso se necesitará una única función que englobe los diferentes criterios.

Si el criterio empleado en el algoritmo de selección es independiente del clasificador, el método se denomina *filtro*, si por otro lado, los resultados del proceso de aprendizaje del clasificador se tienen en cuenta en el criterio, se denomina *wrapper* [John, 1994]. Los *wrappers* obtienen en general mejores resultados frente a los filtros, que son más eficientes computacionalmente (un *wrapper* debe evaluar la precisión del clasificador por cada subconjunto explorado).

En el caso de la predicción de disrupciones, el proceso de aprendizaje depende de los resultados obtenidos por el clasificador. En este caso el modelo es obtenido mediante SVM. El objetivo será, partiendo de las 24 señales que se describen en la *Tabla 5.3* hacer grupos de variables y estudiar cuáles de ellas son las óptimas a utilizar, con el objetivo de maximizar la tasa de aciertos y disminuir la tasa de falsas alarmas.

Si se hace el estudio de manera exhaustiva, utilizando todos los conjuntos de variables posibles dentro de las diferentes agrupaciones, la cantidad de modelos a generar y de test a realizar vendría dado por la siguiente función [Rattá, 2012]:

$$\sum_{i=1}^{n} C_i^n$$

$$\text{siendo } C_i^n = \frac{n!}{(n-i)!i!}$$

donde

n es el número máximo de señales a utilizar

i es el número de señales de cada grupo, es decir, 1, 2, 3,…, 24.

Para el caso de estudio, serían necesarios 16777215 modelos con sus correspondientes fases de test. Suponiendo que cada conjunto de cálculo del modelo y fase de test necesitara un minuto, para la obtención de todos los modelos serían necesarios 31 años de cálculo ininterrumpido. Este dato es lo que ha motivado la implementación de una solución de cálculo más rápida, por lo que se ha hecho uso de los algoritmos genéticos.

7.2 Algoritmos Genéticos

Los algoritmos genéticos se presentan como una herramienta potente para la selección de características [Maimon, 2005]. Entre sus ventajas cabe destacar:

a) Exploran con eficacia grandes espacios de búsqueda.
b) No necesitan que la función criterio sea monótona.
c) Facilitan la optimización de funciones multicriterio.

Un algoritmo genético (AG) [Goldberg, 1989], [Mitchell, 1996] es un método de optimización aleatorio basado en los mecanismos de selección natural. Las posibles soluciones a un problema de optimización se representan mediante individuos artificiales que codifican los parámetros que se han de estimar. Estos individuos se codifican mediante unas estructuras (equivalentes al genotipo de los individuos biológicos), que contienen los parámetros (genes) agrupados en cadenas (cromosomas).

Estos algoritmos codifican las posibles soluciones en vectores binarios. Cada vector está compuesto de *N* componentes, siendo *N* el número total de características [Yang, 1998]. Las componentes de estos vectores representan la ausencia (valor 0) ó presencia (valor 1) de las distintas características, codificando así posibles selecciones. A los vectores se les denomina *individuos* y se les asocia un valor de *adaptación* que es función del criterio que se quiere optimizar. Este valor será mayor cuanto mejor solución represente el individuo.

El algoritmo genético parte de una población de individuos generados aleatoriamente, a partir de la cual el algoritmo va combinando los diferentes individuos y generando nuevas poblaciones. Todo esto se realiza siguiendo unas reglas de selección (los mejor adaptados tendrán más probabilidad de ser seleccionados), combinación y mutación. Tras un cierto número de generaciones el individuo mejor adaptado será la solución aportada por el algoritmo.

Los individuos serán seleccionados en función de la bondad de la solución que representan (en el caso de individuos biológicos se habla de

adaptación al entorno). Los individuos seleccionados serán combinados entre sí para dar lugar a nuevos individuos que representan las nuevas soluciones. El algoritmo va evolucionando, dando lugar a mejores soluciones. Junto a los mecanismos de selección y combinación se introduce un mecanismo de mutación que modifica a algunos de los individuos de manera aleatoria. Con esto se pretende enviar individuos aislados a explorar regiones alejadas de la zona de búsqueda actual, y así evitar que el algoritmo quede prematuramente atrapado en un mínimo local.

7.2.1 Codificación de individuos

Los individuos artificiales de un AG se representan mediante vectores numéricos que codifican en sus componentes los parámetros que se quieren estimar. En principio, es posible emplear cualquier base numérica para codificar las posibles soluciones, no obstante, el empleo de base 2 genera codificaciones de mayor longitud que albergan una mayor cantidad de *esquemas*.

Los *esquemas* son una abstracción y permiten dar una formulación matemática al modo de actuar de un AG. Esta formulación se conoce como *Teoría de los Esquemas* [Goldberg, 1989].

Como se ha comentado, la codificación de los individuos artificiales puede estructurarse en genes y cromosomas a semejanza de la información genética que codifica los individuos biológicos. Un gen artificial se representa mediante unos grupos de valores numéricos (bits, si empleamos base 2) que hacen referencia a un mismo parámetro. Un cromosoma artificial hace referencia a un conjunto de genes que codifican sendos parámetros con algún tipo de relación entre ellos. El conjunto de cromosomas artificiales conforma la estructura genética del individuo artificial. En general, aunque no es necesaria esta estructuración para el correcto funcionamiento de un AG, ésta suele definirse para facilitar la interpretación de la codificación elegida.

La implementación final de un individuo sería mediante un vector de números, un vector de bits si se trabaja en base 2. El modo de codificar los individuos para representar las posibles soluciones es uno de los puntos claves a

la hora de obtener buenos resultados con un AG. En la *Figura 7.1* se muestra un ejemplo de codificación en base 2.

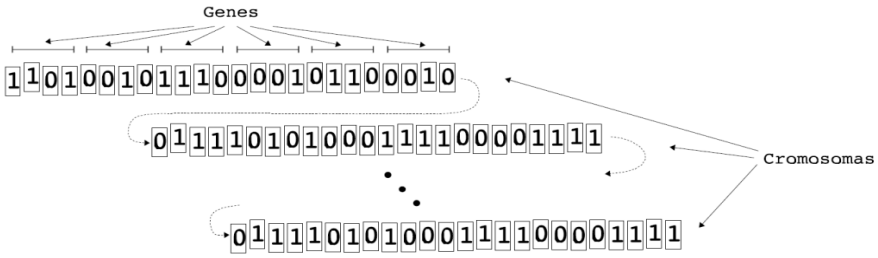


Figura 7.1: Codificación de un individuo empleando base 2.

7.2.2 Selección, combinación y mutación

La operación de *selección* serviría para extraer de la población (o generación) actual los individuos que participarían en la siguiente. Imitando la evolución natural, los individuos mejores serán los que tengan más probabilidad de ser seleccionados. El término *mejor* en el contexto de los AG vendría definido por la *función de adaptación*. Esta función determinaría la bondad de la solución que un determinado individuo representa, asignando una mayor probabilidad de selección a los individuos mejor adaptados. La función de adaptación se definiría a partir de la función que se desee optimizar. La única restricción que se le impone es que asigne valores mayores a las soluciones mejores (independientemente de lo que se quiera maximizar o minimizar).

La operación de combinación (*crossover*) se define sobre dos individuos de una población. Para ello, se selecciona aleatoriamente un lugar en la estructura de éstos (punto de combinación) y se intercambian los valores a partir de este punto. Como se puede ver en la *Figura 7.2,* es posible definir más de un punto de combinación en este tipo de operaciones. No obstante, la convergencia del AG no se ve afectada [De Jong, 1975]. Parte de los individuos seleccionados en cada generación serían combinados mediante esta operación. La intención es generar nuevos individuos (soluciones) a partir de los individuos seleccionados (que en su mayor parte serían los mejor adaptados). La tasa de combinación permite controlar el porcentaje de individuos que se combinarían en cada nueva generación.

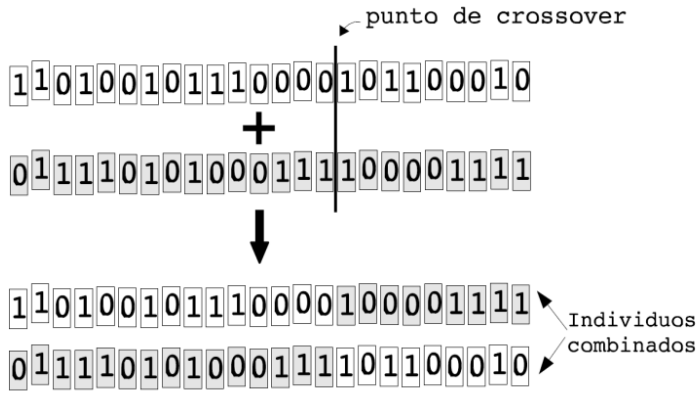


Figura 7.2: Proceso de combinación (*crossover*) de individuos.

Finalmente, la operación de *mutación* se define como la modificación aleatoria de alguno de los valores de la estructura de un individuo. La mutación será la última operación que sufra una nueva población, y sólo afectaría a algunos de los individuos. La *tasa de mutación* permite controlar el número de individuos sobre los que se aplicaría esta operación.

7.3 Métodos de extracción de características

El proceso de extracción de características consiste en recombinar las características disponibles para obtener un menor número de éstas [Hernández, 2005]. Con esto se pretende:

- Obtener una codificación más compacta de los objetos que se van a clasificar.
- Facilitar y mejorar la tarea del clasificador.

Es importante recordar que con la extracción no se reduce el coste asociado a la adquisición de las características que se tienen que obtener (esto sí ocurre en un proceso de selección). Lo que se hace es usarlas recombinadas por las razones expuestas anteriormente.

Los métodos de extracción de características se pueden clasificar en [Fukunaga, 1990]:

- *Lineales y no lineales*, en función de que las nuevas características se obtengan mediante combinaciones lineales o no lineales de las originales.

- *Supervisados y no supervisados*, en función de que se tenga, o no se tenga en cuenta, la clase de los diferentes objetos para realizar la transformación.

- *Aplicados al análisis de datos (mapping) y aplicados a la clasificación,* en función de la finalidad de las nuevas características extraídas. En el análisis de datos el nuevo espacio tendría dimensión 2 ó 3 con el objeto de poder ser analizado a simple vista. Los métodos orientados a la clasificación no se limitan al plano o al espacio, sino que pretenden obtener las ventajas ya comentadas.

7.4 Paralelización de algoritmos genéticos

La paralelización de algoritmos genéticos persigue reducir la cantidad de recursos computacionales utilizados en la ejecución de un problema de programación evolutiva. En esta sección se analizarán los distintos métodos de paralelización de algoritmos genéticos.

7.4.1 Clasificación de los AGs paralelos

Existen varias formas de paralelizar un algoritmo genético. La *Figura 7.3* muestra esquemáticamente los distintos métodos de paralización de algoritmos genéticos. La primera forma, y la más intuitiva, es la global, que consiste básicamente en paralelizar la evaluación de los individuos manteniendo una sola población. Otra forma de paralelización global consiste en realizar una ejecución de distintos AG secuenciales simultáneamente. El resto de aproximaciones dividen la población en subpoblaciones que evolucionan por separado e intercambian individuos cada cierto número de generaciones. Si las poblaciones son pocas y grandes, se tiene la paralelización de grano grueso. Si el número de poblaciones es grande y con pocos individuos en cada población, se tiene la

paralelización de grano fino. Por último, existen algoritmos que mezclan propiedades de estos dos últimos y son denominados mixtos [Hidalgo, 2001].

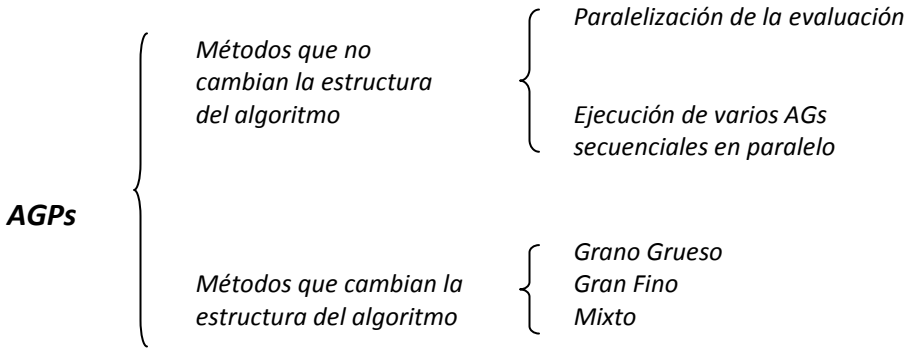


Figura 7.3: Clasificación de los AGPs.

En los AG paralelos de este último tipo, las subpoblaciones van evolucionando por separado para detenerse en un momento determinado e intercambiar los mejores individuos entre ellas. Se dispone de tres factores importantes que determinan la eficiencia de un algoritmo genético paralelo [Cantú Paz, 2000]:

- La topología de la comunicación entre los procesadores.
- La proporción de intercambio: el número de elementos que se intercambian en cada ocasión.
- La frecuencia de intercambio: periodicidad con que se intercambian los individuos.

7.4.2 Paralelización global

La paralelización global es la forma más sencilla de implementar un algoritmo genético paralelo. Consiste en paralelizar la evaluación de los individuos o en realizar una ejecución simultánea de distintos AGs secuenciales. Es muy útil, ya que permite obtener mejoras en el rendimiento con respecto al AG secuencial muy fácilmente, sin cambiar la estructura principal de éste. En la mayoría de las aplicaciones de los AGs la parte que consume un mayor tiempo de cálculo es la evaluación de la función de coste. En estos casos se puede ahorrar mucho tiempo de cálculo simplemente encargando la evaluación de una parte de la población a distintos procesadores y de una forma simultánea. Para

problemas simples con tiempos de ejecución cortos, los Algoritmos Genéticos Paralelos (AGP) globales no son una buena opción para mejorar el rendimiento, pero para problemas con tiempos de ejecución elevados consiguen una mejora sustancial.

Normalmente, la evaluación de un individuo cuesta exactamente igual para todos ellos y el tiempo de cálculo se puede disminuir aproximadamente en *N* veces, siendo *N* el número de procesadores. Evidentemente, el coste de las comunicaciones reducirá el rendimiento, y cuanto más sencilla sea la información a enviar mayor será éste.

A continuación se muestra el pseudocódigo de un algoritmo genético en el que se ha paralelizado la evaluación de la función de coste:

```
Generación de la población inicial
Mientras no se cumpla la condición de parada hacer
    Hacer en paralelo
        Evaluación de los individuos
    Fin hacer en paralelo
    Selección
    Producir nuevos individuos
    Mutación
Fin mientras
```

Figura 7.4: Pseudocódigo de una función de coste paralelizada.

También habrá que tener en cuenta el tipo de arquitectura que se esté utilizando y su facilidad para transmitir un tipo de datos u otro. Una consideración importante a la hora de implementar un AGP global es que si se realizan muchas comunicaciones, se puede perder cualquier reducción del rendimiento alcanzada mediante la paralelización.

La aplicación de los operadores puede mantenerse global o realizarse en paralelo, aunque generalmente el coste de las comunicaciones necesario para paralelizar estas operaciones no compensa el tiempo de cómputo ahorrado.
Dentro de este tipo de paralelización se tienen dos tipos de implementaciones: la *síncrona*, que se produce cuando el programa se para y espera el resultado de la evaluación antes de proceder a crear la siguiente generación, teniendo así las

mismas características que un algoritmo secuencial; y la *asíncrona*, en la que el procesador maestro no espera la llegada de todas las evaluaciones, siendo ésta la forma de implementación más usual debido a su facilidad en la realización.

```
Hacer en paralelo
    Generación de la población inicial
    Mientras no se cumpla la condición de parada hacer
        Evaluación de los individuos
        Selección
        Producir nuevos individuos
        Mutación
    Fin mientras
Fin hacer en paralelo
Escoger la mejor solución
```

Figura 7.5: Pseudocódigo de una función de coste paralelizada evaluando la mejor solución.

En la *Figura 7.5* se muestra el pseudocódigo correspondiente a otra forma de paralelización global, que consiste únicamente en enviar varios algoritmos a distintos procesadores y al final del proceso ver la mejor solución. El resultado es el mismo que si se ejecutasen varios AGs secuenciales y se escogiera la mejor solución de todas las obtenidas.

Por último, comentar que el modelo de paralelización global no hace ninguna distinción acerca de la arquitectura del computador sobre el que se está ejecutando. Se puede implementar tanto en un sistema de memoria compartida como de memoria distribuida.

En un computador de memoria distribuida la población normalmente se almacena en un procesador (maestro) que se encarga de enviar los individuos al resto de procesadores (esclavos), de recoger la información y de aplicar los operadores.

7.4.2.1 AGP de Grano grueso

Las características fundamentales de un AGP de grano grueso son la utilización de varias subpoblaciones relativamente grandes y la migración, entendiendo ésta como el intercambio de individuos entre distintas subpoblaciones. Este tipo de paralelización es el más utilizado debido a:

- La facilidad de implementación: sólo hay que tomar un conjunto de AGs secuenciales, ponerlos en distintos procesadores y cada cierto número de generaciones intercambiar individuos. La mayoría del código de la versión secuencial queda exactamente igual después de paralelizar.
- La disponibilidad de los computadores paralelos en los centros de investigación es habitual, aunque también son fácilmente simulables mediante software como MPI o PVM.

Los AGPs de grano grueso se suelen implementar sobre máquinas de memoria distribuida.

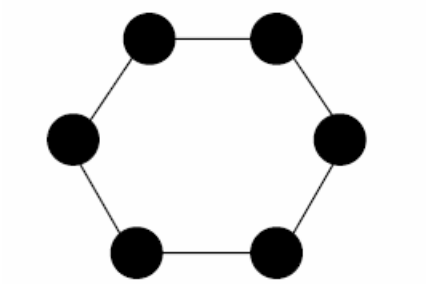


Figura 7.6: Esquema general de un AGP de grano grueso.

Los AGPs de grano grueso simulan el aislamiento geográfico de las diferentes civilizaciones y el intercambio esporádico de características que se realiza con la emigración. A continuación se muestra el pseudocódigo de un esquema de un AGP de grano grueso:

```
Inicializar P subpoblaciones con N individuos cada una
Número de generación = 1
Mientras (no se cumpla la condición de fin) hacer
        Para cada subpoblación en paralelo
            Evaluar y seleccionar individuos por su función de coste
            Si (número de generaciones módulo frecuencia) = 0 entonces
                Enviar K<N mejores individuos a población vecina
                Recibir K mejores individuos de población vecina
                Reemplazar K individuos de la población
            Fin si
            Producir nuevos individuos
            Aplicar operador de mutación
        Fin realizar en paralelo
        Incrementar número de generación
Fin mientras
```

Figura 7.7: Pseudocódigo de un AGP de grano grueso.

Topologías de comunicación

La topología de comunicación es un factor fundamental en el rendimiento de un AGP, ya que determina la velocidad con que una buena solución se propaga de una subpoblación al resto. Si la topología tiene muchas conexiones entre las subpoblaciones, las buenas soluciones se transmiten rápidamente de una población a otra. Por el contrario, si las poblaciones tienen poca comunicación entre ellas, las soluciones se extienden más lentamente, permitiendo la aparición de varias soluciones y una evolución más aislada de cada grupo. Otro factor en el que interviene la topología es el coste de comunicaciones. Es necesario buscar un compromiso entre la topología elegida y el coste de comunicaciones, para no perder las ventajas obtenidas sobre el rendimiento con la paralelización. La norma general es utilizar una topología estática que se mantiene constante a lo largo de toda la ejecución del algoritmo. Una segunda opción es la implementación de una topología dinámica. En ellas el intercambio entre subpoblaciones se realiza entre procesadores distintos cada vez, en función de un criterio que suponga una mejora para el algoritmo. Las topologías más utilizadas son:

- Anillo: las poblaciones están distribuidas en anillo y sólo hay intercambio entre vecinos.

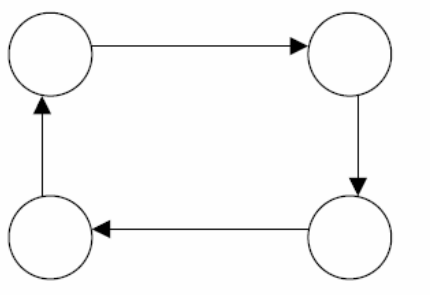


Figura 7.8: Modelo en anillo de un AGP.

- Maestro – Esclavo: todos los procesos esclavos intercambian sus mejores individuos con el maestro.

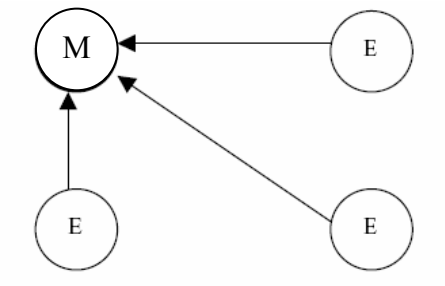


Figura 7.9: Modelo maestro-esclavo de un AGP.

- Todos con todos: todos los procesadores intercambian información con todos.

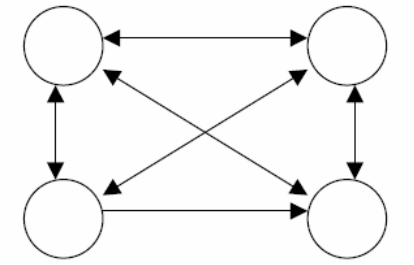


Figura 7.10: Modelo todos con todos de un AGP.

Proporción y frecuencia de intercambio

Tanto la frecuencia como la proporción de intercambio son muy importantes para la convergencia del algoritmo y para la calidad de las poblaciones. Aunque en principio, se puede suponer que cuanto mayor sea el intercambio mejor se propagan las buenas soluciones. Esto no es cierto totalmente, ya que puede suceder que el excesivo intercambio de individuos entre las poblaciones convierta el AGP en una búsqueda prácticamente aleatoria al no permitir que el AGP se desarrolle con normalidad.

Hay diferentes maneras de realizar la implementación: en algunos casos el intercambio se produce únicamente cuando las poblaciones han convergido totalmente; en otros, el intercambio se realiza después de un determinado número de generaciones o con una periodicidad fijada de antemano y que se mantiene constante a lo largo de toda la ejecución del programa.

Debido a la importancia ya citada de estos parámetros, es conveniente realizar un estudio de diferentes políticas de migración teniendo en cuenta la topología de las poblaciones [Cantú-Paz, 1998].

7.4.2.2 AGPs de Grano fino

Los AGPs de grano fino se conocen también como AGP *grid* o en *parrilla* debido a la disposición de las poblaciones sobre los procesadores. Los individuos se disponen en una parrilla de dos dimensiones, con un individuo en cada una de las posiciones de la rejilla. La evaluación se realiza simultáneamente para todos los individuos y la selección, reproducción y cruce se realizan de forma local con un reducido número de vecinos. Con el tiempo, se van formando grupos de individuos que son homogéneos genéticamente como resultado de la lenta difusión de individuos. A este fenómeno se le llama *aislamiento por distancia* y es debido a que la probabilidad de interacción entre dos individuos disminuye con la distancia.

Este tipo de implementación simula las relaciones personales entre individuos de una misma localidad. Es decir, normalmente dos individuos que vivan cerca, tienen más probabilidad de relacionarse que dos que vivan más separados.

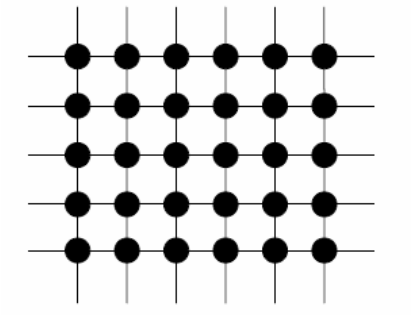


Figura 7.11: Esquema de un AGP de grano fino.

Los AGPs de grano fino se adaptan mejor a las máquinas de tipo SIMD (Single Instrucción Múltiple Data), ya que las operaciones necesarias para las comunicaciones locales son muy eficaces implementadas sobre esta arquitectura.

7.4.2.3 AGPs Híbridos

Un AGP híbrido es una combinación de un AGP de grano grueso con un AGP de grano fino. Algunos de estos algoritmos añaden un nuevo grado de complejidad al entorno de los AGP, pero otros utilizan la misma complejidad que uno de sus componentes. Los AGPs híbridos son normalmente paralelos de grano grueso en el nivel superior. En el nivel inferior algunos tienen un AGP de grano fino y otros tienen de nuevo un AGP de grano grueso. A continuación se muestran dos posibles esquemas de estos tipos de AGPs:

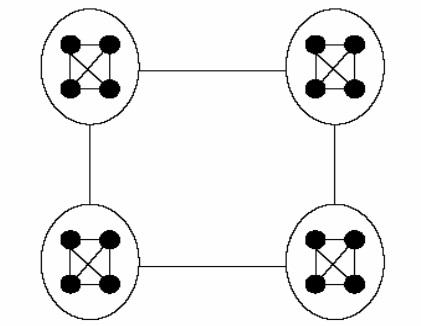


Figura 7.12: AGP híbrido combinando un AGP de grano grueso tanto en el primer como en el segundo nivel.

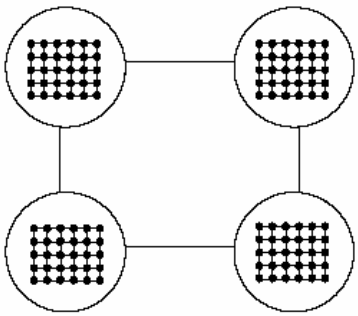


Figura 7.13: AGP híbrido que combina un AGP de grano grueso en el primer nivel con un AGP de grano fino en el segundo nivel.

7.5 Software GASVM

La implementación utilizada hace uso del software paralelo basado en LIBSVM que se ha descrito en el Capítulo 4. Para la implementación de los algoritmos genéticos paralelos se ha utilizado la librería Galib [MIT, 1999], que se

encuentra en la gran mayoría de las distribuciones Linux actuales, por lo que es de libre uso.

En el caso que nos ocupa se ha utilizado una representación binaria de los individuos de la población. La representación se corresponde con las 24 señales que aparecen en el modelo C de la *Tabla 5.3,* siguiendo la siguiente numeración*:* (1)*media*(*Plasma current*)*,* (2)*std*(*fft*(*Plasma current*))*,…..,* (23) *media*(*Vertical centroid position time derivative*)*,* (24)*std*(*fft*(*Vertical centroid position time derivative*))*.* El valor 1 se corresponde con el uso de dicha señal y el valor 0 con su ausencia, por lo que cada individuo de la población tendrá una longitud de 24 bits. Para ello las señales han sido numeradas de 1 a 24, correspondiendo con su posición en el vector de bits que representa cada individuo de la población, siguiendo el esquema presentado en la *Figura 7.1*.

Los datos de entrada a la aplicación son un conjunto de descargas de entrenamiento y un conjunto de descargas de test. De cada conjunto de descargas de entrenamiento se extraen las doce últimas ventanas en las descargas disruptivas, y de las descargas no disruptivas, de forma aleatoria, tantas ventanas como las que se hayan extraído de las descargas disruptivas, con el objetivo de obtener un número equilibrado de características disruptivas y no disruptivas. Como se puede deducir, el número de elementos de entrada es muy reducido, comparado con los utilizados en análisis anteriores, ya que por ejemplo, en el capítulo anterior se extraían las 12 últimas ventanas disruptivas y además, una de cada dos ventanas de todas las descargas no disruptivas para el primer nivel de Apodis. Esto hacía que en algunos casos los datos de entrada llegasen a ser millones de vectores. Ahora no es necesario realizar dos niveles de análisis como en Apodis, ya que se trata de hacer un estudio para ver cuáles son las variables que aportan más información, por lo que sólo se debe mantener un equilibrio entre datos disruptivos y no disruptivos para no alterar los resultados del análisis.

Una vez realizado el proceso de lectura de datos de entrada, se pasa a la fase de entrenamiento, que consiste en calcular el modelo a partir de las ventanas extraídas, para ello hay que tener en cuenta que sólo se utilizará un

número de características fijado al inicio de la ejecución de la aplicación. Este número se mantendrá fijo incluso en los procesos de cruce y mutación, que por su naturaleza podrían llegar a alterarlo.

El tipo de paralelización del algoritmo genético escogido ha sido la paralelización de grano grueso correspondiente a la *Figura 7.6*, al no ser necesarios un gran número de procesadores para evaluar la función de coste, que en este caso consiste en crear un modelo con un conjunto de individuos de la población y testearlo para cada individuo de la población de test, y como se señaló anteriormente los conjuntos de entrenamiento y test van a ser relativamente pequeños. Este proceso no requiere de grandes recursos computacionales, por lo que han sido descartados, de momento, otros tipos de paralelización, y además, la paralelización de grano grueso tiene como ventaja que se pueden realizar menos poblaciones con un conjunto de procesadores más elevado si los datos de entrada fueran más numerosos y se hiciese necesario utilizar un mayor número de procesadores para calcular y testear los modelos.

```
Leer datos de entrada para entrenar y testear los modelos
Enviar los datos a todos los procesos.
Fijar el número de variables a analizar.
Inicializar las poblaciones para realizar el algoritmo genético.
Evaluar poblaciones iniciales y enviar resultados al proceso 0
Mientras no se alcance la condición de finalización
        Cruzar poblaciones
        Mutar poblaciones
        Evaluar poblaciones (entrenar y testar un modelo svm para cada individuo)
        Si se cumple condición1 intercambiar individuos entre poblaciones limítrofes
        Si se cumple condición2 intercambiar con todas poblaciones.
Fin mientras
```

Figura 7.14: Pseudocódigo de la aplicación GASVM.

El pseudocódigo seguido por la aplicación, que se ha denominado GASVM, se presenta en la *Figura 7.14,* en ella se pueden apreciar todos los pasos que incluye la aplicación creada, que no varían mucho del general de un algoritmo genético de grano grueso. La única modificación que se ha realizado es la doble transferencia de los mejores individuos. Por una parte, se hace una transferencia con las poblaciones limítrofes, con una frecuencia más alta que la

segunda transferencia de individuos, con frecuencia más baja, que se hace a todas las poblaciones. El número de poblaciones que se genera en cada ejecución, así como el número de procesadores pertenecientes a cada población, son seleccionados mediante parámetros de entrada a la aplicación.

En la *Figura 7.15* se muestra un esquema de los dos intercambios implementados correspondientes a 8 poblaciones. En este caso, mediante la primera migración se realizan 24 intercambios entre las distintas poblaciones frente a los 56 de la segunda migración.

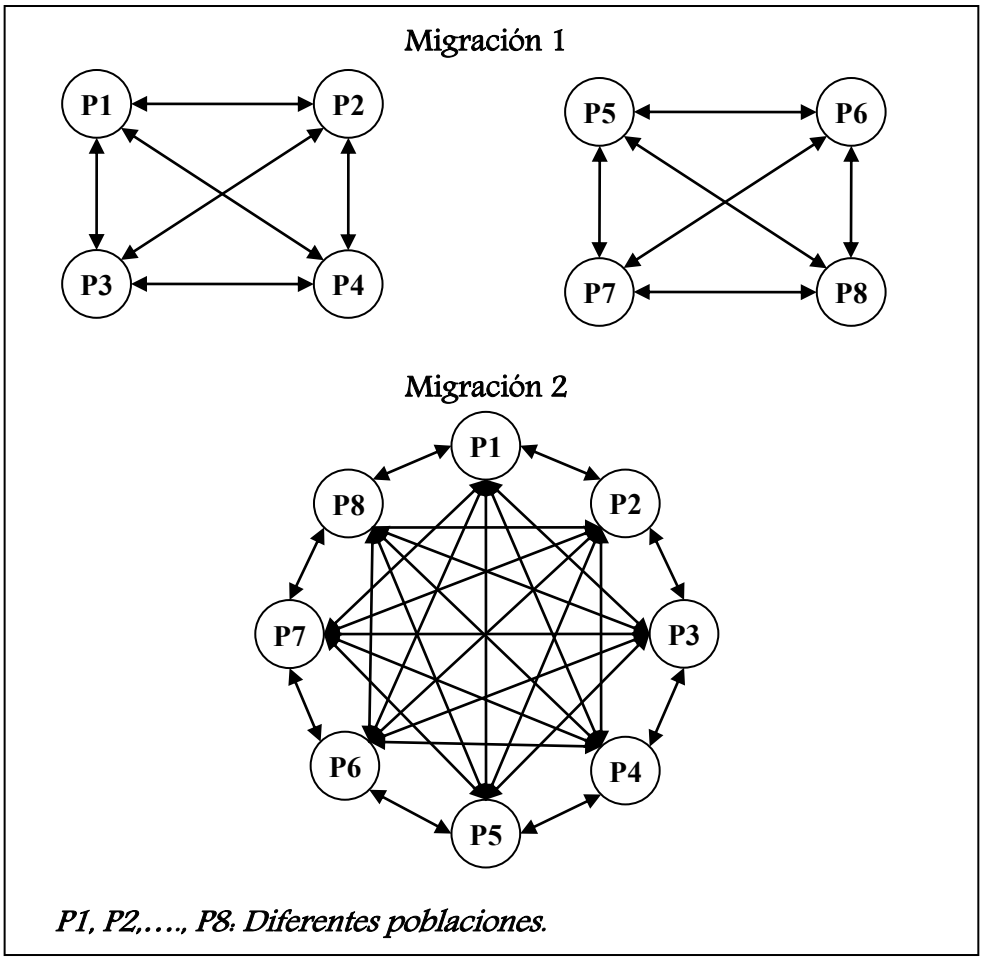


Figura 7.15: Esquema de los dos niveles de migración implementados.

Sin embargo, en el caso de hacer uso de 16 poblaciones, el supuesto utilizado para obtener los resultados que se presentan en este capítulo, los intercambios de individuos en la primera migración son 48 y los de la segunda

240; como se puede ver, se han reducido considerablemente las comunicaciones necesarias para implementar la migración de las poblaciones. Si siempre se utilizase la migración 2 el rendimiento de la aplicación se vería reducido, debido al retardo que generan las comunicaciones entre procesos.

```
1 if (((ga.generation()+factor1)%factor1)==0)
2  {
3    GAPopulation pop(ga.population());
4    GAGenome *tmpind1 = ga.population().individual(0).clone();
5    GAGenome *tmpind2 = ga.population().individual(0).clone();
6
7    int zz1,loz,hiz;
8    loz=rango*(miproc/rango);
9    hiz=rango*((miproc/rango)+1);
10   for (int zz=0;zz<amig;zz++)
11    {
12      zz1=(rand()%nmig);
13      *tmpind1=ga.population().best(zz1);
14        for (int nn=loz;nn<hiz;nn++)
15          if (nn!=miproc)
16           {
17            MPI_Send(tmpind1,sizeof(ga.population().best(zz1)),
18                 MPI_CHAR, nn, 90, MPI_COMM_WORLD);
19            MPI_Recv(tmpind2,sizeof(ga.population().best(zz1)),
20                 MPI_CHAR, nn, 90, MPI_COMM_WORLD, &status);
21            pop.add(*tmpind2);
22           }
23    }
24
25  delete tmpind1;
26  delete tmpind2;
27 }
```

Figura 7.16: Primera migración entre poblaciones.

El segundo objetivo de realizar la migración de las poblaciones en dos pasos es intentar evitar la consecución de un máximo local, ya que si siempre se realizan migraciones del segundo tipo, todos con todos, realmente se estaría trabajando con una población que comparte prácticamente las mismas características y que no evolucionaría correctamente.

En las *Figuras 7.16* y *7.17* se muestra el código asociado a las dos migraciones de individuos programadas en la aplicación. En la primera, la variable *factor1* representa cada cuántas generaciones se realiza la migración, mientras que la variable *amig* indica el número de individuos que se van a migrar y *nmig* el rango de individuos que optan a ser migrados, que varía entre los *nmig* mejores. La creación de las variables *loz* e *hiz* indican los límites inferior y superior, respectivamente, de los procesos vecinos, mientras que la variable

rango indica el número de procesos vecinos. Las variables *tmpind1* y *tmpind2* se utilizan para almacenar los individuos seleccionados y posteriormente transmitirlos a través de las llamadas `MPI_Send` y `MPI_Recv`.

```
1 if (((ga.generation()+factor1)%factor1)==0)
2  {
3    GAPopulation pop(ga.population());
4    GAGenome *tmpind1 = ga.population().individual(0).clone();
5    GAGenome *tmpind2 = ga.population().individual(0).clone();
6
7    int zz1;
8    for (int zz=0;zz<amig;zz++)
9     {
10     zz1=(rand()%nmig);
11     *tmpind1=ga.population().best(zz1);
12     for (int nn=0;nn<numproc;nn++)
13     if (nn!=miproc)
14      {
15        MPI_Send(tmpind1,sizeof(ga.population().best(zz1)),
16              MPI_CHAR, nn, 91, MPI_COMM_WORLD);
17        MPI_Recv(tmpind2,sizeof(ga.population().best(zz1)),
18              MPI_CHAR, nn, 91, MPI_COMM_WORLD, &status);
19        pop.add(*tmpind2);
20      }
21    }
22   delete tmpind1;
23   delete tmpind2;
24 }
```

Figura 7.17: Segunda migración entre poblaciones.

En la *Figura 7.17* el funcionamiento del programa es igual a la primera migración. La única diferencia es que en este caso el intercambio es todos con todos, y por lo tanto, desaparecen las variables *loz, hiz* y *rango*.

Como se señaló anteriormente, la evaluación de la función de coste que se realiza para todos los individuos de cada población, consiste en generar un modelo a partir de los datos de entrenamiento y testearlo con todos los individuos de test. Por otra parte, antes de comenzar la ejecución se fija, mediante un parámetro de entrada, el número de variables sobre las que se va a realizar el estudio. El objetivo final de este análisis es saber si un conjunto mucho más reducido de señales puede obtener unos resultados similares a los obtenidos con 24 señales.

7.6 Resultados del estudio realizado

La importancia de este estudio radica en que si el conjunto de variables es más reducido, la velocidad de extracción de datos en tiempo real y su posterior uso para obtener modelos de predicción será mucho más rápido, cuanto más reducido sea el conjunto de señales utilizadas. Partiendo de las 24 señales iniciales, se han realizado ejecuciones sobre la evolución de poblaciones con diferentes agrupaciones de un número fijo de variables, es decir, se han lanzado ejecuciones utilizando grupos de 2 señales, 3 señales, y así sucesivamente hasta el total de las 24 señales.

Como operadores de selección, cruce y mutación se han utilizado los implementados en la librería Galib. En concreto, para el operador de selección se ha utilizado el *sistema de ruleta*, que consiste en ordenar todos los individuos de mayor a menor, en función del valor devuelto por la función de coste, a continuación, se obtiene un valor aleatorio entre 0 y 1, *valor ruleta*, de forma que todos los individuos cuyo valor en la función coste sea mayor que el *valor ruleta,* serán los elegidos para cruzarse y crear la siguiente generación. Para el operador de cruce se ha hecho uso del cruce de un solo punto de combinación. Además, se ha realizado una variación tras aplicar dichos operadores, ya que ha sido necesario rehacer los cruces y la mutación para obligarlos a cumplir con el número de variables asociadas a cada grupo. Si no se utilizara esa limitación, siempre se obtendrían individuos con cada vez mas variables a 1 porque, tal y como se verá más adelante, los grupos que poseen más variables activas obtienen, por lo general, un mejor resultado en la función de coste.

Para escoger el número de individuos de cada población se ha hecho uso del criterio de Alander, que dice que al menos debe ser el doble del tamaño de cada individuo [Alander, 1992]. En este caso, el número de señales utilizado indicará el tamaño del individuo, si el máximo son 24 señales, el número de elementos debe ser al menos de 48. En las pruebas realizadas se ha fijado en 50 el número de individuos de cada población.

La condición de parada ha sido que todos los individuos de la población compartan al menos el 95% de los genes, es decir, que todos sean iguales hasta en el 95% de sus elementos, por lo que resulta muy difícil crear nuevos individuos que aporten mejoras en la población.

Como factor de migración de individuos entre las poblaciones se ha decidido intercambiar 3 de los 8 mejores individuos cada 10 generaciones con las poblaciones vecinas y cada 25 generaciones con todas las poblaciones. Las poblaciones generadas han sido 16 de 4 procesadores cada una, es decir, se ha hecho uso de 64 procesadores en cada ejecución en la máquina EULER del CIEMAT.

Resumiendo todo lo anterior, se han realizado ejecuciones de la herramienta GASVM para 16 poblaciones, a las que se ha fijado un tamaño de la población de 50 individuos, se ha aplicado una tasa de mutación del 3%, un operador de cruce de un punto, con selección de individuos de tipo ruleta, un parámetro de migración de 3 individuos de los 8 mejores, cada 10 generaciones con las poblaciones vecinas (migración 1), y cada 25 generaciones con todas las poblaciones (migración 2). Por último, se ha fijado como condición de finalización que los individuos de la población compartan el 95% de los caracteres en común. En general, esta condición ha supuesto que las ejecuciones finalizasen cada 90-120 generaciones.

Los experimentos realizados han utilizado datos de JET correspondientes a campañas con pared de carbono. Concretamente, se han utilizado las campañas C15a, C15b, C16, C1617, C18, C19, C20, C21, C22, C24, C25 y C27b. Los datos de estas campañas, así como el rango de las descargas utilizadas, aparecen en las *Tablas 5.1 y 6.11*.

Como datos de entrada para entrenar los modelos se han utilizado 206 descargas disruptivas seleccionadas aleatoriamente de estas campañas y 2016 no disruptivas. Para realizar el test se han utilizado el resto de descargas de las campañas citadas que no se han usado en el entrenamiento, es decir, 143 descargas disruptivas y 3500 no disruptivas.

Finalmente, se han realizado 4 ejecuciones independientes para cada agrupación de señales cuyos resultados totalizados, una vez calculadas las medias tanto de las tasas de acierto, como de las falsas alarmas, se presentan en la *Figura 7.18*. De los resultados obtenidos se puede concluir que a partir de 9 variables la tasa de falsas alarmas cae en el entorno de 5%-6%. Lo mismo ocurre con la tasa de aciertos, que varía entre el 79%-81%.

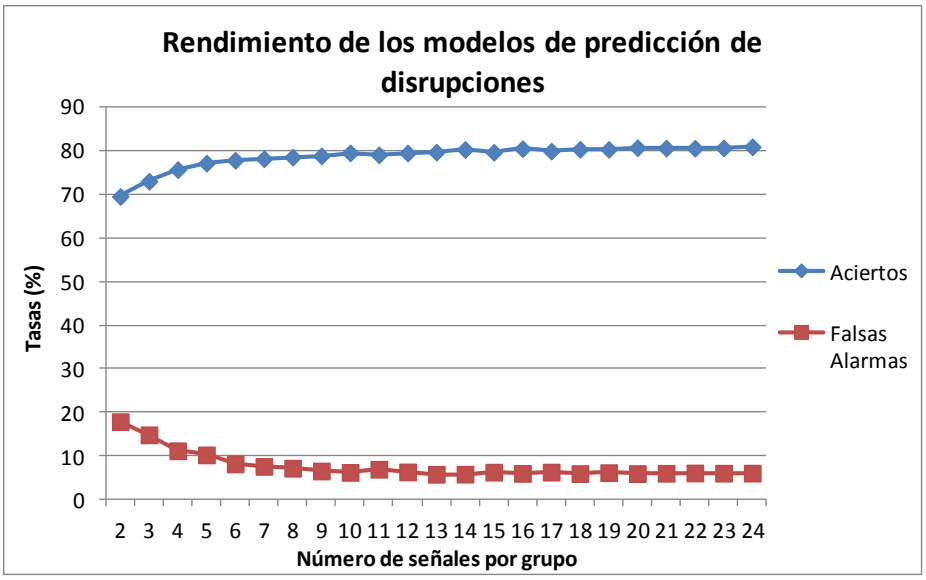


Figura 7.18: Tasas de aciertos y falsas alarmas por agrupaciones de señales.

Como resultado de los cálculos realizados en la *Tabla 7.1* se muestran las mejores señales en función del número de apariciones en cada agrupación. Es decir, para el caso de agrupaciones de dos señales, las señales óptimas, las que más apariciones tienen en los individuos que dan los mejores valores de la tasa de aciertos y menores en la tasa de falsas alarmas, son la señal 8 y la señal 12. Asimismo, y para no incluir todas las señales, en las agrupaciones más numerosas (a partir de 14 señales por grupo), se ha utilizado la palabra *TODAS* para indicar que se incluyen todas las señales y se han excluido del grupo las menos importantes. Por ejemplo, para agrupaciones de 22 señales se utilizarían *TODAS* las señales, excepto la señal número 22 y la señal número 23. También se observa que, dependiendo del número de señales de cada grupo, la identificación de las señales óptimas varía, lo que da una idea de lo compleja que es la tarea de búsqueda de las señales óptimas, ya que una señal óptima para agrupaciones de 2 señales, no lo es para agrupaciones de 3 ó 4 señales.

Para obtener los modelos no se ha realizado un estudio previo de cuáles serían los mejores parámetros a utilizar a la hora de maximizar los resultados de aciertos y minimizar las tasas de falsas alarmas, como sí se hizo en el Capítulo 6 mediante la creación de 50 conjuntos de entrenamiento. Los motivos para no realizar ese estudio son dos:

a) El objetivo en este capítulo es conocer qué agrupación de variables aporta mayor información, no conseguir el mejor modelo posible.

b) Los tiempos de ejecución de cada grupo de variables hubieran sido muy elevados.

Número de variables	Tasa de aciertos	Tasa de falsas	Variables más importantes
2	69,75	17,43	8-12
3	72,87	15,40	11-10-2
4	75,60	11,61	24-11-12-2
5	77,44	9,91	2-11-22-13-15
6	77,70	8,32	24-5-13-9-6-1
7	78,09	7,76	24-2-12-13-15-1-21
8	78,44	7,38	12-15-24-11-1-20-23-2
9	78,94	6,66	24-15-3-1-10-16-18-12-8
10	79,43	6,02	24-12-4-16-17-2-11-10-23-14
11	79,30	6,64	6-24-20-18-21-14-10-3-5-17-13
12	79,58	6,38	20-24-10-6-14-18-21-5-3-12-17-19
13	79,62	6,33	24-12-8-21-3-16-4-23-18-19-17-1-2
14	80,04	5,80	TODAS-22-17-21-13-19-23-4-18-10-2
15	79,98	6,14	TODAS-20-22-16-21-17-23-11-12-10
16	80,34	5,97	TODAS-22-20-11-16-14-18-10-5
17	80,22	6,08	TODAS-20-17-21-18-16-19-15
18	80,44	5,87	TODAS-20-18-16-21-14-11
19	80,30	6,79	TODAS-19-21-20-15-23
20	80,90	6,14	TODAS-23-21-22-20
21	80,61	6,07	TODAS-20-19-17
22	80,74	6,00	TODAS-23-22
23	80,60	6,01	TODAS-21
24	80,94	6,04	TODAS

Tabla 7.1: Tasa de aciertos y falsas alarmas por grupos de variables.

Para hacer una clasificación de la importancia de cada variable, o de la cantidad de información que aportan al resultado final, en la *Tabla 7.2* se presentan los datos del número de apariciones de cada variable en las distintas agrupaciones calculadas, es decir, para conjuntos formados por 2, 3, 4,…., 24

señales. Como se puede apreciar, hay una gran diferencia entre las señales 6, 14, 24, 18 y 20, que doblan el número de apariciones de las señales 22, 11, 19, 23 y 15. De esto se debe deducir que las primeras aportan más información que las últimas.

Señal	Número de apariciones
6	81128
14	73422
24	73375
18	72334
20	69084
10	66108
7	57171
1	56930
4	52044
3	49539
12	49063
2	48551
21	47210
13	46255
17	44519
5	44041
8	37120
16	36438
9	34734
15	33054
23	31906
19	28936
11	28866
22	26492

Tabla 7.2: Clasificación de las señales en función de su frecuencia de aparición.

7.7 Resultados de rendimiento

A la hora de medir el rendimiento de la aplicación GASVM hay que tener en cuenta no sólo el número de procesadores utilizados en cada ejecución, sino también la cantidad de individuos que forman cada población, ya que el tiempo de proceso de la aplicación está muy ligado al tamaño de la población debido a la función de coste del algoritmo genético, consistente en calcular un modelo con el conjunto de entrenamiento y testearlo con el conjunto de test, lo cual lleva asociado, como se vio en los capítulos anteriores, un gran consumo de CPU.

		Individuos por población			
Procesadores (Nº poblaciones)	**Resultados**	**2**	**4**	**8**	**16**
1	**Tiempo(s)**	7830	15762	30972	63498
	% Aciertos	80,27	80,39	80,65	80,69
	% F. Alarmas	5,78	6	5,78	5,81
2	**Tiempo(s)**	9864	20136	34230	71526
	% Aciertos	80,3	80,71	80,57	80,73
	% F. Alarmas	6,09	6,28	5,81	5,77
4	**Tiempo(s)**	3510	7212	13866	27222
	% Aciertos	80,10	80,33	81,12	80,87
	% F. Alarmas	5,71	6,08	6,15	6,04
8	**Tiempo(s)**	2502	4998	9822	19446
	% Aciertos	80,45	80,36	80,48	80,77
	% F. Alarmas	5,82	6,04	5,8	6,08
16	**Tiempo(s)**	2610	5250	10554	20736
	% Aciertos	80,01	80,45	80,74	80,94
	% F. Alarmas	5,64	6,6	6,08	6
32	**Tiempo(s)**	2916	5736	11304	22794
	% Aciertos	79,95	80,24	80,36	80,71
	% F. Alarmas	6,04	5,92	6	5,92
64	**Tiempo(s)**	3048	6192	12348	24360
	% Aciertos	80,48	80,17	80,83	80,80
	% F. Alarmas	5,74	6,12	6,08	6,08

Tabla 7.3: Datos de ejecución para grupos de 20 señales.

La *Tabla 7.3* muestra los tiempos de ejecución para distintos tamaños de la población y diferentes números de procesadores. En este caso, cada población se ha ejecutado en un único procesador. En la tabla se presentan los tiempos de ejecución en segundos, los porcentajes de acierto y de falsas alarmas, de grupos de 20 señales sobre el máximo de las 24 posibles. Se puede observar que para las ejecuciones con 1, 2 ó 4 procesadores los tiempos de proceso son muy altos, si se comparan con el resto de ejecuciones para 8, 16, 32 y 64 procesadores. Esto es debido a que, a partir de 8 procesadores, los datos asociados al entrenamiento de los modelos SVM pueden alojarse por completo en la cache, lo cual acelera el proceso. Los parámetros de entrada, así como los conjuntos de *entrenamiento* y *test* para realizar los cálculos presentados, han sido los mismos que para el estudio del apartado anterior,

recuérdese, 206 descargas disruptivas y 2016 no disruptivas como conjunto de entrenamiento y 143 disruptivas y 3500 no disruptivas como conjunto de test.

La única diferencia con las ejecuciones anteriores es que no se ha colocado un criterio de finalización del 95% de igualdad en los genes de los individuos, sino que se ha fijado un número fijo de generaciones, en este caso 250, para poder hacer comparaciones entre poblaciones con la misma evolución en sus individuos.

En la *Figura 7.19* se presenta la gráfica de escalabilidad de la aplicación GASVM. Aquí hay que señalar que para poder hacer una comparativa de los tiempos de ejecución, se debe utilizar como patrón de comparación el tamaño de las poblaciones que están ejecutándose en cada momento, ya que es lo que condiciona el tiempo de proceso. Partiendo de este dato se está comparando el tiempo de proceso de una población con 16 Individuos, con el de dos poblaciones de 8 individuos, que sería su equivalente. Se puede observar en la *Tabla 7.3* que las tasas de acierto y de falsas alarmas son equivalentes. La *Figura 7.19* presenta resultados de poblaciones entre 16 y 128 individuos.

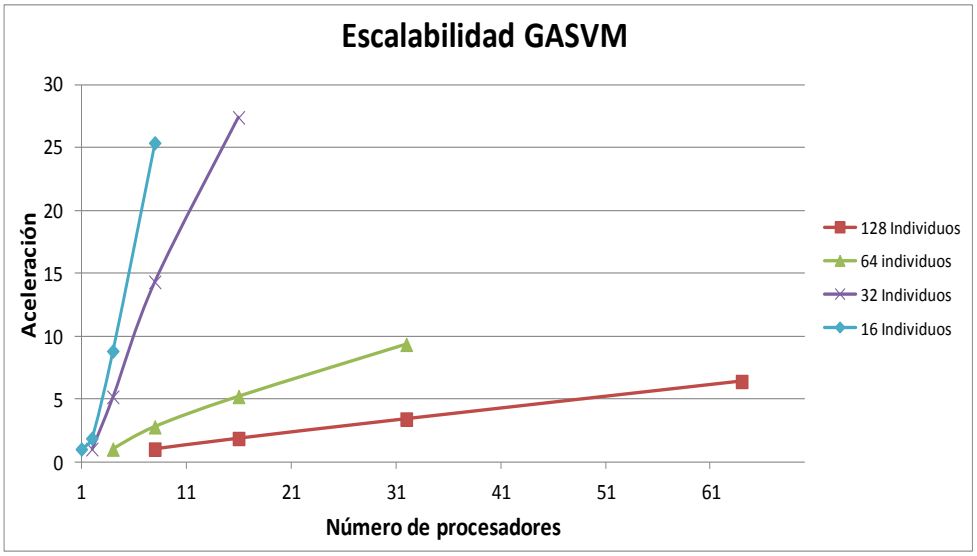


Figura 7.19: Escalabilidad de GASVM.

En la *Tabla 7.4* se presentan los valores de la aceleración del proceso asociado a cada población en función del número de individuos de dicha población. Estos valores son los trazados en la gráfica de la *Figura 7.19*. Se

observa que las aceleraciones son menores a medida que se aumenta el tamaño de las poblaciones, lo cual es lógico, ya que de la misma forma aumenta el número de modelos a entrenar y testear, y también el retardo generado por las migraciones entre poblaciones es mucho mayor.

	Individuos por población			
Procesadores (Nº poblaciones)	***2***	***4***	***8***	***16***
1	-	-	-	*1*
2	-	-	*1,86*	*1*
4	-	*8,80*	*5,16*	*1*
8	*25,38*	*14,31*	*2,77*	*1*
16	*27,40*	*5,19*	*1,84*	-
32	*9,34*	*3,39*	-	-
64	*6,38*	-	-	-

Tabla 7.4: Aceleración de GASVM para grupos de 20 señales.

7.8 Conclusiones

De los resultados obtenidos se puede concluir que, según se van aumentando el número de variables utilizadas, los resultados de predicción son mucho mejores, aumentando la tasa de aciertos y disminuyendo la tasa de falsas alarmas. El problema es que este aumento de variables hace más lenta la extracción de las características y aumenta el tiempo de ejecución del cálculo de los modelos de predicción.

Sin embargo, a partir de 10 variables los resultados obtenidos no ofrecen grandes incrementos en la tasa de aciertos, ni descensos considerables de la tasa de falsas alarmas. Este resultado se corresponde con los datos presentados en los Capítulos 5 y 6, en los que los resultados para 14 y 24 variables no presentan diferencias superiores al 5% ni en la tasa de aciertos, ni en la tasa de falsas alarmas.

Por último, se han analizado los tiempos de proceso de la aplicación en función del número del procesadores, y siempre fijando el tamaño de la población en estudio. De los datos obtenidos se puede deducir que cuando las

poblaciones son muy grandes la aplicación disminuye su rendimiento, bajando la aceleración del proceso. Esto es debido al gran número de modelos de entrenamiento y procesos de test que debe calcular: uno por cada individuo de la población. Además de multiplicar las migraciones entre poblaciones, lo cual hace aumentar el retardo asociado a las comunicaciones.

CAPÍTULO 8

CONCLUSIONES

En esta tesis se han utilizado dos técnicas de minería de datos: las máquinas de vectores soporte y los algoritmos genéticos para la detección y estudio de dos fenómenos físicos asociados a la Fusión Nuclear, como son las transiciones L/H y las disrupciones.

- Para la primera técnica, las máquinas de vectores soporte, se ha implementado una solución basada en el algoritmo Spread-SVM, obteniendo una gran escalabilidad en la aplicación final, que a su vez se ha probado con éxito en el entrenamiento de un modelo que permite clasificar satisfactoriamente transiciones de modo-L a modo-H. Antes de poseer esta herramienta se necesitaban decenas de horas de cálculo cuando los datos de entrada superaban el millón, mientras que ahora sólo se necesitan apenas unos minutos de ejecución en un entorno de supercomputación de al menos 8 procesadores, pudiendo disminuir ampliamente el tiempo de proceso si el número de procesadores es mayor.

- A continuación, utilizando como base la aplicación paralelizada de SVM, se ha implementado un sistema de predicción de disrupciones a partir del sistema Apodis, que se ha entrenado y testado con la mayor base de datos de disrupciones utilizada hasta la fecha para diferentes grupos

señales de entrada, obteniendo modelos con unas tasas de acierto entre el 87,5% y el 95% para el mejor modelo, que venían acompañadas de una tasa de falsas alarmas baja, menor del 5% en casi todas las campañas. Como dato a tener en cuenta, es importante mencionar que el mejor modelo de predicción de disrupciones obtenido está implementado actualmente en la red de tiempo real de JET [Neto, 2010].

- Además, se comprobó la robustez del sistema con un gran número de descargas correspondientes a la campaña más reciente de JET la C2830 (con pared metálica). Este resultado tiene una gran importancia, ya que muestra claramente que las tasas de acierto se mantienen altas, a pesar de los cambios estructurales que han sido efectuados sobre el dispositivo de fusión. Después de dichos cambios las tasas de acierto continúan siendo satisfactorias.

- Se continúa en el Capítulo 6 planteando la posibilidad de obtener modelos de predicción de disrupciones a medida que se van realizando descargas en las distintas campañas, con el objetivo de conocer cuántas descargas son necesarias para crear un modelo fiable. Se implementaron varios métodos, utilizando como base el sistema Apodis, que se han denominado métodos *partiendo de cero*: Desbalanceado, Balanceado e Híbrido.

- De los resultados de los modelos balanceados, desbalanceados e híbridos se deduce que, al menos, son necesarias 40 descargas disruptivas, en el caso de las campañas C2830, para obtener resultados en la tasa de aciertos superiores al 90%. También se puede afirmar que los modelos híbridos, en los que se fija el número máximo de descargas no disruptivas y a los que se les van añadiendo las descargas disruptivas que se van produciendo, son los que mejores resultados obtienen. Estos modelos han obtenido resultados en la tasa de aciertos superiores al 92% para campañas de la C28 a la C30 y superiores al 84% para campañas anteriores a éstas.

- También se ha realizado una estadística de cuántos modelos sería necesario calcular para intentar mantener el 100% de acierto en una determinada campaña, no siendo necesarios más de 15 modelos, por lo que su implementación en tiempo real es posible, al no superar los 300 segundos el tiempo de cálculo de cada modelo en un entorno de supercomputación. Estos datos, de número de modelos y tiempo de cálculo, son relativos a la campaña para la que se ha realizado el estudio, pero la metodología utilizada es totalmente general y válida para futuras campañas.

- Finalmente, se ha hecho uso de los algoritmos genéticos, como método de estudio de la información aportada por las señales físicas utilizadas para crear los modelos de predicción de disrupciones. Del estudio realizado se puede afirmar que a partir de un conjunto de 10 señales las tasas de acierto crecen apenas un 1% y las tasas de falsas alarmas no bajan tampoco en más de un punto porcentual. Por lo tanto, y de cara a la implementación de un sistema de predicción de disrupciones en tiempo real, sería suficiente con 10 señales para obtener unos buenos resultados.

Como líneas futuras de investigación se plantean:

- Detectar el tipo de disrupción: si al detectar un comportamiento disruptivo es posible, además, determinar el tipo de disrupción, se podrían obtener modelos de detección mucho más fiables.

- Usar la herramienta de algoritmos genéticos, bien para mejorar la caracterización de las señales que intervienen en las disrupciones o bien para aplicarlas a otros fenómenos físicos.

- Simular la implantación del sistema de creación de modelos “partiendo de cero” con datos de las campañas más recientes de JET y realizar un estudio de viabilidad.

- En esta tesis las evaluaciones de los predictores desarrollados se ha realizado a partir de las tasas de aciertos tanto en las descargas disruptivas como en las descargas no disruptivas. Es decir, cuando una descarga disruptiva se predice como disruptiva se contabiliza como acierto y si se predice como no disruptiva se contabiliza como fallo. Análogamente ocurriría con las descargas no disruptivas. Como nueva línea de investigación se propone el asociar medidas de probabilidad a las predicciones realizadas. En este caso no solo se predice el tipo de descarga sino que se asocia una medida para decir cuánto de seguro se está al realizar dicha predicción. Para esta nueva aproximación se pueden utilizar predictores bayesianos o predictores Venn, los cuales permiten asociar a cada predicción individual una probabilidad o incluso una barra de error de dicha probabilidad.

- Otra línea futura de investigación consiste en la creación de modelos dinámicos que permitan predecir muestras futuras de las señales experimentales a partir de valores pasados. Este hecho tendría un fuerte impacto a la hora de calcular los tiempos de predicción de las disrupciones, debido a que los tiempos proporcionados por cualquier predictor que se desarrolle pueden ser incrementados con el tiempo de predicción de las muestras futuras.

Bibliografía

[Alander, 1992] Alander J.T. "On Optimal population size of genetic algorithms". *Proceedings CompEuro, Computer Systems and Software Engineering, 6th Annual European Computer Conference.* (1992). pp. 65-70

[Alejaldre, 1999] C. Alejaldre et al. "First plasmas in the TJ-II flexible Heliac". *Plasma Phys. Controlled Fusion* 41, 1, 1999, pp. A539-A548.

[Amdahl, 1967] Amdahl, G.M. "Large-scale Validity of the Single-processor Approach to Achieving Computing Capabilities." *In Proceedings of the American Federation of Information Processing Societies.* Washington, 1967.

[Blazewicz, 2000] Blazewicz, J. *Handbook on Parallel and Distributed Processing, In International Handbooks on Information Systems.* Springer, 2000.

[Bottou, 2007] Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Weston Jason. *Large-scale kernel machines.* MIT Press, 2007.

[Boozer, 2012] A. H. Boozer. "Theory of tokamak disruptions". *Physics of Plasmas* 19. 058101, 2012.

[Braams, 2002] Braams, C.M., and P.E. Stott. "Nuclear Fusion. Half a Century of Magnetic Confinement Fusion Research." *Plasma Physics and Controlled Fusion* 44, no. 8 (2002): 1767.

[Cannas, 2004] Cannas, B., A. Fanni, G. Sias, P. Sonato, M. K. Zedda, and JET-EFDA contributors. "Neural approaches to disruption prediction at JET." *31st EPS Conference on Plasma Phys.* London, 2004.

[Cannas, 2007] Cannas, B., A. Fanni, P. Sonato, M. K. Zedda, and JET-EFDA contributors. "A prediction tool for real-time application in the disruption protection system at JET." *Nuclear Fusion* 47, no. 11 (2007).

[Cantú-Paz, 2000] Cantú-Paz, E. *Efficient and Accurate Paralell Genetic Algorithms.* Kluwer Academic Publishers, 2000.

[Cantú-Paz, 1998] Cantú-Paz, Erick. "A Survey of Parallel Genetic Algorithms." *Calculateurs Paralleles* 10 (1998).

[Chen, 1996] Chen, Chi-hau. *Fuzzy logic and neural network handbook.* McGraw-Hill, 1996.

[Cherkassky, 1998] Vladimir Cherkassky, Filip Mulier. *Learning from Data: Concepts, Theory and Methods (Adaptative and Learning Systems for Signal Processing, Comunications and Control Series).* John Wiley & Sons, 1998.

[Chih-Chung, 2012] Chih-Chung, C., and L. Chih-Jen. *LIBSVM: a Library for support Vector Machines.* 2012. http://www.csie.ntu.edu.tw/~cjlin/libsvm/ (accessed 2012).

[Christianini, 2000] Christianini, N., and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[Collobert, 2001] Collobert, R., and S. Bengio. "SVMTorch: Support Vector Machines for Large-Scale Regresion Problems." *Journal of Machine Learning Research* 1 (2001): 143-160.

[Culler, 1998] Culler, D., J. Pal-Singh, and A. Gupta. *Parallel Computer Architecture.* In Morgan Kaufmann Series in Computer, 1998.

[De Jong, 1975] De Jong, K. A. "An analysis of the behaviour of a class of genetic adaptative systems." Tesis Doctoral. University of Michigan, 1975.

[De Vries, 2009] De Vries, P. C., M.F. Johnson, and I. Segui. "Statistical Analysis of Disruptions in JET." *Nuclear Fusion* 49, no. 5 (2009).

[De Vries, 2011] De Vries, P. C., et al. "Survey of disruption causes at JET" *Nuclear Fusion* 51, no. 5 (2011).

[De Vries, 2012] De Vries, P. C., et al. "The impact of the ITER-like wall at JET on disruptions", *Plasma Phys. Control Fusion* 54, no. 12 (2012).

[Dormido-Canto, 2006] S. Dormido-Canto, G. Farias, J. Vega, R. Dormido, J. Sánchez, M. Santos, J. A. Martín, G. Pajares. "Search and retrieval of plasma wave forms: Structural pattern recognition approach". *Review of Scientific Instruments*. 77 10F514 (2006).

[Dormido-Canto, 2008] S. Dormido-Canto, G. Farias, J. Vega, R. Dormido, J. Sánchez, N. Duro, H. Vargas, A. Murari. "Classifier based on support vector machine for JET plasma configurations". *Review of Scientific Instruments*. 79, 10F326 (2008).

[Dormido-Canto, 2013] S. Dormido-Canto, J. Vega, J.M. Ramírez, A. Murari, R. Moreno, J.M. López, A. Pereira and JET-EFDA Contributors. "Development of an efficient real-time disruption predictor from scratch on JET and implications for ITER". *Nuclear Fusion* 53 (2013) 113001.

[Dror, 2005] Dror, G., R. Sorek, and S. Shamir. "Accurate identification of alternatively spliced exons using Support Vector Machine." *Bioinformatics* 21, no. 7 (2005): 897-901.

[Duro, 2009] N. Duro, R. Dormido, J. Vega, S. Dormido-Canto, G. Farias, J. Sánchez, H. Vargas, A. Murari. "Automated recognition system for ELM classification in JET". *Fusion Engineering and Design*. 84 (2009) 712-715.

[EFDA, 2013] EFDA. *European Fusion Development Agreement.* http://www.jet.efda.org/.

[Fukunaga, 1990] Fukunaga, Keinosuke. "Introduction to Statistical Pattern Recognition. Second Edition." Academic Press, 1990.

[Geist, 1994] Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM, Parallel Virtual Machine.* MIT Press, 1994.

[Goldberg, 1989] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[González, 2012 (1)] González, S., et al. "Automatic location of L/H transition times for physical studies with a large statistical basis." (Plasma Physics and Controlled Fusion.) 54, no. 6 (2012).

[González, 2012 (2)] González, S., et al. "H/L transition time estimation in JET using conformal predictors." (Fusion Engineering and Design) 87 (2012).

[Graf, 2005] Graf, Peter, Eric Cosatto, Leon Bottou, Igor Duradnovic, and V. Vapnik. "Parallel Support Vector Machines: The cascade SVM." *Advances in Neural Information Processing Systems* 17 (2005).

[Grama, 2003] Grama, A., G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing (2nd Edition).* Pearson-Addison Wesley, 2003.

[Gropp, 1999] Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI, 2nd Edition Portable Parallel Programming with the Message Passing Interface.* MIT Press, 1999.

[Guyon, 2013] Guyon, Isabelle. *Clopinet.* http://www.clopinet.com/SVM.applications.html (accessed 03 23, 2013).

[Hager, 2010] Hager, G., and G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers.* CRC Press, 2010.

[Hagersten, 1990] Hagersten, E., A. Landin, and S. Haridi. *Multiprocessor Consistency and Synchronization Through Transient Cache States.* Kluwer Academic Publishers, 1990.

[Haykin, 2009] Haykin, Simon S. *Neural Networks and Learning Machines.* Prentice Hall, 2009.

[Henessy, 2002] Henessy, J., and D. Patterson. *Computer Architecture: A Cuantitative Approach.* In Morgan Kaufmann Series in Computer Architecture and Design, 2002. Tercera Edición.

[Hernández, 2005] Hernández Orallo, J., M.J. Ramírez Quintana, y C. Ferri Ramírez. *Introducción a la minería de datos.* Pearson-Prentice Hall, 2005.

[Hidalgo, 2001] Hidalgo, I., J Lanchares, y A. Ibarra. *Introducción a los Algoritmos Evolutivos*. Departamento de Arquitectura de Computadores y Automática de la UCM, 2001.

[Hilera, 1995] Hilera, J.R., y V.J. Martínez. *Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones.* RA-MA, 1995.

[Hwang, 1998] Hwang, K., and Z. Xu. *Scalable Parallel Computers.* McGraw-Hill, 1998.

[ITER, 1999] ITER Physics Expert Group on Disruptions, Plasma Control and MHD et al. "Chapter 3: MHD stability, operational limits and disruptions." *Nuclear Fusion* 39, no. 12 (1999).

[Japkowicz, 2002] Japkowicz, N., and S. Stephen. "The Class Imbalance Problem: A Systematic Study." *Intelligent Data Analysis* 6 (2002): 429-449.

[Joachims, 2008] Joachims, Thorsten. *SMVLight.* 08 14, 2008. http://svmlight.joachims.org/.

[John, 1994] John, G., R. Kohavi, and K. Preger. "Irrelevant Features and the Subset Selection Problem." *Proceedings of the 11th International Conference on Machine Learning.* 1994.

[Karniadakis, 2003] Karniadakis, George Em, and Robert M. Kirby. *Parallel Scientific Computing in C++ an MPI.* Cambridge University Press, 2003.

[Lawson, 1957] Lawson, J. D. "Some criteria for a power producing thermonuclear reactor." *Proceedings of the Physical Society.* 70, no. 1 (1957).

[Lee, 1980] Lee, R.B. "Empirical Results on the Speedup, Eficiency, Redundancy and Quality of Parallel Computations." *In Proceedings of the International Conference on Parallel Processing.* 1980.

[Lin, 2000] Lin, Y., Y. Lee, and G. Wahba. "Support Vector Machines for Classification in Nonstandard Situations." University of Wisconsin. TECHNICAL REPORT NO. 1016, 2000.

[Liu, 1998] Liu, H., and H. Motoda. *Feature Extraction, Construction and Selection. A Data Mining Perspective.* Kluwer Academic Publishers, 1998.

[López, 2013] López, J. M., et al. "Integration and Validation of a Disruption Predictor Simulator in JET."*Fusion Science and Technology* 63, no. 1 (2013): 26-33.

[Maimon, 2005] Maimon, O., and L. Rokach. *Data Mining and Knowledge Discovery Handbook.* Springer, 2005.

[Mangasarian, 2001] Mangasarian, O. L., and David R. Musicant. "Lagrangian Support Vector Machines." *Journal of Machine Learning Research* 1 (2001): 161-177.

[Marques de Sá, 2001] Marques de Sá, J.P. *Pattern recognition: Concepts, Methods and Applications.* Springer, 2001.

[Matthews, 2011] Matthews, G.F. et al. «JET ITER-like wall—overview and experimental programme.» *Physica Scripta* 2011 (2011).

[McCracken, 2005] McCracken, G., and P. Stott. *Fusion: The Energy of the Universe.* Elsevier Academic Press, 2005.

[Mechoso, 1994] Mechoso, C.R., J.D. Farrara, y J.A. Spahr. *Achieving superlinear speedup on a heterogeneous, distributed system*. *Parallel & Distributed Technology: Systems & Applications, IEEE* 2, nº 2 (1994): 57-61.

[Mingmin, 2008] Mingmin, Chi, Rui Feng, y Lorenzo Bruzzone. «Classification of hyperspectral remote-sensing data with primal SVM for small-sized training dataset problem.» *Advances in Space Research* 41, nº 11 (2008): 1793-1799.

[Mitchell, 1996] Mitchell, M. *An Introduction to Genetic Algorithms.* Cambridge: MIT Press, 1996.

[MIT, 1999] (MIT), Massachusetts Institute of Technology. *GAlib.* December 12, 1999. http://lancet.mit.edu/ga/.

[Murari, 2008] Murari, A., y otros. «How to extract information and knowledge from fusion massive databases.» *Burning Plasma Diagnostics. AIP Conference Proceedings.* 2008.

[Murtagh, 1978] Murtagh, B.A., y M. A. Saunders. «Large-scale linearly constrained optimization.» *Mathematical Programming* 14 (1978): 41-72.

[Nagashima, 1995] Nagashima, U., S. Hyugajia, S. Sekiguchib, M. Satob, y H. Hosoyaa. «An experience with super-linear speedup achieved by parallel computing on a workstation cluster: Parallel calculation of density of states of large scale cyclic polyacenes.» *Parallel Computing* 21, nº 9 (1995): 1491–1504.

[Neto, 2010] Neto A.C., Sartori F., Piccolo F., Vitelli R., De Tommasi G., Zabeo L. et al. "MARTe: a multiplatform real-time framework". *IEEE Transactions on Nuclear Science.* vol. **57**, nº 2, pp. 479-486. April 2010.

[Nguyen, 2010] Nguyen, Giang Hoang, Son Lam Phung, y A. Bouzerdoum. «Efficient SVM training with reduced weighted samples.» *Neural Networks (IJCNN), The 2010 International Joint Conference on.* 2010.

[OpenMP, 2013] OpenMP Architecture Review Board. *OpenMP.* 2013. http://www.openmp.org (accessed 2012).

[Osuna, 1997] Osuna, E., R. Freund, y F. Girosi. «An improved training algorithm for support vector machines.» *Neural Networks for Signal Processing. Proceedings IEEE Workshop.* 1997.

[Patterson, 1996] Patterson, D.W. *Artificial Neural Networks. Theory and Applications.* Prentice Hall, 1996.

[Pautasso, 2002] Pautasso, G., y et al. «On-line prediction and mitigation of disruptions in ASDEX Upgrade.» *Nuclear Fusión* 42, nº 1 (2002): 100-108.

[Platt, 1998] Platt, John C. «Sequential minimal optimization: a fast algorithm for training support vector machines.» Microsoft Research. Technical Report MSR-TR-98-14, 1998.

[Ramírez, 2010] J. Ramírez, S. Dormido-Canto, J. Vega and JET-EFDA Contributors. "Parallelization of automatic classification systems based on support vector machines: Comparison and application to JET database". *Fusion Engineering and Design*. 85 (2010) 425-427.

[Rattá, 2010] Rattá, G.A., et al. "An advanced disruption predictor for JET tested in a simulated real-time environment." *Nuclear Fusion* 50, no. 2 (2010).

[Rattá, 2012] Rattá, G.A., J. Vega, A. Murari, and JET-EFDA Contributors. "Improved feature selection based on genetic algorithms for real time disruption prediction on JET." *Fusion engineering and Design* 87 (2012): 1670-1678.

[Rattá, 2008] Rattá, G.A., J. Vega, A. Murari, and Johnson M. "Feature extraction for improved disruption prediction analysis at JET." *Review of Scientific Instruments* 10F328, no. 79 (2008).

[Raudys, 1991] Raudys, S.J. "Small sample size effects in statistical pattern recognition: recommendations for practitioners." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 13, no. 3 (1991): 252-264.

[Rey, 2011] Rey, Pablo. «Centro de supercomputación de Galicia.» 31 de 01 de 2011. https://www.cesga.es/es/biblioteca/downloadAsset/id/490 (último acceso: 2011).

[Rong-En, 2005] Rong-En, F., C. Pai-Hsuen, and L. Chih-Jen. "LibSVM." 11 2005. http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf (accessed 2010).

[Sánchez, 2011] J. Sánchez, M. Acedo, D. Alegre, A. Alonso, J. Alonso, P. Álvarez, et al. "Overview of TJ-II experiments". *Nuclear Fusion*, 51 (2011) 094022 (10pp).

[Santagiustina, 1993] Santagiustina, A., et al. "Design of the m=2, n=1 tearing mode control system for JET." *Fusion Engineering - Supplement, 1993., 15th IEEE/NPSS Symposium on.* 1993.

[Schuller, 1995] Schuller, F.C. "Disruption in Tokamaks." *Plasma Phys. Control. Fusion* A135–62, no. 37 (1995).

[Sengupta, 2001] Sengupta, A., and P. Ranjan. "Prediction of density limit disruption boundaries from diagnostic signals using neural networks." *Nucear Fusion* 41, no. 5 (2001).

[Sheffield, 1994] Sheffield, J. "The physics of magnetic fusion reactors." *Rev.Mod.Phys* 66 (1994): 1015-1103.

[Snir, 1996] Snir, M., S. W. Ott, S. Huss-Lederman, D.W. Walker, and Dongarra J. *MPI The Complete Reference.* MIT Press, 1996.

[Tanenbaum, 2009] Andrew Tanenbaum. *Sistemas Operativos Modernos*. Prentice-Hall, Tercera Edición. 2009

[Tay, 2001] Tay, F., and L. Cao. "Application of support vector machines in financial time series forecasting." *Omega: The International Journal of Management Science* 29, no. 4 (2001): 309-317.

[Vanderbei, 1999] Vanderbei, R. J. "LOQO User's Manual, Version 3.10." *Optimization Methods and Software* 12 (1999).

[Vapnik, 1971] V. Vapnik and A. Chervonenkis. "On the uniform convergence of relative frequencies of events to their probabilities*." Theory of Probability and its Applications* 16, no. 2 (1971):264–280.

[Vapnik, 2000] Vapnik, Vladimir N. *The Nature of Statistical Learning Theory.* Springer, Second Edition. 2000.

[Vega, 2008] J. Vega, "Intelligent methods for data retrieval in fusion databases*". Fusion Engineering and Design*. 83 (2008), pp. 382-386.

[Vega, 2009 (1)] J. Vega, A. Murari, G. Rattá, G.A. Vagliasindi and JET-EFDA Contributors. "Automated estimation of L/H transition times at JET by combining Bayesian statistics and support vector machines." *Nuclear Fusion* 49, no. 8 (2009).

[Vega, 2009 (2)] J. Vega, A. Murari, A. Pereira, A. Portas, G. A. Rattá, R. Castro and JET-EFDA Contributors. "Overview of intelligent data retrieval methods for waveforms and images in massive fusion databases". *Fusion Engineering and Design* 84. pp. 1916-1919 (2009)

[Vega, 2010 (1)] J. Vega, A. Murari, S. González and JET-EFDA Contributors. "A universal support vector machines based method for automatic event location in waveforms and video-movies: Applications to massive nuclear fusion databases". *Review of Scientific Instruments*. 81, 023505 (2010) 11pp

[Vega, 2010 (2)] J. Vega, A. Murari, G. A. Rattá, S. González, S. Dormido-Canto and JET-EFDA Contributors. "Progress on statistical learning systems as data mining tools for the creation of automatic databases in Fusion environments". *Fusion Engineering and Design* 85. (2010) 399-402.

[Vega, 2013] J. Vega, S. Dormido-Canto, J. M. López, A. Murari, J. M. Ramírez, R. Moreno, M. Ruiz, D. Alves, R. Felton and JET-EFDA Contributors. "Results of the JET real-time disruption predictor in the ITER-like wall campaigns". *Fusion Engineering and Design* 88 (2013) 1228-1231

[Weiss, 2003] Weiss G. M., Provost F. "Learning when training data are costly: The effect of class distribution on tree induction." *Journal of Artificial Intelligence Research* 19, (2003).

[Wesson, 2004] Wesson, J. *Tokamaks.* Oxford Clarendon Press, Tercera edición. 2004.

[Wu, 2011] Wu, Zhenyu, Jun Sun, Lin Feng, and Bo Jin. "A Policy of Cluster Analyzing Applied to Incremental SVM Learning with Temporal Information." *Journal of Convergence Information Technology* 6, no. 7 (2011).

[Yang, 1998] Yang, J., and V. Honavar. "Feature Subset Selection Using a Genetic Algorithm." *Intelligent Systems and their Applications, IEEE* 13 (1998).

[Yoshino, 2003] Yoshino, R. "Neural-net disruption predictor in JT-60U." *Nuclear Fusion* 43, no. 12 (2003).

[Yoshino, 2005] Yoshino, R. "Study of plasma termination using high-Z noble gas puffing in the JT-60U tokamak." *Nuclear Fusion* 45, no. 5 (2005).

[Yuh-Jye, 2000] Yuh-Jye, L., O. L. Mangasarian, and W. H. Wolberg. "Breast Cancer Prognosis: Chemotherapy Effect on Survival Rate." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 55 (2000): 1-10.

ANEXOS

ANEXO I. Trabajos relacionados con esta tesis

- J. Ramírez, S. Dormido-Canto, J. Vega and JET-EFDA Contributors. "Parallelization of automatic classification systems based on support vector machines: Comparison and application to JET database". Fusion Engineering and Design. 85 (2010) 425-427.

- S. González, J. Vega, A. Murari, A. Pereira, J. M. Ramírez, S. Dormido-Canto and JET-EFDA contributors. "Support vector machine-based feature extractor for L/H transitions in JET". Review of Scientific Instruments 81, 10E123 (2010) 3pp.

- J. Ramírez, S. Dormido-Canto, J. Vega and JET-EFDA Contributors. "Automatic parallelization of classification systems based on Support Vector Machines: comparison and application to JET database". 7th IAEA Technical Meeting on Control, Data Acquisition and Remote Participation for Fusion Research. 15-19 June 2009. Aix-en-Provence (France) (http://www-fusion-magnetique.cea.fr/tmiaea2009/index.html).

- J. Vega, A. Murari, G. A. Rattá, S. González, A. Pereira, R. Castro, A. Portas, I. Pastor, S. Dormido-Canto, R. Dormido, N. Duro, J. Sánchez, G. Farias, J. M. Ramírez, L. Makili, H. Vargas, G. Pajares, M. Santos, M. Ruiz, E. Barrera, J. M. López , G. De Arcas. "An overview about recent developments on advanced data analysis techniques in fusion". 6th Workshop on Fusion Data Processing, Validation and Analysis. January 25th-27th, 2010. Madrid (Spain) (http://fudaqs2.ciemat.es/V6/main.htm).

- J. Ramírez, S. Dormido-Canto, J. Vega and JET-EFDA Contributors. "Parallel SVM for pattern recognition and regression". 6th Workshop on Fusion Data Processing, Validation and Analysis. January 25th-27th, 2010. Madrid (Spain) (http://fudaqs2.ciemat.es/V6/main.htm).

- S. González, J. Vega, A. Murari, A. Pereira, J. M. Ramírez, S. Dormido-Canto and JET-EFDA contributors. "SVM-based feature extractor for L/H transitions in JET". 18th Topical Conference on High Temperature Plasma Diagnostics. May 16-20, 2010. Wildwood. New Jersey. (USA). (http://www.htpd2010.pppl.gov/index.aspx).

- S. González, J. Vega, A. Murari, A. Pereira, S. Dormido-Canto, J.M. Ramírez and JET-EFDA contributors. "H/L transition time estimation in JET using conformal predictors". Fusion Engineering and Design. 87 (2012) 2084–2086

- S. González, J. Vega, A. Murari, A. Pereira, S. Dormido-Canto, J. M. Ramírez and JET-EFDA contributors. "Automatic location of L/H transition times for physical studies with a large statistical basis". Plasma Physics and Controlled Fusion. (2012) Volumen 54 Number 6.

- J. M. López, J. Vega, D. Alves, S. Dormido-Canto, A. Murari, J. M. Ramírez et al. "Implementation of the Disruption Predictor APODIS in JET's Real-Time Network using the MARTe Framework". Proc. of the 18th REAL-TIME Conference. June 11-15, 2012. Berkeley (CA), USA.

- Jesús Vega, Sebastián Dormido-Canto, Juan M. López, Andrea Murari, Jesús M. Ramírez, Raúl Moreno, Mariano Ruiz, Diogo Alves, Robert Felton and JET-EFDA Contributors. "Results of the JET real-time disruption predictor in the ITER-like wall campaigns". 27th Symposium on Fusion Technology, 2012. Liege, Belgium.

- A.Murari, J.Vega, P.Boutot, B.Cannas, S.Dormido-Canto, A.Fanni, J. M. López, R. Moreno, A. Pau, G.Sias[4], J. M. Ramírez, G. Verdoolaege , ASDEX Upgrade Team and JET-EFDA Contributors. "Latest Developments in Data Analysis Tools for Disruption Prediction and for the Exploration of Multimachine Operational Spaces". 24th Fusion Energy Conference, San Diego 2012.

- J. M. López, J. Vega, S. Dormido-Canto, A. Murari, J. M. Ramírez, M. Ruiz, and JET-EFDA Contributors. "Integration and Validation of a Disruption Predictor Simulator in JET". Fusion Science and Technology. 63 (2013) 26-33.

- J. Vega, S. Dormido-Canto, J. M. López, A. Murari, J. M. Ramírez, R. Moreno, M. Ruiz, D. Alves, R. Felton and JET-EFDA Contributors. "Results of the JET real-time disruption predictor in the ITER-like wall campaigns". Fusion Engineering and Design 88 (2013) 1228-1231.

- J. Vega, A. Murari, S. Dormido-Canto, R. Moreno, J. M. Ramírez, J. M. López, D. Alves, G. A. Rattá, A. Pereira and JET-EFDA Contributors. "Real-time prediction of disruptions: results in JET and research lines for ITER". 8th Workshop Fusion Data Processing, Validation and Analysis. Ghent (Belgium), November 4-6, 2013

- S. Dormido-Canto, J. Vega, J.M. Ramírez, A. Murari, R. Moreno, J.M. López, A. Pereira and JET-EFDA Contributors. "Development of an efficient real-time disruption predictor from scratch on JET and implications for ITER". Nuclear Fusion 53 (2013) 113001.

ANEXO II. Entornos de ejecución computacional

Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas. CIEMAT, Madrid

- ***Cluster Lince***

El cluster lince está formado por los siguientes componentes:

- Nodo LINSL4 para el acceso interactivo de los usuarios.
- Nodos LINCE12 al LINCE98 que son los nodos de cálculo.
- Interconexión entre los nodos a través de switch Gigabit Ethernet
- Interconexión adicional de 32 nodos a través de la red Infiniband

Todos los nodos tienen dos procesadores Xeon, con capacidad de hyperthreading. Esta capacidad está activada en algunos nodos que se comportan como si tuvieran cuatro procesadores virtuales aunque en los test realizados para este trabajo se han usado los nodos que tenían desactivado el hyperthreading.

Las características de cada nodo son:

LINCE12 al LINCE98

- 2 cpus XEON a 3.2 GHz
- Caché L2: 1024 KB cada CPU
- RAM: 2 GB
- BUS FSF a 800 MHz
- Conexión a Infiniband (lince12 al lince43)

Según la carga y tipos de trabajos que se procesen en el cluster, puede ser conveniente o prejudicial tener el hyperthreading activado. Por este motivo puede haber, según las circunstancias, unos nodos con el hyperthreading activado y otros no. El sistema batch permite dirigir los trabajos a los nodos oportunos. El cluster está organizado de la siguiente manera:

- Sistema Operativo: Scientific Linux.

- Conjunto de usuario único para todos los nodos.
- Directorios HOME, bajo AFS, que son accesibles desde todos los nodos.
- Sistema batch Torque/Maui (PBS), que dirige los trabajos a los nodos apropiados.
- Sistema MPI para programación en paralelo en uno o varios nodos.
- OpenMP para programación en paralelo en un nodo.

- ***Euler (Constellation cluster)***

  - 144 procesadores Dual Xeon quad-core 3.0 GHz (2 GB por core).
  - 96 procesadores Dual Xeon quad-core 2.96 GHz (2 GB por core).
  - Memoria RAM de 3.8 TB para los 1920 cores (todos ellos para ejecución de trabajos).
  - Conectados por Infiniband, que actúan como frontend para que los usuarios puedan conectarse en forma interactiva. El resto de los nodos actúan como servidores de cálculo accesibles sólo a través del entorno batch. La configuración inicial del cluster es la siguiente:

    - 2 nodos actuando como master para atender a los trabajos en interactivo, edición, etc., y 2 nodos de gestión.
    - 144 nodos actuando como "workers" para ejecutar los trabajos.
    - Sistema operativo: Red Hat Enterprise Linux AS release 4 (Nahant Update 6).
    - Conjunto de usuarios único para todos los nodos.
    - Home directories bajo Lustre accesibles desde todos los nodos.
    - Espacio temporal o de "scratch" para ficheros grandes, sin backup.
    - Sistema batch Torque/Moab (PBS), que dirige los jobs a los nodos apropiados.
    - Compiladores de GNU (versiones 3.4 y 4.1).
    - Compiladores de Intel.

- Varias implementaciones MPI para programación en paralelo en uno o varios nodos.
- OpenMP para programación en paralelo intra-nodo.

Todos los nodos se han interconectado a una red de alta velocidad Infiniband, que se utilizará para la comunicación entre los procesos que colaboran en los trabajos paralelos MPI.

El cluster está basado en el sistema operativo linux (Red Hat Enterprise) y dispone las herramientas habituales de software libre.

Centro de Supercomputación de Galicia (CESGA)

- ***Finisterrae***

El tercer entorno de ejecución corresponde al CESGA, que posee la máquina de cálculo Finisterrae cuyos bloques principales son los siguientes:

- Nodos de proceso rx7640.
- Nodos de proceso Superdome.
- Red de interconexión Infiniband.
- Sistema de almacenamiento HP-SFS.
- Nodos interactivos y servidores de ficheros.
- Librería de cintas.

Los nodos rx7640 y Superdome poseen la misma arquitectura y, desde el punto de vista del usuario, la única diferencia entre ellos es el tamaño del nodo. Los rx7640 son servidores con 16 procesadores y 128 GB de memoria, mientras que los Superdome tienen 128 procesadores y 1TB de memoria. El sistema, adicionalmente, tiene otros 2 nodos Superdome de 64 procesadores cada uno y que proporcionan una memoria de 384 GB.

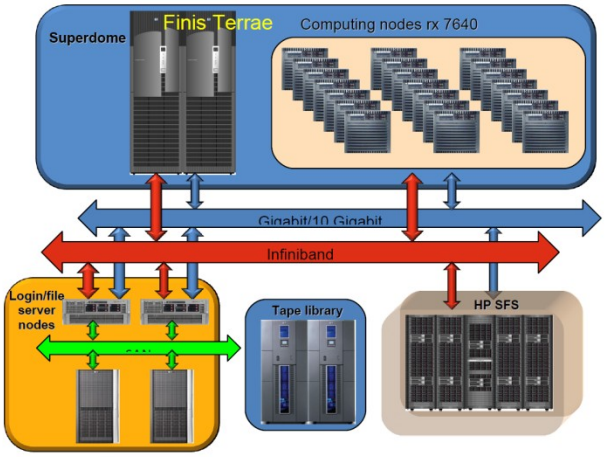


Figura A.1: Esquema del Finisterrae.

Internamente cada nodo está formado por la unión de varias celdas. El rx7640 está formado por dos celdas, mientras que el Superdome por 16. Cada celda incorpora cuatro Itanium Montvale de 1,6 GHz de dos núcleos, con 18MB de caché y 64GB de memoria. Un procesador accede a la memoria local de su celda a través del "controlador de celda", existiendo una interconexión entre cada controlador de celda dentro del nodo, lo que permite a su vez acceder a la memoria de los procesadores de la celda contigua, prácticamente, a la misma velocidad que si fuera la local, pero con una pequeña latencia.

En la *Figura A.1* se puede apreciar el esquema de conexión de los diferentes bloques del Finisterrae. De estos componentes los más importantes para este trabajo han sido los nodos de computación rx7640, en los cuales se han ejecutado las pruebas realizadas, y la red Infiniband, que ha permitido, gracias a su gran velocidad (hasta 16 Gbs) y baja latencia (en torno a 6 microsegundos), la comunicación eficaz entre los procesos.

Anexo III. Instalaciones de Fusión nuclear

Instalaciones CIEMAT

En la Figura A.2 se puede ver un plano general de las instalaciones del reactor TJ-II de tipo stellarator. Alrededor del reactor se encuentran todos los instrumentos de medida, que mediante una red Gigabit transmiten los datos obtenidos al sistema de ficheros asociado al grupo de fusión.

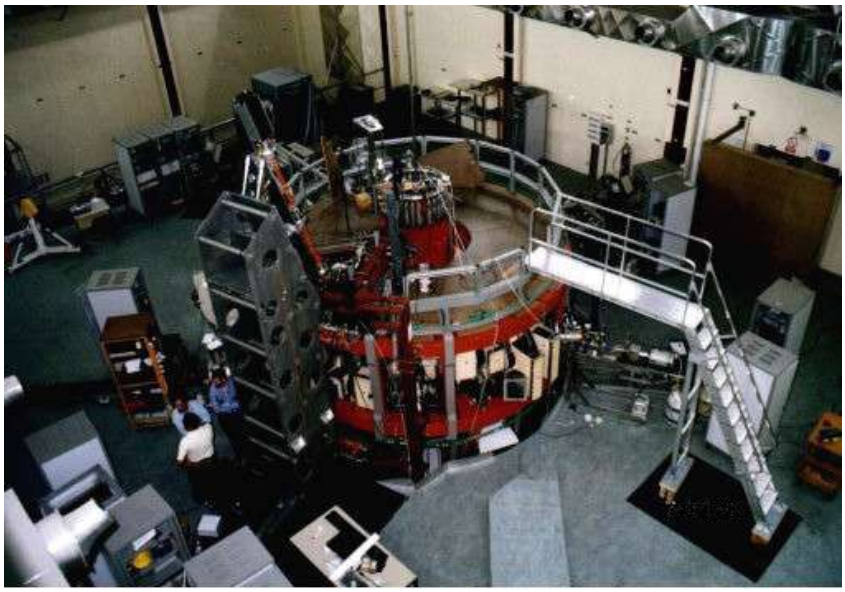


Figura A.2: Instalaciones TJ-II en el CIEMAT.

Instalaciones JET

JET, sigla en inglés de *Joint European Torus*, es un reactor de fusión del tipo tokamak. Se trata del más grande del mundo en la actualidad.

Se encuentra situado en una vieja base de la RAF cerca de Culham, en las afueras de Oxford, en el Reino Unido. Su construcción fue iniciada en 1978, aunque los primeros experimentos no comenzaron hasta 1983.

El JET está equipado con sistemas de manejo a distancia para hacer frente a la radioactividad producida por el combustible de Deuterio-Tritio, que fue el primer combustible propuesto para la fusión. A la espera de la construcción del

ITER, el JET es el único gran reactor de fusión con capacidad de usar este combustible.

En diciembre de 1999 el JET Joint Undertaking, institución que se encargaba del funcionamiento hasta entonces, fue disuelta. La UKAEA (siglas en inglés de Autoridad de la Energía Atómica del Reino Unido) se encargó de la seguridad y del manejo del JET en nombre de los socios europeos. Los experimentos se reanudaron en el año 2000, coordinados por el Acuerdo Europeo del Desarrollo de la Fusión (EFDA).

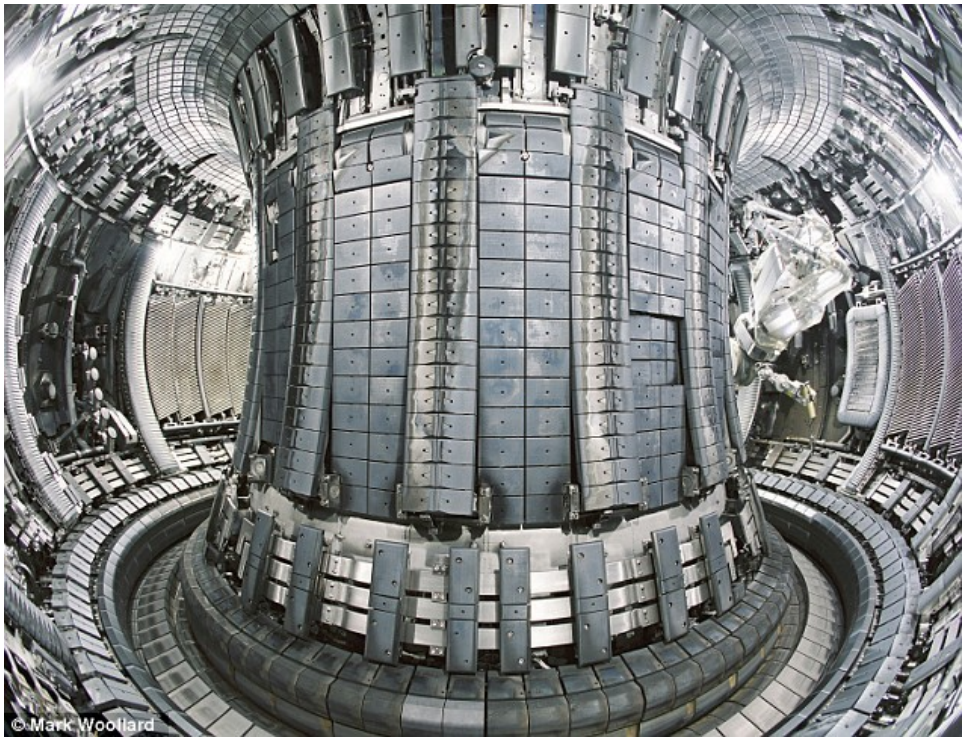


Figura A.3: Esquema del dispositivo