Universidad Nacional
de Educación a Distancia.

Departamento de Informática y Automática.
Escuela Técnica superior de Ingeniería Informática.

Supercomputación aplicada para la identificación de patrones mediante clasificadores probabilísticos en diagnósticos de fusión.

TESIS DOCTORAL

Autor: D. Francisco José Martínez García.
Directores: Dr. D. Sebastián Dormido Canto.
Dr. D. Jesús Antonio Vega Sánchez.

2017

Universidad Nacional
de Educación a Distancia.

Departamento de Informática y Automática.
Escuela Técnica superior de Ingeniería Informática.

Supercomputación aplicada para la identificación de patrones mediante clasificadores probabilísticos en diagnósticos de fusión.

TESIS DOCTORAL

D. Francisco José Martínez García.
Ingeniero Informático.

2017

Agradecimientos.

Este trabajo no se habría podido realizar sin la colaboración de muchas personas que me han brindado su ayuda, sus conocimientos y su apoyo. Quiero agradecerles a todos ellos cuanto han hecho por mí, para que este trabajo saliera adelante de la mejor manera posible.

Quedo especialmente agradecido con mis dos directores de Tesis. El Dr. Jesús Antonio Vega Sánchez y el Dr. Sebastián Dormido Canto me han ayudado y apoyado en todo momento. Han corregido de forma minuciosa este trabajo y me han dado la posibilidad de mejorarlo. Tengo que agradecer sus comentarios, direcciones, sugerencias y las correcciones con las que he podido elaborar una adecuada memoria de todo el trabajo realizado durante estos últimos años. Me acogieron desde un primer momento y me abrieron las puertas de este mundo apasionante. Les agradezco sinceramente su confianza y todo su apoyo. Gracias a ellos he recibido una formación que me ha permitido acometer este trabajo. Gracias.

Mi más sincero agradecimiento a la Dra. Emilia Rodriguez-Solano, del Laboratorio Nacional de Fusión del CIEMAT, por el acceso a las bases de datos de los modos externos. Además, su dedicación en la revisión y comprobación de patrones armónicos así como sus sugerencias.

Debo agradecer toda la ayuda que he recibido de la Unidad de Adquisición de Datos del Laboratorio Nacional de Fusión del CIEMAT. Su ayuda para el tratamiento de descargas fue fundamental para acometer esta Tesis.

Los ensayos han sido realizados en el clúster del Departamento de Informática y Automática de la UNED. He de agradecer toda la ayuda y colaboración recibida del Dr. Jesús Chacón.

Mi agradecimiento más profundo y sabiendo que con estas palabras no puedo expresar lo que siento, a mi familia. Esta Tesis no hubiera sido posible sin la comprensión, paciencia y apoyo incondicional de Elisa, Gonzalo y Jaime. A mis padres y en especial a mi madre por su ayuda.

Para no extenderme demasiado he dejado de nombrar a muchas personas que me han ayudado de una forma u otra en esta Tesis. A todos ellos muchas gracias.

Índice General.

Abreviaturas.

En la redacción de la memoria se ha intentado utilizar los términos traducidos al castellano siempre que ha sido posible. En aquellos casos en los que la traducción estaba algo forzada o podía inducir a equivocaciones se ha preferido mantener el término anglosajón.

	Significado	Uso o procedencia
ANN	Redes neuronales artificiales	Algoritmo Subyacente
BP1	Patrón análisis incremental	Análisis de patrones imagen
BP1	Patrón análisis incremental	Análisis de patrones imagen
BP3	Patrón análisis lineal	Análisis de patrones imagen
CART	Arboles de clasificación	Algoritmo Subyacente
CentOS	Community ENTerprise Operating System	Entidad de investigación
CM	Centro de Masas de una región	Segmentación imagen
DIA	Departamento de Informática y Automática (UNED)	UNED
Dmat	Tipo de datos de PMatlab	Lenguajes de computación
D-T	Deuterio Tritio	Fusión
Ece	Emisión ciclotrónica electrónica	Confinamiento Plasma
ECH	Frecuencia ciclotrónica de los electrones	Confinamiento Plasma
EFDA-JET	European Fusión Development Agreement–JET	Entidad investigación
HIBP	Sonda de iones de cesio	Confinamiento Plasma
HLL	Lenguajes de alto nivel	Lenguajes de computación
HPC	Computación de alto rendimiento	Arquitecturas paralelas
JET	Joint European Torus	Entidad investigación
k-NNR	k-ésimo vecino cercano	Algoritmo Subyacente
LHD	Large Helical Device	Confinamiento Plasma
M/W	Maestro/ *worker*	Arquitecturas paralelas
MC	método montecarlo	Rendimiento
MG	Ganancia Máxima	Machine learning
MIMD	Multiple Instruction Stream, Multiple Data Stream	Arquitecturas paralelas
MIT-LL	Instituto tecnológico de Massachusetts Laboratorio Lincoln	Entidad de investigación
MJ	Mega Joule	Confinamiento Plasma
MR	Medida de relación	Segmentación imagen
MVP	Par de violación máxima	Machine leraning
MW	Mega watios	Confinamiento Plasma
NBI	Inyección de haces de átomos neutros de hidrógeno	Confinamiento Plasma
NPA	Analizadores de partículas neutras	Confinamiento Plasma
NUMA	Non-Uniform Memory Access	Entornos paralelos
ORNL	Laboratios Oak ridge	Entidad investigación
Pa	Medida de presión Pascal	Confinamiento Plasma
RA	Relación de aspecto	Segmentación imagen
RGB	Colores de imagen	Imagen

RHEL	Red Hat Enterprise Linux	Entidad de investigación
RMSE	Root Mean Square Error	Raiz cuadrada de MSE métrica de regresión
SIMD	Single Instruction Stream, Multiple Data Stream	Arquitecturas paralelas
SMO	Optimización Mínima Secuencial	Machine learning
SPMD	Un solo programa, múltiples datos	Arquitecturas paralelas
SVM	Máquinas de vectores soporte	Aprendizaje estadístico
UMA	Uniform Memory Access	Entornos paralelos

Símbolos.

	Significado	Uso o procedencia
A	Aceleración obtenida. Ley de Gustafson	Rendimiento paralelo
θ	Ángulo de pendiente de una región	Segmentación imagen
Φ	Ángulo que determina el patrón BP3	Segmentación imagen
p	Aumento de M para ley de Sun y Ni	Computación paralela
b	Bias. Desplazamiento	Máquinas vector soporte Clasificación
ϵ	Eficiencia minima	Escalabilidad. Rendimiento paralelo
E_p	Eficiencia paralela	Computación paralela
$A_e(i)$	Elipse equivalente	Segmentación imagen
β	Equilibrio de carga	Rendimiento paralelo
λ	Factor de aumento. Adatron	Aprendizaje automático
K	Función kernel	Máquinas vector soporte
$\mathcal{H}$	Hiperplano de separación de clase	Máquinas vector soporte
I	Imagen	Segmentación imagen
σ_B^2	Índice de medida intensidad-umbral	Binarización
I	Instancia de un programa paralelo	Computación paralela
n	Densidad de partículas	Modo de configuración del plasma
ρ	Margen de distancia al hiperplano	Máquinas vector soporte
C	Margen suave	Hiperplano separación SVM
μ_T	Media de nivel de grises	Binarización
MJ	Mega Joule	Medida de energía
s	Modelo de Sun y NI	Rendimiento paralelo
r_k	Nivel de gris	Segmentación imagen
P	Número de procesadores	Computación paralela
f	Segmento de código secuencial	Rendimiento paralelo
Q	Programa paralelo	Rendimiento paralelo
β	Razón entre la presión cinética del plasma y la presión magnética	Modo de configuración del plasma
R_s	Región de una imagen	Segmentación de imágenes
Sp	Speedup aceleración paralela	Rendimiento informático
N	Tamaño de datos	Rendimiento paralelo
M	Tamaño de memoria	Rendimiento paralelo
T	Temperatura del plasma	Modo de configuración del plasma
t	Tiempo de confinamiento del plasma	Modo de configuración del plasma
T_p	Tiempo de proceso paralelo	Computación paralela
T_s	Tiempo de proceso secuencial	Computación paralela
ξ	Variables de holgura	Hiperplano separación SVM
$\omega_B^2(t)$	Varianza entre clases	Binarización
w	Vector de pesos	Machine learning SVM

Índice de figuras.

Índice de tablas.

Índice de Algoritmos.

Introducción y objetivos de la Tésis.

Motivación.

A pesar de los grandes avances tecnológicos y del poder de cálculo alcanzado, cuando se trata de trabajar con grandes bases de datos esto representa un problema. Si al tratamiento de estos grandes volúmenes de información se le añade la complejidad de tratar con imágenes la dificultad aumenta notablemente. El diseño de técnicas y algoritmos eficientes que resuelvan adecuadamente problemas complejos de optimización es uno de los campos dentro de la investigación informática con mayor repercusión en la actualidad.

El constante desarrollo que tienen las aplicaciones que requieren cómputo de alto rendimiento y los avances científicos, originan que las mismas sean diseñadas con mayores niveles de complejidad y precisión. No obstante, estas precisiones necesitan de mayor poder computacional para ser ejecutadas de una forma considerablemente más rápida [Culler 1999]. Por este motivo, el cómputo de altas prestaciones (*High Performance Computing* (HPC)) evoluciona constantemente para adaptarse a los requerimientos de cómputo de aquellas aplicaciones que tienen como objetivo reducir el tiempo de ejecución [Shameem 2006]. Uno de los beneficios de esta constante evolución es la integración en los clúster de nodos con procesadores multicore (anglicismo de multinúcleo) [Ramanthan 2005].

La mayor parte de los sistemas actuales están diseñados mediante nodos con procesadores multicore. Estos nodos de cómputo más potentes plantean nuevos retos, como apunta Muresano en [Muresano 2009], referentes a la gestión jerárquica de comunicaciones, frente a las necesidades de los programadores de arquitecturas paralelas en mejorar las métricas de rendimiento de los sistemas paralelos.

Anderson [Anderson 2006] apunta también que las diferentes rutas de comunicaciones entre los cores hacen que los clústers con procesadores multicore se consideren heterogéneos, debido a que los enlaces de comunicación presentan distintas velocidades de transmisión y anchos de bandas. Estas diferencias de velocidad pueden crear degradación en el rendimiento de la aplicación paralela afectando principalmente al tiempo de ejecución y por consiguiente al *speedup*

(anglicismo de ganancia de velocidad) y a la eficiencia [Anderson 2006, Darema 2001, McCool 2008].

Además, el paralelismo debe hacer frente a los distintos tipos de configuraciones existentes en los entornos multicores, motivado por el número de cores por procesador (2, 4, 8,… *n*), la memoria compartida (Caché L1, L2, L3), etc. Por estos motivos la gestión de trabajos se convierte en un reto cuando se ejecutan aplicaciones que tienen dependencias de la información localizadas en otros nodos; como las aplicaciones desarrolladas bajo el enfoque del paradigma SPMD (*Single Program Multiple Data*). La figura 1 muestra el efecto de una configuración geográfica del clúster DIA[1]. La gráfica es el resultado del test de *ping-pong.* En el test del *ping-pong* se envían y reciben varias veces una serie de mensajes de tamaño variable cronometrando el tiempo empleado en ello. Para cada tamaño se calcula la mitad del tiempo de ida y vuelta (round-trip) . Los tiempos dependen de la ubicación de los *workers*. Una configuración de *workers* distribuidos significa que se encuentran en nodos diferentes y geográficamente distantes. Una configuración local significa que están en el mismo nodo. La figura 1 muestra como afecta la distribución geográfica de los nodos en los tiempos de respuesta.

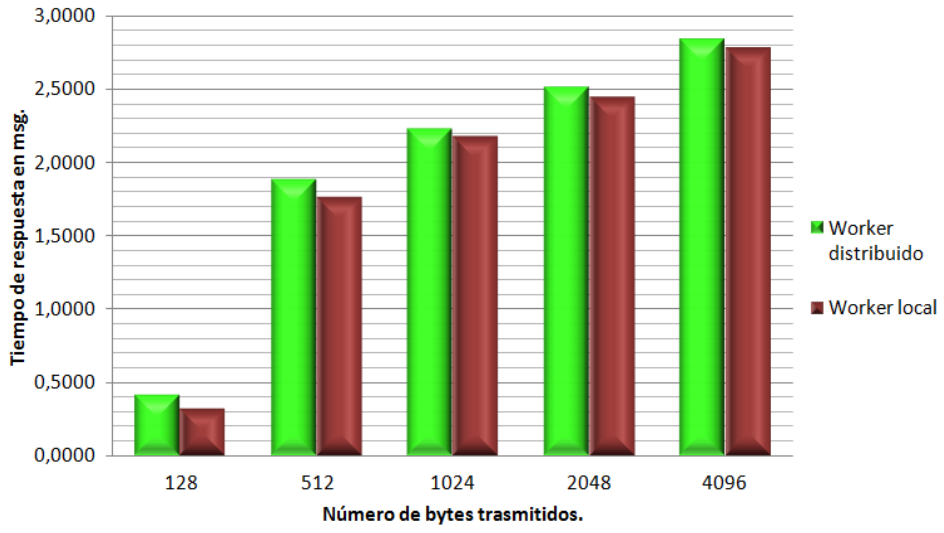


Figura 1. Tiempos de respuesta a la recepción de mensajes del clúster DIA.

[1] Clúster DIA: clúster de supercomputación del Dpto. de Informática y Automática de la UNED compuesto de cinco nodos y 40 cores.

Otro elemento importante que se debe tener en cuenta en la programación paralela es que al utilizar los clústers con procesadores multicore, muchas de las aplicaciones tradicionales mediante paso de mensaje han sido diseñadas para ser ejecutadas en clústers compuestos por nodos con un procesador monocore (anglicismo de mononúcleo), donde la comunicación interna no estaba presente. Sin embargo, cuando estas mismas aplicaciones son ejecutadas en clústers con procesadores multicore, los procesos MPI (interfaz de paso de mensajes) deben intercambiar información con otros procesos que pueden ó no estar localizados en el mismo nodo.

Este intercambio de información entre procesos, puede ocasionar grandes desbalanceos por las distintas velocidades y latencias de los enlaces de comunicación, que al final originan retardos en la ejecución paralela [Darema 2001, Muresano 2009].

Adicionalmente, para ejecutar una aplicación eficientemente en estos entornos jerárquicos de comunicación, se debe considerar el paradigma de programación paralela con el que ha sido diseñada la aplicación. Buyya [Buyya 1999] precisa algunos ejemplos de paradigmas paralelos como por ejemplo el maestro/*worker* (M/W), el SPMD, la segmentación, el divide y vencerás, etc. Cada uno de estos paradigmas se diferencia entre sí debido a que tienen un comportamiento y un patrón de comunicación definido. Estos patrones dentro de la aplicación paralela deben ser caracterizados y evaluados con el fin de ajustar la aplicación lo mejor posible al entorno de ejecución paralelo a utilizar.

Objetivos.

El trabajo realizado en esta Tesis se ha centrado en dos paradigmas de programación paralela, estos son maestro/*worker* (M/W) y SPMD. Y de forma más concreta en la toolbox *parallel distributing* de Matlab®. Básicamente esta toolbox está diseñada con librerías de paso de mensajes. El paradigma maestro/*worker* (M/W) consiste en la existencia de un proceso maestro que divide y distribuye las tareas entre los diferentes *workers*. Una vez que cada *worker* ha terminado su tarea, envía sus resultados al maestro para que compile toda la información y la presente. El paradigma SPMD se caracteriza por ejecutar el mismo programa en todos los *workers* y para un conjunto diferente de datos [Buyya 1999, Darema 2001]. Este comportamiento origina que, al ejecutar una aplicación en un entorno multicore, como puede ser un video-juego, la misma pueda presentar retrasos por el

sincronismo propio de los *tiles* (anglicismo que indica parte o sección de una imagen). Estos *tiles* pueden verse afectados seriamente por los diferentes niveles de comunicaciones que introducen los procesadores multicore, debido a que los mismos incluyen diferencias con respecto a las velocidades y las latencias de comunicación. Las diferencias ocasionan que se deba esperar hasta que el enlace más lento comunique su información para comenzar la siguiente iteración de la ejecución.

Sin la computación de alto rendimiento sería imposible el análisis de ciertos objetos y extraer su información de forma inmediata. A modo de ejemplo se puede mencionar las dificultades que se presentan en la segmentación de imágenes. El objetivo de la segmentación de imágenes es procesar la imagen global mediante la detección de un conjunto de regiones que la componen. La identificación de estas regiones depende de aspectos como pueden ser cambios de textura, cambios de luz, curvas, líneas, etc. Se puede entender también por complejidad la propia naturaleza ambigua de una imagen. En ocasiones la naturaleza humana obvia ciertos detalles fundamentales y que, en función del entorno pueda tener o no cierta repercusión. A pesar de parecer una tarea simple el reconocer un cierto objeto en una imagen, la automatización de esta acción común para el ojo humano puede convertirse en una tarea ardua para un algoritmo de procesamiento de imágenes, y éste es el problema que los algoritmos de segmentación de imágenes tratan de resolver. Pichel dice que es aquí donde la programación paralela optimiza los resultados debido a que es posible repartir entre los diferentes nodos el estudio de cada zona de la imagen [Pichel 2006]. La segmentación de imágenes dispone de varios métodos de análisis [Canny 1986, Petrou 2004, Gonzalez 1978]. El algoritmo puede ser paralelizado aplicando diferentes métodos de segmentación en diferentes nodos. Esta es una posible forma de optimizar el algoritmo debido a que el resultado final sería el obtenido por el nodo que devuelve los mejores resultados o bien realizar una combinación de toda la información obtenida por cada nodo. La naturaleza de cada nodo puede hacer de la tarea de segmentación una labor pesada en el sentido que, en muchas ocasiones es necesaria una captura inicial de los datos.

Los ficheros que almacenan imágenes suelen tener en ocasiones tamaños del orden de varios Megabytes. Como se ha comentado anteriormente en muchas ocasiones es necesario enviar estos ficheros a través de la red. Si el nodo en curso dispone de una arquitectura con memoria común se debe extremar la métrica de caracterización de la carga [Lublin 2001, Calzarossa 1993].

Un mal ajuste de la carga a priori puede llegar a bloquear el clúster y eso motiva que ésta métrica sea tan relevante como difícil de ajustar si se tiene en cuenta que no siempre se conoce en tiempo de ejecución la carga que va entrando dinámicamente al sistema.

La segmentación de la imagen tiene diferentes objetivos y uno de ellos puede ser la automatización de la búsqueda de ciertos patrones [Dormido-Canto 2006]. En particular, en entornos de fusión nuclear en ocasiones es necesario analizar el comportamiento de una señal y posteriormente definir un método automático para el análisis [Vega 2008a]. Verificar, por ejemplo, cómo progresa una determinada señal en frecuencia. Estos comportamientos de la señal pueden ser analizados como patrones [Dormido-Canto 2006]. Si se tiene en cuenta que una imagen es la representación tiempo/frecuencia de una señal, cabe pensar que cobra especial importancia conocer en que posición dentro de la imagen y en que momento temporal se manifiestan estos patrones. En este contexto, puede ser necesario, por ejemplo, conocer si cierto fenómeno se produce en una banda frecuencial determinada.

Una vez determinado el patrón a investigar es muy probable tener que clasificarlo. En este sentido, la clasificación se justifica debido a que en muchas ocasiones el objetivo es encontrar un patrón que venga definido por determinadas características como pueden ser: el rango frecuencial donde se manifiesta, la aceleración de progreso ó el intervalo de tiempo en el que se produce entre otros. Todas estas características pueden ser tomadas como variables de entrada a un sistema de clasificación. Si se añaden muchos atributos al patrón en curso, éste puede terminar siendo una variable con muchas características [Dormido-Canto 2008, Vega 2008b]. Si esta búsqueda se debe realizar en una gran base de datos resulta evidente que la programación paralela es de gran ayuda cuando las decisiones a tomar deben ser rápidas.

En el área del reconocimiento de patrones y en especial en la parte de clasificación, las máquinas de vectores soporte (SVM) [Vapnik 1995] se han convertido en una de las técnicas más importantes en relación a otras como pueden ser el *k-ésimo* vecino cercano más (*k*-NNR), las redes neuronales artificiales (ANN) y los árboles de clasificación (CART). Enmarcado en el campo del aprendizaje automático, el beneficio de utilizar SVM es que son capaces de extraer patrones complejos de los datos sin la necesidad de un entendimiento previo del comportamiento de los mismos. Una máquina de vectores soporte es un sistema de aprendizaje que es capaz de analizar datos, a partir de la extracción de sus

características y relaciones entre ellos [Dormido-Canto 2004]. Una de las limitaciones que tienen las máquinas de vectores soporte es que están pensadas para resolver problemas de dos clases, pero en la práctica no es raro encontrar problemas con dos o más categorías. Para resolver esta limitación existen diversas alternativas como pueden ser estrategias *multi-objetivo* que hacen uso de varias SVM binarias o bien estrategias *mono-objetivo.* En este contexto, en muchas ocasiones no es suficiente con realizar una única predicción por lo que es necesario añadir una medida de probabilidad a cada una de las predicciones realizadas. Todo este procedimiento se realiza mediante la *predicción conformal* [Vovk 2005].

La predicción conformal utilizada en el desarrollo de esta tésis es la basada en los predictores Venn [Vovk 2005] con taxonomía SVM para la predicción de clase y aplicado a datos multiclase. En el clasificador que se ha implementado, el conjunto de datos de entrenamiento utilizado se modifica para cada muestra que se incorpora al sistema. Para cada nuevo dato es necesario calcular de nuevo todos los resultados para posteriormente dar una predicción los más fiable posible [Vega 2015]. La programación paralela es una herramienta de indudable ayuda para conseguir resultados en el menor tiempo posible.

Estructura de la memoria.

El trabajo que se presenta tiene dos partes diferenciadas unidas por el nexo de la computación de alto rendimiento. Por un lado está la paralelización del análisis de patrones en grandes volumenes de datos como pueden ser las descargas asociadas a los dispositivos experimentales de fusión termonuclear. Como se verá en el capítulo cuatro cada descarga de plasma está compuesta por varias señales y su tamaño en bytes es muy alto. No cabe duda que la programación paralela es una herramienta para obtener resultados rápidamente. Por otro lado está la paralelización de métodos para la clasificación y predicción de imágenes de los diferentes estados de operación del dispositivo de fusión TJ-II.

El capítulo 1 muestra el marco de aplicación asi como una introducción a la fusión nuclear y se describe el concepto de Estellarator como dispositivo de generación de plasmas.

El capítulo 2 presenta una introducción a la programación paralela así como a las diferentes arquitecturas y estrategias cuando se programa en un sistema de computación de alto rendimiento. Posteriormente se detallan diferentes entornos paralelos. Se presenta el entorno de Matlab generado en el clúster DIA y que ha

servido de plataforma para esta Tesis. Por último se presentan diferentes algoritmos que han sido codificados y sometidos a pruebas de capacidad en el clúster DIA con intención de analizar su respuesta sobre todos los recursos del clúster.

El capítulo 3 muestra las aplicaciones implementadas. Se realiza una introducción al análisis de patrones en entornos de fusión nuclear bajo una perspectiva de computación paralela. Se presentan las máquinas de vectores soporte y la necesidad de implementar computación paralela para resolver los problemas planteados.

El capítulo 4 detalla la metodología paralelizada que se propone para la detección de los diferentes patrones en espectrogramas de señales en de plasmas nucleares. Se muestran los resultados obtenidos en el proceso ejecutado mediante el clúster de supercomputación DIA. En otra línea el capítulo 5 analiza en detalle el algoritmo SVM. Se presentan distintas implementaciones del algoritmo asi como las estrategias para paralelizarlos. Se finaliza el capítulo presentando fragmentos del código implementado y analizando la implementación de las características de los datos de entrada en los clasificadores basados en SVM

El capítulo 6 detalla las pruebas realizadas en base al código para implementar un predictor Venn bajo taxonomía SVM. También aquí se analiza la importancia de los datos y los valores de los diferentes parámetros que garantizan la convergencia del predictor. Además se muestra la taxonomía definida y su paralelización. Para finalizar el capítulo 7 presenta las conclusiones de los resultados y propone unas líneas futuras de investigación afines a esta Tesis.

Capítulo I.
Fusión nuclear.

1.1 Introducción.

Por definición la fusión nuclear es una reacción que se produce cuando dos núcleos poco densos sometidos a elevadas temperaturas se fusionan y forman un núcleo más pesado; esta reacción libera gran cantidad de energía nuclear.

Pero, ¿cuál es el motivo por el que se necesita esta energía?. La razón está en el consumo elevado de los combustibles fósiles (figura 1.1). Los recursos fósiles tienen importantes efectos no deseados como pueden ser: la escasez de reservas, la falta de garantías en su suministro, los efectos medioambientales tanto en su explotación como consumo, etc. Se hace por tanto necesaria la búsqueda de fuentes masivas de energía que cubran el hueco que dejarán los combustibles fósiles en el futuro. Entre ellas jugará un papel importante la energía fusión termonuclear [Ciemat 2015].

El objetivo final de la investigación a nivel mundial en la fusión nuclear es el desarrollo de la fusión como fuente inagotable de energía eléctrica que sea segura y

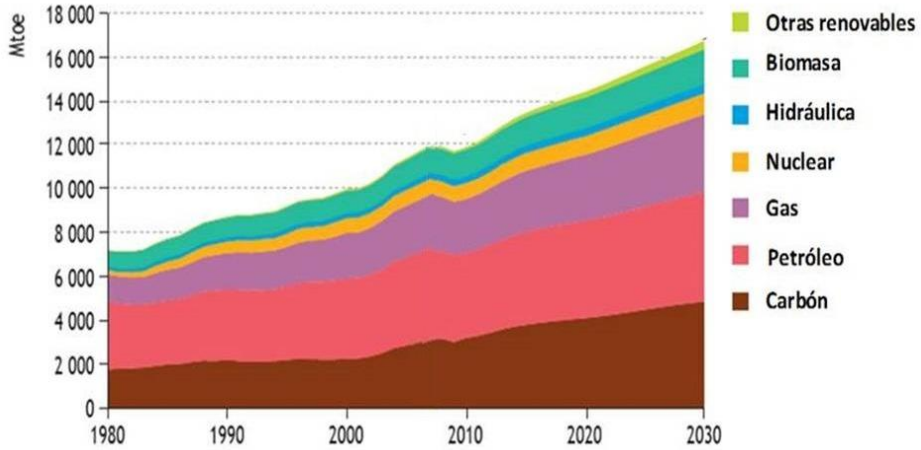


Figura 1.1. Evolución del consumo mundial de energía primaria.

económicamente viable. La energía de fusión puede potencialmente proporcionar una fuente casi ilimitada de energía limpia y sostenible. El problema de los residuos tóxicos se reduce de forma muy considerable y los reactores de fusión tendrían casi

suministros ilimitados de combustible. La situación geográfica de los reactores de fusión podrían estar en cualquier parte del mundo. La energía de fusión está, sin embargo, todavía en fase de desarrollo y no se espera que sea una realidad antes de mediados de este siglo.

1.2 La fusión nuclear fuente de producción de Energía.

En la búsqueda de formas alternativas de energía que cubran el hueco que dejarán los combustibles fósiles, en el futuro tendrá un papel importante la energía nuclear. Existen dos formas de obtener energía en base a reacciones nucleares.

Una de ellas consiste en dividir un núcleo pesado en dos o más núcleos más ligeros (fisión nuclear). La suma de las masas de los núcleos ligeros generados es menor que la del núcleo inicial y ese decremento de masa se transforma en energía, según la expresión $E = mc^2$.

La otra forma de obtener energía es a partir de reacciones nucleares y es justamente la opuesta a la anterior, es decir, consiste en unir dos núcleos ligeros, normalmente hidrógeno en sus dos isótopos: Tritio y Deuterio. En base a la unión se forma otro núcleo más pesado (fusión nuclear). Se entiende por núcleo ligero aquel átomo que presenta pocos neutrones y protones en su núcleo, como es el caso del hidrógeno. En este caso, la masa del núcleo creado es menor que la suma de las masas de los núcleos fusionados, lo que se traduce igualmente en liberación de energía.

Para poder producir una reacción de fusión deben unirse, como se ha indicado, dos núcleos ligeros, lo que en condiciones normales no sucede debido a la fuerte repulsión electrostática entre ambos núcleos, cargados ambos positivamente. Para que los núcleos superen dicha repulsión y se acerquen lo suficiente para producir reacciones de fusión estables se necesitan altas temperaturas y presiones, que conviertan los elementos en un plasma donde los electrones, separados del núcleo, y los iones, se muevan independientemente a gran velocidad.

Son posibles varias reacciones de fusión entre elementos formados por núcleos ligeros. Sin embargo, desde el punto de vista de la mínima temperatura necesaria para conseguir fusionar dos núcleos, la más interesante es la reacción entre el deuterio (^{2}H) y el tritio (^{3}H), ambos isótopos del hidrógeno (figura 1.2). Para dicha reacción se necesita una temperatura que ronde los 100 millones de grados centígrados para poder garantizar que dichas reacciones se producen a un ritmo

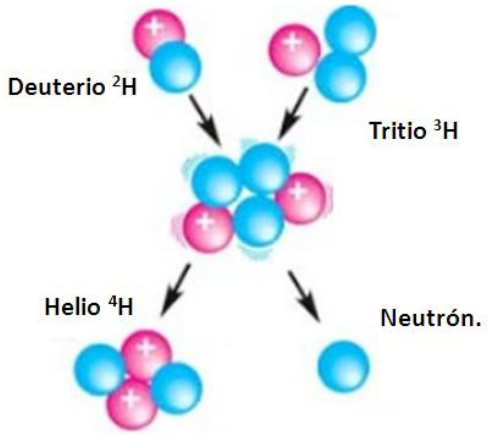


Figura 1.2. Reacción de fusión de Deuterio (^{2}H) – Tritio(^{3}H).

adecuado. En esta reacción se libera nergía según la expresión (1.1).

$$^{2}H + {}^{3}H + {}^{4}H + n + 17{,}6MeV \qquad (1.1)$$

La energía nuclear de los dispositivos experimentales de fusión termonuclear de primera generación hará uso de la fusión de los núcleos del deuterio y del tritio, debido a que es la reacción de fusión más fácil de conseguir, aunque tecnológicamente no la mejor. La energía obtenida por esta vía presenta importantes ventajas frente a otras fuentes actuales, como pueden ser:

- No emite gases de efecto invernadero (su único residuo es el helio, que escapa al espacio exterior).
- Sus reservas son inagotables a escala temporal humana (en esencia el combustible se obtiene del agua del mar).
- Todos los países tienen fácil acceso al combustible.
- El desmantelamiento de sus centrales no generaría residuos de alta actividad.
- Tiene un poder energético muy elevado (con 1 t de litio se produciría la energía eléctrica que consume todo el planeta durante seis años).
- Se trata de una tecnología segura, pues el combustible se inyecta de forma contínua (no hay almacenamiento de energía potencial dentro del reactor).

1.3 Confinamiento del Plasma.

Como se ha indicado, la temperatura necesaria para que los núcleos se aproximen lo suficiente es de millones de grados. Además, es necesario contener el plasma confinado a esa temperatura durante un tiempo lo suficientemente largo como para que se produzcan muchas reacciones de fusión. Se pueden distinguir dos tipos de confinamiento:

- **Confinamiento inercial**, mediante el cual una diminuta cápsula de combustible (D-T) se comprime mediante láseres de gran potencia. Con este método se confina el plasma durante muy poco tiempo -del orden de los microsegundos- pero a densidades extremadamente altas, lo que permite que se produzcan muchas reacciones.
- **Confinamiento magnético**, permite que el plasma, a elevadas temperaturas, aunque con densidades menores que con el confinamiento inercial, se mantenga aislado de las paredes del dispositivo mediante campos magnéticos. El plasma al estar formado por partículas cargadas, en presencia de un campo magnético, dichas partículas están obligadas a seguir un movimiento helicoidal a lo largo de las líneas del campo, de forma que el plasma queda confinado en una región limitada del espacio. Actualmente ésta es la línea de investigación en la que más esfuerzos y expectativas tiene puesta la comunidad internacional.

Independientemente del método de confinamiento utilizado, los parámetros físicos esenciales que intervienen en el proceso de la fusión son la temperatura del plasma (T), la densidad de partículas (n) y el tiempo de confinamiento del plasma (t). El producto de esos tres parámetros, denominado *triple producto*, es un factor que mide la calidad del confinamiento. Cuando el producto de estas tres cantidades excede un determinado valor (dado por el *'criterio de Lawson'*) entonces se puede decir que el plasma se encuentra en ignición, es decir, la energía liberada por las reacciones de fusión es suficiente para mantener la temperatura del plasma [Ciemat 2015].

Para que un dispositivo de fusión llegue a ser energéticamente rentable debe alcanzar la ignición, que es la condición en la que las reacciones de fusión se automantienen, sin necesidad de fuentes externas de calentamiento, generando a partir de ahí más energía que la que se ha aportado al sistema. En un plasma de deuterio y tritio se consigue cuando el calentamiento del plasma por las partículas alfa liberadas iguala las pérdidas de calor. Es decir, para ello debe tenerse en cuenta el denominado *Criterio de Lawson*, que establece la condición de balance

energético en un dispositivo de fusión en base al producto del tiempo de confinamiento de la energía (t) y la densidad de partículas (n).

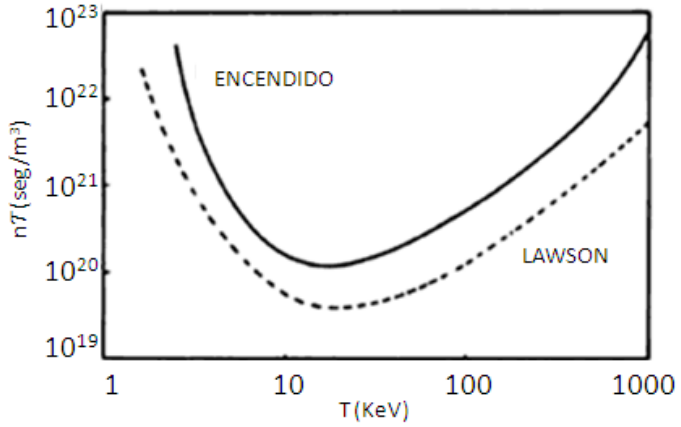


Figura 1.3. Valores del producto de la densidad del plasma por el tiempo que permanece confinada su energía T, necesarios para el emparejamiento de pérdidas y ganancias (criterio de Lawson) y para el encendido, ambos en función de la temperatura del plasma.

En la figura 1.3 se muestra la gráfica que satisface dicho producto en función de la temperatura para una reacción deuterio-tritio. Otro concepto que se maneja con frecuencia en relación a los balances de energía es el de *encendido* de un plasma. Se dice que un plasma termonuclear ha alcanzado el estado de *encendido* cuando la energía de fusión producida (o parte de ella) puede ser reabsorbida manteniendo la temperatura constante. El plasma habrá llegado entonces a una situación autosostenida, donde ya no es necesario que se suministre energía del exterior. La condición de encendido es más fuerte que el criterio de Lawson para el emparejamiento de pérdidas y ganancias.

1.4 Confinamiento Magnético.

Aprovechando el hecho de que las partículas cargadas siguen las líneas de campo magnético, en los años sucesivos a la Segunda Guerra Mundial se diseñaron máquinas lineales de plasma con el objetivo de confinarlas dentro de unos campos establecidos. Este tipo de dispositivos presentaban una gran ventaja, y era su extremada estabilidad y la baja difusión anómala a lo largo del campo magnético. Sin embargo, al ser sistemas abiertos las pérdidas de partículas en su extremo final debido a las partículas que se escapan siguiendo las líneas de campo hace que,

aunque se aumente la intensidad del campo o el tamaño del plasma, el tiempo de confinamiento sea menor. Por lo tanto, este tipo de dispositivos no podrían pensarse como futuros reactores de fusión, y de ahí surge el concepto de máquina toroidal, donde las líneas de campo no tienen un final, sino que generan superficies magnéticas cerradas, y en principio las partículas quedarían perfectamente confinadas. Los dispositivos toroidales son los que han logrado los mejores rendimientos en confinamiento. Dentro de estos dispositivos de fusión destacan dos clases de dispositivos: el *TOKAMAK* y el *ESTELLARATOR*.

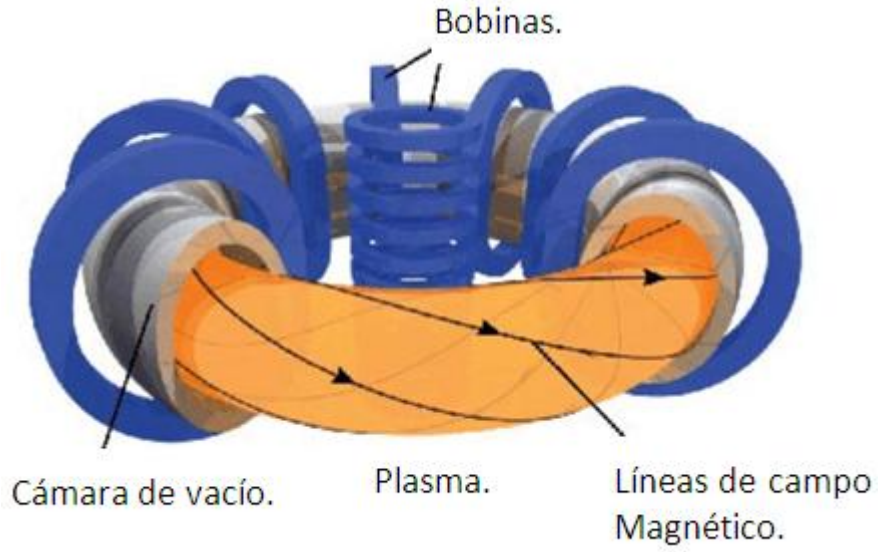


Figura 1.4. Representación esquemática de un Tokamak. Se consigue la generación de corriente por medio de un conjunto de bobinas que emplean el efecto transformador.

El *tokamak*, propuesto por Sajarov y Tamm en 1961 [Tamm 1961], es el dispositivo con el que se han conseguido los mejores resultados en el campo de la investigación en fusión nuclear controlada [Sheffield 1994]. Es simétrico con respecto al eje central, con bobinas externas alrededor del toro que generan un campo magnético toroidal muy alto (varios teslas), véase figura 1.4. El campo magnético poloidal se consigue induciendo una corriente en el seno del plasma en la dirección toroidal. Esta corriente se induce debido a la ley de Lenz (efecto transformador), actuando el propio plasma como circuito secundario. De la combinación de ambos campos resultan las líneas de campo magnético, las cuales tienen forma helicoidal a lo largo del toro. El confinamiento del plasma no se puede mantener indefinidamente sólo con la corriente inductiva, por lo que el *tokamak* es

un dispositivo intrínsecamente pulsado. Mediante inducción de corriente por otros medios y ayudados por la propia corriente de *bootstrap*, parte de esta limitación puede solventarse, con el coste de recircular parte de la potencia de salida y por lo tanto bajando la eficiencia del dispositivo. Sin embargo, la existencia de disrupciones de corriente (pérdida rápida de energía que trae como consecuencia la terminación brusca del plasma), especialmente cuando se experimenta cerca de la frontera de los límites de operación, hace que el *tokamak* no sea el esquema de confinamiento más deseable para un reactor comercial. Los mejores resultados en plasmas de fusión se consiguieron en 1997 en el *tokamak* europeo *JET*, donde en una descarga se produjo el equivalente a 21.7 MJ de energía de fusión, mientras que en otra se alcanzó un pico de potencia de fusión generada de 16.1 MW, usando en ambos casos plasmas compuestos por una mezcla de deuterio y tritio [Keilhacker 2001]. El Estellarator es un dispositivo toroidal de confinamiento magnético que fue propuesto por el astrofísico Lyman Spitzer en 1951 [Spitzer 1958]. Los dispositivos de la clase Estellarator no tienen simetría toroidal, y los campos magnéticos necesarios para confinar el plasma son generados exclusivamente por bobinas externas, sin requerir una corriente eléctrica toroidal en el plasma [Lyon 1990]. Esta característica hace que su confinamiento sea inherentemente estacionario, siempre que se mantengan las corrientes en las bobinas externas. Salvando estas diferencias, la física de los plasmas producidos en Estellarator*s* y *tokamaks* es similar. Debido a que el diseño y la construcción de los Estellarator*s* es más complicado, su desarrollo ha ido siempre una generación por detrás de los *tokamaks*, no alcanzando el rendimiento de estos últimos. Sin embargo, últimamente se ha impulsado la investigación en Estellarator*s*, destacando los Estellarator*s LHD*, en Toki, Japón (cuyo primer plasma se consiguió en marzo de 1998 [Kumazawa 1999]) y *Wendelstein 7-X*, en Greifswald, Alemania finalizada su construcción en 2015 [Wanner 2003]. Ambos utilizan bobinas superconductoras para generar los campos magnéticos, con lo que permiten una investigación más orientada hacia los plasmas estacionarios que serán requeridos en un reactor. La tabla 1.1 muestra algunos de los dispositivos de fusión más relevantes a nivel mundial.

JET	Tokamak	Culham (Reino Unido)
MAST	Tokamak	Culham (Reino Unido
TORE SUPRA	Tokamak	Cadarache (Francia)
ASDEX UPGRADE	Tokamak	Garching (Alemania)
WENDELSTEIN 7-X	Stellarator	Greifswald (Alemania).
TJ-II	Stellarator	Madrid (España)
JT-60	Tokamak	Japon
TFTR	Tokamak	New Jersey (EE.UU)
LHD	Stellarator	Toki (Japon)
KSTAR	Tokamak	Daejon, Corea del Sur
DIII-D	Tokamak	California (EE.UU)

Tabla 1.1. Dispositivos de fusión en el mundo.

En este trabajo JET merece especial atención. Se trata de un dispositivo del tipo Tokamak. Entró en producción en 1983 y fue el primer dispositivo en trabajar con deuterio y tritio. En 1991 alcanzó una potencia de fusión de 1,7 MW, potencia conseguida por primera vez en el mundo. En 1997 consiguió un récord mundial de producción de potencia de fusión con 16,2 MW. Desde enero del año 2000 JET pasó a estar gobernado por EUROfusion, quedando en manos de la asociación británica UKAEA (*United Kingdom Atomic Energy Authority*) la operación y el mantenimiento del dispositivo, mientras los diferentes laboratorios pertenecientes a EURATOM (*European Atomic Energy Community*), entre los que se encuentra el CIEMAT (*Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas*) en España, pasaron a ser los responsables del diseño y la realización de los diferentes experimentos de explotación científica. Actualmente es la mayor instalación de fusión del mundo y la única capaz de trabajar con una mezcla de combustible de D-T. JET es un dispositivo ideal para probar los materiales que han de estar expuestos directamente al plasma, así como prototipos de los sistemas de calentamiento o de diagnóstico, todo ello bajo condiciones próximas a las de los futuros reactores de fusión [www.euro-fusion.org].

1.5 El Estellarator TJ-II.

El dispositivo TJ-II es un Estellarator [Wakatani 1998] de tipo *heliac* flexible [Yoshikawa 1983, Harris 1985] de tamaño medio, cuyo objetivo es el estudio del confinamiento y transporte de energía y partículas en plasmas calientes. Este dispositivo experimental no tiene por finalidad alcanzar la fusión termonuclear, sino producir en condiciones de confinamiento magnético, un plasma de diferentes configuraciones helicoidales, estudiar su comportamiento y ayudar a determinar cuál es el más idóneo para los futuros dispositivos de fusión termonuclear. La disposición de sus bobinas permite una gran variedad de configuraciones magnéticas, y un amplio rango de valores de transformada rotacional ($\tau/2\pi$ en el centro entre 0.9 y 2), siendo ésta su característica más especial, y que lo distingue de otros Estellarator*s*. Además su eje magnético es helicoidal, confiriéndole una extremada tridimensionalidad. En el TJ-II se pueden explorar configuraciones con pequeña cizalladura magnética (-1 a 10 %), profundidad de pozo magnético variable (0 a 6 %) y diferentes radios medios de plasma (0.12 a 0.22 m, con volúmenes entre 0.3 y 1.2 m^3). Además, tiene un alto valor teórico de la β (razón entre la presión cinética del plasma y la presión magnética) de equilibrio y estabilidad, que básicamente representa la eficiencia en el confinamiento del plasma, ($<\beta> > 4$ %) [Alejaldre 2001, Lyon 1990]. El TJ-II tiene una periodicidad M=4 con un campo magnético central de ~1 T y un radio mayor medio de 1.5 m.

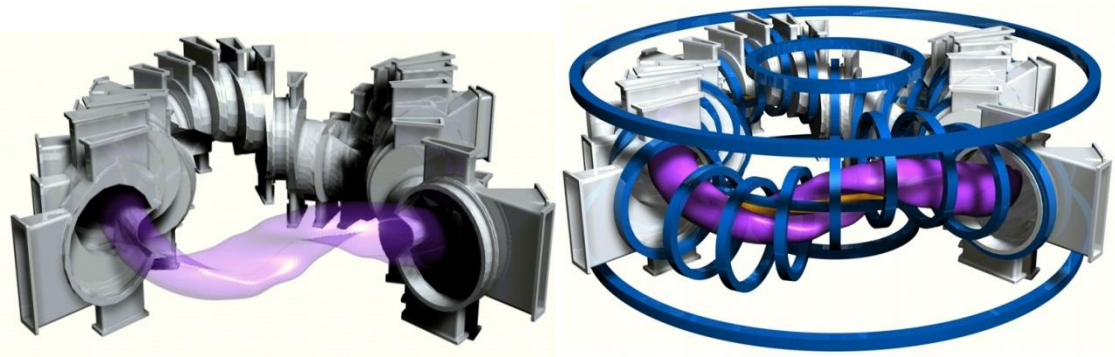


Figura 1.5. Cámara de vacío, bobinas, y representación del plasma (color violeta) del Estellarator TJ-II.

Una característica destacable del TJ-II es que consigue el control efectivo de la transformada rotacional y la cizalladura magnética, lo que permite disponer de un amplio conjunto de configuraciones magnéticas.

En la figura 1.5 puede verse ¼ de la cámara de vacío del Estellarator TJ-II y su plasma. En la figura de la izquierda se aprecia claramente la periodicidad 4, así como la helicidad del plasma alrededor del conductor central. La figura de la derecha muestra las ventanas de observación a través de las cuales puede diagnosticarse el plasma.

Los primeros estudios sobre la configuración magnética del TJ-II se realizaron conjuntamente entre el laboratorio *ORNL* (Oak Ridge, EE.UU.) y el CIEMAT [Hender 1988], cuyo resultado fue el proyecto TJ-II, desarrollado por el CIEMAT [Alejaldre 1990] dentro del programa de fusión europeo. El TJ-II, incluyendo la cámara de vacío, las bobinas y la estructura que las soporta (figura 1.6), tiene un diámetro de cinco metros, una altura de dos metros sobre el suelo de la plataforma experimental y un peso de sesenta toneladas, fabricado en acero de baja permeabilidad magnética (304 LN). A finales de 1997 se consiguieron las primeras descargas con plasma [Alejaldre 1999], y desde entonces el dispositivo ha estado en funcionamiento. La operación del dispositivo se organiza en campañas experimentales que duran varios meses. Después de cada campaña se producen paradas en las cuales se procede a operaciones de mantenimiento y de mejora de la instalación. Para calentar el plasma del TJ-II se utilizan los siguientes sistemas de calentamiento: microondas a la frecuencia ciclotrónica de los electrones (ECH, 1 MW) e inyección de haces de átomos neutros de hidrógeno (NBI, hasta 4 MW).

Los sistemas de control y de adquisición de datos han sido diseñados y desarrollados en su mayor parte por el CIEMAT. La finalidad del programa experimental del TJ-II es profundizar en la física de un dispositivo con eje magnético helicoidal y con gran flexibilidad en las configuraciones magnéticas.

1.5.1 Componentes y características. Bobinas.

El campo magnético del TJ-II es generado por un conjunto de bobinas externas, que le confieren la configuración deseada (figura 1.5). Cada bobina posee un número variable de vueltas, cuya corriente por vuelta depende del tipo de bobina y de la configuración magnética que se desee, llegando desde los 5.2 kA por vuelta en las radiales hasta los 32.5 kA por vuelta en las toroidales. El campo magnético

producido puede alcanzar 1.2 Teslas. La zona de corriente estabilizada tiene una duración nominal, para todas las bobinas, de 1 s con una frecuencia de repetición de un pulso cada ocho minutos. Todas las bobinas se refrigeran con agua que circula a través de orificios longitudinales en los conductores de cobre, y están embridadas por una estructura mecánica para evitar deformaciones.

Cámara de vacío. La cámara de vacío del TJ-II está constituida por ocho octantes iguales (figura 1.5), que se diseñaron para aprovechar la simetría de los campos magnéticos del mismo. Además, cuenta con 92 ventanas de observación para los diagnósticos del plasma, los sistemas de calentamiento y los de inyección de gas. La vasija está preparada para trabajar con presiones menores que 10^{-6} Pa, lo que se denomina ultra alto vacío, siendo su presión base en operación menor que 10^{-5} Pa. Se dispone de un sistema para realizar plasmas ténues de helio, que sirve para eliminar impurezas, principalmente H2O, presentes en las paredes de la vasija. Con el fin de reducir el efecto de las impurezas pesadas (*alta Z*) en el plasma, la cara interior de la cámara de vacío se recubre de boro (~50 nm). Con estas técnicas se pueden conseguir descargas con baja concentración de impurezas y con un apropiado control de densidad [Tabarés 2003], lo cual, como se verá más adelante, constituye un inconveniente a la hora de realizar estudios espectroscópicos relacionados con la dinámica de los iones, ya que las intensidades de las líneas son menores.

Configuraciones magnéticas. Las líneas de campo magnético forman las superficies magnéticas, que son superficies cerradas y anidadas dentro de la cámara de vacío. En primera aproximación, el movimiento de las partículas cargadas está restringido a estas superficies [Miyamoto 1980], y dado que su movimiento a lo largo de las líneas de campo es mucho más rápido que en la dirección perpendicular, presentan las mismas propiedades termodinámicas, como densidad, temperatura y presión. Con el fin de evitar la alteración de las superficies magnéticas de una configuración dada, alrededor del TJ-II sólo se permite utilizar materiales de baja permeabilidad magnética ($\mu r \leq 1.01$ a menos de 3 metros del dispositivo). La configuración magnética está determinada por el valor de las corrientes en las bobinas. Así, la nomenclatura usada en el TJ-II para denominarlas consiste en tomar las décimas de kA-vuelta de las corrientes de la bobina circular

central, la helicoidal y las de campo vertical, que se redondean y se separan por rayas de subrayado. Como ejemplo, la configuración que corresponde a las corrientes por vuelta en las bobinas de $I_{circular}$=10.03 kA, $I_{helicoidal}$=4.01 kA e $I_{vertical}$=6.28 kA, se denomina 100_40_63. Cuando se desea tener una configuración magnética dada se fijan estas corrientes, mientras que la que circula por las

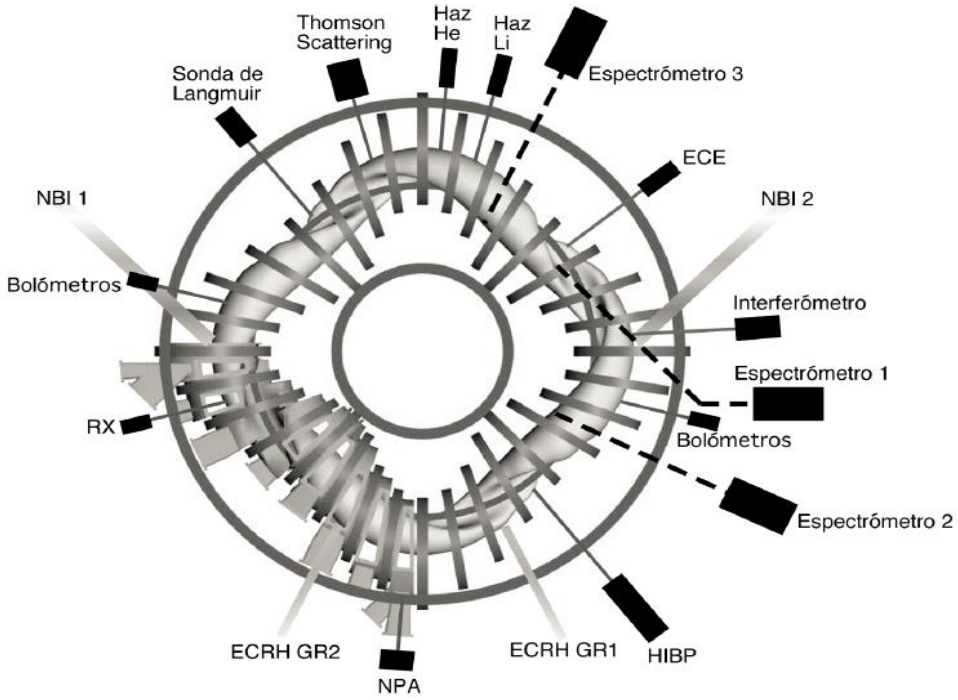


Figura 1.6. Localización de los principales diagnósticos del Estellarator TJII. También se muestran las dos líneas de calentamiento de ECRH (GR1 y GR2), así como las líneas de inyección de partículas neutras para calentamiento (NBI).

bobinas toroidales se ajusta de tal forma que el plasma absorba óptimamente las ondas electromagnéticas de radiofrecuencia empleadas para calentarlo. En la figura 1.6 se muestra la forma teórica de las superficies magnéticas para la configuración estándar en el TJ-II, la 100_44_64, en el corte para el ángulo 0º toroidal. La forma de las superficies varía con el mismo, aunque se repite cada 90º, debido a la periodicidad *M*=4 del campo magnético. Los plasmas de hidrógeno y helio conseguidos con calentamiento por *ECRH* han alcanzado temperaturas y densidades electrónicas centrales de hasta 2 keV y 1.2 1019 m^{-3}, con una energía almacenada en el plasma de 1.5 kJ y una temperatura iónica de 90-120 eV [Alejaldre 2001], pero no simultáneamente.

Sistemas de calentamiento. El TJ-II tiene instalados dos girotrones (generadores de microondas de alta potencia), sintonizados al segundo armónico de la resonancia ciclotrónica electrónica. Cada girotrón es capaz de inyectar en el plasma 300 kW a la frecuencia de 53.2 GHz durante un pulso de duración máxima de 1 segundo [Fernández 2000]. Este tipo de calentamiento, denominado *ECRH*, usa ondas electromagnéticas en modo extraordinario sintonizadas al segundo armónico del giro ciclotrónico de los electrones del plasma alrededor de las líneas de campo [Cairns 1993], con lo que las ondas son absorbidas por éstos. Las ondas son inyectadas perpendicularmente a las superficies magnéticas de la configuración, y los haces se encuentran espacialmente sintonizados para evitar una generación adicional de corriente. La frecuencia a la que son absorbidas las ondas depende de la densidad electrónica, que para los parámetros de las microondas inyectadas en el TJ-II tiene una densidad de corte de $1.2 \cdot 1019\ m^{-3}$ (que equivale a una densidad media de línea de $0.7 \cdot 1019\ m^{-3}$), a partir de la cual las ondas son reflejadas y no calientan el plasma. Desde principios de 2003 opera uno de los haces del sistema de calentamiento por inyección de partículas neutras, con el que se introduce en el plasma un haz de átomos de hidrógeno de alta energía (40 keV) en dirección paralela al campo magnético toroidal y con el mismo sentido (antihorario, visto desde arriba, ver figura 1.5). Al interaccionar con el plasma, los átomos del haz se ionizan y ceden su energía al mismo. Así, se pueden conseguir plasmas con mayores densidades y energías. Los principales parámetros del haz de neutros se encuentran recogidos en la Tabla 1.2. Al encontrarse no compensado también cede momento en la dirección toroidal. Para evitar este efecto, y para alcanzar una mayor β, se encuentra en fase de preparación un segundo inyector, en dirección opuesta al primero [Guasp 1999].

INYECTOR DE PARTÍCULAS NEUTRAS (NBI)	
Potencia inyectada	≤650 kW
Voltaje Accel.	30 – 35 kV
Anchura de pulso	100 – 150 ms
Divergencia teórica	1.3º

Tabla 1.2. Características prinpales del haz de inyección de neutros de hidrógeno para calentamiento.

Sistema de alimentación eléctrica. Para alimentar durante una descarga todo el sistema eléctrico de alta potencia del TJ-II, que comprende las bobinas, los

sistemas de calentamiento y determinados diagnósticos, se dispone de un generador impulsional de 140 MVA, que almacena 100 MJ a 15 kV y 100 Hz.

Operación. La duración de los pulsos eléctricos en el dispositivo TJ-II es de aproximadamente unos dos segundos, en los cuales se alimentan las corrientes de las bobinas, se introducen los gases dentro de la cámara de vacío y se encienden los girotrones, para que calienten el gas. Los girotrones comienzan típicamente 1020 ms después del inicio del pulso y se mantienen emitiendo microondas durante un intervalo de 300 ms. Una vez que se apagan, el plasma se enfría y pierde su energía. El haz de neutros puede ser inyectado en diferentes instantes temporales, dependiendo del experimento que se vaya a realizar. Debido a la existencia de fuertes campos magnéticos y radiación alrededor del TJ-II, el dispositivo se encuentra en un recinto cerrado en el que no se permite la presencia de ninguna persona durante la operación del mismo. La sala de control del TJ-II se encuentra situada junto a la sala experimental, y en ella se encuentran los responsables de la operación y los encargados de los diagnósticos. Así, toda la instrumentación que está en el recinto del TJ-II se opera remotamente, sin que requiera acceso *in situ* durante la descarga. Además, el acceso al recinto del TJ-II durante el tiempo entre las descargas de un día de operación está muy restringido, por lo que es conveniente que todos los controles de la instrumentación que se necesiten modificar entre descarga y descarga sean accesibles remotamente.

1.5.2 Diagnósticos.

El principal objetivo de la diagnosis del plasma es deducir información de su estado a partir de observaciones experimentales de los procesos físicos y los efectos asociados que en él se producen. Para conseguir un rendimiento científico óptimo del TJ-II, se dispone de un conjunto de diversos diagnósticos con el que se pueden determinar las características fundamentales del plasma producido [Sánchez 1993]. En la figura 1.6 se muestra la localización de los principales diagnósticos del TJ-II, así como las líneas de inyección de calentamiento por microondas y de partículas neutras.

Espectroscopía. Se dispone de espectrómetros de baja resolución en el rango del visible para monitorizar la emisión de algunas líneas espectrales de interés de impurezas presentes en el plasma, como la línea 227.1 nm del C^{4+} y la línea 441.5 nm del O^{+}(II). Asimismo, se encuentran instalados espectrómetros de alta resolución en los rangos del visible y del ultravioleta de vacío [McCarthy 1999,

McCarthy 2003], provistos de detectores multicanales de alta sensibilidad, que se utilizan para observar y monitorizar las líneas de emisión de impurezas presentes en el plasma del TJ-II [McCarthy 2001]. También se encuentra operativo un sistema espectral de alta resolución con 8 canales espaciales para analizar la anchura y el desplazamiento de líneas espectrales, con el objeto de determinar la temperatura y la velocidad de rotación de los iones del plasma

Para monitorizar la emisión proveniente de la línea H_α hay un conjunto de fotodiodos de silicio, con filtros de anchos de banda de 1 nm y 10 nm, colocados en diferentes posiciones toroidales del TJ-II. Su intensidad sirve para cuantificar el flujo de partículas.

Bolometría. Los bolómetros son detectores de radiación, con respuesta bien caracterizada y sensibles a todo el rango espectral relevante (2 eV – 4 keV), que sirven para estimar las pérdidas totales de energía del plasma por radiación. En el TJ-II se emplean matrices de fotodiodos de silicio que miden la emisión proveniente de diferentes cuerdas del plasma, y que tras un proceso de inversión permiten calcular su emisión local [Ochando 1999].

Rayos X. Para detectar la radiación en el rango de los rayos X hay instalado un detector de germanio que permite contar los fotones emitidos por el plasma con discriminación en energía. Además, se dispone de cámaras de matrices de diodos que permiten realizar reconstrucciones tomográficas de la emisión de rayos X del plasma, es decir, obtener los valores locales de emisión de rayos X de baja energía [Medina 1999]. Su medida puede emplearse para visualizar la estructura magnética durante la operación del plasma, para la detección de islas, para la obtención de la *Zeff* 2 del plasma, etc.

Esparcimiento Thomson. *Thomson scattering*. El objetivo es determinar los perfiles de densidad y temperatura electrónica en un instante bien definido de la descarga del TJ-II. Para ello se dipone de un espectrómetro provisto de un detector multicanal intensificado. Esto es necesario para registrar el espectro de esparcimiento de un pequeño láser de rubí pulsado pero focalizado en un pequeño volumen de plasma [Barth 1999, Herranz 2003].

Microondas. La evolución temporal de la densidad media a lo largo de una cuerda central se obtiene en el TJ-II con un interferómetro heterodino de 2 mm (140 GHz). Los perfiles locales de densidad electrónica en la zona periférica de gradiente se miden con resolución temporal mediante un reflectómetro de microondas [Estrada 2001]. El perfil de temperatura electrónica y su evolución temporal contínua se

consigue con un radiómetro calibrado absolutamente de 16 canales, que mide la intensidad de emisión alrededor del segundo armónico de la emisión ciclotrónica [De la luna 2001]. Estos canales se denominan canales de *ECE*.

Sondas y partículas. El TJ-II dispone de sondas de Langmuir para obtener información de la densidad, la temperatura y el potencial en el borde del plasma, así como de sus fluctuaciones [Pedrosa 1999]. Para calcular la temperatura iónica y su perfil espacial, se dispone de dos analizadores de partículas neutras (NPA), móviles angularmente, que permiten analizar los flujos y la distribución energética de los neutros rápidos producidos por reacciones de intercambio de carga en el seno del plasma [Fontdecaba 2004]. La evolución espacio-temporal del potencial del plasma se consigue con una sonda de iones de cesio (HIBP), que se inyectan en el plasma con energía entre 100 y 200 keV [Bondarenko 2001].

Magnéticos. Se dispone de bobinas de tipo Rogowski, diamagnéticas y de Mirnov que sirven para determinar, respectivamente, la corriente total, el contenido total de energía del plasma y el nivel de fluctuaciones magnéticas del mismo [Lopez-Bruna 2004].

Haces atómicos. Por medio del estudio de la emisión proveniente de la interacción de haces atómicos (litio térmico y helio supersónico) con el plasma se calculan los perfiles de densidad y temperatura en el borde del mismo [Brañas 2001, Hidalgo 2004].

1.5.3 Adquisición de datos del TJ-II.

Uno de los sistemas esenciales asociados al TJ-II es el sistema de adquisición de datos [Vega 1999a]. En este contexto los dispositivos de fusión pueden clasificarse, entre otros factores, en función de la duración de la descarga. Funcionan como máquinas pulsadas. Es decir, con descargas de corta duración (pulso corto) o descargas de larga duración (pulso largo). El TJ-II funciona en modo de pulsos de una duración aproximada de 500 ms. Tras cada pulso, se reconfigura todo el sistema para el siguiente pulso y así sucesivamente. El sistema de adquisición de datos tiene, entre otras responsabilidades, dotar a los investigadores e ingenieros de los recursos necesarios para una correcta adquisición y análisis de los datos: gestión y control de los canales de digitalización, gestión y control de los procesos de adquisición [Vega 1999a], gestión y acceso a los datos experimentales [Vega 1999b] así como la integración de datos experimentales procedentes de otros sistemas de análisis o procesamiento.

El ciclo de un pulso consiste en una serie de fases, como muestra la Figura 1.7 [Vega 1999a]. Existe una fase de puesta a punto del sistema donde se configuran los equipos y se establecen los parámetros de adquisición. A continuación, durante la descarga, los canales de adquisición muestrean y codifican los valores producidos por los equipos de diagnóstico y generan señales de evolución temporal, cuyos datos son almacenados localmente. En cada descarga es necesario realizar optimizaciones, esto implica que es importante poder acceder a los datos generados lo antes posible. Los dispositivos de adquisición son independientes y solamente un reducido grupo de ellos está disponible al finalizar cada descarga [Vega 1999b]. Debido a que en cada descarga se genera un gran número de datos esta información está almacenada de forma comprimida [Vega 1999b].

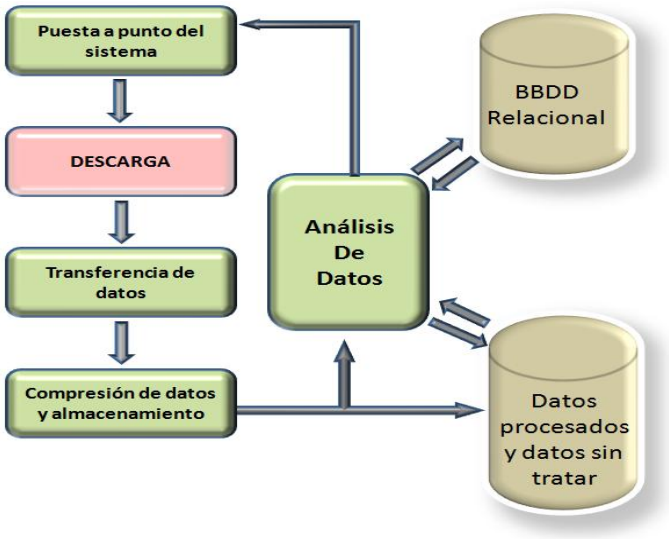


Figura 1.7. Ciclo clásico de operación en fusión **[Vega 1999b].**

El acceso remoto a los datos implica que, la comunidad científica, pueda participar en el progreso diario de la operación sin tener en cuenta su situación geográfica. Al margen del acceso *off-line* de los datos almacenados, los científicos deben poder colaborar en remoto con el equipo responsable de los experimentos en curso. En [Vega 2006] se detalla el acceso y control remoto del dispositivo TJ-II del CIEMAT. El acceso se realizó desde las instalaciones del dispositivo Tore Supra en Cadarache (Francia). Las pruebas contaron con el acceso, gestión on-line y procesado de los datos.

La arquitectura del acceso a los datos del TJ-II esta basada en servidores web y se realiza a través de un navegador. Una vez que se autoriza el acceso, se puede realizar la programación de los canales, de los sistemas de diagnostico de control, acceder a los datos almacenados, a los sistemas de adquisición de datos y al libro de operación [Vega 2003].

Capítulo II.
Programación Paralela.

2.1 Introducción.

En los tiempos actuales la construcción de sistemas paralelos y concurrentes tiene cada vez menos limitaciones y su existencia ha hecho posible obtener gran eficiencia en el procesamiento de datos. Estos sistemas se han expandido a muchas áreas de la Ciencia Computacional e Ingeniería. Como es sabido, existen infinidad de aplicaciones que, utilizando sistemas con un solo procesador, tratan de obtener el máximo rendimiento del sistema para resolver un problema. Sin embargo, cuando el sistema no puede proporcionar el rendimiento esperado, una posible solución es optar por aplicaciones, arquitecturas y estructuras de procesamiento paralelo y concurrente.

El procesamiento paralelo es por tanto una alternativa al procesamiento secuencial. En la computación secuencial un procesador realiza una única operación simultáneamente mientras que en computación paralela hay varios procesadores que cooperan para resolver un determinado problema, lo cual reduce el tiempo de cálculo debido a que varias operaciones pueden llevarse a cabo simultáneamente.

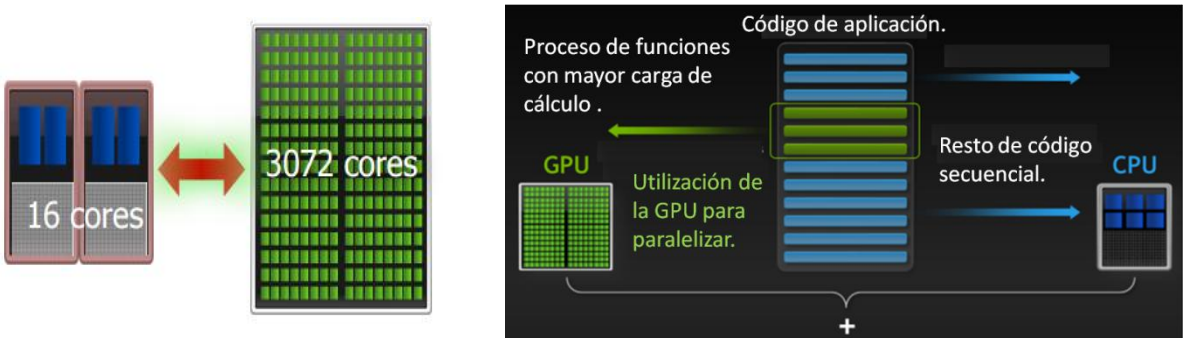
Desde el punto de vista práctico, hoy en día existe suficiente justificación para llevar a cabo investigaciones dentro del procesamiento paralelo y áreas afines (concurrencia, sistemas distribuidos, sistemas de tiempo real, etc.). Parte importante de estas investigaciones son los algoritmos paralelos, metodologías y modelos de programación paralela que se están desarrollando. El procesamiento paralelo envuelve muchos factores que van desde las arquitecturas paralelas y algoritmos paralelos hasta los lenguajes de programación paralela y análisis de rendimiento por mencionar algunos [Clematisa 1999].

En las próximas secciones se analizarán los conceptos más importantes relacionados con el procesamiento/programación paralela, presentando las principales arquitecturas paralelas junto con las estrategias y paradigmas de la programación paralela. A continuación se verán las principales métricas de rendimiento de las aplicaciones paralelas para terminar presentando el entorno de

programación paralelo utilizado en esta Tesis (Matlab Paralelo-Distribuido) y la máquina con la que se han obtenido y validado los resultados.

2.2 Procesamiento paralelo. Objetivos del paralelismo.

El procesamiento paralelo ha provocado un tremendo impacto en muchas áreas donde las aplicaciones computacionales tienen cabida. Un gran número de aplicaciones computacionales en la ciencia, ingeniería, comercio o medicina requieren de una alta velocidad de cálculo para resolver problemas de visualización, bases de datos distribuidas, simulaciones, predicciones científicas, etc. Estas aplicaciones envuelven procesamiento de datos o ejecución de un largo número de iteraciones de cálculo. El procesamiento paralelo es uno de los paradigmas computacionales que hoy en día puede ayudar a realizar estos cálculos de una manera más eficiente. Este enfoque incluye el estudio de arquitecturas paralelas y algoritmos paralelos. Las tecnologías del procesamiento paralelo son actualmente explotadas de forma comercial, principalmente para el trabajo en el desarrollo de herramientas y *frameworks* (anglicismo de entorno de trabajo).



(a) (b)

Figura 2.1. Computación heterogenea. En (a) se utiliza tanto la CPU como la GPU. Cada procesador se encarga de ejecutar aquello en lo que es más eficiente. En (b) se muestra que la mejor estrategia es ver a la CPU y la GPU como mundos complementarios.

El desarrollo de esta área en las redes de computadores ha dado lugar a la llamada *Computación Heterogénea* [Bacci 1999, Clematisa 1999], que proporciona un ambiente donde la aplicación paralela es ejecutada utilizando diferentes computadores secuenciales y paralelos (figura 2.1) en los cuales la comunicación se

se realiza a través de una red inteligente. Este enfoque proporciona un alto rendimiento. Existen muchos factores que contribuyen al rendimiento de un sistema paralelo, entre ellos están la integración entre procesos, es decir, el tener múltiples procesos activos simultáneos resolviendo un problema, la arquitectura del hardware (arquitectura superescalar, de vector o segmentada, etc.), y enfoques de programación paralela, propuestos para comunicar y sincronizar los procesos.

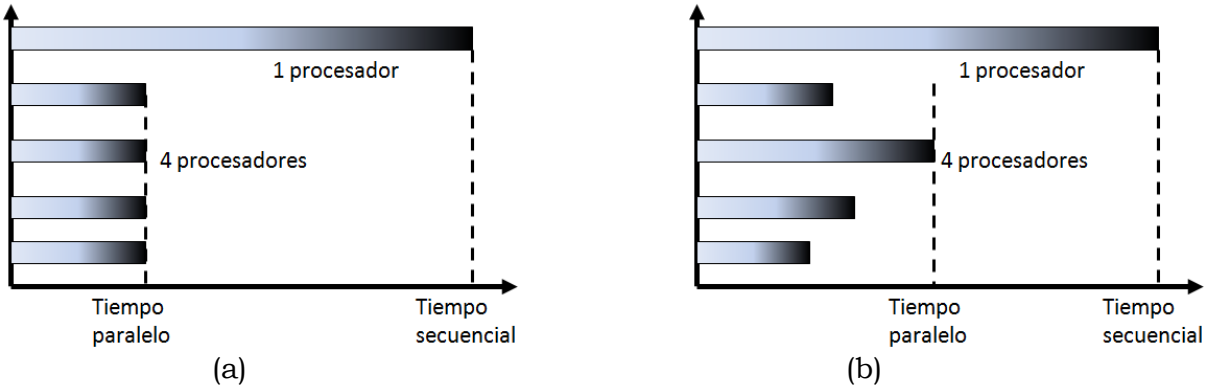


Figura 2.2. (a) Esquema ideal. (b) Esquema real de la paralelización.

Objetivos del paralelismo.

El objetivo del paralelismo es conseguir ejecutar un programa en menos tiempo utilizando varios procesadores. La idea central es dividir un problema grande en varios más pequeños y repartirlos entre los procesadores disponibles.
A veces ocurre que no es posible hacer una paralelización de todo el programa, debido a que aparecen elementos que no se pueden paralelizar. Otras veces ocurre que la repartición del programa entre los procesadores no se hace de manera equitativa, lo que se traduce en una ganancia muy pobre de tiempo. La figura 2.2 muestra un escenario de arquitectura paralela. Si se supone que todos los procesadores llevan a cabo el mismo tipo y número de operaciones, idealmente la ganancia de tiempo debería ser la que se muestra en la figura 2.2.(a), sin embargo realmente ocurre con frecuencia lo mostrado en la figura 2.2(b).

Un ejemplo de paralelización puede ser un ciclo de 40 iteraciones donde cada iteración es independiente del resto. Las iteraciones están repartidas entre 4 procesadores donde uno de ellos ejecutará 10 repeticiones del ciclo original (figura 2.3).

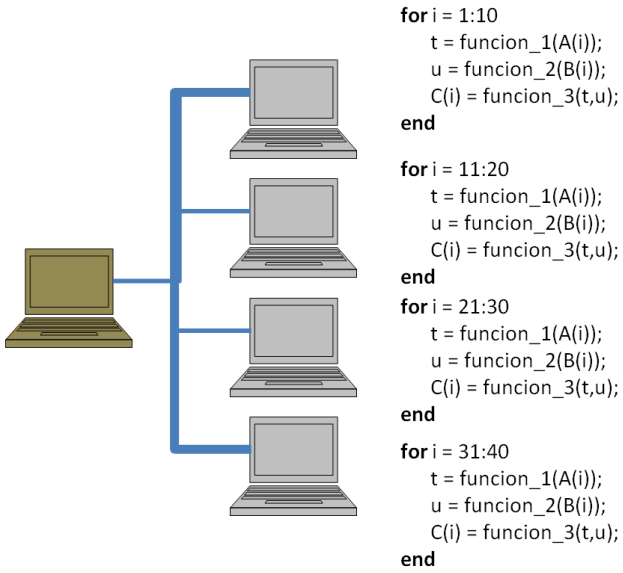


Figura 2.3. Clúster de cinco nodos donde cuatro de ellos trabajan colaborativamente en el procesamiento de un bucle de 40 iteraciones. El bloque *for* llama a su vez a tres funciones que devuelven resultado y éste es almacenado de forma independiente en el vector C común a todos los nodos.

2.3 Programación paralela.

Para poder pasar de un modelo lineal de programación a un modelo paralelo, se debe descomponer el problema en las distintas tareas que lo integran e identificar las dependencias que existen entre ellas. Es necesario identificar las tareas que son independientes y que potencialmente pueden ser ejecutadas de manera simultánea. Existen varios esquemas para la construcción de programas paralelos. Se puede llevar a cabo una paralelización por datos, por tareas o por flujo de control.

Paralelismo de datos.

En este tipo de paralelismo, se descompone el problema dividiendo los datos a ser procesados en varios subconjuntos que son destinados a cada uno de los subprocesos. En [Agarwal 2012] se puede ver como se diseñan planes para conseguir este objetivo. Cada hilo realiza las mismas operaciones pero con un subconjunto distinto de datos, de tal manera que en conjunto se encargan de procesar la totalidad de los datos. En otras palabras, se divide el trabajo a realizar entre los n procesos o hilos.

Paralelismo por tareas.

Este es el método más sencillo de implementar debido a que se paralelizan las tareas que son independientes entre sí y que procesan sus propios datos, por lo que el nivel de sincronización y de dependencia entre los hilos es mínimo.

Paralelismo por flujo de control.

En este tipo de paralelismo se tienen dependencias entre los distintos subprocesos [Hwang 1993], por lo que se debe manejar muy bien la sincronización y los tiempos de latencia. Es común que en ocasiones los datos deban ser preprocesados para luego poder ser divididos y asignados a distintas instancias de código o bien que se tenga que esperar a que todos los subprocesos hayan terminado una determinada etapa para poder recombinar los resultados y pasar a una siguiente parte del proceso o terminar.

2.4 Algoritmos paralelos.

Uno de los ingredientes más importantes para el procesamiento paralelo son sin duda los algoritmos paralelos. Dado un problema a resolver en paralelo, el algoritmo describe como puede resolverse el problema pensando en una determinada arquitectura paralela, mediante la división del problema en subproblemas, comunicando los procesadores entre sí y posteriormente uniendo las soluciones parciales para obtener la solución final. Por ejemplo, los algoritmos basados en la estrategia de *divide y vencerás* usualmente tienen una naturaleza fundamentalmente paralela. Existen dos enfoques en el diseño de algoritmos paralelos con respecto al número de procesadores disponibles.

El primero, es el diseño de un algoritmo en el cual el número de procesadores utilizados por el algoritmo es un parámetro de entrada, lo que hace que el número de procesadores no dependa del tamaño de entrada del problema. El segundo enfoque, permite que el número de procesadores utilizados por el algoritmo paralelo crezca con el tamaño de la entrada, de tal forma que el número de procesadores no es un parámetro de entrada, pero sí una función del tamaño de entrada del problema. Utilizando el esquema "*división de labor*", un algoritmo diseñado con el segundo enfoque puede siempre ser convertido a un algoritmo del primer enfoque. Dentro de los algoritmos paralelos tiene mucha importancia la forma en que los distintos procesadores pueden llevar a cabo su propia comunicación. Los patrones de comunicación en los procesadores son rara vez arbitrarios y no estructurados.

En cambio, las aplicaciones paralelas tienden a emplear patrones de comunicación predeterminados entre sus componentes. Si los patrones de comunicación más comúnmente utilizados son identificados en términos de sus componentes y de sus comunicaciones, es posible entonces crear un entorno o un marco que esté disponible mediante abstracciones de alto nivel para utilizarse en la escritura de aplicaciones. Esto proporciona un enfoque estructurado que puede ser acomodado dentro de un lenguaje orientado a objetos utilizado como lenguaje de programación paralela.

2.5 Planificación estática y dinámica.

Una de las fases de la solución de aplicaciones paralelas es la asignación de tareas a procesadores. En esta fase se trata de realizar la asignación de manera que se minimice el tiempo final de la aplicación. Dicha asignación se lleva a cabo mediante un proceso de planificación, éste determina dónde y cuándo debe ejecutarse cada tarea, es decir, en qué procesador y en qué orden dentro de cada procesador deben ejecutarse las tareas de la aplicación. Cuando el proceso de asignación se centra únicamente en decir donde deben ejecutarse las tareas se denomina *mapeo* [De Giusti 2008]. Es vital para una aplicación paralela disponer de una planificación eficiente para lograr un alto rendimiento [Yu-wong 1996]. La asignación de tareas puede realizarse de forma dinámica durante la ejecución; en este caso no se tienen en cuenta características del comportamiento de la aplicación pero sí puede hacerse teniendo en cuenta los parámetros que reflejan la situación del sistema en cada momento de la ejecución. También la asignación de tareas pueden hacerse de forma estática cuando el tiempo de ejecución de las mismas y las dependencias de comunicación y sincronización son conocidas a priori [Al-mouhamed 1990, Fernandez 1973]. El objetivo de una planificación estática es asignar los nodos del grafo de tareas a los procesadores minimizando el planificador (esquema que indica cuales son las tareas de la aplicación y el momento en que cada una debe ejecutarse teniendo en cuenta las comunicaciones entre las mismas y sin violar las restricciones de precedencia). Para que un planificador sea eficiente debe tener una longitud corta y el número de procesadores utilizados ser razonable [Al-mouhamed 1990, Fernandez 1973]. En [Gerasolis 1992, Po-Jen 2000, Uppaluri 2003] se puede ampliar sobre este tipo de planificación. En [Rusell 2009] se puede ver una iniciación a la planificación dinámica donde se realizan actividades de planificación concurrente en tiempo de ejecución aplicadas a

problemas dinámicos. Es un enfoque general adecuado para un amplio rango de aplicaciones, ya que puede ajustar la distribución de la carga en función de la información del sistema en tiempo de ejecución, pero en muchos casos no se utiliza la información adicional que incluye la carga global de la aplicación según se muestra en [Yu-wong 1996]. Una estrategia que combine las ventajas de la planificación estática y dinámica debe ser capaz de generar una carga balanceada sin incurrir en un gran *overhead* (anglicismo que define el tiempo extra de una operación cuando se le añade una nueva funcionalidad). Esto es posible con técnicas avanzadas de planificación paralela, donde todos los procesadores cooperan para ejecutar un trabajo. En esta Tesis se utilizará la asignación de planificación estática y será tratada en el capítulo 4. Dado que la asignación se realiza de manera previa a la ejecución de la aplicación, el *overhead* requerido por el algoritmo de asignación no tiene efectos sobre la ejecución de la aplicación.

Para realizar esta asignación, se debe encontrar una forma de representar las características más importantes del comportamiento de la aplicación paralela tal como se indica en [Díaz 2006], entre ellas se pueden mencionar: tiempo de cómputo de cada tarea, comunicación entre tareas, dependencia de datos y requerimientos de sincronización.

2.6 Arquitecturas paralelas.

Aún cuando el campo de las computadoras paralelas está en constante cambio, aparecen nuevos modelos de paralelismo. Dada la gran diversidad de sistemas que hay hoy en día, se sigue aceptando una división genérica dentro de los multiprocesadores: *Memoria distribuida, Memoria compartida* y *Memoria compartida distribuida* englobada dentro de la *Memoria compartida*.

Memoria Distribuida.

La figura 2.4 muestra un computador donde la memoria del sistema está distribuida entre todos los procesadores. En este tipo de computadoras paralelas se trabaja con el modelo de programación de *paso de mensajes*. La comunicación entre procesadores es explícita, es decir, el programador es quien la define [CEPBA 2005].

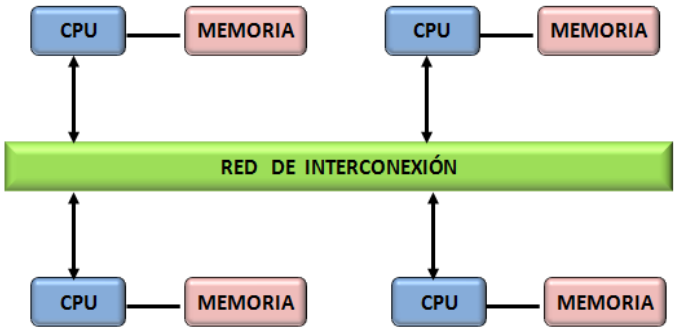
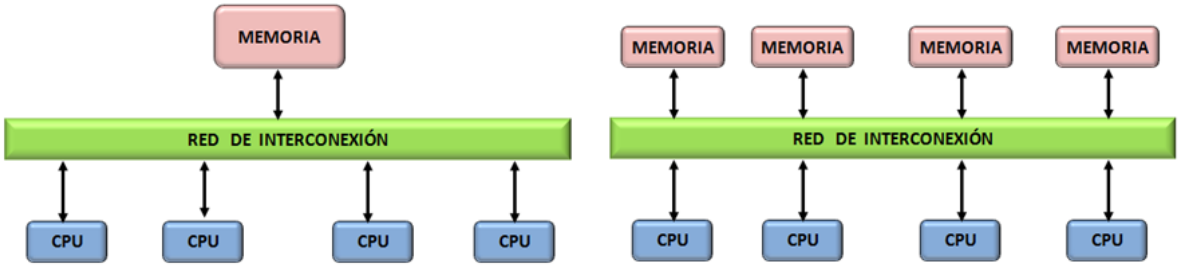


Figura 2.4. Esquema de un procesador con memoria distribuida

Memoria Compartida.

Se define como un sistema donde todos los procesadores acceden a toda la memoria, es decir, varios procesadores pueden acceder a la misma localidad de memoria y en el mismo instante de tiempo. Según sea uniforme o no el tiempo de acceso a memoria por parte de los procesadores, se tendrán dos tipos de memoria. La memoria *NUMA* (*Non-Uniform Memory Access*), donde los procesadores comparten la memoria del sistema de forma que, dependiendo de donde esté ubicado el procesador, puede tardar más o menos en acceder a la memoria, véase figura 2.5(a); y la memoria *UMA* (*Uniform Memory Access*), donde los procesadores comparten la memoria que se encuentra distribuida por todo el sistema ofreciendo así un acceso uniforme, figura 2.5(b) [CEPBA 2005].



(a) (b)

Figura 2.5. (a) Modelo NUMA, (b) Modelo UMA.

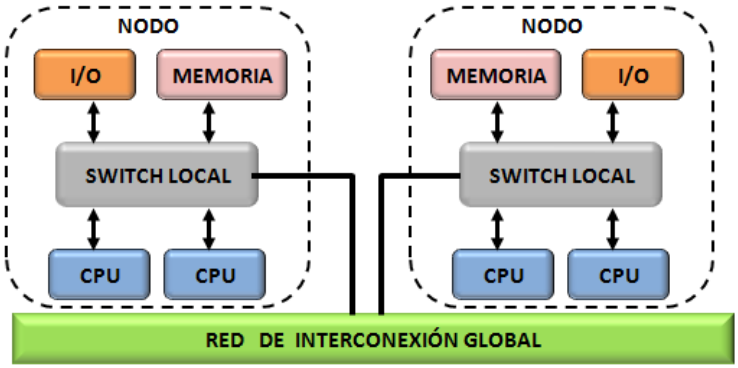


Figura 2.6. Modelo DSM de Memoria Compartida Distribuida.

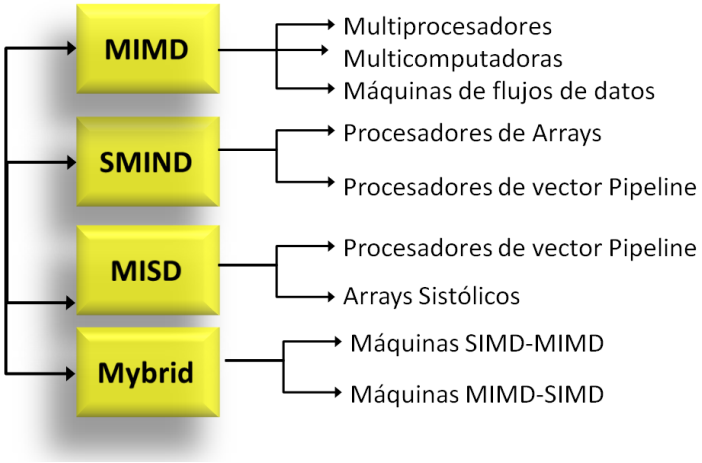


Figura 2.7. Clasificación de Flynn de las arquitecturas de procesamiento paralelo.

Memoria Compartida y distribuida.

Por último y dentro de esta categoría están los sistemas DSM (figura 2.6). Estos sistemas son una combinación de los dos modelos anteriores.

La clasificación de Michael Flynn [Flynn 1995] divide a las arquitecturas paralelas en la familia de las computadoras *SIMD*, *MISD* y *MIMD*. La figura 2.7 muestra su taxonomía.

2.7 Arquitecturas SIMD, MISD y MIMD.

SIMD (*Single Instruction Stream, Multiple Data Stream*). En una máquina SIMD, se supervisan varios elementos mediante una unidad de control. Todas las unidades de procesamiento reciben la misma instrucción desde la unidad de control, pero

operan con diferentes conjuntos de datos, los cuales provienen de distintos flujos de datos (figura 2.8).

Las características principales de este tipo de máquinas paralelas son las siguientes:

- Distribuyen el procesamiento sobre una larga cantidad de hardware.
- Operan concurrentemente con muchos elementos de datos diferentes.
- Realizan el mismo cálculo en todos los elementos de datos.

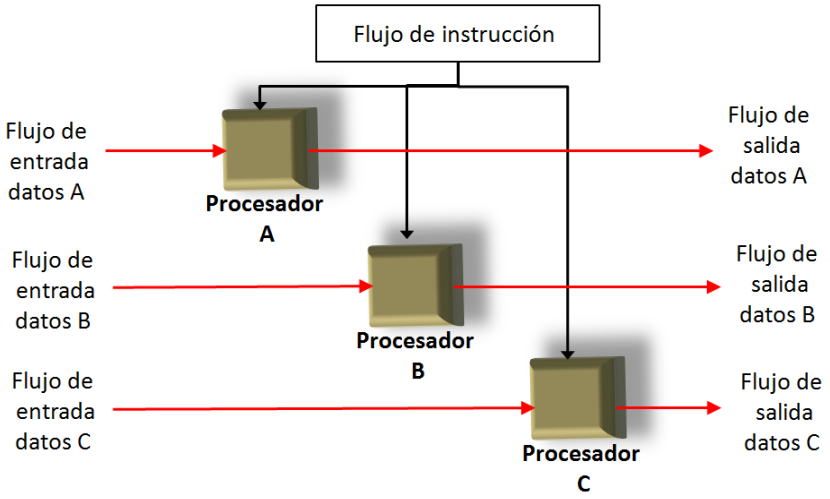


Figura 2.8. Modelo de una arquitectura SIMD.

Cada unidad de procesamiento ejecuta la misma instrucción al mismo tiempo, y los procesadores operan de manera síncrona. El potencial de speedup (anglicismo para definir la aceleración de procesador) [Amdhal 1967] de las máquinas SIMD es proporcional a la cantidad de hardware disponible. Algunas características generales de las computadoras SIMD son:

- Menos hardware, debido a que utilizan solo una unidad global de control.
- Menos memoria, ya que solo se necesita una copia de las instrucciones colocada en la memoria del sistema.

La arquitectura SIMD dispone de un flujo de instrucciones lo que hace de una aplicación SIMD que sea entendible, fácil de programar y fácil de trazar.

Con este planteamiento, la unidad de control ejecuta instrucciones de control de flujo y operaciones escalares comunes a todos los elementos, mientras los procesadores están ejecutando otras instrucciones. Lo que implica menos costo

puesto que solo se necesita un decodificador de instrucción simple en la unidad de control. Como ejemplos de computadoras SIMD mencionar la ILLIAC IV, MPP, DAP, CM-2, MasPar MP-1 y MasPar MP-2 [Seller 1999].

MISD (*Multiple Instruction Stream, Single Data Stream*). Las máquinas de esta categoría pueden ejecutar varios programas distintos con el mismo *ítem* de datos. La arquitectura puede ser dividida en dos categorías:

1. Sistemas con diferentes unidades de procesamiento y que pueden ejecutar diferentes instrucciones pero con los mismos datos. Es importante mencionar que, este tipo de arquitecturas es más una teoría que una configuración práctica.
2. Sistemas con un flujo de datos que circula sobre una serie de elementos de procesamiento. Las arquitecturas segmentadas como pueden ser los vectores sístolicos entran dentro de este grupo de máquinas.

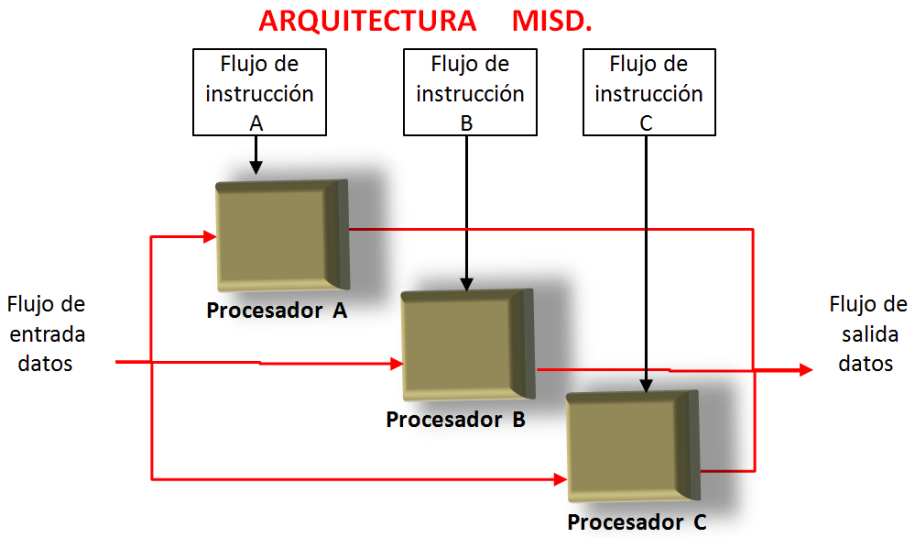


Figura 2.9. Arquitectura MISD**.**

La figura 2.9 representa la estructura general de una arquitectura MISD. Los sistemas MIMD ejecutan operaciones en paralelo de manera asíncrona; los nodos activos cooperan pero operan independientemente. Las arquitecturas MIMD difieren con la interconexión de redes, procesadores, técnicas de direccionamiento de memoria, sincronización y estructuras de control. La interconexión de redes hace que los procesadores se comuniquen e interactúen unos con otros. Ejemplos de

computadoras MIMD incluyen la Cosmic Cube, nCUBE2, iPSC, Symmetry, FX-8, FX-2800, TC-2000, CM-5, KSR-1 y la Paragon XP/s [Seller 1999].

Las computadoras MIMD se pueden categorizar en *sistemas fuertemente acoplados* y *sistemas débilmente acoplados*, dependiendo de cómo los procesadores accedan a la memoria. Los procesadores en un sistema multiprocesador *fuertemente acoplado* generalmente comparten un sistema de memoria global, estos sistemas son conocidos como *sistemas de memoria compartida*. Aquellos sistemas MIMD *débilmente acoplados* pueden compartir un sistema de memoria, pero cada procesador tiene su propia memoria local. A estos sistemas se les conoce como *sistemas de paso de mensajes*. Las computadoras *fuertemente acopladas* y *débilmente acopladas* corresponden a los sistemas MIMD de Memoria Global (GM-MIMD) y MIMD de Memoria Local (LM-MIMD) respectivamente.

Las computadoras MIMD de paso de mensajes, figura 2.10, se refieren a multicomputadoras en donde cada procesador tiene su propia memoria llamada memoria local o privada, y es accesible solo por su propio procesador. Las arquitecturas MIMD de paso de mensajes están referidas tanto a arquitecturas de memoria distribuida como a arquitecturas de memoria privada.

Un sistema MIMD de memoria compartida, figura 2.10, se denomina *Sistema de Acceso Uniforme a Memoria* (UMA, *Uniform Memory Access*), ya que el tiempo de acceso a memoria es el mismo para todos los procesadores que la comparten. En este modelo, los procesadores pueden comunicarse sin restricciones y de una manera simple compartiendo datos y utilizando un mismo espacio de direccionamiento.

Los datos que se comparten pueden protegerse mediante el uso de métodos de ocultamiento de datos que los modernos lenguajes de programación ofrecen. Este modelo está muy extendido debido a la capacidad de soportar una gran variedad de modelos de programación eficientes.

Esto entra en contraste con los sistemas de paso de mensajes que son más fáciles de diseñar pero más difíciles de implementar o programar. En general, los sistemas MIMD *fuertemente acoplados* proporcionan mayor rapidez en el intercambio de datos entre los procesadores que los sistemas MIMD *débilmente acoplados*.

Ejemplos de computadoras GM-MIMD son la serie CDC 6600 y la Cray XM-P. Ejemplos de computadoras LM-MIMD son la Carnegie-Mellon Cm y la Tandom/16 [Seller 1999].

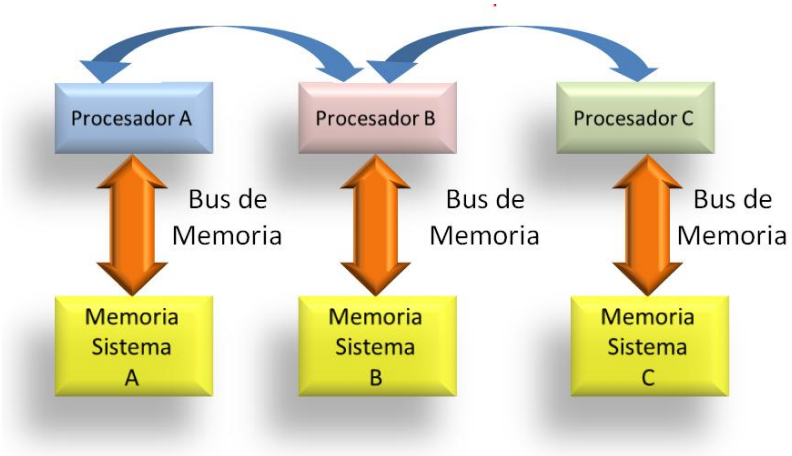


Figura 2.10. Modelo de memoria distribuida en la Arquitectura MIMD.

2.8 Estrategias de programación paralela. Granularidad.

En algún momento la computación paralela sustituirá a la computación secuencial. Si no ha sucedido hasta ahora es porque las tecnologías de desarrollo de programas paralelos no están los suficientemente maduras. A continuación se realiza un repaso a diferentes modelos de programación paralela. Modelos que hacen referencia tanto a la forma de trabajar como a la infraestructura necesaria.

Básicamente existen tres estrategias para diseñar aplicaciones paralelas. La primera está basada en la paralelización automática, la segunda en el uso de librerías paralelas y la tercera está relacionada con el desarrollo de la aplicación. A continuación se detalla cada una de éstas estrategias.

1. **Paralelización automática**: Esta estrategia libera al programador de paralelizar tareas. El compilador es el responsable de generar el código objeto paralelo e integrarlo en el código del programador. Sin embargo, la eficiencia de esta paralelización puede ser baja, ya que depende del compilador.
2. **Utilización de librerías de cálculo paralelizadas**: La idea básica de esta estrategia es la de encapsular código paralelo común a muchas aplicaciones y colocarlo en librerías paralelas, que pueden ser implementadas eficientemente. Estas librerías pueden ser de dos tipos: las que usan tipos de datos abstractos y las que permiten una implementación paralela de rutinas matemáticas.

3. **Implementación de la aplicación paralela**: Esta estrategia ofrece libertad al programador para elegir el lenguaje y el modelo de programación. Sin embargo, en muchos casos es imposible reutilizar el código paralelo.

La figura 2.11 muestra los componentes de la programación paralela y esquematiza la secuencia de pasos necesarios para obtener una implementación paralela utilizando las estrategias de diseño comentadas anteriormente.

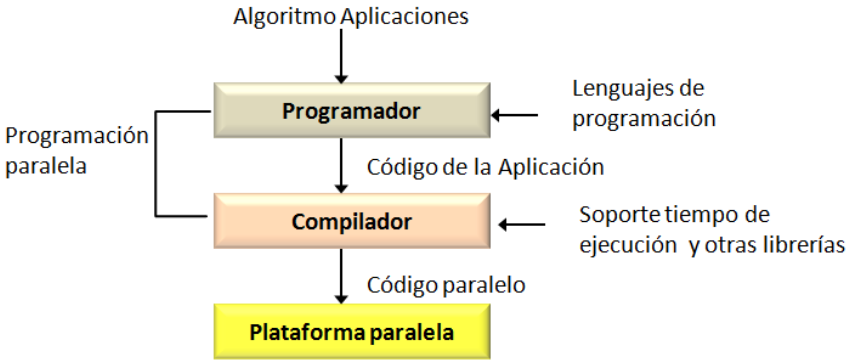


Figura 2.11. Librerías paralelizadas.

Granularidad.

Para garantizar un buen rendimiento del programa paralelo es necesario considerar los costes asociados a las ejecuciones paralelas de las tareas, como por ejemplo, la creación y la gestión de tareas, la migración de las tareas a procesadores remotos y los costes de comunicación (figura 2.12). De esta forma, el paralelismo de una aplicación está relacionado con la granularidad.

En Wilson [Wilson 1993] se define la granularidad como el cociente entre la cantidad de operaciones que puede realizar un proceso y el tiempo de comunicación requerido. Existen diversas clasificaciónes de los niveles de paralelismo [Hockney 1988, Zomaya 1996], entre ellas se destacan las siguientes:

a) Granularidad muy fina: Paralelismo a nivel de instrucciones de lenguaje máquina.

b) Granularidad media: Paralelismo a nivel de subrutinas.

c) Granularidad gruesa: Paralelismo a nivel de aplicación.

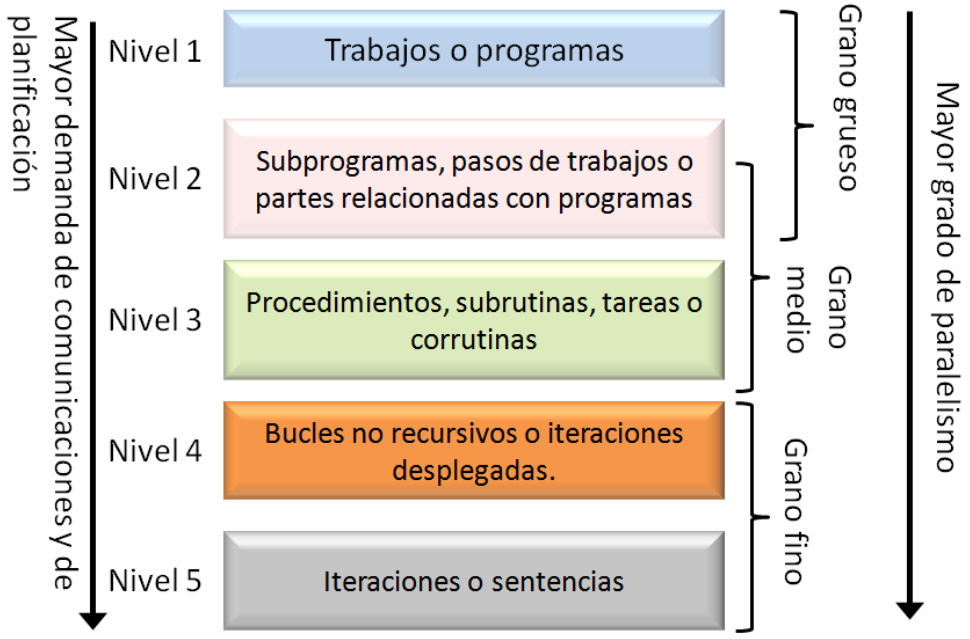


Figura 2.12. Niveles de paralelismo.

El primer nivel de paralelismo está relacionado con el compilador, mientras que los dos niveles siguientes corresponden al programador. Si la granularidad de las aplicaciones es "pequeña", puede ocurrir que los costes asociados a las ejecuciones paralelas de las tareas superen el beneficio de la ejecución paralela. De esta forma, la granularidad que se explote al paralelizar depende de los costes asociados a la ejecución paralela de las tareas. Por lo tanto, la granularidad de las tareas determina el tipo de ejecución (paralela o secuencial). Por otra parte, los costes de sincronización/comunicación en máquinas de memoria compartida son comparativamente pequeños, del orden de los microsegundos (μs). Sin embargo, en arquitecturas de memoria distribuida estos costes son mayores, ya que en algunos casos se alcanza el orden de los milisegundos (ms). En redes de computadores estos costes pueden ser enormes, del orden de los minutos. Por lo tanto, es necesario en cada arquitectura, explotar el grano de paralelismo.

2.9 Paradigmas de programación paralela.

Los paradigmas de programación paralela son clases de algoritmos que solucionan diferentes problemas bajo las mismas estructuras de control [Per Brinch 1993]. Cada uno de estos paradigmas definen modelos agrupados en:

1) Paralelización de una aplicación mediante la descripción general.

2) Distribución de los datos.
3) Interacción entre las unidades de cómputo.

En el momento de diseñar una aplicación paralela es importante conocer que alternativas de paradigmas existen, así como conocer sus ventajas y desventajas. Hay diferentes clasificaciónes de paradigmas de programación, pero un subconjunto habitual es el de Rajkumar [Rajkumar 1999]. A continuación se hace una breve descripción de cada una de ellas.

2.9.1 Divide y vencerás.

En este paradigma el problema se divide en una serie de subproblemas. Se realizan tres operaciones: dividir, computar y unir. La estructura del programa es del tipo árbol, siendo los subproblemas las hojas que ejecutarán los procesos o *threads* (anglicismo para referenciar hilo de ejecución). Cada uno de estos subproblemas se soluciona independientemente y sus resultados son combinados para producir el resultado final. En este paradigma, se puede realizar una descomposición recursiva hasta que no puedan subdividirse más y la resolución del problema se hace a través de operaciones de partición (*split* o *fork*), computación y unión (*join*). También se trabaja con parámetros similares a los que posee un algoritmo de tipo árbol, ejemplos de estos son: la anchura del árbol, el grado de cada nodo, la altura del árbol y el grado de recursión.

2.9.2 Segmentación.

Este paradigma está basado en una aproximación por descomposición funcional. La aplicación se subdivide en subproblemas o procesos y cada subproblema se debe completar para comenzar el siguiente proceso, uno tras otro. En una arquitectura segmentada los procesos se denomina etapas y cada una de estas etapas resuelve una tarea en particular. También es importante destacar que en este paradigma se logra una ejecución concurrente si las diferentes etapas de la arquitectura están llenas o existe en ellas un flujo de datos. Esto no es sencillo de lograr, debido a que la salida de una etapa es la entrada de la siguiente. Uno de los parámetros importantes aquí es el número de etapas, por lo cual la eficiencia de ejecución de este paradigma es dependiente del balanceo de carga a lo largo de las etapas.

2.9.3 Maestro-*worker*.

El paradigma maestro-*worker* también es conocido como *granja de tareas* o maestro-*esclavo.* Este paradigma se compone de dos tipos de entidades: un maestro y varios *workers*. El maestro es el encargado de descomponer el problema en trabajos más pequeños (o se encarga de dividir los datos de entrada en subconjuntos) y distribuir las tareas de la aplicación a los diferentes *workers*. A continuación se realiza una recolección de los resultados parciales para procesar el resultado final. Los *workers* solo cumplen la función de recibir la información, procesarla y enviar los resultados al maestro. Este paradigma es uno de los utilizados en el desarrollo de la Tesis y se analizará detenidamente más adelante.

2.9.4 Spmd.

Es el paradigma más utilizado en las aplicaciones paralelas [Moura 1998] debido a la sencillez y a la gran escalabilidad que posee. En el paradigma *solo un programa múltiples datos* (*spmd*), se ejecuta el mismo código para un determinado número de ordenadores o nodos, pero sobre un subconjunto distinto de los datos de entrada [Oliveira 2003, Quinn 1994]. Es decir, los datos son repartidos entre los diferentes nodos disponibles en el sistema. El paradigma *spmd* describe un comportamiento iterativo en el cual antes de analizar una iteración se realiza una fase de sincronización para no degradar las métricas de rendimiento [Uppaluri 2003]. En esta fase el intercambio de información se hace entre los nodos vecinos. La información de intercambio dependerá de la naturaleza del problema. En la literatura también se conoce como paralelismo geométrico, descomposición por dominios o paralelismo de datos.

2.10 Métricas de rendimiento.

El objetivo del procesamiento paralelo es mejorar la productividad (cantidad de aplicaciones ejecutadas por unidad de tiempo) y/o la eficiencia (disminuir el tiempo de ejecución de la aplicación). Para poder paralelizar una aplicación es necesario explotar la concurrencia con el objetivo de distribuir el trabajo a cada nodo de cómputo y así conseguir un menor tiempo de ejecución de la aplicación. Es decir, la computación paralela permite que las aplicaciones se puedan dividir en varias tareas pequeñas para una ejecución simultánea. Desarrollar aplicaciones de este tipo no es sencillo, pero los beneficios en cuanto a resultados obtenidos, las hacen indispensables para determinados problemas (obtener un buen rendimiento en una

aplicación paralela de este tipo es uno de los principales desafíos dada la gran cantidad de aplicaciones existentes). Las secciones que siguen a continuación muestran las métricas que se utilizan más frecuentemente para la medida del rendimiento en aplicaciones paralelas.

2.10.1 Aceleración (Speedup).

La medición de rendimiento en entornos paralelos es más complejo que en un sistema secuencial. En este contexto, interesa conocer cuál es el beneficio obtenido cuando se utiliza paralelismo y cuál es la aceleración que resulta por el uso de dicho paralelismo. La aceleración o speedup [Amdhal 1967] puede ser definida como la ganancia de velocidad. Básicamente el speedup mide el rendimiento paralelo en comparación con el rendimiento secuencial.

$$S_p = \frac{T_s}{T_p} \tag{2.1}$$

Donde T_s es el mejor tiempo secuencial para la solución del problema y T_p es el tiempo invertido en la solución paralela con P procesadores. De forma ideal se podría decir que S_p podría ser un número cercano a P, es decir, con P procesadores se podría obtener un tiempo de ejecución P veces más rápido. A esta situación se la denomina como el speedup lineal o speedup ideal. En otros casos se puede hablar de $S_p > P$, para este caso se habla de speedup súper lineal, esto puede ocurrir cuando algún procesador aumenta el rendimiento o bien cuando disminuyen los datos de forma local en algún procesador.

Existen diversas formas de expresar la aceleración en función de la forma en que se definan los tiempos de ejecución secuencial y paralelo:

- *Aceleración Relativa.* El tiempo de ejecución secuencial usado es el tiempo total de ejecución del programa paralelo cuando éste es ejecutado sobre un único procesador. Por lo tanto la *aceleración relativa* de un programa paralelo Q cuando se resuelve una instancia I de tamaño n usando P procesadores es:

$$Aceleración\ Relativa(I,P) = \frac{T(I,1)}{Tiempo\ de\ ejecución\ Paralelo} \tag{2.2}$$

- *Aceleración Real.* El tiempo de ejecución secuencial usado es el tiempo del mejor programa secuencial para resolver el problema.

$$Aceleración\ Real(I,P)\ = \frac{T^*}{T(I,P)} \tag{2.3}$$

Donde T^* es el tiempo de ejecución del mejor programa secuencial. Formalmente la aceleración es la relación entre el tiempo de ejecución sobre un procesador secuencial y el tiempo de ejecución en múltiples procesadores.

2.10.2 Eficiencia.

La eficiencia es una medida que relaciona el speedup actual con el speedup ideal. Representa la ganancia de velocidad por procesador. La eficiencia paralela viene dada por la formula:

$$E_p = \frac{S_p}{P} = \frac{T_s}{P * T_p} \tag{2.4}$$

Donde T_s es el tiempo secuencial, P el número de procesadores y T_p el tiempo invertido con el escenario paralelo. La eficiencia proporciona una medida de cómo se están utilizando los recursos de un entorno paralelo. La eficiencia es máxima cuando es igual a uno. Sin embargo se ha visto anteriormente que el speedup puede ser $S_p > P$, entonces para esos casos la eficiencia seria mayor que uno.

2.11 Limitaciones al rendimiento.

Pueden existir múltiples condiciones que limitan el rendimiento de un sistema paralelo. Estas pueden ser el equilibrio de carga, las limitaciones de las comunicaciones, el código fuente, etc. Todas ellas contribuyen de una u otra forma a que el rendimiento del sistema no sea el deseado.

2.11.1 Ley de Amdahl.

Comúnmente cuando se paraleliza un algoritmo o aplicación, existen porciones de código que son inherentemente secuenciales y otras que son paralelizables. La ley de Amdahl [Amdahl 1967a] establece que la aceleración estará determinada por la fracción del código que es mejorada y por tanto es acelerada por dicha fracción. Si f es la parte de código secuencial, Amdahl asume que, al margen del número P de procesadores utilizados, la parte secuencial del código necesita fT unidades de

tiempo para ejecutarse. Por tanto para ejecutar el código en un solo procesador se necesitan (1-*f*)*T* unidades de tiempo. Amdhal indica que, para ejecutar ese mismo código en *P* procesadores, se necesitan (1-*f*)*T*/*P* unidades de tiempo. Si se combinan estos dos tiempos se puede concluir que el tiempo paralelo T_p para ejecutar un código usando P procesadores estará delimitado por la suma de la fracción secuencial *fT* y de la fracción restante (1-*f*)*T* pero dividida por P. Es decir:

$$T_p = fT + \frac{(1-f)T}{P} \tag{2.5}$$

Si el resultado anterior se combina con la expresión del speedup

$$S_p = \frac{T_s}{T_p} \tag{2.6}$$

Entonces se obtiene la conocida ley de Amdhal

$$S_p \leq \frac{1}{f + (1-f)/p} \tag{2.7}$$

Se puede intuir que, para un número infinito de procesadores, el speedup no podría exceder del limite

$$S_p \leq \frac{1}{f} \tag{2.8}$$

2.11.2 Ley de Gustafson-Barsis.

La ley de Gustafson-Barsis considera que la cantidad de paralelismo se escala con el tamaño de los datos (vectores o matrices) de la aplicación. Esta ley es una función alternativa a la ley de Amdahl para el cálculo de la aceleración, y el motivo es que la ley de Amdahl presenta algunas consideraciones que no son del todo apropiadas para aplicaciones del mundo real:

- La ley de Amdahl supone que se toma el mejor algoritmo secuencial y que está limitado por las características del procesador. En una arquitectura multinúcleo con caché independiente para cada núcleo, se podrán separar los datos y manejarse independientemente en cada caché por lo que se reducirá el tiempo de latencia de acceso a la memoria.

- La ley de Amdahl supone que se toma el mejor algoritmo secuencial, pero existen algunos programas que por naturaleza son más eficientes en su versión paralela.
- La ley de Amdahl modela adecuadamente muchos programas de flujo de control pero no modela adecuadamente programas del modelo paralelo en datos. La razón es que no se toma en cuenta el tamaño del problema.
- La ley de Gustafson-Barsis determina el límite en la aceleración que puede obtenerse cuando se paraleliza un programa, y plantea que un programa suficientemente grande puede ser paralelizado de manera eficiente y obtener una aceleración lineal con el número de procesadores o núcleos:

$$A = N + (1 - N) * s \tag{2.9}$$

Donde:

- N es el número de procesadores o núcleos.
- s es la relación del tiempo secuencial y el tiempo total de ejecución.
- A es la aceleración obtenida.

En este caso se puede ver que la aceleración resulta ser lineal. Esta ley es equivalente a la ley de Amdahl pero con la diferencia que es más realista para el caso del cómputo paralelo en procesadores multi-núcleo.

2.11.3 Ley de Sun y Ni.

Esta Ley se aplica a problemas escalables limitados por la capacidad de la memoria. El objetivo es resolver el mayor problema posible que permita la memoria disponible del sistema. Es similar a las leyes de Amdahl y de Gustafson, pero en lugar de mantener fijo el tamaño del problema o el tiempo de ejecución, fija la cantidad máxima de memoria disponible para el problema. La ley se establece bajo la suposición de que, al aumentar la memoria disponible, por ejemplo aumentando el número de procesadores o la capacidad de cada uno, aumenta también la carga de trabajo asignada a cada proceso. Siendo $G(p)$ el incremento de la carga al aumentar la memoria p veces, se obtiene la siguiente expresión:

$$s = \frac{(c + G(p) \cdot f)}{\left(c + G(p) \cdot \frac{f}{p}\right)} \qquad (2.10)$$

Se identifican tres casos especiales:

- $G(p) = 1$ corresponde al caso donde el tamaño del problema es fijo, de manera que la ecuación se simplifica a la Ley de Amdahl.
- $G(p) = p$ corresponde al caso donde la carga se incrementa en la misma proporción que la memoria, que corresponde con la Ley de Gustafson con un tiempo de ejecución fijo.
- $G(p) > p$ corresponde al caso donde la carga se incrementa más que la memoria y, en este caso, probablemente se genera un speedup superior al caso anterior.

Una aplicación de esta ley se podrá ver a lo largo de esta Tesis en el capítulo cuatro donde se trabaja con ficheros voluminosos y que necesitan gran cantidad de memoria para su procesamiento.

2.11.4 Equilibrio y modelos de carga.

El análisis de las cargas de trabajo es importante para la comprensión de cómo se utilizan los sistemas [Lublin 2001]. Además, los modelos de carga de trabajo son necesarios como entrada para la evaluación de nuevos diseños de sistemas, y para la comparación de los mismos. Esto es especialmente importante a gran escala y muy costoso cuando se trabaja con sistemas paralelos. Afortunadamente se dispone de los *logs* (anglicismo para definir los registros de anotaciones del sistema) para analizar su carga y hacer énfasis en aquellos parámetros que son comunes a todos los sistemas. A continuación se analizan alguno de ellos.

Si se tiene en cuenta que cada procesador realiza su trabajo en un tiempo t_i y si se asume que todos los t_i son aproximadamente iguales. Entonces el tiempo de ejecución puede ser definido en términos de t_i de la siguiente forma:

$$T_p = \max\{ t_i : 1 \leq i \leq P\} \qquad (2.11)$$

Es razonable asumir que

$$T_1 \leq \sum_{1 \leq i \leq P} t_i \tag{2.12}$$

Si se dispone de un conjunto de i tareas paralelas donde i=1, …, P , el promedio de ejecución viene dado por

$$prom\{t_i : 1 \leq i \leq P\} \coloneqq P \sum_{1 \leq i \leq P} t_i \tag{2.13}$$

Se define entonces el equilibrio de carga β para un conjunto de tareas paralelas como:

$$\beta \coloneqq \frac{prom\{t_i : 1 \leq i \leq P\}}{P\max\{t_i : 1 \leq \mathrm{i} \leq P\}} \tag{2.14}$$

El caso ideal es cuando β=1 por tanto el equilibrio de carga será optimo en valores cercanos a 1.

Si se relaciona el equilibrio de carga con la ley de Amdahl puede verse que el mejor caso es aquel en el que la carga puede estar repartida en P partes siendo P el número de procesadores. Este es un caso de distribución perfecta de carga. El equilibrio de carga en un entorno paralelo viene dado por la expresión (2.14) si se tienen i procesadores y si se aplica la definición de *eficiencia* se obtiene el siguiente resultado:

$$E_p = \frac{T_1}{PT_p} \leq \frac{\sum_{1 \leq i \leq P} t_i}{P\max\{t_i : 1 \leq i \leq \mathrm{P}\}} = \frac{prom\{t_i : 1 \leq i \leq P\}}{P\max\{t_i : 1 \leq i \leq \mathrm{P}\}} \tag{2.15}$$

Esta relación muestra el tiempo promedio de cada procesador frente al tiempo máximo de cada procesador y es la utilizada para definir el equilibrio de carga. Si en la expresión anterior se sustituye el segundo término, se llega a la conclusión que la *eficiencia* en la resolución de una tarea nunca puede exceder del equilibrio de carga:

$$E_p \leq \beta := \frac{prom\{t_i : 1 \leq i \leq P\}}{P\max\{t_i : 1 \leq \mathrm{i} \leq P\}} \tag{2.16}$$

2.11.5 Escalabilidad.

La escalabilidad está relacionada con la eficiencia. Si se dispone de un número determinado de datos de tamaño N donde se hace variar N de forma significativa, este cambio puede tener mayor impacto en un entorno paralelo que en uno secuencial. La escalabilidad hace referencia a este concepto, es decir, a cómo reacciona el sistema paralelo frente a variaciones. La escalabilidad puede tener dos dependencias, de datos y de memoria. Estas dependencias se analizan a continuación.

2.11.5.1 Dependencia de datos.

La relación de la dependencia de datos del speedup y la dependencia de datos de la eficiencia para un número P de procesadores y un tamaño N de datos viene dada por:

$$S_{P,N} = \frac{T_{1,N}}{T_{P,N}} \tag{2.17}$$

Donde $T_{1,N}$ es el mejor tiempo de ejecución secuencial para un tamaño de datos N y $T_{P,N}$ es el tiempo de ejecución para el mismo tamaño de datos pero con P procesadores. La eficiencia paralela en función de los datos se puede expresar en función del speedup y P:

$$E_{P,N} = \frac{S_{P,N}}{P} = \frac{T_{1,N}}{PT_{P,N}} \tag{2.18}$$

De la expresión anterior se deriva que un algoritmo es escalable si existe una mínima eficiencia $\epsilon > 0$ de tal forma que, dado un conjunto de datos de tamaño N, y un número de procesadores P que tiende a infinito cuando N tiende a infinito, la eficiencia permanece acotada por ϵ, es decir:

$$E_{P(N),N} \geq \epsilon > 0 \tag{2.19}$$

No todos los algoritmos se comportan de la misma forma por tanto cabe pensar que unos algoritmos serán más escalables que otros.

2.11.5.2 Dependencia de memoria.

La memoria $M(N,P)$ requerida para implementar un algoritmo con P procesadores y un tamaño de datos N puede exceder de las limitaciones de físicas de alguno de los procesadores. $M(N,P)$ indica la memoria requerida por cada procesador. Por tanto la memoria requerida para la resolución de la tarea será $P \cdot M(N,P)$. Para la resolución de una tarea secuencial se puede suponer que

$$M(N,P) \leq c_0 + c_1 N \tag{2.20}$$

Donde c_0 y c_1 son constantes que, para el caso secuencial, se hace coincidir con los valores $c_0 = 0$ y $c_1 = 1$. Para el caso paralelo no resulta tan fácil hacer depender $M(N,P)$ *de N y P.* Podría ser posible dividir la memoria uniformemente bajo ciertos requisitos de tal manera que se podrían encontrar unas nuevas constantes c_1' y c_0' dando lugar a:

$$M(N,P) \leq c_0' + \frac{c_1' N}{P} \tag{2.21}$$

Esta expresión sirve para definir la escalabilidad respecto a las restricciones de memoria. Un algoritmo es escalable respecto a la memoria si, dados unos datos de entrada de tamaño N y teniendo un número de procesadores *P(N)* que tiende a infinito cuando N tiende a infinito, la eficiencia $E_{P(N),N}$ está acotada por una constante de tal forma que:

$$M(N,P) \leq M_{max} \tag{2.22}$$

Donde M_{max} es una constante fijada con anterioridad.

2.12 Matlab Paralelo-Distribuido.

Actualmente los sistemas clúster representan la vía más rápida para conseguir computación de alto rendimiento sin grandes inversiones. En este sentido parecía razonable utilizar el clúster DIA (clúster del Dpto. de INFORMÁTICA y AUTOMÁTICA de la UNED) como plataforma física para la realización de esta Tesis. Se tenía conocimiento de las utilidades de Matlab para generar modelos de clasificación pero faltaba formación de entornos paralelos-distribuidos dado que no existían experiencias previas. Como ya se ha comentado, el primer objetivo era instalar el entorno completo de Matlab paralelo-distribuido en el clúster. A la falta de formación de este nuevo entorno, hubo que sumar las eventualidades lógicas de una primera instalación. Se producían diferentes caídas del sistema debido a las inestabilidades del entorno. Fueron necesarias gran número de pruebas hasta conseguir que todos los nodos del clúster fueran estables. Posteriormente se dio paso al comienzo de la formación en entornos paralelos-distribuidos pues era la primera vez que se trabajaba con este entorno de Matlab y más aún con computación paralela. Fue necesario codificar multitud de programas para analizar la efectividad del nuevo entorno en sus diferentes arquitecturas (procesamiento por lotes e interactivo). También se realizaron las primeras pruebas de esfuerzo del clúster. Terminada la etapa de creación y estabilización del sistema se dio paso a la creación de los modelos que se detallan en los siguientes capítulos. Las secciones que siguen a continuación muestran el panorama actual de los entornos paralelos asociados a Matlab. A continuación se detalla el entorno paralelo-distribuido de Matlab y se muestran datos analíticos como resultado de las pruebas de esfuerzo al que se sometió el clúster DIA.

2.12.1 Introducción.

Matlab [Mathworks.a] es un "lenguaje de programación de alto nivel" y un entorno interactivo de computación técnico-científica. Incluye funciones para el desarrollo de algoritmos, análisis de datos, cálculo numérico y visualización [Mathworks.b]. Usando Matlab se pueden resolver problemas de computación técnica más rápidamente que con lenguajes de programación tradicionales, como C, C++ o Fortran [Mathworks.c]. Existen diferentes implementaciones paralelas del lenguaje Matlab desarrolladas por diferentes comunidades. Como por ejemplo: PMatlab, MatlabMPI y MultiMatlab. Algunas de ellas han sido desarrolladas para propósitos o plataformas muy concretas. A continuación se analizarán con cierto

detalle dos de las implementaciones de Matlab para utilizar en entornos paralelos (PMatlab y Start-P). En esta sección se mostrarán dos casos de entre las diferentes implementaciones existentes de Matlab para programación paralela. Además se verá más en profundidad el entorno que ha servido para programar las tres aplicaciones que componen en este trabajo.

PMatlab.

Lincoln Laboratory del MIT (MIT-LL) desarrolla PMatlab, que apoya el paralelismo de datos mediante una programación llamada *mapa*. Para desarrollar una aplicación PMatlab, el usuario escribe un programa de Matlab. El programa es entonces paralelizado agregando mapas. Los mapas son objetos que describen cómo debe ser una matriz distribuida.

PMatlab distribuye y comunica los datos sin que el usuario tenga conocimiento de ello. PMatlab está íntegramente escrito en lenguaje Matlab y funciona en cualquier arquitectura que tenga instalado Matlab. Esto se extiende desde los ordenadores personales hasta los clúster computacionales. Estos clúster pueden estar están configurados con procesadores en serie o multinúcleos homogéneos. PMatlab introduce un nuevo tipo de datos: la matriz de distribución o *dmat. Dmat* es el tipo de dato de almacenamiento fundamental en PMatlab, equivalente al dato doble utilizado en el entorno clásico de Matlab. PMatlab puede trabajar con objetos *dmat* de dos, tres o cuatro dimensiones. Los objetos *dmat* pueden ser explícitamente construidos en PMatlab a través de una función constructora.

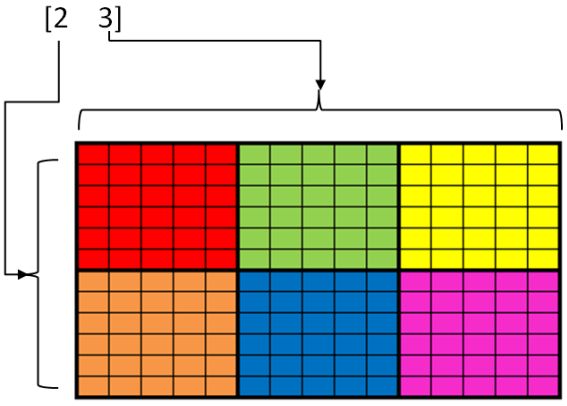


Figura 2.13. Matriz distribuida, *dmat*, en el entorno PMatlab.

En la actualidad, PMatlab posee cuatro funciones del constructor de Matlab: zeros, ones, rand, y spalloc. Cada una de estas funciones acepta el mismo conjunto

de parámetros así como sus correspondientes funciones de Matlab, añadiéndole un parámetro del mapa. El parámetro que se le añade, el mapa, describe cómo distribuir el objeto *dmat* entre varios procesadores. Cada mapa contiene 4 componentes.

• **Grid.-** es un vector que indica cómo se divide la matriz distribuida (*dmat*). Si el grid fuese [2,3] (figura 2.13), la matriz se dividirá, para la primera dimensión, en 2 procesos y para la segunda dimensión, en 3 procesos.

• **Distribution.-** especifica el orden de cómo se distribuyen los datos de cada dimensión a los procesadores. Existen 3 tipos de distribución:

- Block.- cada procesador contiene un solo bloque de datos.
- Cyclic.- los datos se intercalan entre los procesadores.
- Block-cyclic.-bloques de datos que se intercalan entre los procesadores.

• **Processor list.-** especifica cómo se podrían asignar los procesadores a las diferentes dimensiones (figura 2.14).

• **Overlap.-** es un vector que especifica la cantidad de datos que se pueden superponer en cada procesador.

```
A(:,:)      % Referencia a toda la matriz
A(i,j)      % Referencia a un elemento de la matriz
A(i:k,j:l)  % Referencia a una submatriz de la matriz
```

Figura 2.14. Ejemplo de subíndices de referencia con PMatlab.

Start-P.

Start-P es un módulo interactivo de supercomputación, inicialmente dirigido a Matlab, pero que se ha expandido a otros HLLs (anglicismo de lenguajes de alto nivel), por ejemplo, Python. Tanto PMatlab como Start-P comparten un enfoque similar en referencia a la paralelización de los datos. En primer lugar, el usuario escribe un programa en serie funcionalmente correcto. Start-P indica las dimensiones de la matriz de distribución mediante el etiquetado de los argumentos con *p. Un ejemplo de esta asignación sería: a = rand(100*p).

```
% Referencia a una submatriz de la matriz
a=rand(100,100,500);
for i=1:500
   b(:,:,1)=inv(a(:,:,1));
end
% implementación en paralelo Start-P
a=rand(100,100,500*p);
B=ppeval('inv'),a);
```

Figura 2.15. Ejemplo código start-P.

Start-P tiene un cliente-servidor basado en una arquitectura donde el cliente es una sesión de Matlab que se ejecuta en el ordenador del usuario y el servidor es un clúster que ejecuta el software de Start-P. Para una mayor comprensión de cómo funciona el método Start-P de Matlab, se va a ilustrar un ejemplo sencillo (figura 2.15). En este ejemplo se crea una matriz aleatoria de tres dimensiones y se calcula la inversa de esta matriz.

Para ello se ha añadido el parámetro *p en la última dimensión de la matriz para su creación. Con Start-P se utiliza la función "ppeval" para enviar a ejecutar de forma paralela el trabajo deseado, esta función se utiliza de forma análoga a la función "feval" del "Parallel Computing Toolbox" de Matlab. La figura 2.15 muestra la utilización de la función "ppeval".

2.12.2 Entorno paralelo Matlab.

A continuación se verá de forma breve el entorno paralelo que ofrece MATLAB. Las últimas versiones de MATLAB disponen de dos librerías para el cálculo paralelo. Estas son *Parallel Computing Toolbox* y *Matlab Distributed Computing Server* que permiten coordinar y ejecutar operaciones independientes de Matlab de forma simultánea en un clúster (figura 2.16). Estas dos librerías son las utilizadas para las tres aplicaciones implementadas en el desarrollo de la Tesis. La toolbox de computación distribuida ha ido evolucionando hasta la actualidad, de modo que permite simular trabajadores locales (*workers*) dentro de un ordenador con un procesador de varios núcleos. De esta forma cada núcleo trabaja sobre un programa (o un ciclo *for* por ejemplo) y devuelve un resultado al terminar.

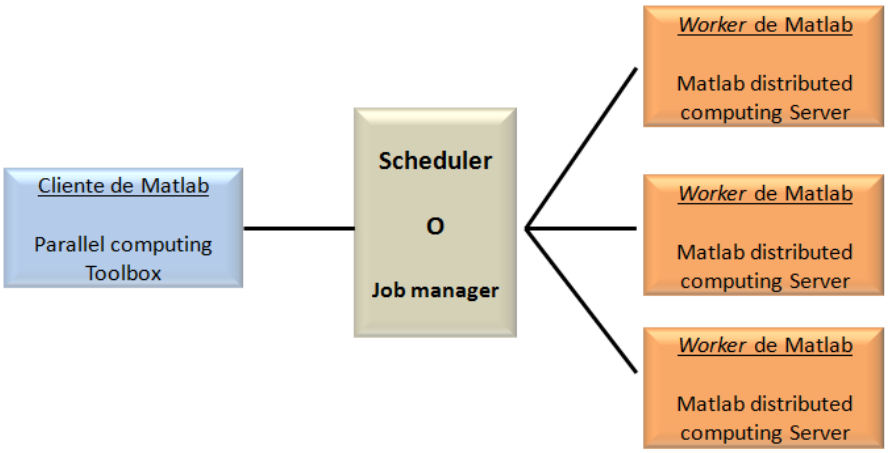


Figura 2.16. Ejemplo de planificador Matlab.

En el entorno MATLAB paralelo un *job* o trabajo es una operación que está dividida en diferentes segmentos llamados tareas. Se podría dividir el trabajo en tareas idénticas, pero las tareas no tienen por qué ser idénticas. La sesión de Matlab en la que el trabajo y sus tareas están definidos, se llama *periodo de sesión de cliente*. A menudo, "Parallel Computing Toolbox" utiliza la sesión de cliente para llevar a cabo la definición de puestos de trabajo y tareas. "Matlab Distributed Computing Server" realiza la ejecución de su trabajo mediante la evaluación de cada una de sus tareas, devolviendo el resultado a su sesión de cliente. El administrador de trabajo o "*job manager*" es la parte del software del servidor que coordina la ejecución de los trabajos y de la evaluación de sus tareas. El "*job manager*" distribuye las tareas de evaluación para cada servidor de sesiones (*workers*) de Matlab.

2.12.3 Programación paralela con MATLAB.

A continuación se estudian tres áreas relacionadas con el entorno paralelo que tienen especial interés. Se detalla el funcionamiento de instrucciones representativas y la forma de trabajar con grandes volúmenes de datos.

a) for-loops. Muchas de las aplicaciones que involucran múltiples líneas de código, pueden tener algunas repetidas. A menudo se pueden utilizar bucles (*for-loops*) para resolver estos casos. Como ya se ha comentado, la capacidad de ejecutar código en paralelo, en un equipo o en un clúster, puede mejorar significativamente el rendimiento de muchas aplicaciones. Estas aplicaciones se pueden clasificar en dos tipos:

Aplicaciones de barrido de parámetros: un barrido puede tardar mucho tiempo porque incluye muchas iteraciones. Cada iteración por sí sola no tardaría mucho tiempo en ejecutarse, pero para completar miles o millones de iteraciones en serie podrían tardar mucho tiempo. Normalmente, la única diferencia entre las iteraciones se define por los diferentes datos de entrada. En estos casos, la capacidad de ejecutar las iteraciones simultáneamente puede mejorar el rendimiento. Se puede optimizar el tratamiento de grandes o múltiples conjuntos de datos mediante la paralelización de las iteraciones. La única restricción importante que se contempla es que en los bucles en paralelo no se permite que las iteraciones sean dependientes unas de otras.

Por ejemplo, un bucle de 100 iteraciones podría ejecutarse en un clúster de 20 *workers*, por lo que cada uno de los *workers* solo tendría que ejecutar 5 iteraciones del bucle. La mejora en la velocidad de ejecución no será exactamente 20 veces más rápida debido a la velocidad de comunicación entre las distintas tareas y el tráfico de red, aunque la ganancia final será muy notable.

Aplicaciones de pruebas con diferentes segmentos de datos: son aplicaciones que requieren que se ejecuten diferentes tareas pero que no están relacionadas entre ellas. Se pueden ejecutar de forma simultánea en diferentes recursos del sistema. Normalmente no se utiliza un bucle *for* para estas ejecuciones, pero en este caso al ser obligatorio que cada iteración sea independiente del resto es posible que la utilización de *parfor* (bucle paralelo) sea una buena solución.

b) Off-loading work o descarga de trabajo. Cuando se trabaja de forma interactiva en una sesión de Matlab, se puede descargar dicho trabajo a un *worker* de Matlab. El comando para realizar esta acción es *asynchronous*, lo que significa que el actual periodo de la sesión de Matlab no estará bloqueado, y se podrá seguir con el periodo de sesiones interactivas propias, mientras que el *worker* está ocupado evaluando el código enviado. El *worker* de Matlab puede funcionar tanto en la misma máquina del clientecomo en una máquina remota pudiendo ser un clúster.

c) Tratamiento masivo de datos. Si se tiene una matriz que es demasiado grande para la memoria de un ordenador, no se puede manejar fácilmente en una sola sesión de Matlab. Matlab paralelo permite la distribución de la matriz entre varios *workers* de Matlab, de modo que cada *worker* sólo contiene una parte de la matriz. Sin embargo, se puede operar en todo el conjunto como una sola entidad. Cada *worker* sólo opera en su parte de la matriz, y los *workers* de forma automática se transfieren datos entre sí cuando sea necesario, como por ejemplo, en la multiplicación de

matrices. Un gran número de operaciones con matrices y funciones han sido mejoradas para trabajar directamente con las matrices distribuidas. En el capítulo cinco se verá la implementación de esta nueva estructura de datos.

2.12.3.1 Jobs.

En un entorno paralelo se entiende por *jobs* a todos aquellos trabajos que se ejecutan en modo desatendido. Son trabajos *batch* (anglicismo de trabajos desatendidos), es decir, el sistema somete a la CPU a uno o varios trabajos que ejecutan diferentes tareas. Por tratarse de un entorno novedoso una de las tareas de este trabajo fue la de analizar el comportamiento del clúster DIA con la planificación de trabajos. El objetivo era analizar el tiempo de respuesta del sistema bien con procesos batch o mediante *workers*. Después de diferentes pruebas se comprobó que la ejecución de trabajos mediante *workers* era más práctica para los objetivos marcados en este trabajo. La figura 2.17 muestra la consola de trabajos batch de la toolbox parallel de Matlab.

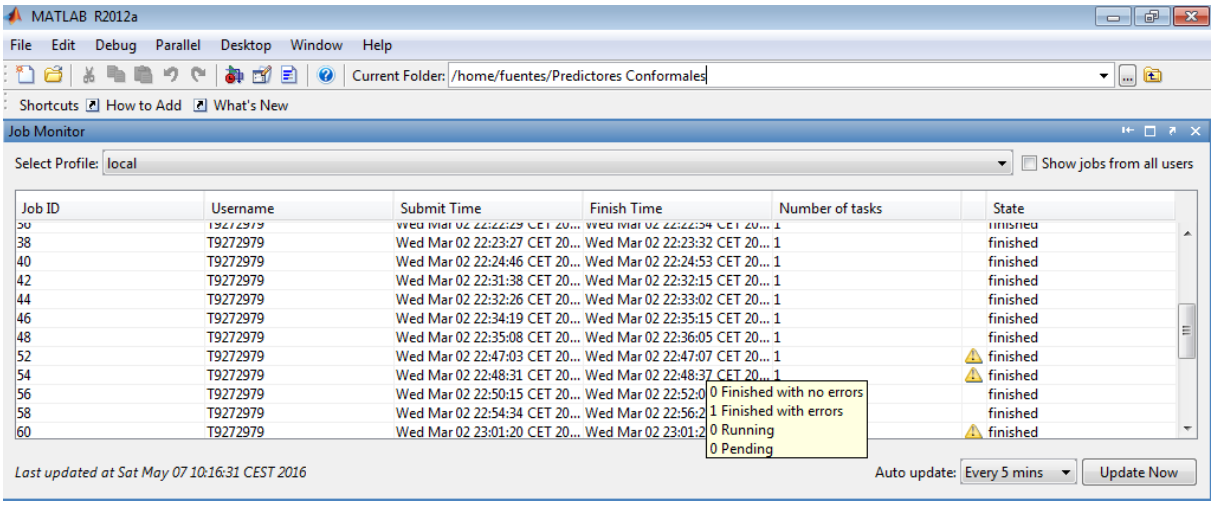


Figura 2.17. Consola Matlab del clúster DIA para control de jobs y tareas.

2.12.3.2 *Workers.*

Los *workers* son trabajos que se arrancan en el sistema. Este concepto va a ser muy recurrente en este trabajo. La idea es que estos *workers* trabajen de forma colaborativa y sincronizada. Hay una gran diferencia cuando se trabaja en modo local (un nodo) frente al trabajo distribuido, es decir cuando se trabaja en modo clúster con diferentes nodos. En este último entorno, los *workers* deben ser definidos en una consola donde es posible seleccionar el número de *workers* que se asignan a

cada nodo. Todos los *workers* definidos pueden ser utilizados conjuntamente o se pueden seleccionar únicamente aquellos que se quieran utilizar en un momento dado. A modo de ejemplo la figura 2.18 muestra un trabajo-cliente que utiliza tres

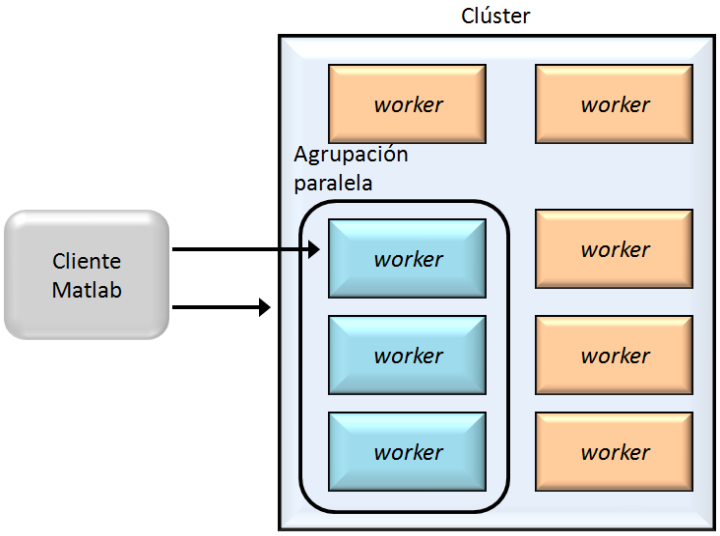


Figura 2.18. Escenario de clúster con *workers*.

de los ocho *workers* definidos en el sistema. Los cinco *workers* restantes serán utilizados por diferentes clientes. La forma en la que éstos *workers* son utilizados depende de cada proceso cliente, pero siempre se seguirá alguno de los paradigmas de programación paralela vistos en la sección 2.5 de este capítulo.

gcat	Concatenación global.
gop	Operación global a traves de todos los *workers*.
labBarrier	Ejecución hasta que la instrucción llegue a todos los *workers*.
labBroadcast	Enviar datos a todos los *workers* o recibir datos de todos *workers*.
labindex	Devuelve el número de *workers* que se ocupan en la comunidad.
LabProbe	Test para saber si los mensajes están listos para ser recibidos por otro *worker*.
pload	Carga de datos desde sesión paralela.
psave	Salvado de datos desde sesión paralela.
Gplus	Recibe una misma variable de todos los *workers* y a su vez es replicada a todos.
numlabs	Devuelve el número de *workers* total en el pool paralelo.
labSendReceive	Envío y recepción de información hacia y desde el *worker*.
labReceive	Recepción de datos de otro *worker*.

Tabla 2.1. Listado de instrucciones para paralelización mediante Matlab.

La forma en la que los *workers* se comunican entre sí hereda en parte el sistema de paso de mensajes *mpi* (message passing interface). La tabla 2.1 muestra

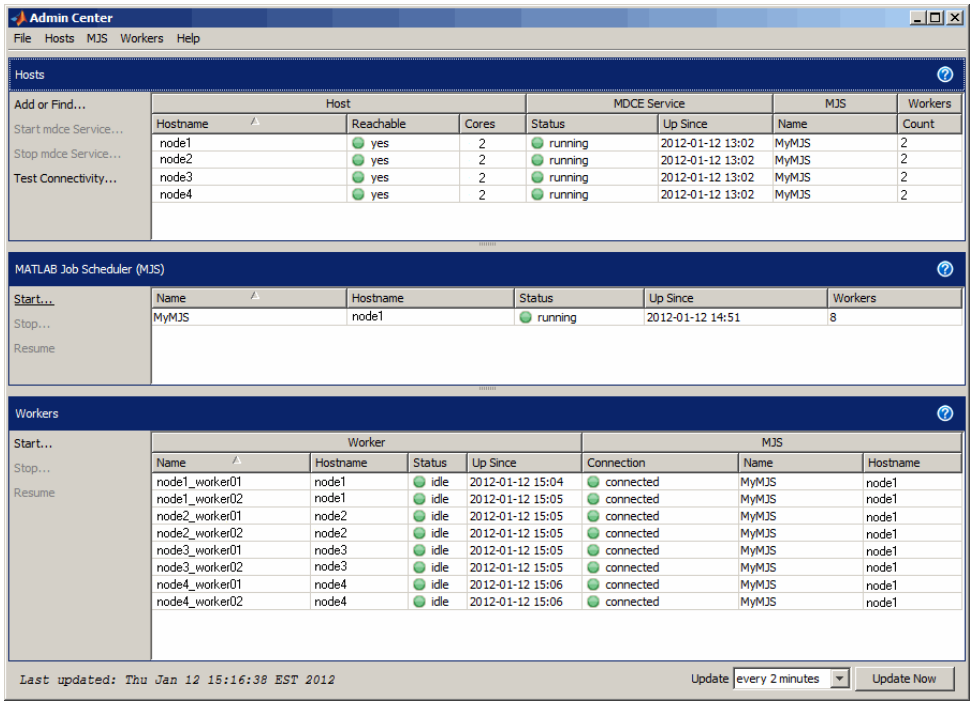


Figura 2.19. Consola de creación de *workers*.

las principales instrucciones especiales en entorno paralelo utilizadas en este trabajo. La figura 2.19 muestra la consola del sistema donde se gestiona todo el entorno paralelo-distribuido del clúster.

2.12.3.3 Tratamiento de conjuntos de datos de gran volumen.

Como ya se ha comentado en secciones previas uno de los objetivos de la computación paralela es indudablemente obtener resultados en el menor tiempo posible. En algunas situaciones, puede ocurrir que los datos a tratar sean tan voluminosos que se esté obligado a repartir por ejemplo una variable multidimensional, como puede ser una matriz, entre diferentes nodos al no ajustarse a la memoria del ordenador. Más adelante, en el capítulo 5 se verán algunos casos donde es necesario procesar una matriz de millones de filas y cientos de columnas en tiempo real y que sumado al procesamiento reiterado de los datos convierte a esta tarea en una problemática más a solucionar. Una variable de millones de filas por cientos de columnas no se ajusta a la memoria de un ordenador y es necesario repartirla entre diferentes nodos para que se pueda

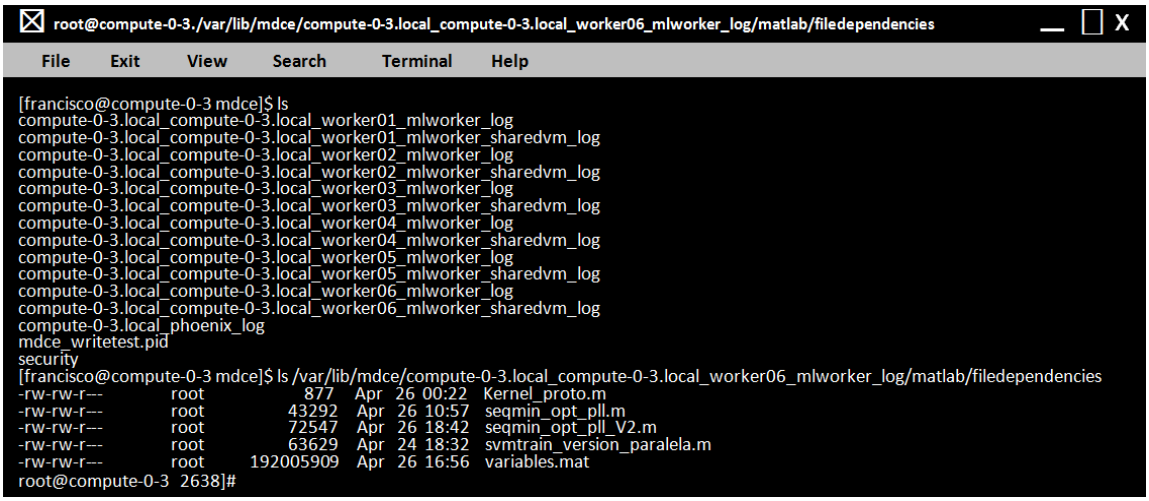


Figura 2.20. Posibles limitaciones en el almacenamiento de datos. Matlab crea una nueva librería o carpeta para cada *worker* donde almacena ficheros y/o datos. En la figura se puede ver la carpeta creada para el *worker* 6 del nodo 3 (compute-0-3). A las limitaciones de memoria se unen las limitaciones de disco pues una variable de 400 Mb por ejemplo, debe ser replicada para todos los *workers* de cada nodo y esto obliga a hacer una previsión de disco.

procesar. A modo de ejemplo la versión de 32 bits de Matlab puede hacer uso de 2 Gb. de memoria. Sin embargo la versión de 64 bits permite hacer uso de toda la memoria física del sistema operativo. Para conseguir los objetivos de cálculo, Matlab dispone de una serie de instrucciones que se adaptan a la computación paralela-distribuida que facilitan la tarea de cálculo. A continuación se detallan parte de estas órdenes.

Arranque de una sesión paralela. La función *matlabpool* arranca los *workers* y también define todos los ficheros (programas y datos) que van a ser usados por todos los *workers*. Es decir, se encarga de crear un entorno de trabajo para cada *worker*. La ventaja es que cada *worker* puede tener los datos de forma exclusiva y sin ningún tipo de bloqueo (véase figura 2.20). Por otro lado, puede ser necesario hacer una previsión de disco, debido a que una variable de 1Gb se convierten en 12Gb si se trata de un entorno paralelo con 12 *workers*. Además cuando la sesión paralela arranca, crea todos los entornos y esto consume memoria del front-end ó nodo maestro (véase figura 2.21) y un tiempo que no es de cálculo del problema sino de

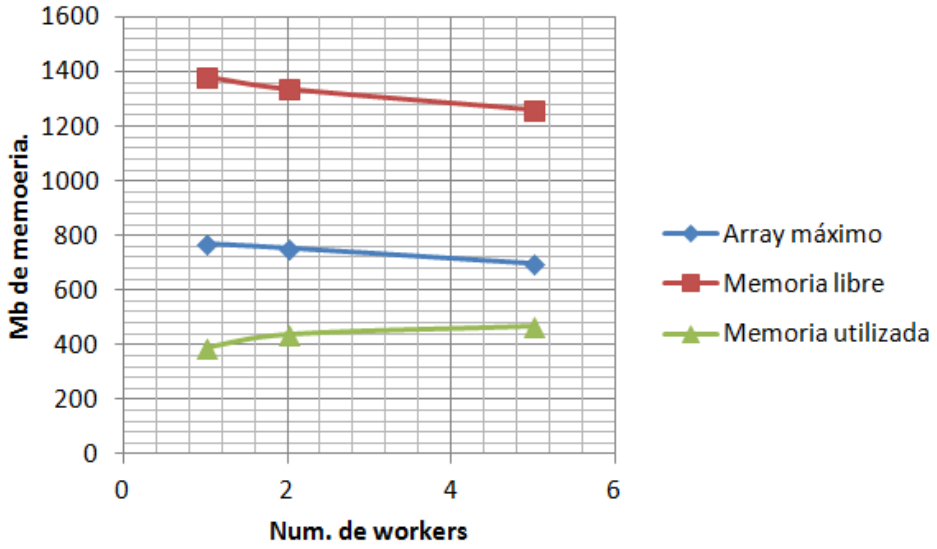


Figura 2.21. Evolución de la carga de la memoria en el arranque de los *workers*.

preparación del entorno. En este contexto, decir que ese consumo de tiempo tiene realmente sentido cuando se trata de un entorno estático[2]. En entornos dinámicos hay que valorar si realmente se va a obtener ganancia. Se habla de entornos dinamicos a la creación y destrucción de *workers* para diferentes necesidades de los modelos a tratar. Un entorno estático es aquel en el que el número de *workers* en el sistema permanece constante independientemente del tratamiento de diferentes modelos.

Variables de disco. La función *matfile* permite acceder a variables de MATLAB directamente desde archivos *.mat* almacenados en el disco a través de comandos de indexación de MATLAB y sin cargar las variables completas en la memoria. Esto hace posible llevar a cabo el procesamiento distribuido por bloques que de otra manera resultaría imposible almacenar en la memoria de un nodo.

Matrices y arrays distribuidos. Una de las funcionalidades que se ha utilizado con bastante frecuencia en el desarrollo de la Tesis han sido las matrices distribuidas. Matlab es capaz de distribuir una matriz entre los diferentes *workers* que componen la sesión paralela. Hay posibilidad de repartir una matriz por filas o por columnas. Cada *worker* trabajará con la parte de la matriz que le haya sido asignada. Por tanto,

[2] En este contexto un entorno estático utiliza una sesión paralela creada con anterioridad. De esta forma el tiempo necesario en crear este entorno no grava al nuevo proceso. En un entorno dinámico es necesario crear la sesión paralela siempre que se inicia un nuevo proceso. El tiempo necesario para disponer de todos los recursos penaliza el nuevo proceso.

las matrices distribuidas cobran especial importancia cuando se trabaja con gandes volúmenes de datos para poder repartir los datos a procesar en las diferentes memorias de cada nodo.

2.13 Clúster DIA.

Toda la aplicación práctica de los algoritmos implementados en el desarrollo de la Tesis ha sido ejecutada en el clúster DIA del departamento de informática y automática de la UNED. El clúster DIA está compuesto por cinco nodos, uno de los cuales actúa como front-end. Cada nodo dispone de 8 cores, con una velocidad de proceso entre 2,2 GHz. y 2,6 GHz. La configuración de la memoria de cada nodo varía de 5,7 Gb. a 7,9 Gb. La gestión del clúster se realiza mediante Rocks 6.1.
Rocks es una distribución para clústers de código abierto, basada en Linux que permite la construcción y uso de clústers y de otros sistemas basados en paradigmas de la paralelización y la ejecución concurrente de aplicaciones. Frente a otros sistemas como por ejemplo Clúster-Knoppix u OSCAR, Rocks cuenta con un apoyo de masas mucho mayor y se actualiza constantemente, usando software mucho más avanzado y optimizado. Rocks clusters está basado en **CentOS** (**C**ommunity **ENT**erprise **O**perating **S**ystem) que está considerado un clon a nivel binario de la distribución Linux **R**ed **H**at **E**nterprise **L**inux **RHEL**, compilado por voluntarios a partir del código fuente liberado por Red Hat, por lo tanto todos los comandos serán típicos de la distribución. Rocks Clusters tiene una serie de *Rolls* (paquetes de software) que permiten la gestión, monitorización y uso efectivo de la estructura. Entre ellos, aparte de permitir y garantizar el buen funcionamiento del clúster, existen varios que se usan específicamente en medicina u otros campos científicos. El software de Rocks siempre es ampliable y por supuesto compatible con cualquier software para Linux que fuera compatible con CentOS. Es una de las distribuciones actuales más moderna y preparada para gestionar un clúster.

La monitorización en tiempo real de recursos como pueden ser la CPU, memoria y disco así como parámetros estáticos (nº de procesadores, velocidad de reloj,...) y parámetros dinámicos (carga del procesador) esta confiada al roll Ganglia (figura 2.22). Ganglia es una solución open-source surgida como proyecto en la universidad de California. Dispone de procesos en cada nodo que le reporta toda la

Figura 2.22. Consola de monitorización Ganglia del clúster DIA. Se puede apreciar el estado de trabajo de los cuatro nodos en rojo-naranja y en azul el nodo front-end.

actividad del cluster. Ganglia se basa en la tecnología XML para la presentación de datos, XDR para el transporte de datos y RRDtool para el almacenamiento y visualización de los datos. Actualmente se utiliza en un gran número de clústers y se suele utilizar para conectar diferentes campus universitarios. En otra línea en las próximas secciones se muestran los resultados de la ejecución de tres algoritmos. El objetivo es evaluar los recursos disponibles, tanto computacionales como de memoria, en el sistema: 1) el cálculo del número PI mediante el algoritmo Montecarlo, 2) la resolución de un sistema lineal para distintas dimensiones y 3) el cálculo del conjunto de Mandelbrot.

Algoritmo 2.1: Cálculo del número PI mediante el algoritmo paralelo Monte Carlo.

```
function T= MonteCarloPI_2_Paralelo(perfil, numW);
 T=0;
 matlabpool('open', perfil, numW);
 R = 1;
 darts = 1e8;
 contador = 0;
 tic;
parfor i = 1:darts
        x = R*rand(1);                  /* Cálculo de las coordenadas X e Y */
        y = R*rand(1);                  /* utilizando distribución uniforme */
     if x^2 + y^2 <= R^2
          contador = contador + 1;  /* Incremento de los aciertos dentro del circulo. */
     end
end
 Cal_PI = 4*contador/darts;             /* Cálculo de PI.*/
 T = toc;
 fprintf('El cálculo del valor de PI es.....: %8.7f.\n',Cal_PI);
 fprintf('El método paralelo de Monte Carlo ha utilizado : %8.2f segundos.\n', T);
 matlabpool close;
```

Algoritmo 2.1. Cálculo del número PI mediante el algoritmo paralelo Monte Carlo.

2.14 Cálculo del número PI mediante el algoritmo Montecarlo.

El método de Montecarlo (MC) es un procedimiento matemático que permite simular un sistema con la ayuda de ordenadores. En general se aplica a problemas cuyo comportamiento global se puede modelar mediante una distribución de probabilidad [Peña 2001]. Es un método numérico que permite resolver problemas físicos, matemáticos y estadísticos mediante la simulación de variables aleatorias. Además se utiliza para aproximar expresiónes matemáticas complejas y costosas de evaluar con exactitud.

El caso al que se sometió al clúster DIA es el cálculo de π mediante la simulación de Montecarlo. Como ya se ha indicado el método de Montecarlo consiste en realizar un experimento aleatorio un determinado número de veces. Es conocido que la frecuencia con que ocurre un suceso se acerca a su probabilidad, a

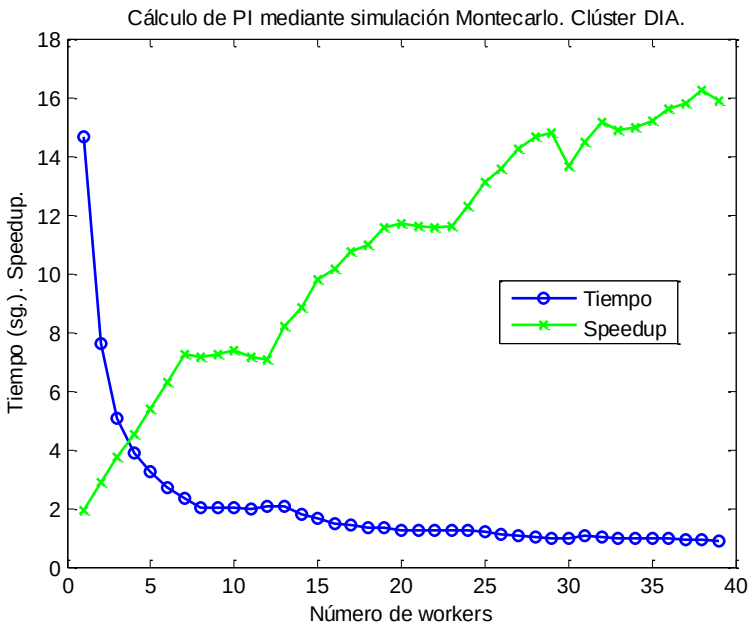


Figura 2.23. Tiempo de respuesta y Speedup para la aproximación de π mediante simulación Monte Carlo.

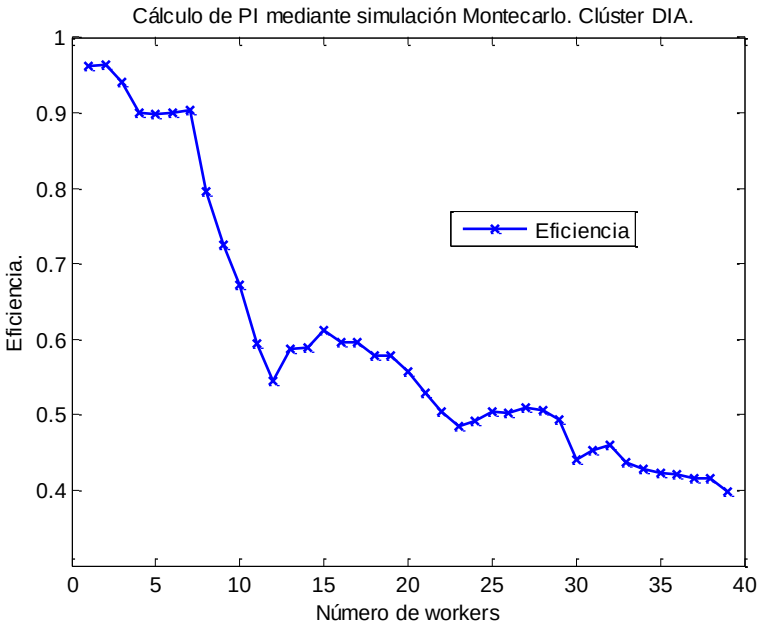


Figura 2.24. Eficiencia del método Monte Carlo.

medida que aumenta el número de ensayos se incrementará la aproximación al valor buscado. Puede utilizarse para estimar probabilidades que serían muy difíciles de calcular de forma teórica, o para corroborar que ocurrirá lo que se

espera de un determinado experimento. Para calcular el número PI, se pueden seguir los siguientes pasos:

1) Inscribir un círculo en un cuadrado de lado 2. El radio del círculo será 1.
2) Elegir al azar un punto del cuadrado y observar si este punto pertenece o no al círculo.
3) La probabilidad de que el punto esté dentro del circulo es la razón entre las áreas, π/4.
4) Se multiplica por 4 la frecuencia de este suceso y se obtiene una aproximación de π.
5) La aproximación será mejor cuanto mayor sea el número.

El proceso de cálculo se ejecutó mediante el algoritmo 2.1. El proceso comienza con el establecimiento de la sesión paralela configurada con un número pequeño de *workers* para más tarde ejecutar la aproximación al valor PI mediante un bucle *parfor*. Todo el proceso anterior se repitió hasta alcanzar un máximo de 39 *workers.* La gráfica de la figura 2.23 muestra el tiempo y el speedup del proceso según se incrementa número de *workers.* Se puede comprobar como el tiempo de respuesta disminuye hasta llegar a los 30 *workers.* A medida que sigue creciendo el número de *workers* el tiempo de respuesta no mejora de forma importante. En la gráfica de la eficiencia (figura 2.24) se aprecia de una mejor forma este cambio debido a que a partir de 30 *workers* la eficiencia decrece de forma notable.

2.15 Resolución de un sistema lineal para distintas dimensiones.

A continuación se mostrarán los resultados correspondientes a una prueba de rendimiento de memoria. Como ya se ha indicado, la memoria de cada nodo es compartida por todos los *workers* que sean definidos en dicho sistema. La prueba que se verá a continuación resuelve un sistema lineal A*x=b. Con frecuencia es posible encontrarse en el lado izquierdo de la igualdad una matriz A dividida por un vector b para calcular x. Este ejemplo muestra como punto de referencia la solución de un sistema lineal en un clúster.

Uno de los aspectos más difíciles de la evaluación es evitar caer en la trampa de buscar un solo número que represente el rendimiento general del sistema. Se pueden observar las curvas de rendimiento que podrían ayudar a identificar los

cuellos de botella en el clúster, y tal vez incluso ayudar a ver cómo comparar su código y poder extraer conclusiones significativas de los resultados.

Es muy importante elegir el tamaño de la matriz adecuada para el clúster. El objetivo estará marcado en asignar paulatinamente un tamaño mayor de memoria a cada *worker* y se verá cómo responde el sistema. Se abordará una paralelización distribuida. El objetivo es calcular el tiempo invertido en la división sin contabilizar el tiempo dedicado en la creación de la matriz. Es decir, se separa el tiempo dedicado a resolver el sistema lineal del resto de tareas. Los datos de entrada se generan mediante la instrucción *codistributor 2-D* debido a que es el esquema de distribución más eficaz. A continuación se mide el tiempo que invierten todos los *workers* en completar y resolver el sistema lineal A*x=b.

El rendimiento de la solución depende en gran medida del tamaño de la matriz A. Se debe elegir un tamaño adecuado para que el sistema no genere mucho *swapping* (anglicismo de paginación de memoria) y el rendimiento no caiga bruscamente. Se puede hacer a priori una categoría de matrices que sería la siguiente:

1. Matrices "demasiado pequeñas" de tamaño 1000 x 1000.
2. Matrices "grandes" cuando ocupen menos del 50% de la memoria del sistema disponible para cada *worker*.
3. Matrices "demasiado grandes" cuando ocupen más del 50% de la memoria del sistema disponible para cada *worker*.

El experimento a realizar consiste en generar diferentes tamaños crecientes de matrices para ejecutar operaciones de punto flotante con el fin de evaluar el rendimiento. En función de las gráficas que se obtengan se podrá responder a preguntas del estilo; ¿Cómo afecta el tamaño de la matriz en el rendimiento?, si la matriz supera el 50% de la memoria ¿en qué grado afecta al rendimiento?, ¿Qué tamaño de matriz es el más óptimo para el número de *workers* que se han definido?, ¿es la memoria la que limita el máximo rendimiento?. El procedimiento será el siguiente: 1) se crea una matriz *A* y un vector *b*. 2) se resuelve *A*/*b* varias veces. Se utilizan operaciones en punto flotante HPC ScaLAPACK. De tal forma que, para una matriz de *n x n* se tiene $(2/3)*n^3 + (3/2)*n^2$. La figura 2.25 muestra el resultado.

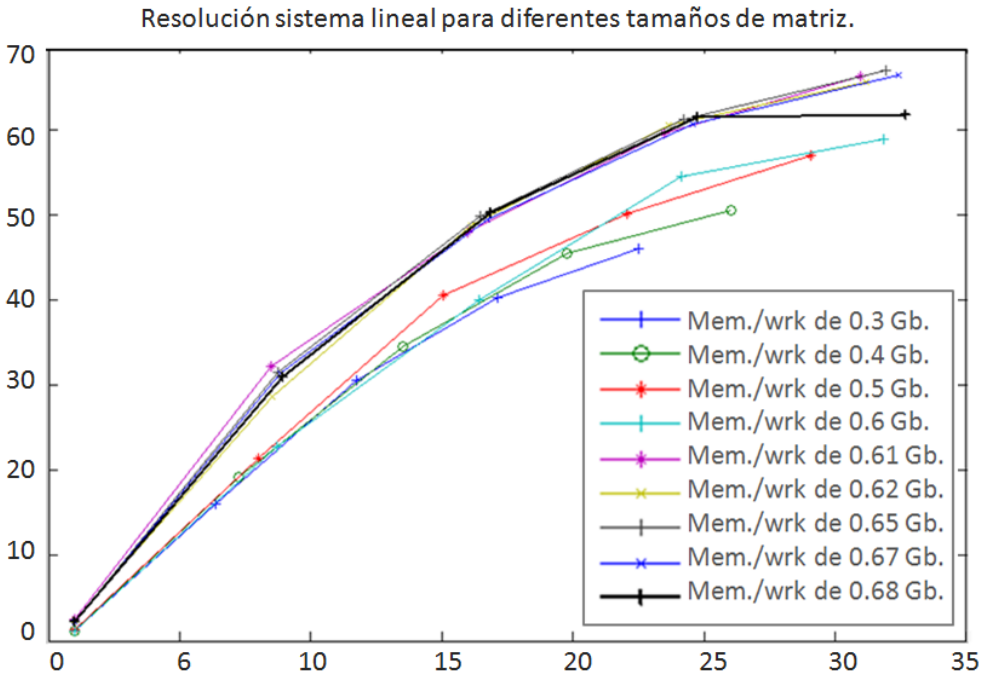


Figura 2.25. Análisis de la respuesta del clúster en paralelo distribuido para la resolución del sistema lineal con diferentes tamaños de matriz *A* y vector *b*. Para asignaciones de memoria de 0.68 Gb. (gráfica en color negro) se comprueba que el sistema alcanza la cota superior pues a más número de *workers* el sistema no progresa en mejora del tiempo de respuesta.

2.16 Cálculo del conjunto de Mandelbrot.

Benoit Mandelbrot fue un matemático francés-polaco estadounidense que pasó la mayor parte de su carrera en el Centro de Investigación Watson de IBM en Yorktown Heights, NY. Acuñó el término fractal y publicó en 1982 un libro referente, Geometría fractal de la Naturaleza [Mandelbrot 1982]. Fue la época en que las pantallas graficas de las computadores se estaban generalizando.

Desde entonces, el conjunto de Mandelbrot se ha utilizado en temas de investigación de matemáticas y también ha sido la base para un incontable número de proyectos gráficos, demostraciones de hardware y páginas web. Este caso servirá para estudiar la carga de trabajo en el clúster DIA. Para cada punto *c* en el plano complejo se calcula la secuencia

$$z_0 = 0; \qquad z_{i+1} = z_i^2 + c \tag{2.23}$$

El **Conjunto de Mandelbrot** es el lugar geométrico de los puntos donde la secuencia (2.23) no diverge. Por motivos prácticos, se selecciona previamente la

porción del plano complejo a visualizar. Las características más interesantes están en el rectángulo [(-2-i), (0.5+i)], o el disco cerrado de radio 2.

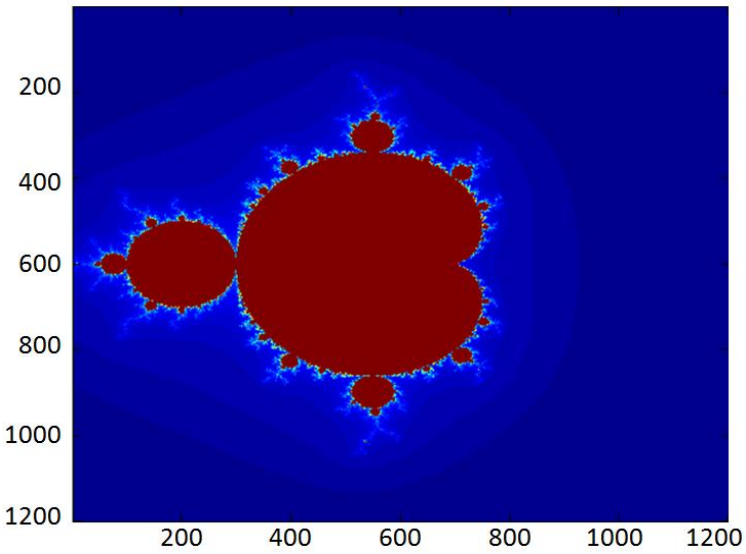


Figura 2.26. Conjunto de Mandelbrot: aproximación gráfica dentro de los límites Re(*c*) Є [-1.5, 0.4] e Im(c) Є [-0.9, 1.0].

La región bajo estudio está definida por Re(*c*) Є [-1.5, 0.4] e Im(*c*) Є [-0.9, 1.0] véase la figura 2.26. El cálculo de la secuencia se restringe a las coordenadas correspondientes a los pixeles de la imagen resultado. Se fija un máximo de iteraciones permitidas por pixel. La condición de divergencia se da por satisfecha cuando el módulo $|z|$ alcance 2 (es decir, fuera del disco).

El objetivo de este experimento es someter al sistema a una prueba de escalabilidad y conseguir una medida límite referente al número de *workers* necesarios para este problema en el clúster DIA. En la figura 2.27 se muestra la gráfica del tiempo de proceso y el speedup en función del número de *workers* y en la figura 2.28 la eficiencia. Se puede observar que, por encima de 26 *workers,* la eficiencia decae.

Esta prueba determina la potencia de cálculo del sistema. Un dato relevante es que cada *worker* no almacena datos en memoria. Si los procesos necesitasen más memoria con seguridad los tiempos serían más altos debido al efecto del *swapping*. Cada *worker* procesa una parte del problema. Al final de la sección se muestra el algoritmo 2.2 utilizado en este experimento y que define la solución paralelizada.

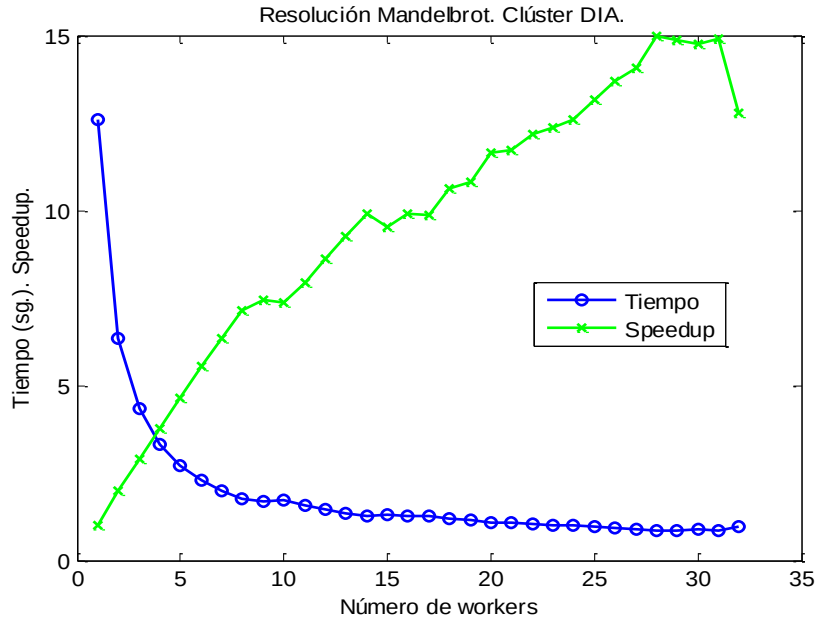


Figura 2.27. Tiempo de proceso y speedup para la versión paralela del conjunto de Mandelbrot.

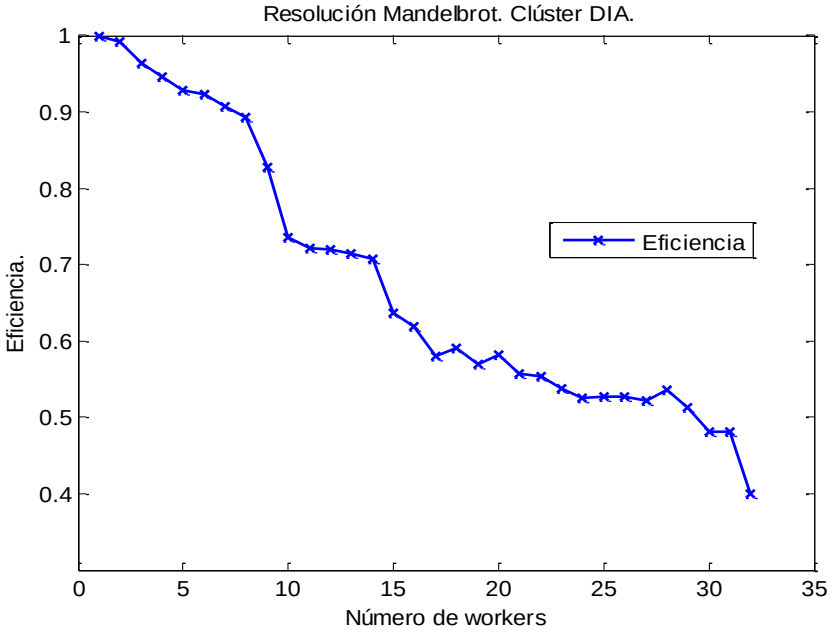


Figura 2.28. Eficiencia para la versión paralela del conjunto de Mandelbrot.

Algoritmo 2.2: Programa paralelizado del conjunto de Mandelbrot.

```
dx=xmax-xmin;
dy=ymax-ymin;
ddy=dy/numTareas;
ddx=dx/numTareas
configName = defaultParallelConfig();
sched = findResource('scheduler', 'Configuration', configName);
trabajo = createJob(sched);
for i = 1:numTareas
      xmin+ddx*(i-1)
      xmin+ddx*i
      createTask(trabajo,@mandel1,1,{steps/numTareas,maxiter,xmin,xmax,ymin+
      ddy*(i-1),ymin+ddy*(i),numTareas});
 end
submit(trabajo);
waitForState(trabajo, 'finished');
y = getAllOutputArguments(trabajo);
Z=cell2mat(y);
destroy(trabajo);

function Z=mandel1(maxiter,steps,xmin,xmax,ymin,ymax,tasks)
  dy=(ymax-ymin)/((steps-1)/tasks);
  dx=(xmax-xmin)/((steps-1));
  for m=1:steps/tasks
    ci=single(ymin+dy*m);
    for n=1:steps
     cr=single(xmin+dx*n);
     zr=single(cr);
     zi=single(ci);
     rmax=maxiter;
     for r=0:maxiter
       zrn=single(zr*zr-zi*zi+cr);
       zin=single(2*zi*zr+ci);
       zi=single(zin);
       zr=single(zrn);
      if single(zr*zr+zi*zi)>10000,
        rmax=r;
        break
      end
     end
       Z(m,n)=rmax;
    end
  end
end
```

Algoritmo 2.2. Paralelización del conjunto de Mandelbrot.

Capítulo III. Introducción a las aplicaciones implementadas.

3.1 Introducción.

En este capítulo se realiza una presentación de las aplicaciones desarrolladas. La primera parte aborda el reconocimiento de patrones que aparecen en señales de evolución temporal y desarrolla una técnica para localizarlos y estandarizar las búsquedas. La segunda parte introduce la clasificación de señales y cómo puede ayudar la computación paralela en la rapidez de la clasificción de señales, además sirve de preámbulo para la tercera y última parte en la que se utiliza la predicción conformal aplicando un tipo específico de predictor que es capaz de dar niveles de probabilidad en sus predicciones.

3.2 Análisis de patrones.

El almacenamiento y gestión de grandes colecciones de imágenes es un campo de investigación importante en muchas disciplinas de la informática. En sectores como la fusión nuclear [Vega 2007], informática médica o la bioinformática, por ejemplo, se generan diariamente una gran cantidad de imágenes. Estas imágenes contienen información de gran valor para profesionales e investigadores. Por esta razón las imágenes son procesadas para extraer la información que contienen y son almacenadas para poder ser consultadas en cualquier momento.

Comúnmente, las instituciones y empresas que poseen grandes colecciones de imágenes, utilizan sistemas de almacenamiento, indexación y recuperación de imágenes. La adquisición e instalación de estos sistemas de administración de imágenes es, por lo general, muy costosa tanto en el aspecto económico como en el de recursos computacionales.

Esto es debido a que estos sistemas requieren grandes servidores o clústers y profesionales expertos para su instalación y adaptación. Todo ello origina que el mantenimiento y las actualizaciones de estos sistemas sea también costoso.

En muchas ocasiones los sistemas de gestión de imágenes necesitan ser ampliados o complementados con funcionalidades que permitan llevar a cabo las tareas que se requieren sobre las colecciones de imágenes. Incluso dentro del

mismo campo de investigación, la misma colección de imágenes necesita distintos métodos añadidos al sistema de gestión de imágenes para poder ser utilizada de forma eficiente. Una imagen contiene gran cantidad de información dentro de los pixeles y regiones de la imagen [Dormido-Canto 2012]. Dichas regiones contienen descriptores de color, de forma y de textura. Otro dato importante son las relaciones existentes entre dichas regiones. Incluso en algunos dominios, algunas imágenes contienen información semántica.

3.2.1 Planteamiento del problema.

La fusión nuclear por confinamiento magnético se perfila, junto con las energías renovables, como la fuente de energía del futuro. A las temperaturas requeridas para que se produzcan reacciones de fusión, los electrones se encuentran completamente desligados de los núcleos, encontrándose la materia en estado de plasma. En el estudio del plasma por confinamiento magnético es crucial conocer con detalle las condiciones físicas del plasma confinado. En este sentido uno de los problemas de la fusión por confinamiento magnético es mantener el plasma confinado dentro del dispositivo de fusión sin que toque las paredes, lo que detendría las reacciones nucleares y dañaría el dispositivo. Las descargas de plasma nuclear generan fenómenos que deben ser estudiados con detalle. Esto hace que sea necesario disponer de herramientas que ayuden a localizar dichos fenómenos y sobre todo que detecten dónde se encuentran.

Como se ha mencionado anteriormente, es necesario disponer de herramientas que ayuden a estudiar el comportamiento del plasma. Parte de este trabajo nace de la necesidad de estudiar un fenómeno que se produce en las descargas de plasma. Se trata de localizar el instante temporal en el que aparecen los modos armónicos, así como la duración de los mismos. Éstos armónicos suelen aparecer en una banda frecuencial concreta, pero el objetivo es detectarlos en todo el periodo temporal de la descarga y en toda su banda frecuencial.

Para conseguir este objetivo se deben analizar ficheros que contienen las medidas de estas descargas. Las medidas que aquí se presentan se han realizado en JET. Las imágenes donde buscar patrones deben ser procesadas previamente y como resultado se obtiene el espectrograma de la señal, en la figura 3.1 puede verse un ejemplo.

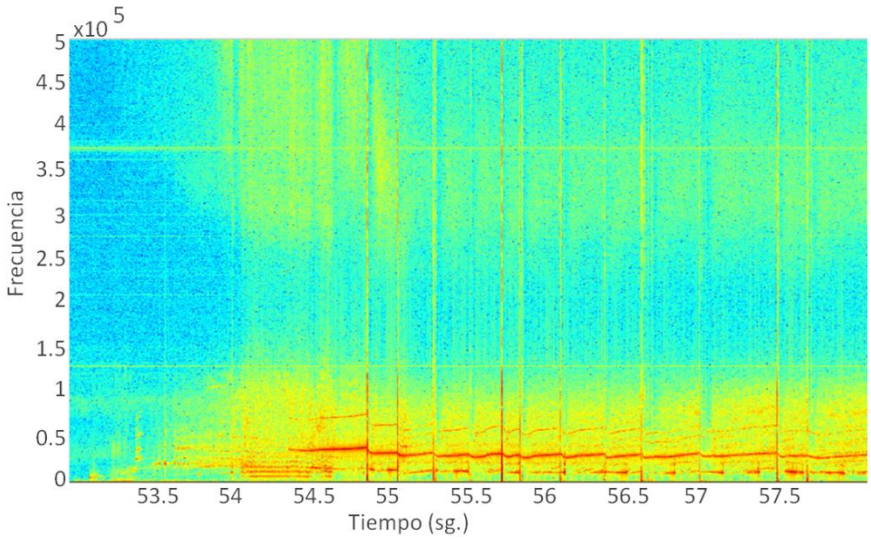


Figura 3.1. Espectrograma de la descarga 79455, señal PP8 canal 6 en pared de carbono.

3.2.2 Reconocimiento de patrones.

El reconocimiento de patrones se puede definir según Ribadas como el proceso de encontrar una subestructura que está incluida dentro de otra mayor [Ribadas 2002], mediante la comparación de ésta con una representación de aquella en forma de patrón. El reconocimiento de patrones ofrece un modo natural para interrogar de forma descriptiva un conjunto de estructuras complejas que podrán tener diferentes formas. Más aún, en la mayor parte de los casos estas formas no podrán ser comparadas con un modelo predeterminado porque no existen modelos con los que comparar. Las técnicas que se detallan en las siguientes secciones mostrarán cómo se puede definir una metodología apropiada para conseguir el objetivo de la localización de áreas de interés.

En general la mayor parte de las herramientas de búsqueda disponibles implementan algunos de los métodos clásicos de análisis, siendo necesaria la intervención del usuario mediante programación para abordar ciertas tareas que involucran aspectos de búsqueda de formas de texturas, por lo que no siempre es factible el uso de estas herramientas. Cuando los métodos clásicos no ofrecen resultados esperados es cuando se debe acudir a herramientas ad-hoc [Dormido-Canto 2008]. Las imágenes donde buscar patrones deben ser procesadas previamente y como resultado se obtiene el espectrograma de la señal. Esta imagen debe ser calculada cuidadosamente pues de ello depende la nitidez y posterior

análisis del contenido del espectrograma. Al proceso de análisis de la imagen se le llama *segmentación*.

3.2.2.1 Segmentación de imágenes.

Al abordar el problema de la segmentación, es importante tener presente que el objetivo que persigue es extraer las regiones que hay en la imagen, no los objetos representados en ella [Cyr 2004, Lindeberg 1999, Sanfeliu 2002]. Debido a la relación entre ambos términos, es habitual que se confundan. Sin embargo son nociones diferentes ya que las regiones sirven como punto de partida para reconstruir objetos, lo cual es una etapa posterior del procesamiento que requiere la aplicación del conocimiento. La detección de objetos es un ejemplo del gran número de aplicaciones que parten de los resultados de una segmentación para llevar a cabo un procesamiento de más alto nivel, como la recuperación, el análisis, la transmisión o la compresión de imágenes.

Esta es una de las razones por las que en los últimos treinta años se han desarrollado un gran número de técnicas de segmentación, con diferentes objetivos y campos de aplicación. Las primeras técnicas desarrolladas daban como resultado segmentaciones precisas *crisp* [Hovmöller 1992] de imágenes en niveles de gris,

Basadas en pixeles	Basadas en fronteras	Basadas en regiones
Umbralización	Detección de bordes local	Fusión y/o división
Histogramas	Detección de bordes global	Crecimiento de regiones
Clustering (K-medias, ISODATA)	Modelos deformables	Morfologicas
Redes neuronales (Hopfield, SOM)		

Tabla 3.1. Clasificación de las técnicas de segmentación.

habitualmente basadas en un proceso de umbralización [Sahoo 1988], en las que cada píxel de la imagen pertenecía a una única región. Sin embargo, como se ha indicado en la introducción de esta memoria, existen diversos efectos en imágenes reales que son difíciles de tratar y representar para las técnicas de segmentación precisas. En palabras de Priese la *segmentación de imágenes se utiliza para obtener una representación simbólica, abstracta, de la imagen, la cual está íntimamente relacionada con la calidad de los posteriores análisis y procesamientos de la imagen* [Priese 2005], por lo que dicha

segmentación debe representar lo mejor posible la información que hay en la imagen.

En la literatura se pueden encontrar diversas formas de clasificar las técnicas de segmentación, aunque la mayoría de ellas tiene una base común, que se resume en el esquema de la tabla 3.1. La distinción entre las tres categorías básicas viene dada por la diferente forma que tienen de definir el proceso de segmentación y por el tipo de resultados que proporcionan:

Las técnicas basadas en píxeles, según Moghaddamzadeh [Moghaddamzadeh 1997] consisten en *agrupar los píxeles de una imagen en conjuntos homogéneos respecto a una o varias características*. Según Lucchesse [Lucchesse 2001], *la segmentación de imágenes es una operación de bajo nivel relativa a la partición de una imagen [...] mediante la localización de bordes*. Para Davis [Davis 1975], las técnicas basadas en fronteras centran su atención *en las características y más aún en los cambios bruscos debidos a la orientación de los píxeles de los bordes de la imagen.* En cuanto a las técnicas basadas en regiones, para Cheng [Chen 2003] *la segmentación de imágenes es el proceso de dividir una imagen en distintas regiones, pertenecientes a diferentes objetos, tales que cada región es homogénea pero si se unen dos regiones adyacentes cualesquiera dejan de ser homogéneas. La etapa de segmentación no se preocupa de la identidad de los objetos*. Básicamente, la noción de región representa la agrupación de un conjunto de píxeles que se parecen entre sí y que están relacionados o conectados formando una componente conexa. Esta idea se encuentra formalizada en diversas referencias [Adams 1994, Pajares 2003], cuyas definiciones se pueden resumir del siguiente modo:

Definición 3.1. *Una región R_s de una imagen I, es un conjunto de píxeles semejantes y conectados.*

Definición 3.2. *La segmentación de una imagen se define como el proceso de particionar la imagen en subconjuntos de píxeles semejantes y conectados.*

De los tres tipos de técnicas (tabla 3.1), únicamente las basadas en regiones son acordes a estas definiciónes y dan regiones como resultado.

3.2.3 Paralelización en la identificación y localización de modos.

En los sistemas de almacenamiento de imágenes es una práctica habitual distribuir las imágenes para mejorar el rendimiento. Como novedad, el planteamiento propuesto también tiene en cuenta dividir las imágenes en varias subimágenes para que puedan ser procesadas de forma independiente. Este planteamiento hace pensar en un procesamiento de regiones en lugar de imágenes.

Por decirlo de alguna manera, el paralelismo hace bajar un nivel más en el procesamiento digital de grandes volúmenes de imágenes. Claramente el planteamiento paralelo anterior cobra más importancia cuanto más grande es la imagen. En el capítulo 4 de la Tesis se hará especial énfasis en esta tarea.

$$X(k) = \sum_{n=0}^{N/2-1} x(n)e^{-i\frac{2\pi nk}{N}} + \sum_{n=N/2}^{N-1} x(n)e^{-i\frac{2\pi nk}{N}} \qquad (3.1)$$

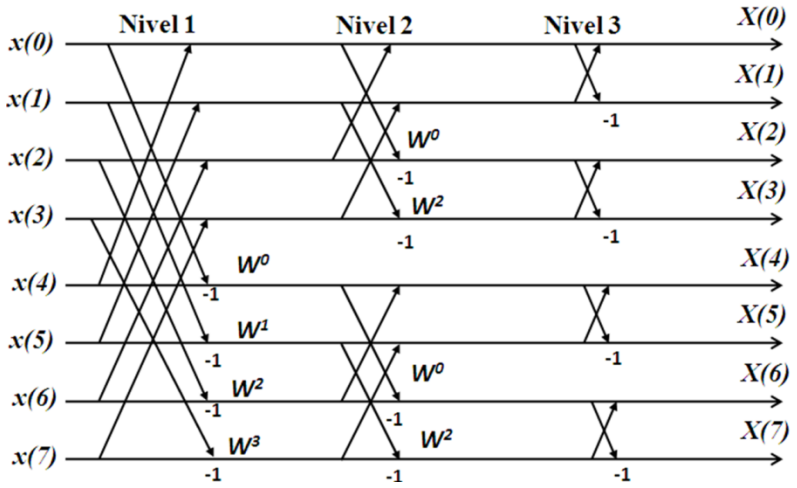


Figura 3.2. Algoritmo Radix para el cálculo de la FFT.

Otro punto a destacar en el proceso de búsqueda es la generación del espectrograma en base a la transformada rápida de Fourrier. También es posible paralelizar la transformada rápida de Fourrier (FFT). En [Inda 1991] se hace un estudio y se propone un sencillo algoritmo para conseguir mejores tiempos. No obstante un algoritmo extendido es Radix. La expresión 3.1 es la utilizada para el cálculo de la FFT. Los dos sumatorios de esta expresión se pueden procesar mediante secuencias pares e impares donde se reemplaza $k=2k$ para las secuencias pares y $k=2k+1$ para las secuencias impares. En la figura 3.2 se puede ver un esquema de operaciones en función de las secuencias para un vector de N=8 y donde $W=e^{-i2\pi/N}$. En cada nivel la secuencia se divide en $N/2^{nivel}$. La paralelización de este algoritmo con p *workers* consiste en mantener éstos activos mientras la longitud de la secuencia sea mayor o igual al número de *workers*. Tan solo en los

últimos niveles del algoritmo es donde se ve reducido el número de *workers* que realizan cálculos.

La mejora de tiempos está en función del lenguaje a utilizar. Es decir, no todos los lenguajes tienen las mismas capacidades y es cierto que día a día las funciones de los lenguajes incorporan nuevas y avanzadas ventajas. Si bien se hicieron pruebas en la paralelización del algoritmo Radix al final no se implementó en el modelo definitivo pues los tiempos de respuesta de Matlab para el cálculo de la FFT eran buenos proporcionalmente hablando.

3.2.3.1 Datos de entrada.

La fusión nuclear por confinamiento magnético ha experimentado un gran progreso. Las características del plasma han ido mejorando pero a medida que aumenta el grado térmico del plasma, las superficies del Tokamak JET que están en contacto con él lo hacen con un orden mucho más elevado y pronto llegan al grado térmico en el que se pueden desintegrar. Uno de los elementos que obedece a criterios termomecánicos óptimos así como de compatibilidad con el plasma es el carbono. Los datos a analizar son resultado de dos tipos de descargas de plasma del Tokamak JET. Las realizadas con pared de carbono y las realizadas con pared metálica. El tipo de pared no presenta cambios en los algoritmos desarrollados porque lo que se analiza son imágenes. Sin embargo y por organización de la información los resultados se presentarán teniendo en cuenta estos dos tipos de entornos.

Los ficheros de donde se deben extraer los datos presentan diferentes tamaños y en total se han procesado más de 6 Tb que suponen un total de 95937 imágenes. La variedad de tamaños junto con el cálculo previo de los espectrogramas hacen que la aplicación del paralelismo cobre más importancia. La posibilidad de configurar el algoritmo para la búsqueda de nuevos patrones y volver a procesar de nuevo todos los ficheros da más protagonismo a la computación paralela.

Se puede plantear un escenario donde el cálculo de los espectrogramas se efectúe una sola vez y se almacenen las imágenes a la espera de futuras búsquedas. En este punto se vuelve al planteamiento inicial en el cual se planteaba el método de paralelismo de regiones. El algoritmo desarrollado es totalmente elástico en el sentido que se pueden buscar diferentes figuras geométricas.

3.3 Aprendizaje automático.

En general, el aprendizaje automático gira en torno a la construcción de modelos que, utilizando la experiencia, sean capaces de mejorar automáticamente su rendimiento. Este campo ha recibido la influencia de otros muchos campos como la estadística, la inteligencia artificial, la biología y la teoría de la información, entre otros. Dependiendo del tipo de realimentación, se puede clasificar el tipo de aprendizaje en tres grandes grupos: *aprendizaje supervisado*, *aprendizaje no supervisado* y *aprendizaje por refuerzo*.

El *aprendizaje supervisado* consiste en un tipo de aprendizaje automático en donde al algoritmo que se utiliza se le proporcionan una serie de ejemplos con sus correspondientes etiquetas, es decir, todos los ejemplos han sido clasificados "a priori". De esta forma en el proceso de aprendizaje, el algoritmo compara su salida actual con la etiqueta del ejemplo para luego realizar los cambios que sean necesarios. En el caso del *aprendizaje supervisado* cada ejemplo (a menudo llamado instancia o muestra) dentro del conjunto de aprendizaje se puede expresar mediante la forma atributo-valor o mediante relaciones. Dentro del grupo de algoritmos que utilizan la representación de atributo-valor existen a su vez dos grupos: *algoritmos simbólicos* y *subsimbólicos*. Entre los *algoritmos simbólicos* se pueden destacar los árboles de decisión [Quinlan 1986], las máquinas de vectores soporte [Boser 1992, Vapnik 1995] y los sistemas basados en reglas [Michalski 1983, Frank 1998]. Ejemplos de aprendizaje *subsimbólico* son las redes de neuronas [Haykin 1999] y los algoritmos genéticos [Holland 1975]. El objetivo del *aprendizaje no supervisado* es clasificar las muestras de entrada en grupos sin tener conocimiento a priori de los grupos que pueden existir. En el *aprendizaje por refuerzo* [Watkins 1988, Watkins 1992], el algoritmo utilizado recibe las entradas y una evaluación (en forma de recompensa, que puede venir retardada en el tiempo) de tal manera que el algoritmo debe aprender que acción es la que brinda mayor rendimiento a largo plazo.

Otro tipo de clasificación de las técnicas de aprendizaje automático es la que clasifica dicho aprendizaje como *aprendizaje inductivo* [Minton 1992] y *aprendizaje deductivo*. La idea principal del *aprendizaje inductivo* es que a partir de un número elevado de ejemplos, asumiendo que existe un concepto o conceptos en los que se encuadran dichos ejemplos, se puede construir una representación de dichos conceptos. Una vez construida la representación, ésta puede ser utilizada para realizar predicciones sobre nuevas instancias, utilizando esencialmente el

conocimiento obtenido a partir de los ejemplos disponibles. Algoritmos que se clasifican como inductivos son los que se han mencionado hasta este punto dentro de la clasificación de aprendizaje automático. Por el contrario, el *aprendizaje deductivo* utiliza, principalmente, conocimiento del dominio y algún ejemplo, de tal forma que su objetivo principal es hacer operativo el conocimiento que posee el sistema y no generar conocimiento nuevo.

3.3.1 Motivación de la utilización del aprendizaje automático.

Las siguientes secciones tienen como objetivo realizar una breve introducción del capítulo cinco de esta Tesis. La idea es mostrar los fundamentos teóricos de uno de los métodos de clasificación comentados en el apartado 3.3, las máquinas de vectores soporte. Se verán las bases teóricas de este sistema de clasificación *supervisado* así como las limitaciones computacionales derivadas de su cálculo y que provocan su necesaria paralelización. En el mismo capítulo se mostrarán varias formas de paralelizar este algoritmo de clasificación. También se realizaron experimentos con diferentes parametrizaciones al que fue sometido el clúster DIA para la creación del modelo de clasificación de imágenes del sistema Scattering Thomson del TJ-II.

3.3.2 Máquinas de vectores soporte.

Las máquinas de vectores soporte (SVM), han mostrado en los últimos años su capacidad en la clasificación y reconocimiento de patrones en general. El objetivo de este capítulo es presentar los fundamentos básicos, tanto teóricos como prácticos de las SVM y mostrar su potencial en tareas de clasificación. Intuitivamente, dado un grupo de datos distribuidos en dos clases, una SVM lineal busca un hiperplano de tal manera que la mayor cantidad de puntos de la misma clase queden al mismo lado, mientras se maximiza la distancia de dichas clases al hiperplano de separación entre las dos clases. Según Vapnik [Vapnik 1995], este hiperplano minimiza el riesgo de clasificaciones erróneas en el grupo tomado para realizar el proceso de validación.

3.3.2.1 Ventajas e inconvenientes de las SVM.

Esta nueva técnica presenta como ventajas más importantes las siguientes:

- **Elevada generalización.** Las SVM consiguen separar correctamente los datos de entrenamiento, pero además, logran que el límite (o margen) trazado entre las distintas zonas sea el máximo posible.
- **Espacio de características.** Las SVM atacan los problemas en los que los datos no son linealmente separables mediante un mapeado del conjunto de datos de entrenamiento a un espacio de dimensión superior llamado *espacio de características* . Gracias a esta transformación se consigue que la separación lineal sea más probable. Las funciones encargadas de llevar a cabo este mapeado se conocen como funciones kernel y deben cumplir una serie de propiedades matemáticas recogidas en el Teorema de Mercer, [Courant 1953].
- **Interpretación de la solución.** La solución al problema final viene dada en función de una superficie generada por combinación lineal de unos puntos del conjunto de entrenamiento, los vectores soporte (*SVs*). Su estudio es fundamental para entender y controlar la capacidad del modelo.

Entre las desventajas cabe citar las siguientes:

- **Elevado número de restricciones.** Dado que el cálculo del hiperplano óptimo de decisión está sujeto a una serie de restricciones sobre el espacio de entrada, cuando el número de patrones de entrenamiento sea elevado, se puede tener una carga computacional muy alta asociada a la resolución del hiperplano.
- **Sensibilidad numérica.** La matriz Hessiana utilizada en la optimización es muy sensible al redondeo o perdida de cifras significativas, lo cual puede afectar al resultado final del problema.
- **Parámetros libres.** En las SVM se dispone de libertad para elegir los valores de varios de sus parámetros de ajuste: penalización, kernel, error permitido. Los valores adecuados de cada uno de ellos deben obtenerse a través de la experiencia y de un conocimiento detallado del problema al que se aplique o mediante un test en un conjunto de datos representativos del problema.
- **Restricciones de clases.** Otra de sus desventajas es que están pensadas para resolver problemas de dos clases. Para solventar esta limitación existen estrategias mono-objetivo y estrategias multi-objetivo. Estas últimas son las que se tratarán con detalle.

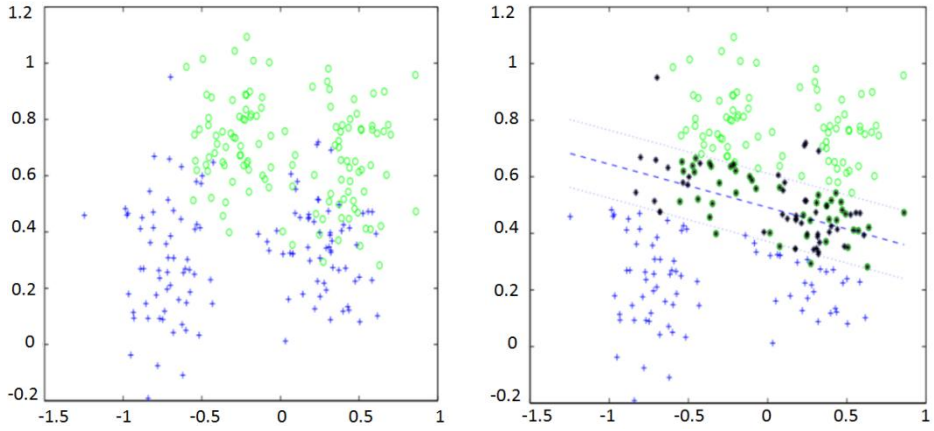


Figura 3.3. Cálculo de vectores soporte con función kernel Lineal.

3.3.2.2 Clasificación con Máquinas Vectores Soporte.

Las SVMs Binarias [Boser 1992, Vapnik 1995] son máquinas de *aprendizaje supervisado* enfocadas a clasificar elementos que pueden pertenecer a una de dos categorías, como se muestra en la figura 3.3. Dado un conjunto de ejemplos, compuestos de una serie de características y una etiqueta que establece a cual de las dos clases pertenece cada ejemplo, la idea principal de las SVMs es disponer situar a los ejemplos en un espacio vectorial y trazar un hiperplano que deje

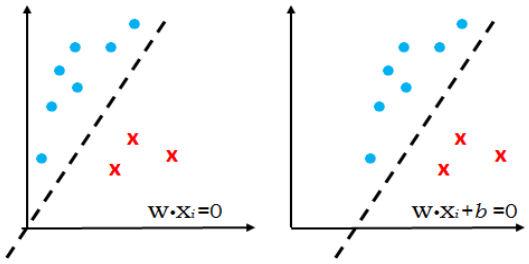


Figura 3.4. Efecto del *bias.*

a todos los elementos de una clase a un lado del hiperplano y los pertenecientes a la otra clase, en el otro lado. Una descripción más formal del método se describe a continuación. Se supone que se dispone de un conjunto de ejemplos $S=\{x_1,x_{2,\ldots,}\ x_m\}$, todos pertenecientes a un espacio característico $S \subset X \subseteq \mathbb{R}^n$, algunos de ellos pertenecen a la clase C_- y otros a la clase C_+. El fin de la SVM binaria consiste en

construir un hiperplano $\mathcal{H} = \{x \in \mathcal{X}: w \cdot x + b = 0\}$ con los parámetros $x \in \mathcal{X}\backslash\{0\}, b \in \mathbb{R}$ tal que se cumplan las dos condiciones siguientes:

1. Los ejemplos de la clase C_- deben quedar a un lado del hiperplano $C_- = \{x_i \in \mathcal{X}: w \cdot x_i + b < 0\}$, y los de la clase C_+ en el otro lado del hiperplano $C_+ = \{x_i \in \mathcal{X}: w \cdot x_i + b > 0\}$. Donde w es conocido como el vector de pesos y b es conocido como *bias* e indica la distancia al origen (figura 3.4).
2. El margen del hiperplano, definido como dos veces la distancia desde el hiperplano hasta el punto/s más cercano/s a éste, debe tener anchura máxima.

Matemáticamente, ρ se puede definir como:

$$\rho = \text{máx } \arg\max_{(w)} \frac{2y_j\left(w \cdot x_j + b\right)}{\|w\|} \tag{3.2}$$

$$\text{Sujeto a } j = \arg\min_{(w)} \frac{2y_j\left(w \cdot x_j + b\right)}{\|w\|}, \forall i \in I$$

Donde $y_i \in \{-1, +1\}$ es la etiqueta que indica a que clase pertenece el ejemplo i y donde $I = \{1,2,\ldots,m\}$ es el conjunto de índices de todos los ejemplos del problema. Si w es el vector que define el hiperplano de separación y se normaliza de tal forma que el producto punto de dicho vector con aquellos ejemplos más cercanos al hiperplano de igual a ±1 (el signo dependiendo de la clase), entonces el margen se puede definir como:

$$\rho = {}^{2}/_{\|w\|} \tag{3.3}$$

Cuando el problema es linealmente separable, es decir, cuando efectivamente es posible trazar un hiperplano que separe correctamente las dos clases, existen infinitos hiperplanos que pueden ser elegidos.

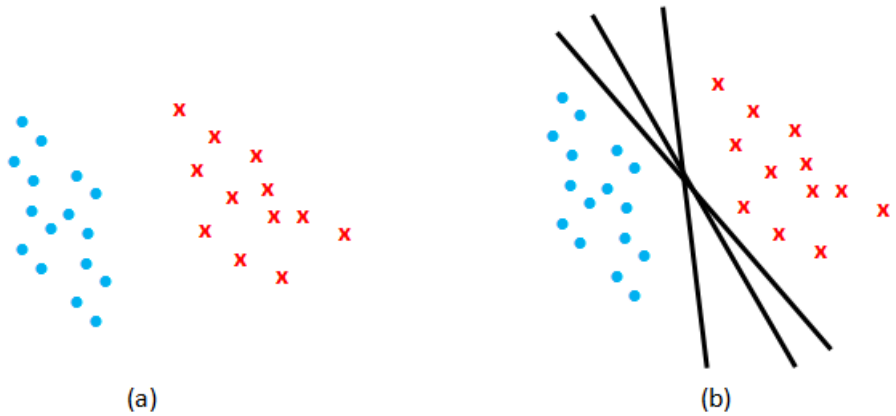


Figura 3.5. Problema Binario hiperplano separación. En 3.5(a) se muestra un problema de clasificación binaria, mientras que en 3.5(b) se presentan varias posibles soluciones de hiperplanos separadores.

En la figura 3.5(b) se pueden apreciar algunas soluciones de ejemplo al problema mostrado en la figura 3.5(a). Matemáticamente, la formulación del hiperplano puede ser definida como:

$$P_{(w,b)} : \text{minimizar } \frac{1}{2}\|w\|^2 \tag{3.4}$$

$$\text{Sujeto a} \quad j = y_i(w \cdot x_i + b \geq 0\}, \forall i \in I \tag{3.5}$$

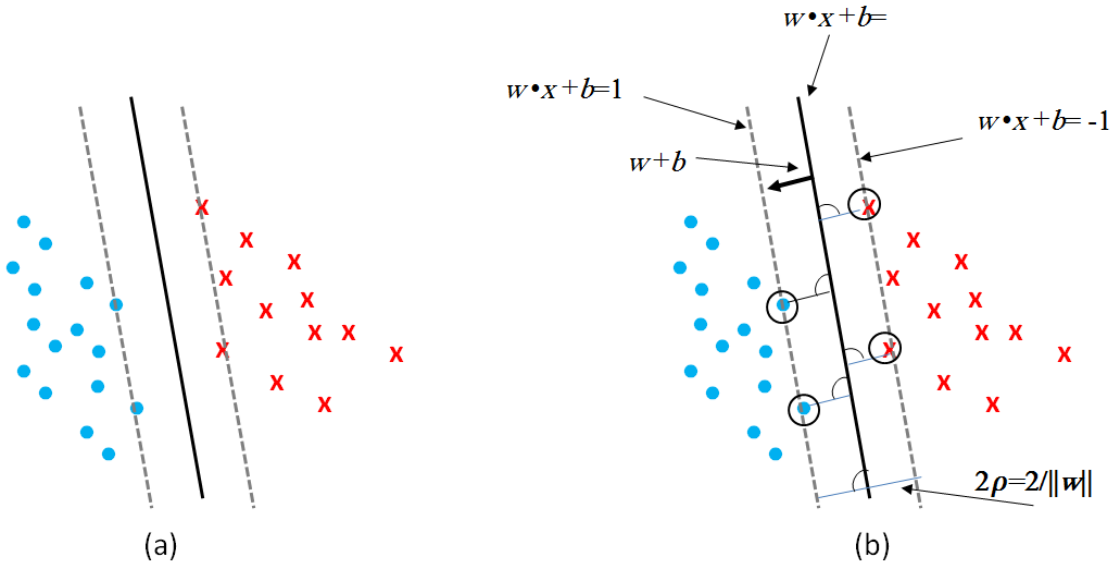


Figura 3.6. Margen óptimo de separación. En la figura 3.6(a) se muestra el hiperplano de mayor margen que separa correctamente ambas clases. En la figura 3.6(b) se muestran los vectores soporte que definen al hiperplano con margen de separación óptimo.

Con este modelo, se puede construir una función de decisión para clasificar futuros ejemplos:

$$f(x) = \textbf{signo}\,(w \cdot\ x + b).$$

La función $f(x)$ devuelve el signo de la clase a la que pertenece la nueva muestra x. En la figura 3.6(b) se puede apreciar que todos los vectores a la derecha del hiperplano gris darán un valor -1 al ser evaluados por la función $f(x)$, mientras que los vectores a la izquierda darán un valor igual a +1.

Algunos autores proponen resolver este problema abordando directamente $P_{(w,b)}$ pero lo que habitualmente se hace es utilizar la técnica de los multiplicadores de Lagrange y resolver el problema dual. Por medio de este método el problema se expresa de la siguiente forma:

$$minimizar_{\{\alpha\}} = \frac{1}{2}\sum_{i\in I}\sum_{j\in I}\alpha_i\,\alpha_j\,y_i y_j\,x_i \cdot x_j^T - \sum_{i\in I}\alpha_i \tag{3.6}$$

$$\text{Sujeto a} \quad \alpha_i \geq 0, \forall i \in I,\ \textstyle\sum_{i\in I}\alpha_i\,y_i = 0 \tag{3.7}$$

Aquellos vectores que tienen un multiplicador de Lagrange α_i con valor mayor que cero se denominan *Vectores Soporte* (VS). Estos son los que definen al hiperplano separador y los límites de su margen. En la figura 3.6(b), los vectores soporte (VS) están simbolizados con círculos. Como se puede intuir, los VS suelen componer una fracción pequeña del total de los ejemplos del problema.

Funciones Kernel.

En muchas ocasiones, los ejemplos que componen los datos de entrada se encuentran en un espacio que no es linealmente separable, pero guardan una estructura que perfectamente podría soportar una frontera de otro tipo. En tales casos, lo que se puede hacer es aplicar una transformación $\phi(\cdot)$ a los ejemplos en el espacio de entrada la cual los traslada a un espacio de mayor dimensionalidad (denominado *espacio característico*), donde un hiperplano separador sea capaz de dividir las dos clases implicadas en el problema. Utilizando esta idea, el problema se puede reformular de la siguiente forma:

$$P_{(w,b)} : \text{minimizar } \frac{1}{2}\|w\|^2 \tag{3.8}$$

$$\text{Sujeto a} \quad y_i(w \cdot \phi(x)_i + b \geq 1\}, \forall i \in I \tag{3.9}$$

Esta estrategia es viable en la mayoría de los casos, pero existe un método alternativo, llamado *kernel trick* (anglicismo de Truco del Kernel) que es menos costoso y que no tiene que trabajar con el problema de espacios de alta o incluso infinita dimensionalidad. El *kernel trick* consiste en utilizar una función Kernel que calcula directamente el producto interno entre dos vectores del espacio característico en un espacio de mayor dimensionalidad, sin necesidad de realizar el mapeo a dicho espacio.

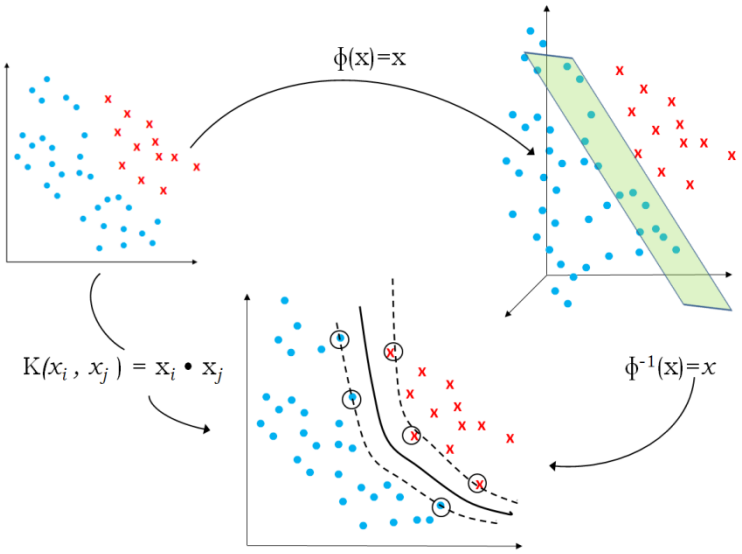


Figura 3.7. Truco de Kernel: El producto de Kernel resume en un paso lo que se tendría que reproducir con una transformación a un espacio vectorial de mayor dimensionalidad, un producto punto en dicho espacio y una proyección posterior al espacio vectorial original.

Dicha función Kernel debe necesariamente ser una función simétrica positiva semi-definida [Scholkopf 2001, Hofmann 2008]. En la figura 3.7 se muestra gráficamente la idea del *kernel trick*. Para hacer uso de este método en las SVMs solo se debe realizar el reemplazo de los producto puntos $x_i \cdot x_j$ por la función kernel deseada $K(w, x_i)$ en la formulación del problema dual:

$$minimar_{\{\alpha\}} D(\alpha) = \frac{1}{2} \sum_{i \in I} \sum_{j \in I} \alpha_i \, \alpha_j \, y_i y_j K_{ij} - \sum_{i \in I} \alpha_i \tag{3.10}$$

$$\text{Sujeto a} \quad \alpha_i \geq 0, \forall i \in I, \; \sum_{i \in I} \alpha_i \, y_i = 0 \tag{3.11}$$

Donde la función kernel $K_{ij} = K(x_j) = \phi(x_i) \cdot \phi(x_j)$.

Las funciones kernel deben cumplir las propiedades de simetría y la propiedad de Cauchy-Swartch, no obstante estas propiedades no son suficientes y se precisa por tanto que la función kernel cumpla las condiciones del teorema de Mercer y de esta forma induzca un espacio de características. Algunas de las funciones kernel más utilizadas y que cumplen las condiciones del teorema de Mercer son las siguientes:

1. **Lineal:** esta función kernel es una transformación lineal del tipo:

$$K(x, z) = \langle Ax \cdot Az \rangle = x^T A^T A z = x^T B Z$$

Dónde $A^T A$ es una matriz semidefinida positiva.

2. **Polinomial:** es una función muy utilizada para funciones no lineales:

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

$$K(x, x) = \langle \langle x, x \rangle + c \rangle^d, \text{ donde c} \in \Re.$$

3. **Radial (RBF):** también conocidas como funciones Gaussianas.

$$K(x, z) = \exp(-\frac{\|x - x\|^2}{\sigma^2})$$

Hay situaciones donde debido a la naturaleza de los datos éstos se encuentran especialmente mezclados justo en la frontera que separa una clase de otra. Las funciones kernel ayudan a separar los vectores. No obstante, es posible que el clasificador trabaje bien con los elementos para los que fue entrenado pero no así

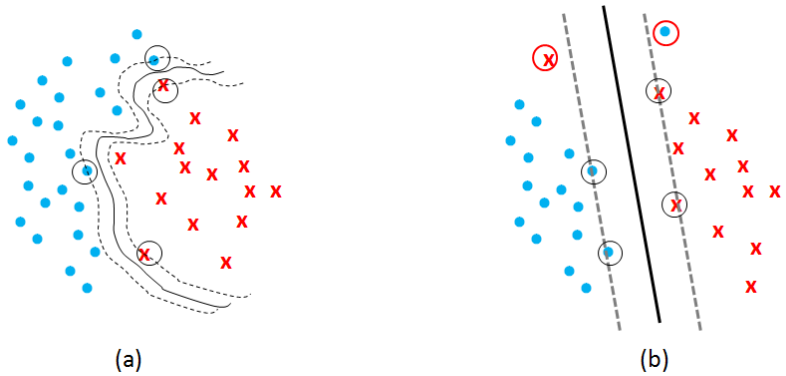


Figura 3.8. SVMs de Margen Suave. 3.8(a) muestra una SVM utilizando una función de Kernel para clasificar correctamente todos los ejemplos de entrenamiento. Bajo círculo negro se muestran los vectores soporte. 3.8(b) presenta una SVM lineal de margen suave. Rodeados en un círculo rojo se muestran los ejemplos mal clasificados.

para nuevos patrones. En estos casos, es aconsejable sacrificar la correcta clasificación de algunos datos de entrenamiento con el fin de lograr una mayor capacidad de generalización de la SVM frente a nuevos patrones. En este contexto entra en juego el concepto de *margen suave* (figura 3.8), éste consiste en permitir un margen más ancho que el óptimo, permitiendo que haya vectores de ambas clases dentro de este o al lado incorrecto de la frontera. La primera implementación de SVM que utilizó este concepto fue la llamada C-SVM. Esta nueva definición tiene un factor de regularización C y variables de holgura ξ_i para ampliar la definición de su función objetivo:

$$P_{\{w,b,\xi\}}: minimizar \frac{1}{2}\|w\|^2 + C\sum_{i\in I}\xi_i \tag{3.12}$$

$$Sujeto\ a\ y_i(w\cdot\phi(x_i)+b) \geq 1-\xi_i, \forall i \in I, \quad \xi_i \geq 0, \forall i \in I \tag{3.13}$$

El termino C se entiende como el grado de importancia que la SVM tiene al clasificar bien cada uno de los vectores frente a la importancia que tiene en lograr el margen más ancho posible. Cuanto mayor sea el valor de C, más importancia le da

a la clasificación y menos al tamaño del margen. Cuando $C \to \infty$ el SVM tiende a comportarse como como una SVM clásica. Cuando C=0 el SVM genera su margen sin preocuparse en la clasificación de datos. La implementación de C en la definición del problema dual es como sigue:

$$minimar_{\{\alpha\}} D(\alpha) = \frac{1}{2} \sum_{i \in I} \sum_{j \in I} \alpha_i \, \alpha_j \, y_i y_j \, k_{ij} - \sum_{i \in I} \alpha_i \tag{3.14}$$

$$\text{Sujeto a } 0 \leq \alpha_i \leq C, \ \forall i \in I, \ \textstyle\sum_{i \in I} \alpha_i \, y_i = 0$$

No obstante la optimización bajo restricciones dada por (3.14) se puede reformular como un problema de *optimización cuadrática* convexa sin restricciones mediante la introducción de multiplicadores Lagrangianos. La nueva expresión sería la siguiente:

$$\begin{aligned} &minimizar \ L(w, b, \xi_i, \alpha, \beta) \\ &\quad = \frac{1}{2} \|w\|^2 + C \sum_{i \in I} \xi_i - \sum_{i \in I} \alpha_i \, [y_i \, (x_i \cdot w + b) - 1 + \xi_i \,] - \sum_{i \in I} \beta_i \, \xi_i \end{aligned} \tag{3.15}$$

Donde $\alpha = (\alpha_1, \ \alpha_2, \ldots\ldots \alpha_i)$ y $\beta = (\beta_2, \ \beta_2, \ldots\ldots \beta_i)$ son multiplicadores lagrangianos no negativos. El problema de optimización (3.15) puede ser reconvertido en un problema dual mucho más fácil de resolver. La optimización del problema puede ser convertida en un problema dual:

$$maximizar L(\alpha) \ = \sum_{i \in I} \alpha_i \ - \frac{1}{2} \sum_{i,j \in I} \alpha_i \, \alpha_j \ y_i \, y_j \ \langle x, x_j \rangle \tag{3.16}$$

$$\text{Sujeto a } \textstyle\sum_{i \in I} y_i \, \alpha_i \ = 0, donde \ 0 \leq \alpha_i \leq C, \ \forall i \in I$$

Las soluciones de (3.16) se obtienen mediante *programación cuadrática*. Una vez encontrados los multiplicadores óptimos α_i el clasificador queda definido por:

$$f(x) = signo(\sum_{i \in I} \alpha_i \, y_i \, x_i \cdot x + b) \tag{3.17}$$

Las condiciones de optimización de **Karush-Khun-Tucker** aseguran que el número de multiplicadores Lagrangianos es mucho más pequeño que el número de datos a tratar. Los puntos de estudio asociados a estos multiplicadores Lagrangianos es lo que ya se ha denominado como vectores soporte. El *bias* puede ser calculado utilizando los vectores soporte como:

$$b = y_{sv} - \sum_{i \in SV} \alpha_i \, y_i \, x_i \cdot x_{sv} \tag{3.18}$$

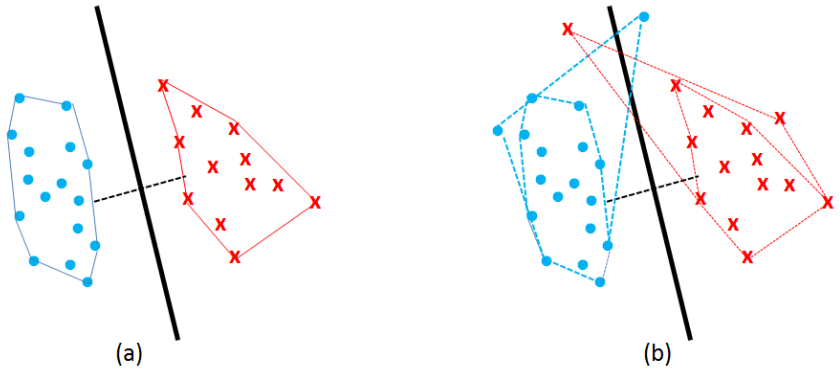


Figura 3.9. Interpretación geométrica de las SVMs: En 3.9(a) se muestran las envolturas convexas que rodean a ambas clases. El hiperplano que divide por la mitad al segmento que une los puntos más cercanos entre ambas envolturas coincide con el generado mediante una SVM binaria tradicional. En 3.9(b) se muestra un caso en donde las envolturas convexas de las clases se superponen. En este caso no tiene sentido trazar el segmento que une a estas envolturas, pero si aquel que separa a las envolturas convexas reducidas. El concepto de envolturas convexas es similar al de márgenes suaves mostrado en la figura 3.8(b).

Interpretación geométrica de las SVMs.

Además de la visión tradicional de interpretar la SVM como un margen del hiperplano que separa a los ejemplos de las distintas clases, existe una forma alternativa de abordar el problema y que resulta ser equivalente a la tradicional. Dicha forma, comúnmente denominada como visión *geométrica* de las SVMs [Bennet 1997, Crisp 2000, Bennet 2000], consiste en considerar las envolturas convexas de

ambas clases (H_- y H_+) y trazar el segmento que une los puntos v_- y v_+ más cercanos de dichas envolturas situando el hiperplano separador en la mitad del segmento.

Los puntos v_- y v_+ que unen al segmento se denominan prototipos de las clases $\mathcal{C}_-$ y $\mathcal{C}_+$. Ambos son combinaciones convexas de sus respectivas envolturas, por lo cual se pueden definir como una suma ponderada de los puntos que pertenecen a cada clase:

$$minimar_{\{\alpha\}} D(\alpha) = \|v_- - v_+\| \tag{3.19}$$

$$\text{Sujeto a } \; v_r = \sum_{i \in I_r} \alpha_i x_i, \; r \in \{-, +\}$$

$$\sum_{i \in I_r} \alpha_i = 1, \qquad r \in \{-, +\}$$

$$0 \leq \alpha_i \leq 1, \qquad \forall i \in I.$$

Donde $I_r = \{i \in I_r : x_i \in \mathcal{C}_r\}$ es el conjunto de todos los índices de los ejemplos pertenecientes a la clase $\mathcal{C}_r$. En caso de que el ruido sea tal que los datos de una y otra clase se mezclen, produciendo que las envolturas convexas se solapen, la idea de buscar el segmento de menor distancia que las une pierde sentido. En estos casos entra en juego la noción de envolturas reducidas. Es decir, en lugar de considerar la envoltura que abarca a todos los puntos de una clase, la combinación convexa se puede restringir para que se considere una envoltura convexa que abarque solo a una porción de dichos datos. Esto se logra restringiendo el valor al cual pueden optar los elementos α_i de la combinación a un máximo de μ , donde $p = \left[1/\mu\right]$ viene a ser el número mínimo de ejemplos que deben participar en la combinación convexa de su clase con un valor de α_i distinto de cero.

En la figura 3.9 se puede apreciar el concepto de envolturas convexas, la forma en que definen el hiperplano separador y como se forman las envolturas convexas reducidas.

3.3.3 Justificación de la paralelización.

Dado un algoritmo de aprendizaje, puede haber dos limitaciones a la hora de resolver el problema desde el punto de vista de los recursos como pueden ser: el

número de muestras de entrenamiento y el tiempo computacional. Teniendo en cuenta estas dos restricciones se puede hablar de aprendizaje a pequeña escala o a gran escala [Joachims 1999].

1. Los problemas de aprendizaje a pequeña escala tienen la restricción de un número bajo de muestras y ello presenta errores de aproximación y estimación. No obstante, los errores de optimización pueden ser reducidos a niveles insignificantes si no se tienen limitaciones de tiempo de computación.
2. Los problemas de aprendizaje de gran escala tienen la restricción del tiempo computacional. Además ajustando los parámetros de aproximación de la familia utilizada se puede obtener una mejor aproximación. En este caso los errores son inferiores por disponer de más muestras de entrenamiento.

La limitación del tiempo computacional ha sido siempre un motivo de estudio en las SVM cuyo objetivo es siempre conseguir reducir el error de optimización.

3.3.3.1 Coste computacional.

Cuando se aborda un problema de clasificación de gran escala los números de muestras que se manejan en algunos casos son del orden de millones. Como ya se ha comentado, la dimensionalidad de las muestras es variable pero debido a las restricciones computacionales estas dimensiones intentan estar en una media de $15 \approx 60$ dimensiones. Estos números tratan de orientar en las limitaciones que siguen a continuación. Existen dos límites en el coste computacional de un algoritmo para abordar un problema de *optimización cuadrática* para resolver matrices kernel del tipo $K_{ij} = K(x_i, x_j)$.

1. Si se supone que se conocen los coeficientes de las muestras que no son vectores soporte ($\alpha_i = 0$) y las muestras que están limitadas por vectores soporte ($\alpha_i = \pm C$), entonces el cálculo de los coeficientes de los R restantes vectores soporte supone procesar una matriz de tamaño $R \times R$. Este proceso requiere normalmente un número de operaciones proporcional a R^3.
2. El mero hecho de verificar que un vector α es una solución, involucra el cálculo de los gradientes de $W(\alpha)$ y comprobar las condiciones de optimización de Karush-Khun-Tucker. Con n muestras y S vectores soporte, este proceso requiere un número de operaciones proporcional a nS.

Pocos vectores alcanzan el límite superior cuando el valor de C es grande. Para ese caso el esfuerzo está determinado por $R^3 \approx S^3$. En el caso contrario la cantidad

nS puede llegar a ser realmente grande. Por lo tanto se concluye el número final de vectores soporte es el componente crítico del costo computacional.

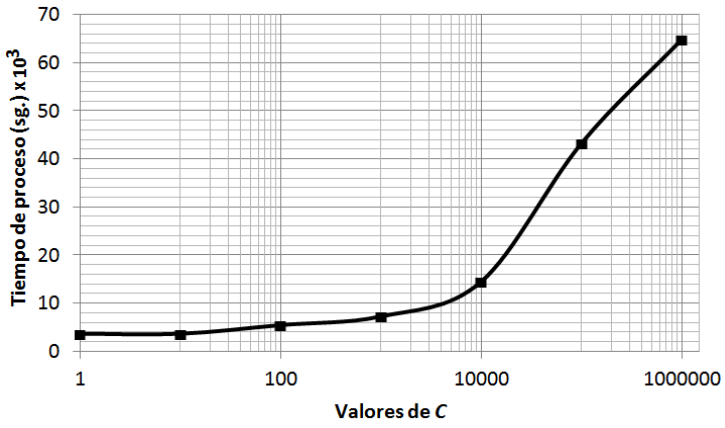


Figura 3.10. Efecto del aumento de C en la clasificación de formas de onda del TJ-II.

Asumiendo que se dispone de unos conjuntos de entrenamiento cada vez más grandes y que se han extraído de una distribución desconocida $P(xy)$, siendo B la tasa del error alcanzado por la mejor función de decisión para esa distribución, cuando $B>0$ se puede asumir que el número de vectores soporte es asintóticamente equivalente a $2nB$ [Steinwart 2008]. Por tanto, independientemente del algoritmo utilizado, el coste computacional de la solución al problema de optimización cuadrática crece al menos como n^2 cuando C es pequeña y n^3 cuando C es grande. Las experiencias actuales demuestran que los clasificadores de hoy en día se acercan mucho a estas leyes de escalabilidad [Chang 2011, Collobert 2001].

A modo de ejemplo la figura 3.10 muestra el tiempo de respuesta para la clasificación de formas de onda del TJ-II a medida que aumenta el parámetro *C*. En general, tener pocas instancias o muestras de entrenamiento y muchos atributos por muestra facilita la separación lineal de los datos.

Sin embargo todo obedece a factores constantes, es decir, cuando el número de muestras crece, el valor de la matriz kernel, $K_{ij} = K(x_i, x_j)$, se hace tan grande que no puede almacenarse en memoria. Esto obliga, como se verá en el capítulo cinco, que los valores de la matriz kernel deben calcularse de manera dinámica y ser almacenados en la memoria caché para su posterior utilización cuando sean necesitados.

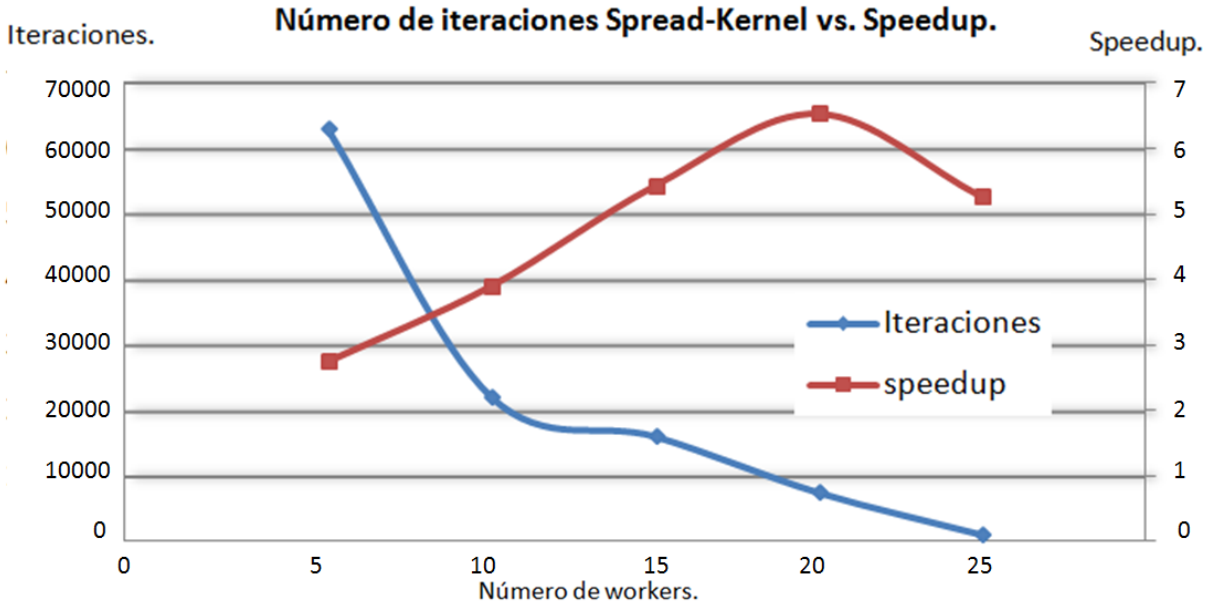


Figura 3.11. Número de iteraciones frente al progreso del speedup. La figura muestra cómo en función del número de *workers* las iteraciones disminuyen lo que implica un tiempo de convergencia menor del algoritmo.

El dimensionamiento de la memoria caché es un punto que debe ser tratado con mucho cuidado más aún si se dispone de una arquitectura distribuida con memoria compartida. Además del problema del almacenamiento, llegar a la convergencia del algoritmo exige muchas más iteraciones. Estas iteraciones pueden ser repartidas entre diferentes *workers.* La figura 3.11 muestra cómo incrementando el número de *workers* se acelera el tiempo en llegar a la convergencia.

Como ya se ha visto, en un problema de clasificación existen dos etapas. La primera es la del entrenamiento del modelo y la segunda la de validación mediante las muestras de test. El reto de paralelizar la fase de entrenamiento pasa por optimizar la carga de muestras de esta fase. Ya se ha comentado que si los datos aumentan, la matriz kernel puede llegar a no poder almacenarse en memoria. Pero ¿qué ocurre con los datos iniciales de entrenamiento de entrada?. Si se tiene en cuenta una variable multidimensional, por ejemplo de 1000000 x 60, el tamaño de la variable a procesar sería de 60 millones de celdas. Si además cada celda es del

tipo *double,* ocupando 8 bytes por cada celda, la variable alcanzaría un tamaño aproximadamente de 457 Mb. Y esto sólo para una variable que pertenece a una clase. Si se trata de un problema multiclase, por ejemplo de cinco clases, la carga de datos se convierte en extremadamente grande. Es decir, en el caso de cinco clases con más de un millón de muestras para cada clase y con el método *uno contra todos* habría algún paso en el que el proceso compararía un millón de muestras frente a cuatro millones de muestras. Además de estas variables, el ordenador debe almacenar en memoria el sistema operativo, gestión de recursos, cálculos intermedios, etc.. No cabe duda por tanto que uno de los retos al abordar un problema de clasificación es la memoria inicial necesaria para cargar las muestras a la hora de construir el modelo.

Actualmente se recopilan datos estructurados y no estructurados para ser procesados en el menor tiempo posible. Con la explosión, por ejemplo, del *big data*[3] el volumen de los datos esta sobrepasando los límites conocidos hasta ahora.

3.4 Predicción.

Las siguientes secciones analizan los predictores conformales como técnica que es capaz de calcular un margen de error en las predicciones. La forma de acotar éstos márgenes es mediante el establecimiento de un margen de confianza. Se mostrarán las características que definen estas técnicas y la necesidad de utilizar la computación paralela en el cálculo de los márgenes de error.

Justificación del Predictor Probabilístico.

Un algoritmo de aprendizaje es aquel proceso capaz de dar respuesta al problema de aprendizaje a partir de ejemplos. En [Cherkassky 1998] se amplía esta definición, en la cual se define *algoritmo de aprendizaje a partir de ejemplos* como aquel proceso que es capaz de elegir una única función a partir de un conjunto de entrenamiento, dando respuesta de esa forma al problema planteado de aprendizaje a partir de ejemplos. La necesidad de asumir cierto conocimiento de antemano sobre la forma del modelo buscado es esencial para conseguir la unicidad de la solución. Este conocimiento será insertado en el algoritmo de aprendizaje en función del principio *inductivo*. La función se debe estimar para todos los tipos de

[3] Grandes volúmenes de información estructurada y no estructurada que no puede ser procesada por medios tradicionales.

valores del espacio de entrada *X*. La función obtenida no solo debe comportarse bien para los valores de entrenamiento sino que debe tener la propiedad de *generalizar bien*, por lo que es necesario estimar un modelo que permita predecir la salida de cualquier entrada del espacio *X,* siguiendo el proceso de inducción-predicción definido.

Ahora bien, en el análisis de la multiprobabilidad se plantean dudas sobre el grado de acierto de las predicciones, o bien la cercanía a la etiqueta real de un objeto para los casos en los que la etiqueta es un número. En [Lambrou 2012a] se muestran diferentes algoritmos que proporcionan estimaciones de probabilidades pero no garantizan que la probabilidad que estiman esté bien calibrada.

Si el objetivo es predecir la etiqueta y de un nuevo objeto x y la predicción es $\hat{y}$, entonces la pegunta que se puede plantear es ¿cuál es el nivel de confianza que tiene $y = \hat{y}$?.

Los predictores conformales [Nouretdinov 2001] utilizan el entrenamiento o bien, el histórico de ejecuciones, para determinar el nivel de confianza en la predicción de la clasificación. Se podría decir que los predictores conformales son "*predictores de confianza*".

En la línea de obtener una predicción de confianza de la medida, se puede indicar también un nivel de error de probabilidad. Por ejemplo, si se asume que el error de probabilidad es del 2% se tendría un nivel de confianza del 98%.

Muchos de los algoritmos de clasificación en *machine learning* (anglicismo de aprendizaje automático) están basados en los principios de *inducción* y *deducción*. En un problema de clasificación, cuando se procesan una serie de datos, el objetivo es generar un modelo de clasificación general. Pero este modelo tiene ciertos riesgos: en primer lugar la limitación existente cuando el número de datos es pequeño. En este caso el modelo general falla debido a los pocos datos que tiene y no puede generar la regla debido a la limitación del número de datos. Y en segundo lugar está el problema de la confianza y credibilidad expuesto anteriormente, debido a que no está capacitado para dichos datos. Los predictores conformales están basados en el método de la *transducción*. La diferencia entre los métodos inductivos y transductivos se detalla en [Vapnik 1998, Vapnik 2000]. La predicción *transductiva* se basa en utilizar directamente los datos de entrenamiento.

Los predictores conformales se pueden aplicar a diferentes métodos de clasificación o regresión como pueden redes neuronales, redes bayesianas, arboles de decisión y por supuesto las máquinas de vector soporte. No obstante el trabajo

que ocupa esta memoria ha sido desarrollado con el método de máquinas de vectores soporte aplicando técnicas de programación paralela.

Los predictores conformales fueron introducidos por Vovk [Vovk 2005]. A partir del método de clasificación que se esté utilizando, el modelo construye una medida de *no conformidad*. Esta medida da una idea sobre la diferencia que hay entre la nueva muestra analizada y el resto de muestras precedentes ya analizadas. El *algoritmo conformal* convierte esa medida de *no-conformidad* en *regiones de predicción*.

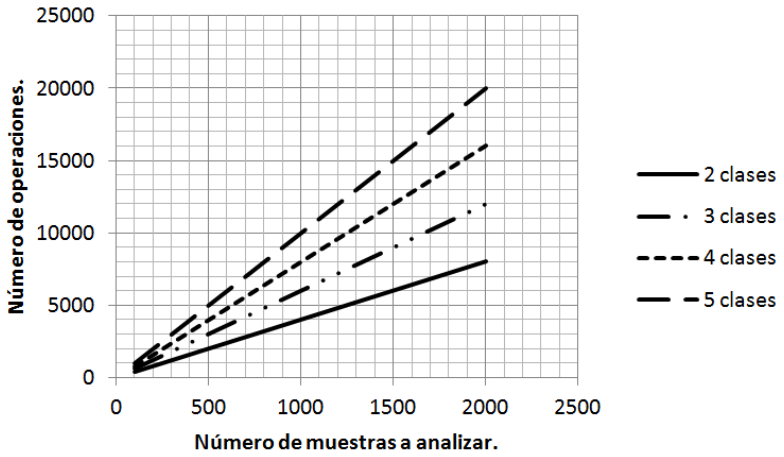


Figura 3.12. Incremento del número de operaciones en función del número de clases y muestras de test.

Coste computacional.

El capítulo seis de esta Tesis se centra en un tipo particular de predictores conformales: los predictores Venn. De forma general, los predictores Venn clasifican una nueva muestra o ejemplo en base a múltiples hipótesis. La clasificación se satisface calculando la probabilidad de cada una de las posibles clases que intervienen en el cálculo. Los cálculos obviamente crecen y hacen del predictor Venn computacionalmente ineficiente (véase figura 3.12). Esta característica la hacen los algoritmos *subyacentes* al predictor. Los algoritmos subyacentes son algoritmos de clasificación necesarios para el predictor. Se analizará la implementación del predictor Venn y se propondrán alternativas para hacer del algoritmo un proceso computacionalmente más ligero sin perder la efectividad.

Capítulo IV.
Búsqueda de patrones en Espectrogramas.

4.1 Introducción.

Con el incremento de la capacidad y la velocidad computacional se ha visto un esfuerzo creciente por desarrollar sistemas sofisticados de tiempo real que emulen las habilidades humanas, entre ellas la visión, abarcando la identificación y clasificación de objetos en grupos o categorías de acuerdo a sus similitudes o semejanzas.

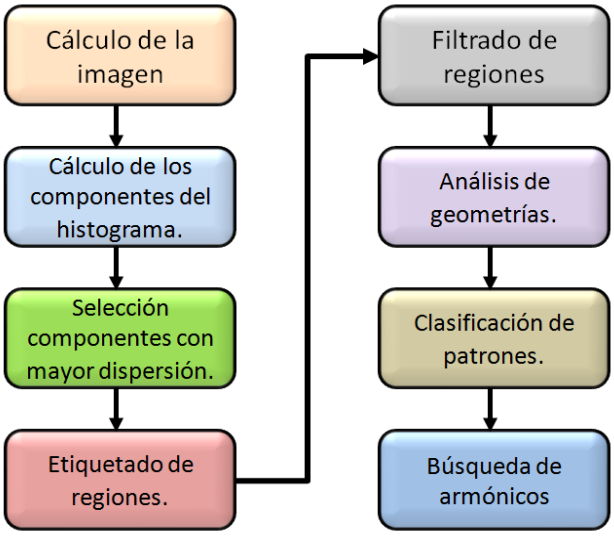


Figura 4.1. Flujo del procedimiento para la búsqueda de armónicos.

El reconocimiento de patrones asume que la imagen puede contener uno o más objetos y que cada objeto pertenece a uno de varios tipos, categorías o clases de patrones [Dormido-Canto 2006, Vega 2008a]. Dada una imagen digital que contiene varios objetos, el proceso de reconocimiento de patrones consta de 3 etapas. La primera etapa se denomina segmentación de la imagen, en donde los objetos de interés son aislados del resto de la imagen. La segunda etapa, es la de extracción de rasgos, en donde los objetos son medidos. Una medida es un valor de alguna

propiedad cuantificable del objeto. Un rasgo es una función de una o más medidas, computadas de tal forma que cuantifican algunas características importantes del objeto. Con estos rasgos se construye lo que se conoce como el vector de rasgos. Por último la tercera fase del reconocimiento de patrones es la clasificación.

Previamente a estas tres tareas se debe realizar la adquisición de la imagen que se consigue mediante el cálculo del espectrograma. Un espectrograma es una gráfica tridimensional que representa la variación temporal de la energía frecuencial de una señal. Los datos para calcular los espectrogramas y que sirven de entrada al modelo están almacenados en ficheros de texto de diferentes tamaños. Hay que tener especial cuidado en el tratamiento de estos datos debido a que con un balanceo de carga eficiente se pueden mejorar sensiblemente los tiempos de respuesta (este hecho se analizó en el capítulo 2 de la Tesis). La figura 4.1 muestra de forma genérica el flujo del procedimiento.

El capítulo 4 se ha estructurado de la siguiente manera: en primer lugar se presentan los principios para el cálculo del espectrograma a partir del cual se realiza el análisis. Posteriormente se realiza una breve introducción a la segmentación de imágenes y técnicas utilizadas en el proceso. Seguidamente se muestra el proceso de búsqueda de armónicos propuesto en la metodológia. A continuación se expone la paralelización de la metodología así cómo las funciones desarrolladas para cada *worker*. Finalmente para terminar se muestran los resultados obtenidos. Los datos procesados corresponden a señales de evolución temporal de la base de datos del Tokamak JET. En total fueron procesadas 95937 descargas con un volumen de 6 Tb. de información.

4.2 Patrones considerados: BP1, BP2 y BP3.

Los espectrogramas permiten el análisis tiempo-frecuencia de señales de evolución temporal. Representan la magnitud de Fourier en función de la frecuencia y el tiempo. Los espectrogramas son muy útiles en números campos de la ciencia para identificar patrones específicos en el espacio tiempo-frecuencia. En fusión nuclear, los espectrogramas se pueden utilizar por ejemplo en el reconocimiento de los modos de tearing, de modos outer, de modos Alfvén, de turbulencias del plasma e inestabilidades, entre otros.

En fusión nuclear, una aplicación típica de reconocimiento de patrones puede ser la identificación de un patrón particular dentro de los espectrogramas de varias señales de evolución temporal de la misma descarga. La búsqueda automática

permite tanto la reducción de los esfuerzos humanos para identificar y localizar eventos como la estandarización de las búsquedas. Este capítulo tiene por objetivo definir una técnica genérica, automática y paralelizable para el reconocimiento de diferentes patrones en espectrogramas. La técnica está compuesta por una serie de fases: cálculo del espectrograma, binarización, segmentación, filtrado de regiones y, por último, la identificación de patrones. Se han considerado tres tipos de patrones básicos: señales con crecimiento lineal de frecuencia (BP1), señales con decrecimiento lineal de frecuencia (BP2) e intervalos con frecuencia constante (BP3). Estas tres estructuras básicas se pueden combinar y es posible reconocer todas las secuencias compuestas a partir de cualquier patrón básico (BP1-BP2-BP3) (figura 4.2).

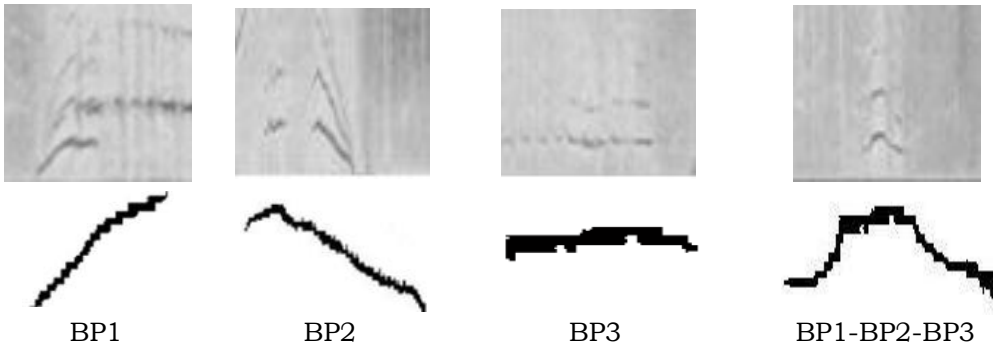


Figura 4.2. Patrones detectados en descargas de plasma del Tokamak JET.

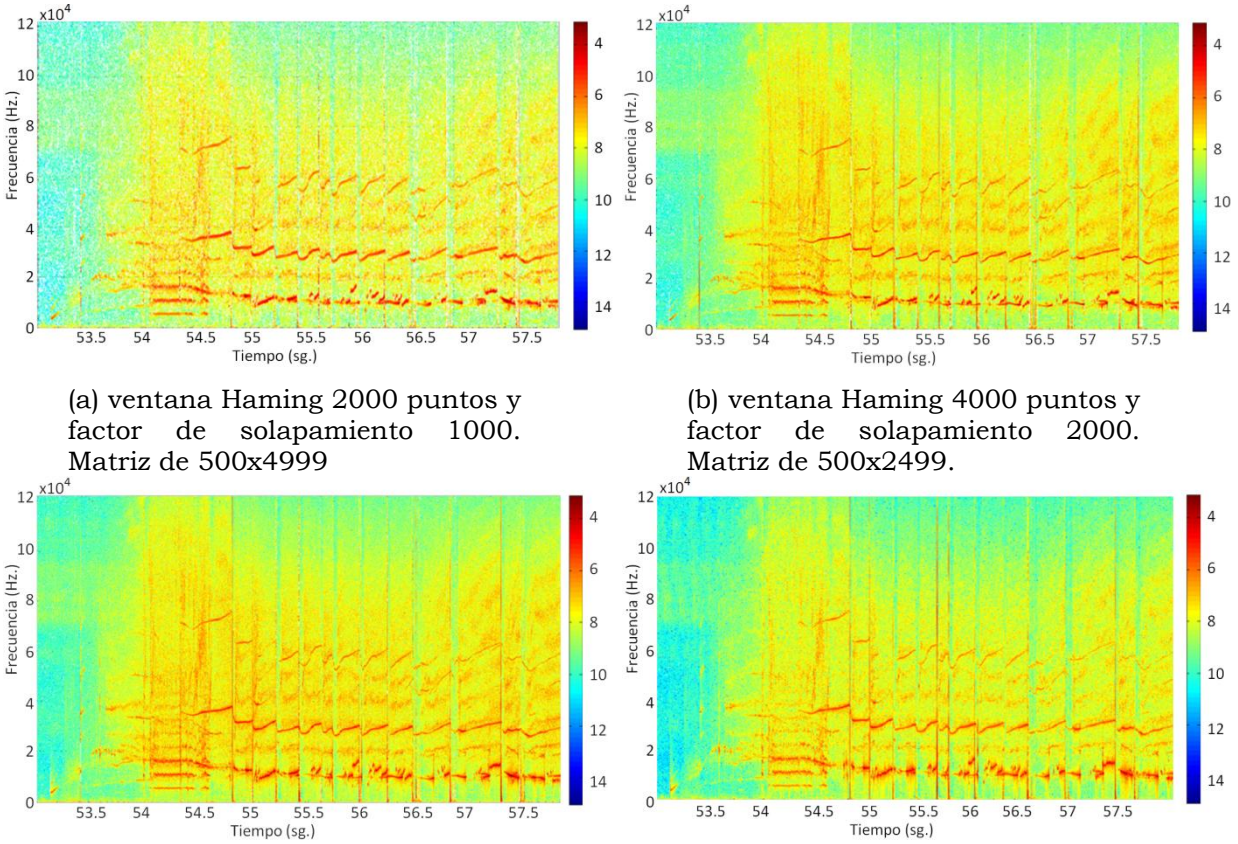
4.3 Metodología.

Esta sección analiza de forma detallada la aplicación de diferentes métodos de segmentación de imágenes para la automatización en la búsqueda de patrones dentro de espectrogramas.

4.3.1 Adquisición de la imagen. Espectrograma de una señal.

El análisis espectral es crucial para comprender las características de la señal, y puede aplicarse a cualquier tipo de señal, incluidas señales de radar, señales de audio, datos sísmicos, datos de valores financieros y señales biomédicas. Un

espectrograma es la representación de la magnitud de la transformada de Fourier de una señal.



(a) ventana Haming 2000 puntos y factor de solapamiento 1000. Matriz de 500x4999

(b) ventana Haming 4000 puntos y factor de solapamiento 2000. Matriz de 500x2499.

(c) ventana Haming 6000 puntos y factor de solapamiento 3000. Matriz de 500x1665.

(d) ventana Haming 10000 puntos y factor de solapamiento 5000. Matriz de 500x999.

Figura 4.3. Cálculo de espectrogramas.

El armónico de una onda es un componente sinusoidal de una señal y su frecuencia es un múltiplo de la fundamental. La amplitud de los armónicos más altos es mucho menor que la amplitud de la onda fundamental y tiende a cero. A modo de ejemplo, en audio, los armónicos por encima del quinto o del sexto generalmente son inaudibles.

La presencia de las señales armónicas, ocasiona que se produzcan por ejemplo interferencia en las señales, incremento de la temperatura, etc. La figura 3.1 del capítulo anterior muestra el espectrograma de la señal PP8 del canal 6 correspondiente a la descarga número 79455 con pared de carbono. En el intervalo temporal comprendido entre 54 y 54.5 sg., se puede comprobar la presencia de

armónicos. Se trata de las líneas paralelas que están en el intervalo frecuencial entre 5 KHz. y 20 Khz.

El espectrograma es la base principal y punto de partida del análisis. Para poder lograr el objetivo es necesario que la imagen tenga la mayor definición posible. Para ello es recomendable que la imagen tenga el mayor número de líneas y columnas posible. Además se debe conseguir que el cálculo sea lo más rápido posible puesto que la metodología está orientada al análisis masivo de datos.

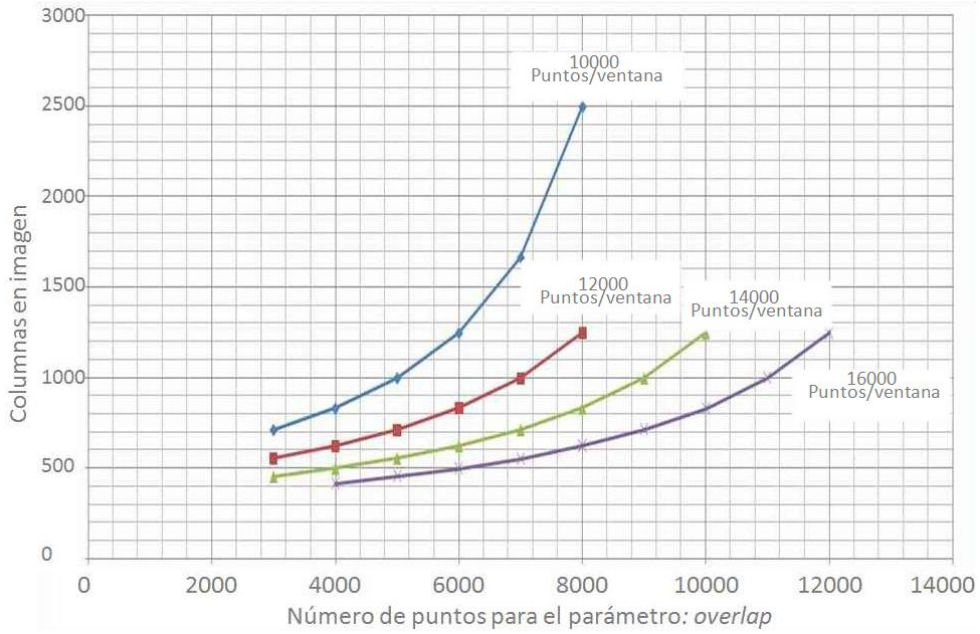
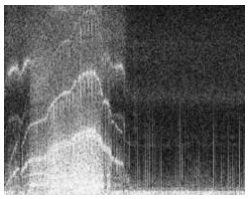
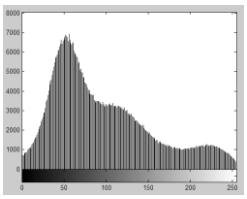


Figura 4.4. Relación entre los parámetros puntos por ventana y solapamiento.

El efecto principal de la ventana en la transformada de Fourier dependiente del tiempo es limitar la extensión de la secuencia que se va a transformar de manera que, las características espectrales, sean razonablemente estacionarias en el intervalo de duración de la ventana. Se puede comprobar que, para unos valores de ventana Hamming de 1000 puntos y un factor de solapamiento de 5000, se obtiene un espectrograma con buena definición (figura 4.3). No obstante se debe tener en cuenta que una mayor definición de imagen requiere un mayor tiempo de cálculo del espectrograma.

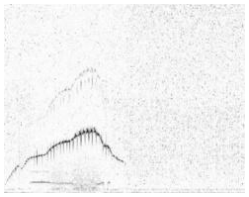


(a). Imagen de la componente roja.

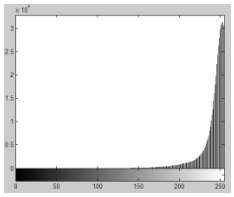


(b). Histograma de la figura 4.5(a).

Figura 4.5. Componente roja.

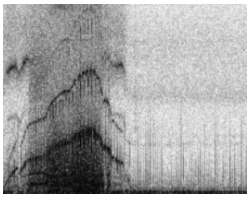


(a). Imagen de la componente verde.

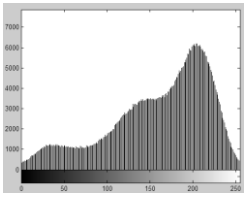


(b). Histograma de la figura 4.6(a).

Figura 4.6. Componente verde.



(a). Imagen de la componente azul.



(b). Histograma de la figura 4.7(a).

Figura 4.7. Componente azul.

Cuanto más rápido cambien las características de la señal más corta deberá ser la ventana. A medida que la longitud de la ventana decrece también la resolución en frecuencia decrece. Por otro lado, a medida que decrece la longitud de la ventana, aumenta la potencia de resolver cambios en el tiempo. Por tanto como se puede apreciar en la figura 4.4, aparece un compromiso en la selección de la longitud de la ventana, entre la resolución en el tiempo y en la frecuencia.

4.3.2 Selección de componente.

Una imagen digital es una función bidimensional con un valor tridimensional (4.1).

$$f: Z^2 \rightarrow Z^3 \qquad (4.1)$$

Donde cada Pixel de $X(x, y)$ se representa como un vector de tres componentes P=(I_1, I_2, I_3). Las imágenes que se tratarán pertenecen al espacio color RGB. En el espacio RGB, se considera la imagen como composición de mapas unidimensionales de valor I_1=*rojo* (4.2), I_2=*verde* (4.3), I_3=*azul* (4.4) (figuras 4.5, 4.6 y 4.7).

$$f_1: Z^2 \rightarrow Z: X = (x, y) \rightarrow I_1 \quad (4.2)$$

$$f_2: Z^2 \rightarrow Z: X = (x, y) \rightarrow I_2 \quad (4.3)$$

$$f_3: Z^2 \rightarrow Z: X = (x, y) \rightarrow I_3 \quad (4.4)$$

Se puede tratar cada mapa por separado y analizar cada uno de ellos como una imagen en blanco y negro. Este tipo de procesamiento se llama *tratamiento marginal*. Los pixeles de una imagen en blanco y negro pueden tomar valores que van desde el 0 al 255. El *histograma* [Cheng 2004] de una imagen digital es una función discreta (4.5) con niveles de gris en el rango [0, 255].

$$p(r_k) = n_k / N \quad (4.5)$$

Donde r_k es el *k-ésimo* nivel de gris, n_k es el número de pixeles de la imagen con ese nivel de gris y N el número total de pixeles de la imagen (k=0,…, 255). De forma general se puede decir que $p(r_k)$ da una idea acerca de la probabilidad que aparezca el nivel de gris r_k. La representación gráfica de esta función para todos los valores de k proporciona una descripción global de la apariencia de la imagen. Aunque el histograma no indica nada específico sobre el contenido de la imagen, su perfil proporciona sin duda información muy útil sobre la posibilidad de mejorar la imagen, entendiendo por mejora destacar las características de interés de la misma y no en el sentido usual visual. La segmentación de una imagen consiste en extraer propiedades o características comunes de alguna región de interés. En este caso la segmentación por histograma realizada en este trabajo consiste en retener aquellos pixeles que se encuentren en un cierto entorno de niveles de gris del histograma. Partiendo de una de una imagen RGB donde el color se codifica con 8 bits y la intensidad de cada color se mide según una escala que va de 0 a 255, el objetivo es conseguir una imagen binaria. Puesto que cada pixel está definido por tres dimensiones se puede encontrar cuál de las tres dimensiones es la que más

interesa analizando de forma independiente los histogramas de cada una de las tres dimensiones. Las figuras 4.5.a, 4.6.a y 4.7.a muestran las tres componentes de una imagen en color pero en su escala de grises, y las figuras 4.5.b, 4.6.b y 4.7.b sus respectivos histogramas. Se puede comprobar en cada histograma cómo la escala de grises va desde 0 hasta 255. Interesa por tanto la componente que tiene más *dispersión* de las tres para aplicar al método. La *dispersión* se mide por el número de niveles de grises que hay en la imagen. Esta métrica será máxima cuando el nivel de grises sea mínimo. Para ello es necesario calcular cada histograma y extraer de la imagen la componente con el mayor número de valores cercanos a 255.

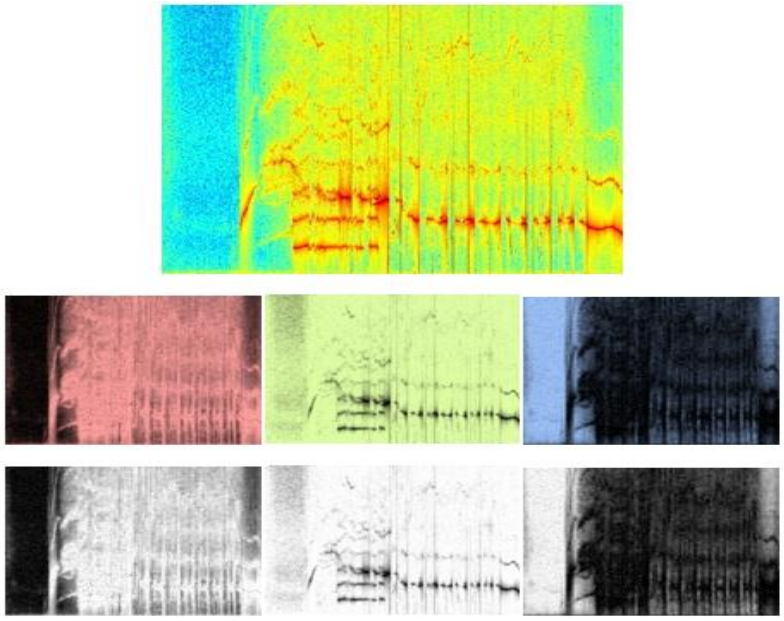


Figura 4.8. Detalle del espectrograma. La imagen está compuesta por las tres componentes (rojo, verde y azul). Debajo de cada una está la misma componente pero en su escala de grises. Se observa como la componente verde es la que tiene un mayor contraste.

Es decir, la componente a seleccionar será $I = max\{ I_1, I_2, I_3 \}$. La figura 4.8 muestra de forma más precisa las componentes del espectrograma. El siguiente paso del método será binarizar la componente resultante.

Binarización. La binarización de imágenes es una técnica que consiste en reducir el nivel de información de la imagen digital a solo dos valores el 0 (negro) y 1 (blanco). Se compara cada pixel con un determinado umbral donde los valores de los pixel por encima de ese umbral toman el valor 255 (blanco) o bien 0 (negro). La segmentación por umbral es una técnica utilizada ampliamente para clasificar

pixeles en una clase u otra. Para una imagen con objetos claros en el método del umbral se puede aislar fácilmente el objeto del fondo. En este trabajo se ha utilizado el método de *Otsu* [Otsu 1975] para determinar el umbral correcto. Este algoritmo consiste en obtener para todos los umbrales posibles, es decir tantos como niveles de gris tenga la imagen, un valor denominado σ_B^2. Este valor mide la diferencia entre los pixeles de la clase intensidad < umbral, y la clase restante intensidad > umbral, que representarían objeto y fondo respectivamente [Frucci 2008]. El umbral que maximice esta varianza entre clases es el óptimo para la binarización. La formulación de estos conceptos se puede ver en (4.6). Donde $\omega_B^2(t)$ es la varianza entre clases $i \leq t$ *e* $i > t$, $\omega(t)$ es la probabilidad de la clase $i \leq t$, $\mu(t)$ es la media de la clase $i \leq t$, n_i es el número de pixeles de intensidad i, N es el número total de pixeles de la imagen y μ_T la media total de la imagen. En este trabajo los umbrales aplicados han estado en el rango : $0.4 \leq \sigma_B^2 \leq 0.6$.

$$\omega_B^2(t) = \frac{(\mu_T * \omega(t) - \mu(t))^2}{\omega(1-\omega)} \qquad (4.6)$$

Donde $\omega(t) = \sum_{i=0}^{t} \frac{n_i}{N}$, $\mu(t) = \sum_{i=0}^{t} i \frac{n_i}{N}$, $\mu_T = \sum_{i=0}^{t_{max}} i \frac{n_i}{N}$

4.4 Segmentación de la imagen aplicada a BP1, BP2 y BP3.

Las imágenes digitales adquieren cada vez mayor importancia debido a los avances tecnológicos de los últimos años. Cada vez hay más demanda de entornos multimedia. Esto hace que los procedimientos de segmentación automáticos tengan un papel destacado. Este es un paso previo a sistemas más amplios y complejos como puede ser la recuperación de información de grandes bases de datos de imágenes [Fauqueur 2004, Fuertes 2001]. Es importante definir lo que implica la segmentación de imágenes y más concretamente que es una región en una imagen. La segmentación se puede definir como *el proceso de dividir una imagen en distintas regiones, pertenecientes a diferentes objetos, de forma que cada región cumpla cierto criterio de homogeneidad* [Cheng 2001]. Se puede definir el concepto de región como *un conjunto de pixeles semejantes y conectados.* La semejanza inducirá el tipo de homogeneidad que caracteriza a la región. A modo de ejemplo, si se habla de semejanza geométrica de objetos de una imagen, existen diferentes técnicas de segmentación y cada una de

ellas utiliza procedimientos propios. A continuación se mostrarán brevemente las técnicas de segmentación más habituales.

Las técnicas basadas en píxeles. Algunas de las técnicas que se enmarcan dentro de esta categoría constituyen las primeras propuestas [Sahoo 1988, Zbigniew 1985, Wojcik 1985], y que con el tiempo fueron evolucionando para aplicarse sobre histogramas de la imagen [Cheng 2003, Gillette 1995]. En muchas ocasiones, se utilizan los picos de estos histogramas para realizar un agrupamiento aunque en las propuestas más recientes se suelen aplicar redes neuronales. Las técnicas basadas en *umbralización* asumen que los píxeles cuyo valor (nivel de gris, color, textura,...) se encuentra dentro de un determinado rango pertenecen a la misma clase [Fan 2001]. Estas técnicas resultan útiles cuando las imágenes solo contienen dos componentes opuestos, por ejemplo al distinguir un objeto del fondo [Sahoo 1988]. Las primeras técnicas basadas en histogramas estaban pensadas para imágenes de niveles de gris, aunque hoy en día existen versiones para imágenes en color [Cheng 2000, Gillet 1995]. Dado que la construcción y manipulación de un histograma 3D puede resultar muy compleja y costosa, suele ser necesario realizar un pre-procesamiento para reducir la dimensionalidad del color (o del conjunto de características utilizado). Los algoritmos de agrupamiento o Clustering [Agarwal 20012, Dong 2000, Lew 2002, Li 2004, Mezhoud 2014] parten de un conjunto de vectores iniciales de características y van clasificando los píxeles de la imagen dentro de las categorías asociadas a dichos vectores. Cada pixel se asigna al grupo al que es más parecido, lo cual se traduce a asociarlo al grupo cuyo centro tiene menor distancia a él.

Las técnicas basadas en fronteras. Estas técnicas se caracterizan porque proporcionan contornos bien definidos, de forma general, cuando las regiones tienen características muy diferentes. Sin embargo esta condición no se suele dar en imágenes reales. Estas técnicas, según Davis [Davis 1975], se agrupan en dos categorías: técnicas locales y globales. Las técnicas locales de detección de fronteras suelen calcular el gradiente mediante máscaras diferenciales con el fin de detectar y seguir el contorno de las regiones. Las técnicas globales de detección de fronteras se basan en la búsqueda de heurísticas y en la programación dinámica o en campos de Markov. Suelen suavizar la información del tono y luego la combinan con la información de los contornos extraídos de los niveles de gris, obteniendo un mapa de fronteras que descarta los efectos derivados de la iluminación. Dentro de este tipo de métodos existe uno muy conocido dentro de la Morfología Matemática, el algoritmo Watershed. La paralelización de este método ha sido estudiado por

diferentes autores [Meijster 1995, Zhou 2004]. El algoritmo se ejecuta de forma paralela en varios procesadores de un mismo ordenador y con memoria compartida. Esto implica que, en un algoritmo paralelo, la información de las regiones y sus vecindades se comparte por los distintos procesadores.
Algunos autores asignan a estas técnicas una tercera clasificación, los modelos *deformables* que están basados en la definición de polígonos cuyos lados son splines (generalización de una curva) lo que permite obtener figuras más redondeadas.

Técnicas basadas en regiones. Las técnicas de segmentación de imagen mediante regiones incorporan información sobre la topología de la imagen. En la tabla 4.1 se muestran las diferentes características de las regiones y la forma en que han sido aplicadas en el análisis de los patrones BP1, BP2 y BP3. Ésta técnica ha sido la

Tipo de patrón	Area	Envolvente	Pendiente	Puntos extremos	Centroide	Ellipse equivalente	Ejes
Incremental BP1	•		•		•		•
Decremental BP2			•		•		•
Trapezoidal BP1-BP2-BP3	•	•		•	•	•	•
Lineal BP3	•		•		•		•

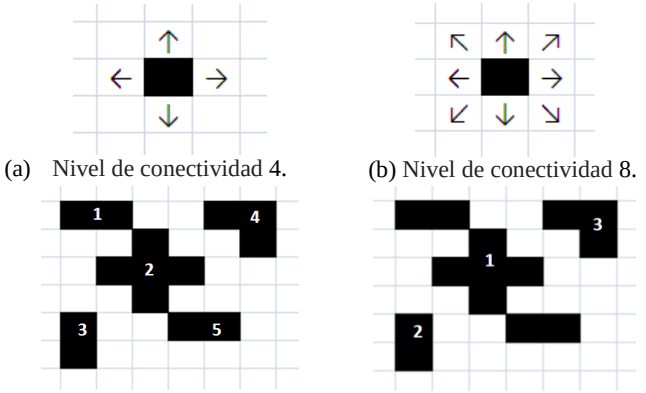
Tabla 4.1. Características topológicas de las regiones utilizadas para el análisis de los patrones.

utilizada en la Tesis. Toda región de una imagen puede ser identificada por unas características determinadas. En la sección 4.4.1 se muestra cada unas de ellas.

4.4.1 Etiquetado y análisis de características de regiones.

Una vez seleccionada la componente de la imagen que tiene más carga informativa el siguiente paso es etiquetar cada una de las regiones y analizar las características de cada una de ellas.

Etiquetado de regiones.
Este paso es previo al análisis en detalle de cada región. El etiquetado de regiones se basa en analizar cada pixel de la imagen para asignarlo a una región determinada. Para asignar un pixel a cada región hay que tener en cuenta los



(a) Nivel de conectividad 4. (b) Nivel de conectividad 8.

(b) Etiquetado de regiones nivel de conectividad 4. (d) Etiquetado de regiones nivel de conectividad 8.

Figura 4.9. Niveles de conectividad de vecinos.

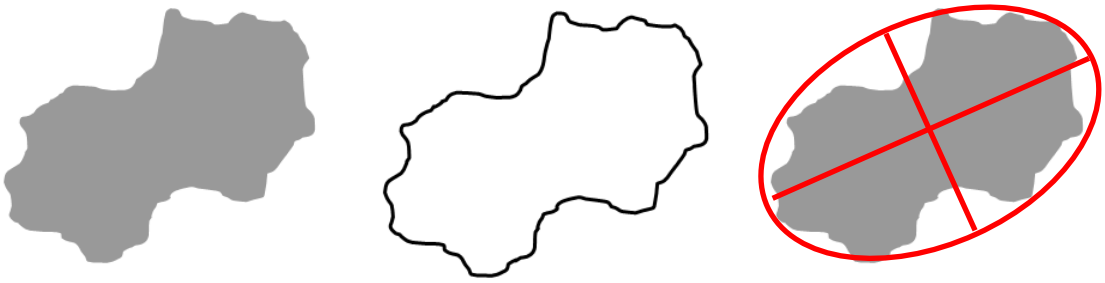
vecinos más cercanos a él. Para tal fin, se define el *nivel de conectividad* de vecinos que puede haber alrededor de un pixel, pudiendo considerarse un nivel 4 ó un nivel 8 de conectividad. La figura 4.9(a) muestra un pixel y las flechas de las casillas de su alrededor indican donde estarían situados los pixel vecinos para asignarlos a su misma región para un *nivel de conectividad* de 4. En la figura 4.9(c) se muestra un ejemplo de una imagen genérica cuando se ha implementado un *nivel de conectividad* 4 dando como resultado un total de cinco regiones. La figura 4.9(b) muestra un pixel y las flechas de las casillas de su alrededor indican donde estarían situados sus vecinos para asignarles a la misma región con un *nivel de conectividad* 8. En este caso, además de considerarse los píxeles superiores, inferiores y laterales como ocurre en el nivel de conectividad 4 se toman en cuenta los vecinos que están en las diagonales del pixel bajo estudio. La implementación del *nivel de conectividad* 8 para el ejemplo considerado anteriormente se muestra en la figura 4.9(d). Se puede comprobar cómo los pixeles vecinos en su diagonal han sido incluidos en una misma región. En este caso las regiones 1, 2 y 5 de la figura 4.9(c) se han fusionado en una sola región, la etiquetada como 1.

Análisis de características de regiones.

Las imágenes están constituidas por regiones o zonas que tienen características homogéneas y generalmente estas regiones se corresponden con objetos de la imagen. A continuación se analizan las características y los cálculos realizados en el algoritmo desarrollado.

Elipse equivalente. A partir de los ejes mayor y menor se puede calcular el área de la elipse equivalente $A_e(i)$ [Matos 2005]. Se trata de la elipse que inscribe a la región

bajo estudio. Este parámetro hace referencia a la angulosidad [Pan 2010] de la región que se está analizado.

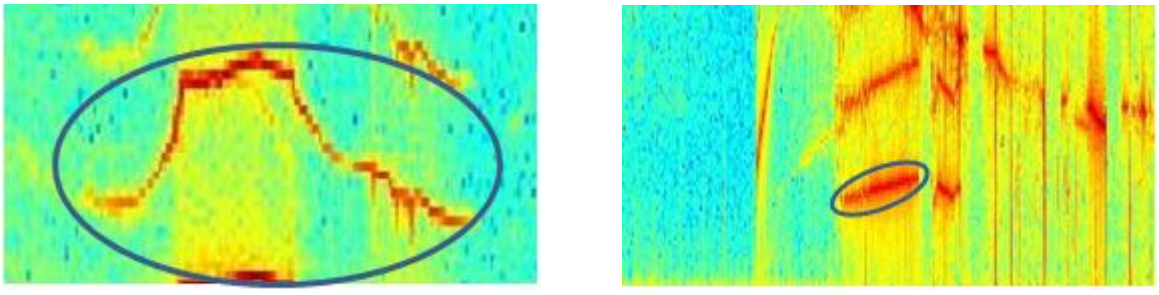


(a) Región con angulos. (b) Perímetro convexo. (c) Elipse que inscribe a la región.

Figura 4.10. Definición de la elipse equivalente.

La angulosidad indica el grado de redondez que tiene la región que se estudia. Para medir a angulosidad se necesitan dos medidas, el $perímetro_{convexo}$ y el $perímetro_{elipse.}$ El $perímetro_{convexo}$ es el límite del polígono que envuelve a la región, véase la figura 4.10(b). Y el $perímetro_{elipse}$ es el perímetro de la elipse equivalente que envuelve a la forma de la figura 4.10(c). La relación entre estas dos medidas proporciona la angulosidad de la región y viene dada por (4.7).

$$Angulosidad = \left(\frac{perímetro_{convexo}}{perímetro_{elipse}}\right)^2 \qquad (4.7)$$



(a) Elipse para patrón trapezoidal (b) Elipse para patrón creciente.

Figura 4.11. Elipses para los patrones trapezoidal y creciente.

No obstante existe una relación entre el número de pixeles que componen la región y el área de la elipse que la inscribe. Cuanto más se ajuste la elipse al área de la

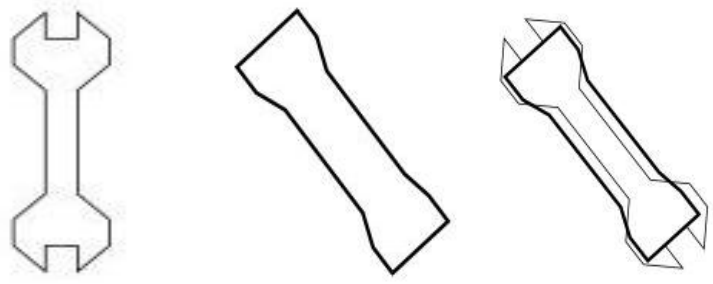
región bajo estudio, más equivalentes serán las dos superficies. Con este planteamiento se puede establecer una relación entre las dos áreas dada que viene expresada por (4.8).

$$Area_{elipse} > \mu\, Area_{regi\acute{o}n} \qquad (4.8)$$

Se puede fijar un umbral que sea la frontera que decida si una región se ajusta al patrón trapezoidal (figura 4.11(a)) y en caso contrario determinar que la región en curso cumple otro patrón diferente. En otras palabras comprobar si se cumple (4.8). El modelo fue entrenado con diferentes valores de μ siendo 5 el que mejores resultados obtuvo.

Envolvente. Para el reconocimiento de formas se utilizan diferentes métodos como pueden ser redes neurales [Pao 1989], métodos estadísticos [Duda 1972], métodos sintácticos [Gonzalez 1978] o bien métodos indexados [Brin 1995, Portegys 1995]. Dentro de este campo existe un campo de estudio que investiga el *reconocimiento espacial de objetos*.

Para el análisis particular de este trabajo la *envolvente* puede ser de gran utilidad. El objetivo del reconocimiento espacial de objetos es encontrar un ajuste entre dos modelos geométricos. El ajuste viene determinado por una función o bien por una transformación geométrica de los objetos bajo estudio [Hagedoorn 2000] y su comparación con un modelo o modelos determinados. En este contexto, se hablaría por tanto del *reconocimiento geométrico de patrones*. Éste reconocimiento a su vez se puede realizar cuando éste mismo sea la combinación de varios modelos como muestra la figura 4.12.



(a). Objeto A. (b). Objeto B. (c). Ajuste.

Figura 4.12. Envolvente. Ajuste de modelos afines

Además de la forma geométrica, también se puede analizar en términos de áreas, donde el análisis de la envolvente es una característica que sirve de apoyo para la clasificación. Relacionando esta métrica con el área de una región similar se puede determinar una *medida de relación (MR)*. Si además se dispone de un histórico almacenado de modelos se pueden contrastar las medidas obtenidas del objeto en curso con las almacenadas [Patel 2013]. De esta forma se puede asignar con cierta exactitud el patrón al que pertenece la región bajo estudio región.

$$MR = \frac{Área\ Región}{Área\ Envolvente} \tag{4.9}$$

Para las pruebas realizadas en este trabajo se usaron los valores que siguen a continuación. Para los casos de patrón creciente o decreciente en frecuencia se tiene que $MR<0.1$ y para patrón lineal y trapezoidal $MR\geq0.1$. Esta característica según se ha apuntado anteriormente se puede apoyar en la característica de los *puntos extremos* (figura 4.13) y de esta forma hacer una clasificación más rigurosa.

Puntos Extremos: los puntos extremos hacen referencia a las coordenadas de los píxeles más extremos de cada región. Para cada región que se encuentra en la imagen bajo estudio debe haber ocho puntos extremos. Cada punto viene definido por las coordenadas (x,y) de la imagen. Esta característica es muy útil para detectar por ejemplo el patrón trapezoidal, duración temporal de la señal o bien su amplitud en frecuencia. La figura 4.13 muestra una imagen de 155x255 pixeles con una región que pertenece a un patrón trapezoidal. En la región se han marcado las ocho coordenadas de los ocho pixeles más extremos. A partir del cálculo del espectrograma, es posible conocer el rango de frecuencias y el tiempo de la imagen. Con estos datos se puede calcular la frecuencia y tiempo que se corresponde con cada coordenada de cualquier región.

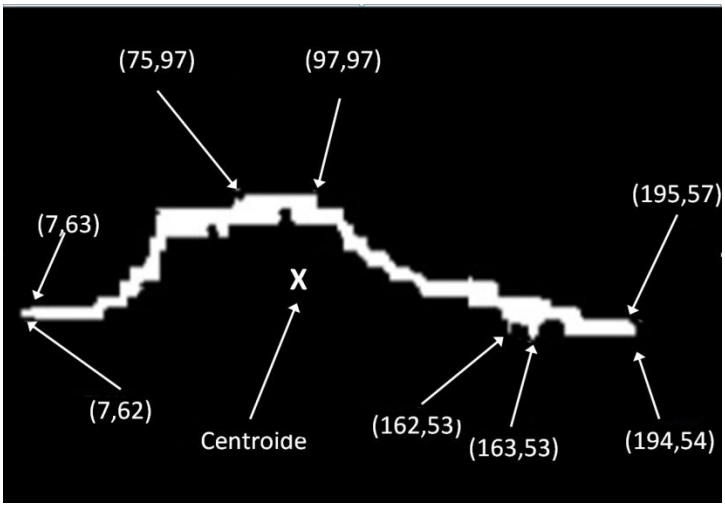


Figura 4.13. Puntos extremos de región.

Sea *PE* una matriz de dimensión 8x2 donde *PE*(*i,1*) son los valores para el eje de abscisas y *PE*(*i,2*) son los valores para el eje de ordenadas de la región *R*. El valor de comienzo temporal de la señal será *min*(*PE*(*i,1*)) y el valor final temporal será *max*(*PE*(*i,1*)). Lo mismo ocurre para el comienzo y final de la frecuencia: el comienzo del intervalo en frecuencia será *min*(*PE*(*i,2*)) y su final *max*(*PE*(*i,2*)). Obtenidos los valores solo queda calcular sus valores temporales y frecuenciales a partir de los resultados obtenidos en el cálculo de su espectrograma.

El cálculo del patrón trapezoidal es un poco más elaborado. Sean (R_1, R_2, R_3) tres regiones conectadas que forman una región mayor llamada *R*. Es decir (R_1, R_2, R_3) $\in R$. Para que el patrón trapezoidal se confirme deben cumplirse (4.10), (4.11) y (4.12) siendo *PE*(*mxn*) la matriz de puntos extremos de la región *R*. Donde *F* y *T* son los vectores de frecuencia y tiempo respectivamente.

$$R_1 = PE(x,y)F(x,y) < PE(x,y+1)\,F(x,y+1) \tag{4.10}$$

$$R_2 = PE(x,y)F(x,y) \in \left[\left(\delta + PE(x,y+1)\right)F(x,y), (PE(x,y+1) - \delta)F(x,y)\right] \tag{4.11}$$

$$R_3 = PE(x,y)F(x,y) > PE(x,y+1)F(x,y) \tag{4.12}$$

Para las tres regiones *y=1..n, x=1* y δ=3.

Una vez calculados los puntos extremos de las regiones, se pueden calcular los grados que forma esa subregión con el eje de abscisas. Por ejemplo para calcular el grado de inclinación de R_1 se procede de la siguiente forma:

sean $PE(x_1,y_1)$ y $PE(x_2,y_2)$ los puntos extremos de la región R_1. Para que R_1 sea una región creciente en frecuencia debe cumplirse (4.13) y que $\Phi \in [5^0,90^0]$.

$$\Phi_1 = arctang\big(PE(1,y_2) - PE(x_2,1), PE(1,y_1) - PE(x_1,1)\big) * 180/\pi \qquad (4.13)$$

El ángulo Φ se calcularía de la misma forma para las regiones restantes pero con sus respectivos puntos extremos. Al final los diferentes ángulos deberían verificar (4.14).

$$\Phi_1 \in [5^0,90^0].$$

$$\Phi_2 \in [-5^0,+5^0]. \qquad (4.14)$$

$$\Phi_3 \in [-90^0,-5^0].$$

Área de la región. Esta característica se refiere al número de pixeles que contiene la región. Si $g_i(x,y)$ es una función que es uno si el pixel pertenece a la región y cero en caso contrario, el área vendrá dado por (4.15).

$$A(i) = \sum_{x=1}^{N}\sum_{y=1}^{M} g_i(x,y) \qquad (4.15)$$

El siguiente paso del método que se propone es eliminar todas aquellas regiones que superan un umbral de superficie. En la imagen en blanco y negro que se analizará, habrá una señal concreta con un área determinada pero también habrá regiones que son objetivo de estudio y que son más pequeñas en superficie. Por tanto en primer lugar habrá que eliminar la mayor parte de esta señal con el objetivo de limpiar lo más posible la imagen. En este trabajo fueron eliminadas todas las áreas superiores a 250 pixeles que tuvieran un *nivel de conectividad* de 8.

Ejes: Esta característica se refiere a los ejes mayor y menor de la elipse que inscribe a la región bajo estudio. Cada objeto de una imagen está contenido en una elipse imaginaria. En la Figura 4.14 se puede apreciar una región que está dentro

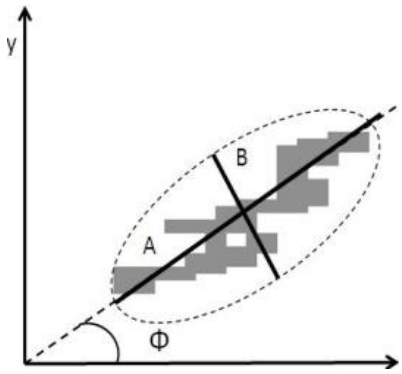


Figura 4.14. Pendiente de una región.

de una elipse con su eje mayor A y el eje menor B. Para calcular los ejes en primer lugar se debe definir el momento de una imagen [Hosny 2010]. De forma breve, La geometría de una región plana se basa en el tamaño, la posición, la orientación y la forma. Todas estas medidas están relacionadas con una familia de parámetros llamada momentos. De tal forma que existe el momento cero, primero y segundo. Si se considera que la región está definida por (4.15), en general se podrá calcular el segundo momento. Para x, y y xy se define mediante (4.16), (4.17) y (4.18) respectivamente.

$$sMX = \sum_x \sum_y x^2 g_i(x,y) \tag{4.16}$$

$$sMY = \sum_x \sum_y y^2 g_i(x,y) \tag{4.17}$$

$$sMXY = \sum_x \sum_y xy g_i(x,y) \tag{4.18}$$

Una vez calculado el segundo momento se puede calcular el eje mayor mediante (4.19) y el eje menor mediante (4.20).

$$Meje = 2\sqrt{2}sqrt((sMX - sMY) + \sqrt{(sMX - sMY)^2 + 4sMXY^2}) \tag{4.19}$$

$$meje = 2\sqrt{2}sqrt((sMX - sMY) - \sqrt{(sMX - sMY)^2 + 4sMXY^2}) \tag{4.20}$$

Estos ejes tienen doble utilidad. Como se verá más adelante el eje mayor se toma como referencia para la característica *pendiente* de la tabla 4.1. En segundo lugar la utilidad que aporta a esta metodología es que los ejes sirven para calcular la *relación de aspecto* (*RA*) definiéndose según (4.21).

$$RA=ejeMayor/ejeMenor \tag{4.21}$$

El significado geométrico de *RA* es que cuanto más alejada esté de 1 indica una forma más regular de la región.

Pendiente: Esta característica indicará la inclinación que tiene la región respecto al eje horizontal. Se toma como referencia el eje mayor de la elipse que inscribe la región. La figura 4.14 muestra una figura inscrita en una elipse donde su eje mayor tiene un ángulo θ respecto al eje de abscisas.

$$\theta = \frac{180}{\pi}\ arctan\frac{A}{B} \tag{4.22}$$

El ángulo θ está definido por la expresión (4.22). Donde A y B se expresan mediante (4.23) y (4.24) respectivamente si $sMY>sMX$, o bien (4.25) y (4.26) si $sMY<sMX$.

$$A = (sMY - sMX) + \sqrt{(sMY - sMX)^2 + 4sMXY^2} \tag{4.23}$$

$$B = 2sMXY \tag{4.24}$$

$$A = 2sMXY \tag{4.25}$$

$$A = (sMY - sMX) + \sqrt{(sMX - sMY)^2 + 4sMXY^2} \tag{4.26}$$

Una pendiente es negativa cuando su ángulo es obtuso y positiva si su ángulo es agudo. Mediante esta característica se puede evaluar si una señal es creciente cuando su pendiente es positiva es decir $\theta \in [5^0,90^0]$, o por el contrario es

decreciente si su pendiente es negativa $\theta \in [-90^0, -5^0]$. Para definir una señal lineal se ha tomado como referencia que $\theta \in [-5^0, 5^0]$.

En los espectrogramas analizados y de forma general aparecen con cierta frecuencia regiones de alta intensidad y que son perpendiculares al eje de abscisas. Suelen tener un área mayor de 6 pixeles y un eje menor de alrededor 2 pixeles. Estas zonas deben ser eliminadas teniendo en cuenta las tres características señaladas: verticalidad, área y tamaño del eje menor.

Centroide: Esta característica es de gran utilidad para el cálculo de armónicos. Para dicho fin la característica que se va a utilizar es el centroide de la región, véase la ecuación (4.27). El centroide proporciona las coordenadas del centro de masas (CM) de la región. En ocasiones este centro no estará dentro del perímetro de la región puesto que se refiere al centro de su envolvente. Se representa por $(\hat{x}_i, \hat{y}_i)$ (4.27). La coordenada $\hat{y}_i$ debe ser normalizada para poder operar con ella. El motivo es que el origen de coordenadas de la imagen no está en el vértice inferior izquierda, el margen de coordenadas en una imagen está situado en el vértice superior izquierda.

$$\hat{x}_i = \frac{\sum_{x=1}^{N} \sum_{y=1}^{M} x g_i(x,y)}{A(i)}$$
$$\hat{y}_i = \frac{\sum_{x=1}^{N} \sum_{y=1}^{M} y g_i(x,y)}{A(i)} \qquad (4.27)$$

$$f_2 = n f_1 , \qquad n \in \mathbb{N} , \qquad n = 2,3,4 \ldots . m, \qquad m \in \mathbb{N} \qquad (4.28)$$

$$c_2 = c_1 , \qquad (4.29)$$

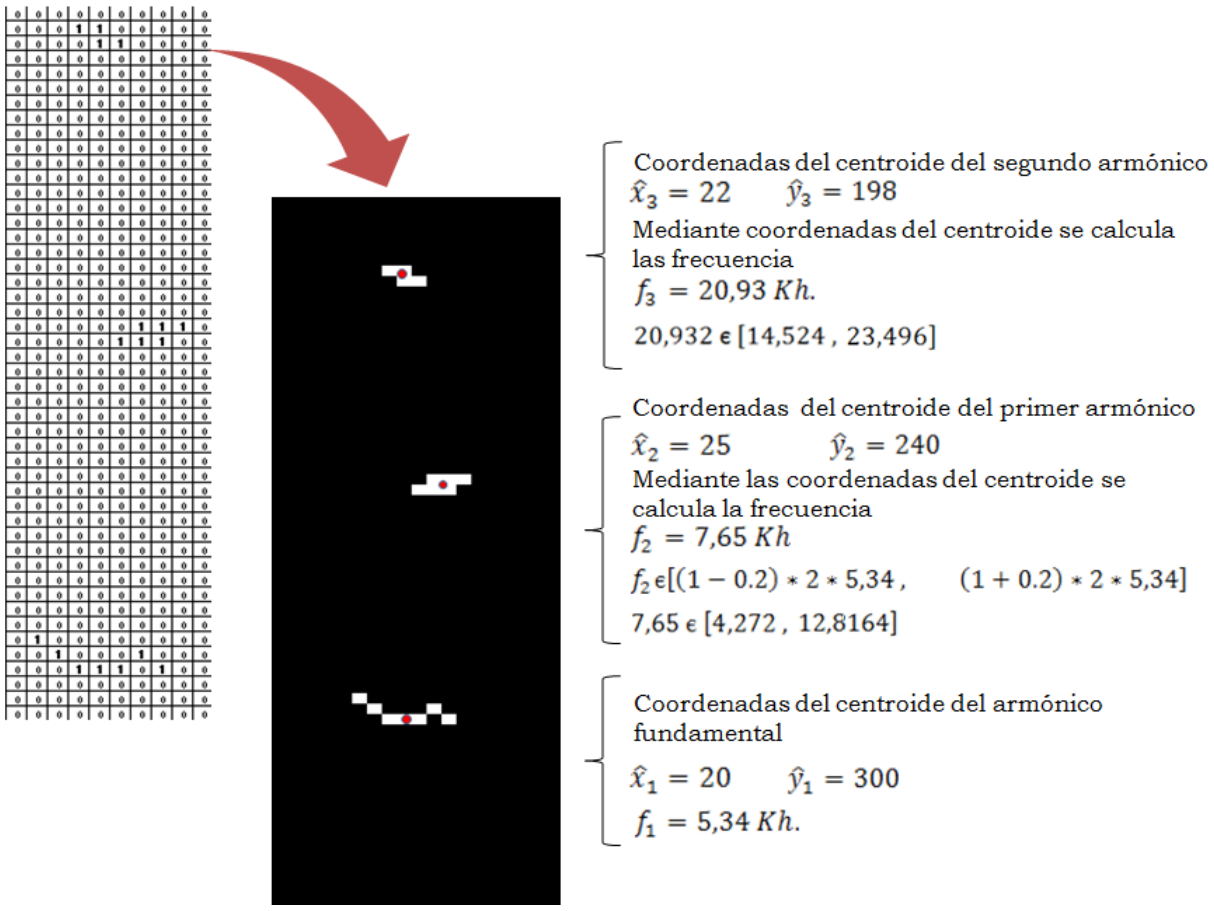


Figura 4.15. Aumento del área de interés para el cálculo de armónicos. Para el primer armónico se utiliza la expresión (4.28) y para el segundo y los sucesivos armónicos se utiliza la expresión (4.30).

Los armónicos se presentan en valores de frecuencia múltiplos del fundamental. Si se dispone de una imagen $I\,[f,\ c]$ y se ha detectado una región $R_1 \in I$ con su centroide situado en $[c_1, f_1]$. El cálculo de las coordenadas $[c_2, f_2]$ del segundo armónico viene determinadas por:

$$f_2 \in [(1-r)nf_1\ ,(1+r)nf_1] \quad r \in \mathbb{R}\ , n \in \mathbb{N} \tag{4.30}$$

Donde $n = 2{,}3{,}4..$ y $0 \leq r < 1$

$$c_2 \in [(1-g)c_1\ ,(1+g)c_1] \quad g \in \mathbb{R}$$
$$0 \leq g < 1 \tag{4.31}$$

Para que una segunda región $R_2 \in I$ con centroide en $[f_2,c_2]$ sea armónico de R_1 se debería cumplir (4.28). En ocasiones (4.28) no se cumple con exactitud porque los armónicos van perdiendo intensidad a medida que su frecuencia se separa de la frecuencia del fundamental. Esto origina que la región pierda definición y su centroide también se desvíe de su teórica posición referente al centroide correspondiente a la frecuencia del fundamental. Para compensar esta desviación se debe ajustar (4.28) y tener en cuenta cierto umbral. En esta metodología se redefine (4.28) como (4.30). La columna del centroide de R_2 se hace en un intervalo $[(1-r)nc_1\,,(1+r)nc_1]$. Para la metodología propuesta se aplicó un valor de r =0.2. La columna donde se sitúa el centroide de R_2 también sufre desviación. Al igual que en el caso anterior (4.29) tuvo que ser redefinida como (4.31). También en este caso para la metodología que se propone se aplicó un valor de g=0.3. Los valores de r y g se seleccionaron experimentalmente a partir de los mejores resultados del entrenamiento del modelo. La fórmula (4.30) varía a partir del tercer armónico.

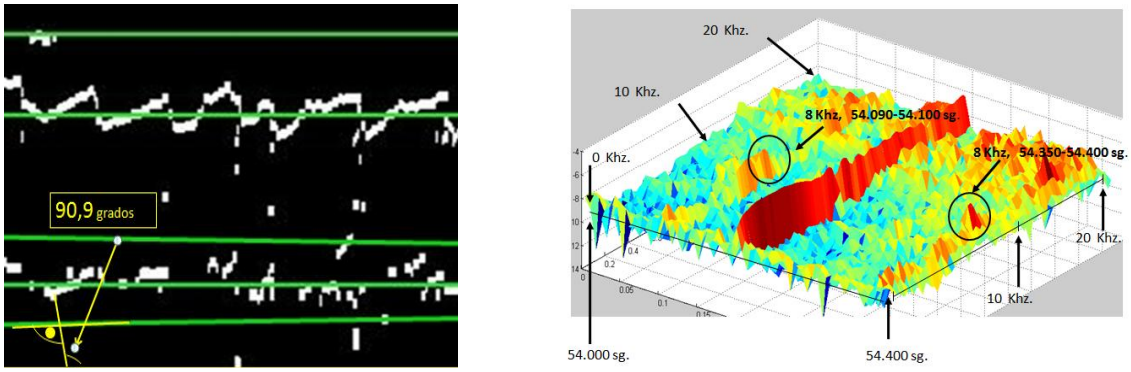
$$f_i \in [(1-r)n(f_1+f_{i-1})\,,(1+r)n(f_1+f_{i-1})] \quad r \in \mathbb{R}\,, n \in \mathbb{N} \qquad (4.32)$$

Donde $n = 2,3,4..,\ 0 \le r < 1, i > 2, i \in \mathbb{N}$

Los cálculos deben realizarse siguiendo la expresión (4.32). La figura 4.15 muestra parte de los cálculos a realizar para localizar armónicos. Estos cálculos son necesarios pero no suficientes, habría que añadir el cálculo de los intervalos temporales.

4.4.2 Automatización de la búsqueda de patrones.

El primer objetivo del proceso era analizar las técnicas de segmentación más acordes para el desarrollo de la metodología y el segundo objetivo era implementar la metodología. Para desarrollar el primer objetivo se diseñó un primer modelo basado fundamentalmente en la transformada de Hough que pertenece al tipo de segmentación basado en fronteras. La transformada de Hough es una herramienta que permite detectar curvas en una imagen. Es una técnica muy robusta frente al ruido y a la existencia de huecos en la frontera del objeto. A la hora de aplicar la transformada de Hough a una imagen es necesario obtener en primer lugar una imagen binaria de los píxeles que forman parte de la frontera del objeto. El objetivo de la transformada de Hough es encontrar puntos alineados que puedan existir en



(a) (b)

Figura 4.16. Transformada de Hough y detalle de la descarga 79455. En (a) se muestra un cuadrante de una imagen binarizada donde se han trazado rectas paralelas que pasan por diferentes picos. En (b) se muestra el nivel de detalle para la descarga 79455.

la imagen, es decir, puntos que satisfagan la ecuación de la recta, para distintos valores de ρ y θ, (la ecuación de la recta en forma polar viene dada por: $\rho = x \cdot \cos \theta$).

Algoritmo 4.1: Segmentación mediante la transformada de Hough
Datos: ficheros de descargas.

```
N ← numero de descargas;          /*  descarga a analizar  */
for i = 1 to  N
    I=leerDescarga();
    Esp=clcEspectro(I);           /*  Cálculo del espectrograma de la descarga  */
    Im=recorta(Esp);              /*  Recortar imagen */
    Im2=ruido(Im);                /*  Convertir espectrograma a escala de grises  */
    Bw=bin(Im2);                  /*  Binarizar la imagen mediante el método Otsu  */
    Bw2=rellena(Bw);              /*  Relleno de huecos (apertura y cierre)  */
    Re=realza(Bw2);               /*  Aumento de las líneas horizontales mediante máscara  */
    Hg=matrizHough(Re);           /*  Cálculo de la matriz de Hough  */
    PcHg=picos(Hg);               /*  Cálculo de los picos de Hough  */
end
```

Algoritmo 4.1. Segmentación mediante la transformada de Hough.

Por tanto hay que realizar una transformación entre el plano imagen (coordenadas x-y) y el plano o espacio de parámetros (ρ,θ). La base de este algoritmo es trazar rectas que pasan por diferentes picos. Una vez trazadas las rectas se puede calcular la distancia y el ángulo que forman estas rectas respecto al eje de coordenadas. En [Pajares 2007] se puede ver la aplicación de este método.

La figura 4.17 muestra el diagrama de flujo del primer modelo y el algoritmo 4.1 detalla los pasos seguidos para implementarlo. El segundo modelo

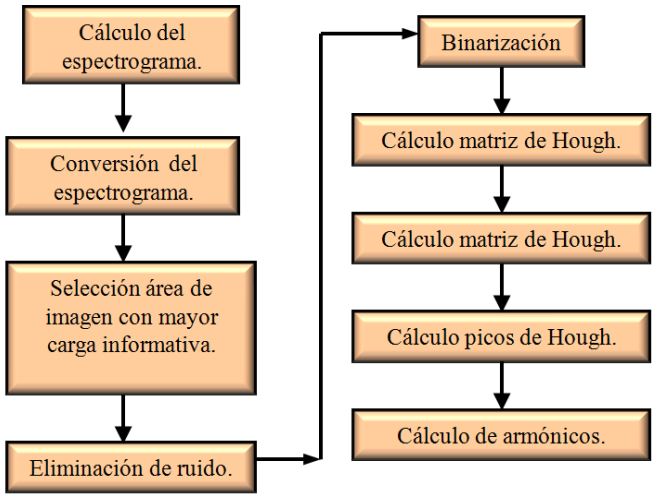


Figura 4.17. Diagrama de flujo del primer modelo. Cálculo de armónicos.

implementa la técnica de segmentación de regiones. Mediante esta técnica el sistema es capaz de descubrir más patrones de armónicos y por tanto se trata de una versión superior al primer modelo. Para la detección de armónicos es necesario analizar todas las características de regiones mostradas en el apartado 4.3.2. El flujo mostrado en la siguiente figura 4.18 es el codificado y posteriormente paralelizado.

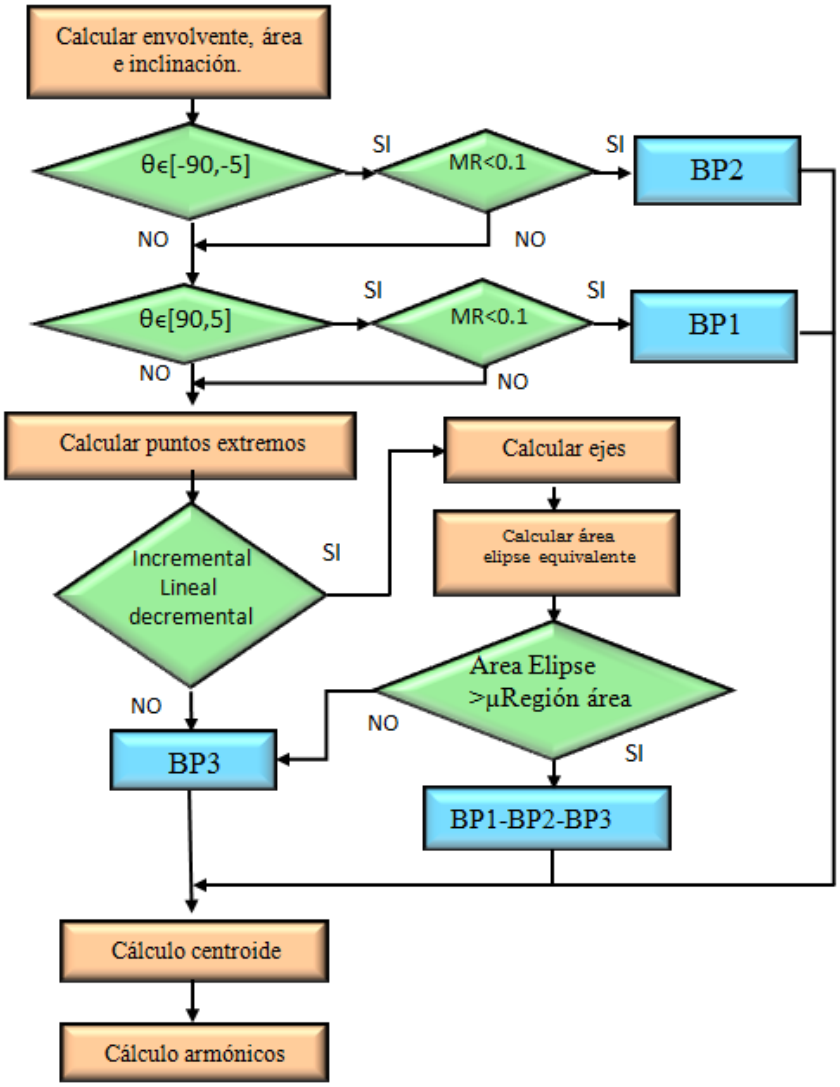


Figura 4.18. Diagrama de flujo del algoritmo de segmentación de regiones.

4.5 Paradigma SPMD aplicado al análisis de patrones BP1, BP2 y BP3.

Esta sección se centra en la paralelización del segundo modelo comentado anteriormente correspondiente a la detección de armónicos utilizando la técnica de segmentación de regiones. Se trata de una tarea con un gran volumen de datos a analizar que hay que cargar desde la unidad de disco y que posteriormente desencadenan otras tareas con un gran coste computacional. En esta situación, la paralelización del código de un entorno de supercomputación se hace

obligatoriamente necesaria. En este contexto el estudio se centra en las fases que son necesarias implementar para poder realizar una computación paralela eficiente. Los conceptos vistos en la sección 4.4 serán de gran utilidad para el trabajo que sigue a continuación.

4.5.1 Balanceo de la carga de trabajo.

El balanceo de carga es una técnica utilizada en sistemas distribuidos para reducir el tiempo de ejecución de las tareas. Son estrategias de distribución de las tareas entre los diferentes *workers* que componen el sistema distribuido. Los beneficios de su utilización en sistemas muti-agente y grid (anglicismo de red de computación) han sido ampliamente estudiados [Leinberger 2000, Yagoubi 2010] En el capítulo 2 de esta Tesis se hizo especial énfasis en los modelos de carga de trabajo en entornos paralelos. A continuación se profundizará más sobre estos modelos. El motivo de analizar los modelos de carga se debe a la diversidad de tamaños de ficheros que tuvieron que ser tratados. Recordar tan solo que el volumen de información analizada por el modelo superó los 6 Tb de espacio en disco.

Existen dos formas comunes de utilizar el modelo de carga registrado en los logs (anotaciones del sistema) del sistema para analizar o evaluar un nuevo diseño sintético del modelo. Por un lado se pueden utilizar directamente los datos grabados [Calzarossa 1995] para ejecutar una simulación del modelo o bien crear un nuevo modelo y utilizarlo para simulación [Calzarossa 1993]. Detalles como pueden ser el sistema compartido, diferentes configuraciones hardware, etc., son solo algunos de los motivos que pueden hacer dudar sobre si un modelo se puede extrapolar al resto de sistemas o no. Los siguientes puntos deben servir de ayuda para modelar una carga de trabajo.

- El diseñador del modelo debe tener pleno conocimiento de la carga de trabajo.
- Posibilidad de cambiar parámetros de la carga de manera dinámica.
- Políticas y limitaciones del sistema.
- Registros de logs erróneos. Como pueden ser las trazas de trabajos eliminados del sistema por exceder los recursos del sistema.

El procesamiento de las descargas a analizar consta de diferentes etapas. En la figura 4.19 se puede observar el progreso temporal del proceso.

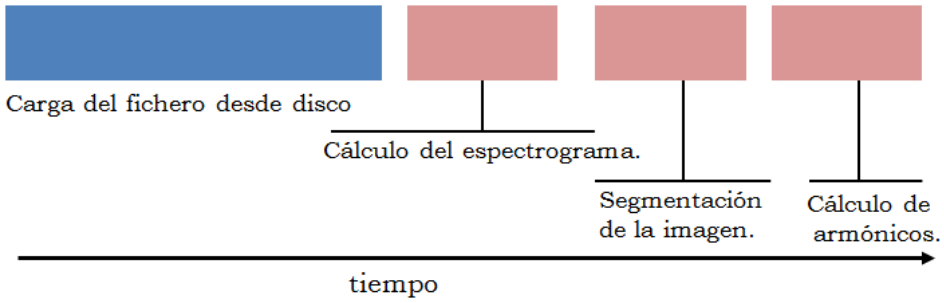


Figura 4.19. Evolución temporal del proceso de análisis de una descarga.

Anteriormente se ha comentado que los ficheros a analizar tienen diferentes tamaños con un rango que varía desde los 20 Mb hasta los 140 Mb. La figura 4.19 muestra que no todas las fases del procesamiento de la imagen tienen los mismos requerimientos de recursos. La primera etapa, en la carga del fichero, es necesario un mayor recurso de memoria pero ésta debe ir conjuntamente con la fase del cálculo del espectrograma que requiere mayor recurso de CPU. Sin embargo, las dos últimas etapas requieren conjuntamente mayor potencia de cálculo. Con este escenario parece lógico separar las dos primeras fases (carga del fichero y cálculo de los espectrogramas) de las dos últimas. Para conseguir este objetivo fue necesario definir una *estrategia de planificación* que se detalla en la sección 4.6.

Aunque toda la gestión de los *workers* se hace mediante la consola de creación de *workers* vista en la figura 2.18, es posible conocer el nodo al que pertenece cada *worker* mediante la función Matlab *system*(*hostname*).

Conociendo el origen de cada uno es inmediato crear una tabla que gestione la asignación de tareas en función del nodo al que esté asignado el *worker*. No se trata de asignar de forma exclusiva los *workers* del nodo 3 a la carga en memoria del fichero y el resto de *workers* al cálculo. El objetivo es que todos trabajen de forma colaborativa pero se prime a los *workers* del nodo 3 en todo lo referente a tareas de carga en memoria.

4.5.2 Roles de los *workers*.

El modelo del sistema que se ha diseñado e implementado utiliza los paradigmas de programación paralela vistos en el capítulo dos. Debido al

funcionamiento del bloque **spmd** de Matlab, se hace necesario realizar una combinación especial de los *workers*. El diseño plantea que uno de los *workers* actúe como gestor de los trabajos del resto de *workers*. Dentro del bloque, hace las funciones de gestor del trabajo. En los capítulos siguientes, se verá que esta estructura cambia cuando se tratan variables multidimensionales distribuidas. Con este planteamiento se pueden distinguir dos tipos de roles de los *workers*:

- Se define la figura del *worker gestor*. Este es el que asigna las tareas de cálculo de la imagen y en caso de necesidad colabora en la segmentación de la imagen. Gestiona dos tipos de tablas. La primera de ellas contiene la lista con los ficheros a procesar. Asigna a cada uno de los *workers* restantes los ficheros que deben ser procesados teniendo en cuenta la especialidad que se ha definido por cada *worker*. La segunda tabla contiene el estado actualizado de cada uno de los *workers*. Como datos importantes, la tabla tiene el nodo al que está asignado, la tarea en curso y si está ocupado o libre. Este *worker* gestor es el único que se define en el nodo Front-end.
- El resto de *workers* están repartidos en dos únicas tareas especiales que son las vistas hasta ahora, segmentación de la imagen y cálculo de armónicos. Como ya se ha comentado, estos *workers* solo estarán en los nodos 0, 1, 2 y 3 del clúster DIA. La comunicación se realiza entre cada *worker* y el *worker* gestor.

En un estado avanzado del proceso, es posible que los *workers* del nodo 3 tuvieran que ejecutar parte de las dos últimas fases del proceso de segmentación para evitar de esta forma tiempos inactivos de CPU. El algoritmo 4.2 muestra la implementación de la estrategia JSS-WDA que se detalla en la sección 4.6. El bloque **spmd** de Matlab es donde se concentra toda la actividad paralela. El gestor interno del entorno paralelo de Matlab siempre asigna el último *worker* de la agrupación paralela al nodo Front-end. Esta configuración hace más cómodo el control de todos los *workers* del proceso. En el algoritmo se pueden observar dos partes principales dentro del bloque **spmd.** La primera es en la que trabaja el *worker* gestor y que comienza por *if labindex=labPpal.* La segunda es la que comienza por *if labindex <> labPpal* y es en la que trabajan el resto de *workers.*

Algoritmo 4.2: Algoritmo de cálculo de armónicos bajo la estrategia JSS-WDA.

```
Datos: Conjunto de descargas a analizar
Resultado: detección de presencia armónicos
N ← número de descargas
ruta ← ruta de las descargas
spmd                                                    /*  bloque pool paralelo de Matlab */
  labPpal=numlabs;                                      /*  lectura del número de workers */
  if labindex=labPpal                                   /*  situación en worker del front-end */
   listaHost ← recibe_Host( worker1,……; workern-1)      /*  crear lista donde esta alojado cada */
  listaDescargas ← lee_rutaDescargas(ruta)              /*  wroker genera la lista de descargas a analizar */
    for i=1 to labPpal-1                                /*   a analizar */
          /*en primera iteración todos  los worker tratan ficheros de descargas */
           enviaNombreDescarga(listaDescargas(i)) → worker1,……,workern-1 ;
    end
     lista_estado ← recibe_Workers(worker1,……,workern-1)  /* estado finalizado workers */
       if N= labPpal-1
         envía_fin(1, worker1,……; workern-1)
       else
         envía_fin(0, worker1,……; workern-1)
       end
     for i= labPpal to N    /*  se continúa con las descargas donde termino la primera iteración*/
                            /* la rutina selecc_Worker_Descarga realiza selección del */
        numWorker ← devuelve_Num( listaHost )       /*  primer worker libre */
        if numWorker ∈ host3
           selec_nombreDescarga(listaDescargas(i )) → workernumWorker
        else
           segmenta_imagen → workernumWorker
        end
        lista_estado ← recibe_Workers(worker1,……,workern-1)
        envía_fin(0, worker1,……; workern-1)
     end
      / * En este paso ya se han terminado todos los ficheros de descargas. */
      /*  repasar el estado de todos los workers */
      for i=1 to length(lista_estado)
              lista_estado ← recibe_Workers(worker1,……,workern-1)
              envía_fin(1, worker1,……; workern-1)
      end
end
if labindex < > labPpal                                 /* situado  en el resto de workers */
  labindex, nombreHost → envía_Host(workern )           /* se envía el nombre del nodo al que pertenece */
  descarga ← recibe_nombFichero(workern)                /* primera iteracción del worker ppal.*/
  terminado  → envía_Host(workern )                     /* envía estado de proceso de la descarga */
  fin ← recibe_Fin(workern)
  while not fin
                                                        /* primera iteración del worker ppal.*/
       (descarga/segmenta) ← recibe_nombFichero(workern)
        terminado ← ejecuta(descarga/imagen )
        terminado  → envía_Host(workern )               /* envía estado de proceso de la descarga */
        fin ← recibe_Fin(workern)
  end
end
end          /* final del bloque paralelo spmd */
```

Algoritmo 4.2. Cálculo de armónicos bajo la estrategia JSS-WDA.

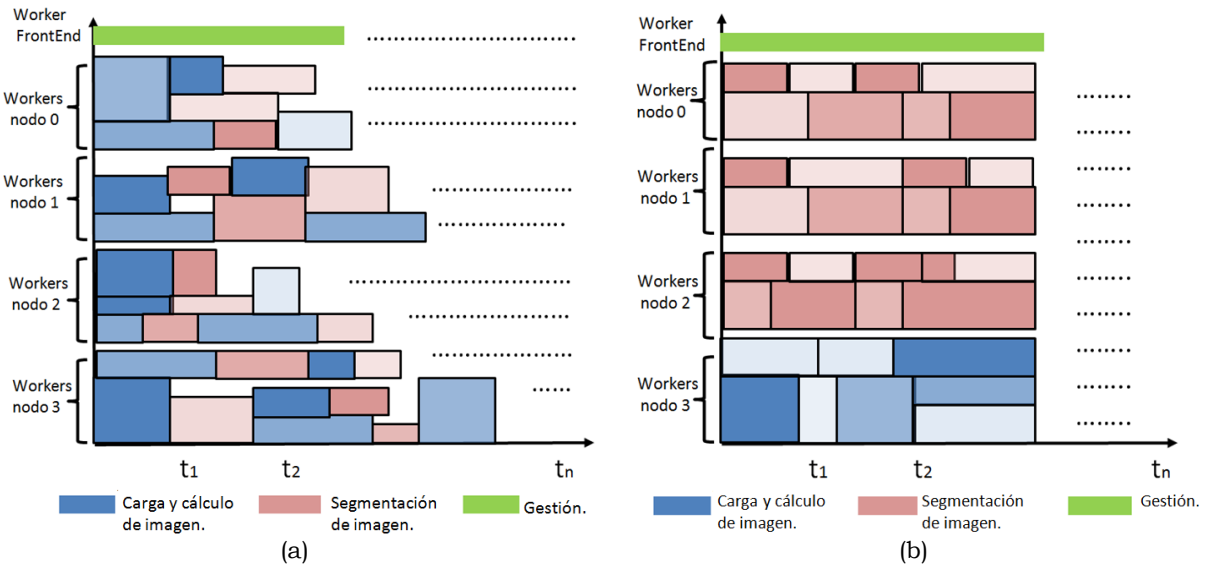


Figura 4.20. Metodología de planificación de trabajos en entorno paralelo distribuido. En (a) se oberva un entorno al que llegan los trabajos, éstos se asignan en función de la disponibilidad de los *workers* sin tener en cuenta las necesidades de recursos de los trabajos. En (b) se aplica una JSS (estrategia de planificación de tareas) para una asignación correcta de trabajos según la necesidad de recursos.

4.6 Carga de trabajo. Planificación de tareas (JSS-WDA).

Una *estrategia de planificación de tareas* (JSS) es un algoritmo que asigna recursos a diferentes trabajos teniendo en cuenta las políticas de administracion del sistema. En [Feitelson 2004] se categorizan y se definen los trabajos que se pueden generar en un sistema de planificación paralela y que son ampliamente aceptados. Estos trabajos pueden ser de dos tipos: los trabajos *rígidos* que son los que necesitan un número exacto de procesadores y los trabajos *moldeables* que son los que se pueden ejecutar con diferente número de procesadores, es decir, aquellos trabajos que parten de un número fijo de procesadores pero que pueden variar su número a medida que el proceso progresa en el tiempo.

Con el objetivo de no fragmentar en exceso el tiempo y la dedicación de los *workers* al procesamiento de cada fichero, se ha implementado una JSS. El motivo es que los ficheros con diferentes tamaños tienen un comportamiento parecido al que podría ser la llegada a un sistema de diferentes clientes con diferentes cargas de trabajo. La figura 4.20(a) muestra un sistema con diferentes nodos al que le llegan trabajos de forma secuencial. Según llegan los trabajos se van asignando a *workers* libres sin tener en cuenta los recursos del sistema. Con este escenario es posible que se genere demasiado swapping (anglicismo de paginación de memoria cuando

el ordenador necesita subir a memoria páginas). Sin embargo en la figura 4.20(b) se muestra un sistema más uniforme donde no hay huecos en los trabajos debido a que se ha implementado una estrategia JSS.

Existen diferentes enfoques para abordar un problema de planificación paralela. En [Buison 2007] se propone una variación dinámica del número de procesadores en función de la carga de trabajo. En [Downey 1998] se define un modelo para calcular la partición más adecuada en función de la escalababilidad y tiempo de respuesta del sistema. En el caso particular que se intenta resolver se han tenido en cuenta diferentes aproximaciones para definir la estrategia final implementada sobre el clúster DIA.

	CPU	velocidad	Memoria	Mem.Swap
Front	8	2261	6113784	1023992
Compute-0-0	8	2261	6113784	1023992
Compute-0-1	8	2261	6113784	1023992
Compute-0-2	8	2261	6113784	1023992
Compute-0-3	8	2660	8186740	1023992

Tabla 4.2. Configuración de recursos del cluster-DIA.

Para tratar el volumen tan heterogéneo de ficheros se ha diseñado una JSS llamada WDA (*workers dynamic assignment*, anglicismo de asignación dinámica de *workers*). El primer objetivo ha sido utilizar toda la potencia que proporciona el cluster pero con la posibilidad de poder modelar la carga de trabajo en función de los trabajos a ejecutar. La tabla 4.2 muestra la configuración del clúster DIA. En ella se observa que el nodo Compute-0-3 tiene mayor configuración de memoria y mayor velocidad de CPU. Por tanto parece lógico que la estrategia a diseñar tenga en cuenta esta característica del clúster DIA y asigne en la medida de lo posible el procesamiento de las dos primeras etapas (figura 4.19) a los *workers* que pertenecen a ese nodo al tener una mayor cantidad de recursos.

4.6.1 Métricas de JSS-WDA.

El objetivo de aplicar las métricas es cuantificar el rendimiento del sistema y analizar la efectividad de la estrategia frente a una estrategia de asignación directa (JSS-AD) de trabajos según se muestra en la figura 4.20(a). En [Utrera 2012] se definen unas métricas que han sido adaptadas para la estrategia JSS-WDA. Los índices medidos en este trabajo han sido el tiempo de respuesta, la desaceleración

media y la fragmentación de tiempos de cpu. El tiempo de respuesta es el tiempo transcurrido desde que se envía un trabajo hasta que éste termina. La desaceleración se refiere al tiempo de ejecución más el tiempo de espera de todos los trabajos. El indicador de fragmentación solo se utiliza para tareas que están en espera. Este índice cobraría más importancia si el escenario tuviera terceros trabajos que tuvieran que analizar *workers* también. Las medidas realizadas en el clúster DIA se multiplican en un sistema clúster con más nodos. Es decir, los avances que se puedan extraer en un sistema con cinco nodos, se hacen mucho más palpables en un sistema con cientos de nodos.

$$desaceleración = \frac{TiempoEspera + TiempoEjecución}{TiempoEjecuciónAjustado} \tag{4.31}$$

$$fragmentación = \frac{\sum_{esperaWorkerOcupado}^{t=comienza\ \ el\ proceso} NumWorkersLibre}{TiempoTotalWorkload * NumTotWorker} \tag{4.32}$$

4.6.2 Implementación de JSS-WDA.

En principio esta implementación tiene ciertas ventajas sobre otras estrategias como puede ser que no se necesita tener un conocimiento a priori de cada trabajo, o bien, que no se debe hacer cambios en el número de procesadores en función de la carga actual. Sin embargo el *worker* gestor necesita tener una tabla de donde lea los ficheros a procesar asi como su tamaño. También necesita otra tabla que relacione *worker* con nodo. Esto es sencillo, pues ya se ha comentado que cada *worker* puede recuperar el nodo que le corresponde mediante la función Matlab *sysname(hostname)* y enviar esta información al *worker* gestor. Otro detalle especial a tener en cuenta es reajustar la carga en los *workers* si en algún momento por motivos de saturación fuese necesario. La expresión 4.33 es un resultado empírico de tal forma que MPL es un parámetro que es mayor de 1 cuando un *worker* ejecuta tareas que no son de su competencia y es igual a 1 cuando sí son de su competencia.

$$TiempoEjecEstimadoNoAsig = \sum_{t=comienzo}^{t=fin} tiempoEjec * MPL(t) \tag{4.33}$$

El modelo de la JSS-WDA queda definido mediante la expresión (4.34). Donde el parámetro OV es un valor entre 0 y 1 que indica los bloqueos producidos por lecturas a disco.

$$TiempoEjecEstimado = TiempoEjecEstimadoNoAsig + tiempoEjec * OV \qquad (4.34)$$

La tabla 4.3 muestra los cuatro tiempos promediados: de espera, de ejecución, de respuesta y de desaceleración. Los resultados se muestran para los dos tipos de estrategias analizadas.

	JSS-AD	JSS-WDA
Tiempo de espera promedio (sg.)	2	1.4
Tiempo ejecución promedio (sg)	4	3.2
Tiempo de respuesta promedio (sg)	3	2.8
Desaceleración promedio (sg.)	4	1.9

Tabla 4.3. Tiempos de proceso mediante estrategia JSS-WDA.

4.7 Resultados en pared de carbono y en pared metálica.

Los datos que siguen a continuación están repartidos en dos clases. La primera de ellas es la detección de los modos y los intervalos temporales en los que son detectados. La segunda información son los tiempos conseguidos mediante la paralelización del proceso. Como dato importante, resaltar que los resultados del modelo fueron contrastados y validados con expertos en el análisis de estos modos.

Los datos de entrada al modelo fueron de dos tipos diferentes. Las descargas dentro del rango [75410, 79461] corresponden a descargas efetuadas en el dispositivo con pared de carbono y las descargas en el rango [81938, 85485] corresponde a descargas de pared metálica (tabla 4.4). Parte de las medidas realizadas por el experto son las que aparecen en las tablas 4.5 y 4.6. Indicar también que, en las columnas donde aparecen datos manuales, se trata de información proporcionada por el experto para contrastar resultados.

Descargas	Señales	Canales
Pared de carbono		
75410 - 79461	H3	1, 2, 3, 4, 5, 6, 7
	I8	1, 2, 3
	PP4	1, 2, 3, 4, 5, 6, 7
	PP8	1, 2, 3, 4, 5, 6, 7
	S40	16
	T0	1, 2, 6, 7, 8, 9
Pared metálica		
81938 - 85485	H3	1, 2, 3, 4, 5, 6, 7
	I8	1, 2, 3
	PP4	1, 2, 3, 4, 5, 6, 7
	PP8	1, 2, 3, 4, 5, 6, 7
	S40	16
	T0	1, 2, 6, 7, 8, 9

Tabla 4.4. Configuración de descargas correspondiente al dispositivo con pared metálica y con pared de carbono.

El significado de las columnas de las tablas 4.5 y 4.6 son los siguientes: la columna número de intervalos manual se refiere al número de tramos detectados y que en conjunto forman un armónico. La columna número de intervalos automático tiene el mismo significado que la anterior pero en este caso ha sido el modelo quien lo ha detectado. La columna intervalo completo manual se refiere al instante inicial y final donde comienza y finaliza el armónico. La columna intervalo completo automático tiene el mismo significado que la columna anterior pero en este caso lo ha detectado el modelo

Número. Descarga	Número Intervalos manual	Número Intervalos aut.	Intervalo completo manual (sg.)		Intervalo ajustado automat. (sg.)	
75410	3	13	53.927	54.176	54.0499	54.1948
75411	3	20	53.986	55.376	53.9029	55.4872
75412	6	4	53.946	54.499	54.0168	54.3214
75414	1	24	55.932	56.180	55.9253	56.1193
75783	1	18	57.104	57.143	57.0674	57.1319
77078	5	17	54.073	54.486	53.8151	54.3800
77919	2	25	46.040	46.337	46.0514	46.3570
77920	3	33	45.907	46.538	45.9119	46.5415
77921	3	15	46.120	46.597	46.1452	46.6469
77925	2	33	46.140	46.340	46.1597	46.3693
77926	2	20	46.120	46.270	46.1334	46.2949
77929	1	18	46.435	46.445	47.2368	47.2626
77931	3	16	46.140	46.597	46.1563	46.6202
78008	2	13	54.136	54.301	52.2084	52.4553
78010	6	25	53.810	54.320	53.7131	54.3540
78011	1	15	53.790	54.257	53.4440	54.1421
78012	3	5	53.707	54.645	53.7660	54.6613
78014	6	9	53.752	54.640	53.8337	54.3980
78017	3	13	53.882	54.226	53.9356	53.9786
78018	7	12	53.892	54.444	53.9965	54.0395
78019	7	7	53.820	54.350	53.9668	54.0313
78130	3	10	54.210	54.475	54.2613	54.5015
79455	6	12	54.010	54.630	54.1268	54.6492
79456	6	5	53.920	54.730	53.0561	54.7566
79458	7	9	53.965	54.413	55.9700	56.0560
79459	6	15	54.120	54.868	54.1594	54.7399
79460	1	26	53.770	53.810	54.3253	54.3597
79461	2	16	54.094	54.370	54.4364	54.4537

Tabla 4.5. Intervalos encontrados por el medio manual y automático para descargas de pared de carbono.

Intervalo temporal del fundamental localizado por el experto.		Intervalo temporal del fundamental localizado automáticamente.	
Inicio (sg.)	fin (sg.)	Inicio (sg.)	fin (sg.)
54.010	54.022	53.0269	53.0484
54.073	54.086	53.0682	53.0897
54.095	54.372	53.1872	53.2087
54.376	54.436	54.1268	54.3849
54.436	54.490	54.3872	54.5162
54.562	54.630	54.4515	54.5053
		54.5954	54.6492
		55.0882	55.0989
		55.2806	55.2914
		55.8200	55.8307
		56.5142	56.5895
		56.7266	56.7588

Tabla 4.6. Tabla de medidas detectadas manualmente y automáticamente para la descarga de pared de carbono 79455.

La tabla 4.6 muestra en detalle los cálculos realizados para la descarga 79455 de pared de carbono. Revisando los datos que aparecen en la tabla 4.6 se comprueba cómo el modelo ha sido capaz de detectar modos en doce intervalos temporales para esta descarga. Parte de estos intervalos son los mismos detectados de forma manual por el experto. Según estos datos el porcentaje de éxito sería del 100%. La tabla 4.5 muestra los números de descarga junto con el número de intervalos detectados de forma manual y el número de intervalos detectados de forma automática. Se indica el inicio y final temporal donde se detectan armónicos así como el número de ellos localizados.

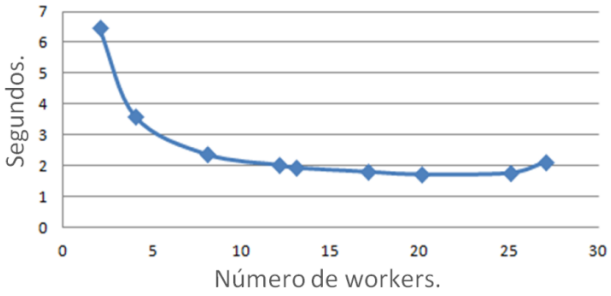


Figura 4.21. Tiempos de proceso por fichero para descargas de pared de carbono.

Figura 4.22. Speedup de proceso para descargas de pared de carbono.

Los datos de escalabilidad sobre la paralelización de la solución implementada se muestran en la figuras 4.21 y 4.22 para descargas de pared de carbono y en las figuras 4.23 y 4.24 para descargas de pared metálica. Los tiempos son similares para los dos tipos de decargas. A medida que los ficheros son más voluminosos, más representativa es la utilidad de la estrategia JSS-WDA. En este sentido las descargas de pared metálica tienen ficheros que sobrepasan los 100 Mb. de volumen y ahí es donde se observan mejores resultados de la aplicación de la metodología debido a que los tiempos de carga son más altos.

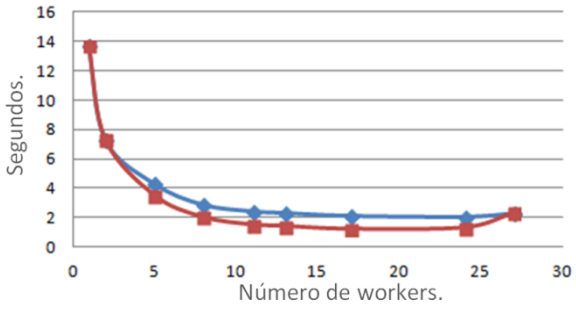


Figura 4.23. Tiempos de proceso por fichero para descargas de pared metálica.

Figura 4.24. Speedup de proceso para descargas de pared metálica.

Merece especial atención la figura 4.23. En esta figura, la gráfica azul muestra los tiempos de proceso por fichero para el procesamiento paralelo sin optimizar. Sin embargo la gráfica roja muestra el proceso optimizado que atiende al reparto de tareas especializadas (JSS-WDA). Asignando al nodo con mayor recursos el procesamiento de ficheros.

ficheros de hasta 20 MB			ficheros de 20 Mb a 50 Mb			ficheros mayores de 100 Mb		
Num. *workers*	Tiempo (sg.)	Speedup (sg.)	Num. *workers*	Tiempo (sg.)	Speedup (sg.)	Num. *workers*	Tiempo (sg.)	Speedup (sg.)
1	2,537	1,000	1	6,347	1,000	1	17,095	1,000
7	0,526	4,820	7	1,305	4,863	7	3,668	4,660
19	0,326	7,774	19	1,084	5,854	11	2,884	5,927
23	0,295	8,607	23	0,984	6,449	16	2,942	5,810
27	0,242	10,955	27	1,074	5,912			
29	0,232	10,955						

Tabla 4.7. Resultados en función de los ficheros a procesar.

La tabla 4.7 muestra los datos obtenidos en función de los diferentes tipos de fichero a procesar. Se analizaron un total de 95937 ficheros. El tiempo de CPU que utilizó un nodo secuencial fue de 302,2 horas. Con la paralelización de la implementación realizada se consiguió bajar a 45 horas.

Esto significa que se logró una ganancia de velocidad de $^{302,2}/_{45} = 6,7$. Los resultados de pared metálica se adjuntan en el anexo B de la memoria.

4.8 Conclusiones.

En este capítulo se aplican y desarrollan diferentes técnicas que sirven de base para el desarrollo posterior de una metodología paralela que descubra la existencia de ciertos fenómenos que se producen en descargas de plasma nuclear. La metodología desarrollada tiene dos partes. Para aplicar esta metodología se desarrolló un algoritmo paralelo que, aplicando los métodos de segmentación y carga, es capaz de procesar las imágenes de forma distribuida con una gestión eficaz del tiempo de procesador. Esta metodología es lo suficientemente general por lo que se puede desplegar en cualquier arquitectura tipo cluster.

El algoritmo codificado dispone de la posibilidad de configuración en el sentido de que el espectro de patrones a determinar sea más amplio. La combinación de las características de las regiones unido al procesamiento distribuido para la localización de patrones hacen de este algoritmo una herramienta eficaz para el descubrimiento rápido de modos multiarmónicos.

Capítulo V.
Identificación de formas de onda del TJ-II.

5.1 Introducción.

La clasificación es uno de los campos importantes dentro del contexto del procesamiento de señales. El procesamiento paralelo de señales puede ser implementado tanto a nivel de datos como a nivel de tareas. El trabajo que se presenta en este capítulo muestra un entorno paralelo para la clasificación de señales del TJ-II. Entre los diferentes paradigmas que abordan el problema de la clasificación se encuentran las *máquinas de vectores soporte* con *aprendizaje supervisado.* La idea principal es utilizar información que reside en un conjunto de datos de entrenamiento para generar funciones de decisión capaces de clasificar automáticamente cada nuevo dato en las diferentes categorías o clases existentes. Es importante resaltar que esta clasificación se basa en generar un hiperplano que sirva para asignar cada muestra a una clase determinada. Las muestras del conjunto de entrenamiento poseen una etiqueta con la clase a la que pertenece, de esta forma se puede evaluar el número de muestras etiquetadas correctamente y posteriormente poder corregir la función para mejorar la clasificación si fuera necesario. Este proceso de verificación y corrección se denomina *aprendizaje supervisado* y entre las máquinas de aprendizaje más populares aplicadas a la clasificación están las máquinas de vectores soporte (SVMs) [Vapnik 1995, Hofman 2008].

Como ya se comentó en el capítulo 3 de esta Tesis una de las limitaciones de las máquinas de vectores soporte (SVM) es que están pensadas solamente para resolver problemas de dos clases, no obstante como el problema que se resuelve abordar en este capítulo será la clasificación de diferentes tipos de se hace necesario extender la formulación de las SVMs al problema multiclase.

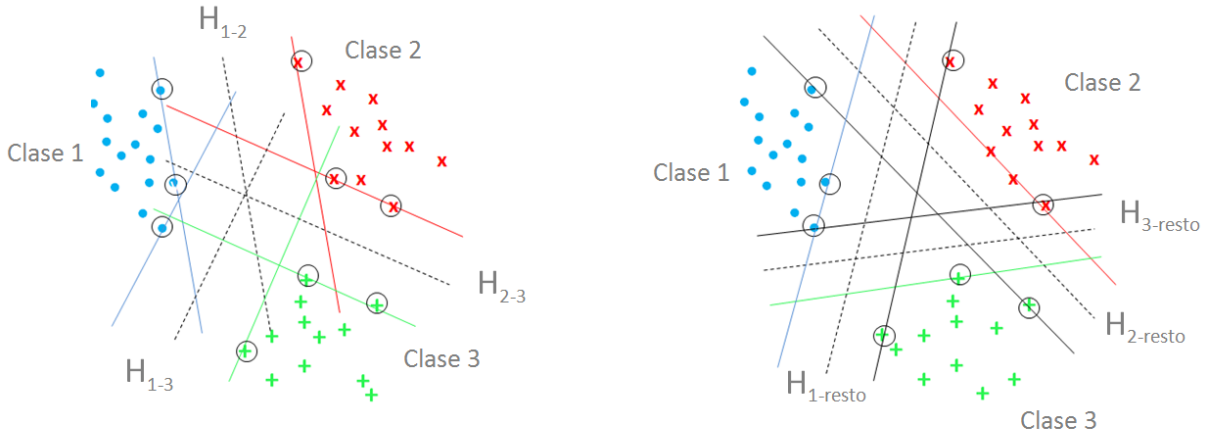
5.2 SVM multiclase.

Como ya se ha señalado en la introducción de este capítulo las SVMs en su forma original son un clasificador binario y no son directamente aplicables a

problemas de clasificación con más de dos categorías. En este contexto decir que existen dos enfoques que sirven para extender su funcionamiento a los problemas multi-clase:

- El enfoque multi-objetivo: utilizar varias SVMs binarias en conjunto.
- El enfoque mono-objetivo: modificar la estructura de la SVM para que una sola máquina sea capaz de separar más de dos clases simultáneamente.

El enfoque multi-objetivo es el que se utiliza en este trabajo y la utilización de varias SVMs binarias es una de las principales razones por las que la computación paralela cobra un especial interés a la hora de abordar el problema.



(a) Paradigma uno contra uno. (b) Paradigma uno contra todos.

Figura 5.1. SVMs Multi-objetivo.

El enfoque multi-objetivo es el más utilizado cuando se quiere extender las SVMs a problemas multi-clase, siendo las implementaciones de *uno contra todos* (uct-SVM) y *uno contra uno* (ucu-SVM) sus exponentes más conocidos. El método *uno contra todos* consiste en implementar un número K de SVMs, en donde la *r-ésima* SVM está encargada de separar a los elementos de la clase r del resto de los ejemplos. Por otro lado, el método *uno contra uno* crea $K(K\text{-}1)/2$ SVMs, es decir una para cada par de clases. En la figura 5.1 se puede apreciar el uso de SVMs multi-objetivo para separar un problema de 3 clases. En 5.1(a) se utiliza una SVM binaria para separar cada par de clases (ucu-SVM). En 5.1(b) se utiliza una SVM por clase para separarla del resto de los ejemplos del problema (uct-SVM). Los colores de los

márgenes indican a que clases está asociado el hiperplano en cuestión. En el caso de la uct-SVM, el color negro corresponde al resto de los ejemplos

5.3 Algoritmos de entrenamiento.

En el capítulo 3 se justificaron los motivos que hacen de la paralelización una solución para abordar problemas de clasificación con las SVM. No todos los algoritmos de entrenamiento trabajan de la misma forma. En [Yom-Tov 2004] se puede ver una clasificación de estos algoritmos. Una aproximación es particionar y tratar de forma separada parte de las muestras de entrenamiento, otra aproximación puede ser trabajar con el *working set* (anglicismo de conjunto de trabajo). No obstante resaltar que en [Scholkopf 2002] se expresa de forma concreta que no todos los algoritmos que sirven para este objetivo son paralelizables.

5.3.1 Introducción.

El entrenamiento de una SVM binaria se puede abordar como un problema de optimización cuadrático (QP) con restricciones lineales de la forma $min_{\{u\}} u^T Q u + c^T$, en donde el vector c puede ser el vector nulo. Normalmente, problemas de esta índole pueden ser resueltos eficientemente con algoritmos similares al método de Newton, tales como el propuesto en [Coleman 1996]. Cuando la matriz Q no es muy densa éstos algoritmos suelen tener un tiempo de ejecución bastante rápido. Sin embargo, cuando la matriz Q es especialmente grande, calcular y almacenar sus valores se hace cada vez más inviable a medida que su tamaño aumenta. Este es el caso en el entrenamiento de las SVMs binarias. En ellas Q tiene una dimensionalidad de $m \times m$ y cada celda corresponde al producto punto de dos elementos del problema. Si bien Q es simétrica y solo es necesario calcular $^{m(m+1)}/_{2}$ de los productos, la gran mayoría de éstos no serán nulos, y el crecimiento de su tamaño seria de todas formas cuadrático en función del número de elementos de entrenamiento m.

5.3.2 Chunking y Algoritmos de Descomposición.

Como se puede apreciar, el uso directo de métodos tradicionales consume rápidamente los recursos de almacenamiento e invierte bastante tiempo en cómputo. Sin embargo, hay características intrínsecas a la formulación de las SVMs que se pueden utilizar a favor para hacer más eficiente su resolución. Una de ellas es lo disperso que suele ser el vector resultante u, que indica cuales de los

elementos del problema son vectores soporte (VS) y en cuanto contribuyen a determinar el hiperplano separador. Como suelen ser pocos los elementos de u que no son nulos y como la fracción de ejemplos que son VS tiende a ser constante a medida que el tamaño del problema aumenta, se hace necesario calcular solamente una cantidad reducida de columnas de Q que están relacionadas con dichos elementos. Con esto se puede asumir con bastante confianza que serán escasos los elementos de u que estarán involucrados en la solución final del problema, pero no se sabe cuáles de todos estos elementos son los que serán realmente los vectores de soporte en dicha solución final del problema, así el inconveniente inicial de calcular y almacenar la matriz Q cambia por el de encontrar los elementos de u que no son nulos en la solución optima.

Uno de los métodos iniciales que se propusieron para encontrar los vectores soporte, denominado Chunking [Boser 1992], consistía en trabajar inicialmente con una fracción B de los elementos de u, denominada *working set*.

Al final de una iteración, después de reajustar los elementos en B, aquellos que terminan siendo nulos son descartados del conjunto de trabajo, mientras que otros M elementos que violen las condiciones del problema son agregados al conjunto. Este ciclo se repite hasta que no haya más elementos que violen las condiciones de convergencia. Aunque el rendimiento mejora usando esta estrategia, en la mayoría de los casos, el número de elementos que salen del conjunto de trabajo es menor que los M elementos que ingresan, por lo que B crece a un tamaño que tampoco es tratable.

En [Osuna 1997], propusieron que el número de elementos que se adhirieran a B en cada iteración fuera el mismo que se sacasen del mismo, aun así todos los elementos en B son vectores soporte. Esto asegura que los recursos de espacio no se vean sobrepasados en ninguna iteración. En este esquema, bajo ciertas circunstancias, se puede asegurar la convergencia del entrenamiento. En [Osuna 1997] se hace referencia a este tipo de métodos como *Algoritmos de Descomposición*.

5.3.3 Algoritmo SMO.

El algoritmo *Secuencial de Optimización Mínima* (SMO) es otro tipo de algoritmo de entrenamiento de las SVMs y se debe a Platt [Platt 1999]. Se obtiene a partir de la idea del método de descomposición pero llevado a su extremo al optimizar un subconjunto mínimo de dos elementos en cada iteración. Para la elección de estos dos elementos existen a su vez diferentes algoritmos. En este trabajo se ha

analizado dos de ellos: el de *máximum Gain* (MG) y *most violatong pair* (MVP). Glasmachers hace un estudio detallado de la comparación de estos dos algoritmos [Glasmachers 2006].

La fortaleza del algoritmo SMO según Joachims [Joachims 1999] reside en el hecho de que el problema de optimización para dos elementos admite una solución analítica, eliminando la necesidad de usar un optimizador de programación cuadrática iterativo como parte del algoritmo.

El requisito (3.7) visto en la sección 3.3.2.2, $\sum_{i \in I} \alpha_i \, y_i = 0$, obliga en todo momento a que el número de multiplicadores que puede ser optimizado en cada paso sea dos. Cada vez que un multiplicador es actualizado, por lo menos otro multiplicador necesita ser ajustado con el propósito de mantener la condición verdadera. En cada paso, SMO elige dos elementos α_i y α_j para optimizarlos y de esta forma minimizar (3.7). La elección de los dos elementos es determinada por una heurística, mientras que la optimización de los dos multiplicadores se realiza analíticamente. El tiempo de computación es reducido. A pesar de necesitar más iteraciones para converger, en cada una de las iteraciones necesita menos operaciones. Además del tiempo de convergencia, otra característica del algoritmo, radica en que este no necesita almacenar la matriz del kernel en la memoria, y el motivo es que no necesita operaciones matriciales. Cuando se trabaja con grandes volúmenes de datos el algoritmo SMO es adecuado porque escala bien con el tamaño del conjunto de entrenamiento.

Para optimizar los elementos α_i y α_j se procede de la siguiente manera: se seleccionan dos límites L y H de tal manera que $L \leq \alpha_i \leq H$ para que a su vez α_j satisfaga la restricción $0 \leq \alpha_j \leq C$. Se puede demostrar que L y H están dados por (5.1) y (5.2) respectivamente.

$$\text{Si } y^{(i)} \neq y^{(j)}, \quad L = \max\left(0, \alpha_j - \alpha_i\right), \; H = \min\left(C, C + \alpha_j - \alpha_i\right) \tag{5.1}$$

$$\text{Si } y^{(i)} = y^{(j)}, \quad L = \max\left(0, \alpha_i + \alpha_j - C\right), \; H = \min\left(C\alpha_j, +\alpha_i\right) \tag{5.2}$$

Posteriormente se trata de calcular α_j para minimizar la función objetivo (3.7). Si este valor termina fuera de los límites L y H entonces se ajusta el valor de α_j dentro de estos dos límites. Se puede demostrar que el valor óptimo de α_j esta dado por:

$$\alpha_i := \alpha_j \frac{y^{(i)}(E_i - E_i)}{\eta} \tag{5.3}$$

donde

$$\begin{gathered} E_k = f(x^{(k)}) - y^{(k)} \\ \eta = 2\langle x^{(i)}, x^{(j)} \rangle - \langle x^{(i)}, x^{(i)} \rangle - \langle x^{(j)}, x^{(j)} \rangle \end{gathered} \tag{5.4}$$

y E_k se puede ver como el error de la salida de SVM para la muestra *k-ésima* y la etiqueta verdadera $y^{(k)}$. Para calcular η se puede utilizar cualquier función kernel en lugar de los productos escalares. Posteriormente se trata de calcular α_j en el rango de $[L, H]$:

$$\alpha_j := \begin{cases} H & si\ \alpha_j > H \\ \alpha_j & si\ L \leq \alpha_j \leq H \\ L & si\ \alpha_j < L \end{cases} \tag{5.5}$$

Una vez calculado α_j se puede resolver α_i que estará dado por:

$$\alpha_i = \alpha_i + y^{(i)} y^{(j)} (\alpha_i{}^{(old)} - \alpha_j) \tag{5.6}$$

Donde $\alpha_i{}^{(old)}$es el valor de α_j antes de la optimización de (5.1) y (5.5).
Una vez optimizado α_i y α_j el siguiente paso es calcular el umbral del *bias* de tal forma que se cumplan las condiciones de *KKT* para los ejemplos *i-ésimo* y *j-ésimo*. Si después de la optimización α_i cumple $0 < \alpha_i < C$ entonces el umbral b_1 es válido porque obliga a SVM a la salida $y^{(i)}$ cuando la entrada es $x^{(i)}$.

$$b_1 = b - E_i - y^{(i)} \left(\alpha_i - \alpha_i^{(old)}\right) \langle x^{(i)}, x^{(i)} \rangle - y^{(j)} (\alpha_j - \alpha_j{}^{(old)}) \langle x^{(i)}, x^{(j)} \rangle \tag{5.7}$$

Y de forma similar, el siguiente umbral b_2 es válido si $0 < \alpha_j < C$.

$$b_2 = b - E_j - y^{(i)} \left(\alpha_i - \alpha_i^{(old)}\right) \langle x^{(i)}, x^{(j)} \rangle - y^{(j)} (\alpha_j - \alpha_j{}^{(old)}) \langle x^{(i)}, x^{(j)} \rangle \tag{5.8}$$

Si se cumple para ambos $0 < \alpha_i < C$ y $0 < \alpha_j < C$ entonces los umbrales b_1 y b_2 son válidos y además son iguales. Si ambos $\alpha's$ están en los límites 0 o C (p.e. α_i=C y $\alpha_j = 0$, entonces todos los umbrales entre b_1 y b_2 satisfacen las condiciones de KKT y el *bias* está dado por $b := {(b_1 + b_2)}/{2}$. Una expresión completa de b sería:

$$b := \begin{cases} b_1 & si\ 0 < \alpha_i < C \\ b_2 & si\ 0 < \alpha_j < C \\ (b_1 + b_2)/2 & en\ otro\ caso \end{cases} \tag{5.9}$$

5.3.4 Algoritmo sSV-Kernel Adatron.

El algoritmo Adatron fue propuesto por primera vez por [Anlauf 1989]. El objetivo de maximizar el margen de separación del hiperplano requiere intensas operaciones de cómputo. Estas tareas están originadas en gran parte por la necesidad de fragmentar los datos que tienen que ser clasificados. Con este nuevo enfoque el problema se resuelve mediante problemas de optimización cuadráticos de menor tamaño. En [Frieb 1998a] se propone aplicar una función kernel al algoritmo Adatron y en [Frieb 1998b], los mismos autores, proponen un nuevo tratamiento para el cálculo del *bias* visto en la sección precedente. De forma paralela en [Vijayakumar 1999] se propone un tratamiento alternativo del *bias* aplicado a un algoritmo con aprendizaje secuencial de gradiente ascendente, el sSV-KA (acrónimo de sequential Support Vector Kernel Adatron). Este nuevo tratamiento tiene efectos directos en la función de coste y afecta a las condiciones de convergencia para obtener la separación máxima del hiperplano a los puntos que estén más cerca de él. Con el nuevo enfoque de [Vijayakumar 1999] el conjunto de vectores se redenomina de la forma $S' = \{x_1, x_2, \ldots . x_m, \lambda\}$ y se incorpora el temino *bias* (b) en el vector de pesos $w' = \{w_1, w_2, \ldots . w_m, b/\lambda\}$, donde λ es un escalar constante llamado *factor de aumento.*

Según este nuevo planteamiento, la ecuación (3.12) se puede redefinir de la siguiente forma:

$$P'_{\{w',\xi\}}: minimizar\ \frac{1}{2}\|w'\|^2 + C\sum_{i\in I}\xi_i \tag{5.10}$$

$$\text{Sujeto a } y_i(w^{'} \cdot x_i^{'}) - 1 + \xi_i, \forall i \in I, \qquad \xi_i \geq 0, \forall i \in I \tag{5.11}$$

Según esta modificación, la definición del margen según la ecuación (3.7) se puede redefinir bien maximizando $^{2}/_{\|w\|}$ o bien minimizando $\|w\|^2$. Si se opta por la segunda opción se tiene la siguiente expresión:

$$minimizar \left\|w^{'}\right\|^2 = \|w\|^2 + b^2/\lambda \tag{5.12}$$

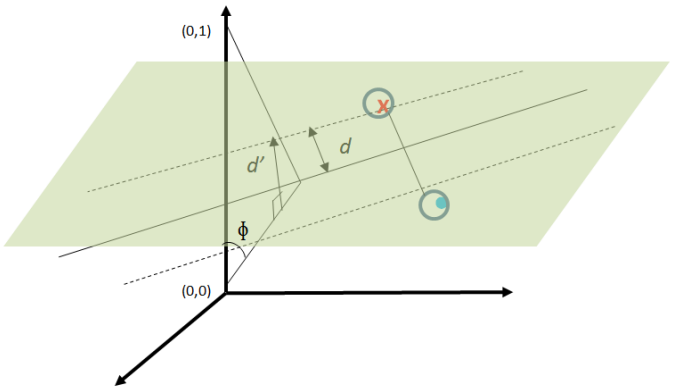


Figura 5.2. Margen modificado *d'* en el espacio de características.

Para ver cómo afecta este nuevo *bias* en la maximización del margen implica introducir el nuevo valor λ. En primer lugar se puede asumir que λ=1. En la formulación inicial del margen se puede maximizar mediante $d^2 = 1/\|w\|^2$. El margen modificado $d^{'}$ es la mínima distancia de los puntos al hiperplano. Esa distancia es:

$$d^{'2} = d^2 \cdot cos^2\phi \tag{5.13}$$

El ángulo ϕ satisface la relación:

$$cos^2\phi = 1/(1+m^2) \text{ donde } m = {}^{1}/_{\|w\|} \tag{5.14}$$

Ahora se está en condiciones de establecer la relación entre el *bias* y el margen modificado mediante la expresión:

$$d'^2 = {}^1/_{(b^2 + \|w\|^2)} \tag{5.15}$$

En casos de datos de entrada de alta dimensión, la contribución del *bias* es despreciable y la aproximación es buena. A partir de (5.13) y haciendo referencia a la figura 5.2 se puede comprobar que si el ángulo ϕ es pequeño entonces la aproximación tiende a mejorar. Con este nuevo planteamiento la formulación del problema dual vista en la ecuación 3.16 adoptaría la siguiente forma:

$$maximizar\ L'_D(h) = \sum_{i \in I} h_i - \frac{1}{2} h \cdot D' h \tag{5.16}$$

$$sujeto\ a\ 0 \leq h_i \leq C, \qquad \forall i \in I$$

Donde $D'_{ij} = y_i y_j K(x_i, x_j) + \lambda y_i y_j$, y $h_i = \alpha_i$. Y en consecuencia 3.17 se reformula de la siguiente forma:

$$f(x) = signo\left(\sum_{i \in I} h_i\, y_i\, K(x_i, x) \cdot x + h_i\, y_i\, \lambda^2\right) \tag{5.17}$$

5.4 Técnicas de medición y análisis.

Para analizar la validez del clasificador se hace uso de técnicas de evaluación de clasificadores binarios. El motivo es que a pesar de ser un clasificador multiclase internamente el sistema se podría considerar como un clasificador binario. Si se considera cada señal de forma independiente, el sistema trata de establecer si esa señal es válida o no (clasificación binaria). Las métricas de medición fundamentalmente son Precisión y Sensibilidad [Makhoul 1999], Especifidad y Exactitud [Baldi 2000] y por último, el coeficiente de correlación de Matthews [Matthews 1975]. A las cuatro primeras se suele referir como las medidas PRAS y a la última como MCC. También se suele utilizar la matriz de confusión con la que resulta fácil entender y explicar los efectos de las predicciones erróneas. A continuación se presentan cada una de éstas métricas.

Matriz de confusión: es el número de muestras que fueron asignadas a cada clase por el clasificador. La diagonal principal indica el número de muestras que han sido correctamente clasificadas para cada clase. En cambio, los elementos ubicados

fuera de la diagonal principal, indican los errores de asignación, ya sea porque perteneciendo a una misma clase han sido asignados a clases distintas (error de omisión), o porque han sido clasificadas en una categoría, cuando en realidad pertenecen a otra (error de comisión). La matriz de confusión considera cuatro casos:

1. Verdadero positivo: caso positivo etiquetado como positivo(VP).
2. Verdadero negativo: caso negativo etiquetado como negativo (VN).
3. Falso positivo: caso negativo etiquetado como positivo (FP).
4. Falso negativo: caso positivo etiquetado como negativo (FN).

• Precisión: Mide el porcentaje de predicciones positivas realizadas por el sistema y que son correctas. Es útil para comprobar que los resultados ofrecidos por el sistema son correctos, pero no tiene en cuenta las clasificaciónes no emitidas (100% si no hubo muestras de la clase asignadas a otra clase).

$$Precisión = \frac{VP}{VP+FN}$$

• Especifidad: Mide el porcentaje de predicciones verdaderos negativos realizadas por el modelo y que son correctas (100%: No se asignaron muestras de otra clase a esta clase).

$$Especifidad = \frac{VN}{VN+FP}$$

• Sensibilidad: Mide la proporción de casos positivos clasificados por el modelo.

$$Sensibilidad = \frac{TP}{TP+FN}$$

• Exactitud: Mide la proporción total de aciertos del sistema, tanto de diagnósticos positivos como negativos. Es similar a la precisión, pero más completa, ya que incluye también los resultados negativos (casos no clasificado por el modelo). Es útil como métrica sencilla del funcionamiento general del modelo, pero no indica qué tipo de fallos de clasificación son más comunes, por lo que se completa con el resto de medidas.

$$Exactitud = \frac{TP+TN}{TP+TN+FP+FN}$$

• MCC: El valor de MCC permite obtener una medida más balanceada basada en los parámetros de PRAS. Esta medida permite hacerse una idea de lo que se conocería como "eficiencia global" del modelo o de los expertos.

$$MCC = \frac{TPxTN - FPxFN}{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}$$

5.5 Paralelización.

Este apartado presenta y analiza la paralelización de los algoritmos presentados en el apartado anterior: 1) sSV-KA (*sequential support vector kernel Adatron*), 2) smo-SVM (*sequential minimal optimization support vector machine*) en cascada y 3) smo-SVM Spread Kernel. Son tres algoritmos vistos desde puntos de vista diferentes. Su entorno de aplicabilidad es el análisis de grandes conjuntos de datos.

No obstante los tres necesitan acondicionar los datos para una mayor optimización de la memoria. Uno de estos pasos previos consiste en ordenar los datos de entrenamiento para que la organización de la matriz Kernel no obligue a realizar saltos. La figura 5.3 muesta esta reordenación de los datos en función de su etiqueta [Bottou 2010]. El motivo de esta reordenación es que los datos con igual

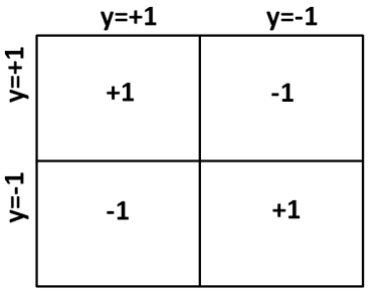


Figura 5.3. Matriz Kernel optimizada.

etiqueta estén agrupados y de esta forma cuando se hace un barrido a la matriz kernel se evita que el sistema ejecute saltos innecesarios con la consiguiente pérdida de tiempo.

En primer lugar se muestra el algoritmo sSV-KA implementado donde es necesario asignar a cada nodo la resolución de un problema de clasificación de forma única. Es un método idóneo para datos inferiores a un millón de muestras debido a que penaliza mucho la memoria disponible del sistema. Este método paraleliza la matriz kernel de modo que cada *worker* trabaja con una parte de la matriz que debe estar en memoria lo que es un problema si la matriz es muy grande. El segundo método a implementar es el smo-SVM en cascada. Éste formaliza una paralelización a nivel de datos de entrenamiento. Para ello es necesario dividir los datos entre el número de *workers* disponibles. De tal forma que, a cada *worker* se le asigna una porción de los datos de entrenamiento. El tercer método a implementar es el smo-SVM Spread-Kernel. Este algoritmo a diferencia de los dos anteriores paraleliza la matriz kernel sin guardarla en memoria. A cada *worker* le asigna unos límites de la matriz Kernel.

5.5.1 sSV-KA.

A diferencia de los otros dos siguientes algoritmos implementados, el sSV-KA [Vijayakumar 1999] no basa la convergencia en la búsqueda del conjunto de trabajo (*working set*).

El problema de optimización se resuelve de forma iterativa donde cada *worker* trabaja y calcula la solución con su parte de la matriz kernel. Es decir, cada *worker* trabaja con una sección de la matriz kernel y envía su actualización al *worker* gestor. El *worker* gestor se encarga de recibir todas las actualizaciones y remitirlas de nuevo al resto de *workers.* El gran problema de este algoritmo es la alta tasa de gestión que tiene el *worker* principal y el tiempo dedicado a las transmisiones de datos además de la limitación de memoria. En [Arana-Daniel 2016, Yom-Tov 2004] se pueden consultar diferentes intentos de paralelización de este algoritmo. La figura 5.4 muestra la estructura de red. El algoritmo 5.1 muestra la paralelización del método. La capacidad del algoritmo está dada por la memoria disponible del sistema.

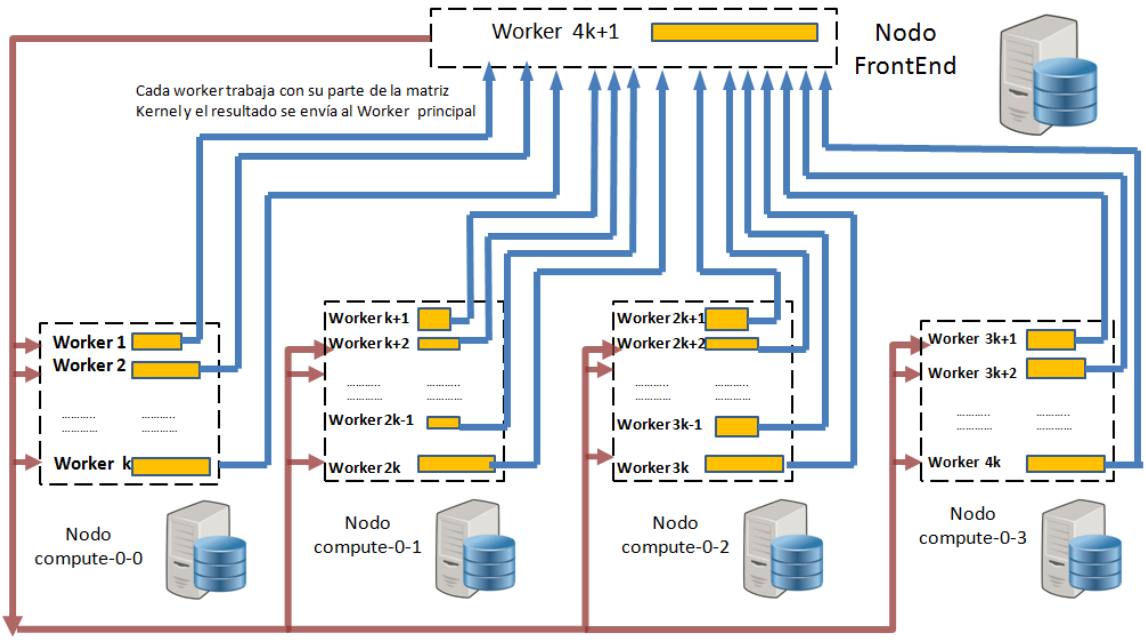


Figura 5.4. Ruta de datos del algoritmo sSV-Kernel Adatron. Las líneas azules señalan la dirección de los resultados de cada *worker* y las líneas marrones la realimentación de los resultados generales hacia los *workers* de cada nodo.

5.5.2 Algoritmo smo-SVM en cascada.

El algoritmo en cascada [Graf 2005] es un concepto de paralelización que optimiza subproblemas de forma independiente y posteriormente combinan los vectores soporte de forma iterativa, véase la figura 5.5. El concepto de trabajo se asemeja en parte al algoritmo sSV-KA en el sentido que se paralelizan los datos de entrada a la SVM. En el caso anterior se repartía la matriz kernel entre los diferentes *workers*. Aquí el concepto cambia, en lugar de repartir la matriz kernel, lo que se reparte son los datos de entrenamiento. La dependencia de los diferentes pasos del algoritmo hace difícil ganar velocidad computacional.

Algoritmo 5.1: sSV-KA.

```
Datos:
N=número de worker
conjunto de entrenamiento Dn, ; n=1.....N.
numWrk=N.
max_diff=1e-5
numIter          % número de iteraciones
Resultado:
convergencia=false;
spmd
        if labindex<> numWrk
                % Conjunto de datos de training que recibe cada worker
                Dn ← distributed (D)   ; n=1.....N
        end
        convergencia=false
  if labindex= numWrk
   then
     h ← zeros(numMuestras,1);
     envia_datos(convergencia,W1 .... Wn-1)
    while ~convergencia & (iter<maxiter)
      δh ← recibe_datos(W1 .... Wn-1)
      if (max(abs(δh))< max_diff
         convergencia=true;
         envia_datos(convergencia,W1 .... Wn-1)
        break;
      end
     δh=min{ max[2/maxD(1-E)-h],C-h }
      h=h+ δh;
      iter=iter+1;
    end
  else
      convergencia ←recibe_datos(Wn)
      D ← calcula_Kernel(Dn)
     % D es la que aparece en la expresión 5.18
    while ~convergencia
       E =h* D
       δ h=min{ max[2/maxD(1-E)-h],C-h }
       h=h+ δh;
       envia_datos(δh,Wn)
       convergencia ←recibe_datos(Wn)
    end
  end
end  % bloque spmd
```

Algoritmo 5.1. Paralelización del algoritmo sSV-Adatron.

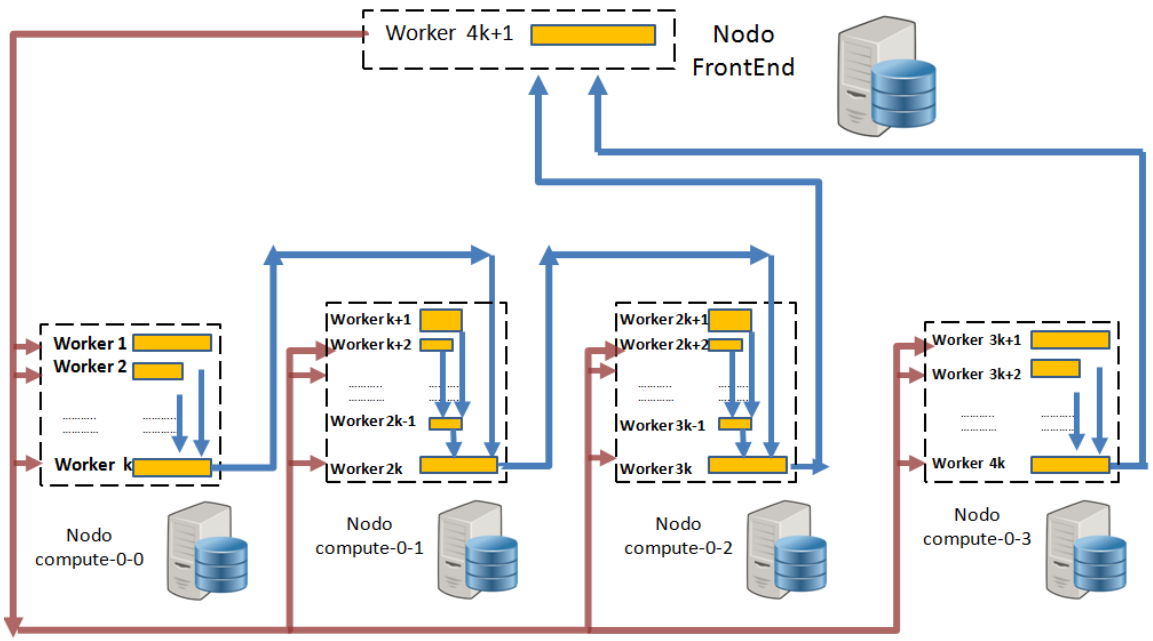


Figura 5.5. Ruta de datos del algoritmo smo-SVM en cascada. Las líneas azules señalan la dirección de los resultados de cada *worker* y las líneas marrones la realimentación de los resultados generales hacia los *worker* de cada nodo. Cada *worker* ejecuta una SVM y desde el primer momento tienen un área de los datos de entrenamiento.

Por ese motivo se divide el conjunto de entrada en conjuntos más pequeños [Collobert 2004] y de esta forma se construye una red donde los *workers* trabajan de forma colaborativa. A modo de ejemplo en la figura 5.6 se muestra la estructura de red en la que cada *worker* ejecuta una SVM para optimizar la búsqueda de los vectores soporte. En cada nivel los vectores del nivel precedente son combinados hasta dejar solo un conjunto de ellos en el último nivel. El resultado del último nivel alimenta de nuevo la entrada del primer nivel. En el primer nivel los *workers* reciben como entradas todos los vectores soporte calculados en el último nivel. Se revisa si hay algún vector soporte que tenga que incorporarse a su conjunto de entrenamiento para mejorar su propio conjunto de vectores soporte.

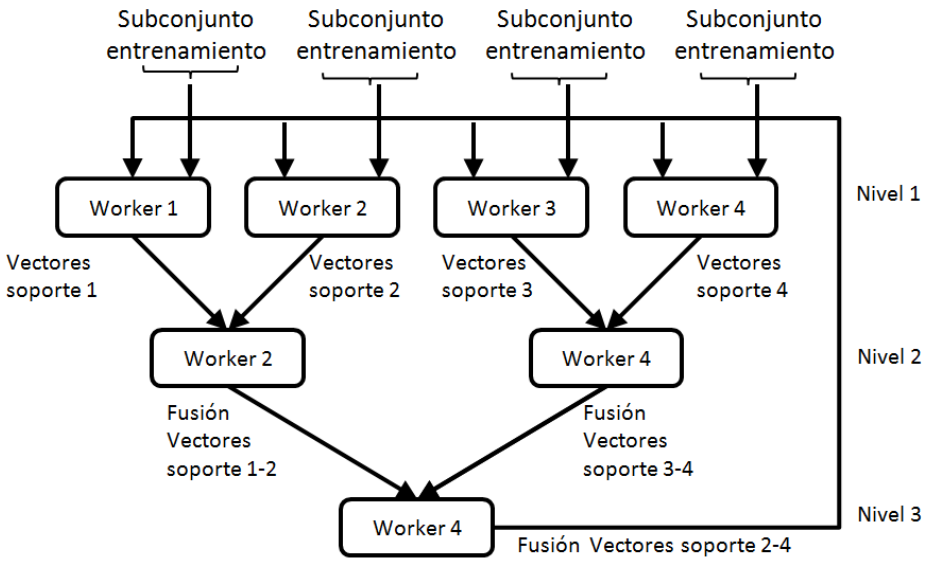


Figura 5.6. Esquema de arquitectura binaria en cascada. Los datos se dividen en subconjuntos y cada uno es evaluado individualmente. Los resultados se combinan dos a dos y entregados al nivel siguiente.

La convergencia se consigue cuando todos los *workers* del primer nivel no tienen ningún vector soporte que añadir. En esta estructura un *worker* nunca debe procesar el conjunto global de los datos de entrenamiento de forma exclusiva. El *worker* de la última capa es el que debe manejar más vectores soporte que el resto. Por ese motivo al *worker n* se le asigna al nodo front-end. El motivo es doble, por un lado la función de distribución de variables (*distributed*) de Matlab asigna a todos los *workers* un área de la matriz de datos y esto obliga a que el *worker n* asignado al front-end esté obligado a procesar una SVM también pues forma parte de la agrupación paralela. Por otro lado al nodo front-end solo se le asigna un *worker* para que disponga de mayor memoria libre y el proceso de gestión sea más ágil.

La figura 5.7 muestra el proceso de filtrado que se realiza en cada nivel. Se inicia el proceso con dos conjuntos de datos disjuntos. En 5.7(a) aparecen

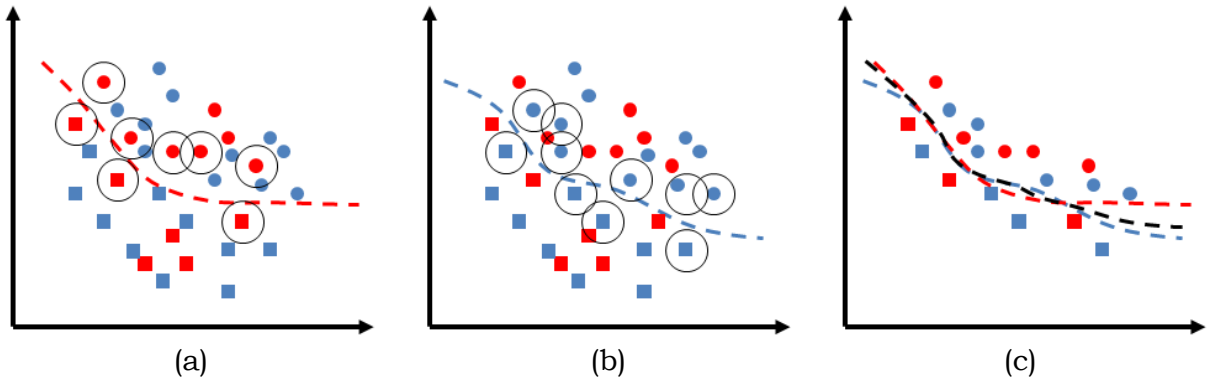


Figura 5.7. Proceso de filtrado del algoritmo SVM cascada.

rodeados de un círculo los vectores soporte del primer conjunto de datos (figuras en rojo) junto con su plano de separación. En 5.7(b) aparecen rodeados de un círculo los vectores soporte del primer conjunto de datos (figuras en azul) junto con su plano de separación. Y en 5.7(c) se muestra el resultado de fusionar los dos conjuntos junto con su plano de separación en color negro. Se aprecia que el plano de separación del total de los dos conjuntos no es muy diferente a si se hubiera tratado los dos conjuntos en total. El algoritmo 5.2 presenta la implementación paralela del SVM cascada.

Algoritmo 5.2: smo-SVM cascada

Datos: conjunto de entrenamiento D_n , ; n=1…..*N*
Resultado: Vectores soporte SV_{new}
N ← número de workers
n=1…..N
K=2 /* nivel de salto entre niveles */
niveles=int(log(N-1)/log /* calcula el nivel de profundidad */
SV_{old} , SV_{new} ← 0 /* grupo de vectores soporte */
D_n ← distributed (D) ; n=1…..*N* /* calculadosen iteración anterior */
numWrktemp=N. /* subconjunto de datos de training para elworker n */
cont=N
convergencia=false
spmd
while ~*convergencia*

```
for i=1 to niveles
    if i==1
        /* la primera iteración es diferente pues se */
        /* debe tomar datos training */
        calcula_alfa_grad_kernel(D_n + SV_new )
        almacena SV_n
        W_n ← -0.5*α_n^T · Q_n · α_n + e_n^T · α_n
        G_n ← - α_n^T · Q_n · e_n
        envia_datos(W_n , G_n, Q_n) → lab+k)
    end
    while cont>1
        if lab ==cont
        then
            /* necesario para contabilizar elemento impar */
            if impar(numWrktemp) and cont==(1+K)
            then
                W_n-k, G_n-k ← recibe de lab-k
                fusiona_grad(G_n-k , G_n, α_n, α_n−k)
                fusiona_fobjetivo(W_n-k , W_n , α_n, α_n−k)
                W_1, G_1 ← recibe de lab(1)
                fusiona_grad(G_1 , G_n, α_n, α_1)
                fusiona_fobjetivo(W_1 , W_n , α_n, α_1)
                /* envía los datos al worker siguiente */
                lab+k ← W_n , Q_n, α_n
            else
                W_n-k, G_n-k ← recibe de lab-k
                fusiona_grad(G_n-k , G_n, α_n, α_n−k)
                fusiona_fobjetivo(W_n-k , W_n , α_n, α_n−k)
                /* envía los datos al worker siguiente */
                lab+k ← W_n , Q_n, α_n
            end
        else
            envia_datos(SV_new , lab+k)
        end
        cont=cont-k
    end
    cont=int(numWrktemp/2);
    numWrktemp=cont;
end
if SV_old== SV_new
    then
        convergencia=true
    else
        SV_old= SV_new
        Enviar a todos los worker SV_new
end
end     /* fin de while ~convergencia */
end     /* del bloque spmd */
```

Algoritmo 5.2. Máquina Vector Soporte en Cascada.

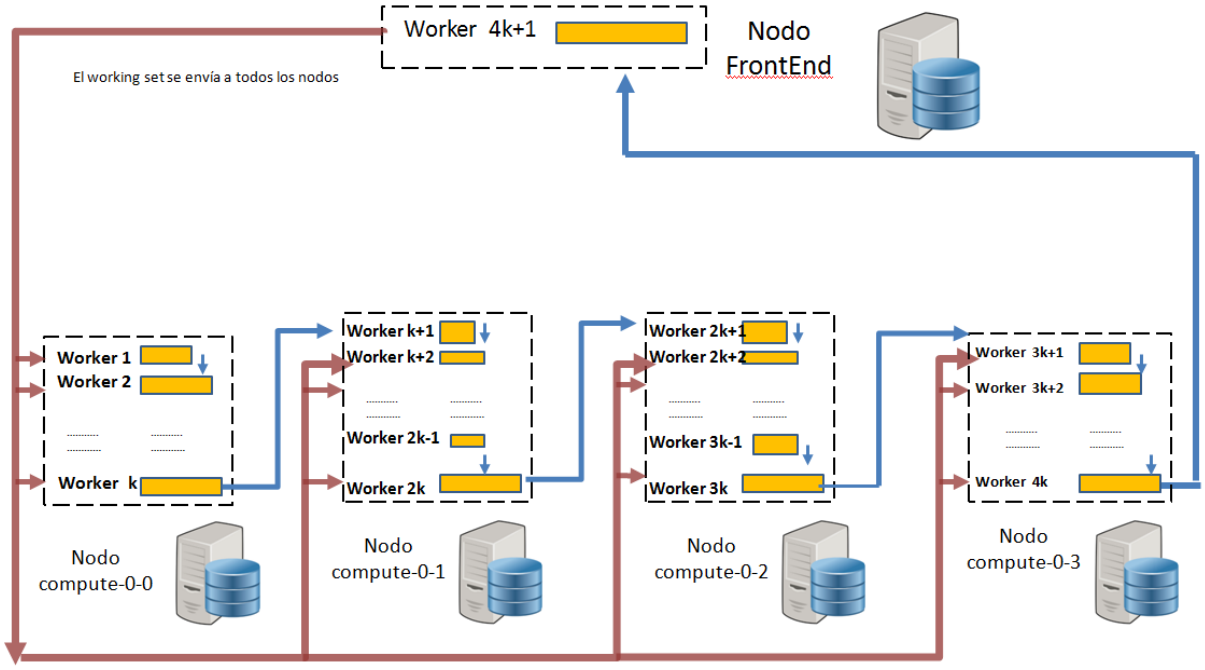


Figura 5.8. Ruta de datos del algoritmo Spread-Kernel SVM. Las líneas azules señalan la dirección de los resultados de cada *worker* que envía el woking set generado. Cada *worker* le envía a su *worker* contiguo y éste envía a su vez el mejor working set de los dos. Esta operación se repite hasta llegar al *worker* del front-end que envía a todos los *workers* el mejor working set.

5.5.3 Algoritmo smo-SVM Spread-Kernel.

La paralelización de la máquina soporte mediante el algoritmo Spread-kernel (figura 5.8) se realiza dividiendo la matriz Kernel entre los *workers* definidos en la agrupación paralela.

```
136 spmd
137     %Se reparten los datos de entrenamiento entre todos los workers
138     temp1=codistributed(data1');
139     [Lmin1,Lmax1]=globalIndices(temp1,2);
140     dataWorker1=getLocalPart(temp1');
141     [tam1, ~]=size(dataWorker1);
142      temp2=codistributed(data2');
143     [Lmin2,Lmax2]=globalIndices(temp2,2);
144     dataWorker2=getLocalPart(temp2');
145     [tam2, ~]=size(dataWorker2);
```

Figura 5.9. Datos de entrenamiento asignados a cada *worker*.

```
366         deleteIndex = leastIndices(end);
367         if deleteIndex == 0
368            subKernel_Index = 1 + max(subKernelIndices);
369         else
370            subKernel_Index = subKernelIndices(deleteIndex);
371            subKernelIndices(deleteIndex) = 0;
372         end
373         % aquí es donde se almacena el índice de la columna que se guardará.
374         subKernelIndices(idx2) = subKernel_Index;
375         % Se calcula la nueva columna kernel y la guardará  con el indice 'kernelColIdx'.
376         Kij=kernel_col(idx2,rbf_sigma,Lmin1,Lmax1,Lmin2, Lmax2,...
377                                                 dataWorker1, dataWorker2,tam1,kfun);
378         leastIndices(2:end) = leastIndices(1:end-1);
379         leastIndices(1) = idx2;
380         cont_Kij_NOAlmac=cont_Kij_NOAlmac+1;
381         if labindex==numlabs
382                 subKernel(:,subKernel_Index)=Kij;
383         end
```

Figura 5.10. Matriz Kernel asignada a cada *worker.*

Ya se ha hecho notar que la matriz kernel puede tomar grandes dimensiones. A modo de ejemplo, un primer conjunto de 1000 muestras de 14 atributos cada una y un segundo conjunto de 2000 muestras con 14 atributos generarían una matriz kernel cuadrada de 3000x3000. Por tanto un problema de 1 millón de muestras para cada conjunto con 24 atributos por cada muestra (algo habitual) no puede almacenarse en la memoria de un ordenador si se tiene en cuenta que además esa memoria debe almacenar y gestionar el sistema operativo, recursos del sistema, etc.

El método cascada visto anteriormente paralelizaba los datos de entrenamiento. El método a revisar ahora paraleliza la matriz kernel. Hay dos formas de cargar en memoria la parte de la matriz kernel que le corresponde a cada *worker*. Una de ellas es tenerla cargada en memoria y una vez arrancada la sesión paralela hacer el reparto correspondiente. La segunda forma sería asignar los datos proporcionales a cada *worker* y generar los índices mínimos y máximos de cada partición. La asignación de estos datos a cada *worker* se haría mediante la función Matlab *getLocalpart*( ) (figura 5.9).

La facilidad que da la función *codistribute* (Matlab) hace que no sea necesario calcular los límites de los tramos asignados de la matriz kernel. Más aún, tampoco es necesario tener los datos cargados en memoria porque mediante las instrucciones vistas en el capítulo dos, como pueden ser *matfile,* es posible cargar desde disco los datos necesarios para el cálculo de la columna Kernel. Esta última forma de trabajar podría será válida para el caso de tener un solo *worker*.

Un paso que merece atención es el cálculo de la columna Kernel. La memoria caché se puede dimensionar con el fin de no repetir los cálculos de la columna

Kernel. Anteriormente se ha indicado que los datos de entrenamiento están repartidos entre todos los *workers* (véase la figura 5.9). El algoritmo paralelizado centraliza en el *worker* gestor la recepción de los tramos de la columna kernel que cada *worker* ha calculado para después enviar esta columna a todos los *workers* del grupo. Una de las funciones importantes que debe ser ejecutada por todos los *workers* es la *kernel_col* que calcula la columna kernel, véase la figura 5.10. Es necesario que el *worker* en cuyos límites está la fila necesaria se la envíe al resto de *workers*. El esquema de transmisión se puede ver en la figura 5.11. Una vez recibida esa fila, cada *worker* calcula su tramo de columna y se la envía al *worker n*+1. Al final el *worker* gestor completa la columna, se la envía al resto y posteriormente la almacena en la memoria caché. Se utiliza parte de la memoria del nodo front-end para crear el entorno de cada *worker*. Esto es necesario para gestionar todo el entorno paralelo distribuido (figura 2.20). Aún así, hay memoria suficiente en el front-end para que el *worker* gestor sea quien gestione y almacene la memoria caché del algoritmo. Además de almacenar la memoria caché es necesario que el *worker* gestor gestione un índice de las columnas almacenadas para no tener que solicitar el recálculo de la columna en curso a cada *worker.*

Indudablemente la creación de la columna de forma colaborativa es mucho más rápida. Anteriormente se indicó que, la desventaja de este método, es que es necesario tener los datos replicados en el disco de cada nodo y esto puede llegar a ser un problema si no hay gran capacidad de disco. También hay que sumar la coordinación de todos los *worker.* Es decir, el sistema distribuido paralelo dispone de de una agrupación global de *workers* (véase figura 2.18 sección 2.8.3.2). Estos *workers* deben ser repartidos entre todos los usuarios del sistema que deseen ejecutar sesiones paralelas distribuidas. Por tanto cuando un usuario arranca su trabajo paralelo el sistema le asigna los *workers* solicitados y mediante la función *matlabpool* el sistema crea el entorno necesario para el trabajo. Cuando se crea este entorno el sistema debe replicar todos los datos necesarios para el trabajo.

Algoritmo 5.3: Smo- SVM Spread-kernel.

Datos:
N=numero de worker
conjunto de entrenamiento $D_{n,}$; n=1.....*N.*
numWrk=*N* /* número de workers */
max_diff=1e-5
numIter /* número de iteraciones para alcanzar convergencia */
Resultado:
convergencia=false;
spmd
D_n ← distributed (D) ; n=1.....*N* /* subconjunto de los datos training para el worker n */
convergencia=false
while ~*convergencia*
if labindex= *numWrk*
then
índice_i ← $maxGrad_{labindex}$()
/* recibe los datos de su worker antecesor
[*índice_i*] ← *recibe_datos*($maxGrad_{numWorker-1}$)
max([*índice_i*]) → *envia_datos*(W_1 ....W_{n-1})
/* no está almacenada la columna en memoria cache */
/* solo el front almacena filas por cuestión de memoria. */
/* El resto ayuda a calcular la columna */
if ~*columna*(*max*([*índice_i*]))
then
calcula_columna(*max*([*índice_i*]))
columna(*max*([*índice_i*])) → *envia_datos*(W_1 ....W_{n-1})
else
/* envía la columna a todos workers */
columna(*max*([*índice_i*])) → *envia_datos*(W_1 ....W_{n-1})
end
indice_j ← *calcula_indice_j* ()
[*índice_j*] ← *recibe_datos*(*índice_j*)
/* cada worker calcula índice j en función de la columna y el */
/* tramo de la diagonal correspondiente de la matriz kernel */
max([*índice_j*]) → *envia_datos*(W_1 ....W_{n-1})
/* no está almacenada la columna en memoria cache */
if ~*columna*(*max*([*índice_j*]))
then
calcula_columna(*max*([*índice_j*]))
columna(*max*([*índice_j*])) → *envia_datos*(W_1 ....W_{n-1})
else
/* envía la columna a todos workers */
columna(*max*([*índice_j*])) → *envia_datos*(W_1 ....W_{n-1})
end
[val1, val2] ← *calcula_val*(*columna*(*índice_j*), *columna*(*índice_i*))
if val1-val1)< *condiciones_KKT*()
then *convergencia*=**true**
end

```
        else
                /*  el worker envía su máximo */
                maxGrad_labindex()→ envia_datos(W_numWorker)
                [índice_i ] ←  recibe_datos(maxGrad_labindex-1)
                /*  no está almacenada la columna en memoria cache los worker */
                /*  solo tienen un índice que les indica si la columna está o no */
                if  ~columna(max([índice_i
                    then
                        calcula_columna(max([índice_i ]))
                        columna(max([índice_i ]))    → envia_datos(W1 ….Wn-1)
                    else
                        columna(max([índice_i ]))    → envia_datos(W1 ….Wn-1)
                end
        end
    end
end    /*  el bloque spmd */
```

Algoritmo 5.3. Algoritmo smo-SVM Spread-Kernel.

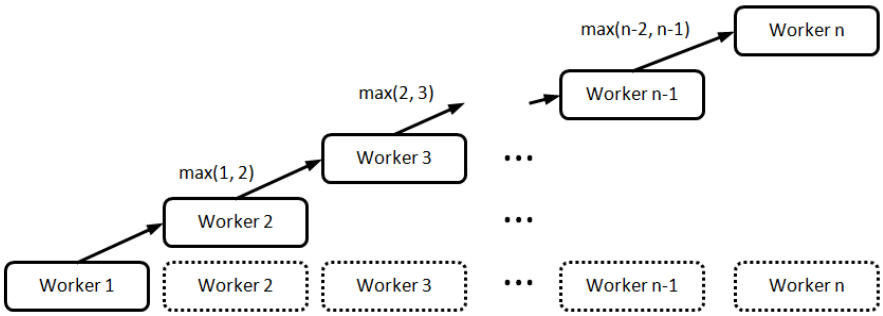


Figura 5.11. Método de envío del cálculo de la columna kernel de los *workers*. Cada *worker n-2* envía a su compañero *n-1* el resultado del índice máximo de cada columna. Al final el *worker* gestor envía a todos los *workers* el resultado.

Estos datos pueden ser programas, ficheros, etc.. Esto motiva que en ocasiones el arranque del entorno paralelo distribuido sea lento. En el capítulo dos se hizo referencia a la complejidad que supone la programación paralela. Se deben cuidar todos estos detalles pues una mala organización del arranque del entorno paralelo puede encarecer el tiempo de proceso.

Para la paralelización del algoritmo smoSVM Spread-Kernel existen tres elementos clave a tener en cuenta. Uno de ellos es la matriz kernel que ya se ha comentado y se ha indicado la forma de trabajar con ella. El segundo de ellos es el vector gradiente y el tercero es el vector de alfas. Éstos dos últimos deben ser calculados y estar actualizados por cada *worker* con los datos recibidos del *worker* gestor.

La figura 5.11 muestra como fluye toda la información entre todos los *workers* que forman la red para el cálculo de la columna kernel. Con esta forma de trabajo se elimina gran cantidad de tráfico en la red. Al final el *worker* gestor debe enviar a todos los *workers* la columna calculada. Con esta información también se calcula el vector gradiente que es enviado a todos los *workers*.

El tercer punto clave es el vector de los alfa necesario para calcular los vectores soporte. Este vector lo calculan de forma local todos los *workers* puesto que el *worker* gestor ha enviado previamente el working set a toda la agrupación de *workers*. Al final el vector alfa así como el vector gradiente están replicados de forma idéntica en todos los *workers*.

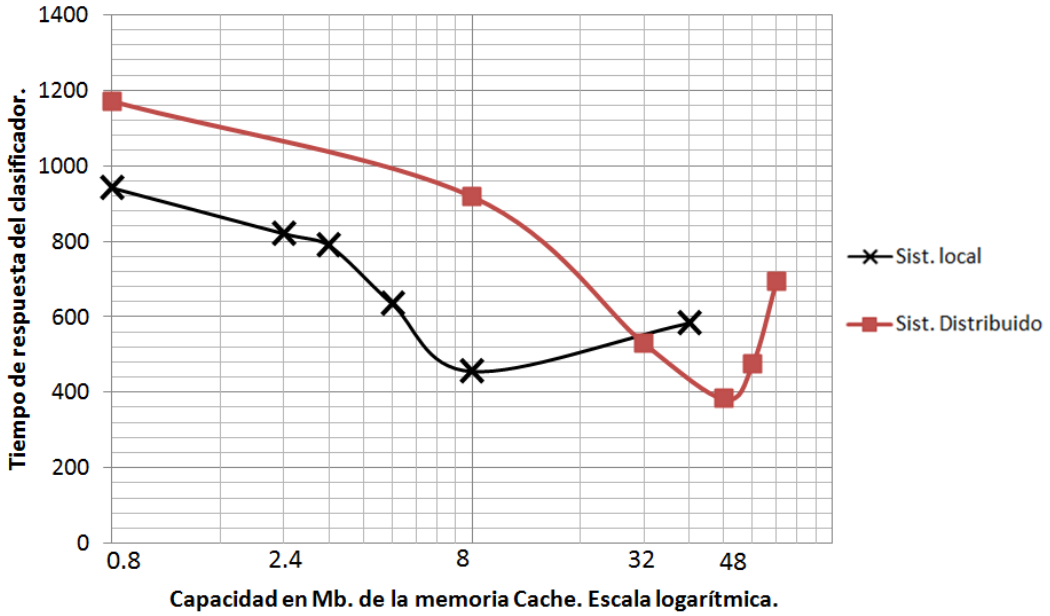


Figura 5.12. Tiempo de respuesta del algoritmo Spread-Kernel en función del tamaño de la memoria caché. Los tamaños han sido tomados para una grupo de cinco *workers* y tamaños de caché que van desde los 100 Mb. hasta los 7000 Mb.

La figura 5.12 muestra la evolución del tiempo de convergencia en función del tamaño de la caché. La escala de la gráfica es logarítmica para una mejor representación de los datos. Para la gráfica de color negro se puede observar que en el tramo de tamaño caché 2.4 Mb.-8 Mb. el tiempo de convergencia desciende. El motivo es que a partir de 8 Mb. existe una paginación excesíva y obliga al sistema a una mayor gestión de la memoria. La gráfica de color rojo perteneciente a otro sistema tarda más en ralentizarse y puede llegar a los 48 Mb. de memoria caché sin

aumentar el tiempo de respuesta. Los dos sistemas gestionan cinco *workers.* El primero es un entorno paralelo en local (un solo nodo) y el segundo es un entorno paralelo distribuido con cinco nodos. Se podría pensar en almacenar en cada *worker* su tramo de columna calculada pero esto conllevaría que habría menos espacio para los *workers* y el objetivo es aprovechar toda la capacidad de memoria que da un nodo para crear y gestionar sus propios *workers.*

5.5.4 Pruebas de escalabilidad.

Las pruebas de escalabilidad han sido realizadas con la base de datos *covtype.* Se trata de un conjunto de datos disponible en UCI [Frank 2010]. Este conjunto tiene un total de 581012 muestras y cada una de ellas tiene un total de 54 atributos. El motivo de utilizar este conjunto de datos es poder contrastar resultados con implementaciones realizadas con otros lenguajes. Los resultados se muestran para los tres algoritmos. Se ha comentado anteriormente que se dispone del *worker* gestor que es quien coordina el resto de *workers.* Para la implementación smo-SVM Spread-Kernel, el gestor debe almacenar las columnas kernel de la caché. Esto quiere decir que el resto de *workers* no almacenan columnas y el motivo es poder disponer de más *workers* en el resto de nodos. Por tanto la gestión de la caché se convierte en un punto crítico. En este caso si se tienen 581012 muestras, y como cada celda de la columna es de tipo *double* hace que su almacenamiento ocupe 4 Mb.

La figura 5.13 muestra la escalabilidad del clúster en función de la caché del *worker* gestor que está en el nodo front-end. A modo de ejemplo se comprueba que, para nueve *workers,* se alcanza un mínimo de 9439 segundos en crear el modelo. Los tiempos de proceso en función del número de *workers* para el análisis de la base de datos *covtype* se muestran en la tabla 5.1 donde se puede comprobar que el límite del modelo para esta base de datos está en 17 *workers.* En estudios similares pero diferentes implementaciones [Bouttou 2010, Ramírez 2014] los tiempos son proporcionalmente similares no obstante se debe tener en cuenta las arquitecturas paralelas vistas en el capítulo 2 apartado 2.4. Es decir, los clúster son diferentes en arquitectura y número de nodos.

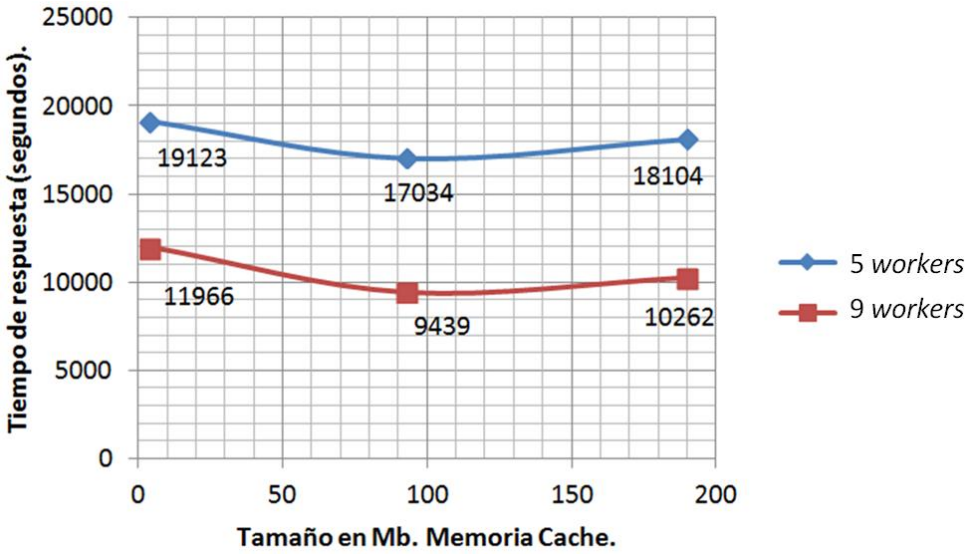


Figura 5.13. Análisis del tiempo de respuesta en función de la memoria caché del *worker* gestor para la clasificación de la base de datos covtype y el algoritmo smoSVM Spread-Kernel.

Num. *workers*	Tiempo de respuesta (seg.)
1	46897
3	22372
5	17034
9	12000
13	8624
17	7169
21	8893

Tabla 5.1. Tiempos de respuesta del clúster DIA en función del número de *workers*. Algoritmo Spread-Kernel.

La figura 5.14 muestra la comparativa de tiempo de los tres algoritmos. El eje de ordenadas está calibrado en escala logarítmica de base dos para hacerlo más legible. El algoritmo que mejor tiempos de respuesta obtiene es el algoritmo smo-SVM Spread-Kernel. El motivo principal es que este algoritmo no mantiene en memoria la matriz Kernel tan solo una mínima parte y solamente en el front-end.

En el caso del algoritmo Adatron, se deben mantener en memoria los datos de entrenamiento asi como la matriz kernel, esto hace que rápidamente el sistema se quede sin memoria y esté obligado a paginar continuamente lo que le obliga a un exceso de gestión que solo aporta desaceleración al sistema. Aunque el impacto no es tan alto, algo parecido ocurre con el algoritmo en cascada.

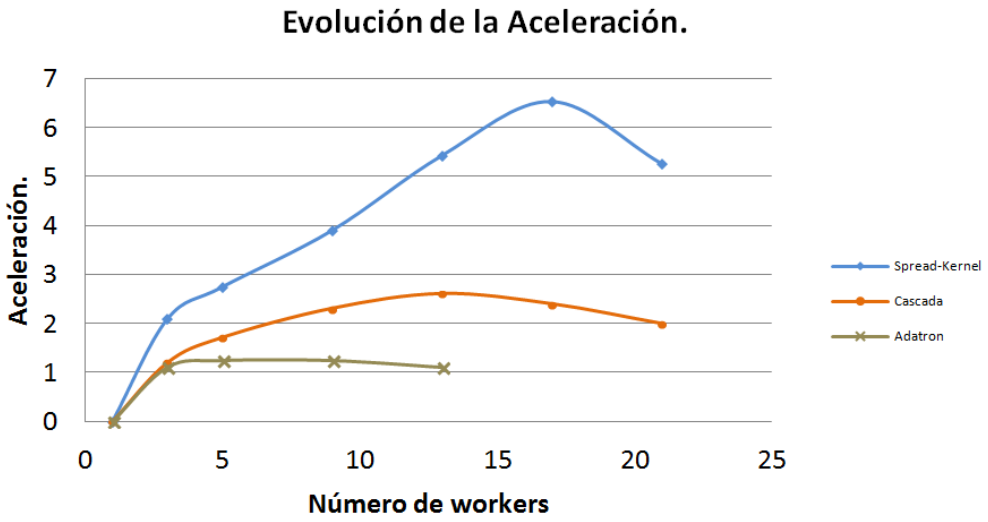


Figura 5.14. Análisis del Speedup y de los tiempos de respuesta de los algoritmos Spread-Kernel, Cascada y Adatron.

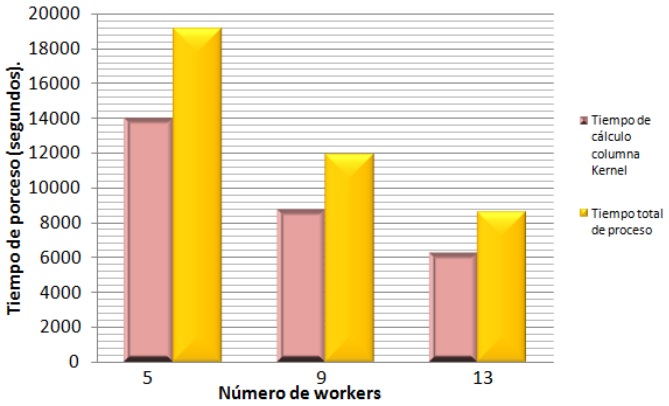


Figura 5.15. Tiempos dedicados a gestión, comunicaciones de red y cálculo.

En la figura 5.15 se puede apreciar el tiempo dedicado al cálculo y el tiempo de gestión. Este último índice engloba los tiempos de comunicaciones de red y de gestión del *worker* gestor. Estos datos están recogidos con un tamaño de memoria

caché de 4 Mb.. La columna amarilla es el tiempo total del proceso y la columna morada es el tiempo dedicado al cálculo. Se observa que, a medida que hay más *workers* en el proceso, la diferencia de las dos columnas se hace más pequeña. El motivo es que al aumentar el número de *workers* el número de iteraciones disminuye lo que conlleva además un menor tiempo de gestión.

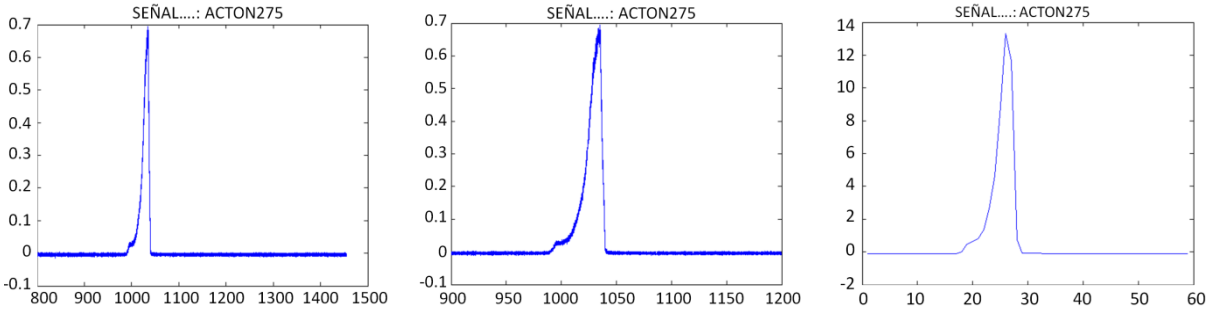
5.6 Selección de características.

La Tabla 5.2 muestra las señales de evolución temporal de la base de datos de TJ-II utilizadas como entrada al clasificador. Cada señal se obtiene de un sensor diferente y muestra la medición de una característica física del plasma así como su comportamiento.

	long. min.	long.max	Número muestras	Clases de señales
ACTON275		65536	1157	Señal espectroscopica (CV)
ECE7		61440	1157	Emisión ciclotrónica
DENSIDAD2	32755	32768	1083	Línea media de densidad electrónica
BOL5		65536	1157	Señal Bolométrica
RX306	45208	50000	1047	Rayos x suave

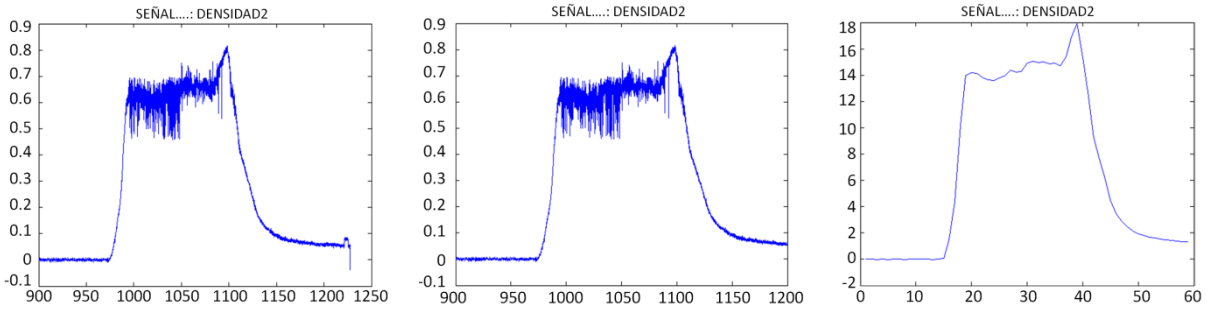
Tabla 5.2. Datos referentes a las descargas del TJ-II.

También en la tabla 5.2 se puede ver la longitud en puntos de cada señal. Todas las muestras de la clase ACTON275 tienen la misma longitud. Es decir, todos los ficheros que contienen las muestras de esta clase tienen la misma longitud. Ocurre lo mismo con las clases ECE7 y BOL5. Sin embargo las muestras de las clases RX306 y DENSIDAD2 no tienen la misma longitud. Para estos dos casos los ficheros que contienen las muestras son de longitud variable. En la tabla 5.2 se pueden observar estas diferencias. El objetivo por tanto es encontrar un punto común a todas las señales donde se concentre el mayor contenido de información. La figura 5.16 muestra el detalle de cada señal en su rango temporal. Cada fila corresponde a una clase de señal. Si se observa la figura 5.16(a) de cada señal se puede ver el comportamiento de la señal en su rango temporal.



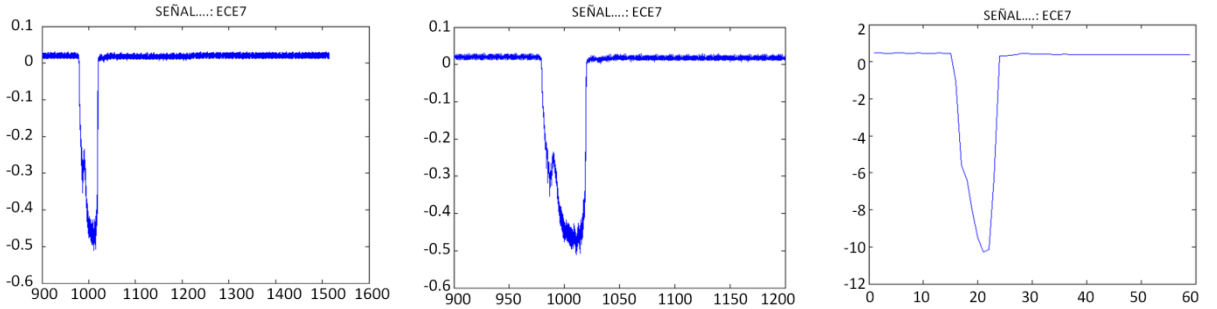
(a) (b) (c)

Señal ACTON275.



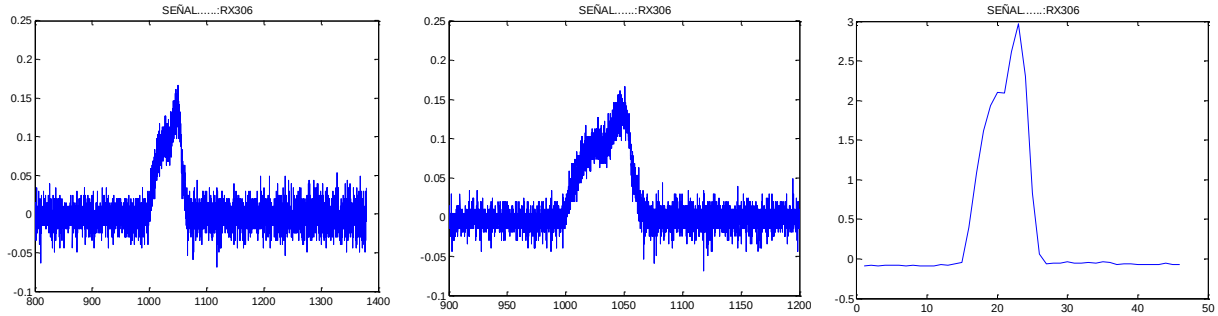
(a) (b) (c)

Señal DENSIDAD2.



(a) (b) (c)

Señal ECE7.



(a) (b) (c)

Señal RX306.

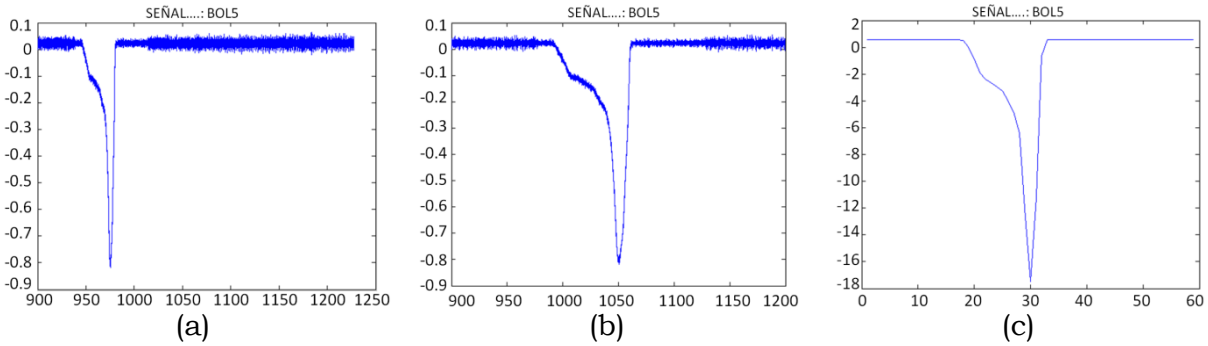


Figura 5.16. Formas de onda del TJ-II.

Se observa también que, a partir de los 900 msg., la gráfica muestra variaciones importantes. Se puede decir que, a partir de los 900 msg. todas las señales comienzan a tener cambios en su comportamiento. Algunas clases tienen información pasados los 1400 msg., sin embargo otras solo llegan hasta los 1200 msg., por tanto la selección de datos de cada señal se toma desde los 900 msg. hasta los 1200 msg.. En la figura 5.16(b) de cada señal se puede observar como se han normalizado todas las longitudes de cada clase. Posteriormente a estas señales se le debe realizar un preprocesamiento debido a su alta dimensionalidad. Para la reducción de dimensionalidad se utiliza la transformada wavelet. La transformada wavelet tiene diferentes aplicaciones, pero para este caso se utiliza con el objetivo de reducir la dimensionalidad de los datos pues las muestras tienen unas longitudes muy altas. Se utilizó la familia de wavelets Haar con diferentes niveles de reducción. En el anexo A de la memoria se detalla el proceso de reducción de dimensionalidad mediante la transformada wavelet.

Si bien el nivel 9 de la transformada wavelet Haar [Haar 1910] fue finalmente el seleccionado, previamente se utilizaron diferentes niveles para contrastar resultados. Concretamente los niveles de la transformada wavelet aplicados fueron desde el nivel 5 hasta el nivel 10. Los experimentos se realizaron variando los parámetros los parámetros C y sigma de la máquina SVM para los tipos de funciones Kernel Lineal, Polinómica y RBF. Una vez reducida la dimensionalidad de cada señal, se procede a realizar la clasificación de las señales. Para ello utilizó el 70% de muestras de cada señal para entrenamiento y el 30 % para validación.

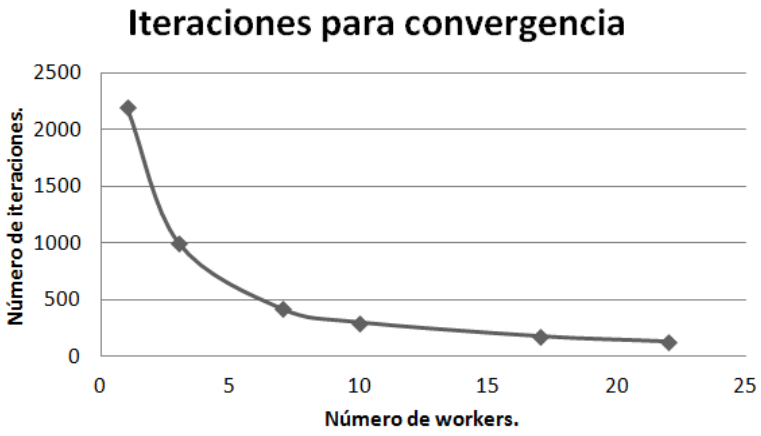


Figura 5.17. Número de iteraciones necesarias para conseguir convergencia con Algoritmo smo-SVM Spread-Kernel.

5.7 Resultados.

En el capitulo 3 se detalló que el algoritmo SVM realiza la búsqueda del conjunto de trabajo óptimo hasta llegar a la convergencia. Ésta se consigue cuando se encuentra el conjunto de trabajo que más se aleja de las condiciones de KKT. Es decir, se violan las condiciones KKT con un error menor que el definido al principio del proceso. Al estar repartidos los datos de entrenamiento entre todos los *workers*, cada *worker* envía su conjunto de trabajo óptimo y por tanto se llega a la convergencia con menos iteraciones que en el proceso secuencial. En la figura 5.17 se pueden ver las iteraciones necesarias mediante el algoritmo smo-SVM Spread-Kernel para los datos de entrenamiento de formas de onda del TJ-II. Cuando se dispone de un solo *worker*, se necesitan 2200 iteraciones para alcanzar la convergencia. Este número va bajando hasta llegar a las 138 iteraciones cuando el clúster tiene 22 *workers* operativos.

Esta reducción considerable de iteraciones no implica directamente reducción de tiempo pues a mayor número de *workers* mayor gestión para el envío de datos y tiempo de comunicaciones. Por estos motivos no siempre se consigue optimización.

No obstante una mayor gestión de la memoria unido a un mayor tiempo de transmisiones empaña en parte la ventaja adquirida por un menor número de iteraciones.

El resultado del tiempo de proceso para el análisis con el método uno contra todos (uct) se muestra en la figura 5.18. Se puede observar que, mediante el

algoritmo smo-Spread-Kernel, se consigue ganancia de velocidad hasta un número de 6 *workers.* A partir de ese número el clasificador pierde velocidad de respuesta y el motivo es porque el conjunto de datos que se analiza no es lo suficientemente grande para obtener beneficios que se obtendrían al aumentar el número de *workers.*

Como ya se ha apuntado, se utilizaron tres tipos de función Kernel para entrenar el clasificador: lineal, polinómica y de base radial (RBF). Para esta última función, el estudio se realizó con los valores $C \in [10^{-1}, \ldots, 10^{8}]$ y sigma $\in [10^{-1}, \ldots, 10^{3}]$. Los valores finales de C y sigma que dieron mejores resultados son los que aparecen en la tabla 5.3. Además se utilizaron diferentes niveles de descomposición de la transformada wavelet (desde el nivel 5 hasta el nivel 10). Los resultados que se aportan son los realizados con nivel 8 y 9 que generan unas muestras de dimensionalidad 123 y 79 caracteristicas respectivamente cada una de ellas.

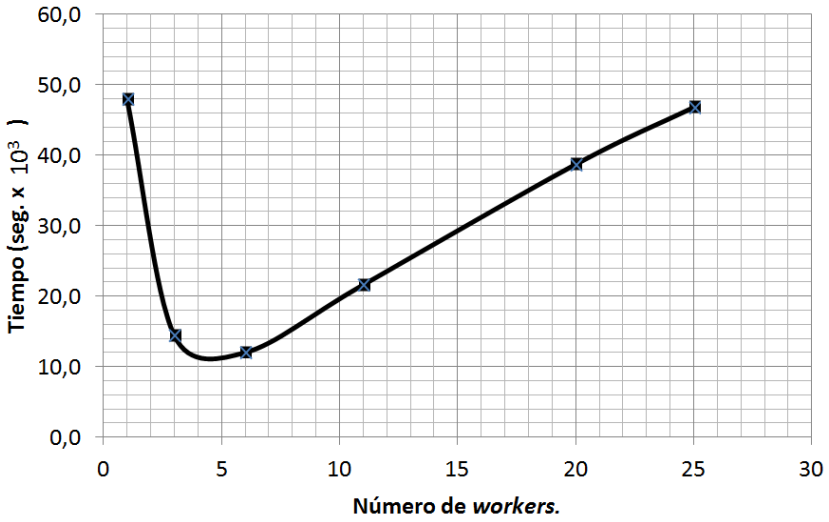


Figura 5.18. Tiempo de proceso para la identificación de formas de onda del TJ-II.

Los mejores resultados se obtienen para la función kernel RBF y los análisis con un nivel de descomposición 8 son ligeramente mejores que para el nivel 9. No obstante los tiempos aumentan considerablemente al tener muchos más datos que tratar lo que hace que el nivel 9 sea más idóneo.

Los métodos utilizados para el clasificador han sido dos:

1. Algoritmo SVM y método de análisis uno-contra-uno.
2. Algoritmo SVM y método de análisis uno-contra-todos.

La función kernel utilizada para los dos algoritmos ha sido la de base radial (RBF) por ser la que mejor resultados presenta. Las tablas 5.3, 5.4 y 5.5 muestran los datos obtenidos. Concrétamente el método uno contra todos es el que proporciona mejores índices además de mejores tiempos de respuesta.

Función Kernel	Transformada Wavelet nivel 8	Transformada Wavelet nivel 9
	Acierto (%)	Acierto (%)
Lineal	87,282	85,952
Polinómica	90,836	87,762
RBF (C=10000, sigma=100)	96,234	95,938

Tabla 5.3. Tasa de acierto con diferentes funciones Kernel.

Función Kernel RBF		Transformada Wavelet nivel 8	Transformada Wavelet nivel 9
C	Sigma	Acierto (%)	Acierto (%)
1	6	91,267	90,438
10	6	92,423	92,031
100	2	95,343	94,023
1000	6	96,986	96,414
10000	6	97,324	96,613
100000	12	96,975	95,617
1000000	4	95,837	94,223

Tabla 5.4. Tasa de acierto para SVM y método uno contra todos.

Función Kernel RBF		Transformada Wavelet nivel 8	Transformada Wavelet nivel 9
C	Sigma	Acierto (%)	Acierto (%)
1	6	88,690	87,982
10	6	88,876	87,523
100	2	89,635	88,654
1000	6	89,708	89,726
10000	6	91,965	90,263
100000	12	87,265	88,019
1000000	4	86,034	85,534

Tabla 5.5. Tasa de acierto para SVM y método todos contra todos.

Capítulo VI.
Predictor probabilístico para imágenes del sistema de esparcimiento Thomson del TJ-II.

6.1 Introducción.

Existen muchos algoritmos de aprendizaje automático que permiten hacer estimaciones de clasificación. Sin embargo muchos de ellos carecen de una medida de confianza para evaluar el error en la predicción. Para un nuevo *ejemplo*, sería muy deseable disponer de un algoritmo que proporcionase una distribución de probabilidad para cada posible etiqueta.

Este capítulo hace especial énfasis en un tipo de algoritmo que proporciona información adicional acerca de cómo es de fiable cada predicción. El capítulo comienza con el planteamiento de los predictores probabilísticos para posteriormente tratar los predictores conformales centrándose en el predictor Venn (PV). El PV es el que se ha utilizado para crear un modelo de clasificación con medidas de confianza. El PV requiere un esfuerzo computacional alto y esto es lo que motivó que el modelo tuviera que ser paralelizado. Finalmente se muestran los resultados del modelo PV paralelo aplicados a las imágenes del sistema de esparcimiento del TJ-II.

6.2 Predictores Probabilísticos. Planteamiento del problema.

Los *ejemplos* consisten en pares de *objetos* y *etiquetas* que son los que componen la información derivada de un espacio real.

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots \dots \dots, (x_i, y_i), \dots \dots$$

Cada ejemplo (x_i, y_i), está compuesto de un objeto x_i y su etiqueta y_i. El objeto x_i es un vector de atributos compuesto de números reales que cuantifican una medida del mundo real. En otras palabras $x_i \in \mathrm{R}^d$. La etiqueta y_i es un número real y pertenece a un espacio medible $Y \subset \mathrm{R}$ al que se denomina espacio de etiquetas. Una notación más compacta para definir un ejemplo es la dada por $z_i = (x_i, y_i)$, de tal

forma que todos los elementos pertenecen a un espacio medible Z=X x Y denominado *espacio de ejemplos.*

Inicialmente se dispone de un conjunto de ejemplos $(z_1, z_2, \ldots\ldots, z_{n-1})$ que contiene *n*-1 ejemplos conocidos, denominado *conjunto de entrenamiento*. Si se recibe un nuevo objeto x_n, se le denomina *ejemplo de test* y al conjunto de ejemplos de test se le denomina *conjunto de test*. El objetivo es conseguir una etiqueta $\hat{y}$ para este nuevo ejemplo. Más aún, el objetivo es conseguir alguna estimación cuando la etiqueta es verdadera, es decir $\hat{y}_n = y_n$ cuando es la verdadera etiqueta del objeto x_n.

Se asume que el espacio de objetos *X* es no vacío y que el espacio de etiquetas *Y* tiene por lo menos dos elementos diferentes. De acuerdo a la cardinalidad del espacio de etiquetas, el problema puede ser dividido en dos tipos: clasificación y regresión. Se denomina un problema de clasificación cuando la cardinalidad $K=|Y|$ del espacio de etiquetas es finito, donde ésta puede ser representada como un conjunto de números $Y=\{0,1,2,\ldots\ldots,K-1\}$ sin tener en cuenta los valores originales del espacio de etiquetas. Si la cardinalidad del espacio de etiquetas es infinito, normalmente las etiquetas son valores reales, y se habla de problemas de regresión.

Para construir un predictor probabilístico es necesario realizar unas hipótesis iniciales. Básicamente, cuando se aprende desde el espacio de ejemplos, el deseo es que esté gobernado por unas leyes a través de las cuales transcurre el proceso de aprendizaje. Si el entorno cambia continuamente será difícil aprender algo sobre él. Tradicionalmente, el aprendizaje consiste en elegir los ejemplos de forma aleatoria ajustada a una distribución de probabilidad *Q* de un espacio de ejemplos *Z*. En otras palabras, todos los ejemplos comparten la misma distribución de probabilidad *Q* y son independientes uno de otro. Se dice entonces que esos ejemplos son *independientes* e *idénticamente distribuidos* (iid). Esto se llama *asunción de aleatoriedad* [Vovk 2005].

6.2.1 Predictores conformales.

Los predictores conformales se pueden utilizar con diferentes métodos de clasificación o regresión, por ejemplo las redes neuronales, las redes bayesianas, los árboles de decisión y por supuesto las máquinas de vector soporte. El trabajo que se ha desarrollado y que se ha implementado en el cluster de supercomputación DIA ha sido utilizando el algoritmo de las máquinas de vectores soporte visto en el capítulo anterior y al que se le han aplicado técnicas de programación paralela.

Los predictores conformales fueron introducidos por Vovk [Vovk 2005] donde se distingue la predicción inductiva de la transductiva (véase figura 6.1). La predicción inductiva se realiza mediante una regla general sin embargo en la predicción transductiva el camino es más directo en el sentido que se pasa directamente de los ejemplos a la predicción del nuevo ejemplo. A partir del método de clasificación que se esté utilizando, el modelo construye una medida de *no-conformidad.* Esta medida proporciona una idea sobre la diferencia que hay entre la nueva muestra analizada frente al resto de muestras precedentes ya analizadas. El *algoritmo conformal* convierte esa medida de *no-conformidad* en regiones de predicción.

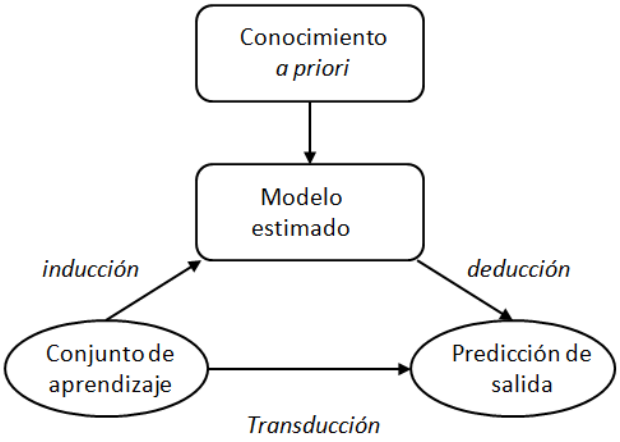


Figura 6.1. Proceso de clasificación.

En un problema de clasificación, tan solo se desea predecir la etiqueta verdadera y_n para un nuevo objeto x_n donde previamente se haya resuelto una secuencia de objetos del *espacio de ejemplos*.

Para una secuencia genérica $(x_1, y_1), \ldots\ldots, (x_{n-1}, y_{n-1}) \in Z^*$ y un nuevo objeto $x_n \in X$ la función que se necesita se puede representar como:

$$D: Z^* \times X \rightarrow Y. \qquad (6.1)$$

La función 6.1 define un predictor simple que determina $D(x_1, y_1, \ldots. x_{n-1}, y_{n-1}, x_n) \in Y$ a y_n como una etiqueta verdadera de x_n. Una noción más avanzada de predicción podría ser complementar cada posible etiqueta con un grado de confianza en lugar de elegir un solo elemento de Y como predicción. Se intenta por tanto determinar un subconjunto de Y con un grado predefinido de confianza. Para cada posible etiqueta, si el grado predefinido de confianza es mayor

que su grado complementario de confianza, se incluirá esa etiqueta en el subconjunto. Se puede asegurar, dentro de un grado predefinido, que la etiqueta verdadera y_n se encuentra en ese subconjunto. Cuanto mayor sea el subconjunto, mayor será el grado de confianza puesto que incluirá mayor número de etiquetas posibles. Este tipo de predictor se llama *predictor de confianza,* este a su vez da como resultado un *conjunto de predicción* [Gammerman 2006].

Un predictor de confianza tiene un parámetro adicional $\varepsilon \in (0,1)$ conocido como *nivel de confianza* que es el valor complementario de 1- ε que se define como *nivel de credibilidad.* Según estas definiciones, dado un conjunto de entrenamiento, un nuevo objeto y un nivel de confianza ε, se define un predictor de confianza Γ como:

$$\Gamma : Z^* \text{ X x } (0,1) \rightarrow 2^Y \tag{6.2}$$

Donde 2^Y es el conjunto total de Y, es decir, la totalidad de los subconjuntos. Este predictor genera un subconjunto de salida:

$$\Gamma^{\varepsilon}(x_1, y_1, \ldots . x_{n-1}, y_{n-1}, x_n) \tag{6.3}$$

Este subconjunto debe satisfacer:

$$\Gamma^{\varepsilon_1}(x_1, y_1, \ldots . x_{n-1}, y_{n-1}, x_n) \subseteq \Gamma^{\varepsilon_2}(x_1, y_1, \ldots . x_{n-1}, y_{n-1}, x_n) \tag{6.4}$$

Para todo $\varepsilon_1 \geq \varepsilon_2$. Con un nivel de credibilidad de 1- ε, Γ predice y_n que a su vez está en su conjunto de predicción:

$$y_n \in \Gamma^{\varepsilon}(x_1, y_1, \ldots . x_{n-1}, y_{n-1}, x_n) \tag{6.5}$$

Un predictor de confianza tiene dos propiedades importantes: *validez* y *eficiencia*. Dado un nivel de confianza ε, si la frecuencia relativa de errores que realiza el predictor de confianza no excede de ε, entonces para cada ε determinado se dirá que es un predictor *válido*. Esto quiere decir que el número de errores o bien la tasa de errores se pueden controlar dentro de una tolerancia aceptable por medio del nivel de confianza ε que se ha definido. La otra propiedad, la *eficiencia*, significa que el predictor proporciona conjuntos de predicción informativos. Un conjunto de predicción con un número menor de etiquetas es más informativo que un conjunto de predicción con mayor número de etiquetas. Esto obliga al predictor de confianza a generar un conjunto de predicción lo más pequeño posible. Estas dos propiedades

y especialmente la *validez* es lo que hace que estos predictores sean más ventajosos sobre otros tipos de algoritmos como pueden ser los predictores simples.

Antes de dar una definición más formal de un predictor conformal, se debe definir el concepto de bolsa o también llamado conjunto múltiple. Una bolsa es una colección de elementos donde el orden de éstos carece de valor, y que es similar a un conjunto. Un conjunto es una colección de objetos no repetidos y no ordenados. La diferencia entre bolsa y conjunto es que en una bolsa los elementos pueden estar repetidos. El tamaño n de una bolsa indica el número de elementos que contiene. Puesto que los elementos pueden estar repetidos, habrá que indicar cuáles de ellos son los repetidos y cuantas veces lo están. Para referirse a una bolsa con n elementos se realiza mediante al notación $\{z_1, z_2, \ldots., z_n\}$ donde alguno de ellos pueden ser idénticos. A pesar de que el orden es irrelevante, es necesario marcarles para posteriormente poder referirse a ellos. Se escribe como $Z^{(n)}$ al conjunto de n elementos de un espacio medible Z y se escribe como $Z^{(*)}$ al conjunto de todas las bolsas de elementos de Z. La idea general de un predictor conformal (PC) es asumir la etiqueta de un nuevo objeto x_n de una de las posibles etiquetas y. Para ello se debe medir (mediante una medida de no-conformidad) cuanto de satisfactorio es el ajuste del nuevo par de ejemplo (x_n, y) a los ejemplos ya conocidos $\{z_1, z_2, \ldots. z_{n-1}\}$. Por tanto se debe evaluar con todas las posibles etiquetas para calcular la medida de no-conformidad para cada etiqueta. De manera más formal una medida de no-conformidad es un método que evalúa como de diferente es el nuevo objeto respecto a los ejemplos anteriores. Esta medida de no-conformidad mapea cada nuevo ejemplo y cada bolsa de antiguos ejemplos mediante una calificación numérica de no-conformidad .

$$\text{A: } Z^{(*)} \text{x Z} \rightarrow \text{R} \tag{6.6}$$

Los algoritmos utilizados por los PC se llaman *algoritmos subyacentes* del PC. Se pueden relacionar de forma separada las medidas de no-conformidad con diferentes tipos de bolsa de ejemplos. El conjunto de medidas de no-conformidad A_n para cada n =1, 2, …., se puede definir de la siguiente forma:

$$A_n: Z^{(*)} Z^{(n-1)} \text{x Z} \rightarrow \text{R} \tag{6.7}$$

Donde n es el tamaño de la bolsa. El resultado de no-conformidad para cada z_i en la bolsa $\{z_1, z_2, \ldots. z_n\}$ es:

$$\alpha_i := A_n(\{ z_1, z_2, \ldots . z_{i-1}, z_{i+1}, \ldots, z_n \}, z_i) \tag{6.8}$$

El valor numérico de α_i da una idea de lo diferente que es el nuevo ejemplo z_i comparado con el resto de ejemplos. Por ese motivo se utilizan los p-valores a modo de indicador de cuan diferente es el resultado de no-conformidad α_i comparado con otros. Se puede conseguir mediante el cálculo de la proporción de los resultados de no-conformidad que son más grandes o igual a α_i de entre todos los resultados. La definición de un p-valor para un ejemplo dado z_i se especifica en (6.9).

$$p := \frac{\left|\{j = 1, \ldots . . n : \alpha_j \geq \alpha_i\}\right|}{n} \tag{6.9}$$

Donde p es el p-valor para el ejemplo z_i e indica de forma cuantitativa lo diferente que es z_i del resto de ejemplos. Si α_i es comparado consigo mismo entonces el numerador es 1. Por tanto el rango de valores para los p-valores está

Algoritmo 6.1: Predictor conformal

Datos: conjunto de entrenamiento $Z^{(n-1)}$, objeto a analizar x_n, nivel de confianza ε
Resultado: Γ^{ε}

```
Γ^ε ← Ø;                          /* conjunto de predicción inicial */
K ← |Y|;                          /* obtener tamaño del espacio de etiquetas */
for y = 0 to K-1 do               /* intentar para cada etiqueta */
    for i= 1 to n do              /* por cada etiqueta calcular todos los resultados NCM */
        α_i ← NCM({(x_1,y_1),....,(x_{i-1},y_{i-1}),(x_{i+1},y_{i+1}),.....,(x_n,y)},(x_i,y_i))
                                  /* calcular el resultado de no-conformidad para cada ejemplo */
    end
    c← 0;                         /* contador inicial a 0 */
    for i = 1 to n do             /* cuenta de cuantas veces NCM ≥ α_n */
        if α_i ≥ α_n
          then
            c← c+1;
        end
    end
    /* Cálculo de la frecuencia de las etiquetas*/
    p_i ← calculaFrecuencia(C);
    if p_i ≥ ε
      then
        C ← añadeAlconjunto(C,y);
        /* añadir etiqueta en curso al conjunto de predicción */
    end
end
return Γ^ε
```

Algoritmo 6.1. Predictor conformal.

comprendido entre $\frac{1}{n}$ y 1, indicando que el ejemplo z_i es muy no-conforme (muy diferente) si el *p*-valor es muy pequeño o bien muy conforme (similar) si el *p*-valor es grande.

A continuación se detalla una definición más formal de un predictor conformal. Dado un conjunto de entrenamiento $\{ z_1, z_2, \ldots . z_{n-1} \}$, un nuevo objeto, x_n y un nivel de confianza ε, el predictor conformal Γ determinado por una medida de no-conformidad A_n proporciona un conjunto de predicción dado por (6.10).

$$\Gamma^{\varepsilon}(x_1, y_1, \ldots . x_{n-1}, y_{n-1}, x_n) \coloneqq \{y \in Y: p_i > \varepsilon\} \tag{6.10}$$

Donde p_i es el *p*-valor del par (x_n, y) y se define como:

$$p \coloneqq \frac{|\{i = 1, \ldots . . n: \alpha_i \geq n\}|}{n} \tag{6.11}$$

Donde

$$\alpha_i = A_n(\{ (x_1, y_1), \ldots ., (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots . ., (x_n, y) \}, (x_i, y_i)) \tag{6.12}$$
$$\alpha_n = A_n(\{ (x_1, y_1), \ldots ., (x_{n-1}, y_{n-1}), (x_n, y) \})$$

Se puede tener una confianza de 1-ε que la etiqueta verdadera y_n de x_n esta incluida en el conjunto de predicción de Γ^{ε}.

Al ser una clase de predictores de confianza, los predictores conformales también satisfacen el criterio de validez. El algoritmo 6.1 es el pseudocódigo de un predictor conformal donde la función NCM calcula los resultados de no-conformidad realizados por los algoritmos subyacentes.

6.2.2 Predictores Venn.

Los predictores conformales proporcionan *p*-valores que dan una idea de cómo son de fiables las estimaciones de las predicciones, pero no son probabilidades directas. En esta sección se introducirá un nuevo tipo de predictor conformal que transforma las predicciones en probabilidades, lo que hace de él que sea más intuitivo.

Se trata de una variedad de algoritmos encuadrados en el contexto del aprendizaje automático y que pueden construir predicciones probabilísticas. A este tipo de algoritmos se les denomina *algoritmos probabilísticos.* Los predictores Venn proporcionan distribuciones de probabilidad basadas en que los ejemplos son

independientes e idénticamente distribuidos (iid). Estos predictores fueron introducidos por Vovk en [Vovk 2003] y tratados con detalle en [Vovk 2005].

Además, los predictores Venn generan para cada etiqueta predicha, un conjunto de distribuciones de probabilidad para la predicción de una etiqueta, y puede ser interpretado como un intervalo acotado de probabilidades donde la etiqueta es verdadera.

Una introducción más detallada de los predictores Venn debe comenzar por una serie de hipóteis adicionales a las comentadas en la sección 6.2 para los predictores conformales. Adicionalmente se deben tener en cuenta:

1. El espacio de etiquetas Y debe ser finito, lo que significa que solo se puede utilizar para un problema de clasificación.
2. La secuencia de ejemplos z_1, z_2,.... es finita en lugar de continuar indefinidamente. Se asume que el número de ejemplos es *N*.

6.2.2.1 Predicción probabilística mediante validación on-line.

La validación on-line es un método muy útil para demostrar las ideas que dirigen la calibración de la validez de las predicciones probabilísticas. Para ver como se realiza una validación on-line se procede como sigue. Se parte de un problema de clasificación binario, donde las posibles etiquetas son $\{0,1\}$ y un predictor probabilístico que determina una probabilidad p_n para la predicción y_n =1. Se asume además que se están observando ejemplos en secuencia, lo que significa que se hace la predicción probabilística p_n después de haber observado un nuevo objeto x_n y antes de que se haya observado la verdadera etiqueta de ese objeto x_n. El método descrito es el protocolo a seguir para hacer una validación on-line.

Para conocer si la probabilidad predicha por p_n tiende a ser muy alta o bien a ser muy baja, se puede realizar mediante la comparación del promedio de las probabilidades predichas con la frecuencia global de "1" entre los ejemplos *N*. El promedio de la probabilidad es el siguiente:

$$\bar{p}_n := \frac{1}{N}\sum_{n=1}^{N} p_n \qquad (6.13)$$

También, la frecuencia global de "1" es:

$$\bar{y}_n \coloneqq \frac{1}{N}\sum_{n=1}^{N} y_n \tag{6.14}$$

Si $\bar{y}_n \approx \bar{p}_n$ se dice que las predicciones son de *promedio imparcial*. Si $\bar{p}_n$ es muy diferente de $\bar{y}_n$ se dice entonces que son de *promedio parcial.* En una versión más avanzada, es posible analizar la calibración de un subconjunto de n ejemplos para los cuales p_n está cerca de un valor dado p^*. Se puede entonces comparar p^* con la frecuencia de "1" en este subconjunto que en este caso seria $\bar{y}_n(p^*)$. Si se confirma

$$\forall p^*, \bar{y}_n(p^*) \approx p^* \tag{6.15}$$

se dice que p_n está *bien calibrado,* en otras palabras, se dice que tiene una *justificación frecuentista*. John Venn [Venn 1866] fue uno de los primeros autores que escribió sobre la *probabilidad frecuentista*.

6.2.2.2 Taxonomías en Predictores Venn.

Para tener una buena calibración no es suficiente realizar predicciones probabilísticas. Si se asume que y_n es igual a 1 si n es impar y 0 en otro caso, por tanto, $\bar{y}_n \approx 0.5$. Según esto se puede decir que $p_n = 0.5$, en este caso las predicciones probabilísticas no tendrían sentido. Para superar esto, lo que se necesita además de una buena calibración es una "alta resolución". Este término sugiere introducir un procedimiento que se utiliza en los predictores Venn.

El predictor divide los objetos en categorías y utiliza las frecuencias en la categoría que contiene x_n como predicciones de probabilidad p_n. Cuanto más números sean las categorías mayor será la resolución y más útil será la predicción probabilística. Volviendo al ejemplo inicial, se pueden clasificar todos los ejemplos en dos categorías en lugar de una sola. Una de ellas para los números pares y la otra para los impares. La predicción será por tanto "1" para los impares y "0" cuando los números caigan en otras categorías.

Se asume que se dispone de un problema de clasificación binario $\lbrace z_1, z_2, \ldots . z_{n-1} \rbrace$, un nuevo objeto x_n y un espacio de etiquetas es Y={0,1}. Una *taxonomía* es un procedimiento que divide los ejemplos en categorías utilizando un algoritmo subyacente. Son varios los algoritmos que pueden utilizarse aunque, en este trabajo se han utilizado las máquinas de vector soporte. Según esto se

puede interpretar a los predictores Venn como un entorno de aprendizaje automático en lugar de entenderlo como un único algoritmo.

Dada una *taxonomía* A_n para $\lbrace z_1, z_2, \ldots . z_{n-1}, (x_n, \mathrm{y}) \rbrace$ donde $y \in \mathrm{Y}$. La categoría τ_i asignada al ejemplo z_i por A_n está definida por (6.16).

$$\tau_i := A_n(\lbrace z_1, z_2, \ldots . z_{i-1}, z_{i+1}, \ldots \ldots, z_{n-1}, (x_n, \mathrm{y}) \rbrace, \mathrm{z_i}) \tag{6.16}$$

Los ejemplos z_i y z_j están en la misma categoría si y solo si $\tau_i = \tau_j$. El predictor Venn asociado con esta taxonomía A_n da el par de probabilidad (p_0, p_1) como predicción para el objeto x_n donde:

$$P_y := \frac{\left|\{(x^*, y^*) \in \lbrace z_1, \ldots \ldots, z_{n-1}, (x_n, y) \rbrace : \tau^* = \tau_n \wedge y^* = y\right|}{\left|\{(x^*, y^*) \in \lbrace z_1, \ldots \ldots, z_{n-1}, (x_n, y) \rbrace : \tau^* = \tau_n\right|} \tag{6.17}$$

para los valores $y \in \{0,1\}$. El denominador es siempre positivo, donde $(x^*, y^*) = (x_n, y) \Rightarrow \tau^* = \tau_n$. Intuitivamente, p_0 es la proporción de ejemplos con etiqueta 0 para todos los ejemplos de la misma categoría de x_n. Y p_1 es la proporción de ejemplos con etiqueta 1. Donde p_0 y p_1 son las predicciones de probabilidad de que la etiqueta de x_n sea 1. Según esto, se puede referenciar un predictor Venn como un *predictor multi-probabilístico.* Estas predicciones son útiles cuando $p_0 \approx p_1$.

Los predictores Venn tienen una importante propiedad de validez y es que están bien calibrados automáticamente. Esta propiedad les permite producir resultados válidos cuando los ejemplos se establecen de forma independiente con la misma distribución de probabilidad sin conocimiento de cual es su distribución, lo cual parece lo más lógico si se aplica a problemas del mundo real. Esta probabilidad se demuestra en el teorema 6.6 de [Gammerman et al.] si se satisface la asunción de intercambiabilidad. De acuerdo con el teorema 6.6, la validez no depende del tamaño de la secuencia de datos *N*. Por lo tanto se puede aplicar la segunda asunción que se ha indicado y que dice que la secuencia de ejemplos es finita. Sin embargo probarla es complicado. Para ello Vovk propuso una versión más sencilla en su teorema 1 [Vovk 2012] donde intuitivamente es más transparente.

El inconveniente de este procedimiento es que puede ser muy especial para algún tipo de conjuntos de datos. Hay que señalar que no siempre más categorías producen mejores resultados. Por tanto, en lugar de hablar de resolución, se puede hablar de eficiencia. Para una mayor eficiencia, a un predictor se le pide que sea

más automático en dar probabilidades, donde estas deben ser lo más cercanas posible a 0 ó 1. Los predictores Venn son adecuados para problemas multi-clase. La idea general de los predictores Venn se puede describir de la siguiente manera: en primer lugar los objetos conocidos se dividen en varias categorías mediante el uso de una *taxonomía* (en este paso no se incluyen las etiquetas de los objetos). El nuevo objeto a analizar es asignado a una de dichas categorías. Posteriormente se calculan las frecuencias de las etiquetas en la categoría que contiene el objeto bajo estudio así como la distribución de probabilidad para el nuevo ejemplo. Este procedimiento es análogo al seguido para la predicción conformal cuando se utiliza una medida de no-conformidad. Al final calcula un conjunto de probabilidades para la etiqueta desconocida del nuevo objeto.

A continuación se verá una definición más explícita de los predictores Venn incluyendo el concepto de taxonomía. Se asume que se trata de un problema de clasificación estándar de aprendizaje estadístico: dado un conjunto de entrenamiento de ejemplos $\{ z_1, z_2, \ldots . z_{n-1} \}$, donde cada z_i consiste en un par compuesto por un objeto x_i y una etiqueta y_i donde el espacio de etiquetas Y es finito. Si se dispone de un objeto de test x_n la tarea a realizar consistirá en predecir la etiqueta y para el nuevo objeto x_n y dar una estimación de que la probabilidad es correcta.

Asumiendo que se dispone de una taxonomía para n objetos $x_1, x_2, \ldots . x_n$ y que los divide de una determinada forma A_n. Hay que tener en cuenta que este procedimiento es diferente al utilizado en la medida de no-conformidad de los predictores conformales. Las medidas de no-conformidad y las taxonomías juegan un papel similar en los predictores conformales y en los predictores Venn. Por tanto se usará el mismo símbolo.

Se puede considerar también una etiqueta arbitraria $y \in Y$ para el nuevo objeto x_n. A_n asigna una categoría τ_i al ejemplo z_i

$$\tau_i := A_n(\{ z_1, z_2, \ldots . z_{i-1}, z_{i+1}, \ldots \ldots, z_{n-1}, (x_n, \mathrm{y}) \}, \mathrm{z_i}) \qquad (6.18)$$

Donde n es el número de elementos en la bolsa, y $\tau_i \in \mathrm{T}$ es una de las categorías finitas del espacio de categorías T. Más aún, se puede asignar z_i y z_j a la misma categoría si y solo si:

$$A_n(\{ z_1, z_2, \ldots . z_{i-1}, z_{i+1}, \ldots \ldots, z_n \}, \mathrm{z_i}) = A_n(\{ z_1, z_2, \ldots . z_{j-1}, z_{j+1}, \ldots \ldots, z_n \}, \mathrm{z_j}) \qquad (6.19)$$

La categoría τ_n es asignada al par (x_n, y). Siendo P_y la distribución empírica de probabilidad de las etiquetas en la categoría τ_n. P_y contendrá K frecuencias de todas las posibles etiquetas.

$$P_y := \frac{|\{(x^*,y^*)\in\lbrace z_1,\ldots\ldots,z_{n-1},(x_n,y)\rbrace : \tau^*=\tau_n \wedge y^*=y'\}|}{|\{(x^*,y^*)\in\lbrace z_1,\ldots\ldots,z_{n-1},(x_n,y)\rbrace : \tau^*=\tau_n\}|} \quad y' \in Y \tag{6.20}$$

P_y es la distribución de probabilidad en Y.

Habiendo calculado cada etiqueta para x_n, se obtiene una matriz P de frecuencias de KxK. El predictor Venn P:=$\{P_y : y \in Y\}$ es un predictor multiprobabilistico que da una probabilidad de K distribuciones, donde K es el número de etiquetas en dicho espacio de etiquetas.

Para interpretar estas predicciones se debe asignar una columna de P que debe tener el valor mínimo. A esta columna se le suele denominar de mejor *calidad.*

Algoritmo 6.2: Predictor Venn

Datos: conjunto de entrenamiento $Z^{(n-1)}$, objeto a analizar x_n
Resultado: etiqueta predicha $\hat{y}_n$, interval de probabilidad $[l_n, u_n]$

```
K ← |Y|;                                   /* obtener tamaño del espacio de etiquetas */
for y = 0 to K-1 do                        /* ejecutar para cada etiqueta */
    for i= 1 to n do                       /* por cada ejemplo, asignarlo a una categoría */
        τ_i ← Taxonomía(⌊ (x_1,y_1),....,(x_{i-1},y_{i-1}),(x_{i+1},y_{i+1}),.....,(x_n,y) ⌋,(x_i,y_i))
    end
    c← 0;                                  /* contador inicial a 0 */
    for i = 1 to n do                      /* asignar una categoría a cada ejemplo */
        if τ_i = τ_n
        then                               /* encontrar ejemplos en la categoría que se sitúa x_n */
        C ← añadeAlconjunto (C, y); /* añadir su etiqueta alconjunto de categorías */
    end
    end
    P_i ←calcularFrecuencia (C)            /* Calcular la frecuencia de etiquetas */
end
j_máximo ← buscaMejorColumna (P)           /* buscar columna con mejor calidad */
[l_n, u_n] ← buscaIntervalo(j_máximo, P)   /* buscar máximo y mínimo en la mejor columna */
return ŷ_n, [l_n, u_n]
```

Algoritmo 6.2. Predictor Venn.

Función: calcularFrecuencia.

```
Function calcularFrecuencia (C)
        total ← tamaño_de(C)          /* obtener tamaño del espacio de etiquetas */
        count ← 0
        for i = 1 to total do         /* ejecutar para cada etiqueta */
            contador[y_i] ← contador[y_i]+ 1
        end
end
```

Algoritmo 6.3. Cálculo de frecuencia para Predictor Venn.

Función: buscaIntervalo.

```
Function buscaIntervalo(j_máximo , P)
        bajo ← 0                      /* inicializar contadores */
        alto ← 0
        for i = 1 to K                /* ejecutar para cada etiqueta */
            if bajo> P_{i,j máximo}
                then
                    bajo ← P_{i,j máximo}
            end
            if alto < P_{i,j máximo}
                then
                    alto ← P_{i,j máximo}
            end
        end
        return [bajo, alto]
end
```

Algoritmo 6.4. Búsqueda del intervalo para Predictor Venn.

Por tanto la mejor columna será la que mayor calidad tenga, $j_{máximo}$. Con lo que la predicción será $j_{máximo}$ y el intervalo de probabilidades sobre si la predicción es la correcta viene definido de la siguiente manera:

$$\left[\min_{i=0,\ldots K-1} P_{i,j_{máximo}} , \max_{i=0,\ldots K-1} P_{i,j_{máximo}}\right] \tag{6.21}$$

En el algoritmo 6.2 se muestra un pseudocódigo del predictor Venn y en los algoritmos 6.3 y 6.4 sus funciones auxiliares. Se utilizan tres curvas para mostrar gráficamente los resultados del predictor: la probabilidad acumulada inferior (*PAI*)

(6.22), la probabilidad acumulada superior (*PAS*)(6.23) y la exactitud acumulada (*EA*) o bien tasa de acierto (6.24).

$$PAI(t) = \frac{1}{t}\sum_{i=1}^{t} L_i \tag{6.22}$$

$$PAS(t) = \frac{1}{t}\sum_{i=1}^{t} U_i \tag{6.23}$$

$$EA(t) = \frac{1}{t}\sum_{i=1}^{t} Acc_i \tag{6.24}$$

Donde t es el número de muestras de test que han sido añadidas al conjunto de entrenamiento y Acc_i =1 si la predicción para la muestra x_i ha sido correcta y 0 en caso contrario.

6.3 Algoritmos subyacentes.

Este apartado intenta dar visibilidad a un componente fundamental de los predictores Venn. Los algoritmos subyacentes son el soporte de los predictores Venn. Se llaman algoritmos subyacentes a aquellos algoritmos tradicionales (por ejemplo *k*-NNR) a los que se les aplica un predictor conformal. Como ya se mencionó anteriormente en este capítulo, los predictores conformales y predictores Venn son un conjunto de algoritmos en lugar de predictores simples. En [Vovk 2005] se indica que, en general, un predictor conformal o bien un predictor Venn puede ser desarrollado en una capa superior de cualquier algoritmo de aprendizaje. A continuación se detallan los algoritmos k-Vecinos más cercanos y las Máquinas Vector Soporte.

6.3.1 k-Vecinos más cercanos.

Este es uno de los métodos de aprendizaje automático más ampliamente utilizado, tanto en problemas de clasificación como en problemas de regresión. Fue desarrollado para llevar a cabo el análisis discriminatorio cuando se desconocen ó son difíciles de conocer las estimaciones de confianza paramétrica de densidad de probabilidad.

En 1951 Fix and Hodges [Fix 1951] introdujeron un método no paramétrico para reconocimiento de patrones al que se denominó regla de k-Nearest Neighbours

(*k*-NNR). Más tarde en 1967 Cover and Hart elaboraron las reglas que formalizarían en [Cover 1967], la regla de k-Nearest Neighbours. En dicho documento se demostró que, para un conjunto infinito de ejemplos, la probabilidad del error de la regla está acotada superiormente por el doble de la probabilidad de Bayes.

La idea básica que hay detrás del algoritmo *k*-NNR es que está basado en la cercanía de los ejemplos de entrenamiento en el espacio de características. El ejemplo a analizar se clasifica con la etiqueta más frecuente que tienen sus vecinos más cercanos. Por vecino más cercano se entiende al ejemplo que tiene la distancia métrica más pequeña entre el ejemplo a analizar y el ejemplo de entrenamiento. Los tipos de distancia más comúnmente utilizados son la distancia geométrica y la distancia euclidea. De todos modos hay otras distancias como la de Hamming introducida por [Hamming 1925] o la distancia de Mahalánobis [Mahalánobis 1963] que también pueden ser utilizadas.

6.3.2 Máquinas Vector Soporte (SVM).

El capítulo anterior trató en detalle el planteamiento teórico de las máquinas de vector soporte. En este se tratarán alternativas para implementar este algoritmo.

Como ya se ha comentado, las SVM son un algoritmo muy popular en el campo del aprendizaje automático. Aunque este algoritmo fue diseñado para resolver problemas de clasificación binaria se puede generalizar para resolver problemas multiclase.

Actualmente hay dos tipos de enfoques para resolver problemas SVM multiclase. El primero de ellos es la descomposición en varios subproblemas de clasificación binarios, y el segundo es considerar directamente un único problema de clasificación. Un planteamiento de este último se puede encontrar en [Guermeur 2000].

Se ha mencionado en esta memoria diferentes métodos a utilizar en combinación con varios clasificadores binarios para resolver un problema multiclase. Algunas de estas estrategias son: uno contra todos, uno contra uno, código corrector de error de salida [Dietterich 1995] y gráfico acíclico dirigido [Platt 1999].

El método uno frente a todos es el más utilizado [Bottou, 1994]. Este clasificador construye K clasificadores binarios $f_0, f_1, f_2, \ldots\ldots, f_{K-1}$, donde K es el número de clases. Cada clasificador binario se entrena con los ejemplos de una clase determinada como etiquetas positivas y el resto de ejemplos pertenecientes a

las otras clases como etiquetas negativas. Es decir, el clasificador binario f_2 se entrenará con los ejemplos de la clase 2 como ejemplos positivos y los ejemplos negativos lo formarán el conjunto del resto de clases. Posteriormente se combinan todos los clasificadores de acuerdo a su salida más alta. La predicción la de etiqueta del objeto se describe mediante la ecuación (6.26).

$$\hat{y}_n \coloneqq \underset{i=0,\ldots..,K-1}{\operatorname{argmax}} f_i(x_n) \tag{6.25}$$

La solución uno contra uno fue introducida por [Knerr 1990] y utilizada por primera vez en [Friedman 1996, Kreel 19999]. Este método construye un conjunto de $\frac{K(K-1)}{2}$ clasificadores binarios con los ejemplos de cada par de clases. Cuando se crea cada clasificador, los ejemplos tomados en primer lugar se toman como positivos y los segundos como negativos (ver figura 5.3). El clasificador de la clase i y la clase j se denominarán como f_{ij}. Además el clasificador f_{ji} es básicamente igual que el f_{ij} salvo que tiene las etiquetas opuestas del tal forma que $f_{ij}(x) = -f_{ji}(x)$. Una vez entrenados todos los clasificadores, se asigna una etiqueta $\hat{y}_n$ al objeto x_n a través de un criterio definido [Friedman 1996]; si el clasificador f_{ij} predice que el objeto x_n pertenece a la clase i entonces se asigna a esta clase, de no ser así se asigna a la clase j. Este proceso continúa para todos los clasificadores binarios. Por tanto la predicción de la clase será la que tenga mayor puntuación.

$$\hat{y}_n \coloneqq \operatorname{argmax}_{i=0,\ldots..,K-1} \sum_{j=0, j\neq 1}^{K-1} \mathcal{V}(f_{ij}(x_n) \qquad \mathcal{V}(x) \coloneqq \begin{cases} 1 \; si \; x > 0 \\ 0 \; si \; x \leq 0 \end{cases} \tag{6.26}$$

Donde $\mathcal{V}(x)$ es la función que asigna “1” o “0” según la predicción de salida. Si dos etiquetas tuvieran la misma puntuación, se tomaría la primera que hubiese aparecido. Este tipo de estrategia se denomina *max-wins voting*.

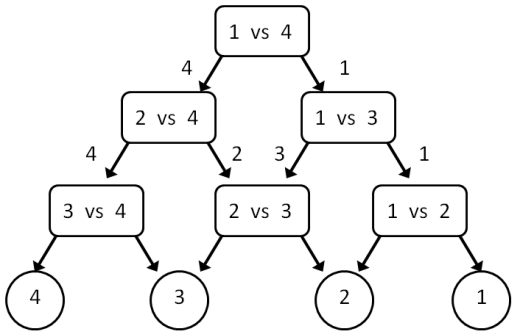


Figura 6.2. Grafo acíclico dirigido para 4 clases.

El *grafo* (*árbol*) *a-cíclico dirigido* (figura 6.2). También conocido como DAGSVM. Se debe a Platt [Platt 1998] y es un método que parte de la misma idea que el uno contra uno. Este método también genera $\frac{K(K-1)}{2}$ clasificadores. Sin embargo DAGSVM predice las etiquetas de los nuevos objetos a través de un grafo binario dirigido que utiliza los $\frac{K(K-1)}{2}$ clasificadores como nodos. Este tipo de gráfico comienza el análisis con el nodo raíz. En cada nodo el análisis de la etiqueta predicha puede determinar la dirección a tomar para el siguiente nodo. En cada nodo se elimina una posible clase, por tanto solo se pueden pasar por (*K-1*) clasificadores para predecir la etiqueta $\hat{y}_n$ del objeto x_n. El tiempo de computación se reduce frente al método uno contra uno. Además este método elimina la estrategia *max-wins voting* porque no pueden quedar dos clases ganadoras.

El método *código corrector de error de salida* (ECOC) (figura 6.3) utiliza también clasificadores binarios. Se trata de un método robusto lo que implica que se pueda implementar en cualquier algoritmo de clasificación. Fue desarrollado por Dietterich [Dietterich 1995]. Dado un problema de multi-clasificación de N clases, ECOC divide al problema en n subproblemas de menor complejidad, se tata de bi-particiones de las N clases. Como resultado de estas bi-particiones cada clase obtiene lo que se denomina como *palabra clave*, formada por tantos bits como clasificadores y donde cada bit pertenece a {+1, -1, 0}. Si se utiliza cada una de las *palabras clave* anteriormente comentadas como filas de una matriz, se forma $M \in \{+1, -1, 0\}^{N\ x\ n}$ siendo N el número de clases y n el número de clasificadores.

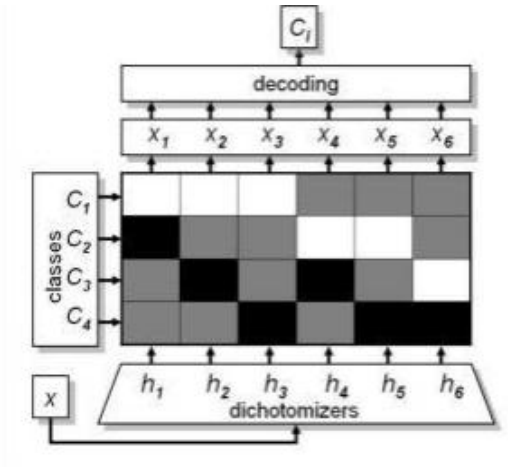


Figura 6.3. Ejemplo de matriz ECOC.

Para clasificar un nuevo objeto x_n cada clasificador entrenado previamente es evaluado para producir una nueva palabra clave. Se asigna entonces a la posible palabra clave más cercana. En este algoritmo se suele utilizar como medida la distancia Hamming. Se asume por tanto que, una palabra clave de k-bits de longitud para cada clase es $W_i = (w_{i,0}, \ldots\ldots, w_{i,k-1})$ y que la palabra predicha es $B_i = (b_{i,0}, \ldots\ldots, b_{i,k-1})$ entonces la etiqueta predicha viene dada por:

$$\hat{y}_n \coloneqq \operatorname{argmin}_{i=0,\ldots\ldots,K-1} \sum_{j=0, j\neq 1}^{K-1} \left| b_j - w_{i,j} \right| \qquad (6.27)$$

La ventaja de este método es que mediante la codificación de cada etiqueta y el entrenamiento de cada bit por separado, el clasificador está preparado para recuperarse de errores. Si la distancia de Hamming mínima entre dos palabras clave es d entonces este algoritmo puede corregir al menos errores en $\frac{d-1}{2}$ bits en la palabra clave predicha B. Para cualquier otra palabra clave W_i sería d-$\left[\frac{d-1}{2} = \frac{d-1}{2} + 1\right] > \frac{d-1}{2}$, lo que significa que la palabra clave más cercana de B sigue siendo la más correcta.

6.4 Diseño de la taxonomía SVM-Venn predictor. Ejemplo de operación.

Este apartado explica en detalle la implementación de la *taxonomía* utilizada en esta Tesis. Los dos métodos de aprendizaje codificados han sido uno contra todos y uno contra uno. Como lenguaje de programación se ha utilizado la toolbox de Matlab *distributing parallel*.

El algoritmo subyacente utilizado en el desarrollo del predictor Venn es el smo-SVM multiclase. Los entornos de predicción han sido tanto on-line como off-line (o batch). Tan solo recordar que el entorno on-line consiste en entregar al modelo los ejemplos de forma aleatoria. Una vez entregado al modelo el ejemplo x_1 éste predice una etiqueta $\hat{y}_1$. A continuación se le entrega al modelo la etiqueta correcta y_1, el ejemplo se incorpora a su clase correcta y se continua el análisis con el siguiente ejemplo. El modelo parte de un pequeño conjunto inicial de ejemplos de cada clase, en este trabajo se consideraron diez muestras o ejemplos de cada clase como conjunto inicial de entrenamiento. Como alternativa al método on-line está el método off-line. Al contrario del método anterior, aquí no existe realimentación del modelo. Es decir, se inicia el proceso de clasificación con un conjunto inicial compuesto por un mayor número de muestras, pero las muestras analizadas no se añaden al conjunto de training.

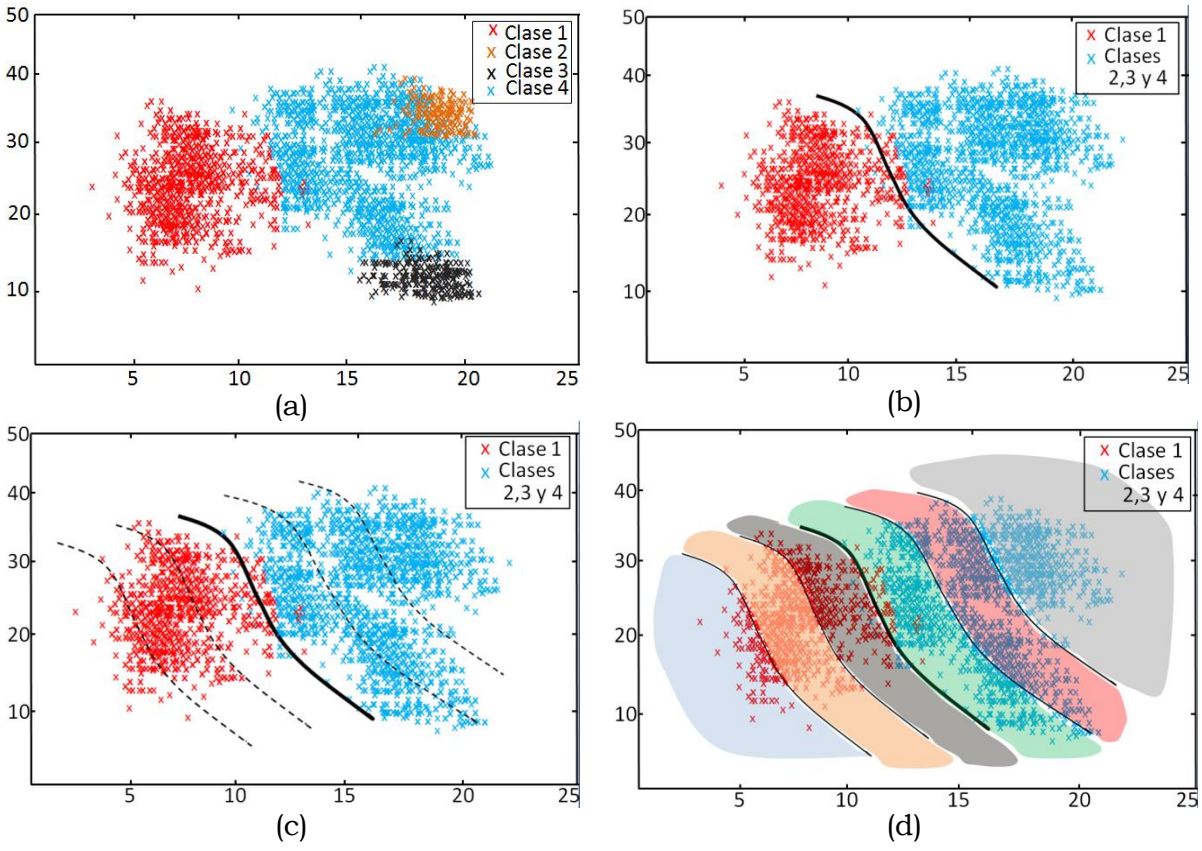


Figura 6.4. Diseño de la taxonomía SVM-Venn Predictor para cuatro clases.

A continuación se explica la forma de definir y construir el predictor Venn. Una vez seleccionado el algoritmo subyacente, que para este predictor es smo-SVM, se determina el número de bandas en las que se va a dividir el conjunto de ejemplos. La figura 6.4(a) muestra cuatro clases diferentes de ejemplos. En el apartado anterior se comentaron diferentes enfoques para abordar problemas binarios multiclase, aquí se utilizará el enfoque uno contra todos. Esto quiere decir

Banda 1	$(-\infty, -d_1]$
Banda 2	$(-d_1, -d_2]$
Banda 3	$(-d_2, 0]$
Banda 4	$(0, d_3]$
Banda 5	$(d_3, d_4]$
Banda 6	(d_4, ∞)

Tabla 6.1. Definición de los intervalos de las bandas para la taxonomía definida**.**

que se construirán 4 clasificadores y cada uno de ellos hará referencia a una clase frente al resto. En la figura 6.4(b) se muestra el primer clasificador. Los ejemplos de la clase 1 (rojo) se comparan con el resto de ejemplos de las clases 2, 3 y 4 (azul). Mediante el algoritmo smo-SVM, el clasificador calcula el hiperplano separador de la clase 1 frente a las clases agrupadas 2, 3 y 4. Como el clasificador calcula las distancias de todos los ejemplos al hiperplano de separación, puede entonces definir los intervalos para cada banda. La taxonomía definida es de seis bandas y la tabla 6.1 muestra los intervalos considerados. En la figura 6.4(c) se ven las fronteras de separación de las bandas y la figura 6.4(d) muestra sombreados los ejemplos asignados a cada banda definida que tendrán que ser considerados para los cómputos a realizar más tarde.

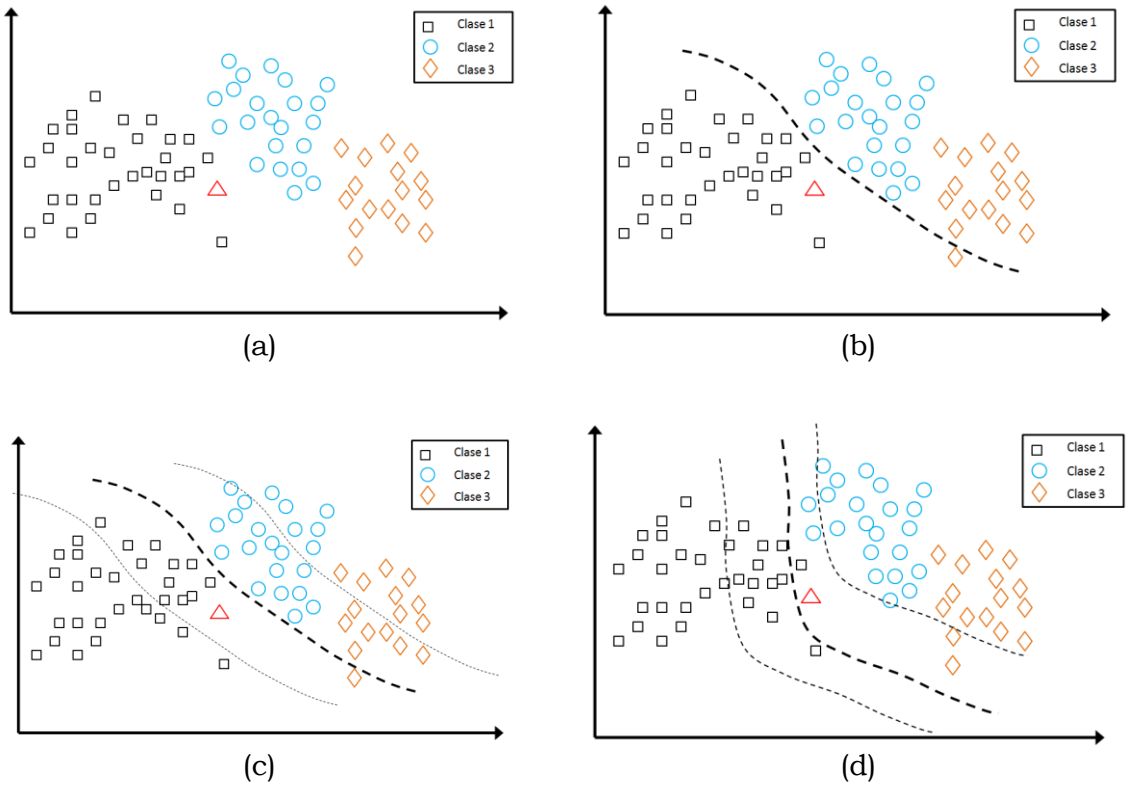
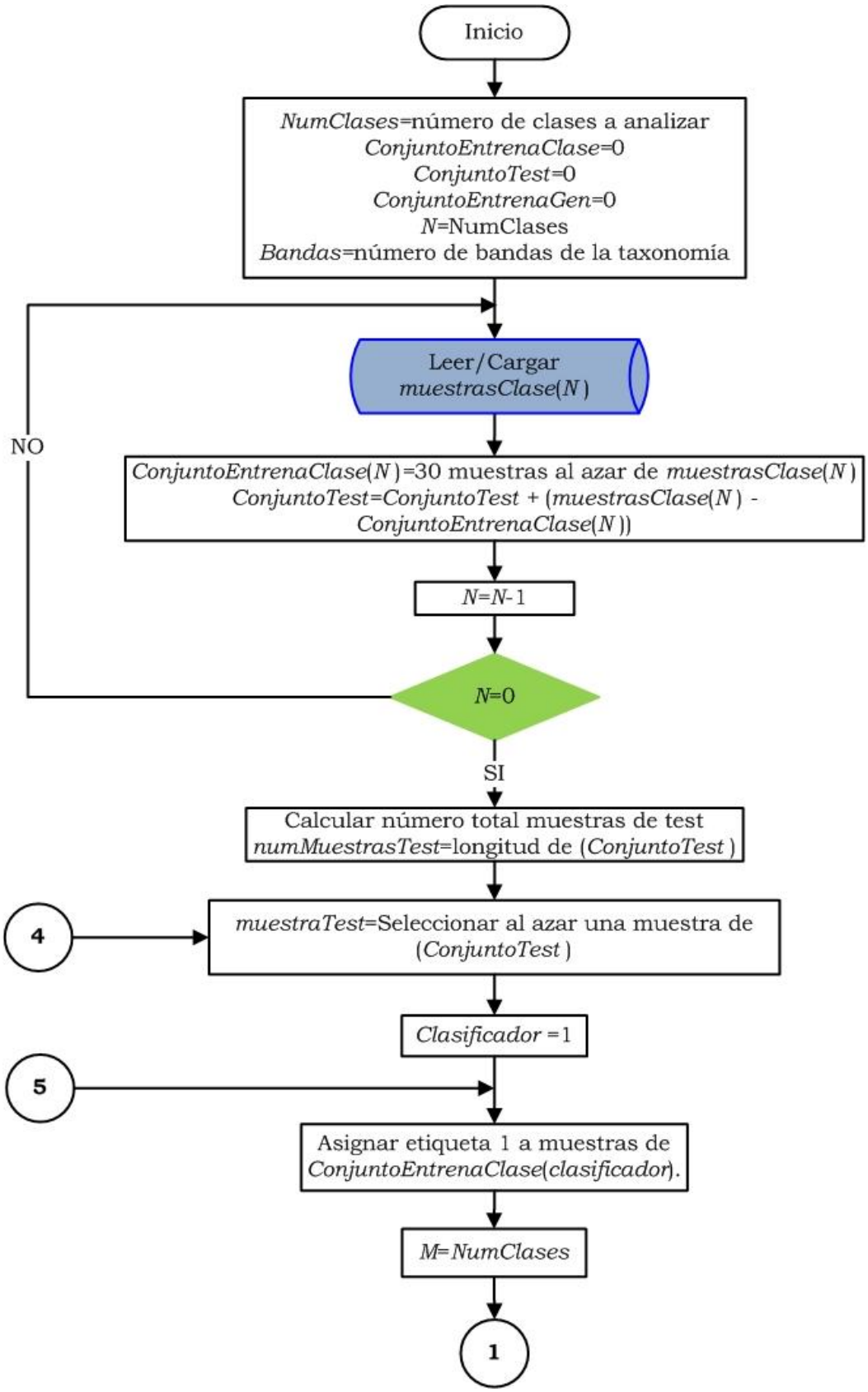
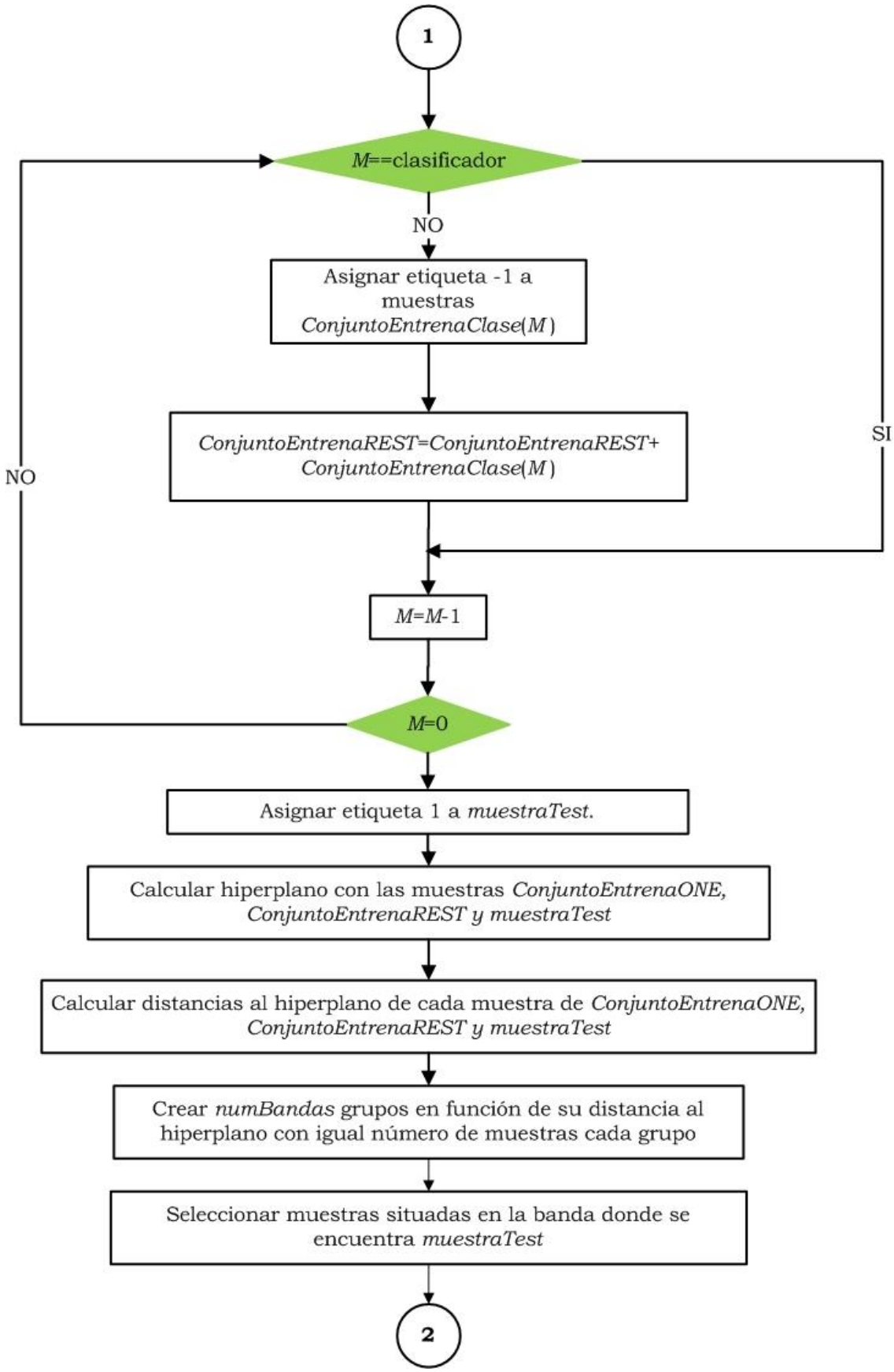


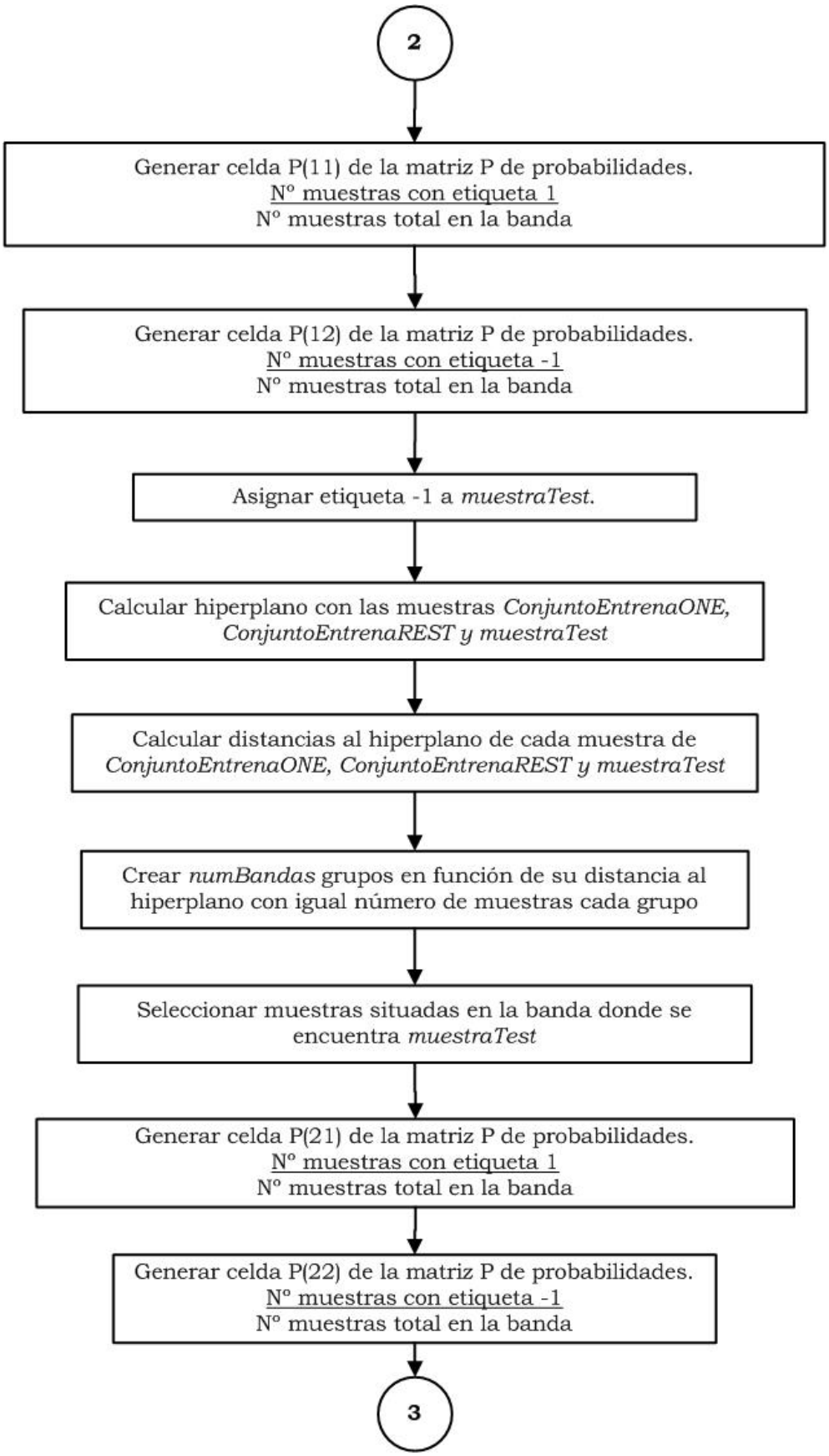
Figura 6.5. Cálculo de probabilidades de una muestra con taxonomía smo-SVM-Venn Predictor. En (a) se puede ver que hay definidas tres clases (clase 1 cuadrados grises, clase 2 círculos azules y clase 3 rombos marrones. El triángulo rojo indica la muestra a analizar. En (b) se han fusionado las clases 2 y 3 en una sola y se ha calculado el hiperplano que separa la clase 1 de las clases 2 y 3. En (c) se muestran las cuatro bandas equidistantes al hiperplano. Y en (d) las nuevas cuatro bandas cuando la muestra se supone que pertenece al grupo contrario.

Ejemplo de operación para Taxonomía smo-SVM.

La figura 6.4 mostraba la forma de definir las bandas siendo éste un paso necesario de la taxonomía. A continuación se detalla el modo de operación del predictor Venn para una clase. El procedimiento que se describe se debe repetir para cada una de las clases restantes bajo análisis. Es fácil deducir la carga computacional que imponen los predictores Venn.


Inicio
NumClases=número de clases a analizar
ConjuntoEntrenaClase=0
ConjuntoTest=0
ConjuntoEntrenaGen=0
N=NumClases
Bandas=número de bandas de la taxonomía
Leer/Cargar
muestrasClase(N)
NO
ConjuntoEntrenaClase(N)=30 muestras al azar de muestrasClase(N)
ConjuntoTest=ConjuntoTest + (muestrasClase(N) -
ConjuntoEntrenaClase(N))
N=N-1
N=0
SI
Calcular número total muestras de test
numMuestrasTest=longitud de (ConjuntoTest)
4
muestraTest=Seleccionar al azar una muestra de
(ConjuntoTest)
Clasificador =1
5
Asignar etiqueta 1 a muestras de
ConjuntoEntrenaClase(clasificador).
M=NumClases
1


1
M==clasificador
NO
Asignar etiqueta -1 a muestras ConjuntoEntrenaClase(M)
ConjuntoEntrenaREST=ConjuntoEntrenaREST+ ConjuntoEntrenaClase(M)
SI
NO
M=M-1
M=0
Asignar etiqueta 1 a muestraTest.
Calcular hiperplano con las muestras ConjuntoEntrenaONE, ConjuntoEntrenaREST y muestraTest
Calcular distancias al hiperplano de cada muestra de ConjuntoEntrenaONE, ConjuntoEntrenaREST y muestraTest
Crear numBandas grupos en función de su distancia al hiperplano con igual número de muestras cada grupo
Seleccionar muestras situadas en la banda donde se encuentra muestraTest
2


2
Generar celda P(11) de la matriz P de probabilidades.
Nº muestras con etiqueta 1
Nº muestras total en la banda
Generar celda P(12) de la matriz P de probabilidades.
Nº muestras con etiqueta -1
Nº muestras total en la banda
Asignar etiqueta -1 a muestraTest.
Calcular hiperplano con las muestras ConjuntoEntrenaONE, ConjuntoEntrenaREST y muestraTest
Calcular distancias al hiperplano de cada muestra de ConjuntoEntrenaONE, ConjuntoEntrenaREST y muestraTest
Crear numBandas grupos en función de su distancia al hiperplano con igual número de muestras cada grupo
Seleccionar muestras situadas en la banda donde se encuentra muestraTest
Generar celda P(21) de la matriz P de probabilidades.
Nº muestras con etiqueta 1
Nº muestras total en la banda
Generar celda P(22) de la matriz P de probabilidades.
Nº muestras con etiqueta -1
Nº muestras total en la banda
3

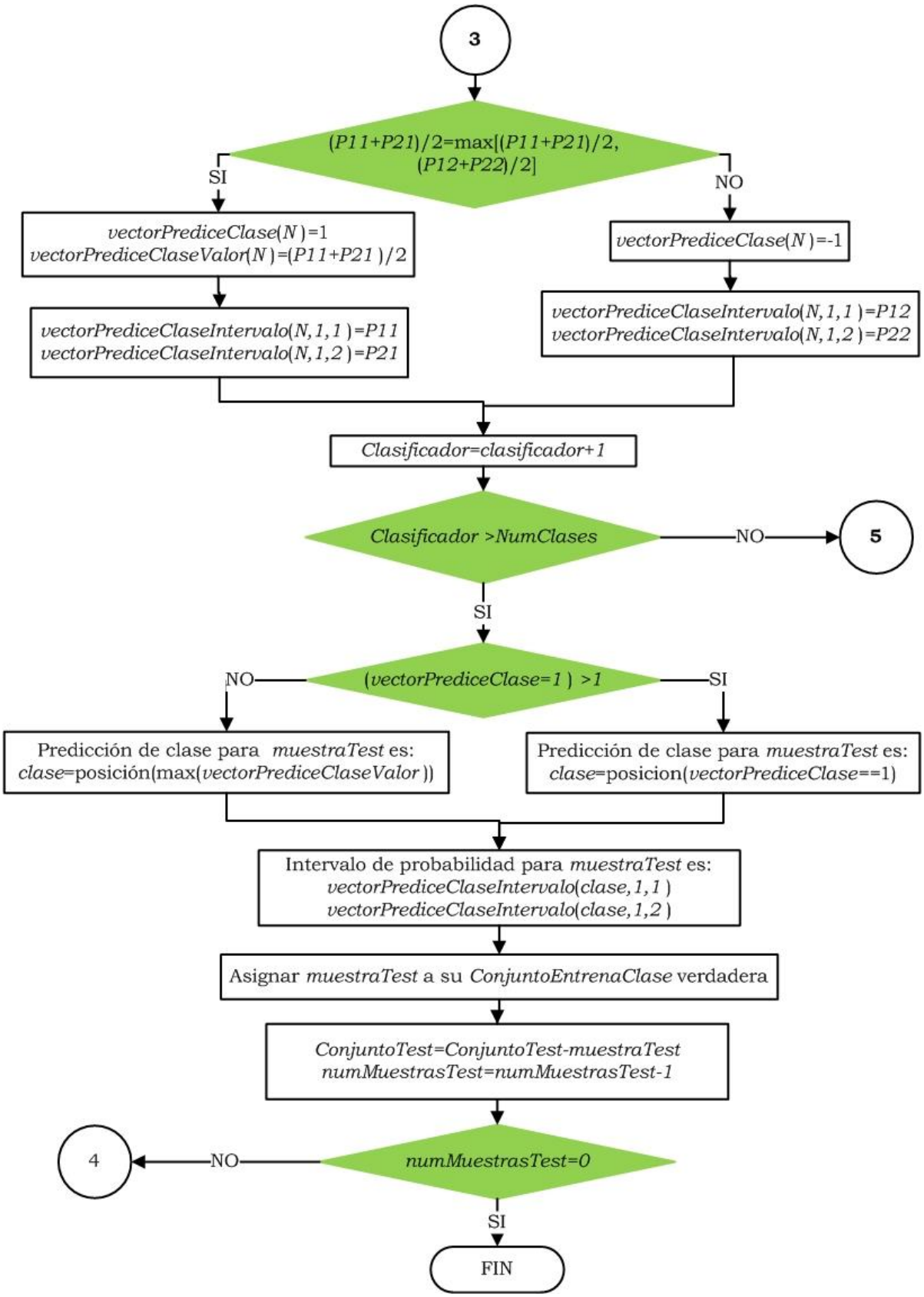


Figura 6.6. Diagrama de flujo del predictor Venn con smo-SVM.

El proceso comienza con tres clases como se puede ver en la figura 6.5(a). Las clases son cuadrados, rombos y círculos. Se presenta un nuevo ejemplo, el triángulo rojo. En principio no se sabe a qué clase pertenece este nuevo ejemplo. Se comienza con el clasificador 1 y esto quiere decir que se procesa esta clase 1 frente al resto (clases 2 y 3). Se asume que el ejemplo pertenece a la clase 1, por tanto se le asigna la etiqueta {1} a la clase 1 y al nuevo ejemplo. Al resto de clases se asigna la etiqueta {-1}. Mediante el algoritmo smo-SVM se calcula el hiperplano de separación con una función kernel de base radial (figura 6.5(b)). Se ha definido una taxonomía con cuatro bandas y por tanto hay que calcular las distancias de todos los puntos al hiperplano. Con las distancias de cada muestra calculada previamente se pueden definir los límites de cada banda, véase la figura 6.5(c).

Seguidamente hay que calcular la primera fila de la matriz P de probabilidades. Para ello se seleccionan los ejemplos situados en la banda donde se ha situado el ejemplo bajo análisis. Para calcular el elemento p_{11} se contabilizan todos los ejemplos con la misma etiqueta de la clase bajo análisis {1}. Para este caso serían 11 cuadrados+1 triángulo en total 12 ejemplos y todo ello dividido entre el total de ejemplos de la banda lo que hace un total de 13. El elemento $p_{11} = \frac{12}{13} = 0.923$. Para calcular el elemento p_{12} se toma como numerador los ejemplos con etiqueta contraria {-1} que pertenece a un rombo y en el denominador el total de ejemplos en la banda. Por tanto el elemento $p_{12} = \frac{1}{13} = 0.0763$. Ahora queda calcular la segunda fila de la matriz *P.* Para ello se cambia la etiqueta del ejemplo bajo estudio y se asume que pertenece al grupo formado por las clases 2 y 3 con etiqueta {-1}. Se vuelve a calcular el hiperplano (este hiperplano es distinto al calculado anteriormente). Se calculan de nuevo las distancias de todos los ejemplos al nuevo hiperplano. También se delimitan las nuevas bandas, véase la figura 6.5(d). Se repite de nuevo el cálculo anterior pero en este nuevo caso los ejemplos situados en la misma banda del ejemplo bajo estudio son diferentes en número. Se calcula el elemento $p_{21} = \frac{1}{6} = 0.200$ y $p_{22} = \frac{5}{6} = 0.833$. La matriz de probabilidades se expresa de la forma:

$$P=\begin{bmatrix} \frac{12}{13} & \frac{1}{13} \\ \frac{1}{6} & \frac{5}{6} \end{bmatrix} = \begin{matrix} 0.923 & 0.076 \\ 0.200 & 0.833 \end{matrix}$$

Ahora ya se está en condiciones de calcular $\overline{p(K)}$ que es la media de las probabilidades obtenidas para la etiqueta Y_k entre todas las distribuciones de probabilidad (la media de cada columna de la matriz P). Esto es:

$$\overline{p(1)} = \frac{0.923 + 0.2}{2} = 0.561$$

$$\overline{p(23)} = \frac{0.076 + 0.833}{2} = 0.454$$

En este caso el predictor Venn da como resultado de la predicción $\hat{y}_n = \{1\}$, es decir que el ejemplo pertenece a la clase 1. El intervalo de probabilidad para esta predicción será [0.200, 0.923]. Cuanto más pequeño sea el intervalo mejor será la predicción.

Una vez analizada la muestra asumiendo que pertenece a la clase 1, el siguiente paso sería asumir que la muestra pertenece a la clase 2 y repetir los mismos pasos realizados para la clase 1, es decir deben repetirse de igual forma para el clasificador 2 que analiza clase 2 frente a la clase 1 y 3. El mismo procedimiento debe realizar el clasificador 3 para analizar la clase 3 frente a la clase 1 y 2. Una vez calculadas las predicciones restantes, se tendrían los siguientes resultados: $\overline{p(1)} = 0.561$, $\overline{p(2)} = 0.394$ y $\overline{p(3)} = 0.187$.

La clase ganadora en este caso sería la 1 (la que tiene la probabilidad media más alta). El último paso para este ejemplo seria asignarle su clase verdadera y continuar el proceso con el siguiente ejemplo a analizar. En la figura 6.6 se detalla el diagrama de flujo completo para un número determinado de clases.

6.5 Paralelización.

La naturaleza transductiva del predictor Venn (TVP) visto hasta ahora le hace ser computacionalmente ineficiente puesto que requiere entrenar el algoritmo subyacente para cada posible clase de cada nueva muestra a analizar. En [Lambrou 2014] se propone un nuevo enfoque, el predictor Venn inductivo (IVP). Este predictor permite ganar velocidad al mismo tiempo que se obtienen buenos resultados en predicción y buena calibración.

Este nuevo enfoque divide en dos partes las muestras. Por un lado están las muestras de entrenamiento del algoritmo subyacente y por otro lado están las muestras de calibración para calcular las distribuciones de probabilidad de cada

nueva muestra a analizar. En cada paso se crea una predicción Venn análoga a la que se realiza en la predicción conformal inducida [Vovk 2005]. El planteamiento es que las muestras una vez analizadas pasen a formar parte del conjunto de calibración. A su vez cada *m* muestras analizadas se le incorporan al conjunto de entrenamiento muestras del conjunto de calibración. De esta forma los dos conjuntos crecen de la misma forma. La forma en la que el IVP gana velocidad es en base a realizar menos cálculos pues el hiperplano de separación del algoritmo subyacente se recalcula cada *m* muestras. En [Lambrou 2014] se puede encontrar más detalle sobre el IVP.

El método de validación on-line visto en la sección 6.2.2.1 ha sido el aplicado en las pruebas. El primer paso para entrenar el modelo comienza con diez muestras de cada clase tomadas al azar. Al ser un número reducido de muestras, el impacto computacional es mucho menor. Si el número de muestras fuese muy elevado se tendría que recurrir a otros métodos de paralelización. El IVP al realizar menos cálculos gana en velocidad de clasificación pero a cambio los resultados a pesar de ser buenos son inferiores al TVP.

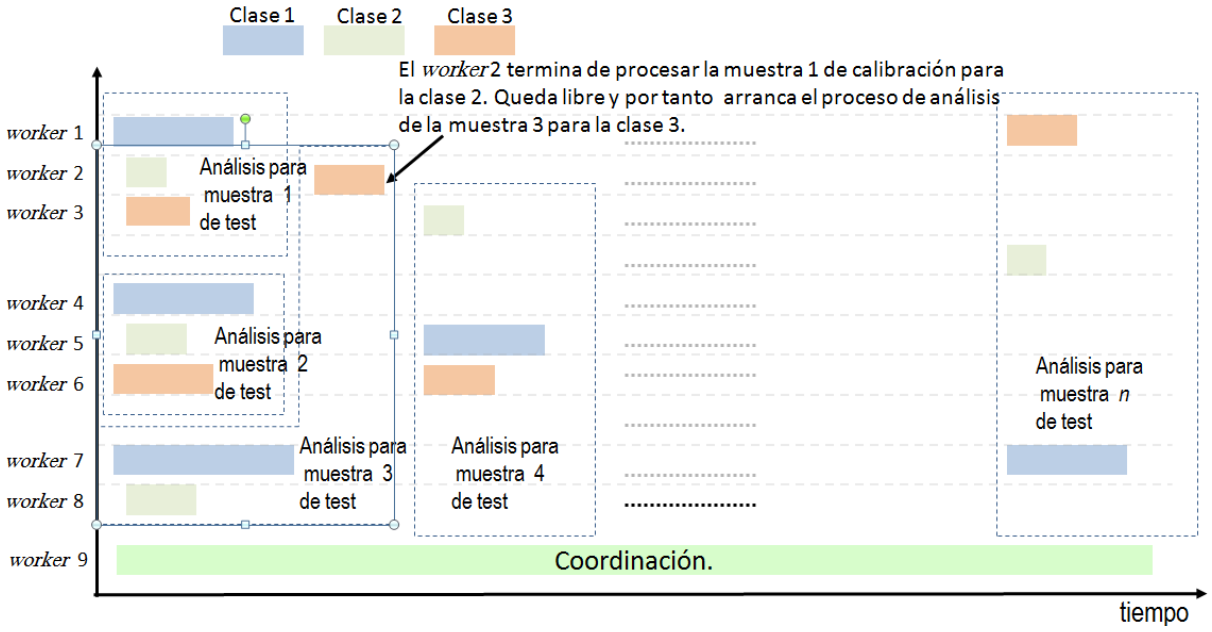


Figura 6.7. Implementación paralela del Predictor Venn Inducido.

El algoritmo diseñado para la paralelización se muestra en la figura 6.7. El gráfico muestra la paralelización de la fase de análisis de muestras. En este ejemplo

el número de las clases a analizar es 3 y el número de *workers* disponibles es de 9. El proceso comienza con el análisis de la muestra 1. Se realiza el cálculo de probabilidad asumiendo que la muestra 1 pertenece a cada una de las tres clases. Esto lo realizan los tres primeros *workers*. De forma simultánea a este cálculo, los *workers* 4, 5 y 6 realizan el mismo proceso para la muestra 2 pero con la muestra 1 ya asignada a su clase correspondiente. También de forma simultánea los *workers* 7 y 8 realizan el mismo cálculo para la muestra 3 pero con las muestras 1 y 2 asignadas a su clase correspondiente. En este último caso ya no quedan *workers* libres a los que asignar el análisis de la muestra 3 para la clase 3. Pero justo cuando termina el *worker* 2 se le asigna el análisis de la muestra 3 para la clase 3. Al *worker* 9 le llegan los análisis de cada *worker* y es el encargado de calcular la predicción final de cada muestra en función de las predicciones enviadas por todos los *workers*. El *worker* 9 también tiene como objetivo asignar cada muestra a su clase correspondiente y posteriormente distribuir esta clase al resto de *workers*. A medida que el proceso de cálculo avanza, ciertas clases pueden tener más muestras que otras. Basta con fijarse por ejemplo que, cuando el *worker* 9 ha enviado la muestra *n* de la clase 3 a los tres primeros *workers*, entonces a los *workers* 4, 5 y 6 esa muestra ya les cuenta como muestra de entrenamiento y así sucesivamente. Según este planteamiento, a los *workers* 7 y 8 les llegarían dos muestras más de entrenamiento. A estos tiempos de retardo hay que sumarles el tiempo de red y de gestión vistos en la sección 5.4.4. Estos motivos hacen que los tiempos de terminación sean diferentes y que el análisis de las muestras puedan realizarlo de forma colaborativa todos los *workers.*

6.6 Experimentos y resultados.

A continuación se detallarán los datos utilizados y su preprocesamiento (en algunos casos) que sirvieron de entrada al algoritmo implementado así como los resultados obtenidos del predictor Venn. En adelante salvo que se indique lo contrario los resultados pertenecen al predictor Venn transductivo (TVP).

6.6.1 Datos utilizados para pruebas.

Con el fin de poder analizar cómodamente los resultados y hacer el trabajo más fluido, se diseñó una interfaz gráfica, la consola VPtrain (figura 6.8). Está desarrollada mediante el lenguaje Matlab y dispone de los elementos necesarios

para definir el modelo. Se puede parametrizar y tiene la posibilidad de mostrar gráficamente los resultados del modelo. Antes de afrontar el análisis de imágenes del sistema de esparcimiento Thomson TJ-II (uno de los objetivos de la Tesis) y con el fin de verificar los resultados con estudios anteriores, se analizaron tres bases de datos diferentes. El motivo es que estas bases de datos no comparten la misma estructura y esta característica las hace aptas para la prueba. Se trata de las bases de datos dermatología, evaluación de vehículos y formas de onda del TJ-II.

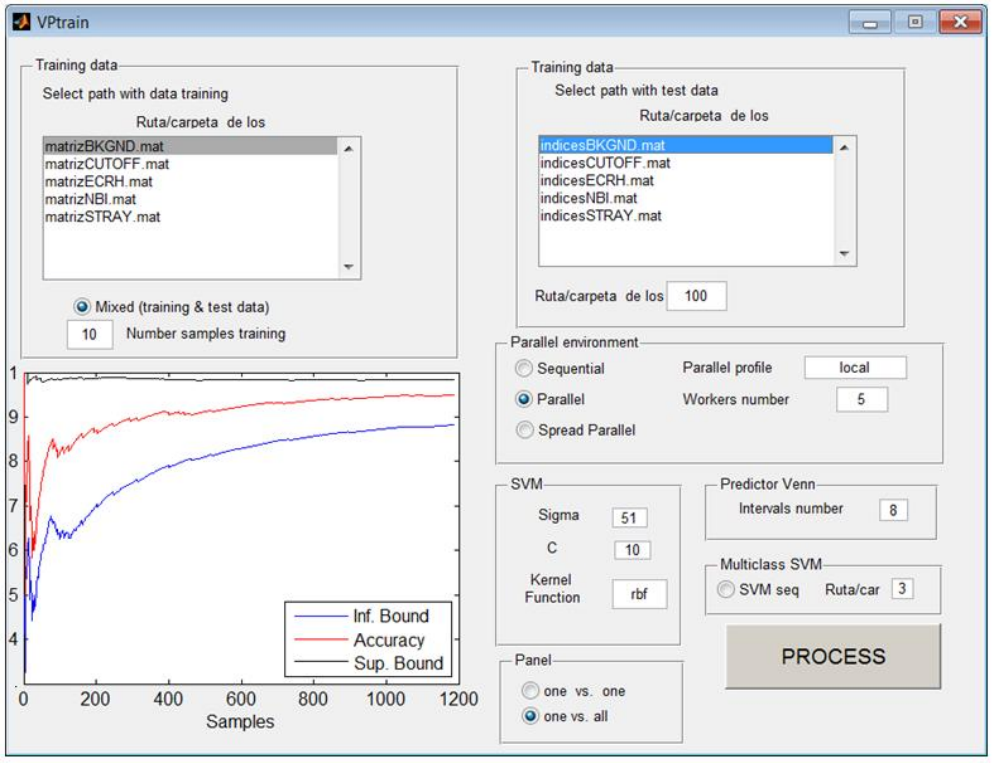


Figura 6.8. Consola de la aplicación VPtrain.

- **BD Dermatología.**

Esta base de datos se debe a un trabajo realizado por Gvnir [Gvnir 1998]. Es una base de datos original de la universidad de California Irvine. Estos datos se encuentran en el repositorio UCI [Lichman 2013]. La naturaleza de la información gira en torno al diagnóstico diferencial de las enfermedades en dermatología. Todos ellos comparten las características clínicas de eritema y descamación, con muy pocas diferencias.

Las enfermedades de este grupo son la psoriasis, la dermatitis seboreic, liquen plano, pitiriasis rosada, dermatitis crónica, y la pitiriasis rubra pilaris.

Una dificultad para el diagnóstico diferencial es que una enfermedad puede mostrar las características de otra enfermedad en la etapa inicial y puede tener los rasgos característicos en las siguientes etapas.

Cod. Clase	Clase.	Instancias.
1	psoriasis	112
2	seboreic dermatitis	61
3	lichen planus	72
4	pityriasis rosea	49
5	cronic dermatitis	52
6	pityriasis rubra pilaris	28

Tabla 6.2. Clases que componen la Base de Datos Dermatología.

1	erythema	18	hyperkeratosis
2	scaling	19	parakeratosis
3	definite borders	20	clubbing of the rete ridges
4	itching	21	elongation of the rete ridges
5	koebner phenomenon	22	thinning of the suprapapillary epidermis
6	polygonal papules	23	spongiform pustule
7	follicular papules	24	munro microabcess
8	oral mucosal involvement	25	focal hypergranulosis
9	knee and elbow involvement	26	disappearance of the granular layer
10	scalp involvement	27	vacuolisation and damage of basal layer
11	family history, (0 or 1)	28	spongiosis
12	melanin incontinence	29	saw-tooth appearance of retes
13	eosinophils in the infiltrate	30	follicular horn plug
14	PNL infiltrate	31	perifollicular parakeratosis
15	fibrosis of the papillary dermis	32	inflammatory monoluclear inflitrate
16	exocytosis	33	band-like infiltrate
17	acanthosis	34	Age (linear)

Tabla 6.3. Atributos de la clase de la base de datos dermatology.

Los valores de cada atributo de las clases que componen esta BD tienen un rango de 0 a 3. Un valor 0 indica la no presencia de síntomas, el valor 3 indica el valor más alto y los valores 1 y 2 indican valores intermedios. La base de datos Dermatology contiene 6 clases. Las instancias por clase van desde 28 para la clase 6 hasta 112 para la clase 1, en total hay 374 instancias. El detalle de estas clases asi como el número de instancias de cada una de ellas figuran en la tabla 6.2.

Además cada instancia contiene 34 atributos y la descripción de cada atributo está reflejado en la tabla 6.3. Uno de los motivos para utilizar esta base de datos es cotejar los resultados descritos en el trabajo [Lambrou 2012b].

Atributo	Valores
compra	bajo, medio, alto o muy alto
mantenimiento	bajo, medio, alto o muy alto
puertas	2, 3, 4, 5 o más
personas	2, 4 o más
maletero capacidad	pequeño, medio y grande
seguridad	bajo, medio o alta

Tabla 6.4. Estructura de los datos de entrenamiento de la base de datos UCI de evaluación de vehículos.

Atributo	Valores nominal	Nuevo valor numérico
compra	muy alto	4
	alto	3
	medio	2
	bajo	1
mantenimiento	muy alto	4
	alto	3
	medio	2
	bajo	1
maletero capacidad	grande	3
	mediano	2
	pequeño	1
seguridad	alta	3
	media	2
	baja	1

Tabla 6.5. Conversión de valores nominales a numéricos de la base de datos de evaluación de vehículo.

- **BD evaluación vehículos.**

Esta es una base de datos de la universidad de california, Irvine. Está disponible en UCI [Frank 2010]. Esta base de datos deriva de un modelo jerárquico de decisión y desarrollado originalmente para la demostración de DEX. Son

registros específicos sobre vehículos y donados por Marco Bohanec en 1998 [Bohanec 1998]. En total hay 1728 instancias agrupadas en cuatro clases. Cada clase contiene seis atributos descritos en la tabla 6.4. Algunos de estos atributos tienen valores nominales que el modelo no puede tratar directamente. Por este motivo fue necesario realizar una conversión de datos en cuatro de los seis atributos. La tabla 6.5 muestra los atributos a los que se le aplicó la conversión asi como el valor numérico asignado a cada valor nominal.

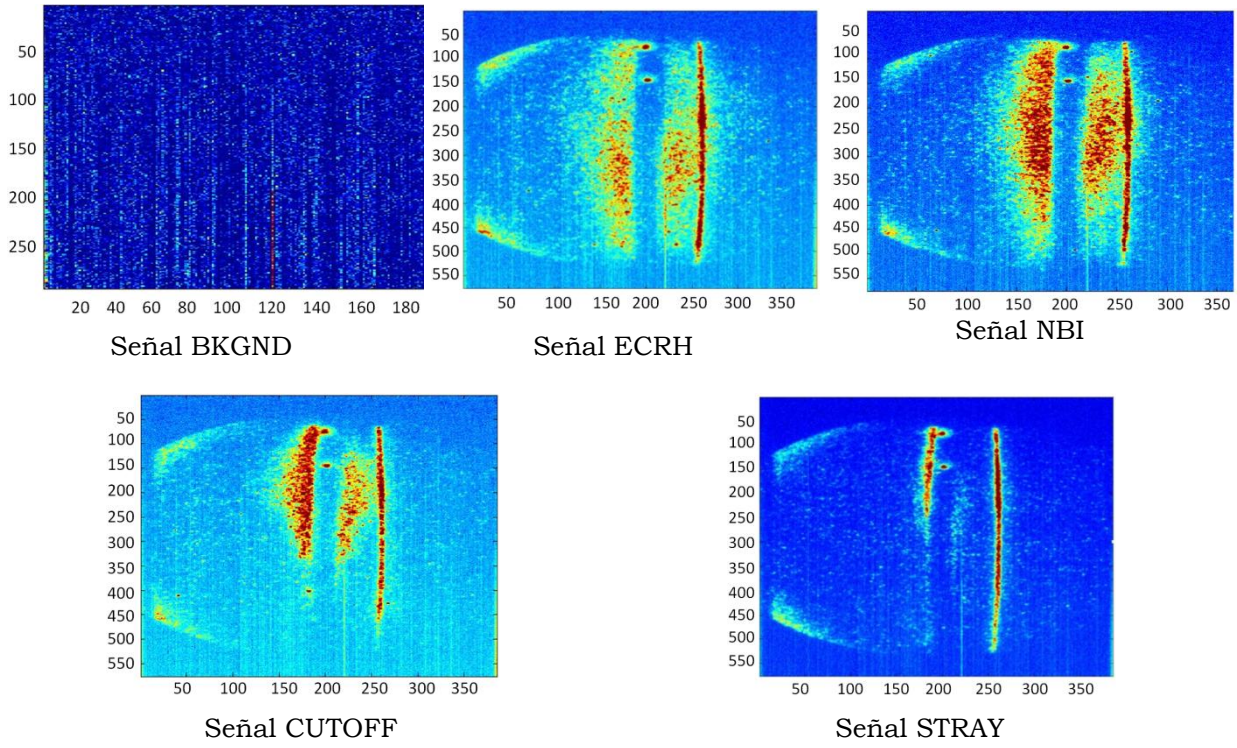


Figura 6.9. Muestras de imágenes del diagnóstico de esparcimiento Thomson del TJ-II.

• Imágenes del diagnóstico de esparcimiento Thomson del TJ-II.

El diagnóstico Scatering Thomson (TSD) mide la temperatura y la densidad electrónica del plasma. Analiza el espectro de la luz dispersa del plasma. La luz emitida por un pulso laser se dispersa por los electrones libres del plasma. Las imágenes del plasma capturadas obedecen a diferentes métodos de calentamiento o a calibraciones del sistema, véase la figura 6.9. El término *esparcimiento Thomson* se

debe al efecto que se produce cuando la luz incidente posee una frecuencia suficientemente baja. La luz es detectada y capturada [Makili 2014].

Clase	Descripción	Nº muestras
BKGND	Fondo de la Cámara ICCD.	107
ECRH	Plasma durante el calentamiento ECRH.	461
CUTOFF	Plasma después de alcanzada la densidad de corte durante el calentamiento ECRH.	42
NBI	Plasma durante el calentamiento NBI.	334
STRAY	Luz parásita sin plasma.	205

Tabla 6.6. Imágenes del diagnóstico de esparcimiento Thomson del TJ-II.

La base de datos está compuesta por cinco clases de imágenes que pertenecen a diferentes estados del plasma. La tabla 6.6 explica de forma breve cada una de ellas. Cada imagen se construye a partir de un fichero que contiene los valores de cada pixel. En total son 221760 pixeles que se corresponden con una matriz 576 x385. Para el proceso de clasificación, cada pixel tiene la función de atributo de la clase.

Es por tanto necesario reducir los atributos de cada clase para que el problema sea más manejable. Esta necesidad ya se ve reflejada en trabajos previos publicados sobre la clasificación de estas imágenes [Vega 2010, Dormido-Canto 2012, Makili 2014]. El proceso de reducción de dimensionalidad debe tratarse mediante la transformada Wavelet. La familia de wavelet utilizada fue la de Haar [Haar 1910]. En el anexo A se detallan los principos de la transformada wavelet y cómo se ha aplicado a la reducción de dimensionalidad de las imágenes.

Haciendo un análisis previo de las imágenes, se comprobó que la mayor parte de información, o mejor dicho, la mayor energía de cada imagen residía en el centro de la misma. Parecía lícito por tanto centrarse en esta área de cada imagen. La figura 6.10 muestra cómo se reduce el área de estudio para una imagen de la clase CUTOFF. En este paso se reduce considerablemente la dimensión de la imagen. A esta imagen resultante se le aplica un nivel de descomposición 4 de TW. El resultado final es una matriz de 31x9. El proceso de reducción de dimensionalidad explicado anteriormente se tuvo que aplicar a todos los ficheros que componen la base de datos de imágenes del diagnóstico de esparcimiento Thomson TJ-II.

El objetivo de este estudio se centra en generar un modelo de predictor Venn asi como su paralelización. Se entiende que los datos a clasificar llegan reducidos

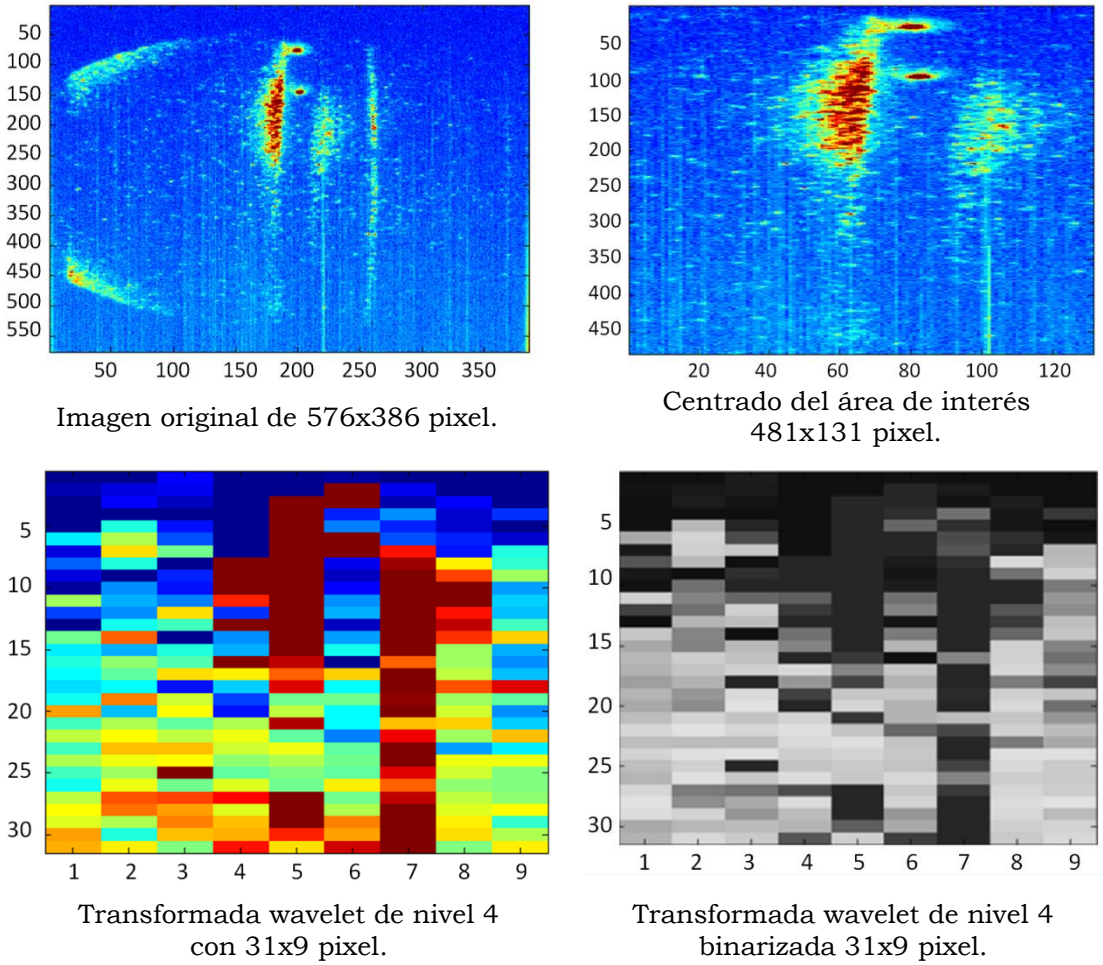


Figura 6.10. Ejemplo de las diferentes fases de preprocesamiento de la imagen de la señal CUTOFF hasta llegar a la imagen a clasificar.

en dimensionalidad, es decir, aptos para entregar al predictor. Por tanto el tiempo invertido en la reducción de dimensionalidad de los datos, no se ha tenido en cuenta en el cómputo de tiempos del proceso paralelo final. No obstante, se podría pensar en una paralelización de la reducción de dimensionalidad para hacer el procesamiento más rápido aún. Sobre esto último existen trabajos previos como se detalla en [Acevedo 2009] en referencia a la paralelización de la transformada wavelet de segundo orden.

6.6.2 Resultados.

Las pruebas se realizaron en el clúster DIA, con diferentes números de *workers* y diferentes funciones Kernel siendo la función de base radial (RBF) la que mejores resultados proporcionó. Se probó con números de bandas de 3 a 8. Los valores de C

fueron de 1 a 1000000 y para sigma de 1 a 100. La figura 6.11 y la tabla 6.7 muestran los resultados de la paralelización para la base de datos de formas de onda del TJ-11. En la gráfica del speedup de la figura 6.11 se aprecia que el speedup aumenta incluso superada la treintena de *workers,* no obstante se comprueba que, a partir de un número de 26 *workers,* el sistema no tiene grandes progresos. A pesar de bajar el tiempo de proceso a medida que se incrementa el número de *workers,* no parece muy efectivo este incremento. Incluso se puede apreciar que se alcanza el límite de Amdhal en torno a los 38 *workers*. Cabe recordar que añadir más *workers* a la agrupación requiere incrementar la memoria que dedica el front-end y dedicar tiempo a la gestión de estos nuevos *workers* y esto encarece el tiempo de respuesta.

Número de *Workers*	Segundos
1	13108
5	3830
9	2018
13	1404
17	1114
21	1028
26	934
30	902
35	872
39	898

Tabla 6.7. Tiempos de proceso vs. *workers*.

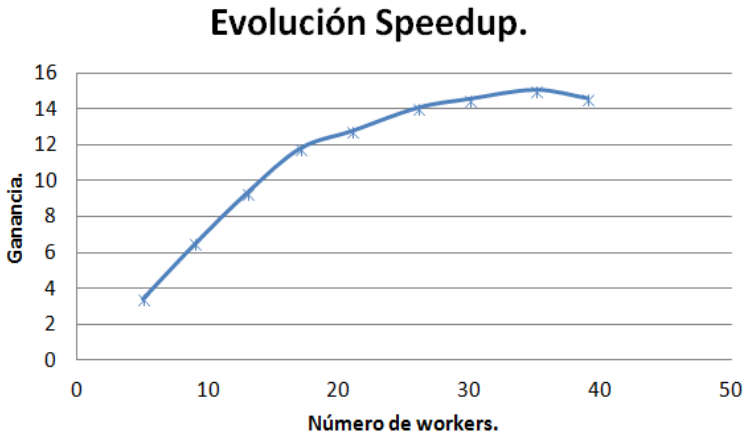


Figura 6.11. Progreso de aceleración para el Predictor Venn aplicado a formas de onda del TJ-II.

Las gráficas de las figuras 6.12 a 6.17 y tablas 6.8 a 6.9 muestran las tasas de aciertos asi como los intervalos de probabilidad referidos en la matriz *P* de probabilidades vista en la sección 6.4 para las bases de datos análisis de vehículos, dermatología y formas de onda del TJ-II. En cada gráfica se muestran tres series. Cada serie se calcula en base a las fórmulas 6.23 a 6.25 vistas en la sección 6.2.2.2. La serie color negro se refiere al intervalo superior y azul para el intervalo inferior. Cuanto más cercanos estén estos dos intervalos y la gráfica de tasa de acierto esté en el interior, más acertada es la predicción del modelo y el motivo es que el intervalo de probabilidad es más pequeño. Debido a esto se puede observar que, en el inicio del proceso y con un número bajo de muestras de entrenamiento, estas dos gráficas están muy separadas. A medida que el modelo adquiere más

"inteligencia" su predicción es más certera. Como se comentó en la sección 6.2.2.1, el método on-line comienza con un número muy bajo de muestras y después de cada predicción se añade la muestra analizada al conjunto de entenamiento de su clase. Esto hace que después de cada análisis el modelo cuente con más muestras para mejorar su predicción. La efectividad (Acc.) o tasa de acierto viene definida por la gráfica roja. Al principio de la clasificación la tasa de acierto es baja porque el clasificador solo cuenta con pocas muestras en cada clase. A medida que analiza nuevas muestras, el predictor aprende y mejora la clasificación porque éstas muestras de test analizadas se incorporan al conjunto de entrenamiento.

Para verificar la exactitud y calibrado del predictor basta con fijarse en la figura 6.17. En ella se observa cómo los límites superior e inferior están muy cercanos y tienden a acercarse cada vez más a la gráfica de la tasa de acierto. Esto quiere decir que el predictor está bien calibrado.

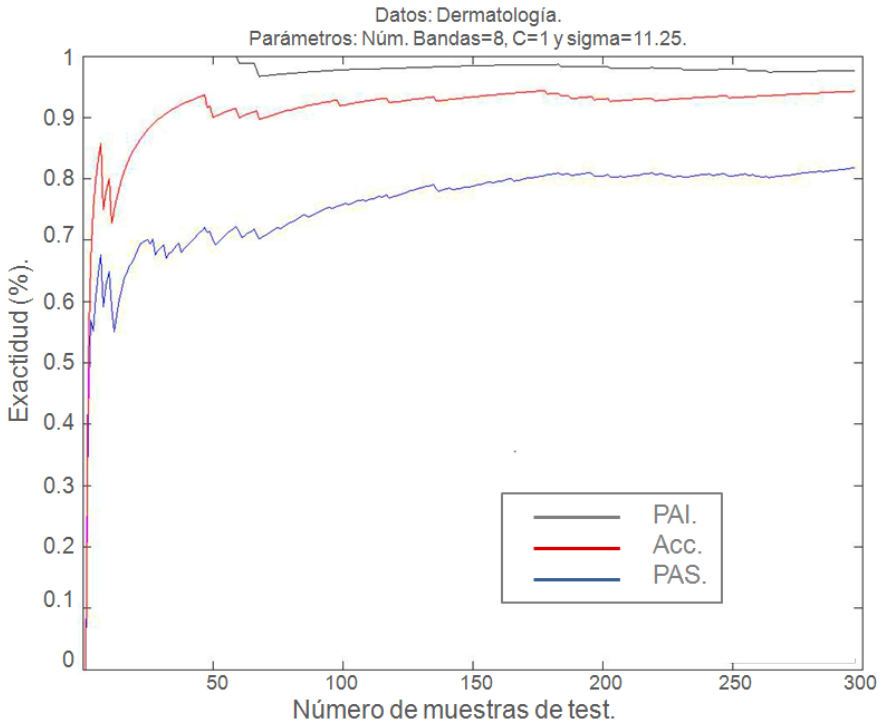


Figura 6.12. Análisis de la base de datos de dermatología UCI. Con número de bandas igual a 8, sigma igual a 11.25 y *C* igual a 1.

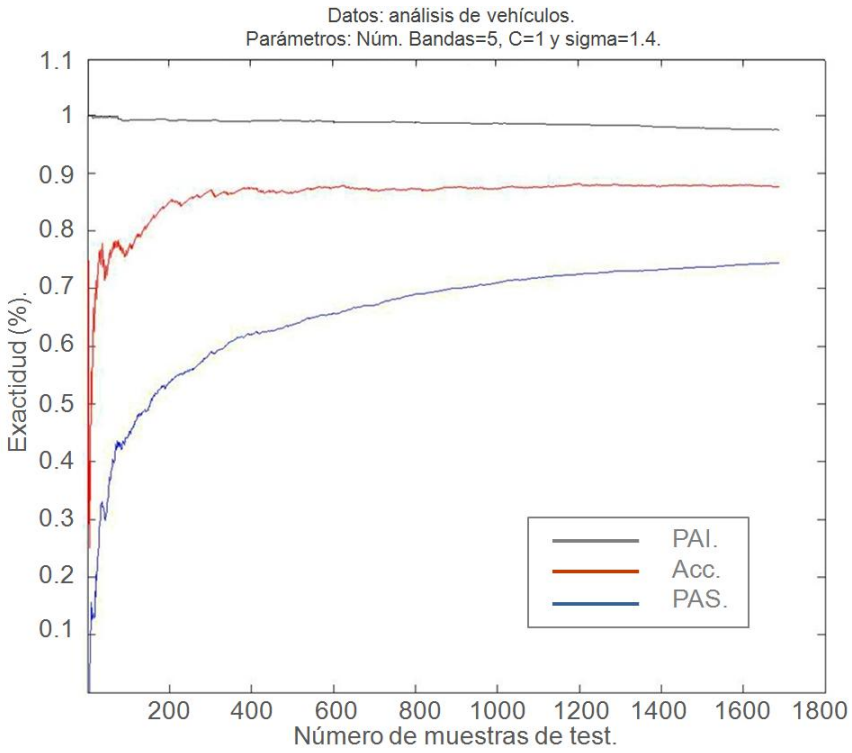


Figura 6.13. Análisis de la base de datos de vehículos de UCI. Con número de bandas igual a 5, sigma igual a 1.4 y *C* igual a 1.

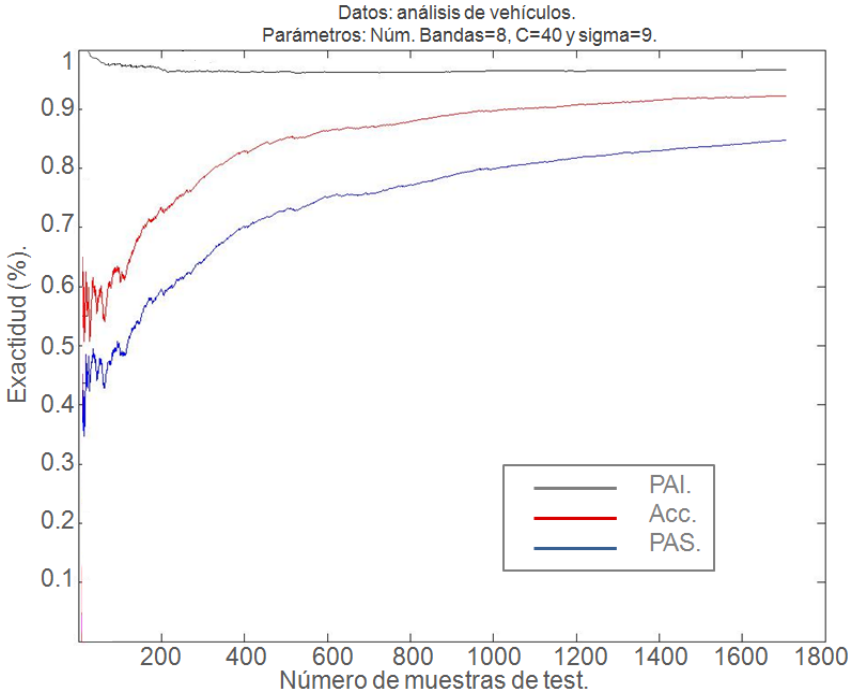


Figura 6.14. Análisis de la base de datos de vehículos de UCI. Con número de bandas igual a 8, sigma igual a 9 y *C* igual a 40.

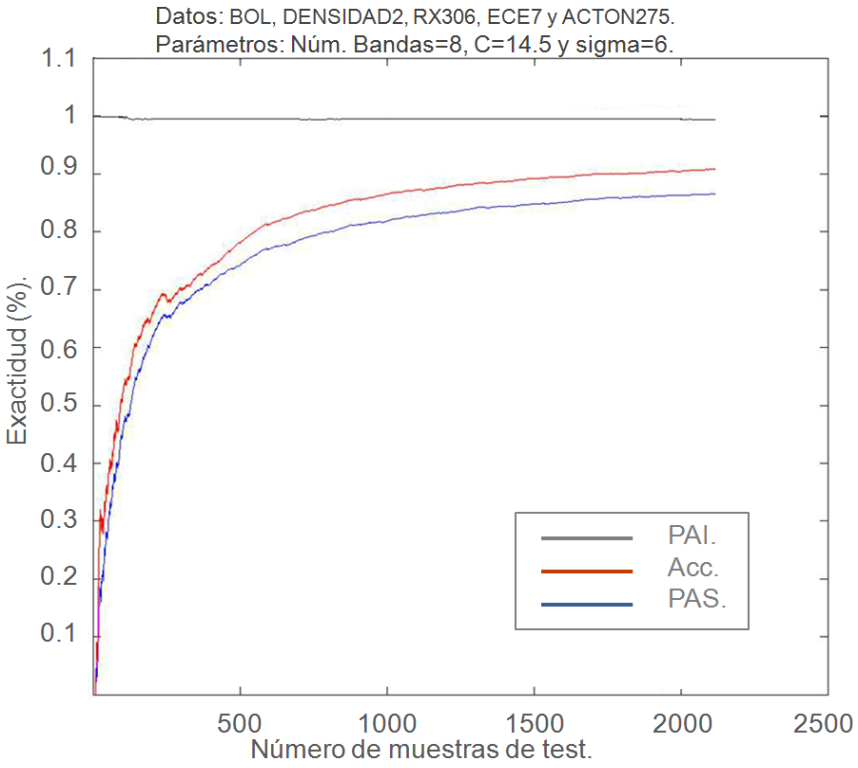


Figura 6.15. Análisis de la base de datos de formas de onda del TJ-II. Con número de bandas igual a 8, sigma igual a 6 y *C* igual a 14.5.

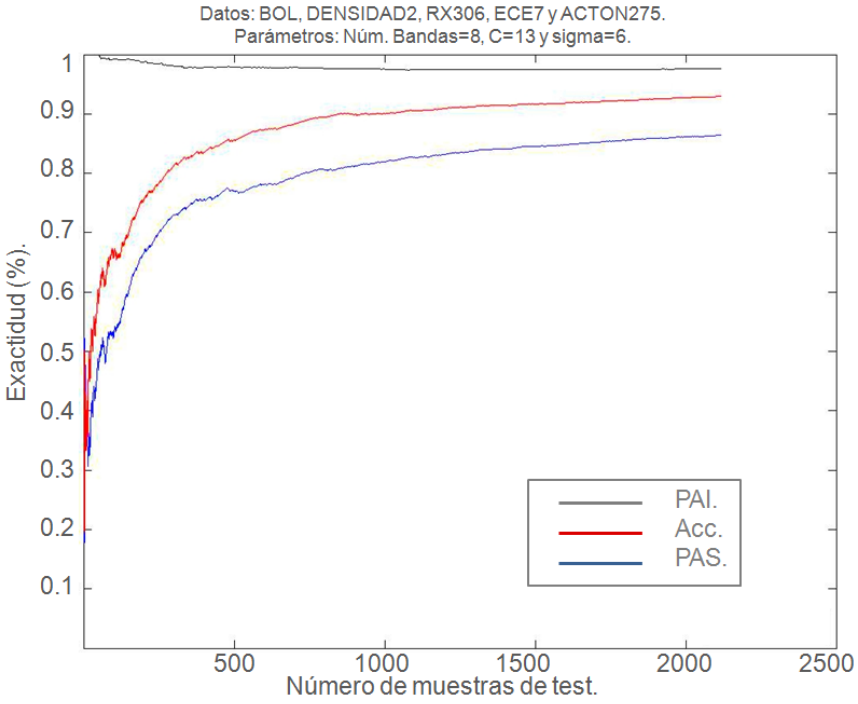


Figura 6.16. Análisis de la base de datos de formas de onda del TJ-II. Con número de bandas igual a 8, sigma igual a 6 y *C* igual a 13.

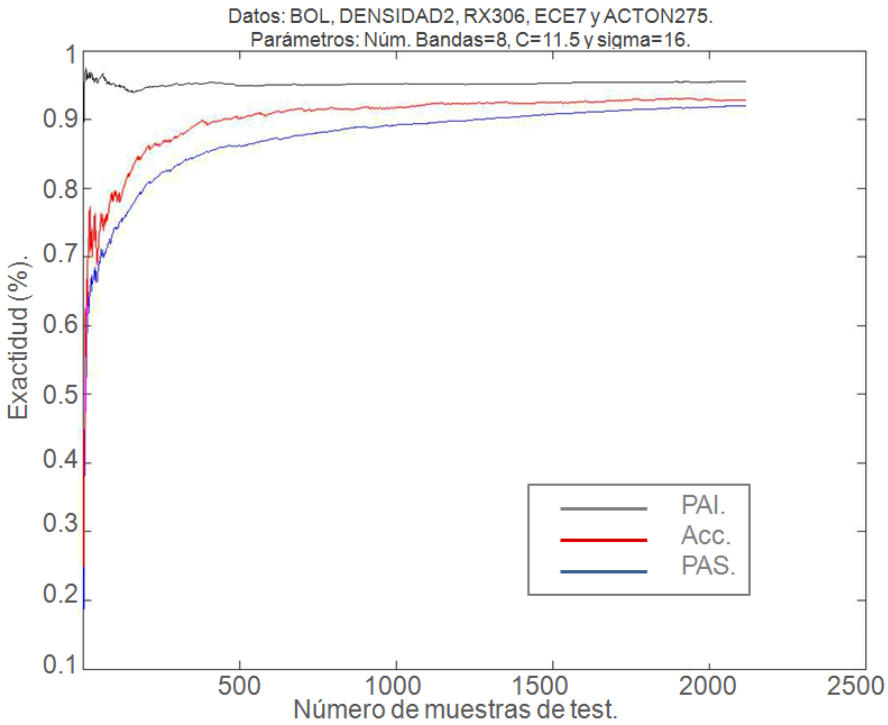


Figura 6.17. Análisis de la base de datos de formas de onda del TJ-II. Con número de bandas igual a 8, sigma igual a 1.6 y *C* igual a 11.5.

Las gráficas de las figuras 6.18 a 6.21 muestran el proceso de calibración del predictor. En ellas se pueden comprobar como afecta el parámetro sigma en el proceso de clasificación y ver la respuesta del modelo. En la figura 6.18, para unos valores de bandas=8, *C*=6 y sigma=9 se obtiene un porcentaje alto de acierto, aproximadamente un 97% pero se puede ver que la gráfica de la tasa de acierto (Acc.) no está entre los límites superior e inferior. Esto indica que el predictor no esta bien calibrado.

En la figura 6.19 se han variado los valores pasando a ser de bandas=8, *C*=6 y sigma=7. La tasa de acierto baja ligeramente pero todavía está por encima del límite superior. En la figura 6.20 se baja un punto el valor de sigma, se ajusta en 6 y se comprueba que la gráfica de tasa de acierto entra en los límites y tiende a estabilizarse.

Para terminar, en la figura 6.21 con unos valores de bandas=8, *C*=6 y sigma=5.7 la gráfica de la tasa de acierto se hace estable entre los límites superior e inferior. Si el valor de sigma siguiera disminuyendo se alcanzaría el límite inferior llegando incluso a superarlo para seguir bajando. Estas figuras muestran el

proceso de calibrado del predictor Venn, pero el aspecto más importante que indican estas figuras es que una tasa de acierto alta (figura 6.18) no indica necesariamente una predicción correcta.

Quizá un clasificador sin la capa adicional de un predictor Venn daría altos índices de efectividad para unos determinados valores de *C* y sigma según se ha visto en la figura 6.18, pero con el predictor Venn se ve realmente que unos valores altos de efectividad no siempre garantizan que la predicción sea la más acertada.

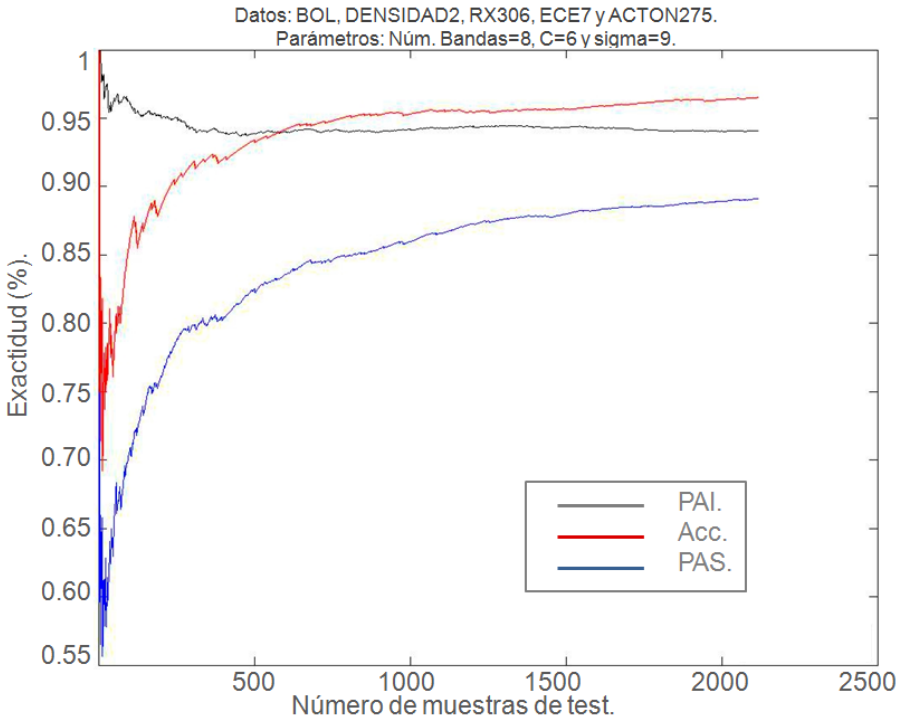


Figura 6.18. Análisis de la base de datos de formas de onda del TJ-II. Con número de bandas igual a 8, sigma igual a 9 y *C* igual a 6.

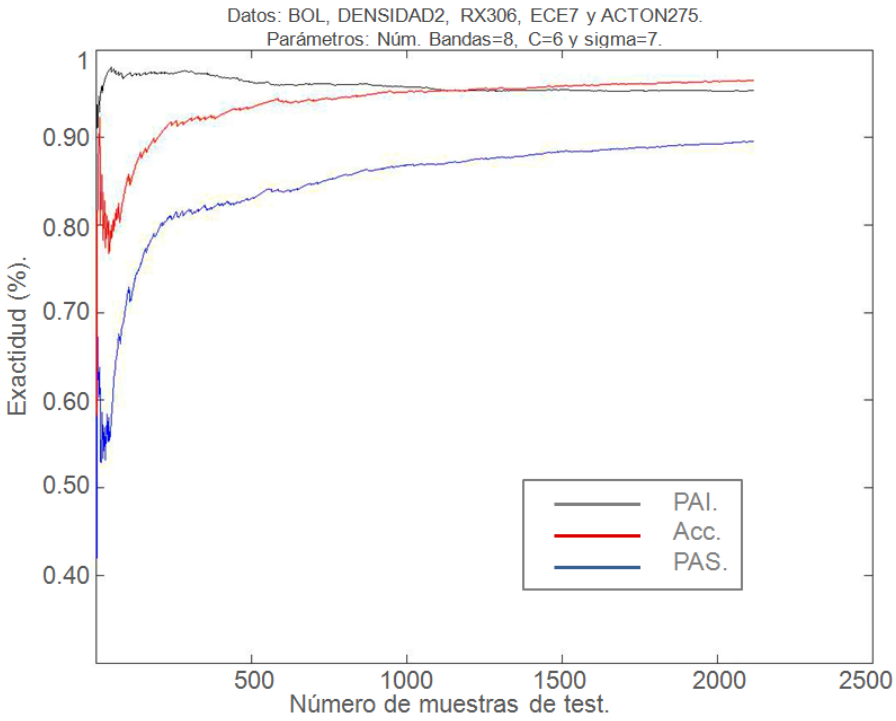


Figura 6.19. Análisis de la base de datos de formas de onda del TJ-II. Con número de bandas igual a 8, sigma igual a 7 y *C* igual a 6.

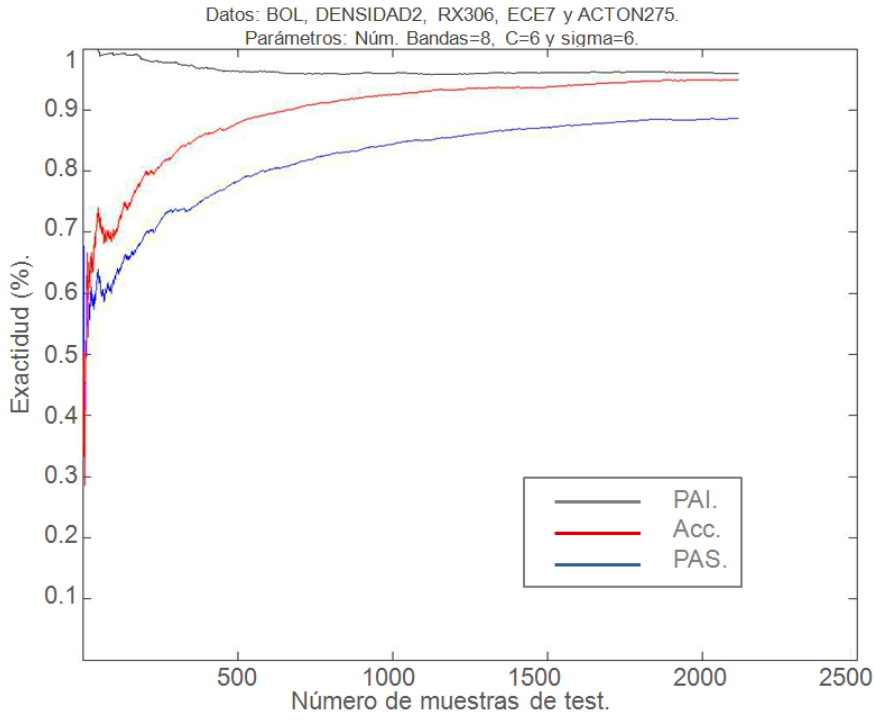


Figura 6.20. Análisis de la base de datos de formas de onda del TJ-II. Con número de bandas igual a 8, sigma igual a 6 y *C* igual a 6.

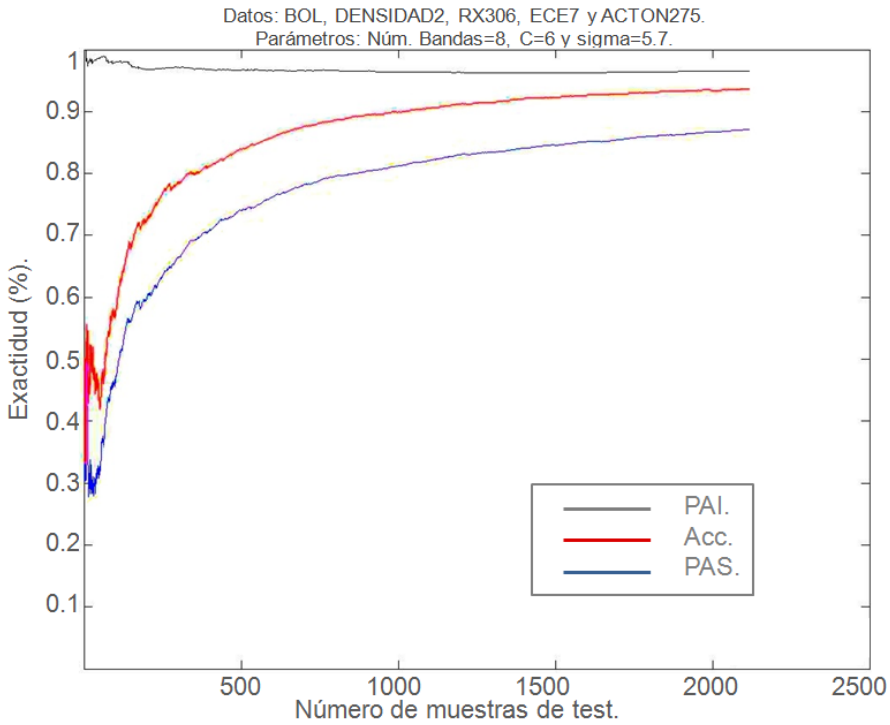


Figura 6.21. Análisis de la base de datos de formas de onda del TJ-II, con número de bandas igual a 8, sigma igual a 5.7 y *C* igual a 6.

Tasa de acierto	C	Sigma	Nº Bandas	Intervalos de probabilidad	Probabilidad promedio	Longitud Intervalo probabilidad
94,58 %	11,53	16	8	91,89 % - 96,78 %	94,33 %	0,04
91,76 %	14,54	6	8	85,78 % - 99,64 %	92,71 %	0,13
92,76 %	13,12	6	8	86,72 % - 98,65%	92,68 %	0,11

Tabla 6.8. Tasa de acierto del Predictor Venn para formas de onda del TJ-II.

Base de datos	Tasa de acierto	C	Sigma	Nº Bandas	Intervalos de probabilidad	Probabilidad promedio	Longitud Intervalo probabilidad
Dermatología	96,29 %	1	11,25	8	82,76 % - 98,55 %	90,65 %	0,15
Evaluación vehículo	91,76 %	40	9	8	86,10 % - 98,33 %	92,21 %	0,12

Tabla 6.9. Tasa de acierto del Predictor Venn para las bases de datos Dermatología y Evaluación vehículos.

Los resultados del análisis de las imágenes de esparcimiento Thomson TJ-II se muestran en las tablas de la 6.10 a 6.12 y figuras de la 6.22 a 6.25. La figura 6.25 hace referencia al IVP. Los datos fueron reducidos en dimensionalidad en un nivel 5, 6 y 7 de transformada wavelet si bien los mejores resultados se consiguieron con el nivel 6 llegando a estar por encima del 96% de tasa de acierto y para diferentes números de banda estando los modelos bien calibrados. Se puede comprobar que el predictor Venn con más muestras iniciales para generar el modelo obtiene mejores intervalos (mas pequeños) de probabilidad. La tabla 6.10 muestra diferentes resultados para tres modelos construidos. Cada modelo comienza con tres conjuntos iniciales de diferentes dimensiones (10, 20 y 30 muestras). Para el primer modelo (bandas=4, *C*=10 y sigma=13) tan solo el construido con 30 muestras iniciales consigue estar dentro de los límites inferior y superior. El modelo construido con bandas=8, *C*=100 y sigma=27 es el que obtiene mejores resultados con menor número de muestras inciales. Además este modelo consigue ser estable. Estos resultados tienen mucha relevancia en el sentido que, para alguna clase, este número de muestras supone el 2,16 % de las muestras total de la clase. Esto quiere decir que el modelo comienza con un número muy pequeño de muestras para el entrenamiento.

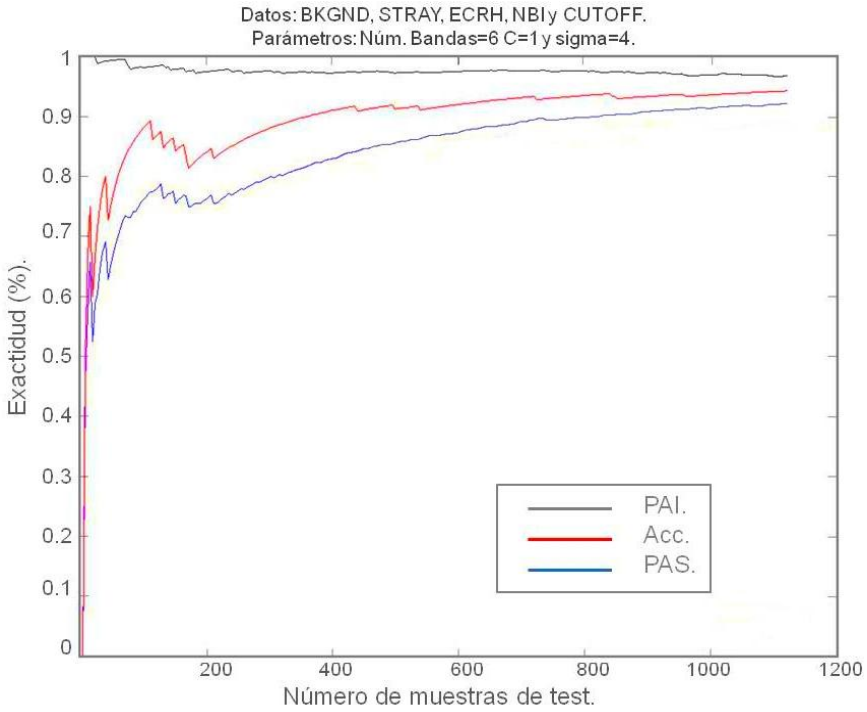


Figura 6.22. Análisis de la base de datos de imágenes de esparcimiento Thomson TJ II. Con número de bandas igual a 6, sigma igual a 4 y *C* igual a 1.

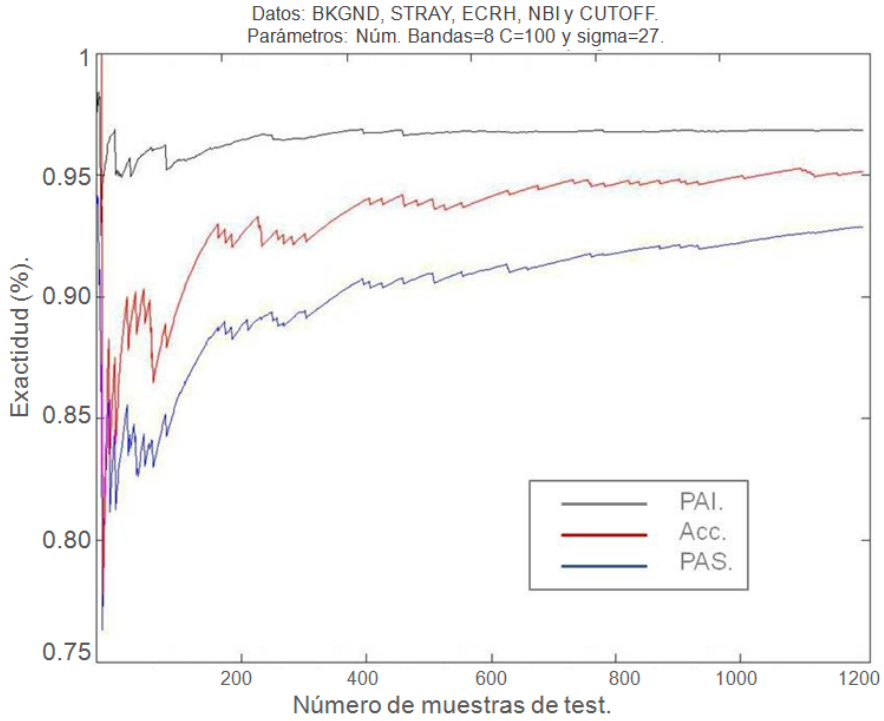


Figura 6.23. Análisis de la base de datos de imágenes de esparcimiento Thomson TJ II con 30 muestras/clase iniciales para generar el modelo.

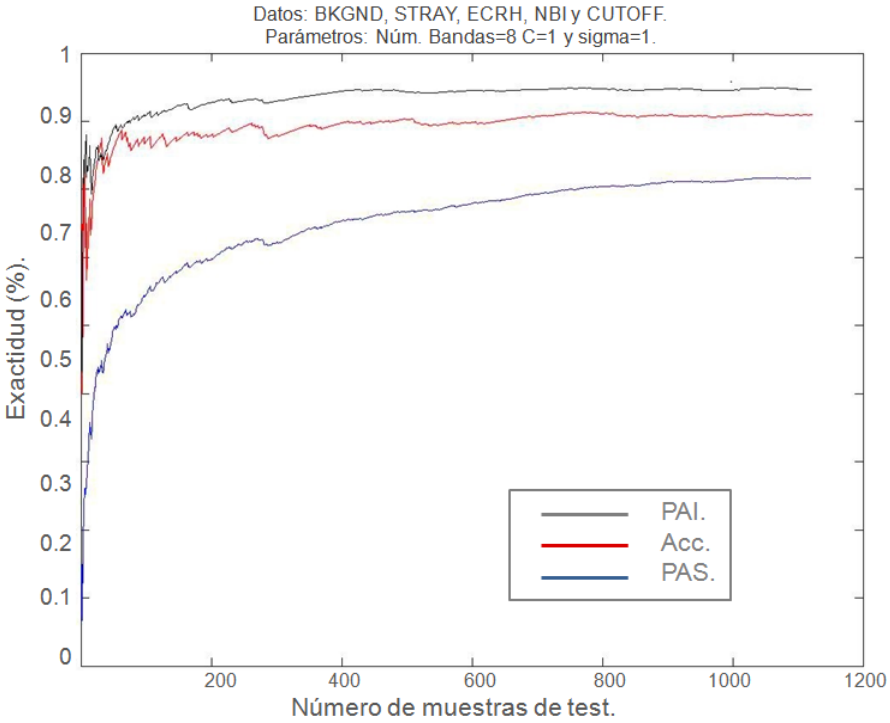


Figura 6.24. Análisis de la base de datos de imágenes de esparcimiento Thomson TJ II mediante Predictor Venn Inducido (IVP). Con número de bandas igual a 8, sigma igual a 1 y *C* igual a 1.

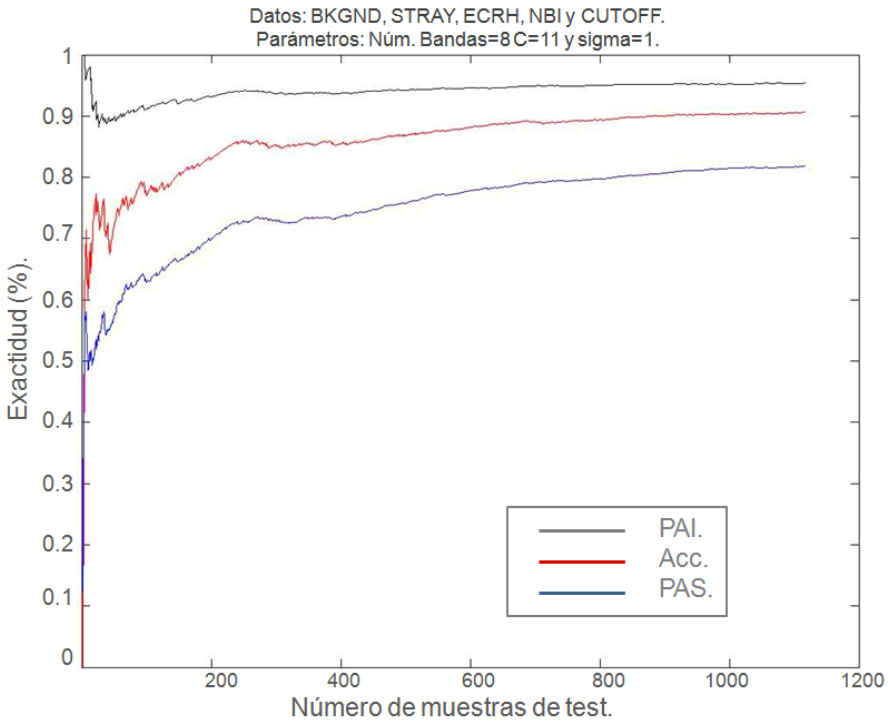


Figura 6.25. Análisis de la base de datos de imágenes de esparcimiento Thomson TJ II mediante Predictor Venn Inducido (IVP). Con número de bandas igual a 8, sigma igual a 1 y *C* igual a 11.

Los resultados de tiempos del análisis paralelo se muestran en la tabla 6.13 y figuras 6.26-6.27 donde se pueden ver las gráficas del tiempo invertido y el speedup para los dos tipos de predictor. En este caso se puede ampliar el número de *workers* hasta 51. Disponer de una base de datos pequeña ayuda para poder generar mayor número de *workers* (ver sección 2.8.3.3). No obstante generar más *workers* no genera beneficio pues se puede comprobar que a partir de 30 *workers* no hay mejora del tiempo de respuesta para el TVP y en el caso del IVP empeora. Además a partir del *worker* 43 hay caída del rendimiento para el TVP. El IVP consigue mayor velocidad pero los resultados son inferiores como era de esperar. La tabla 6.11 muestra los resultados para este predictor. El TVP tiene mayor robustez respecto al IVP pues este último solo consigue buenos resultados para un número de bandas igual a 8, sigma igual a 1 y *C* para valores de 1 a 11. Sin embargo el TVP a pesar de ser ligeramente más lento, sus tres parámetros tienen un rango de valores mucho más amplio y obtiene mejores resultados.

Tasa de acierto	Muestras iniciales	C	Sigma	Nº Bandas	Intervalo de probabilidad	Probabilidad promedio	Longitud Intervalo probabilidad
96,58 %	30	10	13	4	92,32 % - 96,73 %	94,52 %	0,04
96,34 %	30	1	4	6	91,83 % - 96,35 %	94,09 %	0,04
95,32 %	30	100	27	8	92,67 % - 97,32 %	94,99 %	0,04

Tabla 6.10. Tasa de acierto del Predictor Venn para imágenes esparcimiento Thomson TJ-II.

Tasa de acierto	Muestras iniciales	C	Sigma	Nº Bandas	Intervalo de probabilidad	Probabilidad promedio	Longitud Intervalo probabilidad
90,53 %	10	1	1	8	81,20 % - 92,36 %	86,78 %	0,11

Tabla 6.11. Tasa de acierto del Predictor Venn inducido para imágenes esparcimiento Thomson TJ-II.

Muestras iniciales	**C**	**Sigma**	**Nº Bandas**	**Tasa de acierto**	**Intervalo de probabilidad**	**Probabilidad promedio**	**Longitud Intervalo probabilidad**	**Fuera de intervalo**
10	1	4	6	94,96 %	91,62 % - 98,04 %	95,01 %	0,064	NO
20	1	4	6	95,39 %	92,61 % - 97,59 %	95,10 %	0,049	NO
30	1	4	6	96,34 %	91,83 % - 96,35 %	93,82 %	0,045	NO
10	10	13	4	95,32 %	93,34 % - 96,13 %	94,82 %	0,027	NO
20	10	13	4	96,21 %	94,63 % - 95,35 %	94,99 %	0,007	SI
30	10	13	4	96,58 %	92,32 % - 96,73 %	94,52 %	0,044	NO
10	100	27	8	95,84 %	90,83 % - 97,64 %	94,23 %	0,068	NO
20	100	27	8	95,84 %	91,83 % - 96,47 %	94,15 %	0,046	NO
30	100	27	8	95,32 %	92,67 % - 97,32 %	94,99 %	0,046	NO

Tabla 6.12. Tasa de acierto del Predictor Venn para imágenes esparcimiento Thomson TJ-II con diferente número de muestras iniciales.

Número de *workers*	TVP (sg.)	Número de *workers*	TVP (sg.)	Número de *workers*	IVP (sg.)	Número de *workers*	IVP (sg.)
2	2358,98	30	131,42	2	943,59	30	78,85
5	630,87	35	131,96	5	252,34	35	105,56
10	410,23	39	131,56	10	164,09	39	118,42
15	220,43	43	131,45	15	88,17	43	197,17
20	180,65	47	132,34	20	72,26	47	224,97
25	141,76	51	133,32	25	70,88	51	293,32

Tabla 6.13. Relación de tiempos de Proceso vs. Número de *workers*.

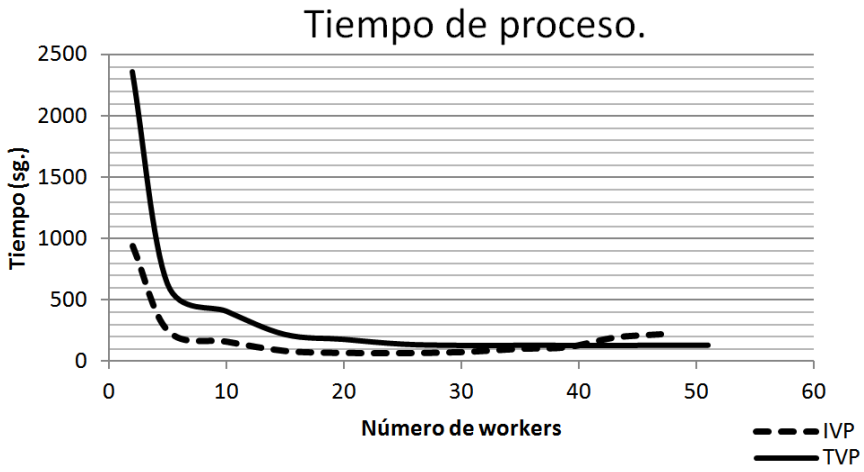


Figura 6.26. Tiempo de proceso para el Predictor Venn (TVP vs. IVP) aplicado a imágenes esparcimiento Thomson TJ-II.

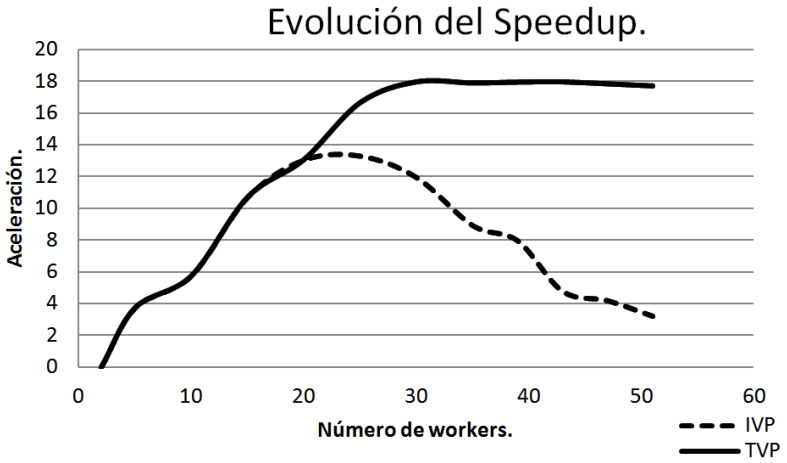


Figura 6.27. Progreso de aceleración para el Predictor Venn (TVP vs. IVP) aplicado a imágenes esparcimiento Thomson TJ-II**.**

Capítulo VII. Conclusiones.

Los escenarios para la búsqueda de información cambian día a día. Hasta hace poco el entorno de búsqueda se centraba en datos almacenados como variables multidimensionales. No cabe duda que las búsquedas de información avanzan hacia entornos con información poco o nada estructurada y que es necesario procesar en poco tiempo.

Como punto de partida del trabajo y primer objetivo era habilitar un entorno paralelo en el clúster del departamento DIA de la UNED que sirviera no solo para implementar las aplicaciones desarrolladas en esta Tesis doctoral, sino para poder utilizarse en futuros trabajos de investigación. Se implementó un entorno de trabajo Matlab. Se trata de un potente lenguaje y que posee una toolbox para generar un entorno paralelo lo que hace que sea el más idóneo para este trabajo. Además del entorno paralelo, este lenguaje posee un potente conjunto de instrucciones para manejar grandes conjuntos de datos.

Una vez generado el entorno paralelo en el clúster del dpto. DIA (UNED), se procedió a realizar las primeras pruebas de capacidad del sistema y se comprobó que el sistema tenía disponibilidad para aceptar altas cargas de trabajo. Se procedió por tanto al análisis y búsquedas de patrones concretos como son los modos multiarmónicos detectados en ciertas descargas de plasmas nucleares. Los hallazgos coincidieron en su totalidad con los detectados por el experto. El modelo fue capaz de aportar el/los intervalo/s temporal/es de cada uno de los armónicos.

La metodología propuesta de automatización de patrones abre una vía complementaria. Es decir la búsqueda y clasificación pueden ser dos funciones totalmente complementarias. Además la misma metodología propone una paralelizacion de datos y de tareas. Puede haber diferentes usuarios utilizando agrupaciones paralelas en el mismo clúster. Este hecho motiva que sea necesario disponer de una estrategia de trabajos que optimize los recursos del sistema para dar servicio a todos los usuarios. Día a día la computación de alto rendimiento está más extendida. Esto quiere decir que, en el futuro, los sistemas se harán más paralelizables. Será imperativo disponer de estrategias de planificación que sean capaces de extraer el mayor partido a los sistemas.

Haciendo foco en el entorno de la clasificación, se han analizado y paralelizado diferentes métodos de clasificación mediante máquinas de vectores soporte para grandes volúmenes de datos. Tomando como base la librería de Matlab para SVM, se paralelizaron tres implementaciones diferentes del algoritmo SVM. Los datos de entrada al clasificador fueron cinco tipos de señales de evolución temporal del TJ-II. Los resultados de clasificación fueron óptimos generando altos valores de tasas de acierto. Además la implementación paralela mostró la escalabilidad del modelo lo que le hace ser en conjunto una buena herramienta como clasificador multiclase.

La paralelización como hilo conductor lleva al último objetivo del trabajo. El desarrollo de una taxonomía basada en máquinas de vectores soporte en los predictores Venn. Las tasas de acierto son buenas y muestran que la paralelización implementada es correcta pues el sistema es capaz de mantener una buena tasa de acierto. Respecto a los intervalos de probabilidad comentar también que se consiguieron intervalos reducidos lo cual repercute en predicciones más fiables. Para validar la robustez de la implementación realizada se utilizaron distintas bases de datos de diferente naturaleza.

Líneas futuras de trabajo.

Indudablemente el futuro pasa por la computación paralela. Es manifiesto que la sociedad está apostando por esta tecnología y se perfila por tanto como un campo de estudio de largo recorrido. Los sistemas en el futuro harán uso cotidiano de la computación paralela. En este sentido, los actuales sistemas que por ejemplo procesan grandes cantidades de datos muy probablemente cambiarán la forma de utilizar los recursos de los sistemas. Para optimizar estos resursos será necesario modelar y muy probablemente paralelizar la carga de trabajo y de esta forma conseguir modelos con mejor respuesta. Por ese motivo, el modelado de la carga se presenta como motivo de estudio con el fin de poder predecir a priori, la carga computacional que puede llevar a cabo cada tarea para poder balancear con bastante exactitud la ocupación de los procesadores que intervienen en la resolución del problema.

La segmentación de imágenes es otro campo que abre nuevas expectativas. Se ha comentado en esta memoria la posibilidad de mantener un repositorio de patrones para comparar regiones determinadas de una imagen. Mantener un repositorio de patrones abre un abanico de posibilidades. Dado un patrón se trataría por tanto de realizar búsquedas muy efectivas y automáticas para la

detección de patrones en bases de datos masivas. De esta forma se podría pensar en un sistema de clasificación basado en patrones que unido a la computación paralela aceleraría el tiempo de respuesta.

Profundizar en la paralelización de algoritmos de clasificación es otra línea de investigación. Los algoritmos Dual Coordinate Descend y Stochastic Subgradient Descent se presentan como una opción a tener en cuenta como algoritmos subyacentes. Los predictores Venn demuestran ser una herramienta eficaz mediante una paralelización sencilla lo que hace de ellos una potente herramienta para ser aplicada en más entornos como puede ser el análisis de regiones de un imagen visto en el apartado 4.4.1. Por último se podría pensar en paralelizar la aplicación de los predictores Venn a un mayor número de algoritmos subyacentes.

Los avances realizados en el campo de la computación en la nube, más conocido por el término inglés de Cloud Computing (figura 7.1), se presenta como el futuro. El Cloud Computing puede ser visto como una computación paralela donde se distribuyen servicios de todo tipo: es posible tener unos centros de computación, otros de almacenamiento, etc.

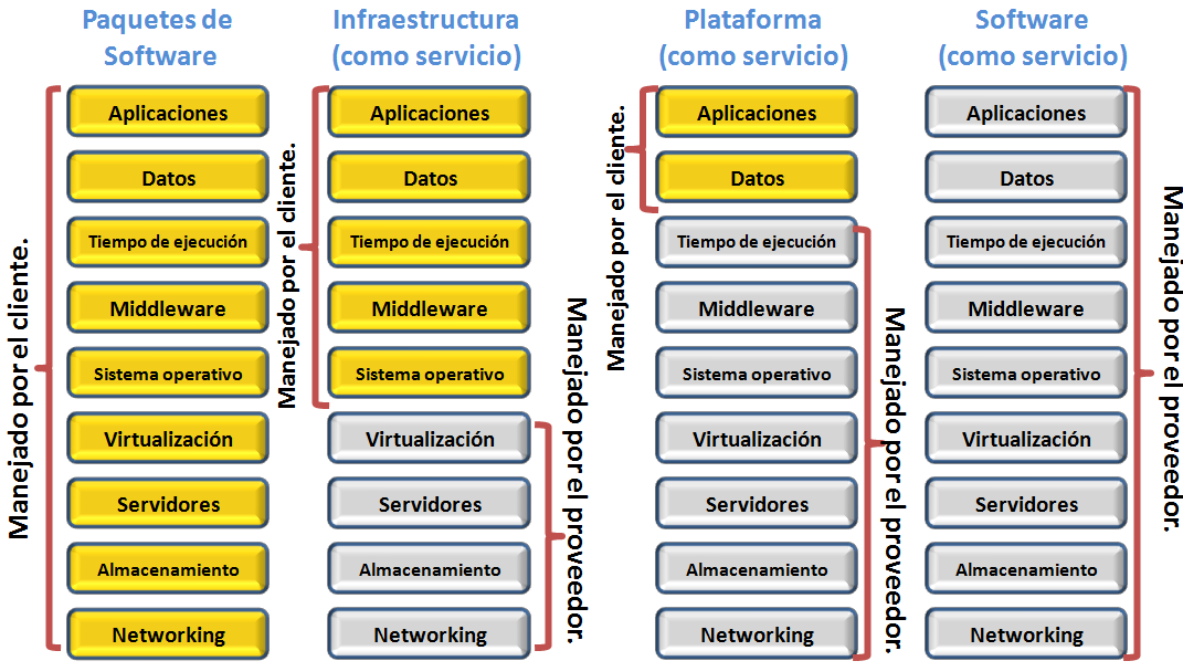


Figura 7.1. Tipos de servicios Cloud.

La combinación de todos estos elementos hace que surja la necesidad de tener herramientas que permitan la virtualización. En este sentido la última versión de Matlab (R2016b) ofrece estas posibililidades. Se refuerzan las instrucciones orientadas al Cloud Computing. Aparecen nuevas funciones para trabajar por ejemplo con vectores de altas dimensiones en memoria, o bien es posible acceder de forma transparente a datos remotos. También esta nueva versión de Matlab refuerza una rama del aprendizaje automático que es el *Deep Learning*. Éste es especialmente adecuado para el reconocimiento de imágenes. En este sentido cobra más importancia lo comentado en el capítulo 4 que indica la posibilidad de generar una base de datos con regiones o bien patrones descubiertos previamente (figura 7.2).

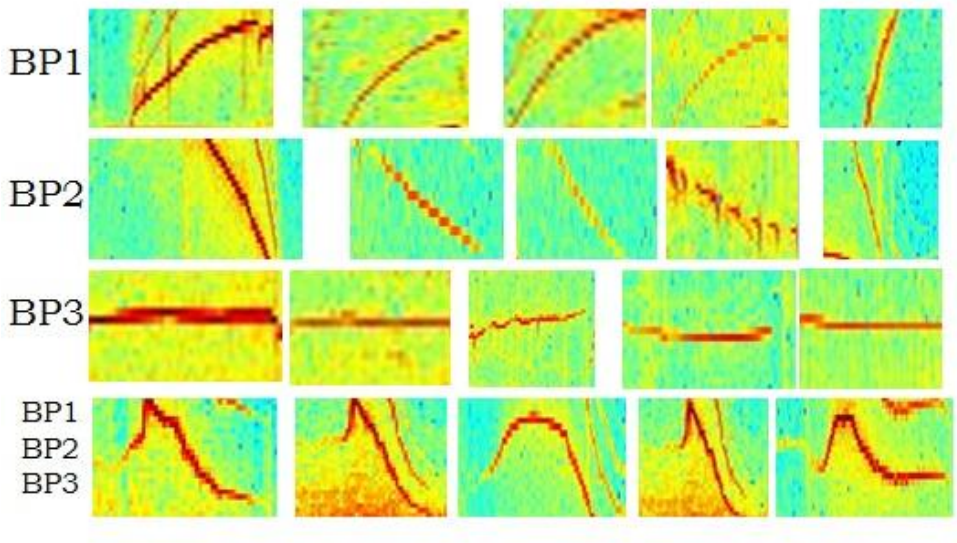


Figura 7.2. Diferentes patrones BP1, BP2, BP3 y BP1-BP2-BP3.

Bibliografía.

[Acevedo 2009] L. Acevedo. *"Computación Paralela de la Transformada Wavelet; Aplicaciones de la transformada Wavelet al álgebra lineal numérica".* Tésis doctoral. Universidad de Valencia. (2009).

[Adams 1994] R. Adams y L. Bischof. Seeded región growing. *"Pattern Analysis and Machine Intelligence".* IEEE Transactions on 16(6), 641–647 (1994).

[Agarwal 2012] Sameer Agarwal, Srikanth Kandula, Nicolas Bruno, Ming-Chuan Wu, Ion Stoica, Jingren Zhou. *"Re-optimizing Data-Parallel Computing".* Microsoft Research, Microsoft Bing, University of California, Berkeley

[Alejaldre 1990] C. Alejaldre, *"TJ-II project: A flexible heliac stellarator".* Technol. 17, 131 (1990).

[Alejaldre 1999] C. Alejaldre, *"First plasmas in the TJ-II flexible heliac".* Plasma Phys. Control. Fusión 41, A539 (1999).

[Alejaldre 2001] C. Alejaldre., *"Review of confinement and transport studies in the TJ-II flexible heliac".* Nuclear Fusión 41, 1449 (2001).

[Al-mouhamed 1990] Al-Mouhamed M. *"Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communications Cost".* IEEE Transactions on Software Engineering, 16 (12): 1390-1410, 1990.

[Amdahl 1967a] 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.

[Amdhal 1967b] G. M. Amdahl. *"Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities".* In AFIPS Conference Proceedings, pages 483–485, 1967.

[Anderson 2006] Anderson J. and Calandrino J. *"Parallel task scheduling on multicore platforms".* Special Interest Group on Embedded Systems (SIGBED) ACM Review - Special issue: The work-in-progress (WIP), Vol. 3, Issue 1, pp.1-6, 2006.

[Anlauf 1989] J. K. Anlauf y M. Biehl, "*The AdaTron: an adaptive perceptron algorithm".* Europhys. Lett., vol. 10, pp. 687-692, 1989.of Sheeld Dept. of ACSE.

[Arana-Daniel 2016] Arana-Daniel, N., Gallegos, A. A., López-Franco, C., Alanís, A. Y., Morales, J., & López-Franco, A. (2016). "*Support Vector Machines Trained with Evolutionary Algorithms Employing Kernel Adatron for Large Scale Classification of Protein Structures*". Evolutionary Bioinformatics Online, 12, 285.

[Bacci 1999] Bacci, Danelutto, Pelagatti, Vaneschi. *"SklE :A heterogeneous Environment for HPC Applications".* Parallel computing 25, pp. 1827-52. 1999.

[Baldi 2000] Baldi P, Brunak S, Chauvin Y, Andersen CA, Nielsen H. *"Assessing the accuracy of prediction algorithms for classification: an overview".* Bioinformatics. 2000, 16 (5):412 -424.1093/bioinformatics/ 16.5.412

[Barth 1999] C.J. Barth, *"High-resolution multiposition Thomson scattering for the TJ-II stellarator".* Rev. Sci. Instrum. 70, 763 (1999).

[Bennet 1997] Kristin P. Bennett and Erin J. Bredensteiner. *"Geometry in Learning. Geometry at Work"*. 1997.

[Bennet 2000] Kristin P. Bennett and Erin J. Bredensteiner. "*Duality and Geometry in SVM Classifers".* Proc. 17th International Conf. on Machine Learning, pages 57-64, 2000.

[Bernhard 2002] Bernhard Scholkopf and Alex Smola. *"Learning with Kernels Support Vector Machines, Regularization, Optimization and Beyond*". MIT Press, Cambridge, MA, 2002. ISBN 0-387-94559-8.

[Bohanec 1998] Marko Bohanec and Vladislav Rajkovi. *"Knowledge acquisition and explanation for multi-attribute decision making".* In 8th International Workshop "Expert Systems and Their Applications", 1988.

[Bondarenko 2001] I.S. Bondarenko. *"Installation of an advanced heavy ion beam diagnostic on the TJ-II stellarator".* Rev. Sci. Instrum. 72, 583 Part 2 (2001).

[Boser 1992] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. *"A training algorithm for optimal margin classifiers".* In Proceedings of the 5th AnnualACM Workshop on Computational Learning Theory, pages 144–152. ACMPress, 1992.

[Bottou 2010] *"Large Scale Kernel Macines"*. León Bottou, Olivier Chapelle, Dennis deCoste and Jason Weston. The Mit Press 2010.

[Brañas 2001] B. Brañas et al., *"Atomic beam diagnostics for characterization of edge plasma in TJ-II stellarator"*. Rev. Sci. Instrum. 72, 602 (2001).

[Brin 1995] S. Brin. *"Near neighbour search in very large databases".* In 21st International Conference on Very Large Data Bases, pages 574-584. Morgan Kaufmann Publishers, September 1995.

[Buison 2007] J. Buisson, O. Sonmez, H. Mohamed, W. Lammers, and D. Epema. *"Scheduling malleable applications in multicluster systems".* In Proc. of the IEEE International Conference on clúster Computing 2007, pages 372–381, 2007.

[Buyya et al. 1999] Buyya R. "High Performance clúster Computing: Architectures and Systems". Prentice Hall PTR Upper Saddle River, New Jersey, USA, 1999.

[Cairns 1993] R.A. Cairns, *"Radio-frequency plasma heating". "Plasma Physics: an Introductory Course".* Cambridge Univesity Press, Cambridge (1993).

[Calzarossa 1993] M. Calzarossa, G. Serazzi. *"Workload characterization: a survey".* Proc. IEEE 81 (8) (August 1993) 1136–1150.

[Calzarossa 1995] M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, D. Tessera. *"A hierarchical approach to workload characterizationfor parallel systems, in: B"*. Hertzberger, G. Serazzi (Eds.), High-Performance.

[Canny et al. 1986] Canny, J.. "*A Computational Approach To Edge Detection".* IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[CEPBA 2005] Centro Europeo de Paralelismo de Barcelona. Grupo de soporte a la paralelización. *"Ejemplos de paralelizacion con Origin 2000 Silicon Graphics y DEC".* Alpha8400". CEPBA. Barcelona 2005.

[Chang 2011] Chang, E. Y. (2011). *"PSVM: Parallelizing support vector machines on distributed computers".* In Foundations of Large-Scale Multimedia Information Management and Retrieval (pp. 213-230). Springer Berlin Heidelberg.

[Chang et al.] C.-C. Chang and C.-J. Lin. *"LIBSVM : a library for support vector machines".* Technical report, Computer Science and Information Engineering, National Taiwan University, 2001-2004.

[Chen et al.] Tie Qi Chen y Yi Lu. *"Color image segmentation–an innovative approach".* Pattern Recognition 35(2), 395–405 (febrero 2002).

[Cheng 2001] H. D. Cheng, X. H. Jiang, Y. Sun, J. Wang. *"Color image segmentation: advances and prospects"*. Pattern Recognition 34(12), 2259–2281 (Diciembre 2001).

[Cheng et al., 2000] H. D. Cheng, Ying Sun. *"A hierarchical approach to color image segmentation using homogeneity".* Image Processing, IEEE Transactions on 9(12), 2071–2082 (2000).

[Cheng 2003] H. D. Cheng, J. Li. *"Fuzzy homogeneity and scale-space approach to color image segmentation".* Pattern Recognition 36(7), 1545–1562 (Julio 2003).

[Cheng 2004] H.D. Cheng, X.J. Shi. *"A simple and effective histogram equalization approach to image enhancement".* Department of Computer Science, Utah State University, Logan, UT 84322-4205, USA. Digital Signal Processing 14 (2004) 158–170.

[Cherkassky 1998] Cherkassky, B. V., Goldberg, A. V., Martin, P., Setubal, J. C., & Stolfi, J. (1998). *"Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms".* Journal of Experimental Algorithmics (JEA), 3, 8.

[Christianini et al.] Christianini, N., and J. Shawe-Taylor. *"An Introduction to Support Vector Machines and Other Kernel-based Learning Methods".* Cambridge University Press, 2000.

[Ciemat 2015] http://fusionsites.ciemat.es/energia-de-fusion/el-proceso-de-la-fusion/ . (consulta 30/05/2015)

[Clematisa 1999] Andrea Clematisa, Angelo Corana. *"Modeling performance of heterogeneous parallel computing systems".* Elsevier. Parallel Computing. Volume 25, Issue 9, September 1999, Pages 1131–1145.

[Coleman 1996] Thomas F. Coleman and Yuying Li. *"A reective newton method for minimizing a quadratic function subject to bounds on some of the variables".* SIAM J. on Optimization, 6(4):1040-1058, 1996.

[Collobert 2001]R. Collobert and S. Bengio. *"SVMTorch: Support vector machines for large-scale regression problems".* Journal of Machine Learning Research, 1:143–160, 2001.

[Collobert 2004] R. Collobert, Y. Bengio, S. Bengio. *"A Parallel Mixture of SVMs for Very Large Scale Problems".* Neural Information Processing Systems, Vol. 17, MIT Press, 2004.

[Courant 1953] R. Courant and D. Hilbert. *"Methods of Mathematical Physics. Interscience Publications".* John Wiley and Sons, New York, U.S.A., 1953.

[Cover 1967] T. Cover and P Hart. *"Nearest neighbor pattern classification".* IEEE Transactions on Information Theory 13 (1967), no. 1, 21–27.

[Crisp 2000] David J. Crisp and Christopher J. C. Burges. *"A Geometric Interpretation of SVM Classifers".* Advances in Neural Information. Cambridge, USA: MIT, (12):244-250, 2000.

[Culler et al. 1999] Culler D. and Pal J.. *"Parallel Computer Architecture a harware and software approach".* Editorial Meghan Keeffe, San Francisco, USA, 1999.

[Cyr 2004] Chistopher M. Cyr y Benjamin B. Kimia. *"A similarity-based aspect-graph approach to 3d object recognition".* International Journal of Computer Vision 57(1), 5–22 (abril 2004).

[Darema et al. 2001] Darema F. *"The SPMD Model: Past, Present and Future".* In proc. of the 8th Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (European PVM/MPI), London, United UK, pp. 1, 2001.

[Darema et al.] Darema F., George D., Norton V. and Pfister G.. *"A single-programmultiple-datacomputational model for EPEX/FORTRAN".* Journal of parallel computing, Vol 7, Issue 1, pp. 11-24, 1988.

[Davis 1975] L. S. Davis. *"A survey of edge detection techniques. Computer Graphics and Image Processing".* Vol 4, No. 3, pp. 248–170 (1975).

[De Giusti 2008] L.C. de Giusti. *"Mapping en arquitecturas heterogéneas".* Tésis doctoral. 2008

[De la luna 2001] E. de la Luna, *"Multichannel electron cyclotron emission radiometry in TJ-II stellarator".* Rev. Sci. Instrum. 72, 379 (2001).

[Devetyarov et al.] Dmitry Devetyarov, *"Confidence and Venn machines and their applications to proteomics".* (2010).

[Díaz 2006.] J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro. *"Un Algoritmo Autoplanificador Cuadrático para Clusters Heterogéneos de Computadores".* XVII Jornadas de paralelismo-Albacete Septiembre 2006.

[Dietterich 1995] T.G. Dietterich and G. Bakiri, *"Solving multiclass learning problems via error-correcting output codes".* ArXiv preprint cs/9501101 (1995), 263–286.

[Dong 2005] G. Dong, M. Xie. *"Color clustering and learning for image segmentation based on neural networks".* Neural Networks, IEEE Transactions on 16(4), 925–936 (2005).

[Dormido-Canto 2004] Sebastián Dormido-Canto y otros.. *"TJ-II Wave Forms Analysis with Wavelets and Support Vector Machines".* Review of Scientific Instruments, vol. 75, no. 10, pp. 4254 - 4257, 2004.

[Dormido-Canto 2005] Sebastián Dormido-Canto, Jesús Vega, Joaquín Sánchez y GonzaloFarias. *"Information Retrieval and Classification with Wavelets and Support Vector Machines".* Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation; Conference on Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach, vol. Part II, Berlin Heidelberg, 2005, pp. 548 - 557.

[Dormido-Canto 2006] S. Dormido-Canto, G. Farias, J. Vega, R. Dormido, J. Sanchez, M. Santos, J. A. Martín, G. Pajares. *"Search and retrieval of plasma waveforms: structural pattern recognition approach".* Rev. Sci. Ins. 77 (2006) 10F514.

[Dormido-Canto 2012] S. Dormido-Cantoa, G. Farias, J. Vega, I. Pastor. *"Image processing methods for noise reduction in the TJ-II Thomson Scattering diagnostic. Fusión Engineering and Design".* 87(12):2170–2173 • December 2012.

[Downey 1998] Downey, A. B. (1998). *"A parallel workload model and its implications for processor allocation".* Cluster Computing, 1(1), 133-145.

[Duda 1972] Duda, R. O., & Hart, P. E. (1972). *"Use of the Hough transformation to detect lines and curves in pictures".* Communications of the ACM, 15(1), 11-15.

[Estrada 2001] T. Estrada. *"Density profile measurements by AM reflectometry in TJ-II".* Plasma Phys. Control. Fusión 43, 1535 (2001).

[Fan 2001] Fan, J., Yau, D. K., Elmagarmid, A. K., & Aref, W. G. (2001*). "Automatic image segmentation by integrating color-edge extraction and seeded region growing".* IEEE transactions on image processing, 10(10), 1454-1466.

[Fauqueur 2004] J. Fauqueur y N. Boujemaa. *"Región based image retreival : fast coarse segmentation and fine color description".* Journal of visual languajes & computing 15(1), 69-95 (Febrero 2004).

[Feitelson et al.] Dror G. Feitelson. *"Workload Modeling for Computer Systems Performance Evaluation".* The Rachel and Selim Benin School of Computer Science and Engineering. The Hebrew University of Jerusalem, Israel.

[Fernández 2000] A. Fernández. *"Quasioptical transmission lines for ECRH at TJ-II stellarator".* Int. J. Infrared Milli. 21 1945 (2000).

[Fernández 1973] Fernández E., Busseli B.. *"Bounds on the Number of Processors and Time for Multiprocessors Optimal Schedules".* IEEE Transactions on Computers, 22(8):745-751.1973.

[Feitelson 2004] Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: *"Parallel job scheduling - a status report".* In Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., eds.: JSSPP. Volume 3277 of Lecture Notes in Computer Science., Springer (2004) 1–16.

[Fix 1951] Evelyn Fix and Joseph L Hodges Jr.. *"Discriminatory analysis-nonparametric discrimination: consistency properties".* Tech. Report May, DTIC Document, 1951.

[Fontdecaba 2004] J.M. Fontdecaba et al., *"Energy-resolved neutral particle fluxes in TJ-II ECRH plasmas".* Fusión Eng. Des. 46, 271 (2004).

[Frank 1998] E. Frank and I. Witten. *"Generating accurate rule sets without global optimization in Proceedings".* Fifteenth International Conference on Machine Learning, pages 144–151. Morgan Kaufmann, 1998.

[Frank 2010] A. Frank and A. Asuncion. *"UCI machine learning repository".* 2010. http://archive.ics.uci.edu/ml/.

[Frieb 1998a] T. Frieb, N. Cristianini y C. Campbell, *"The kernel-Adatron algorithm: a fast and simple learning procedure for support vector machines",* in Machine Learning: Proc. of the 15th Int. Conf., Shavlik, J. Ed., San Francisco: Morgan Kauffman Publishers, 1998.

[Friedman 1996] Jerome Friedman. *"Another approach to polychotomous classification".* Tech. report, Technical report, Department of Statistics, Stanford University, 1996.

[Frucci 2008] M. Frucci, G. Sanniti. *"From Segmentation to Binarization of Gray-level Images".* Journal of Pattern Recognition Research 1 (2008) 1-13.

[Fuertes 2001] J. M. Fuertes, M. Lucens, N. Perez de la Blanca y J. Chamoroor-Martínez. *"A scheme of color image retrieval from databases".* Pattern Recognition Letters 22(3-4), 323-337 (Marzo 2001).

[Fynn 1995] Flynn, Michael J.. *"Computer architecture: pipelined and parallel processor design".* Boston, MA: . Jones and Bartlett.ISBN 0-86720-204-1. 1995.

[Gammerman et al., 1998] A. Gammerman, V Vovk, and V Vapnik. *"Learning by transduction".* Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (1998), 148–155.

[Gammerman 2006] Alexander Gammerman and Vladimir Vovk. *"Hedging predictions in machine learning: The second computer journal lecture".* The Computer Journal 50 (2006), no. 2, 151–163.

[Gerasolis 1992] Gerasoulis A. and Yang T. *"A Comparison of Clustering Heuristics for Scheduling DAG on Multiprocessors".* Journal of Parallel and Distributed Computing. Vol 16 nro (4). Pags 276-291. 1992.

[Gillette 1995] Gillette, J. C., Stadtmiller, T. M., & Hardie, R. C. (1995). *"Aliasing reduction in staring infrared imagers utilizing subpixel techniques".* Optical Engineering, 34(11), 3130-3137.

[Glasmachers 2006]. Tobias Glasmachers, Christian Igel. *"Maximum-Gain Working Set Selection for SVMs".* Institut fur Neuroinformatik ¨ Ruhr-Universitat Bochum ¨ 44780 Bochum, Germany. 2006

[Gonzalez 1978] R. C. Gonzalez and M. G. Thomason. *"Syntactic Pattern Recognition".* Addison-Wesley, Reading, Mass., 1978.

[Gonzalez et al.] R. C. Gonzalez and R.E. Woods. *"Digital Image Processing".* 2nd Edition, Prentice Hall, New Jersey, 2002.

[Graf 2005], 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).

[Guasp 1999] J. Guasp. *"Thermal Load Calculations at TJ-II Vacuum Vessel Under Neutral Beam Injection".* Fusión Sci. and Technol. 35 32 (1999).

[Guermeur 2000] Guermeur, Y., Elisseeff, A., Paugam-Moisy, H., 2000. *"A New Muti-class lSVM Based on a Uniform Convergence Result".* Proc. IJCNN, Como, Italy, 4:183-188. [doi:10.1109/ IJCNN.2000.860770]

[Gvnir 1998] H. A. Gvenir, G. Demirz, H. A. G. (corresponding, G. Demiroz, and N. Ilter. *"Learning differential diagnosis of erythemato-squamous diseases using voting feature intervals"*. Artificial Intelligence in Medicine, vol. 13, pp. 147–165, 1998.

[Haar 1910] Alfred Haar. *"Zur Theorie der orthogonalen Funktionensysteme".* Mathematische Annalen 69 (1910), pp. 331-371.

[Hagedoorn 2000] M. Hagedoorn, *"Pattern Matching Using Similarity Measures".* PhD thesis, Universiteit Utrecht, 2000.

[Hamming 1925] Richard W Hamming. *"Error detecting and error correcting codes".* Bell System Technical Journal 29 (1950), no. 2, 147–160.

[Harris et al. 1985] J.H. Harris. *"A flexible heliac configuration".* Nuclear Fusión 25, 623 (1985).

[Haykin 1999] S. Haykin. *"Neural networks: a comprehensive foundation".* Prentice Hall, 2nd edition, 1999.

[Hender et al. 1988] T.C. Hender. *"Studies of a flexible heliac configuration".* Fusión Technol. 13, 521 (1988).

[Herranz 2003] J. Herranz. *"The spectrometer of the high-resolution multiposition Thomson scattering diagnostic for TJ-II".* Fusión Eng. Des. 65, 525 (2003).

[Hidalgo 2004] A. Hidalgo. *"Multipulse supersonic helium beam diagnostic in the TJ-II stellarator".* Rev. Sci. Instrum. 75, 3478 (2004).

[Hockney 1988] Hockney R. W. and Jesshope C. R. *"Parallel Computers: Architecture, Programming and Algorithms"*. Adam Hilger Ltd., Bristoll, UK, 2nd edition, 1988.ISBN 0-85274-811-6.

[Hofmann 2008] Thomas Hofmann, Bernhard Scholkopf, and Alexander J. Smola. *"Kernel methods in machine learning".* Annals of Statistics, 36(3):1171-1220, 2008.

[Holland 1975] J. H. Holland. *"Adaptation in Natural and Artificial Systems".* The University of Michigan Press, 1975.

[Hosny 2010] K.M. Hosny, *"Fast and accurate method for radial's moments computation, Pattern Recogn".* Lett. 31 (2) (2010) 143–150.

[Hovmöller 1992] Sven Hovmöller. *"CRISP: crystallographic image processing on a personal computer".* Ultramicroscopy.Volume 41, Issues 1–3, April–May 1992, Pages 121-135.

[Hwang 1993] Kai Hwang. *"Advanced computer architecture: Parallelism, scalability, programmability".* McGraw-Hill, 1993.

[Inda 1991] Marciá A. Inda, Rob H. Bisseling.*"A simple and efficient parallel FFT algorithm using the BSP model".* FOM Institute for Atomic and Molecular Physics (AMOLF), Kruislaan 407, 1098. SJ Amsterdam, The Netherlands and Instituto de Matemática, Universidade Federal do Rio Grande do Sul, Av. Bento Gon¸calves 9500, 91509-900 Porto Alegre, RS, Brazil.1991.

[Joachims 1999] Joachims T. *"Making large-scale support vector machine learning practical".* In A.S.B. Scholkopf, C. Burges, editor, Advances in Kernel Methods: support vector machine . pp.169-184, 1999.

[Keilhacker 2001] M. Keilhacker. *"The scientific success of JET".* Nuclear Fusión 41, 1925 (2001).

[Knerr 1990] Stefan Knerr, Lon Personnaz, and Grard Dreyfus. *"Single-layer learning revisited: a stepwise procedure for building and training a neural network",* Neurocomputing, Springer, 1990, pp. 41–50.

[Kreel 1999] Ulrich H-G Kreel. *"Pairwise classification and support vector machines, Advances in kernel methods".* 1999, pp. 255–268.

[Kumazawa 1999] R. Kumazawa. *"Liquid stub tuner for ion cyclotron heating".* Rev. Sci. Instrum. 70, 2665 (1999).

[Lambrou 2012a] Lambrou, A., Papadopoulos, H., Nouretdinov, I., & Gammerman, A. (2012, September). *"Reliable probability estimates based on support vector machines for large multiclass datasets".* In IFIP International Conference on Artificial Intelligence Applications and Innovations (pp. 182-191). Springer Berlin Heidelberg.

[Lambrou 2012b] Lambrou, A., Papadopoulos, H., & Gammerman, A. (2012, November). *"Calibrated probabilistic predictions for biomedical applications".* InBioinformatics & Bioengineering (BIBE), 2012 IEEE 12th International Conference on (pp. 211-216). IEEE.

[Lambrou 2014] Antonis Lambrou, Ilia Nouretdinov, Harris Papadopoulos. *"Inductive Venn Prediction".* Annals of Mathematics and Artificial Intelligence, volume 74 pages 181-201. Springer, 2014.

[Leinberger 2000] W.Leinberger, G.Karypis, and V.Kumar, *"Load Balancing Across Near-Homogeneous Multi-Resource Servers",* 9th Heterogeneous Computing Workshop, May 01, 2000, Cancun, Mexico, pp. 60-71.

[Lew 2002] Y.P. Lew, A.R. Ramli, S.Y. Koay, R. Ali Y V. Prakash. "*A hand segmentation scheme using clustering technique in homogeneous background".* En "Research and Development, 2002. SCOReD 2002. Student Conference on", páginas 305–308 (2002).

[Li 2004] G.LI, C. An, J. Pang, Color image adaptive clustering segmentation. *"Image and Graphics, 2004. Proceedings".* Third International Conference, páginas 104–107 (2004).

[Lichman 2013] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[Lindeberg 1999] T. Lindeberg. *"Handbook on Computer Vision and Applications".* Tomo 2, capíıtulo Principles for Automatic Scale Selection, páginas 239–274. Academic Press (agosto 1999).

[Lo 1999] Y.S. Lo Y S.C. Pei. *"Color image segmentation using local histogram and self-organizatin of kohonen feature map".* "International Conference on Image Processing", páginas 232–239 (1999).

[Lo 2001] Chi-Chun Lo, Shuenn-Jyi Wang. *"Video segmentation using a histogram-based fuzzy c-means clustering algorithm".* "Fuzzy Systems", 2001. The 10th IEEE International Conference on", tomo 2, páginas 920–923 vol.3 (2001).

[Lopez-Bruna 2004] D. Lopez-Bruna. *"Effects of ohmic current in the TJ-II stellarator".* Nuclear Fusión 44, 645 (2004).

[Lublin 2001] "The workload onparallel supercomputers: modeling the characteristics of rigid jobs". Uri Lublin and Dror G. Feitelson School of Computer Science and Engineering, The Hebrew University, 91904 Jerusalem, Israel Received 5 October 2001.

[Lucchese 2001], L. Luchesse y S. K. Mitra. *"Colour image segmentation: a state-ofthe-art survey".* En "Proceedings of the Indian National Science Academy", tomo 67(2), páginas 207–21 (2001).

[Lyon et al. 1990] J.F. Lyon. *"Stellarators".* Nuclear Fusión 30, 1695 (1990).

[Mahalanobis 1963] P.C. Mahalanobis. "On the generalised distance in statistics". Proceedings of the National Institute of Science of India 12 (1936) 49-55.

[Makili 2014] L.E. Makili. Tesis doctoral. *"Sistemas de clasificación automáticos con confianza y credibilidad en fusión termonuclear".* Uned 2014.

[Mandelbrot 1982]. Benoit Mandelbrot. *"The Fractal Geometry of Nature".* 1982.

[Markhoul 1999] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. *"Performance measures for information extraction".* In Broadcast News Workshop 1999 Proceedings, page 249, (1999).

[Martínez 2015] F. J. Martínez, S. Dormido, J. A. Vega. EFDA-JET Conferences Pinboardhttp://users.jet.efda.org/webapps/pinboard/EFDA JET/confer ence /index.html#Document12692. 05/01/2015

[Mathworks.a] C. B. Moler. *"Numerical Computing with MATLAB".* Revised Reprint. SIAM, 2004, 2008.

[Mathworks.c] Web de The MathWorks, Disponible en: http://www. mat hwor ks . com/product s/matlab/.

[Mathworks.c] Web de The MathWorks, Disponible en: http://www.mathworks.com/products/pfo/ .

[Matos 2005] J.C. Matos, B. González, J. Escuadra y J. Toribio. *"Un procedimiento automático de ajuste de una elipse para modelizar el frente de una fisura superficial".* Anales de mecánica de la fractura. Vol.22 (2005).

[Matthews 1975] B.W. Matthews. *"Comparison of the predicted and observed secondary structure of T4 phage lysozyme".* Biochim. Biophys. Acta. 405: 442–451. (1975).

[McCarthy 1999] K.J. McCarthy. *"A toroidal focusing mirror based vacuum ultraviolet diagnostic for TJ-II".* Rev. Sci. Instrum. 70, 312 (1999).

[McCarthy 2001] K.J. McCarthy. *"An Impurity Emission Survey in the near UV and Visible Spectral Ranges of Electron Cyclotron Heated (ECH) Plasmas in the TJ-II Stellarator".* CIEMAT Report 965 (2001).

[McCarthy 2003] K.J. McCarthy. *"Results of an experiment relating apparent doppler ion temperatures with non-thermal velocities in hot-fusion plasmas".* Europhys. Lett. 63, 49 (2003).

[McCool et al. 2008] Mccool M. D. *"Scalable programming models for massively multicore processors".* Proceedings of the IEEE , Vol. 96, Issue 1, pp. 816-831, 2008.

[Medina 1999] F. Medina. *"X-ray diagnostic systems for the TJ-II flexible heliac".* Rev. Sci. Instrum. 70, 642 (1999).

[Meijster 1995] Meijster, A., y Roerdink, J. B. T. M. (1995). *"A proposal for the implementation of a parallel watershed algorithm".* En In proceedings of caip1995 (pp. 790-795).

[Mezhoud 2004] N. Mezhoud, I. Merzoug, R. Boumaza y M.C. Batouche. *"Color image segmentation using a new fuzzy clustering method".* "Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference ", tomo 3, páginas 1209–1214 Vol. 3 (2004).

[Michalski 1983] R. S. Michalski. *"A theory and methodology of inductive learning. Artificial Intelligence".* 20, 1983.

[Minton 1992] S. Minton, J. Carbonell, C. Knoblock, O. Etzioni, and Y. Gil. *"ExplanationBased Learning: A problem-solving perspective".* Artificial Intelligence, 40,1989.

[Miyamoto 1980] K. Miyamoto, *"Plasma Physics for Nuclear Fusion".* MIT Press, Cambridge (1980).

[Moghaddamzadeh 1997] A. Moghaddamzadeh y N. Bourbakis. *"fuzzy región growing approach for segmentation of color images".* Pattern Recognition 30(6), 867– 881 (junio 1997).

[Moura 1998] L. Moura, E. Silva and R. Buyya. *"Parallel programming models and paradigms".* 1998.

[Muresano et al. 2009] Muresano R, Rexachs D. y Luque E.. *"Administración Eficiente de Recursos de Cómputo Paralelo en un Entorno Heterogéneo".* XX Jornadas de Paralelismo, A coruña, España, pp.427-432, 2009.

[Nouretdinov 2001] I. Nouretdinov, T. Melluish, and V. Vovk. *"Ridge Regression Confidence Machine".* In: In Proceedings of the Eighteenth International Conference on Machine Learning. Morgan Kau_man, pp. 385-392. (2001).

[Ochando 1999] M.A. Ochando. *"Bolometry systems for the TJ-II flexible heliac".* Rev. Sci. Instrum. 70, 384 (1999).

[Oliveira 2003] Paulo Oliveira and Hans. *"SPMD Image Processing on Beowulf Clusters: Directives and Libraries".* Vision Laboratory, University of Algarve Campus de Gambelas – FCT, Faro, Portugal. 2003

[Osuna 1997] Edgar E. Osuna, Robert Freund, and Federico Girosi. *"Support vector machines: Training and applications".* Technical report, 1997.

[Otsu 1975] N. Otsu. *"A threshold selection method from gray-level"* histograms. *Automatica*, *11*(285-296), 23-27. (1975).

[Pajares 2003] Gonzalo Pajares Martinsanz, Jesús Manuel de la Cruz García, José M. Molina Pascual, Juan Cuadrado, Alejandro Lopez.*"Imágenes digitales. Procesamiento práctico con Java"*. RA-MA S.A. Editorial y Publicaciones, 2003.

[Pajares 2007] Gonzalo Pajares Martinsanz, Jesús Manuel de la Cruz García.*"Visión por computador. Imágenes Digitales y Aplicaciones"*. 2a Edición. RA-MA S.A. Editorial y Publicaciones, 2007.

[Pan 2010] Pan, X., & Lyu, S. (2010, March). *"Detecting image region duplication using SIFT features".* In Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on (pp. 1706-1709). IEEE.

[Pao 1989] Y. H. Pao. *"Adaptive Pattern Recognition and Neural Networks".* Addison-Wesley, Reading, Mass., 1989.

[Patel 2013] Shalinee Patel, Pinal Trivedi, and Vrundali Gandhi. *"2D Basic Shape Detection Using región Properties".* International Journal of Engineering Research & Technology, vol. 2, no. 5, pp. 1147-1153, May 2013.

[Pedrosa 1999] M.A. Pedrosa. "*Fast movable remotely controlled Langmuir probe system".* Rev. Sci. Instrum. 70, 415 Part 2 (1999).

[Peña 2001.]. Peña Sánchez de Rivera, Daniel. *"Deducción de distribuciones: el método de Monte Carlo".* Fundamentos de Estadística. Madrid: Alianza Editorial. ISBN 84-206-8696-4. 2001

[Per Brinch 1993] P. B. Hansen. *"Model programs for computational science: A programming methodology for multicomputers".* Concurrency - Practice and Experience, 5(5):407-423, 1993.

[Petrou 2004] M. Petrou and P. Bosdogianni. *"Image Processing the Fundamentals".* Wiley, UK, 2004.

[Pichel 2006] "*Image Segmentation based on merging of sub-optimal segmentations".* Juan C. Pichel, David E. Singh and Francisco F. Rivera, Pattern Recognition Letters, 27, 0, 2006, 1105-1117, journal.

[Platt 1998] J. Platt. *"Fast Training of support vector machine using sequential minimal optimization".* In A.S.B. Scholkopf, C. Burges, editor, Advances in Kernel Methods:support vector machine . MIT Press, Cambridge, MA 1998.

[Platt 1999] J. Platt, N Cristianini, and J Shawe-Taylor. *"Large margin dags for multiclass classification".* Advances in Neural Information Processing Systems 12, 1999, pp. 547–553.

[Po-Jen 2000] Po-Jen C., Chih-Ming W. *"AN Eficient Recognition-Complet Processor Allocation Strategy for k-ary n-cube Multiprocessors"*. IEEE Transactions on Parallel and Distributed Systems, 11(5):485-490, 2000.

[Portegys 1995] T. E. Portegys. *"A search technique for pattern recognition using relative distances".* IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(9):910-913, September 1995.

[Priese 2005] L. Priese, P. Sturm y H. Wang. *"Hierarchical Cell Structures for Segmentation of Voxel Images".* Tomo 3540, páginas 6–16. (junio 2005).

[Quinlan 1986] J. Quinlan. *"Induction of decision trees".* Machine Learning, 1(1):81–106,1986.

[Quinn 1994] Quinn M. *"Parallel Computing: Theory and Practice".* McGraw-Hill, 1994.

[Ramírez 2014] J.M. Ramírez. *"Máquinas de vectores soporte en entornos de supercomputación: aplicación a fusión nuclear".* Tésis doctoral 2014. UNED.

[Rajkumar 1999] Rajkumar Buyya. *"High Performance clúster Computing: Architectures and Systems"*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[Ramanthan 2005] Ramanathan R.. *"Intel Multi-Core Processors: Leading the Next Digital Revolution"*. Intel Magazine, Vol. 1 , Issue 1, pp. 1-8, 2005.

[Ribadas 2002] *"Reconocimiento de patrones en bosques compartidos".* Francisco Jose Ribadas Pena. Tesis doctoral. A coruña 2002.

[Rusell 2009] Stuart Russell & Peter Norvig. *"Artificial Intelligence: A Modern Approach".* Prentice-all, 3rd edition, 2009 ISBN 0136042597 http://aima.cs.berkeley.edu/

[Sahoo 1988] P. K. Sahoo, S. Soltani y A. K. C. Wong . *"A survey of thresholding techniques".* Computer Vision, Graphics, and Image Processing 41(2), 233–260 (febrero 1988).

[Sánchez 1993] J. Sánchez. *"Diagnostic systems for the TJ-II flexible heliac",* J. Plasma Fusión Res. Series 1, 338 (1993).

[Sanfeliu 2002] A. Sanfeliu, R. Alquezar, J. Andrade, J. Climent, F. Serra y J. Verges. *"Graph-based representations and techniques for image processing and image analysis"*. Pattern Recognition 35(3), 639–650 (marzo 2002).

[Saunders 1999] C. Saunders, A. Gammerman, and V. Vovk. *"Transduction with confidence and credibility"*. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99) 2 (1999), 722–726.

[Saunders 2000] C. Saunders, A. Gammerman, and V. Vovk. *"Computationally efficient transductive machines".* In International Conference on Algorithmic Learning Theory (pp. 325-337). Springer Berlin Heidelberg. (2000, December).

[Scholkopf 2001] Bernhard Scholkopf and Alexander J. Smola. *"Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond".* MIT Press, Cambridge, MA, USA, 2001.

[Scholkopf 2002] B. Scholkopf and A.J. Smola. *"Leaning with kernels: Support vector machines, regularization, optimization, and beyond".* MIT Press, Cambridge,MA, USA, 2002.

[Seller 1999] Seller Roosta.*"Parallel Procesinf and Parallel Algorithms. Theory and computation".* Spinger. 1999.

[Shameem 2006] Shameem A. and Jason R. *"Multicore Programming"* Intel Press, USA, 2006.

[Sheffield 1994] J. Sheffield, *"The physics of magnetic fusion-reactors",* Rev. Mod. Phys. 66, 1015 (1994).

[Spitzer et al. 1958] L. Spitzer, *"The stellarator concept".* Physics of Fluids 1, 253 (1958).

[Steinwart 2008] Ingo Steinwart, Andreas Christmann. *"Support Vector Machines".* Springer Science & Business Media, 15 sept. 2008.

[Tabarés 2003] F.L. Tabarés et al., *"Impact of wall conditioning and gas fuelling on the enhanced confinement modes in TJ-II".* J. Nucl. Mater. 313-316, 839 (2003).

[Tamm 1961] I.Y. Tamm y A. Sakharov, *"Theory of a magnetic thermonuclear reactor".* Pergamon, Oxford (1961).

[Uppaluri 2003] Uppaluri S., Izadi B. and Radhakrishnan D. *"Low-Power Dynamic Scheduling in Heterogeneous Systems".* In Proceedings of the International Conference onEmbedded Systems and Applications, (ESA '03) ISBN: 1-932415-05-X Pags 261-267. 2003.

[Utrera 2012] G. Utrera, S. Tabik, J. Corbalan, and J. Labarta. *"A job scheduling approach for multi-core clusters based on virtual malleability".* Technical University of Catalonia (UPC) 08034 Barcelona, Spain. University of Malaga, 29071 Malaga, Spain. Barcelona Supercomputing Center (BSC) 08034 Barcelona, Spain. 2012.

[Vapnik 1995] V. Vapnik. *"The Nature of Statistical Learning Theory".* Springer, N.Y., 1995. ISBN 0-387-94559-8.

[Vapnik 1998] Vapnik V. *"Statistical Learning Theory".* Springer, N.Y., 1998.

[Vapnik 2000] Vapnik V.N. *"The Nature of Stastical Learning Theory".* 2nd edn, Springer, New York, 2000.

[Vega 1999a] J. Vega, E. Sánchez, C. Crémy, A. Portas, C. M. Dulya, J. Nilsson. *"TJ-II data retrieving by means of a client/server model"*. Review of Scientific Instruments, Vol. 70, No. 1, Pag. 498, January 1999.

[Vega 1999b] J. Vega, C. Creémy, E. Sánchez, A. Portas. *"The TJ-II data acquisition system: an overview".* Fusión Engineering and Design, Volume 43, Issues 3-4, Pages 309-319, January 1999.

[Vega 2008a] J. Vega, A. Pereira, S. Dormido-Canto, G. Farias, R. Dormido, J. Sánchez, N. Duro, M. Santos, E. Sánchez, G. Pajares. *"Data mining technique for fast retrieval of similar waveforms in Fusión massive databases. Fusión".* Engineering and Design. Vol 83, Issue 1, pp 132-139. January 2008. ISSN: 0920-3796.

[Vega 2008b] J. Vega, A. Murari, A. Pereira, A. Portas, P. Castro and JET-EFDA Contributors. *"Intelligent technique to search for patterns within images in massive databases".* Proceedings of the HTPD High Temperature Plasma Diagnostic 2008, Albuquerque, New Mexico. 11-15 May 2008. Review of Scientific Instruments. Vol. 79. Issue 10. 10F327 (2008). ISSN: 0034-6748, E-ISSN: 1089-7623.

[Vega 2015] J. Vega, S. Dormido-Canto, F. Martínez, I. Pastor, M. C. Rodríguez. *"Computationally efficient five-class image classifier based on Venn predictors"*. The third international Symposium on statistical Learning ans data sciences.(SLDS 2015) .April 20-23, 2015.

[Venn 1866] John Venn. *"The logic of chance: An essay on the foundations and province of thetheory of probability, with especial reference to its application to moral and socialscience",* Macmillan, 1866.

[Vijayakumar 1999] Sethu Vijayakumar, Si Wu. *"Sequential Support Vector Classifiers and Regression".* RIKEN Brain Science Institute, The Institute for Physical and Chemical Research, Hirosawa 2-1, Wako-shi, Saitama, Japan 351-0198. 1999

[Volodya 2005] Volodya Vovk, Alexander Gammerman, and G. Shafer. *"Algorithmic Learning in a Random World".* New York, Springer, 2005.

[Vovk 1999a] V. Vovk. *"Derandomizing stochastic prediction strategies".* Machine Learning 35 (1999), no. 3, 247–282.

[Vovk 1999b] V. Vovk and A. Gammerman. *"Algorithmic randomness theory and its applications in computer learning"*. Technical Report CLRC-TR-00-02, Royal Holloway, University of London, 1999.

[Vovk 2003] Vladimir Vovk, Glenn Shafer, and Ilia Nouretdino. "*Self-calibrating probability forecasting".* Advances in Neural Information Processing Systems, 2003, p. None.

[Vovk 2005] V Vovk, A Gammerman, and G Shafer. *"Algorithmic learning in a random world".* Springer Science & Business Media, 2005.

[Vovk 2012] Vladimir Vovk. *"Venn predictors and isotonic regression".* arXiv preprint arXiv:1211.0025 (2012), 1–4.

[Wakatani 1998] M. Wakatani. *"Stellarator and Heliotron Devices".* Oxford University Press, Oxford (1998).

[Wanner 2003] M. Wanner. *"Status of Wendelstein 7-X construction".* Nucl. Fusión 43, 416 (2003).

[Watkins 1989] C. Watkins. *"Learning from Delayed Rewards".* PhD thesis, King's College,Cambridge, UK, 1989.

[Watkins 1992] C. Watkins and P. Dayan. *"Q-Learning".* Machine Learning, 8:279–292,1992.

[Wilson 1993] Wilson G. V. "A Glossary of Parallel Computing Terminology". IEEE Parallel and Distributed Technology: Systems and Applications, Vol. 1, N° 1, pp. 52-67, 1993.

[Wojcik 1985] Z. M. Wojcik. "A natural approach in image processing and pattern recognition: Rotating neighbourhood technique". self-adapting threshold, segmentation and shape recognition. Pattern Recognition 18(5),299–326 (1985).

[www.euro-fusion.org] https://www.euro-fusion.org/jet/.

[Yagoubi 2010] Yagoubi, B., & Meddeber, M. (2010). Distributed load balancing model for grid computing. *ARIMA journal*, *12*, 43-60.

[Yom-Tov 2004] Elad Yom-Tov. "A parallel training algorithm for large scale support vector machines". IBM Haifa Research Labs Haifa 31905 Israel. November 28, 2004.

[Yoshikawa 1983] S. Yoshikawa, "Design of a helical-axis stellarator". Nucl. Fusión 23, 667 (1983).

[Yu-wong 1996] Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 7(5):506-521, May 1996.

[Zbigniew 1985] Zbigniew M. Wojcik. "A natural approach in image processing and pattern recognition: Rotating neighbourhood technique". self-adapting threshold, segmentation and shape recognition. Pattern Recognition 18(5), 299–326 (1985).

[Zhou 2004] Zhou, H., Yang, X., Tang, Y., y Xiao, N. (2004). "Further optimized parallel algorithm of watershed segmentation based on boundary components graph".

[Zomaya 1996] Zomaya A. Y. "Parallel and Distributed computing: The Scene, The Props, The Players". Parallel and Distributed Computing Handbook Chapter 1, pp. 5-23. ISBN 0-07-073020-2. 1996

Anexos.

A. Reducción de dimensionalidad.

La transformada wavelet es una herramienta muy utilizada en el preprocesamiento de imágenes como puede ser reconocimiento de bordes, clasificación de texturas y la compresión de imágenes. Las wavelets son familias de funciones que se emplean como funciones de análisis. La primera mención de una wavelet fue hecha por Alfred Haar [Haar et al.].

Pero hasta los años 80 no arranca la evolución importante de las wavelets donde Ingrid Deaubechies [Daubechies et al.] publica un conjunto de wavelets orto-normales que se convierten en el pilar de las aplicaciones wavelets. Existen dos tipos de transformadas wavelets, la contínua (CWT) y la discreta (DWT). La DWT se utiliza para diferentes objetivos, entre ellos está el tratamiento de imágenes donde se suele utilizar para la reducción del ruido, la compresión (de vital importancia tanto en la transmisión de grandes cantidades de datos como en su almacenamiento) o la detección de determinados objetos en ciertos tipos de imágenes. La DWT permite a su vez realizar una descomposición de la imagen en detalles y a diferentes niveles de resolución (distintas escalas y orientaciones diferentes.

Para la descomposición wavelet aplicada a imágenes se utiliza la DWT 2D que es una extensión de la DWT 1D. La DWT 2D aplicada a imágenes trata cada fila y cada columna por separado como si fueran una señal-unidimensional.

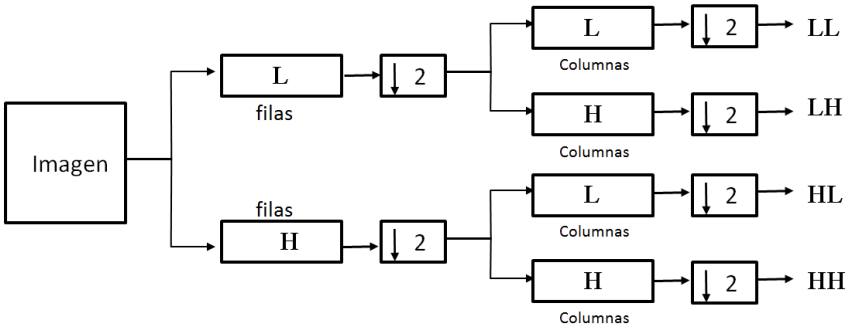


Figura A.1. Descomposición de un nivel de la DWT 2D.

La figura A.1 muestra el proceso. La primera secuencia es separar las filas y aplicarlas la TW. Se aplica primero un filtro H (pasa-altos) y un filtro L (pasa-bajos). Posteriormente se aplica el mismo proceso a cada columna de la imagen. De este proceso se obtiene cuatro sub-matrices. Una sub-matriz que contiene los detalles de aproximación LL, después está la sub-matriz HL que contiene los detalles horizontales, posteriormente se encuentra la sub-matriz LH que contiene que contiene los detalles verticales y finalmente la sub-matriz HH que contiene los detalles diagonales de la imagen. El símbolo ↓2 se refiere al proceso de decimación. Básicamente lo que hace la decimación es tomar cada *D* muestras equidistantes de la señal original, y descarta las demás. Dejando de esta forma una señal equivalente pero con diferente tasa de muestreo. Esto hace que las sub-matrices resultantes tengan una dimensionalidad más baja.

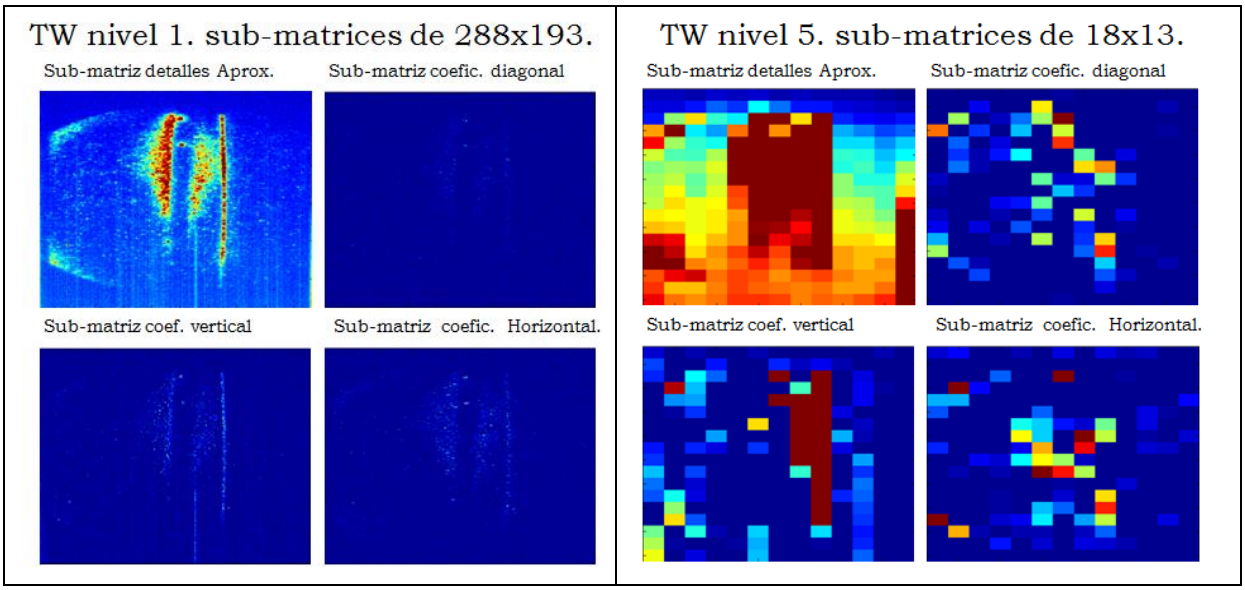


Figura A.2. Descomposición del nivel 1 y 5 de la DWT 2D.

Para la descomposición wavelet aplicada a imágenes se utiliza la DWT 2D que es una extensión de la DWT 1D. La DWT 2D aplicada a imágenes trata cada fila y cada columna por separado como si fueran una señal-unidimensional. La figura

La figura A.2 muestra la aplicación de la transformada wavelet de ordenes 1 y 5 a la imagen de la señal CUTOFF de la figura 6.9. El proceso se inicia con una imagen CUTOFF de dimensiones 576x386. Después de aplicar el primer nivel de TW se obtienen sub-matrices de dimensiones 288x193. En un nivel 5 de TW se obtendría sub-matrices de dimensiones 18x13. Existe por tanto una reducción considerable de dimensionalidad

B. Resultados en pared metálica.

La tabla que figura a continuación muestra la relación de descargas de pared metálica en las que se ha detectado los patrones BP1, BP2 y BP3. La primera columna indica el número de descarga, la segunda columna la frecuencia donde se ha encontrado el fundamental y las dos columnas restantes indican el comienzo y fin de intervalo temporal donde se han localizado.

Número Descarga	Frec. Ppal (KHz.)	Inic. Intervalo (sg.)	Fin Intervalo (sg.)
81886	6,6111	61,1007	61,3443
82214	3,5556	53,5175	53,7523
82129	4,3704	52,5648	52,6009
82226	6,5556	59,0939	59,2548
81884	6,3333	59,0858	59,4121
82228	2,3333	59,0858	59,4121
82038	5,5107	53,5363	53,7320
82230	2,5556	59,2154	59,5509
82141	5,6389	56,8218	56,8980
82215	5,4195	56,7061	57,0394
82373	2,3595	52,3607	52,8485
82754	6,3370	57,3499	57,9325
82749	2,4679	57,2568	57,6872
	5,7161	56,8193	57,0345
82485	5,6172	58,5696	59,0631
82722	7,9556	48,2281	48,3629
82753	6,0806	57,4960	57,6395
	4,6725	57,7566	58,1066
82414	3,1556	56,7714	56,8171
82483	5,2359	59,0175	59,2914
	4,5000	59,5883	59,6079
	4,4472	59,7472	59,7863
83188	4,6222	57,5521	57,8940
82792	5,5556	50,1977	50,2817
82880	5,1472	57,2637	57,4156
83225	5,1111	53,3718	53,4978
83231	7,7991	56,7004	56,8264
83364	5,5556	58,1535	58,6405
83577	6,3993	63,4549	63,5266

	6,0556	63,5342	63,5521
	5,6024	61,5328	61,6763
	5,6389	61,6848	61,7226
83509	5,0944	55,0520	55,4781
83575	5,2601	60,6905	61,1210
	5,3909	63,1567	63,3720
	3,8889	62,9961	63,0678
	3,7222	63,0718	63,1196
	4,9444	63,1277	63,1316
84343	5,0491	54,2764	54,4938
84365	4,6181	52,7487	53,0748
	5,3510	52,3064	52,7411
84520	5,0172	60,2384	60,6340
84535	6,5556	59,9623	60,0058
	6,4722	59,8427	59,9514
	6,3889	59,7340	59,8427
	5,8254	59,0482	59,4177
84363	5,1980	52,0057	53,1470
84525	4,8547	61,0366	61,4323
84538	3,9287	48,5617	48,7139
85073	5,7371	53,7330	53,7267
	4,3389	53,7634	53,9156
84948	5,3056	57,2607	57,3259
	7,5760	56,7650	57,1259
	8,7250	56,9050	56,9670
	9,8340	56,8230	56,8760
85038	2,3422	54,2982	54,6460
	2,1441	54,6789	54,9632
84820	3,8690	52,5710	52,6771
	4,5463	52,4920	52,5010
	5,4896	52,3150	52,4759
84617	5,3601	50,5978	51,2545
	8,9770	51,6043	51,8087
84646	5,2087	55,3997	55,4040
	5,2087	55,6788	55,9354
	4,9484	55,4767	55,5811
85183	4,9167	54,9473	55,0603
	4,7824	55,1532	55,2228
	5,4942	56,2274	56,3969
	6,1685	56,4876	56,7650

85184	5,0802	58,2463	58,5855
85237	5,1794	53,2764	53,6025
	5,8704	60,3222	60,4200
	3,6806	60,5845	60,7497
	3,1389	60,7871	60,8149
85268	3,6204	56,8168	56,9147
	2,4268	58,0228	58,1858
85403	5,2151	49,7283	50,0544
85235	5,9889	55,6596	55,7248
	5,3889	55,2390	55,3042
	3,2222	56,0216	56,1520
	3,4246	56,2265	56,3244

Tabla B.1. Relación de patrones encontrados de descargas de pared metálica.

C. Código.

```
function  pingpongV2(numWorTot);
% función para analizar el tiempo de transmision enter los diferentes worker de la
% comunidad con diferente longitud de mensaje
warning off;
mpiInit;
MJS = findResource('scheduler','type','jobmanager');
matlabpool close force;
% arranque de la sesion paralela
 matlabpool('open','MJS',numWorTot, 'AttachedFiles',...
                        {fullfile(pwd, 'VPmoduloPPAL.m')});
tic;
% limpieza de ficheros de logs
delete('temp_2*.txt');
global h;
contadorBucle=1;
global contadorBucle
% función para tomar tiempo
    tiempo=tic;
     spmd(numWorTot);
              % envio comienzo a todos los worker
                    broadcast_id = 1;
                     if labindex==broadcast_id
                                  for i=1:512
                                        tabla(i)=8;
                                  end
                                  msg4096=tabla;
                                  msg2048=tabla(1:256);
                                  msg1024=tabla(1:128);
                                  msg512=tabla(1:64);
                                  msg128=tabla(1:16);
                                  tic;
                                  for i=1:10
                                    labSend(msg128,2);
                                    ok=labReceive(2);
                                  end
                                 disp(['tiempo.  para 128 bytes. en 10 iteraciones...:.', num2str(toc(tiempo)/10)]);
                                 for i=1:10
                                          labSend(msg512,2);
                                          ok=labReceive(2);
                                  end
                                 disp(['tiempo.  para 512 bytes. en 10 iteraciones...:.', num2str(toc(tiempo)/10)]);
                                  for i=1:10
                                    labSend(msg1024,2);
                                    ok=labReceive(2);
                                  end
                                 disp(['tiempo.  para 1024 bytes. en 10 iteraciones...:.', num2str(toc(tiempo)/10)]);

                               for i=1:10
                                    labSend(msg2048,2);
                                    ok=labReceive(2);
                                  end
                                 disp(['tiempo.  para 2048 bytes. en 10 iteraciones...:.', num2str(toc(tiempo)/10)]);
                                  for i=1:10
                                    labSend(msg4096,2);
                                    ok=labReceive(2);
                                  end
                                 disp(['tiempo.  para 4096 bytes. en 10 iteraciones....:.', num2str(toc(tiempo)/10)]);
                        else
                             for i=1:10
                                      tabla=labReceive(1);
                                        labSend(tabla,1);
                                  end
                              for i=1:10
                                          tabla=labReceive(1);
                                            labSend(tabla,1);
                              end
                                    for i=1:10
```

```
                                    tabla=labReceive(1);
                                      labSend(tabla,1);
                                end
                               for i=1:10
                                    tabla=labReceive(1);
                                      labSend(tabla,1);
                               end
                                  for i=1:10
                                    tabla=labReceive(1);
                                      labSend(tabla,1);
                                end
                end;
    end;
 ================================================================================================
 function Radix2_PLL(x,numWorker)
% función que paraleliza calculo de la FFT mediante
% algoritmo RADIX
clc;
warning off;
[N,col]=size(x);
x=x';
M=N/2;
mpiInit;
matlabpool('open','MJS',numWorker, 'AttachedFiles',...
            {fullfile(pwd, 'Radix2_PLL.m')});
numlabsW=numlabs-1;
tiempo=tic;
spmd
    if labindex==numlabs        %este worker es el que lleva el control y es necesario que este en el nodo ppal ni que
        for NivelSec=1:col      % sea el último
                numlabsW=numlabs-1;
                NumPw=numlabs-1;
                M=N/2;
                nivel=1
                while  nivel<(log2(N))
                     x= labBroadcast(labindex,x);  %se envía el vector a todos los wrk
                    % ahora se reciben las casillas a modificar de cada worker activo
                     for numlab=1:NumPw
                                    idx1=labReceive(numlab); % se carga posicion de la primera mitad
                                    x(idx1)=labReceive(numlab); %  x(n+index+1)+ x(n+index+M+1);
                                    idx1=labReceive(numlab);     % se carga posicion de la segunda mitad
                                     x(idx1)=labReceive(numlab); % se carga dato
                     end
                    nivel=nivel+1;
                     M=M/2;
                     if M<NumPw    NumPw=NumPw/2;  end;
                       if     nivel==log2(N) %aqui se finaliza porque el último calculo lo hace el wrk master
                            nivel=log2(N);  % se hace para parar el bucle
                        end
                end
        end
    else
            for NivelSec=1:col
                M=N/2;
                 numlabsW=numlabs-1;
                 NumPw=numlabs-1;
                 proceso=true;
                 nivel=0;
                 while proceso
                     nivel = nivel+1;
                     x= labBroadcast(numlabs);
                     if (M>=NumPw)&& (labindex<=NumPw)
                            for indice =0:(N/(2^(nivel-1))):(N-1);
                                for n=0:(NumPw):M-1
                                        idx1=n+indice+1+(labindex-1);
                                        idx2=n+indice+M+1+(labindex-1);
                                       a=x(idx1)+x(idx2);
                                       b=x(idx1)- x(idx2).*exp((-j*(2*pi)/N)*(2^(nivel-1))*(n));
                                end
                            end
```

```
                                labSend(idx1,numlabs);
                                labSend(a,numlabs);
                                labSend(idx2,numlabs);
                                labSend(b,numlabs);
                            end
                            M=M/2;
                            % se ajusta el numero de workers a trabajar en el siguiente nivel
                            if M<NumPw    NumPw=NumPw/2; end
                           if (NumPw==1) proceso=false; end
                        end
                    end
            end
end
disp(['Tiempo total .....', num2str(toc(tiempo))]);


==========================================================================================


function [alphas offset] = seqminopt_version_paralela_codistribute(targetLabels, boxConstraints, kernelFunc,
rbf_sigma,smoOptions,kfun,numlabs)
% función que implementa el algoritmo spread Kernel en version paralela MATLAB
narginchk(4,8);
fecha=datestr(now);disp(fecha);
warning off;
tiempo4=tic;
% se analiza porque en el vector de etiquetas las primeras son las -1
indice=(targetLabels==-1);
 tope=length(find(indice));
 var=matfile('variables.mat');
data1=var.training(1:tope,:);
% ahora se mira a ver hasta donde llega el tamaño de la segunda variable y la cargo
[tope2,~]=size(var.training);
data2=var.training(tope+1:tope2,:);
[groupIndex, groupString] = grp2idx(targetLabels);
targetLabels= 1 - (2* (groupIndex-1));
data=[data1;data2];
disp(['Nombre del proceso en curso....:', mfilename('fullpath')]);
if nargin == 4
    smoOptions = svmsmoset;
end;
switch(smoOptions.Display)
    case 'iter'
        smoOptions.verbose = 2;
    case 'final'
        smoOptions.verbose = 1;
    otherwise
        smoOptions.verbose = 0;
end
% se revisa la consistencia de los datos
try
if isempty(data1) &   isempty(data2)
    error(message('stats:svmtrain:InvalidoTraining'));
end
catch
    disp('**** error^****');
end
nPoints1 = size(data1, 1);
nPoints2 = size(data2, 1);
nPoints = nPoints1+nPoints2;
if ~isequal(size(targetLabels), [nPoints 1])
    error(message('stats:svmtrain:InvalidYSize'));
end
if ~all(boxConstraints >= smoOptions.TolKKT)
    error(message('stats:svmtrain:InvalidInput'));
end
%
%Implementacion de: Sequential Minimal Optimization
% ------------------------------------
% INITIALIZATION
```

```
% ------------------------------------
% se chequea las condiciones de KKT para final convergencia
tolKKT = smoOptions.TolKKT;
npgs=0;
% Tolerancia a chequear por cada vector
svTol = sqrt(eps);
% Numeo de datos en total:
nPoints = length(targetLabels);
% Number of accepted KKT violations
acceptedKKTviolations = smoOptions.KKTViolationLevel * nPoints;
% Creacion de la matril kernel
fullKernel = [];  % initialiciala
 kernelDiag = [];
subKernel = [];
subKernelIndices = [];
leastIndices = [];
if nPoints <= smoOptions.KernelCachéLimit
    try
        fullKernel = kernelFunc(data, data);
    catch ME
        error(message('stats:svmtrain:ErrorfuncionKernel', ME.message));
    end
else
    try
     if strcmp(kfun,'@rbf_kernel')
            Points = size(data, 1);
            tic;
            % se calcula con parfor la digonal de la kernel
            parfor i=1:Points
                    kernelDiag(i,1) = exp(-(1/(2*rbf_sigma^2))*(repmat(sqrt(sum(data(i,:).^2,2).^2),1,...
                        size(data(i,:),1))-2*(data(i,:)*data(i,:)')+repmat(  sqrt(  sum(data(i,:).^2,2)'.^2), size(data(i,:),1),1))  );
           end
    else
         kernelDiag = arrayfun(@(idx) kernelFunc(data(idx, :), data(idx, :)),(1:nPoints)');
   end
    catch ME
        error(message('stats:svmtrain:ErrorfuncionKerner', ME.message));
    end
    clear('data');
    s = max(1, floor(smoOptions.KernelCachéLimit^2 / nPoints));
    leastIndices = zeros(1,s);
    subKernel = zeros(nPoints, s);
    subKernelIndices = zeros(1, nPoints);
end
 if ~isempty(fullKernel)
              kerDiag =diag(fullKernel);
            else
              kerDiag =  kernelDiag;
            end
% se inicializan los alfas,y el gradiente
alphas = zeros(nPoints, 1);
objGrad = ones(nPoints, 1);
offset = NaN;
offset_1=NaN;
Avec = zeros(size(targetLabels));
Avec(targetLabels==-1) = - boxConstraints(targetLabels==-1);
Bvec = zeros(size(targetLabels));
Bvec(targetLabels==1) = boxConstraints(targetLabels==1);
upMask = targetLabels .* alphas < (Bvec - svTol);
downMask = targetLabels .* alphas > (Avec + svTol);
idxHelper = 1:nPoints;
%%%%%%%%%%%%%%%%%%%%%%%%
% ------------------------------------
% BUCLE PRINCIPAL
% esta es la zona de trabajo de los workers
% ------------------------------------
% Este bucle principal calcula la pareje de Alfas y el working set usanfo el Maximun Gain Method.
 itCount=0;
 disp(['tiempo invertido en segundos tiempo_4 .....:', num2str(toc(tiempo4))]);
 tiempo5=tic;
```

```
 fecha=datestr(now);disp(fecha);
 cont_kercol_SIAlmac=0;
cont_kercol_NOAlmac=0;
cont_Kij_SIAlmac=0;
cont_Kij_NOAlmac=0;
cont_kerCol_idx1_NOAlmac=0;
cont_kerCol_idx1_SIAlmac=0;
cont_kerCol_idx2_NOAlmac=0;
cont_kerCol_idx2_SIAlmac=0;
%clear('data*');
spmd
% se reparte la matriz Kernel entre todos los worker
temp1=codistributed(data1');
[Lmin1,Lmax1]=globalIndices(temp1,2);
dataWorker1=getLocalPart(temp1');
[tam1, ~]=size(temp1');
 temp2=codistributed(data2');
[Lmin2,Lmax2]=globalIndices(temp2,2);
dataWorker2=getLocalPart(temp2');
upMaskWr=logical(zeros(size(upMask)));
downMaskWr=logical(zeros(size(downMask)));
upMaskWr(Lmin1:Lmax1,:)= upMask(Lmin1:Lmax1,:);
upMaskWr((Lmin2+tam1):(Lmax2+tam1),:)=upMask((Lmin2+tam1):(Lmax2+tam1),:);
downMaskWr(Lmin1:Lmax1,:)= downMask(Lmin1:Lmax1,:);
downMaskWr((Lmin2+tam1):(Lmax2+tam1),:)=downMask((Lmin2+tam1):(Lmax2+tam1),:);
tiempoTOT=0;
if labindex==numlabs
    % se envia un primer mesaje a todos los worker para que  entren en un blucle de ejecucion mediante sentencia:
    shared_data = labBroadcast(numlabs, 0);% envio un primer mesaje a todos los worker para que
    % entren en un blucle de ejecucion mediante sentencia:
    para=0;
else
    para  =  labBroadcast(numlabs);  % se recibe control de terminacion desde el wrk ppal
end
if labindex==numlabs
   disp('Soy el worker ppal y entro en bucle ppal')
end
while ~para
   % se busca el primer alfa
   [val1 idx1] = max(targetLabels(upMaskWr) .* objGrad(upMaskWr));
   tmp = idxHelper(upMaskWr);
   idx1 = tmp(idx1);
   %try
   if labindex==1
           labSend(idx1,2);  % se envia al wrk ppal
           labSend(val1,2);
   elseif  labindex==numlabs        %este es el worker que esta en el front-end
          lab=labindex-1;
           idx1Ant=labReceive(lab);
           val1Ant=labReceive(lab);
         if val1Ant<=val1      % tomar el valor minimo y ese el que envio al resto de Wr
                 shared_data = labBroadcast(numlabs, idx1Ant);
                 shared_data = labBroadcast(numlabs, val1Ant);
                 idx1=idx1Ant;
                 val1=val1Ant;
          else
                 shared_data = labBroadcast(numlabs, idx1);
                 shared_data = labBroadcast(numlabs, val1);
          end
   else
           lab=labindex-1;
          idx1Ant=labReceive(lab);   %  se toman los valores del Wr anterior
          val1Ant=labReceive(lab);
           lab=labindex+1;
          if val1Ant<=val1
                 labSend(idx1Ant,lab);
                 labSend(val1Ant,lab);
          else
                 labSend(idx1,lab);
                 labSend(val1,lab);
```

```
        end
end
% todos los Wr reciben el idx1 y su valor val1
if labindex~=numlabs
          idx1= labBroadcast(numlabs);
          val1=labBroadcast(numlabs);
end
% se busca el segundo alfa de acuerdo la algoritmo de ganancia maxima.
% hay que encontrar el idx2
 mask = downMaskWr & (targetLabels .* objGrad < val1);
gainNumerator = (targetLabels(mask) .* objGrad(mask) - val1) .*(targetLabels(mask) .* objGrad(mask) - val1);
if isempty(fullKernel)
        if ~isempty(fullKernel)
             kerCol1 = fullKernel(:, idx1);
       else
            if subKernelIndices(idx1) == 0
               % aqui es que no he encontrado la columna almacenada en la caché  da el param:  subKernel_Index
                deleteIndex = leastIndices(end);
               if deleteIndex == 0
                  % la variable subKernel no esta complete aun
                  subKernel_Index = 1 + max(subKernelIndices);
               else
                  subKernel_Index = subKernelIndices(deleteIndex);
                  subKernelIndices(deleteIndex) = 0;
               end
               % introducir  la columna calculada
               subKernelIndices(idx1) = subKernel_Index;
               % se calcual  la nueva columna kernel y se guradara con el indice 'kernelColIdx'.
               % la función kernle_col devuelve la columna de la matriz kernel correspondiente a la fila idx1
               tiempo1=tic;
               kerCol1=kernel_col(idx1,rbf_sigma,Lmin1,Lmax1,Lmin2, Lmax2, dataWorker1, dataWorker2,tam1,kfun);
               tiempoTOT= toc(tiempo1)+tiempoTOT;
               if labindex==numlabs
                     subKernel(:,subKernel_Index)=kerCol1;  % el unico que almacena las columnas calculadas es el worker
             ppal   por eso tiene una caché alta
               end
               leastIndices(2:end) = leastIndices(1:end-1);
               leastIndices(1) = idx1;
               cont_kercol_NOAlmac=cont_kercol_NOAlmac+1;
            else
               if labindex==numlabs
                        cont_kercol_SIAlmac=cont_kercol_SIAlmac+1;
                        kerCol1 = subKernel(:, subKernelIndices(idx1));
                        dato=labBroadcast(numlabs, kerCol1);
                else
                        kerCol1=labBroadcast(numlabs);
                end
             end
        end
       gainDenominator = -4 * kerCol1(mask) + 2 * kerDiag(mask) + 2 *kerDiag(idx1);
else
       gainDenominator = -4 * fullKernel(mask, idx1) + 2 * kerDiag(mask) + 2 *kerDiag(idx1);
end
[~, idx2] = max(gainNumerator ./ gainDenominator);
tmp = idxHelper(mask);
idx2 = tmp(idx2);
val2 = targetLabels(idx2)' * objGrad(idx2);
    if isempty(idx2)
       % Si el metodo ganancia maxima falla pues se toma el metodo par  que viola de forma maxima.
       [val2 idx2] = min(targetLabels(downMaskWr) .* objGrad(downMaskWr));
       tmp = idxHelper(downMaskWr);
       idx2 = tmp(idx2);
end
   % se envia el idx2 al resto de Wr. Se hace igual como para el Idx1. Cada Wr le envia el datos al inmediato superior
if labindex==1
        labSend(idx2,2);  % le envio al wrk inmediato ppal. Por comodidad trato el Wr 1 diferente al rsto
        labSend(val2,2);
        %este es el worker que esta en el front
elseif  labindex==numlabs
       lab=labindex-1;
```

```
        idx2Ant=labReceive(lab);
        val2Ant=labReceive(lab);
    if (val1-val2Ant)>=(val1-val2)      % se toma el valor minimo y ese el que envio al resto de Wr
            shared_data = labBroadcast(numlabs, idx2Ant);
            shared_data = labBroadcast(numlabs, val2Ant);
            idx2=idx2Ant;
             val2=val2Ant;
    else
          shared_data = labBroadcast(numlabs, idx2);
          shared_data = labBroadcast(numlabs, val2);
    end
    itCount= itCount+1;
    if labindex==numlabs
        if mod(itCount, 500) == 0
            fecha=datestr(now);disp(fecha);
            disp(['Numero de iteracciones.....:', num2str(itCount)]);
        end
    end
    % val1-val2
    if ( (val1 -val2) <= tolKKT)                % Análisis de las condiciones de KKT
            para=1;
            shared_data = labBroadcast(numlabs, para);
            offset_1 = (val1 + val2) / 2;
            disp(['Numero de iteraciones.....' , num2str(itCount)]);
            %   se coje el vector de alfas del primer nodo pues da igual. Es decir todos los nodos
            %   tienen el mismo vector de alfas y por eso da igual coger cualquiera.  bucle=false;
            %   salida pues al estar en un bloque spmd no se puede usar el return;
    elseif itCount > smoOptions.MaxIter
             para=1;
             shared_data = labBroadcast(numlabs, para);
     else
            offset_1=(val1+val2)/2;
            para=0;     % orden de parada a todos los Wr xq se ha llegadoa la convergencia
            shared_data = labBroadcast(numlabs, para);
    end
 else   %  se trata el envio que hacen el resto de Wr
        lab=labindex-1;
        idx2Ant=labReceive(lab);   % se toma los valores del Wr anterior
        val2Ant=labReceive(lab);
        if (val1-val2Ant)>=(val1-val2)
              lab=labindex+1;
              labSend(idx2Ant,lab);
              labSend(val2Ant,lab);
        else
             lab=labindex+1;
             labSend(idx2,lab);
              labSend(val2,lab);
        end
 end
if labindex~=numlabs
            idx2= labBroadcast(numlabs);
            val2=labBroadcast(numlabs);
            % se recibe el control de parada desde el Wr ppal.
            para =   labBroadcast(numlabs);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %   ahora ya se tiene el working set es decir idx1 e idx2.
 %  ahora se debe calcular la solucion analitica es decir los alfas
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if ~para   % si no se ha terminado  hay que seguir actualizando
    if ~isempty(fullKernel)   % se tiene la matriz kernel almacenada
          Kii = fullKernel(idx1,idx1);
          Kjj = fullKernel(idx2,idx2);
          Kij = fullKernel(idx1,idx2);
    else
          Kii = kernelDiag(idx1);
          Kjj = kernelDiag(idx2);
          if subKernelIndices(idx1) > 0
                    if labindex==numlabs    % esto es necesario porque solo el wrk ppal tiene la matriz subkernel
                              cont_Kij_SIAlmac= cont_Kij_SIAlmac+1;
```

```
                    Kij = subKernel(idx2, subKernelIndices(idx1));
                    dato=labBroadcast(numlabs, Kij);
            else
                    Kij =labBroadcast(numlabs);
             end
elseif subKernelIndices(idx2) > 0
              if labindex==numlabs
                    cont_Kij_SIAlmac= cont_Kij_SIAlmac+1;
                    Kij = subKernel(idx1, subKernelIndices(idx2));
                    dato=labBroadcast(numlabs, Kij);
               else
                     Kij =labBroadcast(numlabs);
               end
else
      %  Se llega a este apartado solamente  si la columna necesaria
      %  no esta en la caché
      deleteIndex = leastIndices(end);
      if deleteIndex == 0
         subKernel_Index = 1 + max(subKernelIndices);
      else
         subKernel_Index = subKernelIndices(deleteIndex);
         subKernelIndices(deleteIndex) = 0;
      end
      % aqui es donde se almacena el indice de la columna que
      % se guardará.
      subKernelIndices(idx2) = subKernel_Index;
      % Se calcula la nueva columna kernel y la guardará  con el indice 'kernelColIdx'.
      Kij=kernel_col(idx2,rbf_sigma,Lmin1,Lmax1,Lmin2, Lmax2, dataWorker1, dataWorker2,tam1,kfun);
      leastIndices(2:end) = leastIndices(1:end-1);
      leastIndices(1) = idx2;
      cont_Kij_NOAlmac=cont_Kij_NOAlmac+1;
       if labindex==numlabs
             subKernel(:,subKernel_Index)=Kij;
       end
 end
 eta = Kii + Kjj - 2 * Kij;   % ya estan calculados los indices porque no se tiene almacenada matriz kernel
 % los indices directamente
if targetLabels(idx1) * targetLabels(idx2) == 1
   Low = max(0, alphas(idx2) + alphas(idx1) - boxConstraints(idx1));
   High = min(boxConstraints(idx2), alphas(idx2) + alphas(idx1));
else
   Low = max(0, alphas(idx2) - alphas(idx1));
   High = min(boxConstraints(idx2), boxConstraints(idx1) + alphas(idx2) - alphas(idx1));
end
if eta > eps
      % se calculan  los nuevos valores de alpha(i) alpha(j) pero aun no se almacenan
      lambda = - targetLabels(idx1) * objGrad(idx1) + targetLabels(idx2) * objGrad(idx2);
      alpha_j = alphas(idx2) + targetLabels(idx2) / eta * lambda;
      if alpha_j < Low
         alpha_j = Low;
      elseif alpha_j > High
         alpha_j = High;
      end
else
      s = targetLabels(idx1) * targetLabels(idx2);
      fi = - objGrad(idx1) - alphas(idx1) * Kii - s * alphas(idx2) * Kij;
      fj = - objGrad(idx2) - alphas(idx2) * Kjj - s * alphas(idx1) * Kij;
      Li = alphas(idx1) + s * (alphas(idx2) - Low);
      Hi = alphas(idx1) + s * (alphas(idx2) - High);
      psi_l = Li * fi + Low * fj + Li * Li * Kii / 2 + Low * Low * Kjj / 2 +  s * Low * Li * Kij;
      psi_h = Hi * fi + High * fj + Hi * Hi * Kii / 2 + High * High * Kjj / 2 + s * High * Hi * Kij;
      if psi_l < (psi_h - eps)
            alpha_j  = Low;
      elseif psi_l > (psi_h + eps)
            alpha_j = High;
      else % no hay progreso
            alpha_j = alphas(idx2);
         npgs=npgs+1;
      end
end
```

```
% nuevo valor alpha(i)
alpha_i = alphas(idx1) + targetLabels(idx2) * targetLabels(idx1) *(alphas(idx2) - alpha_j);
 if alpha_i < eps
    alpha_i = 0;
elseif alpha_i > (boxConstraints(idx1) - eps)
    alpha_i = boxConstraints(idx1);
end
% se actualiza el gradiente
if isempty(fullKernel)
     if ~isempty(fullKernel)
          kerCol_idx1 = fullKernel(:, idx1);
     else
          if subKernelIndices(idx1) == 0
               % si no esta almacenada la columna
               % kernel debe calcularse
                deleteIndex = leastIndices(end);
                if deleteIndex == 0
                     subKernel_Index = 1 + max(subKernelIndices);
                else
                   subKernel_Index = subKernelIndices(deleteIndex);
                   subKernelIndices(deleteIndex) = 0;
                end
                  subKernelIndices(idx1) = subKernel_Index;
                % se calucla la nueva columna kernel y se guardara con el indice 'kernelColIdx'.
                % la función kernel_col devuelve la columna de la matriz kernel correspondiente a la fila idx1
                tiempo1=tic;
                     kerCol_idx1=kernel_col(idx1,rbf_sigma,Lmin1,Lmax1,Lmin2, Lmax2, dataWorker1,
               dataWorker2,tam1,kfun);
               tiempoTOT=toc(tiempo1)+tiempoTOT;
                leastIndices(2:end) = leastIndices(1:end-1);
                leastIndices(1) = idx1;
            %    kerCol_idx1 = subKernel(:, subKernel_Index);
                if labindex==numlabs
                          subKernel(:,subKernel_Index)=kerCol_idx1;
                end
                cont_kerCol_idx1_NOAlmac=cont_kerCol_idx1_NOAlmac+1;
          else
                cont_kerCol_idx1_SIAlmac=cont_kerCol_idx1_SIAlmac+1;
                 if labindex==numlabs
                          kerCol_idx1 = subKernel(:, subKernelIndices(idx1));
                          dato=labBroadcast(numlabs,kerCol_idx1);
                else
                          kerCol_idx1=labBroadcast(numlabs);
                 end
          end
     end
      if ~isempty(fullKernel)
           kerCol_idx2 = fullKernel(:, idx2);
      else
          if subKernelIndices(idx2) == 0
                deleteIndex = leastIndices(end);
             if deleteIndex == 0
                % la variable subKernel no esta completa aun
                subKernel_Index = 1 + max(subKernelIndices);
             else
                subKernel_Index = subKernelIndices(deleteIndex);
                subKernelIndices(deleteIndex) = 0;
             end
              subKernelIndices(idx2) = subKernel_Index;
              % se calcula la nueva columna kernel y se guradara con el indice 'kernelColIdx'.
             % la función kernle_col devuelve la columna de la matriz kernel correspondiente a la fila idx2
             tiempo1=tic;
              kerCol_idx2=kernel_col(idx2,rbf_sigma,Lmin1,Lmax1,Lmin2, Lmax2, dataWorker1,
             dataWorker2,tam1,kfun);
              tiempoTOT=toc(tiempo1)+tiempoTOT;
               leastIndices(2:end) = leastIndices(1:end-1);
             leastIndices(1) = idx2;
             if labindex==numlabs
                          subKernel(:,subKernel_Index)=kerCol_idx2;
             end
```

```
                    cont_kerCol_idx2_NOAlmac=cont_kerCol_idx2_NOAlmac+1;
                else
                    cont_kerCol_idx2_SIAlmac=cont_kerCol_idx2_SIAlmac+1;
                    if labindex==numlabs
                        kerCol_idx2= subKernel(:, subKernelIndices(idx2));
                        dato=labBroadcast(numlabs,kerCol_idx2);
                    else
                        kerCol_idx2=labBroadcast(numlabs);
                    end
                end
            end
            objGrad = objGrad - (kerCol_idx1 .* targetLabels) * (alpha_i - alphas(idx1)) * targetLabels(idx1) - ...
                        ( kerCol_idx2 .* targetLabels) * (alpha_j - alphas(idx2)) * targetLabels(idx2);
        else
            objGrad = objGrad -  (fullKernel(:,idx1) .* targetLabels) *  (alpha_i - alphas(idx1)) * targetLabels(idx1) - ...
                        (fullKernel(:,idx2) .* targetLabels) * (alpha_j - alphas(idx2)) * targetLabels(idx2);
        end
        % se actualizan las mascaras
        upMask(idx1) = targetLabels(idx1) * alpha_i < (Bvec(idx1) - svTol);
        downMask(idx1) = targetLabels(idx1) * alpha_i > (Avec(idx1) + svTol);
        upMask(idx2) = targetLabels(idx2) * alpha_j < (Bvec(idx2) - svTol);
        downMask(idx2) = targetLabels(idx2) * alpha_j > (Avec(idx2) + svTol);
        upMaskWr(Lmin1:Lmax1,:)= upMask(Lmin1:Lmax1,:);
        upMaskWr((Lmin2+tam1):(Lmax2+tam1),:)=upMask((Lmin2+tam1):(Lmax2+tam1),:);
        downMaskWr(Lmin1:Lmax1,:)= downMask(Lmin1:Lmax1,:);
        downMaskWr((Lmin2+tam1):(Lmax2+tam1),:)=downMask((Lmin2+tam1):(Lmax2+tam1),:);
        % se actualiza el vector de alfas
        alphas(idx1) = alpha_i;
        alphas(idx2) = alpha_j;
      end
  end % del if ~para
 end    % del while
end %del spmd
disp(['tiempo invertido en segundos tiempo_5 .....:', num2str(toc(tiempo5))]);
tiempo=tiempoTOT{numlabs};
disp(['tiempo invertido en busquda columna kernel .....:', num2str(tiempo)]);
tiempo6=tic;
% el offset lo tiene el worker ppal que es el que lleva el control de los Working set
offset=offset_1{numlabs};
% como todos los vectores de alphas deben ser iguales porque se han actualizado de igual forma
 alphas=alphas{1};
% Se chequean las condiciones de 0<=alpha<= boxconstraints..
for i = 1:length(alphas)
    if ~( (alphas(i) >= -eps) && (alphas(i) <= boxConstraints(i) + eps))
        error(message('stats:svmtrain:BoxConstraintViolation', ...
            i, sprintf('%g',alphas(i)), sprintf('%g',boxConstraints(i))));
    end
end

linEqConstraint = targetLabels' * alphas;
if abs(linEqConstraint) > 5
    error(message('stats:svmtrain:se hanvioladoPostcondiciones'));
end
disp(['tiempo invertido en segundos tiempo_6 .....:', num2str(toc(tiempo6))]);
end
```

```
=============================================================================================

function    [tasaExito_int_Inf,  tasaExito_int_Sup,   tasaExito_tot_Acum]  =VPprocesa_pll(VP_opts,ruta,varargin)
% función que implementa el algoritmo del predictor Venn IVP.
 warning off
numW=VP_opts.numWorkers;
C=VP_opts.C;
sigma=VP_opts.sigma;
numBandas=VP_opts.numBandas;
kernel=VP_opts.funcionKernel;
valorTrain=VP_opts.valortrain;
numClases=length(varargin{1}{1});
for j=1:numClases
     vars = whos('-file', fullfile(ruta, char(varargin{1}{1}(j))));
     load( fullfile(ruta, char(varargin{1}{1}(j))) );
     if  VP_opts.porcentaje>0
        eval([' [cont,~ ]=size(' vars.name ');']);
%           eval([' cont=fix(((VP_opts.porcentaje)*length(' vars.name '))/100);' ]);
         eval([' matriz_' num2str(j) '='  vars.name '(1:cont,:);' ]); %t  esto es para el caso que no queramos usar todas las muestras
     else
          eval([      'matriz_' num2str(j) '='  vars.name ';' ]);
     end
     eval([      '[~,long(j)]=size(matriz_' num2str(j) ');' ]);%en long tengo las longitudes detodas las matrices pori si luego hay que
normalizar
     clear('vars(1).name');
     eval([      'indices_' num2str(j) '=  randperm(length([1:length( matriz_'       num2str(j) '(:,1))]" ));']); % caluclo indices para
luego tomar
     eval([ '[cnt,ln]=size(matriz_' num2str(j) ');']);
end%al azar las muestras
  MATRIZTot_test=[];
  MATRIZTrain=[];
for j=1:numClases
                eval([ '[cnt,ln]=size(matriz_' num2str(j) ');']);
                if ln>min(long)
                        matriz_tmp=zeros(cnt,min(long));
                        eval([ 'matriz=  matriz_' num2str(j) ';']);
                        tiempo=[1:1:ln];
                        intervalo=(    (ln-1)/min(long)    );
                        tiempo2=[1:intervalo:ln];
                        for k=1:cnt
                                temp=spline(tiempo,matriz(k,:),tiempo2);
                                 matriz_tmp(k,:)=temp(1,1:min(long));
                        end
                       % la matriz ya esta normalizada con el minimo de long de todas las matrices
                        eval([ 'matriz_' num2str(j) '=matriz_tmp;']);
                        clear('matriz_tmp');
                 end
                % alteramos las filas para tomar al azar las muestras
                eval([ 'matriz_' num2str(j) '=matriz_' num2str(j) '(indices_' num2str(j) ',:);']);
                % creamos la matriz de test y training  de cada clase
                eval([ 'matrizTest_' num2str(j) '=matriz_' num2str(j) '(valorTrain+1:end,:);']);
                eval([ '[~,longMuestra]=size(matrizTest_' num2str(j) ');']);
               eval([ 'matrizTest_'  num2str(j)  '(:,longMuestra+1)=' num2str(j) ';']);
                eval([ 'MATRIZTest{'  num2str(j)  '}=matrizTest_'  num2str(j)  ';']);
                eval([ 'MATRIZTrain{' num2str(j) '}=matriz_' num2str(j) '(1:valorTrain,:);']);
                eval([ '[muestrasTest_' num2str(j) ',~]=size(matrizTest_' num2str(j) ');']);
                %creo la matriz general de test de donde ire sacando las muestras para analizarlas
                eval([ '[MATRIZTot_test]=[MATRIZTot_test;matrizTest_' num2str(j)  '];']);
%                    eval([ '[MATRIZTrain]=[MATRIZTrain;MATRIZTrain{' num2str(j) '}];');  % la MATRIZtrain de cada calse debo
guardarla
%                                                                                       % pq en cada iteracion debo añadir la muestra
a cada clase
%                                                                                       %una vez analizada la muestra
end
clear('matrizTest_*');
clear('matriz_*');
 % Muy importante: como se han añadido todas las muestras de test de cada  clase
 % entonces las muestras de la matriz de test estan ordenadas. En la fase de test del predictor las muestras le deben llegar al
 % azar. Por este motivo se deben alterar (mediante la instrucción randperm) el orden de las muestras de esta matriz test y
 % de esta forma se consigue que todas las muestras de test lleguen de forma aleatoria, como si las muestras llegaran al azar.
```

```
 indices_Gen=randperm(length([1:length(MATRIZTot_test(:,1))]'))';
 MATRIZTot_test=MATRIZTot_test(indices_Gen,:);
 analiza=true;
 disp('Numero de muestras...:');
 [numMuestras,~]=size(MATRIZTot_test)
 %numMuestras=500;
 tablaWorker=zeros(numW,1);
 [~,longMuestra]= size(MATRIZTrain{1});   % para ver la long de cada muestra me vale con ver la de la primera matriz, el resto
es igual
%  mkdir(fullfile(pwd,'intercambio'))
% workersEstado=[NumW,N];   % en esta variable tendremos el estado de cada worker para cada iteracción;
spmd
    if labindex==numW   % porque en el clustrer DIA al ult worker es el que se pone en solitario llevara el control de  los Wk's
                        contSamples=0;
                          contSamplesG=0;
                        PREDICCIONES=zeros(numMuestras,numClases);
                        PREDICCIONES_LIMITES=zeros(numMuestras,numClases);
                        INTERVALOS=zeros(numMuestras,11);
                        INTERVALOS1=zeros(numMuestras,11);
                       %       INTERVALOS2=zeros(numMuestras,9);
                        % preparacion de los datos
                         p=1;
                        CL=1;
                         claseEstado=zeros(numClases,numMuestras);  % para controlar si por cada muestra tengo la prediccion de
                       cada clase
                        tablaWorker=zeros(numW-1);  % porque no cuenta el wrk 1 wue es el ppal
                        for  worker=1:(numW-1)     % ahora a cada worker le envio datos hasta completar todos los worker
                                  dato.activo=true;
                                  % creo tantos campos como clases hay
                                 %   size(MATRIZTrain{1})
                                     dato.MATRIZ=MATRIZTrain;    % aqui van todas las matrices de training. El programa prediceClase
                                     se encargara de generar
                                    dato.muestra=MATRIZTot_test(p,1:longMuestra);    % la matriz de training que corresponda en
                                     función de la clase a evaluar
                                   dato.numMuestra=p;
                                   dato.clase=CL;
                                    labSend(dato,worker);   % xq comienza cn e lqorker 2
                                    tablaWorker(worker)=1;  % aqui conocen  los workers activos
                                    CL=CL+1;
                                    if CL>numClases      % hay que introducir que introducir la muestra de la matriz a la que pertenece
                                        cod=MATRIZTot_test(p,longMuestra+1);
                                        MATRIZTrain{cod}=[MATRIZTrain{cod};MATRIZTot_test(p,1:longMuestra)];
                                        p=p+1;
                                        CL=1;
                                    end
                        end
                        %Ahora comienzo a recoger y enviar
                        sigue =true;
                        numMuestraOk=1;
                        while sigue
                                  tstart=tic;
                                  [datoRec, worker, tag] = labReceive;
                                  PREDICCIONES(datoRec.numMuestra,datoRec.clase+1)=datoRec.prediccion;
                                  PREDICCIONES(datoRec.numMuestra,1)=datoRec.numMuestra;
                                  PREDICCIONES_LIMITES(datoRec.numMuestra,datoRec.clase)=datoRec.limites;
                                  INTERVALOS(datoRec.numMuestra,datoRec.clase*2)=datoRec.intSup;
                                  INTERVALOS(datoRec.numMuestra,(datoRec.clase*2)+1)=datoRec.intInf;
%                                    chequeo e informo de las muestras
%                                    procesadas
                                  contSamples=     contSamples+1;
                                  if mod(contSamples,2000)==0    %es para llevar conteo del numero de Análisis en curso
                                        contSamplesG=contSamplesG+1;
                                        % Cada muestra se analiza    para cada una de las  clases existentes por eso
                                        % los Análisis se multiplican por num de  clases.
                                         tstart=tic;
                                  end
                                 %
                                  dato.activo=true;
                                   dato.MATRIZ=MATRIZTrain;
                                  dato.muestra=MATRIZTot_test(p,1:longMuestra);
```

```
                        dato.numMuestra=p;
                        dato.clase=CL;
                        labSend(dato,worker);  % se lo envio al worker que ha quedado libre
                        tablaWorker(worker)=1;  % aqui conozco los workers activos estados 0:libre 1:trabaja y 2:liberado
                        CL=CL+1;
                        if (CL>numClases)&&(p<=numMuestras)
                                cod=MATRIZTot_test(p,longMuestra+1);
                                MATRIZTrain{cod}=[MATRIZTrain{cod};MATRIZTot_test(p,1:longMuestra)];
                                p=p+1;
                                CL=1;
                        end
                        if p>numMuestras
                                sigue=false;     % me salgo de este bucle porque he llegado al final de muestras
                        end
                end  % end del while
                % a partir de aquie ya no queda mustras que y tengo que recoger los resultados de lo worker
                % que quedam;
                revisaWorker=true;
%                   p=numMuestraOk+1;
                while revisaWorker
                        trabajando=find(tablaWorker==1);
                        if ~isempty(trabajando)   % esto indica que hay workers trabajando
                           for worker=1:length(trabajando)
                                [datoRec, worker, tag] = labReceive;
                                %           claseEstado(datoRec.clase,datoRec.numMuestra)=datoRec.valor; % esto indica el estado
                                si analizado o no de la muestra
                                PREDICCIONES(datoRec.numMuestra,  datoRec.clase+1)=datoRec.prediccion;
                                PREDICCIONES(datoRec.numMuestra,1)=datoRec.numMuestra;%disp('1');
                                PREDICCIONES_LIMITES(datoRec.numMuestra,datoRec.clase)=datoRec.limites;%disp('2');
                                INTERVALOS(datoRec.numMuestra,datoRec.clase*2)=datoRec.intSup;%disp('3');
                                INTERVALOS(datoRec.numMuestra,(datoRec.clase*2)+1)=datoRec.intInf;%disp('4');
                               dato.activo=false;
                               for t=numClases
                                   MATRIZTrain{t}=0;
                               end
                               dato.MATRIZ=MATRIZTrain;
                               dato.muestra=0;
                               dato.numMuestra=0;%disp('5')
                               dato.clase=0;
                               labSend(dato,worker);  % se lo envio al worker que ha quedado libre
                               tablaWorker(worker)=2;%disp('6')
                           end
                        end
                        workerActivos=length(find(tablaWorker==2));
                        %disp('7')
                        if workerActivos==(numW-1)
                           revisaWorker=false;
                        end
                end % del  while revisaWorker
  % AQUI TRABAJAN EL RESTO DE WORKERS
 else
     datoRec= labReceive(numW);  %lectura adelantada
     activo=datoRec.activo;
     while activo
                [datoEnv.prediccion,datoEnv.limites, datoEnv.intSup, datoEnv.intInf] =
prediceClase(C,numBandas,sigma,kernel,datoRec);
                datoEnv.numMuestra=datoRec.numMuestra;     % es el numero de la muestra
             datoEnv.clase=datoRec.clase;
             labSend(datoEnv,numW);
             datoRec= labReceive(numW);
             activo=datoRec.activo;
     end
       %  %disp('me desconnnen');
    end
    %disp('paso por aqui 222');
end
clear('matrizTrain*');
PREDICCIONES=PREDICCIONES{numW};
PREDICCIONES_LIMITES= PREDICCIONES_LIMITES{numW};
INTERVALOS=INTERVALOS{numW};
```

```
INTERVALOS1=INTERVALOS1{numW};
contadorTot=0;
tasaExito_tot_Acum=zeros(numMuestras,1);
tasaExito_int_Sup=[];
tasaExito_int_Inf=zeros(numMuestras,1);
 tasaExito_tot=0;
 maxSUPERIORESgen=0;
 maxSUPERIORES=0;
 maxINFERIORESgen= 0;
 maxINFERIORES =0;
 minSUPERIORESgen=0;
 minSUPERIORES=0;
  tic
for j=1:numMuestras
                [~, ~, intCorto]=calculaAciertos_acumulado_siete_V4(MATRIZTot_test(j,end), PREDICCIONES(j,:),
PREDICCIONES_LIMITES(j,:));
                clase=MATRIZTot_test(j,end);
                intSup=[];
                intInf=[];
                for h=1:numClases
                      intSup(h)=INTERVALOS(j,h*2);
                       intInf(h)=INTERVALOS(j,h*2+1);
                end
                 maxSUPERIORES=max( intSup);
                  maxINFERIORES=max( intInf);
                 minSUPERIORES=min(  intSup);
                  minSUPERIORESgen=minSUPERIORES+ minSUPERIORESgen;
                  maxSUPERIORESgen=maxSUPERIORES+ maxSUPERIORESgen;
                 maxINFERIORESgen= maxINFERIORES +maxINFERIORESgen;
              if intCorto==1
                      tasaExito_tot= tasaExito_tot+1;
                      tasaExito_tot_Acum(j,1)=tasaExito_tot/j;
                      tasaExito_int_Sup(j,1)=(      maxSUPERIORESgen            )/j;
                     tasaExito_int_Inf(j,1)= (      maxINFERIORESgen    )/j;
              else
                     tasaExito_tot_Acum(j,1)=(tasaExito_tot)/j;
                      tasaExito_int_Sup(j,1)=(      maxSUPERIORESgen            )/j;
                      tasaExito_int_Inf(j,1)= (      maxINFERIORESgen     )/j;
               end;
end
toc
pause(1);

=============================================================================================


function [minimo   intervalo intSup intInf]=prediceClase(C_param,numBandas,sigma,kernel,dato)
% función que calcula la mtriz de probabilidades del predictor Venn
[~, numClases]=size(dato.MATRIZ);
matriz1_2=dato.MATRIZ{dato.clase};
matriz2_2=[];
for  h=1:numClases
   if h~=dato.clase
        matriz2_2=[matriz2_2;dato.MATRIZ{h}];
   end
end
 muestraTest=dato.muestra;
 clear('dato*');
[lon_dat1,~]=size(matriz1_2);
[lon_dat2,~]=size(matriz2_2);

etiq1(1:lon_dat1)=1;
etiq2(1:lon_dat2)=0;
etiq=[etiq1,etiq2];
etiq=etiq';
Etiq_02= nominal(ismember(etiq2,1));
Etiq_01= nominal(ismember(etiq1,1));
datos=[matriz1_2;matriz2_2];
etiquetas= [Etiq_01,Etiq_02]';       % se puede comprobar que analiticamente P_01.training=P_03.test
optiones = statset('MaxIter',120000);
                      % ------------------------------------------------------------------------
```

```
%   LE PONEMOS ETIQUETA  TRUE A LA MUESTRA  ACTUAL TESTING Y  CALCULAMOS LAS
 %  MUESTRAS EN CADA
%   UNA DE LAS BANDAS. OJO despues de la reu
%   del 2/11/15 se deb ecalcular las bandas
%   cada vez
%-------------------------------------------------------------------------------
datos=[matriz1_2;matriz2_2];
[puntero,~]=size(datos);
       % Cambio las lineas de codigo para meter en la primera fila el punto a analizar. El motivo es que las
        %  filas de las matrices
% deben ser multiplo del numero de woker porque en caso contrario falla la aplicacion del metodo spread
 %  Kernal
datos(puntero+1,:)=   muestraTest;
etiquetas(puntero+1)='true';      %  le pongo la primera de las muetras a evaluar y le pongo true
%
% cambio despues de la reunion del 2/11/15
[bandas] =calculaBandas_V2(C_param,numBandas,sigma,kernel,matriz1_2, matriz2_2);
    [svm_struct] =...
    svmtrain(datos,etiquetas,'showplot',false,'options',optiones,'kernel_function',kernel,'RBF_Sigma', sigma,…
     'BoxConstraint', C_param);

clear('svmtrain');
 for c = 1:size(datos, 2)
       datos(:,c) = svm_struct.ScaleData.scaleFactor(c) * (datos(:,c) +  svm_struct.ScaleData.shift(c));
 end
 [~,distancias] = svmdecision(datos,svm_struct);
 %
[~,col,~] = find(bandas<distancias(end));  % establecemos en que banda esta el punto a analizar que estará
                                           %  en "col(end)" porque hemos
 if isempty(col)       % este caso es cuando tiende a - infinito
              indices2=(distancias<=bandas(1));           % miro cuantas muestras hay por debajo de esa banda
              vector=etiquetas(indices2);                % en vector almaceno todas las etiquetas de esas
                                                          %  muestras
              Falsos=length(find(vector=='false'));       % cuento las que hay con etiqueta -1
              Verdad=length(find(vector=='true'));        % cuento las que hay con etiqueta -1
 elseif   (col(end)==length(bandas))                      % este caso es cuando tiende a + infinito
              indices1=(distancias>bandas(end));         % miro cuantas muestras hay por encima o igual de esa
                                                         %  banda
              vector=etiquetas(indices1);                % en vector almaceno todas las etiquetas de esas
                                                         %  muestras
              Falsos=length(find(vector=='false'));       % cuento las que hay con etiqueta -1
              Verdad=length(find(vector=='true'));
 else
              indices1=(distancias>bandas(col(end)));    % analizo el limite de la banda inferior
              indices2=(distancias<=bandas(col(end)+1)); % analizo el limite de la banda superior
              vector=etiquetas(indices1&indices2);          % con este AND tengo los indices del vector de
                                                            %  distancias
              Falsos=length(find(vector=='false'));         % cuento las que hay con etiqueta -1
              Verdad=length(find(vector=='true'));          % cuento las que hay con etiqueta 1
 end
 fil11=Verdad/(Verdad+Falsos);
 fil12=Falsos/(Verdad+Falsos);
% -------------------------------------------------------------------------------
%   LE PONEMOS ETIQUETA  FALSE A LA MUESTRA  ACTUAL, CALCULAMOS LAS MUESTRAS EN
 % CADA
%   UNA DE LAS TRES BANDAS.
%-------------------------------------------------------------------------------
datos=[matriz1_2;matriz2_2];
[puntero,~]=size(datos);
% Cambio las lineas de codigo para meter en la primera fila el punto a analizar. El motivo es que las filas de
 % las matrices
% deben ser multiplo del numero de woker porque en caso contrario falla la aplicacion del metodo spread
 % Kernal
%datos(puntero+1,:)=  matrizTEST(k,:);
%etiquetas(puntero+1)='true';
datos(puntero+1,:)=   muestraTest;
etiquetas(puntero+1)='false';      %  le pongo la primera de las muetras a evaluar y le pongo true
%
% cambio despues de la reunion del 2/11/15
[bandas] =calculaBandas_V2(C_param,numBandas,sigma,kernel,matriz1_2, matriz2_2);
```

```
                    [svm_struct] =
svmtrain(datos,etiquetas,'showplot',false,'options',optiones,'kernel_function',kernel,'RBF_Sigma', sigma, 'BoxConstraint',
C_param);
                    clear('svmtrain');
                   for c = 1:size(datos, 2)
                         datos(:,c) = svm_struct.ScaleData.scaleFactor(c) * (datos(:,c) +  svm_struct.ScaleData.shift(c));
                    end
                    [~,distancias] = svmdecision(datos,svm_struct);
                   [~,col,~] = find(bandas<distancias(end));
               % establecemos en que banda esta el punto a analizar que estara en "col(end)" porque hemos
                    if isempty(col)   % este caso es cuando tiende a - infinito
                              indices2=(distancias<=bandas(1));          % miro cuantas muestras hay por debajo de esa banda
                              vector=etiquetas(indices2);                % en vector almaceno todas las etiquetas de esas
                                                                         muestras
                              Falsos=length(find(vector=='false'));      % cuento las que hay con etiqueta -1
                              Verdad=length(find(vector=='true'));       % cuento las que hay con etiqueta -1
                    elseif   (col(end)==length(bandas))                  % este caso es cuando tiende a + infinito
                              indices1=(distancias>bandas(end));        % miro cuantas muestras hay por encima o igual de esa
                                                                         banda
                              vector=etiquetas(indices1);                % en vector almaceno todas las etiquetas de esas
                                                                           muestras
                              Falsos=length(find(vector=='false'));      % cuento las que hay con etiqueta -1
                              Verdad=length(find(vector=='true'));
                    else
                              indices1=(distancias>bandas(col(end)));    % analizo el limite de la banda inferior
                              indices2=(distancias<=bandas(col(end)+1)); % analizo el limite de la banda superior
                              vector=etiquetas(indices1&indices2);          % con este AND tengo los indices del vector de
                                                                              distancias
                              Falsos=length(find(vector=='false'));         % cuento las que hay con etiqueta -1
                              Verdad=length(find(vector=='true'));          % cuento las que hay con etiqueta 1
                    end
                    fil21=Verdad/(Verdad+Falsos);
                  minimo=min(fil11,fil21);
                  intervalo=(fil11+fil21)/2;
                   intSup=max(fil11, fil21);
                    intInf=min(fil11, fil21);
                  clear('a', 'b', 'c', 'tot*','svm_struct',' distanciaG');
                   %

==========================================================================================================


function [bandas]=calculaBandas_V2(C_param,numBandas,sigma,kernel,matriz1_2, matriz2_2);
% Este pgm calcula los límites de cada banda en función del numero de bandas
% la variable bandas es un vector de 1xn
[lon_dat1,~]=size(matriz1_2);
[lon_dat2,~]=size(matriz2_2);
etiq1(1:lon_dat1)=1;
etiq2(1:lon_dat2)=0;
etiq=[etiq1,etiq2];
etiq=etiq';
Etiq_02= nominal(ismember(etiq2,1));
Etiq_01= nominal(ismember(etiq1,1));
datos=[matriz1_2;matriz2_2];
etiquetas= [Etiq_01,Etiq_02]';       % se puede comprobar que analiticamente P_01.training=P_03.test
optiones = statset('MaxIter',120000);
%
[svm_struct] = svmtrain(datos,etiquetas,'showplot',false,'options',optiones,'kernel_function',kernel,'RBF_Sigma', sigma,
'BoxConstraint', C_param);
%
%           '
clear('svmtrain');
% Ahora se calcula la distancia de todos los puntos  y normalizo las distancias
 for c = 1:size(datos, 2)
         datos(:,c) = svm_struct.ScaleData.scaleFactor(c) * (datos(:,c) +  svm_struct.ScaleData.shift(c));
end
[out,distancias] = svmdecision(datos,svm_struct);
 %  voy a tomar  "numBandas+2" bandas, donde las bandas intermedias tendrán valores positivos y negativos.
 % para ello debo dividir el array de distancias en partes iguales. Las bandas de los extremos   van desde menos infinito hasta
%la primera  celda del vector de las distancias y desde la ultima celda del vector distancias hasta más infinito
```

```
%  tramo= fix(length(distancias)/(numBandas-2));   % en variable tramo tengo el numero de celdas
 resto=rem(numBandas,2);
 % en el vector bandas tendre los intervalos que delimitan cada banda
 distanciasGEN=sort(distancias, 'ascend');
 clear('banda');
%  resto=rem((numBandas-2), 2);
%  intervalo=distanciasGEN(1)/((numBandas-2)/2);
 if (numBandas==3) | (resto~=0)
%                resto=rem((numBandas-2), 2);
               intervalo=distanciasGEN(1)/((numBandas-2));
               bandaP(1)=abs(intervalo);
               bandaP(2)=intervalo;
               k=3;
               top=fix(numBandas/2);
               F=3;
               while k<=(2*top)
                                bandaP(k)=(intervalo*F);
                                bandaP(k+1)=abs(bandaP(k));
                                k=k+2;
                                F=F+2;
               end
                bandas=sort(bandaP, 'ascend');
             return;
 end
if distanciasGEN(1)<distanciasGEN(end)
            resto=rem((numBandas-2), 2);
            intervalo=distanciasGEN(1)/((numBandas-2)/2);
            if resto~=0
                 error('myApp:argChk', 'solo numero bandas  pares');
                 return;
            end
             banda(1)=0;
            k=0;
            for i=1:(numBandas-2)/2
                      k=i+1;
                      banda(k)=intervalo*((i+1)-1);
            end
            for j=1:(numBandas-2)/2
               k=k+1;
               banda(k)=abs(intervalo*((j+1)-1));
            end
 else
              resto=rem((numBandas-2), 2);
            intervalo=distanciasGEN(end)/((numBandas-2)/2);
            if resto~=0
                 error('myApp:argChk', 'solo numero bandas  pares');
                 return
            end
            banda(1)=0;
            k=0;
            for i=1:(numBandas-2)/2
                      k=i+1;
                      banda(k)=intervalo*((i+1)-1);
            end
            for j=1:(numBandas-2)/2
               k=k+1;
               banda(k)=(-1)*(intervalo*((j+1)-1));
            end
end
bandas=sort(banda, 'ascend');

function analisis_multiclase(Xprueba)
close all
load;
close all
figure(1);plot(x3(:, 1), x3(:, 2), '*g');hold on;
plot(x2(:, 1), x2(:, 2), 'xr');hold on;
plot(x1(:, 1), x1(:, 2), 'ok');hold on;
hold on
figure(1)
```

```
plot(Xprueba(:, 1), Xprueba(:, 2), 'Ob')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  clase 1 frente a 2 y 3
etiq1=[];
etiq2=[];
Xresto=[x2;x3];
%Xone=[x1;Xprueba];
Xone=[x1];
[pValor]=calculaPvalor(Xone,Xresto, Xprueba);
disp(['Probabilidad normal de 1 frente a clase 2 y 3......:', num2str(pValor)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  clase 2 frente a 1 y 3
etiq1=[];
etiq2=[];
Xresto=[x1;x3];
%Xone=[x2;Xprueba];
Xone=[x2];
[pValor]=calculaPvalor(Xone,Xresto, Xprueba);
disp(['Probabilidad normal de 2 frente a clase 3 y 1......:', num2str(pValor)]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  clase 3 frente a 1 y 2
etiq1=[];
etiq2=[];
Xresto=[x1;x2];
Xone=[x3];
[pValor]=calculaPvalor(Xone,Xresto, Xprueba);
disp(['Probabilidad normal de 3 frente a clase 2 y 1......:', num2str(pValor)]);
%
function [pValor]=calculaPvalor(Xone,Xresto, Xprueba)
[lon_dat1,~]=size(Xone);
[lon_dat2,~]=size(Xresto);
etiq1(1:lon_dat1)=1;
etiq2(1:lon_dat2)=0;
etiq1=etiq1';
etiq2=etiq2';
etiq=[etiq1;etiq2;1];    % asumo que la muestra es de la clase 1
etiquetas= nominal(ismemberf(etiq,1));
training=[Xone;Xresto;Xprueba]; %incluyo la muestra
figure(2);
[svmStruct] = svmtrain(training,etiquetas,'showplot',true,'kernel_function','rbf');
[~,distancia1]=svmdecision(   training, svmStruct  );
pValor=mean(abs(distancia1)>=abs(distancia1(end)));
```