Universidad
Carlos III de Madrid

TESIS DOCTORAL

Paralelización Eficiente en Métodos de Núcleo

Autor:

Roberto Díaz Morales

Director:

Dr. Ángel Navia Vázquez

Dpto. de Teoría de la Señal y Comunicaciones

Leganés, Febrero de 2016

TÉRMINOS DE USO

"*Cuando bebas agua,*
recuerda la fuente"
Proverbio Chino

E-mails: rdiazm@tsc.uc3m.es, rober.diaz@gmail.com

Dirección/Address:
Departamento de Teoría de la Señal y Comunicaciones
Universidad Carlos III de Madrid
Avenida de la Universidad 30
Leganés 28911 - España

TESIS DOCTORAL

Paralelización Eficiente en Métodos de Núcleo

Autor:
Roberto Díaz Morales

Director:
Dr. Ángel Navia Vázquez

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Secretario:

Calificación:

Leganés, de Febrero de 2016

A mi hija Lucía de todo

$(x^2 + y^2 - 1)^3 - x^2 y^3 = 0$

Resumen

"*He redactado esta carta más extensa de lo usual porque carezco de tiempo para escribirla más breve.*"

Blaise Pascal (1623 -1661)

Los métodos de núcleo son técnicas muy utilizadas en el área de aprendizaje automático debido a que son capaces de obtener buenos resultados en un gran número de tareas reales. El principal motivo de su éxito es que permiten crear soluciones no lineales de forma muy sencilla, transformando el espacio de entrada en un espacio de alta dimensionalidad donde los productos escalares se obtienen utilizando una función de núcleo. Entre las técnicas más extendidas tenemos por un lado las Máquinas de Vectores Soporte (SVMs), que funcionan muy bien en problemas de clasificación, y por otro lado los Procesos Gaussianos (GPs), muy utilizados en problemas de regresión.

Los métodos de núcleo siempre han tenido dos puntos débiles que limitan su aplicación práctica. En primer lugar está el coste computacional asociado a entrenar este tipo de algoritmos, que es $\mathcal{O}(n^3)$ siendo n el número de datos que contiene el conjunto de entrenamiento, este problema de escalabilidad reduce mucho su campo de actuación. El segundo problema está asociado a que sean métodos no paramétricos, en los métodos de núcleo la complejidad de los modelos se ajusta automáticamente y no viene prefijada de antemano. Esta capacidad de autoajuste hizo muy populares este tipo de técnicas pero cuando se trabaja con volúmenes de datos grandes se

obtienen modelos demasiado complejos que son muy lentos para procesar nuevas muestras.

En los últimos años una serie de factores como la redes sociales, el comercio electrónico, los dispositivos móviles y la colaboración utilizando las nuevas tecnologías han hecho que las bases de datos se hayan incrementado enormemente y que se haya pasado de la problemática de no tener suficientes datos para solucionar una tarea al problema completamente opuesto de tener tal volumen de datos que resulte muy costoso y complicado procesarlos.

Si al hecho de que actualmente se disponga de bases de datos mucho más grandes le unimos la problemática que tienen los métodos de núcleo en cuanto a coste computacional y escalabilidad, nos encontramos con que este tipo de técnicas ha visto muy reducido su campo de acción.

Afortunadamente también han surgido líneas de investigación que permiten enfrentarnos a este tipo de problemas como son la paralelización y las aproximaciones semi-paramétricas.

Por un lado, el número de núcleos de procesamiento en los ordenadores se ha incrementado considerablemente dando lugar a nueva líneas de investigación para adaptar técnicas clásicas de aprendizaje automático a un nuevo escenario de paralelización en el que múltiples recursos trabajan de forma simultánea bajo el principio de que los problemas pueden dividirse en tareas más pequeñas que pueden resolverse al mismo tiempo.

Por otro lado, las soluciones semi-paramétricas son aproximaciones que fijan de antemano la complejidad de los modelos que se obtienen y evitan el problema de obtener soluciones demasiado lentas a la hora de evaluar nuevas muestras. Este tipo de soluciones pueden además reducir el coste computacional del entrenamiento para evitar que escale de forma cúbica con el número de muestras.

En esta Tesis se hace uso de estas líneas y se presentan nuevos algoritmos paralelos para entornos multiprocesador y multinúcleo de versiones tanto completas como semi-paramétricas de diferentes métodos de núcleo.

En primer lugar, se presenta un nuevo esquema de paralelización que se beneficia de la formulación matricial de los métodos de núcleo y propone unos criterios de diseño tanto de bajo como de alto nivel de las operaciones que se llevan a cabo en este tipo de algoritmos y permiten una implementación paralela eficiente. Para demostrar la utilidad de estas técnicas y con el objeto de aunar las dos principales líneas de investigación en lo referente escalabilidad de métodos de núcleo, se ha realizado una implementación de SVMs semi-paramétricas y GPs tanto completos como semi-paramétricos. Los resultados obtenidos al utilizar como referencia otras soluciones existentes demuestran la utilidad y eficiencia de estas técnicas.

En segundo lugar y tras analizar las limitaciones del modelo propuesto anteriormente, que hacía uso de una resolución de sistemas lineales paralela utilizando una inversión por bloques, se ha propuesto un nuevo esquema basado en una factorización de Cholesky paralela que mejora la estabilidad numérica y es más eficiente computacionalmente. Con este nuevo esquema se ha realizado una implementación de SVMs tanto completas como semi-paramétricas y los resultados obtenidos han mostrado su superioridad respecto al modelo anterior.

Como añadido, también se han muestran otros aportes que se corresponden a resultados y premios obtenidos en competiciones de aprendizaje automático gracias a los conocimientos sobre paralelización adquiridos con este trabajo.

El trabajo finaliza con una revisión de las aportaciones realizadas y sugerencias de nuevas líneas de investigación abiertas.

Abstract

"*I have only made this letter longer*
because I have not had the time to make it shorter"
Blaise Pascal (1623 -1661)

Kernel methods are very popular techniques in the Machine Learning area because they can produce highly competitive results in many practical tasks. The main reason of this success is their ability of creating non linear solutions very easily, transforming the input data space onto a high dimensional space where inner products between projected vectors can be computed using a kernel function. Among the most extended techniques we have Support Vector Machines (SVMs), that performing very well in classification problems, and Gaussian Processes (GPs), that are widely used in regression problems.

Kernel methods have two main weaknesses that limit their practical application. The first one is the computational cost associated with their training procedure, which is $\mathcal{O}(n^3)$ being n the number of samples in the training set. Due to this scalability problem, the run time obtaining the models is too high when working with large scale data and the field of action of this family of algorithms is reduced. The second problem is associated to their non parametric nature. In kernel methods the complexity of the models is not preset in advance, the machine size depends on the training set size. This ability made them very popular, but the resulting models

are often very slow processing new samples when the training set contains a high number of samples.

In recent years, a number of factors including social networks, electronic commerce, movie devices and collaboration using the new technologies have led to a huge increase in the dataset's size. We have gone from the problem of not having enough data to solve a task to the opposite problem of having so many data that is very costly and complicated to process them.

If, to the fact that currently there are much larger datasets, we add the problem that kernel methods have about computational cost and scalability, we find that the field of action of these techniques has been drastically reduced.

Fortunately, new research lines such as parallel architectures and semi-parametric approximations of the models have also emerged to allow us to face this kind of problems.

On the one hand, the number of processing cores in computers has increased significantly, giving rise to new research lines to adapt classical machine learning techniques to a new parallel scenario in which multiple processing resources work simultaneously following the principle that problems can be divided in smaller subtasks that can be done at the same time.

On the other hand, semi-parametric solutions are approximations that fix the complexity of the resulting models in advance and thanks to them the problem of obtaining a solution too slow to process new samples is avoided. Using these approximations we can also reduce the computational cost and the run time of the training procedure.

This PhD Thesis make use of these research lines and presents new parallel algorithms for multicore and multiprocessor systems of non parametric and semi parametric versions of different kernel methods algorithms.

First of all, we show a new parallel schema that takes advantage of the matrix formulation to solve kernel methods. This schema proposes some low and high level design criteria to achieve an efficient parallelization when implementing the proce-

dures that are carried out to solve this kind of algorithms. With the aim of showing the usefulness of these techniques and to join the different research lines related with the scalability, some algorithms have been implemented: a semi-parametric version of SVMs and two GPs solvers (a semi-parametric one and a full non parametric version). Results obtained when benchmarking these algorithms evidence the usefulness and efficiency of these techniques.

Secondly and after analyzing the main limits of the previous model, that made use of a parallel block matrix inverse to solve linear systems, a new schema based on a parallel Cholesky factorization has been proposed. This new architecture improves the numerical stability and it is more computationally efficient. Using this schema, we have implemented two algorithms, a full and a semi-parametric SVM, and the results show the superiority over the previous model.

In addition, we show some indirect contributions that correspond to results and prizes obtained in machine learning competitions thanks to the knowledge about parallelization arising from this work.

This document concludes describing the main contributions and some suggestions about new research lines that emerge from this work.

Agradecimientos

"*Agradece a la llama su luz,*
pero no olvides el pie del candil
que paciente la sostiene"
Rabindranath Tagore (1861 -1941)

Tras todos estos años de trabajo, es muy difícil condensar en unos pocos párrafos mi agradecimiento a todas las personas que han estado a mi lado. Durante este tiempo he aprendido muchísimo y todo esto ha sido posible gracias a las personas que me han acompañado en esta aventura y me han animado constantemente. Por eso, desde aquí voy a tratar de transmitir mi agradecimiento.

En primer lugar a mi director de Tesis, el Dr. Ángel Navia Vázquez por brindarme la oportunidad de empezar a trabajar en un mundo que era desconocido para mí, la investigación. Un mundo fascinante y que me ha permitido adquirir un gran conocimiento de un área muy interesante, la del aprendizaje automático, una campo que llamó mi atención de inmediato por parecer algo de "ciencia ficción", después me trajo muchos quebraderos de cabeza por el gran trabajo que supone ponerse al día en algo que avanza tan rápido y finalmente me ha dado grandes satisfacciones cuando he sido capaz de utilizarla correctamente.

La Universidad Carlos III como institución y en concreto el departamento de Teoría de la Señal y Comunicaciones también se merecen una mención especial. En

esta universidad he pasado muchos años de mi vida, en primer lugar formándome como Ingeniero de Telecomunicación y volví unos años después ávido de más conocimiento con la intención de seguir formándome.

Como no podría ser de otra manera, también se merecen un especial agradecimiento mis compañeros de los laboratorios 4.2C01 y 42C03, con los que he pasado numerosos días intercambiando experiencias. Con los que comencé inicialmente Adil Omari, Efraín Mayhua, Luis Muñoz, Marilea Vilela, Sergio Muñoz y Soufiane El Jelali con los que surgió la gran tradición de desayunar una "barriga con tomate" y los que llegaron después Fernando Rabanal, Juan José Choquehuanca, Lorena Álvarez, Oscar García-Hinde y Rafael Hernández que desgraciadamente sustituyeron esa tradición milenaria por una "Nespresso" que permitía hacer más eficiente el desayuno al reducir costes y no tener que desplazarnos.

También tengo mucho que agradecer al resto de amigos fuera del mundo académico, ya que me hacían desconectar y aprender cosas sobre otros temas tan dispares como pueden ser micología, política, cine o videojuegos. Aquí tengo que hacer varias menciones especiales. En primer lugar a los amigos del colegio que llevan conmigo tantísimos años, en especial a Carlos, David y Luis, con los que aún me reúno semanalmente desde hace tanto tiempo pese a que recientemente hayan cerrado nuestro punto de encuentro (el bar Sahara), y en segundo lugar también tengo mucho que agradecer a los amigos de la carrera Bea, Jose, Juanjo, Julio, algunos de los cuales también se han embarcado en un doctorado y con los que tengo numerosas conversaciones sobre cómo solucionar los problemas del mundo. Y en último lugar (aunque no por ello son menos importantes) están los amigos del pueblo, con los que muy a mi pesar sólo hablo de política. Y como no a mis padres, Arturo y Margarita, y mi hermano Oscar que han sido como de costumbre las personas con las que más he podido contar ya que al ser mi familia no les ha quedado más remedio que darme su apoyo incondicional en la vida por erráticos que fuesen mis pasos. Siempre recordaré el orgullo que se veía reflejado en los de mi padre con cada logro que he tenido.

También al resto de mi familia, que al ser tan extensa en cuanto a tíos y primos no

puedo nombrarlos individualmente, pero haré una excepción con mi abuela Demetria que se ha encargado de que esté tan bien alimentado por miedo a que me quedase lo que sea que ella entienda por "Tísico".

A Eva, que empezó esta aventura siendo mi novia y por algún motivo que aún desconozco quiso terminarla siendo mi esposa. Muchas gracias por haber estado siempre ahí, haber renunciado inmerecidamente a algunas cosas y haber sido "el socio capitalista" de la relación por lo paupérrimas que son las becas de investigación en este país.

De forma intencionada en dejado en último lugar a la persona más especial de todas, mi hija Lucía, que nació durante el transcurso de este trabajo y cuya existencia ha hecho que le encuentre un nuevo significado a la vida. A ti, Lucía, te dedico este trabajo ya que tuviste que soportar largas horas sin la compañía de tu padre, sin poder entender a tu corta edad el por qué prefería estar delante del ordenador y no jugando contigo. Pese a todo, cada sonrisa tuya siempre me ha llenado de ánimo y fuerzas.

A todos vosotros, muchas gracias.

Índice general

Índice de figuras

Índice de tablas

Parte I:

Estado del Arte

Capítulo 1

Introducción

"*Olvida que has dado*
para recordar lo recibido"
Mariano Aguiló (1825-1897)

Este capítulo repasa las nociones que son necesarias para entender el contexto en el que se ha desarrollado esta tesis. Su principal objetivo es que el lector pueda localizar las líneas de investigación actuales que están relacionadas con el trabajo realizado.

En primer lugar se da una visión general de la teoría de estimación y decisión para después hacer un repaso de los conceptos básicos de aprendizaje máquina, centrándose en aquellas técnicas que motivan estas contribuciones. A continuación, se revisan los procedimientos para realizar procesado algorítmico en paralelo, haciendo hincapié en los sistemas multiprocesador y multinúcleo.

Tras esta revisión, se hace un análisis de la motivación y objetivos perseguidos y se justifica la dirección de los trabajos realizados. Para finalizar, se hace una descripción del contenido por capítulos.

1.1. El problema de la toma de decisión

El problema de decisión se presenta cuando ante la observación de un dato o instancia $\mathbf{x} = \left[x_1, \ldots, x_p\right]^\top$, que a su vez se compone de un conjunto de p atributos, se tiene que realizar una elección entre un conjunto de posibles hipótesis, $\Omega = \{\omega_1, \ldots, \omega_J\}$ debido a que dicho dato contiene información relacionada con ellas.

Este tipo de problemas responden a la demanda de poder asignar datos a una determinada categoría. Por este motivo, cuando dichas hipótesis se corresponden con la pertenencia de una instancia a diferentes clases, el problema se llama clasificación y se aborda utilizando una función capaz de determinar la clase a la vista de una instancia no conocida.

Si se tiene un conocimiento estadístico del problema se puede recurrir a métodos analíticos para resolverlo [Poor, 2013]. La forma más extendida de abordar el problema es modelar el coste medio que acarrea cada una de las posibles decisiones. Para ello se considera un coste c_{ij} asociado a asignar la muestra a la clase ω_i cuando pertenece a la clase ω_j y $P(\omega_j|\mathbf{x})$ es la probabilidad de que la muestra pertenezca a la clase ω_j y se obtiene el coste asociado a aceptar la hipótesis ω_i [Duda et al., 2012]:

$$C(\omega_i|\mathbf{x}) = \sum_{j=1}^{J} c_{ij} P(\omega_j|\mathbf{x}) \tag{1.1}$$

A la hora de tomar la decisión, se elegirá la clase que tenga un menor coste asociado:

$$i = \arg\min_i C(\omega_i|\mathbf{x}) \tag{1.2}$$

Si asumimos un coste unitario en las decisiones incorrectas y no penalizamos las correctas nos encontramos ante lo que se conoce como criterio MAP (Máximo a Posteriori) y el problema se reduce a la selección de la hipótesis más probable:

$$i = \arg\min_i P(\omega_i|\mathbf{x}) \tag{1.3}$$

En la práctica, en la mayoría de situaciones reales, se desconocen las probabilidades a posteriori $\{P(\omega_i|\mathbf{x})\}_{i=1}^{J}$ al no disponer de un conocimiento lo suficientemente avanzado del dominio del problema [Bishop, 2006].

Una de las posibles soluciones consiste en recurrir al teorema de Bayes para modelar la probabilidad a posteriori a partir de la verosimilitud $p(\mathbf{x}|\omega_j)$ y la probabilidad a priori de cada clase $P(\omega_j)$:

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{\sum_{j=1}^{J} p(\mathbf{x}|\omega_j)P(\omega_j)} \tag{1.4}$$

Mediante esta regla, a medida que se tienen observaciones de una variable aleatoria, la verosimilitud va transformando la probabilidad a priori en la probabilidad a posteriori.

Cuando además de desconocer la probabilidad a posteriori también se desconoce la verosimilitud, se puede recurrir a métodos de estimación de funciones de densidad de probabilidad para estimar la verosimilitud a partir de muestras disponibles. Ejemplos de métodos de estimación son los Modelos de Mezcla de Gaussianas (GMMs, "Gaussian Mixture Models") [Dempster et al., 1977] o las ventanas de Parzen.

Respecto a la probabilidad a priori $P(\omega_i)$, puede conocerse de antemano utilizando un conocimiento del dominio del problema estudiado o se puede aproximar utilizando la frecuencia relativa observada en el conjunto de datos de entrenamiento [MacKay, 2003] [Bishop, 2006].

Una alternativa a la utilización de funciones de densidad de probabilidad es la utilización de aprendizaje máquina. Este tipo de métodos utilizan un conjunto de observaciones de datos para crear un clasificador mediante un proceso de aprendizaje inductivo.

1.2. Aprendizaje Máquina

En las situaciones en las que no se dispone de un conocimiento estadístico de la tarea a resolver, todo el conocimiento e información necesarios para realizar el

proceso de clasificación debe extraerse a partir de una colección de datos representativos del problema. Es decir, se dispone de un conjunto de muestras etiquetadas, $\{x_l, y_l\}_{l=1}^{n}$, que han sido generadas a partir de una distribución desconocida y de manera independiente.

Con este conjunto de datos de entrenamiento, se diseña un clasificador que implementa, para cada una de las clases, una función discriminante $f_i(x)$.

El clasificador selecciona al final la clase correspondiente a la función discriminante con mayor valor. De tal forma que si tenemos un conjunto de funciones $f_i(x), i = 1, \ldots, J$, el clasificador asignará la muestra $\mathbf{x}$ a la clase j si:

$$f_j(\mathbf{x}) > f_i(\mathbf{x}), \forall i \neq j \tag{1.5}$$

Este criterio de decisión es el equivalente al utilizado en los métodos estadísticos en la fórmula 1.3 donde la función discriminante era la probabilidad a posteriori.

La primera decisión a la hora de abordar un problema de aprendizaje máquina es la elección del tipo de arquitectura. No es una elección sencilla ya que influyen muchas variables como la dificultad de la tarea a resolver y el número de observaciones disponibles. La importancia de esta elección reside en que marcará el tipo de regiones de clasificación que el modelo podrá crear. Los modelos más extendidos de función discriminante son:

- Modelos Lineales: Este tipo de clasificadores utiliza como función discriminante una combinación lineal de los atributos de los datos de entrada:

$$f_j(\mathbf{x}) = \sum_{i=1}^{p} (\beta_{ji} x_i) \tag{1.6}$$

  En un problema binario, donde los datos pertenecen a una de dos posibles categorías, podemos visualizar el problema de clasificación como la separación del espacio de entrada por un hiperplano. Cada una de las dos regiones en las que queda dividido el espacio corresponde a una de las clases.

Este tipo de clasificadores se suelen usar cuando la velocidad de clasificación es un factor importante o cuando el conjunto de atributos de cada muestra es grande como en el caso de clasificación de documentos.

- Redes Neuronales: Este tipo de redes intentan modelar niveles de abstracción más altos de los datos utilizando unidades de procesamiento no lineales, conocidas como neuronas, que se organizan en capas [Bishop, 1995] [Haykin and Network, 2004].

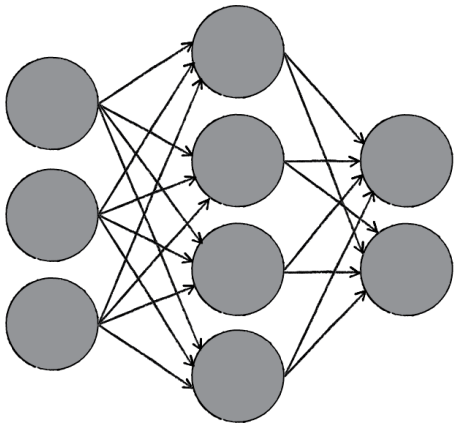


Figura 1.1: Estructura de Red Neuronal

Este tipo de arquitectura tenía dos problemas, el primero era la capacidad de generalización, ya que debido al elevado número de parámetros libres (un peso asociado a cada conexión entre neuronas) cuando se tienen varias capas ocultas es muy sencillo clasificar correctamente los datos de entrenamiento utilizando únicamente un reducido porcentaje de esos parámetros lo que hace que sea muy sencillo realizar un sobreajuste en la fase de entrenamiento. El segundo era que el algoritmo de entrenamiento clásico, conocido como propagación hacia atrás ("backpropagation"), se va diluyendo de forma exponencial a medida que atraviesa capas en su camino a la entrada de la red. En una red con numerosas capas casi todo el peso de la clasificación se realiza en las últimas capas mientras que las primeras apenas sufren cambios.

Estos factores han hecho que en el pasado solo se trabajase con una capa de entrada, una capa intermedia (conocida como capa oculta), y una capa de salida. El Aprendizaje Profundo (DL, "Deep Learning") es un conjunto de técnicas surgidas para poder trabajar con varias capas ocultas. Algunas de las técnicas más conocidas son los Autocodificadores Eliminadores de Ruido Apilados (SDAs, "Stacked Denoising Autoencoders"), que inicializan la red pre-entrenando capas que tratan de recuperar a la salida el mismo dato de entrada al que previamente se le ha añadido un ruido [Bengio et al., 2007] [Vincent et al., 2010], las Redes de Evidencia Profundas (DBNs, "Deep Belief Networks"), que pre-inicializan cada capa utilizando maquinas de Boltzman restringidas [Hinton et al., 2006], o técnicas de eliminación de conexiones en cada iteración del algoritmo para evitar el sobreajuste [Srivastava et al., 2014].

- Métodos de núcleo: Los métodos de núcleo deben su nombre a la utilización de funciones de núcleo para modelar la no linealidad de los problemas. Con ellas pueden operar en un espacio de alta dimensión computando productos escalares entre pares de datos en lugar de las coordenadas. Al ser este procesado computacionalmente más barato que el computo de las coordenadas se le conoce como 'el truco del núcleo". Se podría hablar de este tipo de métodos como aprendizaje basado en instancias, en lugar de fijar parámetros en los distintos atributos recuerdan un subconjunto de instancias del conjunto de entrenamiento y realizan la clasificación utilizando funciones que obtienen información de similitud con las observaciones recordadas y aprenden un conjunto de pesos diferentes para cada una de estas medidas.

  Los algoritmos más conocidos de esta familia son las Máquinas de Vectores Soporte (SVMs,"Support Vector Machines") [Vapnik, 2013], utilizadas muy comúnmente en problemas de clasificación y los procesos Gaussianos (GPs, "Gaussian Processes") [Rasmussen, 2006] que son métodos Bayesianos utilizados en problemas de regresión.

- Basados en arboles: Compuestos por una o varias arquitecturas en forma de árbol en los que los datos recorren la estructura decidiendo en cada nodo la rama por la que desplazarse en función del valor de sus atributos. Ejemplos de este tipo de algoritmos son los arboles de decisión, en los que se crea un árbol a partir de los datos de entrenamiento eligiendo en cada nodo el atributo y umbral que mejor separe las diferentes clases, o algoritmos basados en combinaciones de dichos árboles, donde se generan diferentes arboles creando diferentes conjuntos de entrenamiento tomando muestras del original y tomando en cuenta solo un subconjunto de atributos en cada nodo para crear arboles con tipologías diferentes para después fusionar sus resultados.

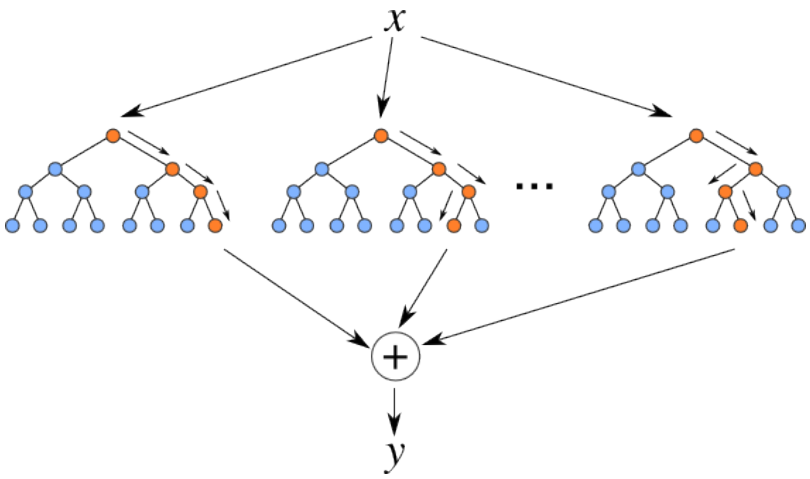


Figura 1.2: Arquitectura basada en árboles

Los algoritmos más extendidos de este tipo son los bosque aleatorios [Breiman, 2001] o los arboles combinados con potenciación por gradiente [Friedman, 2001] (GBTs, "Gradient Boosted Trees").

Estos algoritmos han demostrado prestaciones sorprendentes para resolver problemas no lineales con conjuntos de datos de gran envergadura ya que por un lado pueden modelar la no linealidad de un problema de forma sencilla y rápida y por otro con gran certeza si disponen de un conjunto de datos de entrenamiento lo suficientemente grande.

Para obtener el valor de los parámetros internos de estas funciones o estructuras se utiliza una función de coste [Bishop, 2006] que tiene típicamente una componente asociada a minimizar el error de clasificación y otra componente regularizadora para obtener buenas propiedades de generalización que permita al modelo funcionar correctamente con datos que no han sido utilizados durante el entrenamiento. Mediante esta función de coste la tarea de obtención de parámetros internos se reduce a un problema de optimización.

Existen numerosas alternativas en la componente asociada a minimizar el error de clasificación, entre ellas, las más extendidas son:

- Error Cuadrático Medio (MSE, "Mean Squared Error"): Consiste en la media de los errores al cuadrado obtenidos por el modelo sobre los datos de entrenamiento, esta medida es el segundo momento (sobre el origen) del error. Si y_i es la variable a estimar de una muestra (en el caso de un problema de clasificación binario podría tomar valores $+1$ y -1) y $\hat{y}_i$ la salida del modelo, entonces la función de coste tiene la siguiente forma:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1.7}$$

- Pérdida Logarítmica (LL,"Log Loss"): Este tipo de error modela la probabilidad de pertenencia a una clase en un problema de clasificación binario. Para ello se normaliza la salida con una función sigmoide que garantice el estar acotada entre 0 y 1 y se minimiza el la siguiente función:

$$LL = \frac{1}{n}\sum_{i=1}^{n} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i) \tag{1.8}$$

- Pérdida de Bisagra (HL,"Hinge Loss"): Utilizada en clasificadores de máximo margen, donde las etiquetas de un problema de clasificación binario toman los valores $+1$ y -1 y el error empieza a tenerse en cuenta en el margen y no en la frontera de clasificación. La expresión analítica es la siguiente:

$$HL = \frac{1}{n}\sum_{i=1}^{n} \text{máx}(0, (1 - y_i \hat{y}_i)) \tag{1.9}$$

En la figura 1.3, en verde aparece el error de clasificación y en azul la HL, nótese que es una cota superior y de forma convexa (aunque no diferenciable).

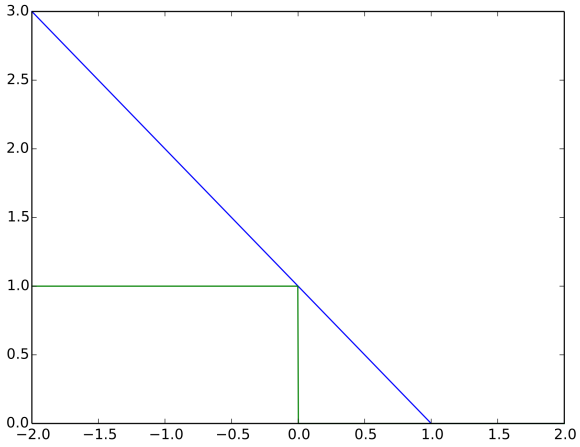


Figura 1.3: Función de bisagra

En cuanto a la componente de la función de coste asociada a la regularización [Bishop, 2006], se encarga de solucionar los problemas asociados al mal condicionamiento de la formulación y ayuda a evitar el problema del sobreajuste haciendo que los clasificadores puedan generalizar a nuevas muestras no vistas durante la fase de entrenamiento.

La forma más extendida de regularización es incluir en la función de coste una componente relacionada con la norma. Por ello, siendo $\phi = [\phi_1, \phi_2, \ldots, \phi_s]$ el vector de parámetros libres del algoritmo, la norma l_p tiene la siguiente expresión:

$$||\phi||_p = \sqrt[p]{\phi_1^p + \phi_2^p + \ldots + \phi_s^p} \tag{1.10}$$

Cada tipo de norma tiene diferentes propiedades y tiene diferente efecto sobre la solución, las más utilizadas son:

- Norma l_1: Incorporar el valor absoluto del valor de los parámetros internos a la función de coste. Tiene la propiedad de crear valores dispersos (intenta dar valor 0 a un gran número de coeficientes o parámetros) lo que permite utilizarla como un método de selección de características.

- Norma l_2: Se corresponde con la norma euclídea. La ventaja respecto a la norma l_1 es que tiene solución única y puede ser calculada eficientemente.

- Red Elástica: Consiste en una combinación lineal de las dos normas anteriores, asignando un peso diferente a cada una de ellas.

1.3. Obtención de la Complejidad del Modelo

Atendiendo a la forma de obtener la complejidad de los modelos se puede hablar de tres tipos diferentes de algoritmos.

- Métodos Paramétricos: Este tipo de modelos tienen un conjunto fijo de parámetros. Dentro de esta rama podemos encontrar métodos estadísticos como los modelos analíticos donde se asumen funciones de densidad y se estiman sus parámetros a partir de las observaciones o algoritmos de aprendizaje máquina como los clasificadores lineales donde hay un parámetro por cada atributo del conjunto de entrenamiento.

- Métodos No Paramétricos: Cuando la complejidad de las funciones que puede aprender crece según crece la cantidad de datos de entrenamiento. En este tipo de algoritmos no se asume un modelo en particular, si no que se considera un modelo capaz de aproximar cualquier tipo de función. En esta familia encontramos modelos bayesianos como los Procesos Gaussianos (GPs, "Gaussian Processes") [Rasmussen, 2006] o algoritmos de aprendizaje máquina como las SVMs [Cortes and Vapnik, 1995].

- Métodos Semi-Paramétricos: Emplean modelos flexibles. Se pueden obtener buenas aproximaciones utilizando pocos parámetros. Un ejemplo son los GMMs, cuyos parámetros se pueden obtener utilizando el algoritmo de optimización Esperanza-Maximización (EM, "Expectation-Maximization") [Dempster et al., 1977].

1.4. Métodos de Núcleo

1.4.1. Máquinas de Vectores Soporte

Las SVMs [Vapnik, 2013] abordan problemas de clasificación binarios donde se tiene un conjunto de entrenamiento $\mathcal{D}$ que contiene n muestras en un espacio de p dimensiones de la forma:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^{n} \tag{1.11}$$

donde y_i indica la clase de la muestra $\mathbf{x}_i$. El problema de optimización para encontrar el máximo margen en un caso separable (ver figura 1.4) viene dado por [Vapnik, 1963]:

$$\begin{aligned} &Minimiza \ \frac{1}{2}\|\mathbf{w}\|^2 \\ &Sujeto\, a \ y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1, i = 1, 2, ..., n \end{aligned} \tag{1.12}$$

El hiperplano que separa ambas clases es $\mathbf{w}\mathbf{x} + b = 0$ y los hiperplanos paralelos con máximo margen son $\mathbf{w}\mathbf{x} + b = \pm 1$ respectivamente.

La mayoría de problemas en el mundo real no son linealmente separables, la clasificación de margen blando [Cortes and Vapnik, 1995] usa una función de bisagra ("hinge loss") que separa los datos de entrenamiento permitiendo que algunas muestra caigan dentro del margen o queden mal clasificadas. Formalmente, al añadir estas

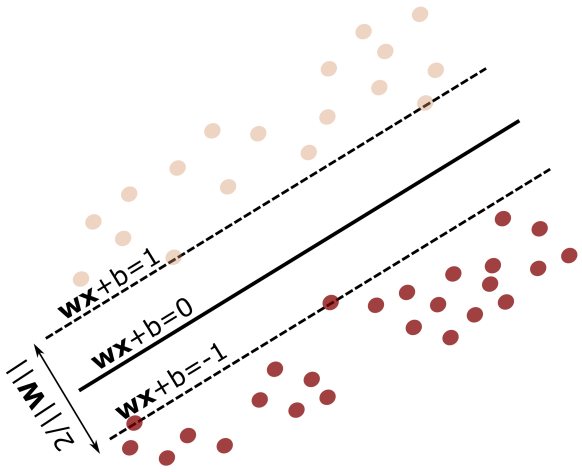


Figura 1.4: Máximo Margen

restricciones al problema de optimización obtenemos la siguiente expresión:

$$\begin{aligned} Minimiza\ & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ Sujeto\ a\ & y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, 2, ..., n \\ & \xi_i \geq 0, i = 1, 2, ..., n \end{aligned} \tag{1.13}$$

donde C es un parámetro con valor positivo que controla el compromiso de error de clasificación (error HL) y generalización (regularización norma L2). Este es un problema de optimización convexo, lo que garantiza una solución única.

El problema puede ser reescrito como una formula de Lagrange (ver Apéndice A), obteniendo así las correspondientes condiciones de Karush-Kuhn-Tucker (KKT). A esta formulación se le denomina formula dual y puede ser resuelta mediante programación cuadrática (ver Apéndice A.3):

$$\max_{\boldsymbol{\alpha}} F(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\mathbf{x}_j \tag{1.14}$$

$$Sujeto\ a\ \sum_{j=1}^{n}\alpha_i y_i = 0\ \ y\ \ 0 \leq \alpha_i \leq C \tag{1.15}$$

donde α_i es el multiplicador de Lagrange asociado a la muestra de entrenamiento $\mathbf{x}_i$. El valor de estos multiplicadores es 0 para puntos correctamente clasificados fuera del margen ya que la frontera de decisión solo se ve afectada por muestras dentro de él. Los puntos cuyo multiplicador de Lagrange asociado son mayores que 0 se llaman Vectores Soporte.

Atendiendo al valor de α_i podemos hablar de tres tipos de datos:

- Si $\alpha_i = 0$ entonces la muestra $\mathbf{x}_i$ se encuentra bien clasificada y fuera del margen.
- Si $0 < \alpha_i < C$ la muestra $\mathbf{x}_i$ está ubicada dentro del margen y puede estar bien o mal clasificada. Son vectores soporte no delimitadores.
- Si $\alpha_i = C$ la muestra $\mathbf{x}_i$ se encuentra bien clasificada y situada exactamente en el margen. En este caso hablamos de vectores soporte delimitadores.

El proceso más común de crear un clasificador no lineal es aplicando el truco del núcleo [Boser et al., 1992], que mapea el espacio de entrada en un espacio de mayor dimensión donde los productos vectoriales se computan mediante una función de núcleo. Ahora podemos reemplazar el producto de la ecuación 1.14 por dicha función.

$$\max_{\boldsymbol{\alpha}} F(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{1.16}$$

Este problema de optimización, donde $K(\mathbf{x}_i, \mathbf{x}_j)$ es la función de núcleo, es equivalente a resolver un problema de programación cuadrática n-dimensional.

Existe una gran cantidad de funciones de núcleo, las más comunes son:

- Polinómico: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \mathbf{x}_j + 1)^p$
- Base radial (Gaussiano): $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2}$
- Tangente Hyperbólica: $K(\mathbf{x}_i, \mathbf{x}_j) = tanh(k\mathbf{x}_i \mathbf{x}_j + c)$

Utilizando una función de núcleo, el clasificador SVM no lineal toma la forma:

$$f(\mathbf{x}) = \sum_{j=1}^{n} y_j \alpha_j K(\mathbf{x}, \mathbf{x}_j) + b \tag{1.17}$$

1.4.2. Procesos Gaussianos

De acuerdo a [Rasmussen, 2006], se define un GP como un conjunto de variables aleatorias que tienen una distribución conjunta Gaussiana. Dicho GP viene definido por función de media y su función de covarianza (que se define mediante una función de núcleo):

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{1.18}$$

$$K(\mathbf{x}_1, \mathbf{x}_2) = \mathbb{E}[(f(\mathbf{x}_1) - m(\mathbf{x}_1))(f(\mathbf{x}_2) - m(\mathbf{x}_2))] = cov(f(\mathbf{x}_1), f(\mathbf{x}_2)) \tag{1.19}$$

El GP suele expresarse de la siguiente forma:

$$\mathcal{GP}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}')) \tag{1.20}$$

Regresión mediante Procesos Gaussianos

En problemas de regresión, las etiquetas toman valores reales y se asume que hay un modelo que relaciona los datos con las etiquetas:

$$y_i = f(\mathbf{x}_i) + \xi_i \tag{1.21}$$

En este problema $f(.)$ es la función que se debe obtener y asumimos que los valores observados difieren de la función debido a la existencia de un ruido aditivo y asumimos que tiene la forma de una distribución Gaussiana de media nula y varianza σ_n^2:

$$\xi_i \sim \mathcal{N}(0, \sigma_n^2) \tag{1.22}$$

La regresión mediante GPs es un enfoque Bayesiano que asume un GP a priori sobre funciones:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{0}, \mathbf{K}) \tag{1.23}$$

donde $\mathbf{f}$ es un vector de valores de funciones latentes, $f_i = f(\mathbf{x}_i)$, donde cada valor está indexado a una de las entradas, $\mathbf{X}$ el conjunto de datos de entrada $[\mathbf{x}_1, \ldots, \mathbf{x}_n]$ y $\mathbf{K}$ es una matriz de covarianza $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.

De 1.21 y 1.23 podemos dar la expresión de la probabilidad marginal de la salida:

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_n^2 I) \tag{1.24}$$

donde $\mathbf{y} = [y_1, \ldots, y_n]$ e I es la matriz identidad. Si queremos obtener la distribución a posteriori de $\mathbf{f}$ podemos aplicar el teorema de Bayes:

$$p(\mathbf{f}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} \tag{1.25}$$

Con esta definición, se puede modelar la evidencia de verosimilitud marginal integrando sobre $\mathbf{f}$ el producto de la probabilidad a priori y la verosimilitud:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) &= \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{X})d\mathbf{f} \\ &\sim \mathcal{N}(\mathbf{0}|\mathbf{K} + \sigma_n^2 I) \end{aligned} \tag{1.26}$$

Al ser tanto la probabilidad a priori, la verosimilitud y la evidencia Gaussianas, la probabilidad a posteriori es también Gaussiana y tiene la siguiente expresión (ver Apéndice B):

$$p(\mathbf{f}|\mathbf{y}) \sim \mathcal{N}(\mathbf{K}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, (\mathbf{K}^{-1} + \sigma_n^{-2} I)^{-1}) \tag{1.27}$$

Para realizar predicciones sobre una nueva muestra no vista en el entrenamiento $\mathbf{x}_i$, la distribución condicionada $f(\mathbf{x}_i)$ es entonces:

$$p(f(\mathbf{x}_i)|\mathbf{f}, \mathbf{x}_i, \mathbf{X}) \sim \mathcal{N}(\mathbf{k}_i^T\mathbf{K}^{-1}\mathbf{f}, K(\mathbf{x}_i, \mathbf{x}_i) - \mathbf{k}_i^T\mathbf{K}^{-1}\mathbf{k}_i) \tag{1.28}$$

Donde $\mathbf{k}_i$ contiene la covarianza de $\mathbf{x}_i$ con los elementos del conjunto de entrenamiento. Integrando los valores de la función (fórmula 1.27) sobre la probabilidad a

posteriori (fórmula 1.28), se obtiene la distribución de $f(\mathbf{x}_i)$.

$$\begin{aligned} & p(f(\mathbf{x}_i)|\mathbf{y}, \mathbf{x}_i, \mathbf{X}, \sigma_n^2) \\ & \sim \mathcal{N}(\mathbf{k}_i^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, K(\mathbf{x}_i\mathbf{x}_i) - \mathbf{k}_i^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_i) \end{aligned} \tag{1.29}$$

Como se puede observar, una de las principales ventajas de los GPs es que la estimación de la varianza en la predicción de una nueva muestra aparece en el modelo.

También se puede observar la relación entre la formulación de la predicción de la media y la ecuación del predictor de Wiener [Wiener, 1949].

1.4.3. Modelos semi-paramétricos

Tanto las SVMs como los GPs son métodos no paramétricos; en este tipo de métodos la complejidad del modelo se ajusta automáticamente, pudiendo dar lugar a modelos excesivamente grandes y lentos. De la necesidad de lidiar con este problema han surgido versiones semi-paramétricas que fijan de antemano la complejidad.

En relación a las SVMs, algunos trabajos [Schölkopf et al., 1997] [Osuna and Girosi, 1998] [Schölkopf et al., 1998] [Smola and Schölkopf, 2000] calculan primero la solución y después realizan una reducción del tamaño del clasificador resolviendo un problema de pre-imagen [Kwok and Tsang, 2004]. En [Navia-Vázquez, 2007] [Parrado-Hernández et al., 2003], para evitar el cálculo inicial de la SVM, proponen una arquitectura creciente utilizando la Aproximación Matricial Codiciosa Dispersa (SGMA, "Sparse Greedy Matrix Approximation") para seleccionar iterativamente los candidatos para hacer crecer un modelo semi-paramétrico. Otras alternativas [Peng et al., 2013] proponen métodos de selección hacia adelante y hacia atrás para llegar a un conjunto reducido de vectores soporte.

En cuanto a los GPs, se han propuesto muchos esquemas para obtener modelos reducidos. Los métodos [Csató and Opper, 2002] y [Lawrence et al., 2003] están basados en minimizar la divergencia de Kullback-Leibler, [Smola and Bartlett, 2001] usa un criterio MAP para seleccionar en cada iteración el candidato para hacer crecer

el modelo y [Quinonero-Candela and Rasmussen, 2005] selecciona en cada iteración el elemento que maximiza la evidencia para evitar problemas de sobreajuste.

1.5. Volumen de Datos y Escalabilidad

En los últimos años, factores como la evolución de la tecnología, el uso de redes sociales, nuevos dispositivos con acceso a internet o el comercio electrónico han dado lugar a un gran incremento en el volumen de las bases de datos. Si anteriormente era muy común el problema de no tener un conjunto de datos lo suficientemente grande para ser representativo del problema, actualmente se ha pasado al problema opuesto de tener tal cantidad de datos que resulta muy costoso procesarlos.

Si a este hecho le sumamos el problema derivado de la falta de escalabilidad de los métodos de núcleo nos encontramos con que este tipo de técnicas han visto su campo de acción muy reducido.

1.5.1. Paralelización

Una de las principales técnicas para abordar el alto coste computacional de algunos algoritmos y tratar de hacerlos más escalables es la paralelización. Mediante la computación paralela se realizan muchos cálculos de forma simultánea [Almasi George and Allan, 1989] operando bajo el principio de que los grandes problemas puedan dividirse en subproblemas más pequeños que a menudo pueden resolverse al mismo tiempo.

En un entorno óptimo, la aceleración obtenida con procesamiento en paralelo sería lineal (doblando el número de elementos de procesamiento deberíamos reducir a la mitad el tiempo de ejecución). En entornos reales no es posible alcanzar este óptimo, aunque es posible para muchos algoritmos alcanzar una aceleración cercana a la lineal para un número acotado de núcleos de procesamiento.

La cota máxima de aceleración de un algoritmo en una plataforma de procesa-

miento paralelo viene dada por la ley de Amdahl [Amdahl, 1967]:

$$S = \frac{1}{1 - p_p + \frac{p_p}{N_p}} \tag{1.30}$$

Donde p_p es el ratio del tiempo de ejecución que es paralelizable y N_p el número de procesadores.

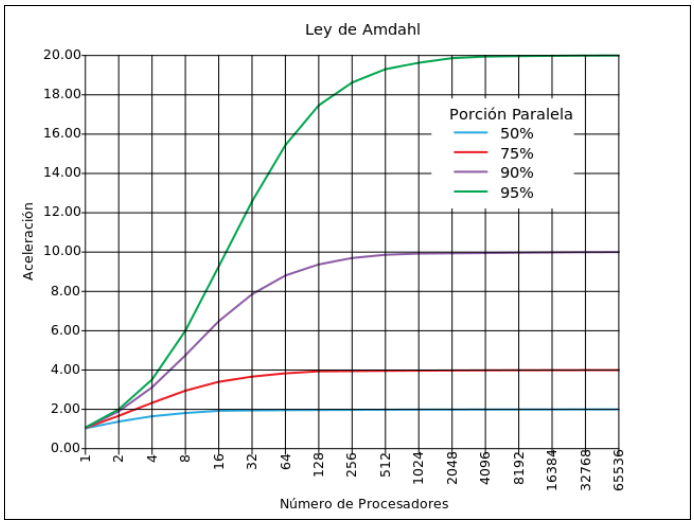


Figura 1.5: Ley de Amdahl

1.5.2. Tecnologías de Paralelización

Actualmente hay tres líneas principales de investigación: computación distribuida, multinúcleo y mediante unidades de procesamiento gráfico, todas ellas basadas en la división en subtareas pero en entornos diferentes.

Computación distribuida

En este tipo de entornos los sistemas trabajan de forma independiente debido a que la memoria está distribuida. Estos sistemas comunican resultados intermedios periódicamente mediante la utilización de un intercambio de mensajes [Andrews, 1999] ya que sus elementos están conectados por una red.

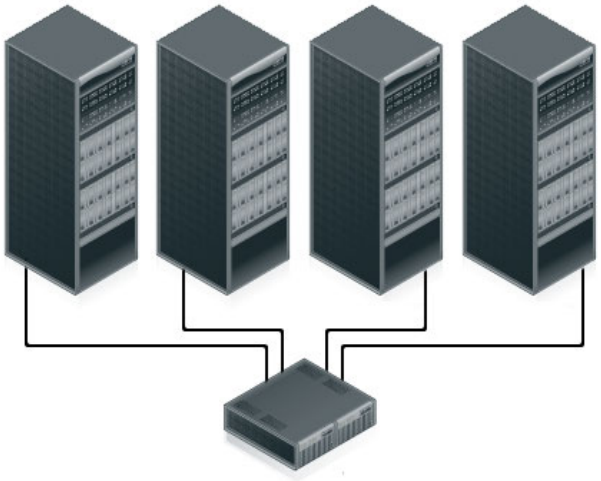


Figura 1.6: Computación Distribuida

Debido a la latencia de la red este tipo de arquitecturas están más enfocadas a trabajar con grandes volúmenes de datos donde una única máquina no puede almacenar la totalidad de los datos o a problemas que requieran gran tiempo de procesado y poca comunicación entre los diferentes nodos de la red.

Computación Multinúcleo y Multiprocesador

La industria de los semiconductores ha creado nuevos diseños de procesadores que incrementan el rendimiento mediante la inclusión de varios núcleos de computación en el mismo procesador.

Para poder aprovechar correctamente su potencial, han surgido interfaces de programación [Chu et al., 2007] [Dagum and Enon, 1998] que permiten desarrollar software utilizando múltiples núcleos en un entorno de memoria compartida. La principal ventaja de este entorno radica en que la comunicación entre los núcleos es extremadamente rápida al compartir todos ellos la memoria de acceso aleatorio (RAM, "Random Access Memory"). En la figura 1.7 podemos observar cómo la memoria en este tipo de arquitectura se organiza por niveles (L1, L2, ...) teniendo como base común la RAM para intercambiar datos.

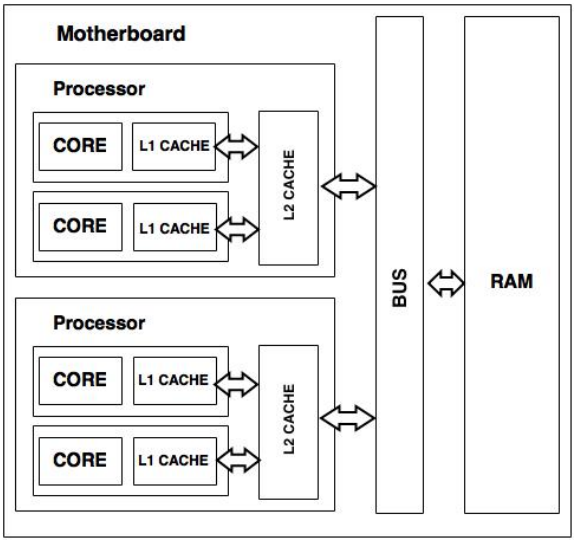


Figura 1.7: Computación Multicore

Por un lado la ventaja de estos sistemas radica en que cada núcleo puede realizar tareas totalmente independientes y la comunicación es muy rápida si la comparamos con alternativas como la computación distribuida, pero por otro lado el número de núcleos de computación es limitado por la capacidad física de los procesadores. El mejor escenario para este tipo de arquitectura [Hill and Marty, 2008] se da cuando los diferentes núcleos realizan tareas diferentes al mismo tiempo y su comunicación requiere el intercambio de grandes cantidades de datos.

Computación con Unidades de Procesamiento Gráfico

Las unidades de procesamiento gráfico (GPUs, "Graphical Processing Units") [Owens et al., 2008] [Sanders and Kandrot, 2010] son dispositivos con un gran número de núcleos de capacidad reducida. Las GPUs están diseñadas para que todos los núcleos que contienen ejecuten la misma instrucción y al mismo tiempo sobre posiciones de memoria diferentes. Este tipo de arquitectura permite resolver de forma muy eficiente algunas tareas como productos o sumas de matrices donde todas las diferentes unidades de procesamiento realizan la misma operación en sobre diferentes posiciones de la matriz (por ejemplo, cada núcleo puede operar sobre un vector fila

de la matriz).

Han tenido mucho éxito en los últimos años debido al auge de las redes neuronales profundas ya que el entrenamiento de estas tiene un coste computacional elevado y el algoritmo de propagación hacia atrás, que es el más extendido para entrenar una red neuronal, se resuelve mediante operaciones de productos y sumas matriciales.

Esta arquitectura está enfocada a problemas donde las subtareas en las que se divide el problema a resolver son exactamente iguales.

1.5.3. Paralelización en el ámbito de los métodos de núcleo

En los primeros trabajos de paralelización de las SVMs, se proponía dividir el conjunto de entrenamiento y entrenar diferentes modelos para combinar al final el resultado utilizando una Red Neuronal [Collobert et al., 2002] o entrenar una nueva SVM utilizando los vectores soporte obtenidos [Dong et al., 2003]. Otros trabajos proponen la utilización de una SVM en cascada [Zhang, 2012]. Este tipo de soluciones son subóptimas al no dividirse linealmente el tiempo de ejecución al paralelizar las tareas.

También se han propuesto métodos más avanzados, como PSVM [Zhu et al., 2008] que utiliza una implementación paralela del método de punto interior (IPM,"Interior Point Method") y aproxima la solución de un sistema lineal mediante una factorización de Cholesky Incompleta y Parallel SMO [Cao et al., 2006] [Herrero-Lopez et al., 2010] que utiliza una versión paralela del algoritmo de optimización mínima secuencial (SMO, "Sequential Minimal Optimization") [Platt et al., 1999]. Existen también versiones para GPU [Cotter et al., 2011] y tras la llegada de las tecnologías para grandes volúmenes de datos (Big Data) también han surgido versiones distribuidas [You et al., 2014].

En GPs, han surgido versiones como la desarrollada por [Park et al., 2012] que resuelve en paralelo problemas de 2 dimensiones.

1.5.4. Big Data

El Big Data (o datos masivos) es un término que se popularizó a raíz de [Mashey, 1997] y es un concepto muy amplio que hace referencia a volúmenes de tal tamaño que no pueden ser tratados de manera convencional ya que superan los límites y capacidades de las herramientas habituales. Sus diferentes disciplinas se encargan de estudiar la recolección, almacenamiento, compartición, procesado, análisis y visualización de los datos.

Este tipo de tecnologías, procesadas normalmente mediante paralelización distribuida debido al volumen de datos (aunque no tiene por qué ser así necesariamente), ha dado lugar a numerosos patrones de diseño como MapReduce [Chu et al., 2007] o a tecnologías como Spark [Zaharia et al., 2010] o Hadoop [Shvachko et al., 2010].

Aunque los métodos de núcleo todavía distan mucho de poder aplicarse directamente a los tamaños de petabytes a zettabytes de información que son el objetivo final de este campo (a no ser que se realice un submuestreo o selección de características que haga el problema abordable), los patrones de diseño y técnicas que se utilizan sí pueden tener aplicación directa a una escala menor de datos.

1.6. Motivación

Los métodos de núcleo tienen dos problemas fundamentales:

- Los métodos de núcleo no paramétricos como las SVMs y los GPs tienen la capacidad de ajustar automáticamente la complejidad del modelo en función del tamaño del conjunto de entrenamiento. Esto los ha hecho muy populares porque producen buenos resultados en muchos problemas reales, pero tienen el problema de que el tamaño del clasificador es a menudo demasiado grande cuando se dispone de grandes volúmenes de datos de entrenamiento. Esto impide que se puedan utilizar para hacer clasificación o regresión con nuevos datos en un entorno de ejecución en "tiempo real".

- El coste computacional de estos algoritmos es $O(n^3)$, donde n es el tamaño del conjunto de entrenamiento. Esto hace que su aplicación directa esté estrictamente limitada ya que su tiempo de ejecución es excesivo a la hora de entrenar los modelos.

Actualmente existen dos ramas principales de investigación para abordar estos problemas. Por un lado tenemos la creación de versiones semi-paramétricas de estos algoritmos que garanticen la obtención de un modelo con una complejidad predeterminada. Por otro lado, para abordar el problema del excesivo tiempo de entrenamiento se han propuesto modelos paralelos que permiten reducir dicho tiempo el tiempo de ejecución añadiendo recursos de computación que trabajen de forma simultánea.

Las versiones actuales de métodos de núcleo paralelos sufren alguna limitación:

- La paralelización es subóptima, ya que la suma del tiempo de ejecución de las subtareas en las que se divide el proceso total es superior al tiempo de ejecución total.

- Un porcentaje del tiempo de ejecución no es paralelizable, lo que termina acotando la aceleración máxima que se puede obtener al incluir más recursos de computación.

- Realizan alguna aproximación para obtener de forma rápida las tareas no paralelizables, lo que reduce las prestaciones en algunos problemas.

1.7. Objetivos de la Tesis

Debido a que las limitaciones de los métodos de núcleo residen en su tiempo de ejecución y en la complejidad de los modelos que obtienen, el principal objetivo que se ha perseguido en esta Tesis es la reducción de estas limitaciones para ampliar el abanico de problemas que estas técnicas pueden resolver eficientemente.

Nuestras propuestas consisten en el desarrollo de nuevos esquemas que puedan enfrentarse a estos problemas mejorando las soluciones actuales. Para ello hacemos uso de dos técnicas diferentes:

- Modelos semi-paramétricos: Abordan el problema del excesivo tamaño del modelo ya que el resultado final tiene prefijados los elementos que lo componen. Este tipo de modelos alcanzan prestaciones similares a las del modelo no paramétrico reduciendo el coste computacional y la complejidad.

- Cómputo en paralelo: En segundo lugar se ataca el problema del tiempo de ejecución utilizando simultáneamente múltiples recursos de computación.

1.8. Estructura del documento

El resto de los capítulos que contiene esta Tesis se organiza de la siguiente manera:

En el **Capítulo 2** se hace una revisión más extensa de las versiones no paramétricas y semi-paramétricas de los principales métodos de núcleo SVMs y GPs, haciendo hincapié en los principales problemas que presentan en relación a la escalabilidad y coste computacional. También se hablará de los principales métodos de optimización empleados para resolverlos.

El **Capítulo 3** presenta una forma de abordar el problema de paralelización de métodos de núcleo basada en el lema de inversión de matrices por bloques y el patrón de diseño Mapeo-Reducción (MapReduce), se presentará también la implementación algorítmica aplicada en distintos algoritmos:

- **PS-SVM**: Una versión paralela de SVMs semi-paramétricas.

- **P-GP**: Una versión paralela de los GPs.

- **PS-GP**: Una versión paralela de los GPs semi-paramétricos.

En este capítulo también se mostrarán los resultados obtenidos con estos algoritmos y se analizarán los tiempos de ejecución y la capacidad de aceleración de cada uno de ellos comparándolos con otras soluciones.

Después, en el **Capítulo 4**, se presenta una mejora respecto al método anterior de paralelización que emplea una versión paralela de la descomposición factorial de Cholesky. Esta arquitectura se emplea para implementar dos nuevos algoritmos:

- **PIRWLS**: Una versión paralela de SVMs.
- **PSIRWLS**: Una versión paralela de las SVMs semi-paramétricas.

También se incluyen los resultados y prestaciones obtenidos en estos algoritmos, de los que se realizará un análisis en profundidad.

Hay otros aportes que han surgido indirectamente a raíz de los conocimientos adquiridos durante el desarrollo de esta Tesis que se resumirán brevemente en el **Capítulo 5**; dichos aportes se corresponden a resultados y premios obtenidos en competiciones de aprendizaje automático gracias a la utilización de código paralelo.

Finalmente, en el **Capítulo 6**, se hace un resumen de los resultados obtenidos y se muestran las aportaciones principales de la tesis, reflexionando sobre el trabajo realizado y señalando un conjunto de líneas de investigación abiertas que sería interesante abordar en un futuro próximo.

Capítulo 2

Cómputo de los Métodos de Núcleo

"*Si he logrado ver más lejos*
ha sido porque he subido
a hombros de gigantes"
Isaac Newton (1643 -1727)

Los métodos de núcleo son técnicas de aprendizaje máquina que han sido ampliamente estudiados. En este capítulo, en primer lugar se hace un análisis de las diferentes alternativas que existen a la hora de trabajar con SVMs y GPs y después se analizan los requisitos necesarios para obtener buenas prestaciones en una implementación paralela. También se hablará en detalle de los algoritmos que han sido de interés para esta Tesis, se ha seleccionado un algoritmo de resolución de cada uno de los principales métodos de núcleo (SVMs y GPs) y una aproximación semi-paramétrica de cada uno de ellos.

2.1. Máquinas de Vectores Soporte

La formulación de la SVM es un problema de optimización con restricciones. Existen numerosas técnicas para abordar este tipo de problemas, a continuación

se describirán las más importantes y se justificarán los motivos que han llevado a seleccionar una de ellas.

2.1.1. Alternativas de Optimización

Optimización Mínima Secuencial

El algoritmo más utilizado para resolver el problema de programación cuadrática del entrenamiento de la SVM es el algoritmo SMO [Platt et al., 1999], desarrollado en Microsoft Research. Este algoritmo es el que utiliza LibSVM [Chang and Lin, 2011], la librería más extendida para resolver SVMs.

El algoritmo SMO divide el problema en subproblemas del menor tamaño posible, para ello en cada iteración selecciona dos de los pesos (o multiplicadores de Lagrange) y optimiza la función de coste. Utilizando la formulación dual (1.15) y dejando únicamente como parámetros libres estos dos multiplicadores de Lagrange α_1 y α_2 sabemos que:

$$\begin{aligned} &0 \leq \alpha_1, \alpha_2 \leq C \\ &y_1 \Delta\alpha_1 = -y_2 \Delta\alpha_2 \end{aligned} \tag{2.1}$$

Este problema se puede resolver analíticamente de forma muy sencilla ya que hay que encontrar el mínimo de un problema cuadrático unidimensional en el que se pueden aplicar las restricciones directamente.

El algoritmo funciona de la siguiente forma:

- Selecciona en cada iteración un multiplicador que viole las condiciones de Karush-Kuhn-Tucker.
- Selecciona un segundo multiplicador y optimiza ambos valores.
- Repite los pasos anteriores hasta que converge.

Método del Punto Interior

Los métodos de punto interior (IPM, "Interior Point Method") son métodos eficientes de coste polinómico [Karmarkar, 1984] capaces de obtener soluciones de varias familias de problemas de optimización convexa (programación lineal, cuadrática y semidefinida) con un número relativamente reducido de iteraciones.

Para problemas no lineales:

$$\begin{aligned} &Minimiza \;\; f(\mathbf{w}) \\ &Sujeto\, a \;\; c_i(\mathbf{w}_i) \geq 0, \forall i = 1, 2, ..., n \end{aligned} \tag{2.2}$$

Aplican una función barrera que incluye el conjunto convexo, por ejemplo:

$$B(\mathbf{w}, \mu) = f(\mathbf{x}) - \mu \sum_{i=1}^{n} ln(c_i(\mathbf{w})) \tag{2.3}$$

μ es un escalar positivo que hace converger la función barrera a 2.2 cuando se acerca a 0.

Obteniendo el gradiente de la función barrera:

$$\nabla B(\mathbf{w}) = \nabla f(\mathbf{w}) - \mu \sum_{i=1}^{n} \frac{\nabla c_i(\mathbf{w})}{c_i(\mathbf{w})} \tag{2.4}$$

Aplicando el método de Newton se puede resolver iterativamente el problema:

$$\begin{bmatrix} \mathbf{W} & -\mathbf{A}^\top \\ \Lambda\mathbf{A} & \mathbf{C} \end{bmatrix} = \begin{bmatrix} p_{\mathbf{w}} \\ p_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f + \mathbf{A}^\top \lambda \\ \mu\mathbf{1} - \mathbf{C}\lambda \end{bmatrix} \tag{2.5}$$

Donde $\mathbf{W}$ es la matriz Hessiana de $B(\mathbf{w}, \mu)$, $\mathbf{A}$ una matriz que contiene el Jacobiano de las restricciones c_i, Λ una matriz diagonal que contiene los multiplicadores de Lagrange λ y $\mathbf{C}$ una matriz diagonal donde $\mathbf{C}_{ii} = c_i\mathbf{w}$

Este tipo de métodos se utilizan comúnmente en la resolución de SVMs de tamaño medio, con un número de datos de entrenamiento que no superan las unidades de millar. Esto se debe a que tiene iteraciones con un alto coste tanto computacional

como de memoria haciendo imposible manejar conjuntos de entrenamiento con decenas de miles de datos. Para abordar este problema se suelen realizar aproximaciones dispersas [Ferris and Munson, 2002].

Método Iterativo de Mínimos Cuadrados Re-Ponderados

El Método Iterativo de Mínimos Cuadrados Re-Ponderados (IRWLS, "Iterative Re-Weighted Least Squares") presenta un método para resolver las SVMs [Pérez-Cruz et al., 2001][Pérez-Cruz et al., 2005] que reorganiza la formulación básica (1.13) para hacerla independiente de ξ_i:

$$Minimiza \ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i^n a_i e_i^2 \tag{2.6}$$

$$Sujeto\, a \tag{2.7}$$

$$w_i = \alpha_i y_i \tag{2.8}$$

$$\sum_{j=0}^{n} w_i = 0 \tag{2.9}$$

$$e_i = y_i - \left(\sum_{j=1}^{n} w_j K(\mathbf{x}_i, \mathbf{x}_j) + b \right) \tag{2.10}$$

$$a_i = \begin{cases} 0, y_i e_i \leq 0 \\ \frac{C}{e_i y_i}, y_i e_i \geq 0 \end{cases} \tag{2.11}$$

Para entender este algoritmo, si consideramos que cada a_i no depende de $\mathbf{w}$ se puede obtener el óptimo global de manera directa resolviendo un sistema lineal. Al existir dependencia, se actualiza el valor de a_i en cada iteración. La convergencia de este procedimiento se demuestra en [Pérez-Cruz et al., 2005]. El procedimiento global tiene la siguiente forma:

1. Se suponen los valores de a_i independientes y obtiene el valor de $\mathbf{w}$ resolviendo el sistema de mínimos cuadrados ponderados.

2. Se recalculan los valores de a_i usando el nuevo valor de $\mathbf{w}$

3. Se repite el proceso hasta que converge.

2.1.2. Selección del Método de Optimización

Si analizamos las estrategias de optimización que se acaban de mencionar tenemos una elección clara de trabajo:

- Se ha descartado SMO ya que como hemos podido ver, pese a ser la más eficiente de las opciones, no es la más apropiada para una implementación en paralelo debido a que cada iteración del algoritmo tiene un coste computacional muy reducido y una arquitectura basada en descomponer esta iteración en subtareas se ve afectada enormemente por el tiempo empleado en el lanzamiento y recuperación de resultados de cada una de ellas.

- Se ha descartado IPM por dos motivos, el primero es que, aunque es el método que normalmente converge en un menor número de iteraciones, tiene un coste en cada iteración tanto computacional como de memoria que puede ser muy elevado. Esto acota el número de situaciones en que es práctico. El segundo motivo es que al necesitar el cómputo del gradiente requiere un modelado especial para cada función de núcleo, lo que hace que no sea una aproximación universal.

- Se han seleccionado algoritmos basados en IRWLS ya que son un compromiso intermedio entre las opciones anteriores. Existen versiones tanto completas como dispersas que permiten ajustar el coste computacional y de memoria de cada iteración, lo que da una gran flexibilidad a la hora de analizar la complejidad adecuada de cara a la paralelización de los algoritmos.

2.1.3. Resolución completa de la SVM utilizando IRWLS

Como vimos en (2.6) el problema cuadrático a resolver en cada iteración es del tamaño del conjunto de entrenamiento. Este problema no es abordable en la mayoría

de los casos ya que es necesario guardar en memoria una matriz que contiene la función de núcleo de cada pareja de datos del conjunto de entrenamiento para poder resolver el sistema lineal que obtiene en cada iteración el nuevo valor de los pesos $\mathbf{w}$.

Existen dos técnicas para reducir la complejidad de cada iteración:

- Diferenciación del tipo de vector: El sistema lineal puede ser simplificado como se muestra en [Pérez-Cruz et al., 2001] y [Pérez-Cruz et al., 2005] debido a que el valor del multiplicador de Lagrange asociado a cada muestra bien clasificada fuera del margen es 0 y tiene un valor C en las muestras que son vectores soporte fronterizos (es decir, están bien clasificados y ubicados en el margen). El sistema lineal no necesita trabajar con el conjunto de entrenamiento total, solo con los vectores soporte que no son fronterizos. Esta reducción del tamaño del sistema lineal a resolver reduce significativamente el tamaño de la matriz y el tiempo de resolución del sistema lineal (que tiene un coste $O(n^3)$).

  La forma de trabajar es dividiendo en cada iteración los datos de entrenamiento en tres conjuntos. El primer $\mathcal{S}_1$ confine los vectores soporte no fronterizos, el segundo $\mathcal{S}_2$ contiene las muestras correctamente clasificadas fuera del margen y el tercer conjunto $\mathcal{S}_3$ contiene los vectores soporte fronterizos. Este método, aunque reduce enormemente el coste computacional y de memoria, no garantiza que podamos trabajar con cualquier base de datos, ya que hay problemas donde el número de vectores soporte no fronterizos puede ser demasiado elevado.

- Utilizar un conjunto de trabajo: Se puede realizar un sistema de particionado como los propuestos en [Osuna and Girosi, 1998], [Joachims, 1999] y [Pérez-Cruz et al., 2001], donde se dividen los elementos de entrenamiento en dos grupos. Cada iteración se crea un conjunto de trabajo $\mathcal{S}_w$ con los elementos cuyos pesos se van a actualizar y un conjunto inactivo $\mathcal{S}_{IN}$ que contiene aquellos elementos cuyos pesos van a permanecer constantes en cada iteración. De esta forma, en cada iteración, el tamaño máximo del sistema lineal a resolver será el del conjunto de trabajo, permitiendo que siempre tengamos la complejidad

de la iteración bajo control.

En el apéndice A podemos ver el procedimiento completo de entrenamiento de la SVM utilizando estas dos técnicas para reducir y controlar la complejidad del algoritmo. Se hace uso de un bucle externo que selecciona el conjunto de trabajo e inactivo de cada iteración y un bucle interno que actualiza iterativamente los pesos del conjunto de trabajo utilizando el procedimiento IRWLS.

En [Joachims, 1999] se definen varias condiciones para estudiar la convergencia de una SVM:

$$y_i e_i < \epsilon \quad \forall i | w_i = 0 \tag{2.12}$$

$$y_i e_i > -\epsilon \quad \forall i | w_i = y_i C \tag{2.13}$$

$$|y_i e_i| < \epsilon \quad \forall i | w_i \neq y_i C \quad \& \quad w_i \neq 0 \tag{2.14}$$

Para seleccionar el conjunto de trabajo en cada iteración hemos seguido el método propuesto en [Pérez-Cruz et al., 2005] que elige como candidatos las muestras que son vectores soporte no fronterizos y aquellas que no cumplen las condiciones 2.12 y 2.13. El conjunto de trabajo se inicializa con una muestra de cada clase que viola cada una de las condiciones y después se completa seleccionando muestras aleatoriamente entre las candidatas.

2.1.4. Resolución semi-paramétrica de la SVM utilizando IRWLS

Las SVMs son métodos no paramétricos, el tamaño de la función de clasificación (que es el número de vectores soporte), no está predeterminado y se modela en función de las muestras de entrenamiento. Este tamaño, al no estar bajo control puede ser excesivamente grande y el clasificador no apto para muchas aplicaciones que requieran trabajar en "tiempo real".

En los trabajos [Navia-Vázquez et al., 2001], [Parrado-Hernández et al., 2003],

[Navia-Vázquez, 2007] se selecciona un conjunto de m elementos base $\{\mathbf{c}_i, \ldots, \mathbf{c}_m\}$ y $\mathbf{w}$ se aproxima como $\mathbf{w} \simeq \sum_{i=1}^{m} \beta_i \phi(\mathbf{c}_i)$ dando lugar a un clasificador de tamaño m:

$$f(\mathbf{x_i}) = \sum_{j=1}^{n} \beta_j K(\mathbf{x}_i, \mathbf{c}_j) \tag{2.15}$$

Los pesos $\boldsymbol{\beta}$ se obtienen entonces con el siguiente problema de optimización:

$$\begin{aligned} Minimiza\ & \frac{1}{2}\boldsymbol{\beta}^{\mathbf{T}}\mathbf{K_c}\boldsymbol{\beta} + C\sum_{i=1}^{n}\xi_i \\ & (\mathbf{K_c})_{i,j} = K(\mathbf{c}_i, \mathbf{c}_j), i, j = 1, ..., m \\ Sujeto\ a & \\ & y_i(\boldsymbol{\beta}^T\mathbf{k}_i + b) \geq 1 - \xi_i, i = 1, 2, ..., n \\ & \xi_i \geq 0, i = 1, 2, ..., n \\ donde & \\ & (\mathbf{k}_i)_j = K(\mathbf{x}_i, \mathbf{c}_j) \end{aligned} \tag{2.16}$$

El entrenamiento de este tipo de procedimientos consta de dos etapas, la primera selecciona el conjunto de m muestras que se van a utilizar como elementos base y una segunda etapa que obtiene los pesos óptimos $\boldsymbol{\beta}$.

Aproximación Matricial Codiciosa Dispersa

La Aproximación Matricial Codiciosa Dispersa (SGMA, "Sparse Greedy Matrix Approximation") es una técnica que se usa en la selección de elementos base para modelos semi-paramétricos haciendo crecer iterativamente dicho conjunto utilizando el elemento que maximiza un criterio.

Teniendo el conjunto de entrenamiento de n muestras y un conjunto de elementos base de m elementos podemos aproximar una función de núcleo como una combinación lineal de otras funciones de núcleo:

$$k(\mathbf{x}_i, .) = \sum_{j=1}^{m} \alpha_{i,j} k(\mathbf{c}_j, .) \tag{2.17}$$

Como se muestra en [Smola and Schölkopf, 1998] el error de aproximación de esta combinación lineal es:

$$Err(\alpha^{n,m}) = tr\mathbf{K} - \sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{i,j}k(\mathbf{x}_i, \mathbf{c}_j) \tag{2.18}$$

$$(\mathbf{K})_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j), \forall i, j = 1, ..., n \tag{2.19}$$

El error al añadir un nuevo elemento $\mathbf{c}_{m+1}$ a la base puede expresarse en función del error anterior:

$$Err(\alpha^{n,m+1}) = Err(\alpha^{n,m}) - \underbrace{\eta^{-1}\|\mathbf{K}_{nm}\mathbf{z} - \mathbf{k}_{m+1,n}\|^2}_{\text{Descenso de Error}} \tag{2.20}$$

Donde:

$$\begin{aligned}
&(\mathbf{k}_{m+1,n})_i = k(\mathbf{c}_{m+1}, \mathbf{x}_i) && \forall i = 1, ..., n \\
&(\mathbf{k}_{nc})_i = k(\mathbf{x}_i, \mathbf{c}_{m+1}) && \forall i = 1, ..., m \\
&(\mathbf{k}_{mc})_i = k(\mathbf{c}_i, \mathbf{c}_{m+1}) && \forall i = 1, ..., m \\
&\mathbf{z} = {\mathbf{K}_\mathbf{c}}^{-1}\mathbf{k}_{mc} \\
&\eta = 1 - \mathbf{z}^T\mathbf{k}_{nc} \\
&(\mathbf{K}_{nm})_{ij} = k(\mathbf{x}_i, \mathbf{c}_j) && \forall i = 1, ..., n, j = 1, ..., m
\end{aligned} \tag{2.21}$$

SGMA intenta encontrar el mejor elemento para añadir a la base de un modelo semi-paramétrico seleccionando en cada iteración el elemento cuyo descenso de error es mayor.

Obtención de pesos mediante IRWLS

En los trabajos [Navia-Vázquez et al., 2001, Parrado-Hernández et al., 2003, Navia-Vázquez and Díaz-Morales, 2010] un esquema con IRWLS se ha aplicado con éxito para entrenar una SVM semi-paramétrica. Se reorganiza la función de coste

(2.16) para obtener la expresión:

$$Minimiza\ \frac{1}{2}\boldsymbol{\beta}^{\mathbf{T}}\mathbf{K_c}\boldsymbol{\beta} + \sum_i a_i e_i^2 \tag{2.22}$$

$$Sujeto\,a \tag{2.23}$$

$$e_i = y_i - \left(\sum_{j=0}^{m} \beta_j K(\mathbf{x}_i, \mathbf{c}_j) + b\right) \tag{2.24}$$

$$a_i = \begin{cases} 0, y_i e_i \leq 0 \\ \frac{C}{e_i y_i}, y_i e_i \geq 0 \end{cases} \tag{2.25}$$

Y de forma análoga a la de la SVM completa, se considera que a_i no es una función de $\boldsymbol{\beta}$ y se sigue un proceso iterativo que obtiene $\boldsymbol{\beta}$ y recalcula a_i usando dicho pesos.

2.2. Procesos Gaussianos

Al contrario que en el problema de las SVMs, que requieren técnicas de optimización para obtener el mínimo de la función de coste, en el caso de los procesos Gaussianos, vimos en la sección 1.4.2 que todos los cálculos que intervienen en su resolución hacen uso de las propiedades de la distribución Gaussiana para ser tratables analíticamente de forma directa.

2.2.1. Cálculo de los GPs

En el caso de los procesos Gaussianos para regresión, vimos en la sección 1.4.2 que para realizar predicciones sobre una nueva muestra $\mathbf{x}_i$, la obtención de la predicción de la varianza de la predicción es una expresión matricial:

$$m(f(\mathbf{x}_i)) = \mathbf{k}_i^T(\mathbf{K} + \sigma_m^2\mathbf{I})^{-1}\mathbf{y} \tag{2.26}$$

$$v(f(\mathbf{x}_i)) = K(\mathbf{x}_i\mathbf{x}_i) - \mathbf{k}_i^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_i \tag{2.27}$$

El principal coste computacional reside en la inversión de la matriz $(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}$, que es $O(n^3)$ con el tamaño del conjunto de datos de entrenamiento.

Se puede ver esta expresión de una forma similar a la función de clasificación de la SVM:

$$m(f(\mathbf{x}_i)) = \sum_{j=1}^{n} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.28}$$

donde el valor de los pesos $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]$ se obtendría utilizando la siguiente expresión:

$$\boldsymbol{\alpha} = (\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} \tag{2.29}$$

2.2.2. Aproximación Semi-Paramétrica

Aunque este modelo es analíticamente tratable, en muchos casos no es abordable computacionalmente. Por un lado tenemos el coste asociado a la inversión de la matriz, que limita su aplicación práctica a problemas donde el tamaño del conjunto de entrenamiento es inferior en la actualidad a los centenares de miles de muestras. Por otro lado, si además de la predicción se desea obtener la varianza de esta predicción, es necesario almacenar la inversa de esta matriz en memoria y no solo resolver el sistema lineal. Esto limita aún más la puesta en práctica de esta técnica a conjuntos de entrenamiento con decenas de miles de muestras.

Por estos motivos es interesante realizar aproximaciones, el hecho de tener que tener un peso asociado a cada una de las muestras de entrenamiento puede ser relajado y se puede realizar una aproximación de forma análoga a la de las SVMs semi-paramétricas tomando un subconjunto de elementos base:

$$\mathbf{f} = K_{nm}\boldsymbol{\alpha}_m \tag{2.30}$$

$$p(\boldsymbol{\alpha}_m|\mathbf{X}) \sim \mathcal{N}(0, K_{mm}^{-1}) \tag{2.31}$$

La verosimilitud de los pesos es Gaussiana e $\mathbf{y}$ es una combinación lineal de $\boldsymbol{\alpha}_m$:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}_m) \sim \mathcal{N}(K_{nm}\boldsymbol{\alpha}_m, \sigma_n^2 I) \tag{2.32}$$

Para una expresión semi-paramétrica, la verosimilitud marginal se obtiene:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) &= \int p(\mathbf{y}|\mathbf{X}, \boldsymbol{\alpha}_m) p(\boldsymbol{\alpha}_m|\mathbf{X}) d\boldsymbol{\alpha}_m \\ &\sim \mathcal{N}(\mathbf{0}|\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^{\top} + \sigma_n^2 I) \end{aligned} \tag{2.33}$$

La probabilidad a posteriori de los pesos asociados a la función de regresión es entonces:

$$p(\boldsymbol{\alpha}_m|\mathbf{y}, \mathbf{K}_{mn}) \sim \mathcal{N}(\mu, \Sigma) \tag{2.34}$$

Donde:

$$\begin{aligned} \boldsymbol{\mu} &= \boldsymbol{\Sigma}\mathbf{K}_{nm}^{\top}\mathbf{y}/\sigma_n^2 \\ \boldsymbol{\Sigma} &= [\mathbf{K}_{nm}^T\mathbf{K}_{nm}/\sigma_n^2 + \mathbf{K}_{mm}]^{-1} \end{aligned} \tag{2.35}$$

Basándonos en esta probabilidad a posteriori, se pueden realizar predicciones [Smola and Bartlett, 2001]:

$$\begin{aligned} m(\mathbf{x}_i) &= \mathbf{k}_i^T\boldsymbol{\Sigma}\mathbf{K}_{nm}\mathbf{y} \\ v(\mathbf{x}_i) &= \sigma_n^2 + \mathbf{k}_i^T\boldsymbol{\Sigma}\mathbf{k}_i \end{aligned} \tag{2.36}$$

En [Quinonero-Candela and Rasmussen, 2005] se presentó una versión semi-paramétrica llamada Evidencia Codiciosa Dispersa (SGEV, "Sparse Greedy EVidence"). Esta técnica obtiene los elementos base seleccionando en cada iteración un grupo aleatorio de candidatos y evaluando la evidencia en todos ellos para después añadir aquel que ha obtenido un mayor valor. La log evidencia negativa $\mathcal{L}$, teniendo m elementos base y n datos en el conjunto de entrenamiento se puede evaluar de la siguiente forma:

$$\begin{aligned}
\mathcal{L} &= \mathcal{Q} + \mathcal{G} \\
\mathbf{\Sigma} &= [\mathbf{K}_{nm}^T \mathbf{K}_{nm}/\sigma_n^2 + \mathbf{K}_{mm}]^{-1} \\
\mathcal{Q} &= \frac{1}{2\sigma_n^2}\mathbf{y}^T\mathbf{y} - \frac{1}{4\sigma_n^4}\mathbf{y}^T\mathbf{K}_{nm}\mathbf{\Sigma}\mathbf{K}_{nm}^T\mathbf{y} \\
\mathcal{G} &= \frac{1}{2}[log|\mathbf{\Sigma}| - log|\mathbf{K}_{mm}| + m\ log\ \sigma_n^2]
\end{aligned} \tag{2.37}$$

El valor de la log evidencia se puede calcular de forma eficiente utilizando el resultado de la iteración anterior [Quinonero-Candela and Rasmussen, 2005].

Utilizando este criterio se puede hacer crecer el modelo hasta que tenga el tamaño deseado. El coste computacional de obtener $\mathbf{\Sigma}$ es $\mathcal{O}(nm^2)$, una vez que se obtiene su valor, el coste de hacer una predicción es $\mathcal{O}(m)$ para la media y $\mathcal{O}(m^2)$ para la varianza.

Parte II:

Contribuciones

Capítulo 3

Arquitectura Paralela en Métodos de Núcleo

"*Divide et Impera*"

Julio Cesar (100 a.C. - 44 a.C.)

Con el objetivo de enfrentarnos a las limitaciones existentes en los métodos de núcleo, que son por un lado su excesivo tiempo de entrenamiento y por otro la complejidad de los modelos que obtienen, se han propuesto nuevos esquemas que tratan de reducir estos problemas.

En este Capítulo se proponen varios algoritmos paralelos [Díaz-Morales et al., 2011], [Díaz-Morales and Navia-Vázquez, 2016a]. Uno de ellos destinado a resolver problemas de clasificación utilizando SVMs semi-paramétricas llamado PS-SVM y dos métodos cuyo fin es resolver problemas de regresión: una versión paralela de los Procesos Gaussianos (P-GP) y una implementación paralela del algoritmo SGEV para entrenar Procesos Gaussianos dispersos (PS-GP). La idea subyacente de esta paralelización consiste en la división recursiva de matrices en submatrices para poder dividir sus operaciones en diferentes subtareas que puedan realizarse simultáneamente por varios núcleos de

procesamiento.

Este capítulo se organiza de la siguiente forma: En la Sección 3.1 se analizan los factores que tienen un alto impacto en las prestaciones obtenidas por algoritmos paralelos. Las decisiones de diseño de bajo nivel que se han tomado se han descrito en la Sección 3.2. La descripción de cómo se ha realizado la paralelización de las operaciones aparece en la Sección 3.3 y los algoritmos que se han implementado se describen en la Sección 3.4. Los resultados experimentales se muestran en la Sección 3.5. Finalmente, se muestran las conclusiones obtenidas de este trabajo en la Sección 3.6.

3.1. Factores a Considerar

A la hora de implementar un algoritmo paralelo destinado al Cómputo de Alto Rendimiento (HPC, "High Performance Computing"), existen numerosos factores que es necesario tener en cuenta ya que influyen de manera directa en el rendimiento que se va a obtener.

En primer lugar tenemos la arquitectura del microprocesador. Como vimos en la figura 1.7, un sistema multiprocesador y multinúcleo contiene varias memorias caché que se encuentran organizadas por niveles (L1, L2, etc.) y una memoria RAM que es común para todos los procesos. Cuando se divide una tarea en subprocesos, al tener todos ellos que compartir recursos, existen muchas situaciones en las que se pueden producir "cuellos de botella" en el acceso y procesado de los datos. Las más destacables son las siguientes:

- Acceso concurrente a la memoria: Cuando varios núcleos del procesador necesitan acceder de forma simultánea a datos de la RAM ya que no los tienen disponibles en su memoria caché, este acceso se produce de forma secuencial y es especialmente ineficiente si los distintos datos se encuentran en posiciones muy separadas de la memoria.

- Diferentes copias de los datos en distintas memorias caché: Si uno de estos núcleos modifica la copia que tiene en la caché se tienen que actualizar el resto de copias y la memoria RAM para mantener la consistencia, esta comunicación afecta a la eficiencia de la paralelización.

- Es útil considerar que la lectura de la memoria es mucho más eficiente a la hora de leer datos que están guardados en posiciones contiguas, ya que disminuye el tiempo destinado al indexado de los datos.

Existen también otros factores de más alto nivel, relacionados con la elección del algoritmo a paralelizar, que también afectan enormemente a las prestaciones:

- Secciones de código no paralelizable: Esto supondrá, de acuerdo a la ley de Amdhal (fórmula 1.30), una cota superior a la aceleración que podamos alcanzar.

- Comunicación: Cuando el tiempo empleado en comunicar cálculos intermedios por parte de las diferentes subtareas no es despreciable frente al tiempo de ejecución de éstas se produce un cuello de botella que impide mejorar la aceleración.

3.2. Elecciones de diseño

Tras estudiar los factores que pueden tener un impacto negativo en el rendimiento, se han tomado algunas estrategias de diseño que permitan obtener buenas prestaciones en términos de tiempo de ejecución y paralelización.

Para evitar los posibles problemas que pueden surgir relacionados con la arquitectura de los microprocesadores se han tomado las siguientes medidas:

- Bloques contiguos de memoria para almacenar datos: Una forma muy común de almacenar matrices es reservar memoria para cada una de las filas que la componen. En ese tipo de diseños, cuando se trabaja sobre una fila y se

pasa a la siguiente, se pierde tiempo indexando la nueva fila en la memoria y accediendo a ella. Para evitar esto, se ha reservado una zona de memoria única para almacenar matrices recurriendo a punteros para referenciar la posición de inicio de cada fila. De esta forma el paso de una fila a la siguiente se realiza de forma directa, en la figura 3.1 se puede ver un ejemplo de una matriz con este formato, donde se reserva el espacio de memoria A, que almacena la totalidad de la matriz, y se reserva el vector C para indicar dónde comienza cada fila. Las matrices de funciones de núcleo y los conjuntos de entrenamiento han sido almacenados internamente de esta forma para mejorar las prestaciones.

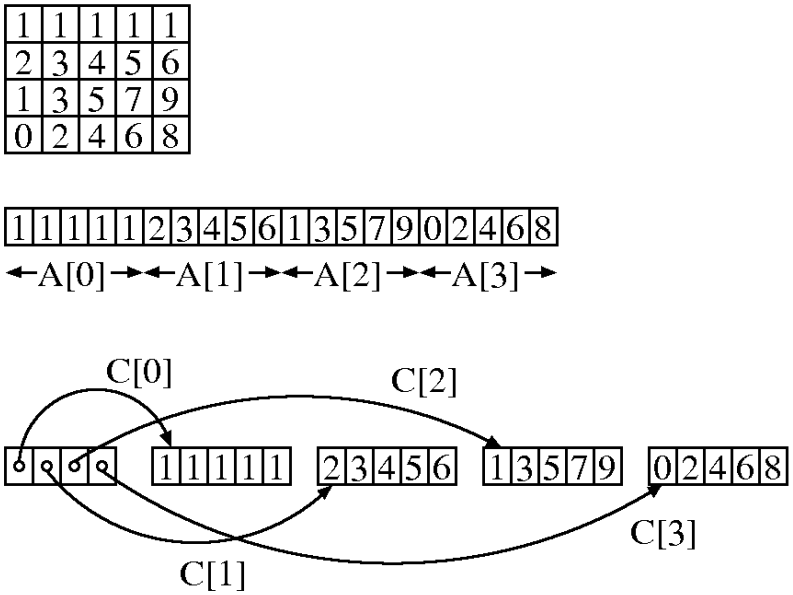


Figura 3.1: Almacenamiento en Memoria de una Matriz

- Si una variable sólo va a ser utilizada por un núcleo del procesador no se utiliza memoria compartida para evitar que se actualicen las memorias caché en cada modificación.

- Siempre que es posible, cada núcleo del sistema trabaja sobre un área de memoria diferente. Por ejemplo, en el caso del los datos de entrenamiento cada uno de los procesos accede a un subconjunto diferente de ellos, de esta forma

si se produce una modificación en los datos el resto de núcleos no tienen copias de ellos en su memoria caché particular y no es necesario actualizarlos.

También se han tomado otras consideraciones de más alto nivel para obtener buenas prestaciones de paralelización:

- Se ha realizado un diseño, en el que el tiempo de ejecución de las partes no paralelas del código es absolutamente despreciable en comparación con el tiempo de ejecución total.
- Se ha optado por trabajar con un esquema de memoria compartida para minimizar el tiempo de comunicación entre subtareas ya que cuando una tarea necesita acceder a los resultados de otra los tiene inmediatamente disponibles en la memoria RAM.

3.3. Paralelización de Operaciones Básicas

3.3.1. Productos de Matrices

Los productos de matrices se pueden realizar de forma eficiente, como se ha visto en [Ranger et al., 2007], ya que el cómputo de cada fila de la matriz resultante se puede realizar de forma independiente por los diferentes núcleos del procesador. Como se puede ver en la figura 3.2, en un entorno de memoria compartida se puede implementar de forma sencilla la paralelización. Un proceso puede obtener la mitad superior de la matriz ($\mathbf{X}^{\mathrm{TOP}}$) y el otro la mitad inferior ($\mathbf{X}^{\mathrm{Bottom}}$).

Este proceso puede a su vez repetirse iterativamente hasta llegar al punto de que una tarea obtenga únicamente una fila de la matriz resultado.

3.3.2. Inversión de matrices

Los métodos tradicionales de inversión de matrices son secuenciales, lo que hace realmente difícil su paralelización. Para realizar esta tarea se ha optado por subdividir

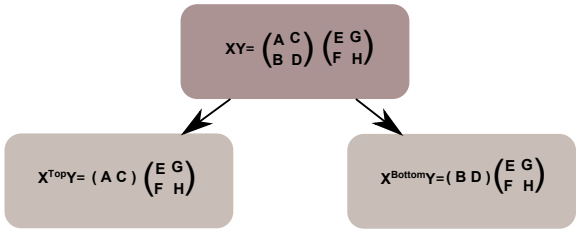


Figura 3.2: Subdivisión de un producto Matricial en dos Subtareas

la matriz en cuatro submatrices y usar la inversión de matrices por bloques:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} & (\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \end{bmatrix} \tag{3.1}$$

El motivo de esta elección es que las operaciones se pueden subdividir en dos tareas. La figura 3.3 muestra como se descomponen las operaciones en dos tareas que se pueden realizar de forma simultánea.

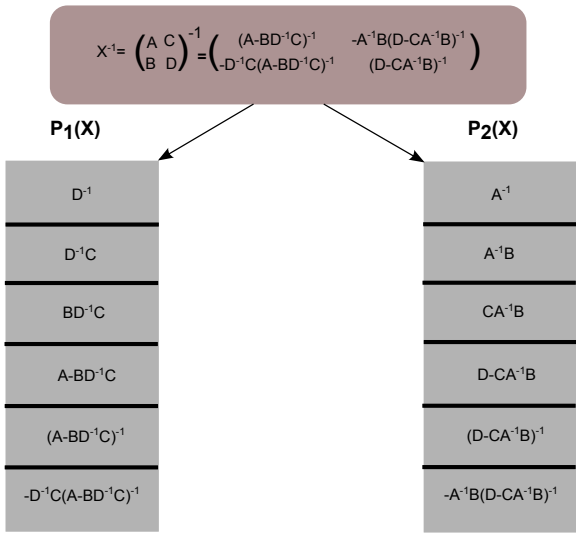


Figura 3.3: Subdivisión de una Inversión de Matrices en dos Subtareas

Cada subtarea está compuesta de dos inversiones y tres productos de matrices, el tiempo de ejecución de cada subtarea es ligeramente superior a la mitad del tiempo de ejecución de invertir una matriz completa. Esto se puede solventar utilizando

descomposición LU [Press et al., 1996] y sustitución hacia atrás, ya que es posible implementar una inversión y un producto (por ejemplo $\mathbf{A^{-1}B}$) con prácticamente el mismo coste computacional que una inversión ($\mathbf{A^{-1}}$).

Al estar cada una de las tareas compuesta por productos e inversiones, se puede emplear un proceso recursivo que permita a su vez seguir dividiendo las tareas. En la figura 3.4 se muestra una inversión utilizando cuatro núcleos de un procesador.

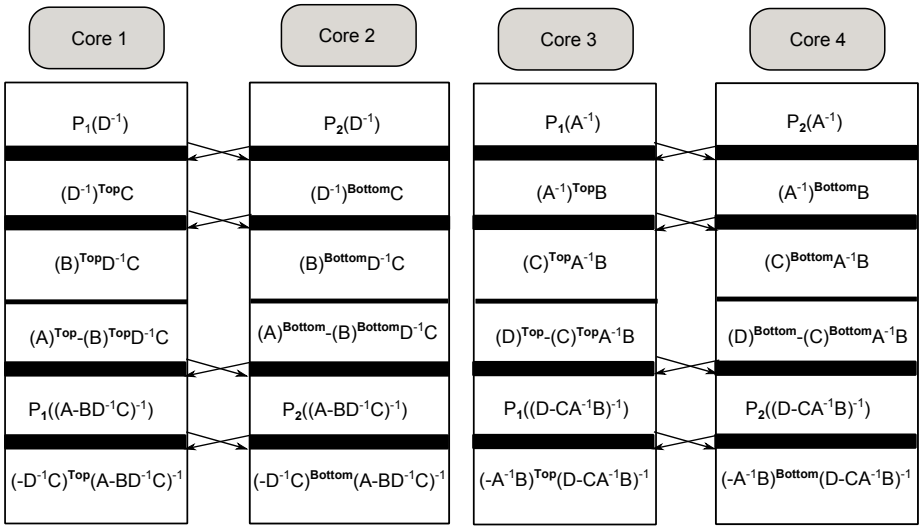


Figura 3.4: Subdivisión de una Inversión de Matricial en cuatro Subtareas

3.3.3. Arquitecturas de crecimiento iterativo

Los algoritmos codiciosos ("Greedy") son técnicas que, para resolver una tarea, actúan de forma iterativa resolviendo en cada iteración un optimo local con el objetivo de aproximar al final un óptimo global. Se utilizan muy comúnmente en arquitecturas crecientes, donde se selecciona un grupo de candidatos $\mathbf{x}_1, ..., \mathbf{x}_m$ de un conjunto, se evalúan utilizando una función objetivo $f(\mathbf{x}_i)$ y se actualiza la arquitectura utilizando el elemento que ha obtenido mejor resultado.

Esta evaluación puede realizarse distribuyendo a los candidatos entre los núcleos del sistema. La figura 3.5 muestra la paralelización de este tipo de arquitectura.

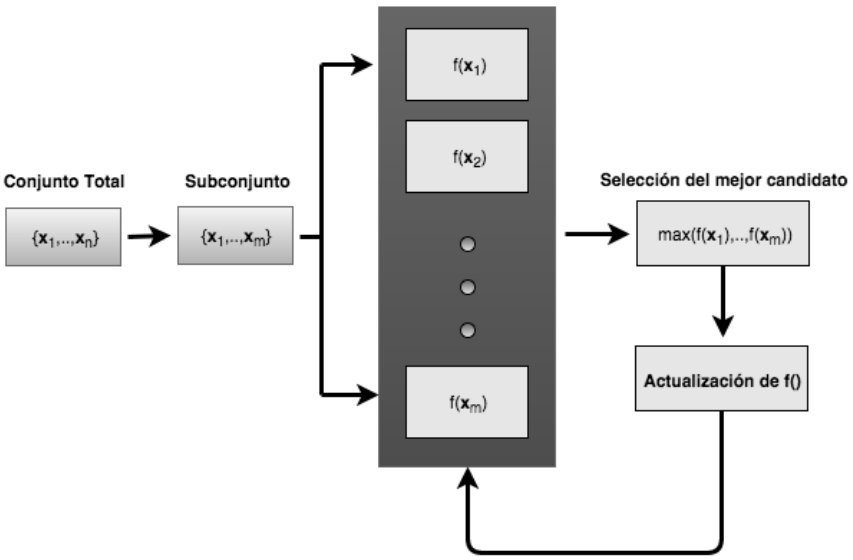


Figura 3.5: Paralelización de técnicas "Greedy"

3.4. Algoritmos Seleccionados

Se han seleccionado diferentes algoritmos para ser implementados en paralelo. El objetivo es ver que este tipo de técnicas son útiles a la hora de trabajar con algoritmos que realizan operaciones sobre matrices de funciones de núcleo.

Los modelos elegidos han sido por un lado versiones semi-paramétricas de SVMs y GPs (ver Capítulo 2) ya que este tipo de técnicas permite mantener la complejidad bajo control y unir esto a la paralelización puede incrementar enormemente el ámbito de utilización de estas técnicas. Por otro lado se ha implementado también una versión paralela de un GP para poder evaluar la utilidad en modelos completos de métodos de núcleo.

3.4.1. SVMs Semi-paramétricas Paralelas

Se ha implementado un modelo semi-paramétrico de SVMs llamado PS-SVM ("Parallel Semi-parametric Support Vector Machine") utilizando SGMA para obtener los elementos base del modelo semi-paramétrico y IRWLS para obtener los pesos

de estos elementos.

SGMA

Se ha utilizado la arquitectura descrita en el apartado 3.3.3 para la implementación de este algoritmo. En cada iteración se puede evaluar en núcleos independientes el descenso del error que conlleva incorporar cada candidato al modelo semiparamétrico.

La actualización del modelo al añadir un nuevo elemento a la base requiere aumentar de rango la matriz de funciones de núcleo de los elementos utilizados como base y su inversa. Esta operación se realizada mediante productos de matrices, por ello se ha utilizado la paralelización descrita en el apartado 3.3.1.

IRWLS

En el algoritmo IRWLS, utilizado para la obtención de los pesos del modelo semiparamétrico, el principal coste computacional reside en la resolución del sistema de ecuaciones, que es $\mathcal{O}(n^2m)$. En este sistema lineal en primer lugar se obtienen las matrices $\mathbf{K_1}$ y $\mathbf{k}_2$ (ver Apéndice C.2) mediante operaciones matriciales que pueden paralelizarse como se indica en 3.3.1. Para resolver el sistema, se ha utilizado la inversión matricial descrita en el apartado 3.3.2.

La actualización del error e_i y de la variable a_i asociados a cada elemento del conjunto de entrenamiento se ha realizado de forma paralela distribuyendo los elementos del conjunto de entrenamiento entre los diferentes núcleos disponibles del procesador.

3.4.2. GPs Paralelos

Para ver la validez de estas técnicas fuera de las SVMs, también se ha realizado una versión paralela de los GPs llamada P-GP ("Parallel-Gaussian Processes"). El principal coste computacional, que reside en la inversión de una matriz de funciones

de núcleo, se ha abordado utilizando la inversión matricial por bloques descrita en la sección 3.3.2. El resto de operaciones se han paralelizado utilizando las operaciones matriciales descritas en la sección 3.3.

3.4.3. GPs Semi-paramétricos Paralelos

También se ha realizado una versión paralela del algoritmo SGEV (ver sección 2.2.2) llamada PS-GP ("Parallel Semi-parametric Gaussian Processes"). En su implementación se ha utilizado la arquitectura "Greedy" descrita en el apartado 3.3.3, evaluando en cada iteración la log evidencia del grupo de candidatos de forma paralela. La actualización del modelo en cada iteración con el candidato ganador se ha realizado mediante las operaciones matriciales descritas en la sección 3.3.

3.5. Experimentos

Todos los algoritmos han sido implementados en C debido a la velocidad de ejecución que ofrece este lenguaje. Para las operaciones de algebra lineal se ha utilizado la librería Intel MKL [Wang et al., 2014], que ofrece rutinas de bajo nivel como productos y sumas de matrices.

Para desarrollar el código paralelizable, se ha utilizado OpenMP [Dagum and Enon, 1998], una interfaz de programación para implementar código multiprocesador y multinúcleo en entorno de memoria compartida.

Se han realizado experimentos para evaluar la eficiencia y aceleración de los algoritmos. Los experimentos se han ejecutado en un servidor HP DL160 G6 con 12 núcleos de procesamiento. Las características se pueden ver en la tabla 3.1.

Para evaluar la paralelización se ha tomado como parámetro la aceleración:

$$\text{Aceleración} = \frac{\text{Tiempo Medio Ejecución Lineal}}{\text{Tiempo Medio Ejecución Paralela}} \tag{3.2}$$

Tabla 3.1: Arquitectura del Hardware

Característica	Valor
Modelo	HP DL160 G6
Instrucciones	64-bit
Frecuencia Base	3.06 GHz
Procesadores	2 Intel Xeon E5-2666 v3
Núcleos Reales	12 (2 x 6)
Memoria	48 GB
Caché	12 MiB
Hyperthreading	Si (24 Núcleos Virtuales, 12 Reales)
Sistema operativo	Linux Gentoo

3.5.1. PS-SVM

Este algoritmo ha sido evaluado tomando como referencia PSVM [Zhu et al., 2008] y LibSVM [Chang and Lin, 2011]. El motivo de elegir estos algoritmos ha sido que ambos están implementados también en C y que cuentan con versiones en paralelo.

LibSVM es utilizada muy comúnmente como referencia por investigadores en el área de aprendizaje automático. Es una versión completa (y por tanto no paramétrica) de la SVM que utiliza el algoritmo SMO para su resolución. Para su paralelización se han seguido las instrucciones descritas en [Chih-Chung and Chih-Jen, 2015] y se ha utilizado la versión 2.91.

PSVM es una implementación dispersa y paralela de SVMs muy popular. Utiliza el algoritmo IPM para su resolución, para ello utiliza una Factorización Paralela de Cholesky Incompleta (PICF, "Parallel Incomplete Cholesky Factorization") basada en filas. Mediante esta factorización aproxima una matriz de funciones de núcleo de tamaño n x n utilizando una de tamaño n x p, donde n es el tamaño del conjunto de entrenamiento. Recibe un parámetro llamado coeficiente de rango cuyo valor es p/n.

PSVM utiliza esta factorización para resolver un sistema de mínimos cuadrados, esta implementación no es completamente paralelizable, con lo que para valores

pequeños del coeficiente de rango el porcentaje de código no paralelizable tiene un tiempo de ejecución despreciable, pero según incrementamos el valor de este parámetro, la máxima aceleración posible disminuye.

PS-SVM recibe como parámetro el número de elementos base que se seleccionan mediante SGMA.

Tanto PS-SVM como PSVM convergen a una solución completa de la SVM cuando el coeficiente de rango es igual a 1 y el número de elementos base es igual al número de elementos del conjunto de entrenamiento. En los experimentos se han analizado las prestaciones para diferentes valores de estos parámetros.

Para comparar estos algoritmos se han seleccionado cuatro conjuntos de datos que varían en tamaño y número de características que se utilizan muy comúnmente como referencia. El objetivo es poder comparar la aceleración y el tiempo de ejecución en diferentes entornos.

Adult

Adult es una base de datos del repositorio de la universidad de California Irvine (UCI)[Asuncion and Newman, 2007] que tiene 123 atributos y se compone de 32561 muestras de entrenamiento y 16281 de test. Es un problema de clasificación binario cuyo objetivo es predecir cuando una persona tiene unos ingresos superiores a 50,000$ anuales.

En todos los experimentos se ha utilizado una función de núcleo Gaussiana. El parámetro de la función de núcleo γ y el parámetro C de la función de coste se han obtenido utilizando validación cruzada con 10 particiones y explorando los siguientes valores:

$$\gamma = 10^n \qquad n : \text{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6}$$
$$C = 10^m \qquad m : \text{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6}$$

Cada algoritmo se ha ejecutado cinco veces y se han obtenido las prestaciones utilizando diferente número de núcleos de procesamiento. La tabla 3.2 muestra los

parámetros utilizados en cada algoritmo y la precisión y el tamaño del clasificador obtenido. En el caso de PS-SVM se incluye también la desviación típica del resultado ya que depende de la selección aleatoria de elementos candidatos a ser incorporados al modelo en cada iteración de SGMA.

Tabla 3.2: Parámetros, precisión y tamaño del clasificador con el conjunto Adult

Algoritmo	C	γ	Precisión Media(Desviación)	Tamaño
LibSVM	100000	0.0001	0.8503	10515
PS-SVM 50	100	0.0001	0.8488(1e-003)	50
PS-SVM 100	1000	0.1	0.8483(1e-004)	100
PS-SVM 200	1000	0.01	0.8506(2e-003)	200
PS-SVM 300	1	0.1	0.8512(6e-004)	300
PSVM 0.003	100	0.001	0.8469	11918
PSVM 0.008	10000	0.0001	0.8478	11526
PSVM 0.012	1	0.01	0.8479	11932

La tabla 3.3 muestra la media y la desviación típica del tiempo de ejecución.

En la figura 3.6(a) se ha representado la precisión (tasa de acierto) en función del tiempo de ejecución lineal (sin paralelizar). Para ello se han tomado valores del tiempo de ejecución utilizando diferentes valores del coeficiente de rango (en el caso de PSVM) y del número de elementos base (en el caso de PS-SVM).

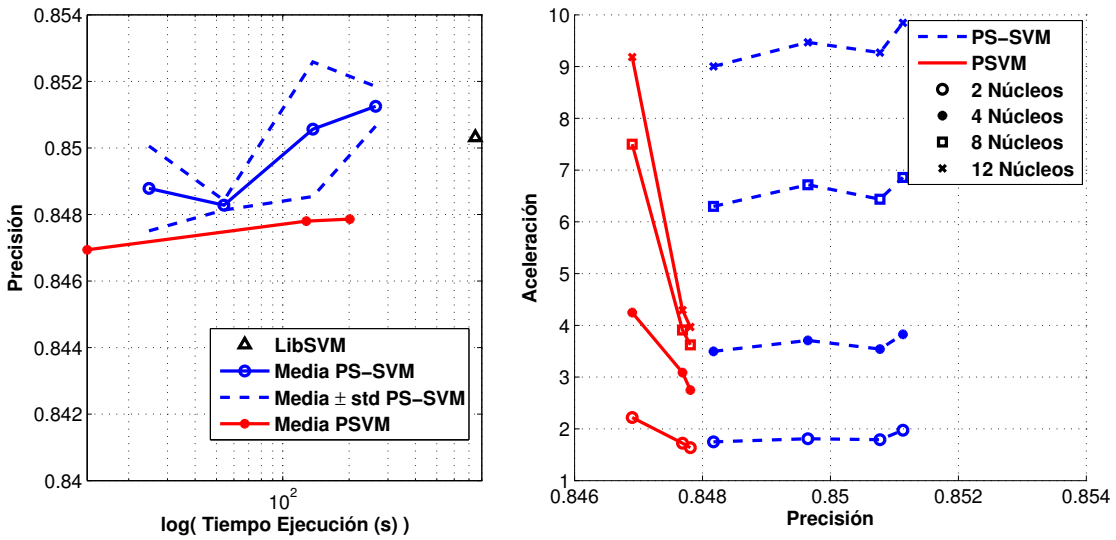
Cuando comparamos el tiempo de ejecución, PS-SVM y PSVM pueden obtener una precisión similar a la de LibSVM pero 10 veces más rápido. PS-SVM es el algoritmo que converge más rápido a la solución obtenida por una SVM completa.

El tamaño del clasificador en el caso de PS-SVM es dos órdenes de magnitud menor que en el caso de LibSVM y PSVM, ésta es la mayor ventaja de los modelos semi-paramétricos, los modelos resultantes son mucho más rápidos a la hora de evaluar nuevas muestras.

En la figura 3.6(b) hemos representado la aceleración en función de la precisión. PS-SVM escala mejor que PSVM. En el caso de PSVM la aceleración disminuye al aumentar el valor del coeficiente de rango ya que el porcentaje de tiempo de ejecución no paralelo se incrementa. Por ejemplo, para 12 núcleos y valores pequeños

Tabla 3.3: Tiempo de ejecución(s) con el conjunto Adult

Algoritmo	Núcleos de Procesamiento				
	1	2	4	8	12
Media LibSVM	748.5	441.3	287.8	209.3	183.5
Desviación LibSVM	2.4	1.5	0.5	0.3	0.2
Media PS-SVM 50	24.7	13.7	6.7	3.7	2.6
Desviación PS-SVM 50	1.2	0.2	0.2	0.1	0.2
Media PS-SVM 100	54.0	30.9	15.5	8.6	6.0
Desviación PS-SVM 100	1.6	0.9	0.3	0.1	0.2
Media PS-SVM 200	137.1	76.5	38.7	21.3	14.8
Desviación PS-SVM 200	2.1	2.0	1.0	0.4	0.5
Media PS-SVM 300	263.9	133.8	69.0	38.5	26.8
Desviación PS-SVM 300	2.5	2.0	0.5	0.6	0.4
Media PSVM 0.003	12.9	5.8	3.0	1.7	1.4
Desviación PSVM 0.003	0.1	0.2	0.1	0.2	0.1
Media PSVM 0.008	127.9	74.3	41.4	32.7	29.8
Desviación PSVM 0.008	0.1	7.3	2.5	0.6	1.9
Media PSVM 0.012	201.5	123.1	73.3	55.6	50.8
Desviación PSVM 0.012	0.0	7.0	0.3	0.2	5.4



(a) Precisión en función del tiempo

(b) Aceleración en función de la precisión

Figura 3.6: Resultados del conjunto Adult

del coeficiente de rango y número de elementos base ambos algoritmos obtienen una buena aceleración, pero según incrementamos su valor la aceleración de PS-SVM se incrementa mientras que la de PSVM disminuye.

WEB

El segundo conjunto de entrenamiento utilizado es WEB, del repositorio que tiene LibSVM. Contiene un total de 24692 muestras de entrenamiento, 25057 de test y 300 atributos. Cada muestra consiste en una bolsa de palabras de páginas web.

Este conjunto de entrenamiento es muy desbalanceado. Para obtener un número de elementos base representativos de ambas clases, PS-SVM selecciona un 50 % de candidatos de ambas clases en el algoritmo SGMA.

Como en el caso anterior, se emplea función de núcleo Gaussiana y los parámetros han sido obtenidos utilizando validación cruzada con 10 particiones. Cada algoritmo se ha ejecutado 5 veces y sus resultados han sido promediados.

La tabla 3.4 muestra los parámetros obtenidos mediante validación cruzada, la precisión (tasa de acierto) y el tamaño del clasificador para libSVM, PS-SVM y PSVM. La tabla 3.5 muestra la media y la desviación típica del tiempo de ejecución para diferente número de núcleos de procesamiento.

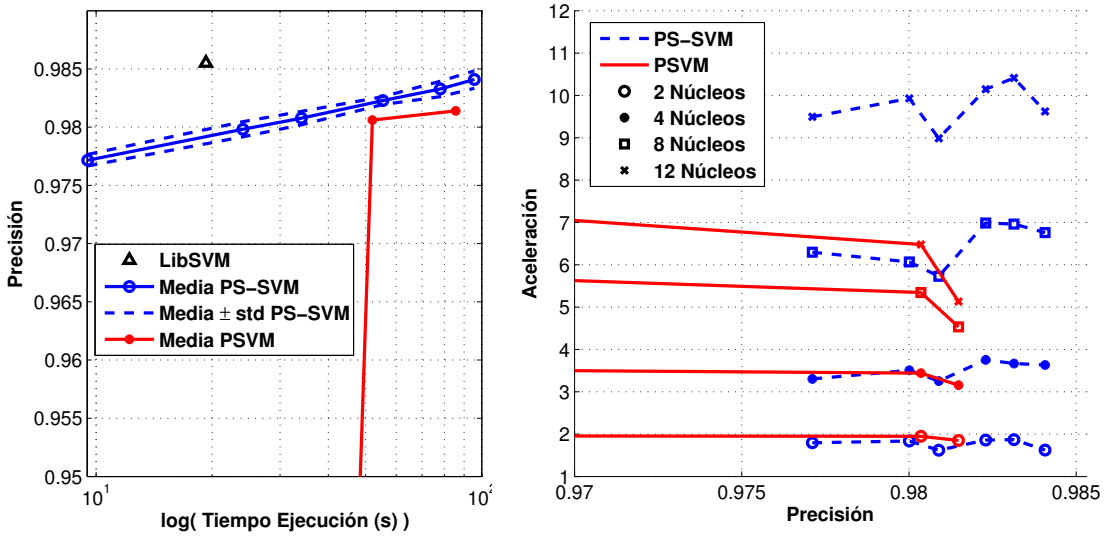
La figura 3.7(a) representa la precisión en función del tiempo de ejecución cuando solo un núcleo de procesamiento es utilizado.

El tiempo de ejecución empleado por los tres algoritmos para alcanzar una precisión similar tiene el mismo orden de magnitud. Utilizando un núcleo del procesador, el tiempo de ejecución de LibSVM es menor, pero al paralelizar los algoritmos tanto PSVM como PS-SVM superan a LibSVM si se utilizan más de dos núcleos de procesamiento.

La figura 3.7(b) muestra la aceleración de PSVM y PS-SVM en función de la precisión. PSVM alcanza valores muy altos de aceleración para valores pequeños del coeficiente de rango, pero el algoritmo aún no ha convergido a una buena solución. Como en el experimento anterior, la aceleración de PS-SVM es estable para un mayor

Tabla 3.4: Parámetros, precisión y tamaño del clasificador en WEB

Algoritmo	C	γ	Precisión Media(Desviación)	Tamaño
LibSVM	100	0.1	0.9855	2531
PS-SVM 30	100	0.01	0.9772(5e-004)	30
PS-SVM 60	1000	0.1	0.9798(7e-004)	60
PS-SVM 90	1000	0.01	0.9808(6e-004)	90
PS-SVM 120	100	0.1	0.9823(3e-004)	120
PS-SVM 150	100	0.01	0.9833(7e-004)	150
PS-SVM 180	1000	0.01	0.9841(8e-004)	180
PSVM 0.002	1000	10	0.8944	24692
PSVM 0.004	1	0.01	0.8748	1533
PSVM 0.006	1	10	0.8944	24692
PSVM 0.008	100	0.0001	0.9806	1596
PSVM 0.010	100	0.001	0.9814	1398



(a) Precisión en función del tiempo

(b) Aceleración en función de la precisión

Figura 3.7: Resultados con el conjunto WEB

rango de sus parámetros.

Tabla 3.5: Tiempo de ejecución(s) con el conjunto WEB

Algoritmo	Núcleos de Procesamiento 1	2	4	8	12
Media LibSVM	19.3	11.0	6.6	4.2	3.3
Desviación LibSVM	0.1	0.1	0.0	0.0	0.0
Media PS-SVM 30	9.5	5.3	2.9	1.5	1.0
Desviación PS-SVM 30	1.5	0.2	0.2	0.2	0.1
Media PS-SVM 60	24.0	13.1	6.8	4.0	2.4
Desviación PS-SVM 60	4.0	0.7	0.8	0.6	0.1
Media PS-SVM 90	34.1	21.0	10.5	5.9	3.8
Desviación PS-SVM 90	0.3	1.2	0.1	0.2	0.1
Media PS-SVM 120	55.4	29.8	14.8	7.9	5.5
Desviación PS-SVM 120	1.7	2.5	1.8	0.4	0.2
Media PS-SVM 150	78.1	41.8	21.3	11.2	7.5
Desviación PS-SVM 150	13.1	4.4	1.1	1.1	0.5
Media PS-SVM 180	95.7	59.0	26.3	14.2	9.9
Desviación PS-SVM 180	1.4	4.4	1.5	1.1	1.0
Media PSVM 0.002	9.5	4.8	2.4	1.3	1.0
Desviación PSVM 0.002	0.0	0.0	0.0	0.0	0.0
Media PSVM 0.004	10.7	4.6	2.3	1.3	1.1
Desviación PSVM 0.004	0.1	0.0	0.0	0.0	0.1
Media PSVM 0.006	42.4	21.3	10.8	5.5	3.8
Desviación PSVM 0.006	0.0	0.0	0.0	0.0	0.0
Media PSVM 0.008	52.1	26.7	15.1	9.7	8.0
Desviación PSVM 0.008	0.7	0.2	0.1	0.1	0.8
Media PSVM 0.010	85.6	46.3	27.1	18.9	16.7
Desviación PSVM 0.010	0.1	0.3	0.2	0.1	1.0

USPS

El siguiente conjunto de entrenamiento utilizado es USPS. Contiene 7291 dígitos en el conjunto de entrenamiento y 2007 en el de test, el número de atributos en cada muestra es 256. Contiene dígitos escritos manualmente (cada muestra es una imagen de 16×16 pixeles), con lo que hay 10 posibles clases, para transformarlo en un problema de clasificación binaria se ha entrenado el clasificador para diferenciar dígitos pares de dígitos impares.

La tabla 3.6 muestra los parámetros obtenidos mediante la validación cruzada con 10 particiones así como la precisión y el tamaño del clasificador. La tabla 3.7 contiene la media y la desviación típica obtenida por cada algoritmo (se han realizado 5 ejecuciones).

Tabla 3.6: Parámetros, precisión y tamaño con USPS

Algoritmo	C	γ	Precisión Media(Desviación)	Tamaño
LibSVM	100000	0.0001	0.9731	646
PS-SVM 60	1000	0.1	0.9423(5e-003)	60
PS-SVM 90	1000	0.001	0.9561(3e-003)	90
PS-SVM 120	1000	0.01	0.9581(3e-003)	120
PS-SVM 150	1000	0.001	0.9652(2e-003)	150
PS-SVM 180	1	0.1	0.9668(2e-003)	180
PSVM 0.004	100	0.001	0.9576	1088
PSVM 0.006	1000	0.0001	0.9646	678
PSVM 0.008	1000	0.0001	0.9676	623

En la figura 3.8(a) se representa la precisión en función del tiempo de ejecución y en la figura 3.8(b) se puede ver la aceleración de PSVM y PS-SVM para diferentes valores de sus parámetros.

En este caso, LibSVM tiene mejor precisión y tiempo de ejecución porque el algoritmo tiene una solución que contiene un número reducido de vectores soporte, con lo que LibSVM es muy eficiente y extremadamente rápido resolviendo el problema. Para alcanzar la misma precisión, el tiempo de ejecución de PS-SVM es menor que el obtenido con PSVM.

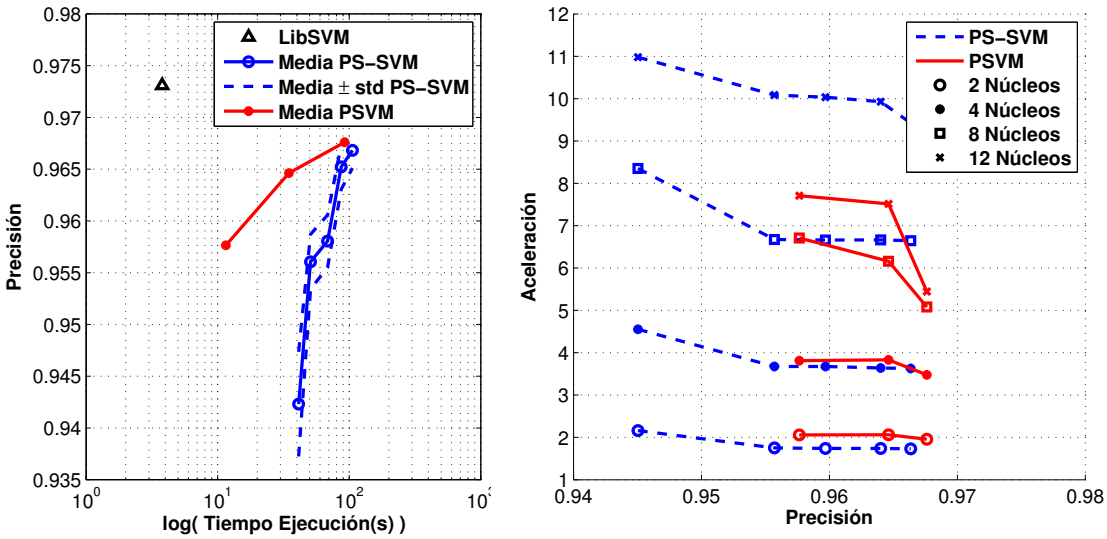
La aceleración, como en los experimentos anteriores, es mejor para PS-SVM que para PSVM para valores altos del coeficiente de rango.

MNIST

El conjunto de entrenamiento MNIST contiene un total de 60000 datos de entrenamiento y 10000 de test. El número de atributos es 784 (son imágenes de 28

Tabla 3.7: Tiempo de ejecución(s) con el conjunto USPS

Desviación Media LibSVM	3.8	2.6	1.7	1.4	1.4
Desviación PS-SVM 30	0.0	0.0	0.0	0.0	0.0
Media PS-SVM 60	41.2	19.1	9.1	4.9	3.8
Desviación S-SVM 60	0.2	0.0	0.0	0.0	0.7
Media PS-SVM 90	50.8	29.0	13.8	7.6	5.0
Desviación PS-SVM 90	0.1	0.3	0.0	0.1	0.0
Media PS-SVM 120	68.5	39.4	18.6	10.3	6.8
Desviación PS-SVM 120	0.2	0.1	0.2	0.0	0.0
Media PS-SVM 150	87.0	50.0	23.9	13.0	8.8
Desviación PS-SVM 150	0.1	0.1	0.0	0.1	0.2
Media PS-SVM 180	105.9	61.2	29.2	15.9	11.2
Desviación PS-SVM 180	0.3	0.1	0.1	0.2	1.2
Media PSVM 0.004	11.5	5.6	3.0	1.7	1.5
Desviación PSVM 0.004	0.0	0.0	0.0	0.0	0.1
Media PSVM 0.006	34.8	16.9	9.1	5.6	4.6
Desviación PSVM 0.006	0.1	0.1	0.1	0.1	0.1
Media PSVM 0.008	92.3	47.2	26.5	18.2	17.0
Desviación PSVM 0.008	0.3	2.4	1.0	0.1	0.8



(a) Precisión en función del Tiempo

(b) Aceleración en función de la Precisión

Figura 3.8: Results of USPS dataset

pixeles).

Hay un total de 10 clases y se han entrenado 10 clasificadores "uno contra todos". En todos ellos se ha utilizado función de núcleo Gaussiana y debido al coste computacional su parámetro γ se ha fijado a 0.05. El parámetro C se ha obtenido utilizando validación cruzada con 10 particiones. La tabla 3.8 muestra el valor obtenido del parámetro C para cada algoritmo y cada uno de los clasificadores.

Tabla 3.8: Valor de m ($C = 10^m$) obtenido mediante validación cruzada para el conjunto MNIST para cada uno de los clasificadores "uno contra todos"

Algoritmo	0	1	2	3	4	5	6	7	8	9
LIBSVM	+5	+5	+5	+5	+5	+5	+5	+5	+5	+5
PS-SVM 100	+5	+1	+2	+5	+2	+3	+5	+1	+3	+4
PS-SVM 200	+5	+3	+3	+4	+4	+5	+3	+3	+2	+5
PS-SVM 300	+3	+5	+4	+4	+4	+2	+4	+5	+2	+2
PS-SVM 400	+2	+3	+3	+5	+3	+1	+4	+5	+3	+3
PS-SVM 500	+4	+2	+5	+3	+5	+2	+4	+4	+3	+5
PS-SVM 1000	+2	+3	+5	+5	+2	+3	+2	+4	+3	+2
PSVM 0.001	+0	+5	-3	+3	+1	+0	-2	-2	+3	+5
PSVM 0.003	-1	-2	+0	+0	+0	+0	-2	-2	+0	+0
PSVM 0.005	-1	-1	-1	-1	-1	-2	-2	+0	-1	-2
PSVM 0.007	-1	+4	+0	-1	-2	-2	-1	+0	-1	+0
PSVM 0.009	+0	+4	+0	+0	-2	-2	-1	-1	+0	-2

El parámetro del coeficiente de rango de PSVM ha variado de 0.001 a 0.009 y el número de elementos base de PS-SVM de 100 a 1000. El objetivo es trabajar con valores que obtienen un tiempo de ejecución similar en ambos algoritmos y que obtienen suficiente reducción de la complejidad del problema para ser más rápidos que LibSVM.

La tabla 3.9 muestra el tiempo de ejecución de los clasificadores así como la precisión (tasa de acierto) y el tamaño del clasificador.

Cuando el tiempo de ejecución de utilizar LibSVM es demasiado alto, se puede utilizar PS-SVM y PSVM para alcanzar unas prestaciones competitivas en un periodo de tiempo mucho menor. PS-SVM presenta de nuevo mejor tiempo de ejecución que

Tabla 3.9: Precisión, Tamaño y Tiempo de Ejecución (s) en el conjunto MNIST

			Núcleos				
Algoritmo	Precisión	Tamaño	1	2	4	8	12
LIBSVM	99.59	6850	13598	7273	3882	2369	2106
PS-SVM 100	97.73	100	665	352	168	96	67
PS-SVM 200	98.52	200	1496	806	391	222	153
PS-SVM 300	98.86	300	2500	1252	688	389	292
PS-SVM 400	98.98	400	3261	1747	851	475	352
PS-SVM 500	99.07	500	4795	2489	1239	740	552
PS-SVM 1000	99.28	1000	10446	5471	2646	1818	1327
PSVM 0.001	95.92	10498	481	219	112	61	51
PSVM 0.003	97.49	11708	1910	959	510	326	368
PSVM 0.005	97.67	11842	6215	3138	1680	1111	1197
PSVM 0.007	97.93	14429	11695	5992	3441	2182	2282
PSVM 0.009	98.14	11562	17631	8919	4998	3265	3289

PSVM para obtener la misma precisión.

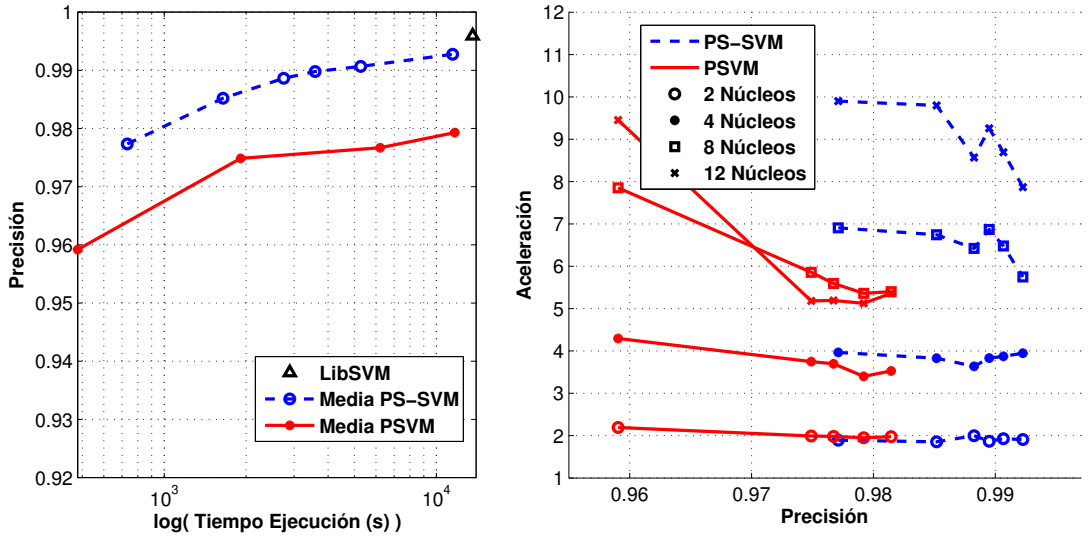
PS-SVM necesita calcular mediante SGMA los elementos base que va a utilizar una única vez y entonces ejecutar IRWLS para cada uno de los clasificadores "uno contra todos" lo que le da una ventaja adicional en problemas multiclase.

La Figura 3.9(a) muestra la precisión y la Figura 3.9(b) la aceleración en el conjunto de entrenamiento MNIST. PS-SVM obtiene mejor aceleración y converge a un buen resultado más rápido que PSVM.

3.5.2. P-GP y PS-GP

Para validar la utilidad de estas técnicas se han implementado también versiones paralelas de GPs y GPs dispersos. El objetivo es ver que estas técnicas no se limitan a la utilización de SVMs y pueden ser de utilidad en otras técnicas de aprendizaje automático.

Se han comparado estas implementaciones, P-GP para GPs completos y PS-GP para GPs dispersos, contra pyXGPR [Neumann et al., 2009], una librería de



(a) Precisión en función del Tiempo de Ejecución

(b) Aceleración en función de la Precisión

Figura 3.9: Resultados con el conjunto MNIST

python que contiene código para GPs. Para evaluar la precisión del algoritmo se ha utilizado el Error Cuadrático Media Normalizado (NMSE, "Normalized Mean Squared Error"), que para un conjunto de test $\mathbf{X} = \{\mathbf{x_1}, ..., \mathbf{x_s}\}$, cuyas etiquetas son $\mathbf{y} = \{y_1, ..., y_s\}$ y se han realizado las siguientes predicciones $\mathbf{m} = \{m_1, ..., m_n\}$ tiene la siguiente expresión:

$$NMSE = \frac{\frac{\sum_{i=1}^{s}(m_i - y_i)^2}{s}}{\frac{\sum_{i=1}^{s}(media(\mathbf{y}) - y_i)^2}{s}} \tag{3.3}$$

Se han utilizado tres conjuntos de entrenamiento de tamaños diferentes muy comunes en problemas de regresión:

- Boston Housing, del repositorio de la Universidad de California Irvine UCI [Asuncion and Newman, 2007]: Este conjunto de datos contiene valores de casas en los suburbios de Boston, está compuesto por 506 muestras con 14 atributos. Se han utilizado 300 muestras para entrenamiento y 206 para test.

- Abalone, del repositorio de la UCI [Asuncion and Newman, 2007]: El conjunto

contiene 4177 muestras y se ha dividido en dos grupos, 3000 datos para entrenamiento y 1177 para test. Cada muestra tiene 8 atributos, pero el género (hombre / mujer / niño) se ha dividido en tres atributos binarios (1,0,0),(0,1,0),(0,0,1).

- Cadata, del repositorio Statlib [Meyer and Vlachos, 2009]: Contiene 20640 datos con 8 atributos, se han utilizado 10000 datos en el entrenamiento y 10640 en el test.

Como paso inicial se han normalizado los datos reescalándolos para tener media 0 y varianza unidad. Se ha utilizado una función de covarianza Gaussiana y el valor final de los hiperparámetros (el parámetro de la función de núcleo γ y la varianza del ruido σ^2) se ha obtenido previamente utilizando como criterio de selección la maximización de la log evidencia.

- Boston Housing: A=229.76, γ=0.0343 and σ^2=7.7391.
- Abalone: A=10.8721, γ=0.0241 and σ^2=0.4244.
- cadata: A=0.4466, γ=0.7521 and σ^2=0.0179.

Las tablas 3.10, 3.11 y 3.12 muestran el NMSE para PS-GP, P-GP y pyXGPR y el tiempo de ejecución de P-GP y PS-GP cuando se utilizan diferentes tamaños en el modelo y utilizando diferentes núcleos del procesador.

Para ver la eficiencia de combinar modelos dispersos y paralelización, en el caso de PS-GP se ha utilizado el siguiente criterio para medir la aceleración:

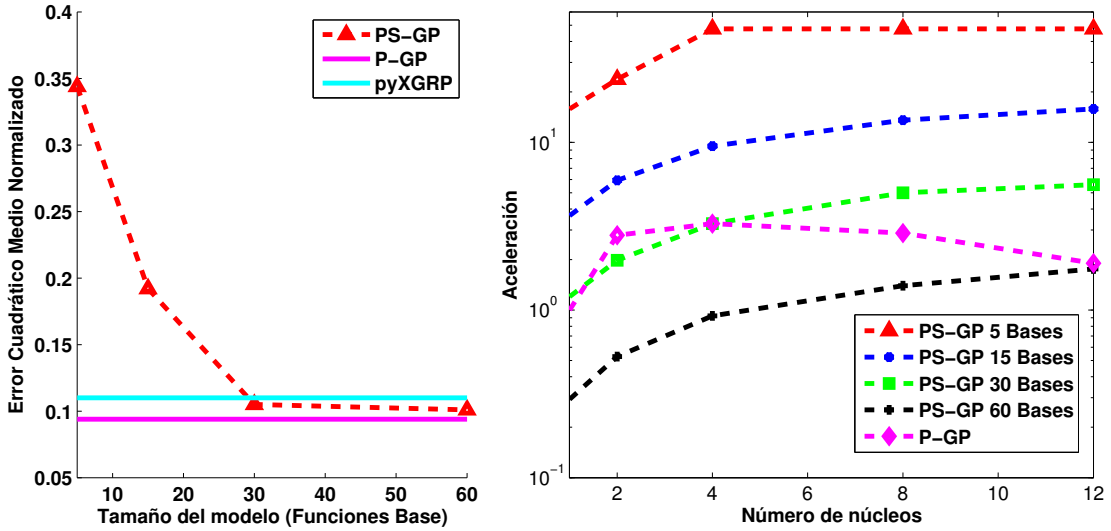
$$\text{Aceleración PS-GP} = \frac{\text{Tiempo de Ejecución P-GP Lineal}}{\text{Tiempo de Ejecución PS-GP Paralelo}} \tag{3.4}$$

En las Figuras 3.10, 3.11 y 3.12 se ha representado la aceleración para los algoritmos P-GP y PS-GP.

La aceleración está relacionada con el tamaño del conjunto de entrenamiento. Si las subtareas son demasiado pequeñas, el tiempo empleado por el procesador en lanzarlas afecta negativamente a los resultados. Como se puede ver en el caso de

Tabla 3.10: Tiempo de ejecución y NMSE con Boston Housing

Algoritmo	Tiempo de ejecución (ms)					NMSE
	1	2	4	8	12	
PS-GP$_{5\ \text{Bases}}$	6	4	2	2	2	0,344
PS-GP$_{15\ \text{Bases}}$	26	16	10	7	6	0,192
PS-GP$_{30\ \text{Bases}}$	79	48	29	19	17	**0,105**
PS-GP$_{60\ \text{Bases}}$	324	180	103	68	54	0,101
P-GP	95	34	29	33	50	**0,094**
pyXGPR	166					**0,11**



(a) NMSE (b) Aceleración

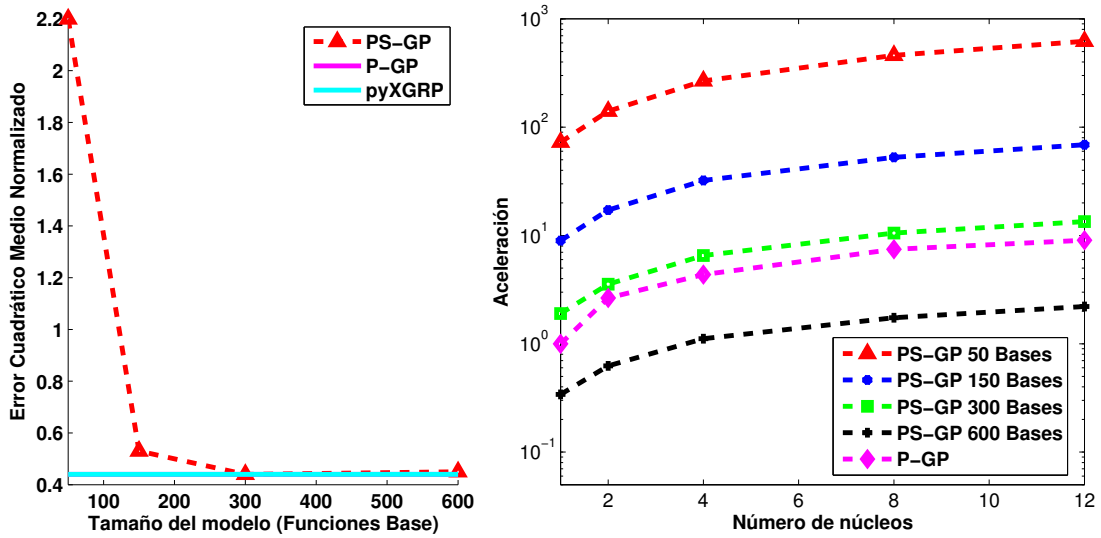
Figura 3.10: NMSE y aceleración utilizando el conjunto Boston Housing

Boston Housing, al estar compuesto de únicamente 300 datos, se obtiene la peor aceleración.

En el caso de Boston Housing y Abalone, utilizando PS-GP con un tamaño del modelo de un 10 % del tamaño del conjunto de entrenamiento se obtiene un NMSE muy cercano al obtenido por un GP completo y en el caso de cadata utilizando únicamente un 5 % de los datos el resultado ya son muy similares. Utilizando este tamaño, cuando comparamos el tiempo de ejecución de los modelos dispersos en paralelo uti-

Tabla 3.11: Tiempo de ejecución y NMSE con Abalone

Algoritmo	Tiempo de Ejecución(ms)					NMSE
	1	2	4	8	12	
PS-GP$_{50\ \text{Bases}}$	1229	637	334	194	144	2,2
PS-GP$_{150\ \text{Bases}}$	9977	5210	2772	1698	1298	0,53
PS-GP$_{300\ \text{Bases}}$	46948	25137	13691	8497	6650	**0,44**
PS-GP$_{600\ \text{Bases}}$	261962	142798	80132	51358	40506	0,45
P-GP	89378	33794	20521	11985	9898	**0,44**
pyXGPR	102983					**0,44**



(a) NMSE (b) Aceleración

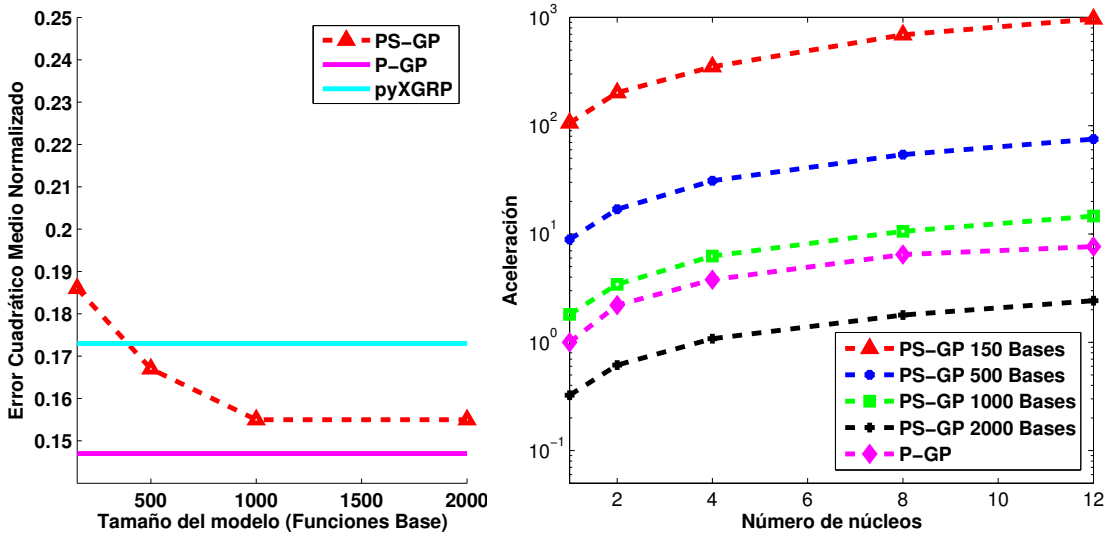
Figura 3.11: NMSE y aceleración utilizando Abalone

lizando 12 núcleos de procesamiento con el tiempo empleado por el modelo completo sin paralelizar, la aceleración que se alcanza es de 5.58 en el caso de Boston Housing, 13.44 en el caso de Abalone y 75.25 en el caso de cadata. Combinar paralelización con modelos dispersos parece ser una buena opción para poder ampliar el abanico de problemas que este tipo de técnicas pueden resolver.

El tiempo de ejecución, cuando se utiliza un único núcleo del procesador, es similar para pyXGPR y P-GP ya que son dos implementaciones diferentes del mismo

Tabla 3.12: Tiempo de ejecución y NMSE en cadata

Algoritmo	Tiempo de Ejecución (s)					NMSE
	1	2	4	8	12	
PS-GP$_{150\ \text{Bases}}$	29.21	15.35	8.82	4.48	3.2	0.186
PS-GP$_{500\ \text{Bases}}$	347	183	99.7	57.26	41.13	**0.167**
PS-GP$_{1000\ \text{Bases}}$	1708	903	495	292	211	0.155
PS-GP$_{2000\ \text{Bases}}$	9545	5026	2867	1733	1276	0.155
P-GP	3095	1405	819	479	404	**0.147**
pyXGPR	7396					**0.173**



(a) NMSE

(b) Aceleración

Figura 3.12: NMSE y aceleración utilizando el conjunto cadata

algoritmo. La principal ventaja de P-GP es la capacidad de reducir el tiempo de ejecución gracias a la paralelización. PS-GP presenta un mejor tiempo de ejecución cuando el número de funciones base utilizadas en el modelo es inferior al 20 % del tamaño del conjunto de entrenamiento.

3.6. Conclusiones

Se han propuesto varios algoritmos paralelos que trabajan con métodos de Núcleo. Uno de ellos destinado a resolver problemas de clasificación utilizando SVMs semi-paramétricas llamado PS-SVM y dos métodos cuyo fin es resolver problemas de regresión: una versión paralela de los Procesos Gaussianos (P-GP) y una implementación paralela del algoritmo SGEV para entrenar Procesos Gaussianos semi-paramétricos (PS-GP).

La idea de esta paralelización consiste en la partición iterativa de matrices en submatrices para poder dividir sus operaciones (producto, inversión) en diferentes subtareas que puedan realizarse simultáneamente por varios núcleos de procesamiento. Se han implementado los métodos mencionados utilizando OpenMP, una interfaz de programación para desarrollar código paralelo en entorno de memoria compartida. Se han realizado experimentos comparando los métodos propuestos con LibSVM, PSVM y pyXGPR y utilizando conjuntos de entrenamiento estándar.

Respecto a las SVMs, los resultados muestran que PS-SVM presenta una precisión igual o superior en clasificación para el mismo tiempo de ejecución y mejor aceleración en un rango mayor de valores de sus parámetros (número de elementos base) que PSVM (coeficiente de rango). En relación al tamaño del clasificador, PS-SVM, al ser un modelo semi-paramétrico, siempre produce modelos con un menor tamaño que PSVM.

También se ha mostrado que estos métodos pueden ser utilizados para paralelizar técnicas como GPs o GPs dispersos mejorando sus tiempo de ejecución. P-GP reduce el tiempo de ejecución utilizando paralelización y si el tiempo empleado aún es excesivo PS-GP puede reducirlo aún más utilizando un modelo semi-paramétrico para aproximar la solución.

Capítulo 4

Aumento de Eficiencia utilizando Factorización de Cholesky Paralela

"*Llegar juntos es el principio.*
Mantenerse juntos es el progreso.
Trabajar juntos es el éxito."
Henry Ford (1863 -1947)

Pese a que los modelos que han sido presentados en el apartado anterior obtenían buenas prestaciones en los resultados relativos a la aceleración, también presentaban algunas deficiencias que se han tratado de resolver en el trabajo presentado en este capítulo.

Por un lado, el cómputo de la inversa de una matriz utilizando la inversión por bloques puede presentar cierta inestabilidad cuando se trabaja con matrices que pese a ser invertibles son casi singulares al tener un determinante cercano a cero. Esta inestabilidad no se debe a la formulación, que es correcta, sino al error que se introduce al no ser posible tener números con precisión infinita en los ordenadores. En los experimentos realizados en el capítulo anterior la inestabilidad no era apreciable debido al limitado número de procesadores utilizado, pero es de esperar que a

medida que se aumente el número de núcleos de procesamiento empiecen a apreciarse ciertos errores de cálculo.

Por otro lado, si lo que se necesita es resolver un sistema de ecuaciones lineales de la forma $\mathbf{x} = \mathbf{A^{-1}b}$, existen soluciones más eficientes en términos de tiempo de ejecución que obtener la matriz inversa $\mathbf{A^{-1}}$. Un ejemplo de ello son las factorizaciones de matrices.

En este capítulo se presentan nuevos modelos [Díaz-Morales and Navia-Vázquez, 2016b] que solucionan estas deficiencias. PIRWLS que es una versión paralela de SVMs y PSIRWLS que es un modelos semi-paramétrico de SVMs.

Este Capítulo explica en la Sección 4.1 el nuevo esquema de paralelización. En la Sección 4.2 se describen los algoritmos que han sido seleccionados para su implementación en paralelo. Los experimentos realizados y los resultados obtenidos se detallan en la Sección 4.3. Para finalizar, en la sección 4.4 aparecen las conclusiones obtenidas con este trabajo.

4.1. Esquema Propuesto

4.1.1. Operaciones

Algunas operaciones, debido al orden de su coste computacional, juegan un papel muy importante a la hora de obtener soluciones eficientes (tal es el caso de los productos y sistemas lineales matriciales en el caso de las SVMs cuyo coste asociado es $\mathcal{O}(n^3)$).

Se ha trabajado en un entorno de memoria compartida, para paralelizar las operaciones se ha seguido un esquema quadtree [Samet, 1984], dividiendo iterativamente cada matriz en 4 submatrices y realizando las operaciones sobre dichas submatrices. Este esquema permite utilizar un número de núcleos de procesamiento que sea una potencia de dos.

Productos

Al utilizar un esquema quadtree (Fórmula 4.1), la paralelización de los productos es similar a la mostrada en el capítulo anterior, en este caso cada producto se divide en la obtención de 4 submatrices y dos subprocesos en el que cada uno obtendrá dos submatrices.

$$\begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} \\ \mathbf{A_3} & \mathbf{A_4} \end{bmatrix} = \begin{bmatrix} \mathbf{B_1} & \mathbf{B_2} \\ \mathbf{B_3} & \mathbf{B_4} \end{bmatrix} \begin{bmatrix} \mathbf{C_1} & \mathbf{C_2} \\ \mathbf{C_3} & \mathbf{C_4} \end{bmatrix} \tag{4.1}$$

$$\begin{aligned} &Subtareas: \\ &Subttarea\ 1 \begin{cases} \mathbf{A_1} = \mathbf{B_1C_1} + \mathbf{B_2C_3} \\ \mathbf{A_2} = \mathbf{B_1C_2} + \mathbf{B_2C_4} \end{cases} \\ &Subtarea\ 2 \begin{cases} \mathbf{A_3} = \mathbf{B_3C_1} + \mathbf{B_4C_3} \\ \mathbf{A_4} = \mathbf{B_3C_2} + \mathbf{B_4C_4} \end{cases} \end{aligned}$$

En el caso anterior, cada una de las cuatro subtareas tiene el mismo tiempo de ejecución. Podría suceder que una de las matrices envueltas en el producto fuese triangular. En este caso, las operaciones deben seleccionarse de tal manera que todas las tareas tengan el mismo tiempo de ejecución. Por ejemplo, en el caso de que la matriz $\mathbf{C}$ fuese triangular inferior, entonces $\mathbf{C_1}$ y $\mathbf{C_4}$ son también triangulares inferiores (sus productos tienen la mitad de operaciones) y $\mathbf{C_2} = \mathbf{0}$. Con lo cual, para obtener subtareas con el mismo tiempo de ejecución, se deben dividir de la siguiente forma:

$$Subtarea\ 1 \begin{cases} \mathbf{A_1} = \mathbf{B_1C_1} + \mathbf{B_2C_3} \\ \mathbf{A_2} = \mathbf{B_2C_4} \end{cases} \tag{4.2}$$

$$Subtarea\ 2 \begin{cases} \mathbf{A_3} = \mathbf{B_3C_1} + \mathbf{B_4C_3} \\ \mathbf{A_4} = \mathbf{B_4C_4} \end{cases} \tag{4.3}$$

Sumas y Restas

En el caso de Sumas y Restas de matrices la paralelización utilizando quadtrees es trivial y directa:

$$\begin{bmatrix} \mathbf{A_1} & \mathbf{A_2} \\ \mathbf{A_3} & \mathbf{A_4} \end{bmatrix} = \begin{bmatrix} \mathbf{B_1} & \mathbf{B_2} \\ \mathbf{B_3} & \mathbf{B_4} \end{bmatrix} + \begin{bmatrix} \mathbf{C_1} & \mathbf{C_2} \\ \mathbf{C_3} & \mathbf{C_4} \end{bmatrix} \tag{4.4}$$

$$\begin{aligned} &Subtareas: \\ &Subtarea\,1 \begin{cases} \mathbf{A_1} = \mathbf{B_1} + \mathbf{C_1} \\ \mathbf{A_2} = \mathbf{B_2} + \mathbf{C_2} \end{cases} \\ &Subtarea\,2 \begin{cases} \mathbf{A_3} = \mathbf{B_3} + \mathbf{C_3} \\ \mathbf{A_4} = \mathbf{B_4} + \mathbf{C_4} \end{cases} \end{aligned}$$

Inversión de Matriz Triangular

Otra operación con la que se tiene que trabajar es la inversa de una matriz triangular. Teniendo una matriz triangular $\mathbf{L}$, su inversa $\mathbf{L^{-1}}$ es también triangular y $\mathbb{I} = \mathbf{L}\mathbf{L^{-1}}$, subdividiendo las matrices obtenemos:

$$\begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbb{I} \end{bmatrix} = \begin{bmatrix} \mathbf{L_1} & \mathbf{0} \\ \mathbf{L_3} & \mathbf{L_4} \end{bmatrix} \begin{bmatrix} (\mathbf{L^{-1}})_\mathbf{1} & \mathbf{0} \\ (\mathbf{L^{-1}})_\mathbf{3} & (\mathbf{L^{-1}})_\mathbf{4} \end{bmatrix} \tag{4.5}$$

$$\begin{aligned} &Subtareas: \\ &Subtarea\,1 \begin{cases} (\mathbf{L^{-1}})_\mathbf{1} = {\mathbf{L_1}}^{-1} \end{cases} \\ &Subtarea\,2 \begin{cases} (\mathbf{L^{-1}})_\mathbf{4} = {\mathbf{L_4}}^{-1} \end{cases} \\ &(\mathbf{L^{-1}})_\mathbf{3} = -(\mathbf{L^{-1}})_\mathbf{4}\mathbf{L_3}(\mathbf{L^{-1}})_\mathbf{1} \Rightarrow \text{Paralelizable} \end{aligned} \tag{4.6}$$

En este caso, $(\mathbf{L^{-1}})_\mathbf{1}$ y $(\mathbf{L^{-1}})_\mathbf{4}$ se pueden obtener en paralelo y $(\mathbf{L^{-1}})_\mathbf{3}$ se puede después seguir ejecutando en paralelo utilizando el producto visto en la Sección 4.1.1.

4.1.2. Resolviendo el Sistema Lineal

Debido a que el procedimiento IRWLS resuelve un sistema de ecuaciones lineales diferente en cada iteración, no es computacionalmente eficiente invertir la matriz para obtener los pesos. Debido a que las matrices con las que trabajamos contienen la evaluación de funciones de núcleo para pares de elementos cuyos índices vienen dados por la fila y la columna de la matriz, estas matrices son definidas positivas.

La factorización de Cholesky es la descomposición de una matriz definida positiva en el producto de una matriz triangular inferior por su conjugada transpuesta. Es muy eficiente en su resolución, ya que es dos veces más rápida que la descomposición LU a la hora de resolver sistemas de ecuaciones lineales.

$$\mathbf{A} = \mathbf{L_A}\mathbf{L_A}^* \tag{4.7}$$

$$\mathbf{L_A} = Chol(\mathbf{A}) \tag{4.8}$$

Una vez que la factorización se ha realizado, es posible resolver muy eficientemente un sistema lineal utilizando sustitución hacia atrás sin necesidad de obtener la inversa de la matriz.

Para realizar la paralelización utilizando un esquema quadtree, se ha dividido la matriz en cuatro submatrices:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{L_1} & \mathbf{L_2} \\ \mathbf{L_3} & \mathbf{L_4} \end{bmatrix} \begin{bmatrix} \mathbf{L_1^*} & \mathbf{L_3^*} \\ \mathbf{L_2^*} & \mathbf{L_4^*} \end{bmatrix} \tag{4.9}$$

La descomposición obtiene una matriz triangular inferior, con lo que a partir de (4.9) sabemos que:

$$\mathbf{L_2} = \mathbf{0} \tag{4.10}$$

$$\mathbf{A} = \mathbf{L_1}\mathbf{L_1^*} \Rightarrow \mathbf{L_1} = Chol(\mathbf{A}) \tag{4.11}$$

$$\mathbf{C} = \mathbf{L_3}\mathbf{L_1^*} \Rightarrow \mathbf{L_3} = \mathbf{C}\mathbf{L_1^{*-1}} \tag{4.12}$$

$$\mathbf{D} = \mathbf{L_3}\mathbf{L_3^*} + \mathbf{L_4}\mathbf{L_4^*} \Rightarrow \mathbf{L_4} = Chol(\mathbf{D} - \mathbf{L_3}\mathbf{L_3}^*) \tag{4.13}$$

4.2. Algoritmos Desarrollados

Para demostrar la eficiencia y la mejora que suponen las técnicas descritas en este capítulo, se han desarrollado dos algoritmos:

- PIRWLS: Una implementación paralela SVMs utilizando el procedimiento IRWLS visto en la Sección 2.1.3 con un procedimiento externo que utiliza un grupo inactivo y otro activo de trabajo en cada iteración para tener la complejidad del procedimiento IRWLS bajo control [Pérez-Cruz et al., 2005] y dividiendo los datos del conjunto de entrenamiento en tres grupos [Osuna and Girosi, 1998], [Joachims, 1999], [Pérez-Cruz et al., 2001] dependiendo de si son vectores soporte fronterizos, vectores soporte no fronterizos o muestras correctamente clasificadas fuera del margen.

- PSIRWLS: Una implementación paralera de SVMs semi-paramétricas como la descrita en 2.1.4 que utiliza SGMA [Smola and Schölkopf, 2000] para obtener los centroides que utilizará la función de clasificación e IRWLS para obtener los pesos de cada centroide.

4.3. Experimentos

Ambos algoritmos, PIRWLS y PSIRWLS, han sido implementados en C utilizando OpenMP [Dagum and Enon, 1998] como entorno de desarrollo para la programación multiprocesador con memoria compartida. Para las operaciones algebraicas se

ha utilizado LAPACK (Linear Algebra PACKage), una librería estándar que tiene rutinas de algebra lineal.

Se han realizado experimentos para evaluar la precisión y la aceleración. Estos algoritmos han sido evaluados utilizando como referencia LibSVM y PS-SVM, dos algoritmos que también han sido implementados en C. El motivo de seleccionar PS-SVM es que resuelve el mismo problema cuadrático que PIRWLS pero utilizando una inversión por bloques paralela. En el caso de LibSVM es porque es la librería más extendida para resolver SVMs completas y se utiliza muy extensamente como referencia.

4.3.1. Reproducibilidad de los experimentos

Con el objetivo de facilitar la reproducibilidad de estos experimentos, se han tomado las siguiente medidas:

- El código fuente de ambos algoritmos tiene licencia de software libre y está disponible en un repositorio:
  - https://github.com/RobeDM/PIRWLS
  - https://github.com/RobeDM/PSIRWLS
- Se han utilizado servidores de alto rendimiento (HPC) disponibles en la plataforma Amazon EC2 [Juve et al., 2009]. Se ha seleccionado una máquina c4.8xlarge ya que las instancias c4 son ideales para aplicaciones de alto rendimiento y cómputo intensivo. Este servidor HPC tiene 18 núcleos de procesamiento y sus especificaciones se pueden ver en la Tabla 4.1.

4.3.2. Conjuntos de datos

Se han seleccionado cuatro conjuntos de datos diferentes que varían en tamaño y dimensión para medir las prestaciones en entornos diferentes:

Tabla 4.1: Arquitectura

Característica	Valor
Modelo	c4.8xlarge
Instrucciones	64-bit
Frecuencia Base	2.9 GHz
Procesadores	2 Intel Xeon E5-2666 v3
Núcleos Reales	18 (2 x 9)
Memoria	60 MB
Caché	25 MiB
Hyperthreading	Si (36 Núcleos Virtuales, pero 18 Reales)
Sistema Operativo	Linux Ubuntu

- SPLICE: Un problema de clasificación para reconocer dos clases de empalmes en una secuencia de ADN.
- ADULT: Del repositorio de la UCI [Asuncion and Newman, 2007]. El objetivo es predecir cuándo una familia tiene ingresos superiores a $50,000.
- MNIST (Mixed National Institute of Standards and Technology database): Es un conjunto de entrenamiento de dígitos escritos a mano. Contiene 10 clases pero se ha reducido a un problema de clasificación binario de dígitos pares e impares.
- COVTYPE: Es un conjunto de datos para predecir tipos de superficies forestales a partir de variables cartográficas. Contiene 6 clases, una de ellas se ha seleccionado para crear un problema de clasificación binario. El 90 % de los datos se ha utilizado como entrenamiento y el 10 % restante como test.

La Tabla 4.2 muestra la información de cada conjunto de entrenamiento utilizado en los experimentos.

4.3.3. Selección de Hiperparámetros

En el caso de Splice, Adult y MNIST los hiperparámetros se han seleccionado utilizando validación cruzada con 10 particiones. Se han explorado los siguientes

Tabla 4.2: Descripción de los Datos

	Número de Muestras			
	Entenamiento	Test	Clases	Características
Splice	2175	1000	2	60
Adult	32561	16281	2	123
MNIST	60000	10000	2	784
COVTYPE	522910	58102	2	54

valores:

$$\gamma = 10^n \qquad n : \text{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6}$$

$$C = 10^m \qquad m : \text{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6}$$

En el caso de COVTYPE, debido al alto tiempo de ejecución, los hiperparámetros se han seleccionado aleatoriamente. El objetivo es observar la aceleración en problemas de gran escala.

Debido a que la validación cruzada podría obtener diferentes valores de hiperparámetros en diferentes ejecuciones y por tanto una comparación injusta de los algoritmos en términos de tiempo, el mejor valor de hiperparámetros obtenido para LibSVM es el mismo que se ha utilizado para PIRWLS y el mejor valor obtenido para PS-SVM es el que se ha utilizado también con PSIRWLS. De esta forma tanto los dos modelos completos como los dos modelos semi-paramétricos utilizan los mismos valores de hiperparámetros.

El criterio de parada η de PIRWLS ha sido obtenido para obtener la misma precisión que LibSVM.

La Tabla 4.3 muestra los hiperparámetros utilizados en los experimentos.

4.3.4. Resultados

Se han ejecutado los algoritmos utilizando 1, 8 y 16 núcleos del procesador. Por un lado al tener operaciones matriciales que han sido diseñadas para dividir recursivamente las matrices en submatrices el mejor rendimiento se produce cuando el

Tabla 4.3: Hiperparámetros en SVMs completas y Semi-paramétricas

	Completa		Semi-paramétrica Tamaño 50		Semi-paramétrica Tamaño 500	
Conjunto	γ	C	γ	C	γ	C
SPLICE	10^{-2}	10^{1}	10^{-2}	10^{2}	10^{-2}	10^{2}
ADULT	10^{-3}	10^{3}	10^{-4}	10^{3}	10^{-3}	10^{5}
MNIST	10^{-2}	10^{2}	10^{-2}	10^{2}	10^{-2}	10^{5}
COVTYPE	10^{2}	10^{-1}	10^{-1}	10^{4}	10^{-1}	10^{4}

número de núcleos de procesamiento es una potencia de dos. Por otro lado siempre quedan dos núcleos libres reduciendo la posibilidad de que las tareas se ejecuten en núcleos que estén siendo utilizados por procesos del sistema operativo.

La Tabla 4.4 muestra la precisión, tiempo de ejecución y el tamaño del clasificador (número de vectores soporte) obtenidos con LibSVM. Este software, que implementa el algoritmo SMO es la librería más eficiente para resolver una SVM completa, por tanto se utilizará como referencia. Observando los resultados se puede observar el problema de la escalabilidad y del tamaño del clasificador.

Tabla 4.4: Resultados de LibSVM

	Precisión	Tiempo (s)	Tamaño
SPLICE	89 %	0,37	1053
ADULT	85,1 %	88	11429
MNIST	98,9 %	1008	5376
COVTYPE	92,5 %	18499	306143

Los resultados de PIRWLS se muestran en la Tabla 4.5. Se puede observar como el algoritmo alcanza la misma precisión que LibSVM. La aceleración del algoritmo utilizando 16 núcleos de procesamiento va desde 5.5x en un conjunto de datos muy pequeño como splice hasta 14x en conjuntos de datos más grandes.

LibSVM es más rápido que PIRWLS y especialmente eficiente en bases datos pequeñas, pero al paralelizar PIRWLS en bases de datos grandes PIRWLS supera enormemente a LibSVM en tiempo de ejecución.

Las Tablas 4.6 y 4.7 muestran los resultados de PS-SVM. Este algoritmo alcanza

Tabla 4.5: Resultados de PIRWLS

	Precisión	Tiempo (s)			Tamaño
		1 Núcleo	8 Núcleos	16 Núcleos	
SPLICE	88,8 %	7,6	1,7	1,4	1090
ADULT	85,1 %	287	41,6	27,0	11431
MNIST	98,9 %	2373	329	168	6048
COVTYPE	92,5 %	33763	4872	2751	305663

resultados competitivos utilizando un tamaño de clasificador menor al de las SVMs completas. Por ejemplo, en el caso del conjunto ADULT alcanza el mismo resultado utilizando únicamente 50 centroides y en el caso de SPLICE y MNIST alcanza un resultado comparable (aunque algo menor) utilizando únicamente 500 centroides. Este tipo de modelos se utilizan muy comúnmente en problemas de tiempo real donde el tiempo disponible para clasificar una nueva muestra es muy reducido.

Tabla 4.6: PS-SVM (50 Centroides)

	Precisión	Tiempo (s)			Tamaño
		1 Núcleo	8 Núcleos	16 Núcleos	
SPLICE	83,4 %	0,264	0,237	0,24	50
ADULT	85,1 %	18,3	3,3	1,9	50
MNIST	90,7 %	235	34,8	22,1	50
COVTYPE	75,7 %	357	61	43	50

Tabla 4.7: PS-SVM (500 Centroides)

	Precisión	Tiempo (s)			Tamaño
		1 Núcleo	8 Núcleos	16 Núcleos	
SPLICE	88,4 %	30,1	4,6	4,1	500
ADULT	85,1 %	750	131	111	500
MNIST	97,43 %	2778	526	351	500
COVTYPE	80,8 %	11028	2070	1540	500

Los resultados de PSIRWLS aparecen en las Tablas 4.8 y 4.9. En términos de precisión es equivalente a PS-SVM ya que ambos algoritmos minimizan la misma función de coste utilizando un enfoque similar. La principal diferencia es en términos

de cómo llevan a cabo la paralelización y cómo resuelven el sistema de mínimos cuadrados en cada iteración. PIRWLS es más rápido, el principal motivo es que la resolución de un sistema lineal utilizando una factorización de Cholesky es más rápida que utilizando una inversión por bloques. PSIRWLS también tiene una mejor aceleración, lo que lo hace todavía mejor en entornos paralelos.

Tabla 4.8: PSIRWLS (50 Centroides)

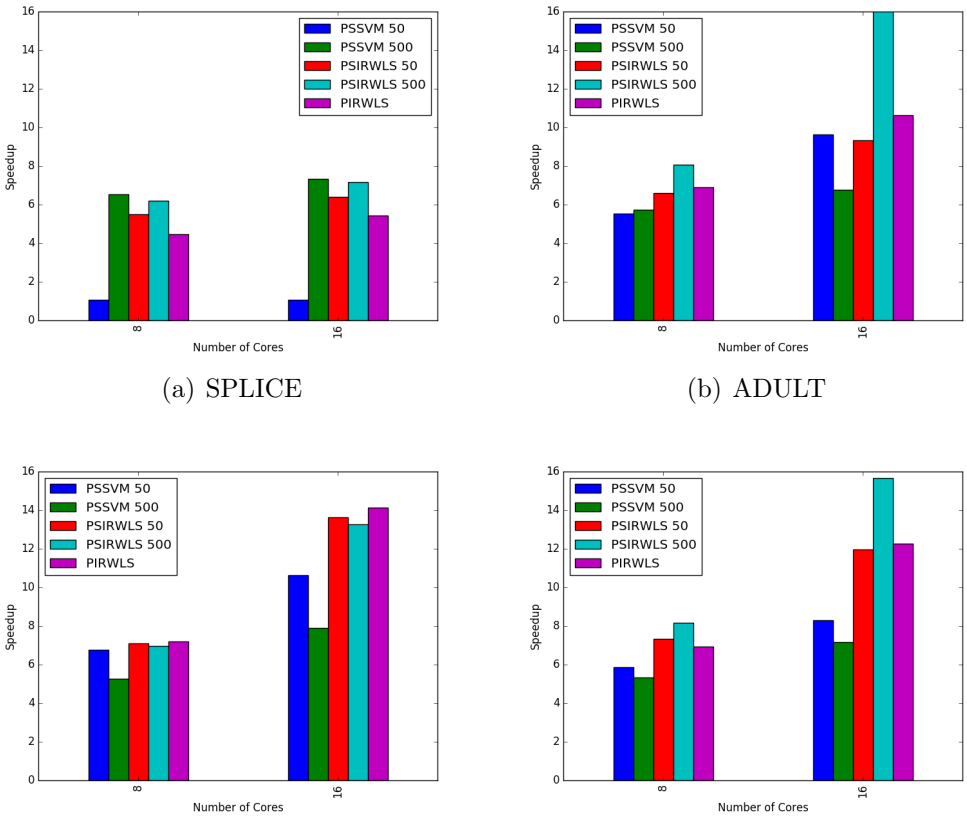
	Precisión	Tiempo (s)			Tamaño
		1 Núcleo	8 Núcleos	16 Núcleos	
SPLICE	83,4 %	0,77	0,14	0,12	50
ADULT	85,1 %	11,2	1,7	1,2	50
MNIST	90,7 %	139	19,6	10,2	50
COVTYPE	75,7 %	147	20	12,3	50

Tabla 4.9: PSIRWLS (500 Centroides)

	Precisión	Tiempo (s)			Tamaño
		1 Núcleo	8 Núcleos	16 Núcleos	
SPLICE	88,4 %	22,9	3,7	3,2	500
ADULT	85,1 %	589	72,9	36,4	500
MNIST	97,43 %	2243	322	169	500
COVTYPE	80,8 %	9009	1101	575	500

Las Figuras 4.1(a), 4.1(b), 4.1(c) y 4.1(d) muestran la aceleración de los algoritmos en cada conjunto de datos al utilizar 8 y 16 núcleos de procesamiento. Existen muchas variables que influyen en la aceleración, como el tamaño del conjunto de entrenamiento, el número de características o cómo de disperso es el conjunto de datos. Al observar los resultados, obviamente existe una fuerte correlación entre la aceleración y el tamaño de los conjuntos de entrenamiento, en el caso de SPLICE la aceleración se satura muy pronto alcanzando un valor máximo de 6x utilizando 16 núcleos de procesamiento.

A la hora de comparar los algoritmos, en el caso de modelos semi-paramétricos, PSIRWLS ha alcanzado mayor aceleración que PS-SVM en la mayoría de los casos.



(a) SPLICE

(b) ADULT

(c) MNIST

(d) COVTYPE

Figura 4.1: Aceleración en los diferentes conjuntos de datos

Se ha alcanzado una paralelización que roza la perfección (16x utilizando 16 núcleos de procesamiento) en el caso de PSIRWLS en el conjunto de datos ADULT y también en el caso de PIRWLS en el conjunto COVTYPE.

En los experimentos, excepto en el caso de SPLICE, la aceleración todavía no se ha visto saturada, lo que implica que se puede seguir mejorando si se incluyen más núcleos de procesamiento.

4.4. Conclusiones

En este capítulo se proponen dos nuevas implementaciones paralelas de SVMs para sistemas multinúcleo y multiprocesador. El primero de los algoritmos, llamado PIRWLS, resuelve una SVM completa y el segundo, PSIRWLS, es un modelo semi-paramétrico de SVM. Ambos métodos minimizan la función de coste mediante un procedimiento IRWLS que se paraleliza utilizando una factorización de Cholesky paralela, un método eficiente de resolver sistemas de ecuaciones lineales con matrices definidas positivas.

Los métodos propuestos han sido ejecutados utilizando conjuntos de entrenamiento estándar y se ha medido contra LibSVM, la librería más extendida de SVMs y PS-SVM el modelo semi-paramétrico de SVMs propuesto en el capítulo anterior que hace uso de la inversión paralela por bloques.

En lo referente a los modelos semi-paramétricos, PSIRWLS ha sido más rápido y ha obtenido mayor aceleración que PS-SVM y ambos obtienen una precisión similar a la de la SVM completa en un menor tiempo de ejecución. Cuando comparamos los modelos completos de SVMs, aunque LibSVM es más rápido utilizando un único núcleo de procesamiento, las buenas capacidades de paralelización de PIRWLS hacen que sea más rápido.

Estos métodos han sido implementados utilizando OpenMP, una interfaz de programación para sistemas multinúcleo con memoria compartida. Estos tipos de sistemas multiprocesador están muy extendidos y son la tendencia dominante en el desa-

rrollo de microprocesadores. Esto hace que PIRWLS y PSIRWLS sean muy prácticos ya que cada ordenador puede beneficiarse de ellos.

Capítulo 5

Otras contribuciones

"A través de la perseverancia mucha gente alcanza el éxito desde lo que parecía destinado a ser un fracaso seguro"

Benajamin Disraeli (1804 -1881)

Una de las partes más satisfactorias tras todo aprendizaje académico es comprobar que los conocimientos adquiridos son de utilidad para solucionar problemas prácticos. Los conocimientos sobre paralelización adquiridos durante el desarrollo de esta tesis doctoral han influido de forma indirecta en otras contribuciones. Estos trabajos se corresponden con publicaciones y premios recibidos a raíz de competiciones de aprendizaje automático.

5.1. Competición: Bosón de Higgs

La Competición de Aprendizaje Automático sobre el Bosón de Higgs [HiggsML, 2014], se organizó con el objetivo de fomentar la colaboración entre físicos de partículas de alta energía y científicos de datos, el objetivo era explorar el potencial que tienen los métodos de aprendizaje automático para mejorar el análisis de los datos producidos por el experimento ATLAS [Aad et al., 2012] [Chatrchyan et al., 2012].

Esta competición fue alojada por Kaggle y tuvo lugar entre del 12 de mayo al 15 de septiembre de 2014 y tuvo un gran éxito ya que reunió a 1785 competidores.

5.1.1. El problema

El propósito de la física de alta energía es descubrir la estructura de la materia estudiando sus partículas. Actualmente se centra en el estudio de partículas subatómicas que se producen mediante colisiones en aceleradores. Procesar de forma rápida y correcta la información generada en estos aceleradores es crítico debido al coste de los experimentos y a que se produce una gran cantidad de datos pero la mayoría de ellos irrelevantes. Por esta razón, el campo de la clasificación de eventos utilizando aprendizaje automático es un línea de investigación muy importante.

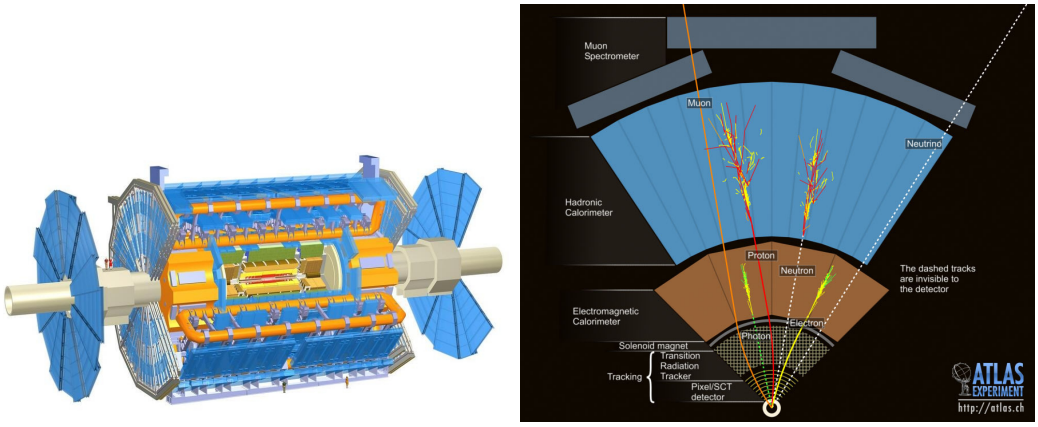
La desintegración de partículas es el proceso mediante el cual partículas elementales se transforman en otras partículas elementales. Existen muchos procesos en los cuales un Bosón de Higgs puede desintegrarse produciendo otras partículas. El experimento ATLAS ha observado señales de desintegraciones del Bosón de Higgs en dos partículas tau, pero esta señal es muy pequeña y se produce en un ambiente muy ruidoso.

Los datos han sido producidos por el simulador oficial del detector ATLAS y contiene dos tipos de eventos: "señal" si se producía una desintegración de un Bosón de Higgs, y "ruido" producidos por la desintegración de otros tipos de partículas.

El objetivo del campeonato era, dado un espacio de características para los eventos, encontrar la región donde la cantidad de los eventos de señal era mucho más significativa que la de ruido. Para ello se aplica un test estadístico a la región para determinar la significancia de dicho exceso.

5.1.2. El conjunto de datos y Objetivo

Debido a la complejidad del proceso de simulación y a lo desbalanceado que es el problema, cada evento dispone de un peso positivo w_i de tal forma que la suma de



(a) El detector Atlas (b) Un evento

Figura 5.1: El Experimento Atlas

pesos en una región es un estimador insesgado de la esperanza del número de eventos que caen en esa región durante un intervalo de tiempo fijo. Estos pesos solo venían dados para el conjunto de entrenamiento, con lo que no forman parte de la entrada del clasificador.

$$\begin{aligned} &\mathcal{D} = \{(\mathbf{x}_1, y_1, w_1), ..., (\mathbf{x}_n, y_n, w_n)\}, \\ &\mathbf{x}_i \in \mathbb{R}^d, \\ &y_i \in \{r, s\}, \\ &w_i \in \mathbb{R}^+. \end{aligned} \tag{5.1}$$

El número de características d es 30 (17 primitivas medidas por el detector y 13 obtenidas a partir de las primitivas). La descripción de estas características se puede ver en la documentación oficial del campeonato [Adam-Bourdarios et al., 2015]. El objetivo del campeonato era obtener el clasificador que obtuviese la mayor Significancia Media Aproximada (AMS, "Approximate Median Significance").

$$AMS = \sqrt{2\left((s_T + r_T + r_r)log\left(1 + \frac{s_T}{r_T + r_r}\right) - s_T\right)},$$
$$s_T = \sum_{i=1}^{n} w_i 1\{y_i = s\}1\{\hat{y}_i = s\},$$
$$r_T = \sum_{i=1}^{n} w_i 1\{y_i = r\}1\{\hat{y}_i = s\}. \tag{5.2}$$

Donde s_T y r_T son la suma de los pesos de los eventos verdaderos positivos y los falsos negativos respectivamente, $\hat{y}_i$ es la predicción obtenida por el clasificador y r_r es un término de regularización que premia la selección de regiones del espacio más grandes (para este campeonato se utilizaba un valor de 10).

5.1.3. Esquema General Multinúcleo

Debido al tamaño de este conjunto de datos y al limitado tiempo del campeonato, la utilización de algoritmos y técnicas capaces de ser ejecutados de forma paralela en entornos multiprocesador y multinúcleo se hacía indispensable para poder obtener un clasificador con buenas prestaciones en un espacio de tiempo razonable.

Para ello se diseño una estrategia en la que tanto el preprocesado de los datos, los algoritmos de aprendizaje automático y el postprocesado se podía realizar en un entorno multinúcleo.

- En primer lugar se descartó un esquema distribuido ya que requiere la creación y configuración de una red destinada a este fin. La utilización de hardware como GPUs también se descartó ya por un lado no dispone de memoria suficiente para trabajar con el volumen de datos que se requería en este problema y por otro la solución estaría limitada a plataformas que tuviesen este tipo de dispositivos. Por estos motivos se optó por una paralelización multiprocesador y multinúcleo, ya que es universal para cualquier dispositivo.

- La extracción de características se llevó a cabo siguiendo un patrón de diseño de estilo map-reduce, donde el preprocesado de cada dato se realizaba de forma

independiente y se podía dividir entre las diferentes unidades de procesamiento.

- Para evitar un consumo innecesario de memoria RAM se optó por un esquema de memoria compartida donde todos los hilos de procesamiento leen de la misma copia de los datos.

- Se utilizaron algoritmos de aprendizaje automático que pudiesen ser ejecutados en un entorno multiprocesador y multinúcleo.

Preprocesado

Cuando se mira el marco de referencia (Figura 5.2), se puede deducir que la distribución de los eventos es invariante a una rotación del eje **z**. Se ha aplicado este conocimiento a priori para transformar el espacio de entrada, esto permite crear un modelo que sea robusto a rotaciones de los eventos alrededor del eje **z** y que pueda aprender la naturaleza del problema utilizando un menor número de muestras ya que no tiene que aprender las relaciones entre las variables de entrada que provocan esta invarianza.

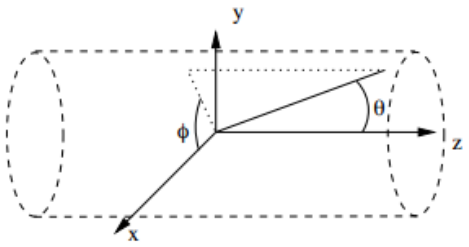
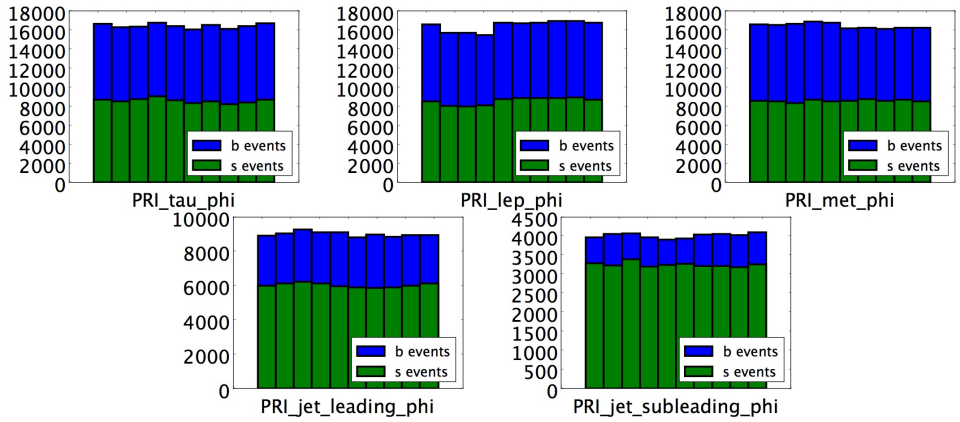


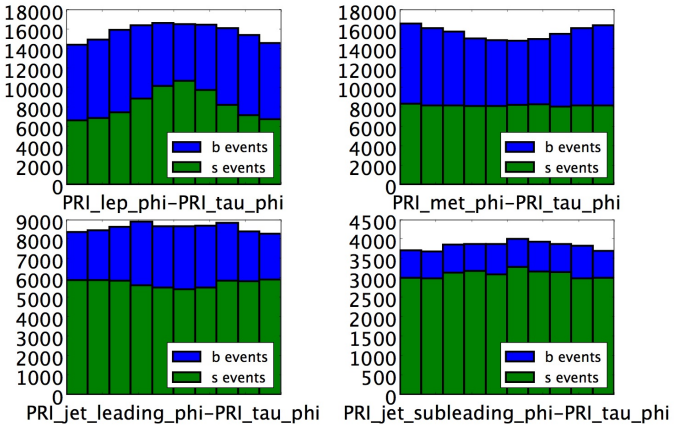
Figura 5.2: Marco de Referencia

Este cambio de variable permitía además reducir la dimensionalidad. En las Figuras 5.3(a) y 5.3(b) se aprecia el efecto que tiene en las variables de entrada dicha transformación.

En esta base de datos los eventos también podían tener un valor desconocido para algunas variables. Para poder trabajar con ellos y no perder información, se imputó



(a) Histograma de las variables relacionadas con el ángulo ϕ en el marco de referencia (ver documentación oficial [Adam-Bourdarios et al., 2015])



(b) Histograma tras el cambio de variable

Figura 5.3: Cambio de variables para crear un espacio de características robusto a rotaciones en el eje z

cada valor que faltaba a la media y se creó una variable adicional binaria indicando si se había realizado una imputación o no.

Pese a no ser un factor decisivo, ya que el tiempo de ejecución de esta etapa no era elevado respecto al tiempo de entrenamiento de los distintos modelos, se realizó una implementación paralela multinúcleo distribuyendo los datos a procesar entre

los diferentes núcleos de procesamiento.

Expansión del espacio de características

Se construyeron varios conjuntos de entrenamiento utilizando diferentes expansiones polinónicas. Dichos conjuntos contenían nuevas características que consistían en el producto, cociente, suma y resta de cada par de características.

Selección de algoritmos

Se seleccionaron algoritmos que se pudiesen ejecutar en un entorno multinúcleo y que pudiesen manejar una gran cantidad de datos en un tiempo razonable.

- Boosting por Gradiente: Utilizando árboles de decisión como predictores débiles. Se utilizó el software XGBoost [Chen, 2014], que permite ejecución multinúcleo.

- Máquinas de Vectores Soporte: Se utilizó por un lado la versión de sklearn [Pedregosa et al., 2011] utilizando un kernel lineal y descenso por gradiente estocástico y por otro nuestra versión con función de núcleo Gaussiano desarrollada en el capítulo 4 de esta tesis.

- Redes Neuronales: Se utilizó un perceptrón multicapa utilizando la cross-entropía como función de coste. Por las limitaciones del tiempo de ejecución se utilizó una única capa oculta. Se utilizó una implementación que utilizaba Theano [Bergstra et al., 2010] para poder ejecutarse en un entorno multinúcleo.

- Bosques Aleatorios: Operan construyendo una multitud de arboles de decisión. Para esta versión se utilizó la versión de la librería de python sklearn [Pedregosa et al., 2011].

Criterio de entrenamiento de cada algoritmo

La Característica Operativa del Receptor (ROC, "Receiver Operating Characteristic") es una técnica muy común para evaluar modelos y seleccionar parámetros que ha sido utilizada originalmente en la teoría de detección de señales y después se extendió al área de aprendizaje automático. Es una representación gráfica de la tasa de positivos verdaderos (TPR, "True Positive Rate") en función de la tasa de falsos positivos (FPR, "False Positive Rate"). La curva se obtiene variando el umbral de clasificación desde el valor más alto al más bajo. Ilustra de forma muy intuitiva la precisión de un clasificador binario.

El área bajo la curva (AUC, "Área Under the Curve") es el área bajo la curva ROC, que es 1.0 en el caso de un clasificador perfecto y 0.5 en el caso de una clasificación totalmente aleatoria. Como se muestra en [Ling et al., 2003] es estadísticamente consistente y una medida más discriminativa que la tasa de acierto.

La AMS depende de la suma de pesos de los eventos reales positivos (s_T) y la suma de los pesos de los eventos falsos positivos (r_T). Un incremento de s_T o una reducción de b_T, manteniendo el otro valor constante, conllevan un valor más alto de AMS.

Como medida de calidad de un clasificador se ha seleccionado un AUC ponderado (WAUC, 'Weighted AUC"), que tiene en cuenta los pesos de los eventos en la evaluación del TPR y FPR, llamados ahora WTPR y WFPR:

$$WTPR = \frac{\sum_{i=1}^{n} w_i 1\{y_i = s\} 1\{\hat{y}_i = s\}}{\sum_{i=1}^{n} w_i 1\{y_i = s\}}, \tag{5.3}$$

$$WFPR = \frac{\sum_{i=1}^{n} w_i 1\{y_i = r\} 1\{\hat{y}_i = s\}}{\sum_{i=1}^{n} w_i 1\{y_i = r\}}. \tag{5.4}$$

De hecho el WTPR y el WFPR es el valor de b_T y s_T normalizados, así que el WAUC parece un valor razonable de medida de calidad para este problema:

$$WTPR = \frac{s_T}{max(s_T)}, \tag{5.5}$$

$$WFPR = \frac{b_T}{max(b_T)}. \tag{5.6}$$

La forma tradicional de llevar a cabo la búsqueda de los hiperparámetros es mediante una búsqueda en rejilla, que consiste en una especificación manual de los posibles valores que puede tener cada hiperparámetro y se exploran todas las posibles combinaciones de ellos utilizando alguna técnica de validación. Debido al excesivo tiempo de ejecución que llevaría esta búsqueda para esta tarea se escogió una estrategia de búsqueda lineal, que es subóptima pero que conllevaba un coste mucho menor.

1. Inicialización: Se define el conjunto de posibles valores manualmente. El valor inicial se selecciona aleatoriamente entre los candidatos.

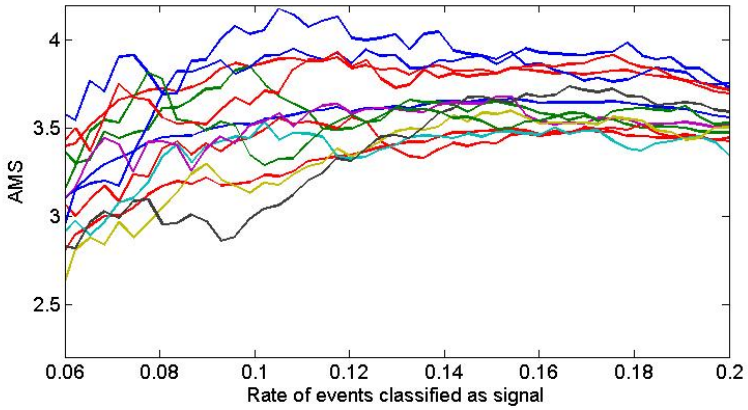
2. Optimización: Se selecciona un parámetro de forma iterativa siguiendo un orden secuencial. Se explora el valor anterior y el siguiente utilizando validación cruzada con 10 particiones como técnica de validación.

3. Criterio de parada: Si la WAUC no ha mejorado durante un número de iteraciones igual al número de hiperparámetros se termina, en caso contrario se vuelve al paso 2.

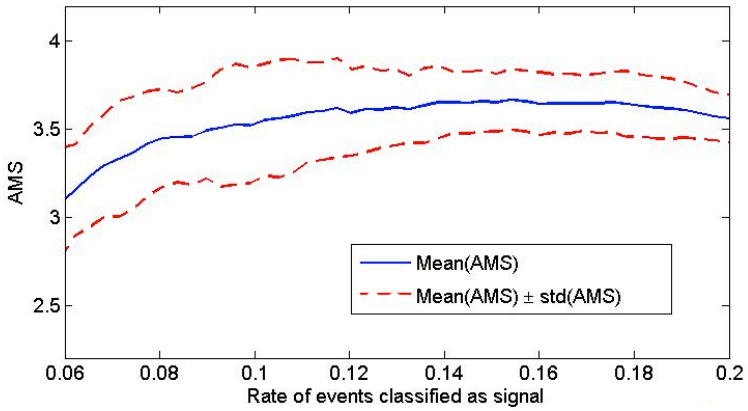
Bootstrap Aggregating

Cuando se empezó a trabajar con este conjunto de datos, se observó que había mucha inestabilidad en los resultados. Pequeños cambios en el valor de los hiperparámetros o eliminar un pequeño número de observaciones podía dar lugar a resultados muy diferentes en la función de evaluación. Para ilustrar esta inestabilidad se puede ver en la Figura 5.4(a) el valor de la AMS en función del umbral de clasificación cuando se utilizada el algoritmo de Boosting por Gradiente en las 10 particiones de la validación cruzada. Los resultados difieren significativamente en cada partición, superando 4.0 en el mejor de los casos y no llegando a 3.0 en el peor caso. Mirando

la Figura 5.4(b) se puede deducir que un umbral de clasificación que seleccione del 0.14 al 0.16 de los eventos parece un valor razonable.



(a) AMS en las diferentes particiones de la validación cruzada



(b) Media AMS and y Media ± Desviación Típica

Figura 5.4: AMS en función de la tasa de eventos clasificados como señal

Bootstrap Aggregating, también llamado Bagging [Breiman, 1996] es una técnica muy común que permite mejorar la estabilidad así como prevenir el sobreajuste dando lugar a una mejora en la precisión. Teniendo un conjunto de entrenamiento $\mathbf{X}$ es posible generar diferentes conjuntos de entrenamiento tomando muestras uniformemente de $\mathbf{X}$ con reemplazo. Para cada conjunto de entrenamiento se puede construir un modelo y combinar los resultados promediando las salidas. Resultados

experimentales y teóricos muestran que esta técnica puede acercar procedimientos inestables hacia el óptimo.

Combinación

Los conjuntos de máquinas de aprendizaje utilizan la salida de múltiples clasificadores para obtener mejores prestaciones. Empíricamente este tipo de técnicas obtienen mejores resultados cuando existe diversidad entre los clasificadores. La explicación estadística es que cuando no se tienen suficientes datos en una región del espacio de características un algoritmo puede encontrar diferentes hipótesis con las mismas prestaciones sobre el conjunto de entrenamiento. Si diferentes modelos utilizan criterios diferentes, se reduce el tiesto de elegir la hipótesis incorrecta.

Otro motivo de utilizarlos es que los algoritmos pueden converger a un óptimo local. Al utilizar diferentes criterios para encontrar un objetivo, esta posibilidad se reduce.

Pese a las ventajas que ofrecen este tipo de técnicas también tenemos el inconveniente de que este tipo de técnicas, al aumentar la flexibilidad, se puede realizar un sobreajuste más fácilmente. Para evitar este problema se ha elegido un modelo sencillo que consiste en una combinación lineal de las salidas de los algoritmos:

$$f(\mathbf{x_i}) = \sum_{a=1}^{l} \alpha_a f_a(\mathbf{x_i}) \underset{0}{\overset{1}{\gtrless}} \eta. \tag{5.7}$$

5.1.4. Resultados

La competición se evaluaba utilizando un conjunto de test de 550000 eventos cuyas etiquetas eran desconocidas. Durante el campeonato la clasificación se realizaba evaluando un 18 % del conjunto de test (llamada clasificación pública). Después de la competición, el resultado final se hacía evaluando únicamente el 82 % restante (clasificación privada).

Se construyeron varios clasificadores combinando diferentes algoritmos. Al final se seleccionó la solución que obtuvo el mejor AMS en la clasificación pública, que

consistía en la combinación de tres modelos entrenados con Gradient Boosting sobre tres de los conjuntos de entrenamiento generados. Este modelo alcanzó un AMS de 3,76 en la clasificación privada finalizando en la posición 9º de 1785 equipos.

Una descripción más detallada de la solución puede encontrarse en [Dıaz-Morales and Navia-Vázquez, 2015].

5.2. Competición: Rastreo entre Dispositivos

Esta competición, llamada "ICDM 2015: Drawbridge Cross-Device Challenge" [ICDM, 2015], fue organizada por el Congreso Internacional en Minería de Datos (ICDM, "International Conference on Data Mining"), patrocinada por la empresa Drawbridge y alojada en Kaggle. Tuvo lugar entre el 1 de Mayo y el 24 de Agosto del 2015 y reunió 340 equipos.

5.2.1. El problema

En los últimos años los hábitos de consumo de Internet por parte de los usuarios ha cambiado muchísimo. Si en un principio el ordenador era el único dispositivo con acceso a internet y estaba ubicado en un lugar fijo, la aparición de nuevos dispositivos que disponen de acceso a la red como los portátiles, teléfonos móviles, televisión "inteligente" e incluso pequeños dispositivos como relojes ha hecho que los usuarios puedan acceder tanto desde cualquier sitio como de forma simultánea utilizando varios dispositivos.

Ejemplos de estos nuevos hábitos son el ver una película utilizando una SmartTV mientras se opina sobre ella con el teléfono móvil, o hojear el catalogo de una tienda desde el teléfono y comprar algo al llegar a casa desde un ordenador portátil. Los datos para entender este comportamiento están muy fragmentados y la identificación de los usuarios supone un nuevo reto.

El rastreo de usuarios entre dispositivos ("Cross-Device Tracking") es el problema de saber si la persona que utiliza el ordenador A es la misma que utiliza el teléfono

Figura 5.5: Nuevos hábitos de acceso

móvil B o la tablet C. Es una tecnología emergente que tiene un gran auge últimamente debido a que puede ser utilizada para mostrar publicidad más eficientemente, algo muy valioso en el marketing de Internet.

Actualmente existen tres tipos de formas de abordar el problema:

- Grandes compañías ofrecen el servicio de identificar a personas que han iniciado una sesión en sus servicios (Google, Facebook,...). Simple, pero es un requerimiento que no se cumple en muchos casos.

- Utilizar información determinista y reglas de emparejamiento exactas (por ejemplo, si ve el mismo número de tarjeta de crédito en un formulario rellenado desde una tablet y desde un teléfono móvil se puede inferir que es muy probable que pertenezcan a la misma persona), se puede aplicar en un número muy reducido de situaciones.

- Utilizando aprendizaje automático para crear un modelo predictivo. Lo que es aplicable a cualquier situación y permite incluir cualquier tipo de variable informativa pero existe un gran desconocimiento acerca de qué variables y algoritmos pueden obtener un buen resultado.

5.2.2. El conjunto de datos y Objetivo

Dado datos de uso, el objetivo consistía en determinar que cookies pertenecían a un individuo individual utilizando un dispositivo.

El conjunto de datos consistía en una base de datos relacional que contenía información sobre dispositivos, cookies, direcciones IP y comportamiento:

- Existían tablas con información de alto nivel acerca de dispositivos y cookies.
- Otra tabla describía el comportamiento conjunto de dispositivos y cookies en las direcciones IP.
- También contenía tablas que describían propiedades y categorías sobre las páginas web visitadas y las aplicaciones móviles utilizadas.

El objetivo consistía en obtener el clasificador con mayor puntuación $\mathcal{F}_{0,5}$. Esta métrica, utilizada muy comúnmente en el campo de la recuperación de información tiene en cuenta la precisión p y la memoria r.

La precisión es la tasa de verdaderos positivos (tp) de entre todos los valores devueltos como positivos, tanto verdaderos positivos como falsos positivos ($tp + fp$). La memoria es la tasa de verdaderos positivos de entre todos los positivos, tanto verdaderos positivos como falsos negativos ($tp + fn$).

$$\begin{aligned} \mathcal{F}_\beta &= (1+\beta^2)\frac{pr}{\beta^2 p + r}, \\ p &= \frac{tp}{tp+fp}, \\ r &= \frac{tp}{tp+fn} \end{aligned} \tag{5.8}$$

Utilizando $\beta = 0{,}5$ la puntuación pondera la precisión más que la memoria. La puntuación se forma promediando los $\mathcal{F}_{0,5}$ individuales obtenidos para cada dispositivo del conjunto de test.

5.2.3. Esquema general Multinúcleo

Al contrario de lo que ocurría en el problema anterior, en este caso el principal coste computacional residía en la extracción de características, que debía llevarse a

cabo a partir de una base de datos relacional que tiene una representación en forma de grafo.

Se tomaron los siguientes criterios para poder llevar a cabo los cálculos en paralelo:

- Se escogió un esquema multiprocesador y multinúcleo. Se descartó un esquema distribuido ya que requiere la creación y configuración de una red destinada a este fin y también se descartó una paralelización utilizando hardware específico como GPUs ya que por un lado no disponen de memoria suficiente para albergar el grafo y por otro la solución estaría limitada a plataformas que tuviesen este tipo de dispositivos.

- La extracción de características se llevo a cabo siguiendo un patrón de diseño de estilo map-reduce, donde se extraía la información de cada muestra de forma independiente y se podían distribuir entre las diferentes unidades de procesamiento.

- Por el gran consumo de memoria que suponía el tener el grafo alojado en la RAM, se utilizó un entorno de memoria compartida para tener una única copia del grafo para todos los procesadores.

- Se utilizaron algoritmos de aprendizaje automático que pudiesen ser ejecutados en un entorno multiprocesador y multinúcleo.

Preprocesado

Debido al elevado número de dispositivos y cookies era inabordable tener en cuenta cada posible combinación de dispositivo/cookie. Por ello se crearon unas reglas básicas para seleccionar un subconjunto de cookies candidatas para cada dispositivo. Las reglas se basaban en la las direcciones IP que ambos compartían y cómo de frecuentes eran en otras cookies.

El proceso de selección de candidatos se realizaba siguiendo el siguiente proceso:

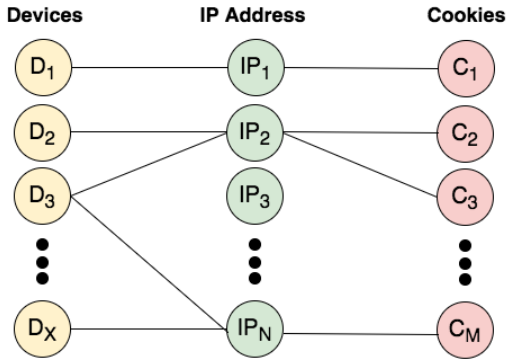


Figura 5.6: Ejemplo de relaciónes Dispositivos-Cookies a través de direcciones IP

1. Inicialización: Cada dispositivo se iniciativa con un conjunto vacío de cookies candidatas.

2. Si una dirección IP aparece en menos de A dispositivos y B cookies entonces todas las cookies en las que aparece son es incluyen en el conjunto de todos sus dispositivos.

3. Para los dispositivos cuyo conjunto de candidatos permanece vacío se incrementan los valores de A y B y se vuelve al paso 2.

4. Para cada cookie candidata, se incluyen en el conjunto el resto de cookies que comparten su mismo handle ID.

Utilizando esta selección inicial de candidatos, en el 98.3 % de los dispositivos el conjunto contenía las cookies auténticas.

Creación del Conjunto de Entrenamiento

En el conjunto de entrenamiento, cada muestra representa una pareja dispositivo/cookie candidata y se compone de 67 características. Existen tres tipos de características:

- Información del dispositivo (Sistema Operativo, País, ...)

- Información de la cookie (Versión del Navegador, País, ...)
- Información relacional (Número de direcciones IP en común, ...)

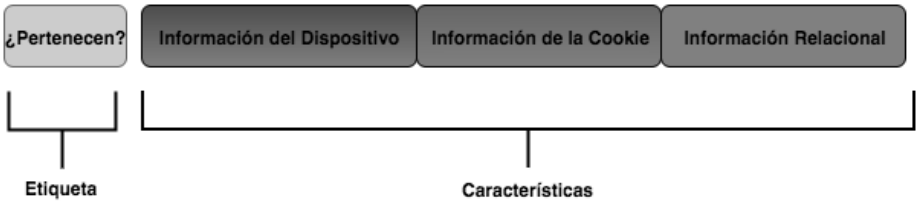


Figura 5.7: Estructura de una muestra del conjunto de entrenamiento

Algoritmo

Tras hacer pruebas con varios algoritmos paralelos el algoritmo que mostró superioridad fue Boosting por Gradiente [Chen, 2014]. El motivo es que al utilizar como predictores débiles arboles de decisión puede manejar fácilmente características muy heterogéneas y modela la no linealidad del problema de forma muy rápida. En el entrenamiento se utilizaron los siguientes criterios:

- Se utilizó en la función de coste una pérdida logarítmica para que la salida representase la probabilidad de que el par dispositivo-cookie estuviese asociado.
- Se utilizó Bagging para mejorar la estabilidad.
- La selección de hiperparámetros se llevo a cabo mediante validación cruzada utilizando 10 particiones.

Entrenamiento Semi-supervisado

El entrenamiento semi-supervisado hace uso de datos no etiquetados. En los casos donde el primer candidato obtenía una puntuación alta y el segundo una muy baja se observó que la probabilidad de que el dispositivo y la primera cookie estuviesen asociados era extremadamente alta.

La Figura 5.8 muestra el $\mathcal{F}_{0,5}$ y el porcentaje de dispositivos que se utiliza en su cálculo cuando se tienen en cuenta únicamente los dispositivos cuyo segundo candidato no supera cierto umbral. Se puede ver que para cuando el segundo candidato obtiene una probabilidad inferior a 0.1 de pertenecer al dispositivo la $\mathcal{F}_{0,5}$ media es superior a 0.99 y esta condición se cumple en el 62 % de los dispositivos.

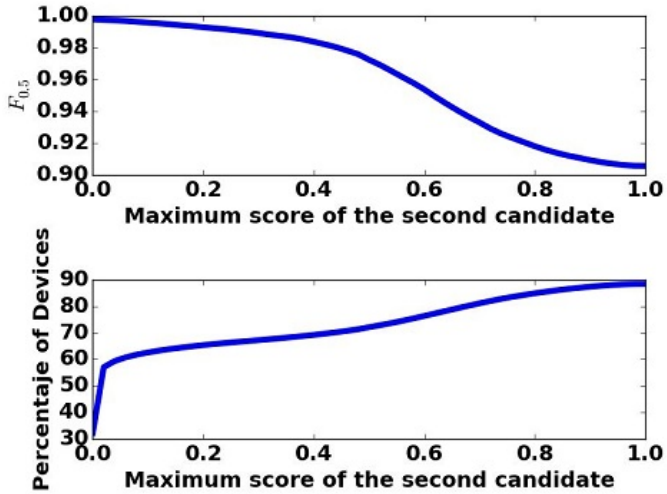


Figura 5.8: $\mathcal{F}_{0,5}$ media y porcentaje de dispositivos empleados en el cálculo cuando el segundo candidato obtiene una puntuación inferior a 0,1

En nuestro caso, el aprendizaje semisupervisado se ha hecho tomando los dispositivos del conjunto de test donde el primer candidato obtenía una puntuación superior a 0.4 y el segundo inferior a 0.05 e incluyéndolos en el conjunto de entrenamiento suponiendo que el primer candidato estaba asociado al dispositivo.

Postprocesado

La selección final de cookies para un dispositivo se realizó utilizando los siguientes criterios:

1. Para cada dispositivo del conjunto de test se obtiene su conjunto de cookies candidatas con el criterio descrito en la sección de preprocesado.

2. Se crean las muestras dispositivo/cookie candidata y se evalúan.

3. Si ninguna muestra dispositivo/cookie obtiene una puntuación superior a cierto umbral, se amplía el conjunto de cookies candidatas incluyendo cualquier cookie que comparte una dirección IP con el dispositivo.

4. Tras evaluar todas las muestras, la cookie que obtiene mayor puntuación y también otras cookies con el mismo handle ID se toman como cookies asociadas al dispositivo.

5. Si la segunda y la tercera cookie con mayor puntuación superan cierto umbral, también se seleccionan como cookies.

5.2.4. Resultados

La puntuación de esta competición se llevaba a cabo evaluando el modelo con un conjunto de test que contenía 61156 dispositivos. Durante la competición la clasificación se realizaba teniendo en cuenta el 30 % de este conjunto de test (clasificación pública) y al terminar la competición el resultado final se obtenía evaluando el otro 70 %.

La Tabla 5.1 muestra la puntuación obtenida en función de las diferentes técnicas utilizadas. Se obtuvo un $\mathcal{F}_{0,5}$ de 0,88, finalizando en tercera posición de un total de 340 equipos, demostrando su buen rendimiento y recibiendo por ello el tercer premio.

Tabla 5.1: Puntuación obtenida en función de las técnicas utilizadas

Sel = Selección Inicial de Candidatos
SL = Aprendizaje Supervisado
B = Bagging
SSL = Aprendizaje Semi-Supervisado
PP = Post Procesado

Procedimientos	Clasificación Pública	Clasificación Privada
Sel	0.498	0.5
IL + SL	0.872	0.875
Sel + SL + B	0.874	0.876
Sel + SSL +B + PP	0.878	0.88

Una descripción más detallada de la solución puede encontrarse en [Díaz-Morales, 2015].

Capítulo 6

Conclusiones

"*A veces creo que hay vida en otros planetas, y a veces creo que no. En cualquiera de los dos casos la conclusión es asombrosa.*"

Carl Sagan (1934 -1996)

En esta Tesis, por un lado se han propuesto nuevos esquemas de paralelización que pretenden reducir los principales problemas que tienen en la actualidad los métodos de núcleo y por otro lado se ha demostrado la utilidad y eficiencia de estos esquemas mediante la implementación de una gran variedad de algoritmos.

En este último capítulo, como es tradicional, en primer lugar se enumeran y analizan las principales aportaciones de esta Tesis Doctoral y en segundo lugar se hace un listado de las líneas de investigación que abre este trabajo.

6.1. Aportaciones

Como se ha visto en el Capítulo 1, los métodos de núcleo tienen serias limitaciones. Por un lado tenemos el elevado coste computacional de estos algoritmos, $\mathcal{O}(n^3)$ con el número de muestras del conjunto de entrenamiento, que es excesivo en la

mayoría de problemas prácticos y limita mucho los problemas que pueden resolver eficientemente. Por otro lado, al ajustar la complejidad del modelo en función del tamaño del conjunto de entrenamiento a menudo obtienen modelos cuyo tamaño es excesivamente grande y por consiguiente son demasiado lentos a la hora de procesar una nueva muestra.

Cuando se ha analizado el estado del arte, se ha visto que las soluciones paralelas existentes tienen algunas limitaciones debido a una paralelización subóptima.

El principal objetivo que se ha perseguido es reducir estas limitaciones y conseguir que los métodos de núcleo puedan abordar un abanico más amplio de problemas.

- En el Capítulo 2 se ha visto en detalle la formulación de las principales familias de métodos de núcleo, SVMs y GPs, tanto en sus versiones normales (no paramétricas) como la formulaciones semi-paramétricas que permiten evitar uno de los principales problemas de este tipo de métodos, el excesivo tamaño de los modelos.

  En este capítulo se han seleccionado, dando la debida justificación, el tipo de algoritmos y estrategias de optimización que se iban a utilizar en los trabajos de esta Tesis:

  - En el caso de la SVM se ha optado por el algoritmo IRWLS como estrategia de optimización ya que tiene un buen compromiso entre número de iteraciones y tiempo de ejecución de cada iteración que lo hace prometedor en un esquema paralelo. En el caso de la SVM completa se han seleccionado estrategias como la división según el tipo de muestra y la utilización de un grupo inactivo para reducir el tiempo de ejecución y tener la complejidad de cada iteración bajo control. En el caso de la SVM semi-paramétrica se ha optado por el algoritmo SGMA para seleccionar los centroides que tendrá el modelo.
  - En el caso de los GPs, en el caso de la versión semi-paramétrica, se ha optado por utilizar SGEV, una estrategia que selecciona centroides que

maximizan la log evidencia en cada iteración.

- En el Capítulo 3 se propone un esquema de paralelización para métodos de núcleo que ayuda a lidiar con las limitaciones de las implementaciones existentes.

  Este nuevo esquema en primer lugar tiene en cuenta los problemas de bajo nivel que pueden tener las implementaciones paralelas de algoritmos destinados a sistemas HPC y toma medidas para evitarlos:

  - Se utilizan bloques contiguos de memoria para almacenar datos.
  - Elección eficiente a la hora de decidir que variables se almacenan en memoria compartida y cuáles no.
  - División eficiente de los datos que va a procesar cada núcleo del procesador.

  También se han tenido en cuenta consideraciones de alto nivel para obtener buenas prestaciones en la paralelización de los algoritmos:

  - Se ha seleccionado un esquema de memoria compartida para minimizar el tiempo de comunicación entre las diferentes subtareas.
  - Se ha realizado un diseño en el que el tiempo de ejecución de las partes no paralelas sea absolutamente despreciable. Para ello se han definido procedimientos paralelos que van desde operaciones en matrices de funciones de núcleo como la resolución de sistemas de ecuaciones utilizando un esquema de inversión por bloques paralelo hasta esquemas MapReduce para modelos que crecen iterativamente.

  Se han desarrollado tres algoritmos haciendo uso de estas técnicas:

  - P-GP: Una implementación paralela de los Procesos Gaussianos.

- PS-GP: Una implementación paralela de Procesos Gaussianos Semi-Paramétricos.
- PS-SVM: Una versión paralela de SVMs Semi-Paramétricas

Se han evaluado experimentalmente las prestaciones de estos algoritmos. Como resumen de los resultados podemos destacar los siguientes puntos:

- PS-SVM obtiene mejores prestaciones en cuanto a paralelización para un rango mayor de valores de sus parámetros.
- Estas técnicas de paralelización, cuando el número de muestras del conjunto de entrenamiento no es muy pequeño reducen enormemente el tiempo de ejecución de los algoritmos acercándose a una aceleración casi lineal con el número de núcleos de procesamiento para el rango de procesadores utilizado (1-12).
- Los modelos semi-paramétricos tienen prestaciones similares que los modelos completos en un tiempo de ejecución bastante menor.
- La combinación de estos esquemas de paralelización en unión con la utilización de modelos dispersos o semi-paramétricos puede incrementar enormemente el rango de problemas que los métodos de núcleo pueden abordar.

Una versión preliminar de las ideas que contiene este capítulo, en la que había una primera versión de PS-SVM que permitía ver cómo de prometedores eran este tipo de técnicas se publicó en:

1. Díaz-Morales, R., Molina-Bulla, H. Y., and Navia-Vázquez, A. (2011). Parallel Semiparametric Support Vector Machines. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 475-481. IEEE.

Los resultados finales, donde se exponen las ideas, experimentos y conclusiones de este capítulo han sido publicados en:

2. Díaz-Morales, R., and Navia-Vázquez, A. (forthcomming, In press). Efficient Parallel Implementation of Kernel Methods. In *Neurocomputing*. Elsevier.

- En el Capítulo 4 se analiza la eficiencia de las soluciones propuestas y se propone un nuevo esquema que reduce las limitaciones que tenían y mejora su coste computacional:

  - Por un lado, la inversa de la matriz por bloques presenta cierta inestabilidad numérica cuando se subdivide en un número muy elevado de tareas.
  - Por otro lado, existen soluciones más eficientes en términos de tiempo de ejecución a la hora de resolver un sistema de ecuaciones lineales que obtener la inversa de una matriz.

  En este nuevo esquema se han seguido las siguientes consideraciones:

  - Para trabajar con las matrices de funciones de núcleo se ha utilizado un esquema "quadtree", que divide iterativamente cada matriz en 4 submatrices.
  - A la hora de resolver sistemas lineales de ecuaciones, debido a que las matrices con las que se trabaja son definidas positivas, se ha utilizado una factorización de Cholesky paralela que permite resolver los sistemas de ecuaciones de forma mucho más estable y en un tiempo de ejecución menor.

  Se han desarrollado dos nuevos algoritmos con estas técnicas:

  - PIRWLS: Una implementación paralela de SVMs.

- PSIRWLS: Una nueva versión paralela de SVMs Semi-Paramétricas

Se han evaluado experimentalmente las prestaciones de estos algoritmos. Como resumen de los resultados podemos destacar los siguientes puntos:

- A la hora de comparar los modelos semi-paramétricos PSIRWLS ha obtenido mejores capacidades de aceleración y un tiempo de ejecución menor que PS-SVM, demostrando las mejoras de este nuevo esquema.
- Pese el tiempo de ejecución de PIRWLS es superior al de LibSVM, sus capacidades de paralelización hacen que sea mucho más rápido al utilizar varios núcleos del procesador.

Para facilitar la reproducibilidad de los experimentos el código fuente está disponible en un repositorio y ambos algoritmos tienen licencia de software libre.

Los resultados finales, donde se exponen las ideas, experimentos y conclusiones de este capítulo han sido enviados y actualmente se encuentran bajo revisión:

3. Díaz-Morales, R., and Navia-Vázquez, A. (Under Review). Improving the efficiency of IRWLS SVCs using Parallel Cholesky Factorization. In *Pattern Recognition Letters*. Elsevier.

- En el Capítulo 5 se muestran otros aportes surgidos indirectamente a raíz de los conocimientos adquiridos durante el desarrollo de la presente tesis. Dichos aportes se corresponden con resultados y premios obtenidos en competiciones de aprendizaje automático gracias a la utilización de código paralelizable y que han dado lugar a publicaciones.

  En primer lugar se muestra un clasificador de eventos en física de partículas llevado a cabo con el simulador del proyecto ATLAS del CERN:

4. Díaz-Morales, R. and Navia-Vázquez, A. (2015). Optimization of AMS using Weighted AUC optimized models. In *JMLR W&CP, 42*, pages 109-127.

El segundo lugar muestra un sistema para identificar a usuarios únicos tras diferentes dispositivos electrónicos con acceso a Internet. Esta solución recibió el tercer premio en la competición del año 2015 del Congreso Internacional en Minería de Datos (ICDM, "International Conference on Data Mining"):

5. Díaz-Morales, R. (2015). Cross-Device Tracking: Matching Devices and Cookies. In *Data Mining Workshops (ICDMW), 2015 IEEE International Conference on*. IEEE

6.2. Líneas de Investigación Abiertas

Tras cualquier trabajo de investigación que culmina con resultados favorables emergen numerosas posibilidades en cuanto a líneas de trabajo a seguir. Las más inmediatas consisten en generalizaciones o extensiones de aplicar los nuevos principios a otras técnicas disponibles. Otro tipo de líneas consisten en nuevas direcciones en los que expandir el trabajo para obtener ventajas adicionales. Aquí se presenta un resumen de las líneas de investigación que se abren con esta Tesis Doctoral.

6.2.1. Extensiones Directas

- Una extensión directa sería la aplicación de las técnicas desarrolladas y conocimientos adquiridos en nuevos tipos de algoritmos y problemas. Como ejemplo de ello se podría realizar una extensión de la formulación para su aplicación en problemas de aprendizaje no supervisado como segmentación.

6.2.2. Extensiones Mayores

- Como vimos en el Capítulo 1, el entorno multiprocesador y multinúcleo es tan solo uno de los posibles escenarios de paralelización, resulta inmediata la idea de extender estos trabajos a nuevos entornos de computación como entornos distribuidos, donde habría que buscar alternativas a la hora de resolver sistemas lineales ya que la comunicación de matrices supondría un cuello de botella severo en las prestaciones del algoritmo.

- Otra opción sería la utilización de GPUs. Para poder obtener un buen rendimiento habría que seleccionar un esquema diferente que en el que todos los procesos realizasen exactamente la misma operación sobre posiciones de memoria diferentes.

- Aunque se puede trabajar con un volumen mayor de datos, aún es pronto para decir que se pueden utilizar en un entorno de los que actualmente se conocen como "Big Data". Pese a ello, gran parte de las técnicas diseñadas serían de aplicación directa, en especial las que tienen un esquema Map-Reduce. Cabe la posibilidad de ampliar estos conceptos para realizar una implementación que sea propiamente 'Big Data" buscando el mejor compromiso de por un lado reducir prestaciones realizando simplificaciones que permitan procesar volúmenes mucho mayores de datos y por otro aumentar las prestaciones al tener mucha más información.

Parte III:

Apéndices

Apéndice A

Multiplicadores de Lagrange y Condiciones de Karush-Kuhn-Tucker

A.1. Restricciones de igualdad

Considérese el problema de optimización:

$$\max_{\mathbf{u}} f(\mathbf{u}) \tag{A.1}$$

s.t.

$$g(\mathbf{u}) = 0 \tag{A.1a}$$

Resulta evidente que en el punto solución $\mathbf{u}^*$ se dará que $\bigtriangledown g(\mathbf{u}^*) \perp g(\mathbf{u})$, y también $\bigtriangledown f(\mathbf{u}^*) \perp g(\mathbf{u})$. Por tanto $\bigtriangledown g(\mathbf{u}^*)$ y $\bigtriangledown f(\mathbf{u}^*)$ son paralelas (ver ejemplo de la Figura A.1) y se cumplirá que para algún λ:

$$\bigtriangledown f(\mathbf{u}^*) + \lambda \bigtriangledown g(\mathbf{u}^*) = 0 \tag{A.2}$$

De modo que al maximizar el llamado lagrangiano primal L_p

$$\max_{\mathbf{u},\lambda} L_p(\mathbf{u}, \lambda) = \max_{\mathbf{u},\lambda} [f(\mathbf{u}) + \lambda g(\mathbf{u})] \tag{A.3}$$

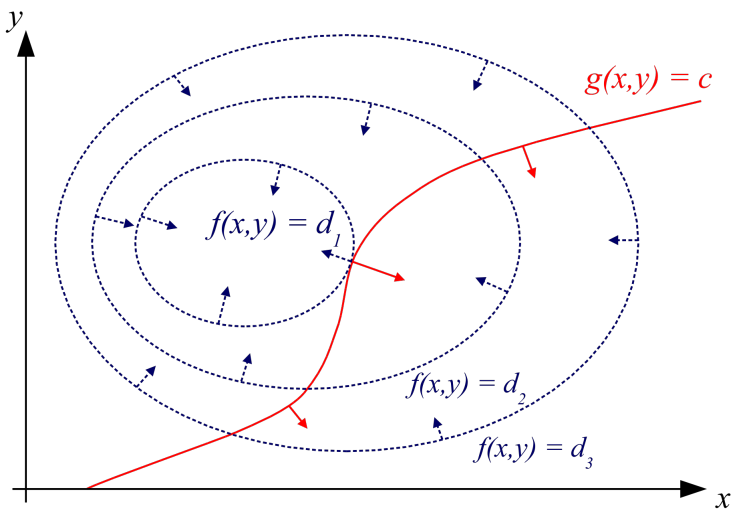


Figura A.1: Minimización con restricciones de igualdad, se puede observar en el punto óptimo el gradiente de ámbas funciones es paralelo y tangencial a la función g

se obtendrá la solución debido a que $\partial L_p/\partial \mathbf{u} = \mathbf{0}$ es un máximo (si el problema está correctamente formulado), y $\partial L_p/\partial \lambda = 0$ implica que $g(\mathbf{u}^*) = 0$.

Si de la ecuación $\partial L_p/\partial \mathbf{u} = \mathbf{0}$ se obtiene $\mathbf{u} = \mathbf{u}(\lambda)$, al sustituir dicha solución en L_p se obtendrá el lagrangiano dual

$$L_p(\lambda) = L_p(\mathbf{u}, \lambda) \tag{A.4}$$

y $\partial L_p/\partial \lambda = 0$ dará la solución λ^*, en la que $\mathbf{u}^* = \mathbf{u}(\lambda^*)$.

A.2. Restricciones de desigualdad

Considérese el problema de optimización:

$$\max_{\mathbf{u}} f(\mathbf{u}) \tag{A.5}$$

s.t.

$$g(\mathbf{u}) \geq 0 \tag{A.5a}$$

(en caso de una desigualdad de signo contrario, $g(\mathbf{u}) \leq 0$ se puede reformular como $-g(\mathbf{u}) \geq 0$).

El lagrangiano primal tiene la forma $f(\mathbf{u}) + \lambda g(\mathbf{u})$. La solución:

- Puede estar en $g(\mathbf{u}) > 0$: con lo que sería el equivalente a que no existiese restricción y puede tomarse $\lambda = 0$;

- Puede estar en $g(\mathbf{u}) = 0$: la restricción $g(\mathbf{u}) = 0$ está activa, y $\lambda > 0$ para que se pueda cumplir

$$\bigtriangledown f(\mathbf{u}) + \lambda \bigtriangledown g(\mathbf{u}) = 0 \tag{A.6}$$

  ya que $\bigtriangledown f(\mathbf{u})$ apunta en sentido opuesto a $\bigtriangledown g(\mathbf{u})$.

El problema se convierte en:

$$\max_{\mathbf{u},\lambda} L_p(\mathbf{u}, \lambda) = \max_{\mathbf{u},\lambda} [f(\mathbf{u}) + \lambda g(\mathbf{u})] \tag{A.7}$$

s.t.

$$g(\mathbf{u}) \geq 0 \tag{A.7a}$$

$$\lambda \geq 0 \tag{A.7b}$$

$$\lambda g(\mathbf{u}) \geq 0 \tag{A.7c}$$

Las restricciones (A.7a,b,c) se denominan de Karush-Kuhn-Tucker, KKT.

El paso a la formulación dual es análogo al anterior.

A.3. El caso particular de la SVM

Las máquinas de vectores soporte son un ejemplo de restricciones de desigualdad. En el caso lineal el problema de optimización venía dado por la función de coste y una restricción por cada una de las muestras del conjunto de entrenamiento:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_n \tag{A.8}$$

s.t.

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \tag{A.8a}$$

El Lagrangiano primal tomando la función a minimizar y un multiplicador λ_i asociado a cada una de las restricciones de desigualdad tendrá entonces la siguiente forma:

$$L_p = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n}\alpha_i y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \tag{A.9}$$

Se pueden tomar las derivadas parciales del Lagrangiano por cada una de las variables $\partial L_p/\partial \mathbf{w} = 0$, $\partial L_p/\partial \lambda_i = 0$ y $\partial L_p/\partial b = 0$ para obtener las condiciones de Karush-Kuhn-Tucker asociadas a este problema:

$$\mathbf{w} = \sum_{i=1}^{n}\lambda_i y_i \mathbf{x}_i \tag{A.10a}$$

$$\sum_{i=1}^{n}\lambda_i y_i = 0 \tag{A.10b}$$

$$\lambda_i \geq 0 \tag{A.10c}$$

Si eliminamos $\mathbf{w}$ y b de la formulación del Lagrangiano utilizando las condiciones de KKT que se han obtenido se llega a la formulación dual, que proporciona una perspectiva diferente de optimización y da una visión de cómo crear soluciones no

lineales transformando el espacio de entrada en un espacio de alta dimensionalidad donde los productos escalares se obtienen mediante una función de núcleo:

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{A.11}$$

Sujeto a:

$$\sum_{i=1}^{n} \lambda_i y_i = 0 \tag{A.12b}$$

$$\lambda_i \geq 0 \tag{A.12c}$$

Apéndice B

Identidades Gaussianas

B.1. Distribución Gaussiana Multivariante

La función de densidad de probabilidad conjunta de una distribución Gaussiana multivariante (n-dimensional) viene dada por:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{n}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{B.1}$$

Donde $\boldsymbol{\mu}$ es el vector de medias de longitud n y $\boldsymbol{\Sigma}$ es la matriz de covarianzas de las distintas componentes de $\mathbf{x}$, que es definida positiva (tamaño $n \times n$).

B.2. Distribución Marginal y Condicional

Si se tienen dos vectores aleatorios $\mathbf{x}_1$ y $\mathbf{x}_2$ que son conjuntamente gaussianos:

$$p(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix}\middle|\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\mathbf{A} & \mathbf{C}\\ \mathbf{C}^\top & \mathbf{B}\end{bmatrix}\right) \tag{B.2}$$

La distribución marginal para la variable $\mathbf{x}_1$ viene dada directamente por sus componentes:

$$p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1, \mathbf{A}) \tag{B.3}$$

Y la distribución condicional de $\mathbf{x}_1$ dado $\mathbf{x}_2$ viene dada por:

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1 + \mathbf{C}\mathbf{B}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^{\top}) \tag{B.4}$$

B.3. Producto de distribuciones Gaussianas

El producto de dos distribuciones gaussianas $\mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ y $\mathcal{N}(\mathbf{x}_2|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ viene dado por:

$$\mathcal{N}(\mathbf{x}_1|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\mathcal{N}(\mathbf{x}_2|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) = Z^{-1}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{B.5}$$

Donde $\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1})^{-1}$, $\mu = \boldsymbol{\Sigma}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2)$ y se aplica la constante de normalización Z^{-1} para obtener una distribución de area unidad:

$$Z^{-1} = (2\pi)^{-\frac{n}{2}}|\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2|^{-\frac{1}{2}} exp\left(-\frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\top}(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1})(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\right) \tag{B.6}$$

Apéndice C

IRWLS para resolver SVMs

C.1. SVM Completa

Para abordar el problema de mínimos cuadrados a resolver en cada iteraición mediante IRWLS y tener la complejidad bajo control se ha optado por tener un conjunto inactivo en cada iteración.

El algoritmo 1 muestra el bucle externo que se encarga de la creación del conjunto activo e inactivo. El algoritmo 2 muestra el funcionamiento de IRWLS para la actualización de pesos del conjunto activo.

Algoritmo 1: Bucle externo, selección del conjunto inactivo en cada iteración

Entrada: $\mathbf{X_{tr}} = \{(\mathbf{x}_i, y_i)|\mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$ Conjunto de entrenamiento
Entrada: S Tamaño del conjunto activo de cada iteración
Salida : $\mathbf{w}$ Pesos del modelo

Inicialización
- **for** $i = 1, \ldots, n$ **do**
  - $e_i = 1$, $w_i = 1$
- **end**
- $b = 0$
- S_W Conjunto activo, contiene S elementos de $\mathcal{D}$ aleatorios.
- S_{IN} Conjunto inactivo, contiene el resto de elementos.

Actualización Variables Conjunto Inactivo
- $G_{IN} = \mathbf{H}_{S_W, S_{IN}} \mathbf{w}_{S_{IN}}$
- $G_{bIN} = \mathbf{y}_{S_{IN}}^T \mathbf{w}_{S_{IN}}$

Actualización de pesos mediante IRWLS
- $(\boldsymbol{w}'_w, b') = IRWLS(S_W, G_{IN}, G_{bIN})$

Actualización de Variables
- **for** $i = 1, \ldots, n$ **do**
  - $(\mathbf{k}_i)e = K(\mathbf{x}_i, \mathbf{x}_{(S_w)_e})$
  - $e_i = e_i - \mathbf{k}_i^T(\boldsymbol{w'}_w - \boldsymbol{w}_w) - (b' - b)$
- **end**
- $b = b'$
- Para todo elmento en $S_w \rightarrow w_i = w'_i$

Actualización de Conjuntos
- Una muestra de entrenamiento $\mathbf{x}_i$ es elegible para el conjunto de trabajo si:
  - No es un vector soporte.
  - No cumple las condiciones (2.12) o (2.13)
- Se incluye en S_W un candidato por clase que incumpla cada condición (2.12),(2.13) y (2.14), se completa con muestras elegibles aleatorias.

Criterio de Parada
- $criterio = \|\mathbf{w}' - \mathbf{w}\|_2 + \|b' - b\|_2$
- **if** $criterio \leq \eta$ **then**
  - go to Termina
- **else**
  - $w_i = w'_i, \forall i \in \mathcal{S}_w$, $b = b'$
  - go to Actualización Variables Conjunto Inactivo
- **end**

Algoritmo 2: Obtención de pesos del conjunto activo utilizando el algoritmo IRWLS

Entrada: Conjunto de trabajo $\mathcal{S}_w = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$
Entrada: Error y Pesos del conjunto activo: $\mathbf{e}_w$,$\mathbf{w}_w$ y bias b
Entrada: Variables del conjunto inactivo: $\mathbf{G}_{IN}$,G_{bIN}
Salida : $\mathbf{w}$ Pesos del conjunto activo y bias b

Asignación al Grupo

$a_i = \begin{cases} 0, y_i e_i \leq 0 \\ \frac{C}{e_i y_i}, y_i e_i \geq 0 \end{cases}$

Cada muestra con $a_i = 0$ a $S2$.

Cada muestra con $a_i \neq 0$ y $w_i = y_i C$ a $S3$.

Cada muestra con $a_i \neq 0$ y $w_i \neq y_i C$ a $S1$.

Actualizar Pesos

$(\mathbf{H}_{S1,S1})_{i,j} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \forall i, j \in S1$

$(\mathbf{H}_{S1,S3})_{i,j} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \forall i \in S1, \forall j \in S3$

$(\mathbf{D}_{aS1})_{i,j} = a_i \delta[i - j], \forall i, j \in S1$

$(\mathbf{y}_{S1})_i = y_i, \forall i \in S1$, $(\mathbf{y}_{S3})_i = y_i, \forall i \in S3$

$(\mathbf{G}_{INS1})_i = (\mathbf{G}_{IN})_i, \forall i \in S1$

$(\mathbf{w}_{S3})_i = w_i, \forall i \in S3$

$\mathbf{G}_{13} = \mathbf{H}_{S1,S3}\mathbf{w}_{S3}$, $G_b = \sum_{i \in S3} y_i C$

$\mathbf{K_1} = \begin{bmatrix} \mathbf{H}_{S1,S1} + \mathbf{D}_{aS1}^{-1} & \mathbf{y}_{S1} \\ \mathbf{y}_{S1}^T & 0 \end{bmatrix}$, $\mathbf{k_2} = \begin{bmatrix} \mathbf{1} - \mathbf{G}_{13} - \mathbf{G}_{INS1} \\ -G_b - G_{bIN} \end{bmatrix}$

$\begin{bmatrix} \boldsymbol{\beta}_{S1} \\ b \end{bmatrix} = \mathbf{K_1}^{-1}\mathbf{k_2}$

$w_i = y_i \beta_i, \forall i \in S1$, $w_i = 0, \forall i \in S2$, $w_i = y_i C, \forall i \in S3$

Actualizar Variables

for $i \in \mathcal{S}_w$ **do**

$(\mathbf{k}_i)e = K(\mathbf{x}_i, \mathbf{x}_e) \forall e \in \mathcal{S}_w$

$e_i = e_i - \mathbf{k}_i(\mathbf{w}'_w - \mathbf{w}_w) - (b' - b)$

end

Criterio de Parada

$criterio = \|\mathbf{w}' - \mathbf{w}\|_2 + \|b' - b\|_2$

if $criterio \leq \eta$ **then**

go to Termina

else

$w_i = w'_i, \forall i \in \mathcal{S}_w$, $b = b'$

go to Asignación al Grupo

end

C.2. SVM semipramétrica

Las SVMs son métodos no paramétricos, el tamaño de la función de clasificación (que es el número de vectores soporte), no está predeterminado y se modela en función de las muestras de entrenamiento. Este tamaño, al no estar bajo control puede ser excesivamente grande y el clasificador no apto para muchas aplicaciones que requieran trabajar en "tiempo real".

El entrenamiento de este tipo de procedimientos consta de dos etapas. La primera selecciona el conjunto de m muestras que se van a utilizar como elementos base, que en nuestro caso se ha realizado mediante el algoritmo SGMA, y la segunda etapa que obtiene los pesos $\boldsymbol{\beta}$ óptimos, que en nuestro caso se ha realizado utilizando el algoritmo IRWLS.

El algoritmo 3 muestra la estructura del algoritmo SGMA, en cada iteración se evalúa el descenso de error de una lista de candidatos seleccionada aleatoriamente y se incorpora a la base el candidato que ha obtenido un mayor descenso de error.

El algoritmo 4 muestra la estructura del algoritmo IRWLS, en cada iteración se resuelve un sistema de mínimos cuadrados ponderados hasta que la solución converge.

Algoritmo 3: Selección de centroides del modelo semi-paramétrico utilizando SGMA

Entrada: $\mathbf{X_{tr}} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$ Conjunto de entrenamiento
Entrada: L Número de candidatos en cada iteración
Entrada: M Número de elementos base de salida
Salida : $\mathbf{X_C}$ Conjunto de elementos base del modelo semi-paramétrico

---

Inicialización
  $\mathbf{X_C} = \emptyset$ Conjunto de elementos base $\{\mathbf{c}_1, \ldots\}$, inicialmente vacío
  $\mathbf{K_C} = [] \rightarrow (\mathbf{K_C})_{ij} = k(\mathbf{c}_i, \mathbf{c}_j)$ (inicialmente tamaño 0×0)
  $\mathbf{K_{nm}} = [] \rightarrow (\mathbf{K}_{nm})_{ij} = k(\mathbf{x}_i, \mathbf{c}_j)$ (inicialmente tamaño tamaño $n \times 0$)
for $o = 1, \ldots, M$ **do**
  for $e = 1, \ldots, L$ **do**
    Selección aleatoria
      $\mathbf{x_e}$ es seleccionado aleatoriamente entre los elmentos de $\mathbf{X_{tr}}$
    Cálculo de Descenso de Error
      $(\mathbf{k}_{nc}^e)_i = k(\mathbf{x}_i, \mathbf{x}_e)\ \forall i = 1, ..., n$
      $(\mathbf{k}_{mc}^e)_i = k(\mathbf{c}_i, \mathbf{x}_e)\ \forall i = 1, ..., (o-1)$
      $\mathbf{z} = \mathbf{K}_C^{-1} \mathbf{k}_{mc}^e$
      $\eta = 1 - \mathbf{z}^T \mathbf{k}_{nc}^e$
      $ED_e = \eta^{-1} \|\mathbf{K}_{nm}\mathbf{z} - \mathbf{k}_{nc}^e\|^2$
  end
  Selección de elemento y actualización
    $k = \underset{i}{\text{argmáx}}(ED_i)$
    $\mathbf{X_C} = \mathbf{X_C} \cup \{\mathbf{x_k}\}$
    $\mathbf{K_{nm}} = \begin{bmatrix} \mathbf{K_{nm}} \\ \mathbf{k}_{nc}^k \end{bmatrix}$
    Actualización de rango de $\mathbf{K_C}$ y $\mathbf{K_C}^{-1}$ con el nuevo elemento.
end

Algoritmo 4: Obtención de pesos utilizando el algoritmo IRWLS

Entrada: $\mathbf{X_{tr}} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$ Conjunto de entrenamiento
Entrada: $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$ Centroides seleccionados mediante SGMA
Salida : $\boldsymbol{\beta}$ Pesos de los centroides

Inicialización

$a_i = 1, \forall i = 1, ..., n$
$(\mathbf{K}_{nm})_{ij} = k(\mathbf{x}_i, \mathbf{c}_j); \forall i = 1, ..., n; j = 1, ..., m$
$(\mathbf{K}_{\mathbf{c}})_{ij} = k(\mathbf{c}_i, \mathbf{c}_j); \forall i = 1, ..., m; j = 1, ..., m$
$(\mathbf{D}_a)_i = a_i; \forall i = 1, ..., n$
$\mathbf{1} = [1, ..., 1]^T$
$\mathbf{y} = [y_1, ..., y_n]^T$

Actualización de Pesos

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{K}_C + \mathbf{K}_{nm}^T \mathbf{D}_a \mathbf{K}_{nm} & \mathbf{K}_{nm}^T \mathbf{D}_a \mathbf{1} \\ \mathbf{1}^T \mathbf{D}_a \mathbf{K}_{nm} & \mathbf{1}^T \mathbf{D}_a \mathbf{1} \end{bmatrix}$$

$$\mathbf{k}_2 = \begin{bmatrix} \mathbf{K}_{nm}^T \mathbf{D}_a \mathbf{y} \\ \mathbf{1} \mathbf{D}_a \mathbf{y} \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{\beta} \\ b \end{bmatrix} = \mathbf{K}_1^{-1} \mathbf{k}_2$$

Actualización del Error

$e_i = y_i - \sum_{j=1}^m \beta_i k(\mathbf{x}_i, \mathbf{c}_j)$

Actualización de la Ponderación

$$\alpha_i = \begin{cases} 0; y_i e_i \leq 0 \\ \frac{C}{e_i y_i}; y_i e_i \geq 0 \end{cases}$$

Criterio de Parada

$criterio = \|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2 + \|b' - b\|_2$
if $criterio \leq 10^{-6}$ **then**
  go to Termina
else
  $\boldsymbol{\beta} = \boldsymbol{\beta}'$:
  $b = b'$
  go to Actualización de Pesos
end

Bibliografía

[Aad et al., 2012] Aad, G., Abajyan, T., Abbott, B., Abdallah, J., Abdel Khalek, S., Abdelalim, A., Abdinov, O., Aben, R., Abi, B., Abolins, M., et al. (2012). Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29.

[Adam-Bourdarios et al., 2015] Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., and Rousseau, D. (2015). The higgs boson machine learning challenge. In *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42, page 37.

[Almasi George and Allan, 1989] Almasi George, S. and Allan, G. (1989). *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc.

[Amdahl, 1967] Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM.

[Andrews, 1999] Andrews, G. R. (1999). *Foundations of parallel and distributed programming*. Addison-Wesley Longman Publishing Co., Inc.

[Asuncion and Newman, 2007] Asuncion, A. and Newman, D. (2007). Uci machine learning repository.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.

[Bergstra et al., 2010] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7.

[Bishop, 1995] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.

[Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[Cao et al., 2006] Cao, L., Keerthi, S. S., Ong, C. J., Uvaraj, P., Fu, X. J., and Lee, H. (2006). Developing parallel sequential minimal optimization for fast training support vector machine. *Neurocomputing*, 70(1):93–104.

[Chang and Lin, 2011] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27.

[Chatrchyan et al., 2012] Chatrchyan, S., Khachatryan, V., Sirunyan, A. M., Tumasyan, A., Adam, W., Aguilo, E., Bergauer, T., Dragicevic, M., Erö, J., Fabjan, C.,

et al. (2012). Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716(1):30–61.

[Chen, 2014] Chen, T. (2014). A general purpose parallel gradient boosting (tree) library. `https://github.com/tqchen/xgboost`. Last visited: 2014-09-26.

[Chih-Chung and Chih-Jen, 2015] Chih-Chung, C. and Chih-Jen, L. (2015). Libsvm faq. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html`. Last visited: 2015-09-30.

[Chu et al., 2007] Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281.

[Collobert et al., 2002] Collobert, R., Bengio, S., and Bengio, Y. (2002). A parallel mixture of svms for very large scale problems. *Neural computation*, 14(5):1105–1114.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Cotter et al., 2011] Cotter, A., Srebro, N., and Keshet, J. (2011). A gpu-tailored approach for training kernelized svms. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 805–813. ACM.

[Csató and Opper, 2002] Csató, L. and Opper, M. (2002). Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668.

[Dagum and Enon, 1998] Dagum, L. and Enon, R. (1998). Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

[Díaz-Morales, 2015] Díaz-Morales, R. (2015). Cross-device tracking: Matching devices and cookies. In *Data Mining Workshops (ICDMW), 2015 IEEE International Conference on*, pages 170–177. IEEE.

[Díaz-Morales et al., 2011] Díaz-Morales, R., Molina-Bulla, H. Y., and Navia-Vázquez, A. (2011). Parallel semiparametric support vector machines. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 475–481. IEEE.

[Dıaz-Morales and Navia-Vázquez, 2015] Dıaz-Morales, R. and Navia-Vázquez, A. (2015). Optimization of ams using weighted auc optimized models. *JMLR W&CP*, 42:109–127.

[Díaz-Morales and Navia-Vázquez, 2016a] Díaz-Morales, R. and Navia-Vázquez, A. (in press, forthcominga). Efficient parallel implementation of kernel methods. *Neurocomputing*.

[Díaz-Morales and Navia-Vázquez, 2016b] Díaz-Morales, R. and Navia-Vázquez, A. (under review b). Improving the efficiency of irwls svcs using parallel cholesky factorization. *Pattern Recognition Letters*.

[Dong et al., 2003] Dong, J.-x., Krzyżak, A., and Suen, C. (2003). A fast parallel optimization for training support vector machine. In *Machine Learning and Data Mining in Pattern Recognition*, pages 96–105. Springer.

[Duda et al., 2012] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[Ferris and Munson, 2002] Ferris, M. C. and Munson, T. S. (2002). Interior-point methods for massive support vector machines. *SIAM Journal on Optimization*, 13(3):783–804.

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

[Haykin and Network, 2004] Haykin, S. and Network, N. (2004). A comprehensive foundation. *Neural Networks*, 2(2004).

[Herrero-Lopez et al., 2010] Herrero-Lopez, S., Williams, J. R., and Sanchez, A. (2010). Parallel multiclass classification using svms on gpus. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 2–11. ACM.

[HiggsML, 2014] HiggsML (2014). Higgs boson machine learning challenge. `https://www.kaggle.com/c/higgs-boson`. Last visited: 2015-12-31.

[Hill and Marty, 2008] Hill, M. D. and Marty, M. R. (2008). Amdahl's law in the multicore era. *Computer*, 7(41):33–38.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[ICDM, 2015] ICDM, K. (2015). Icdm 2015: Drawbridge cross-device connections. `https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections`. Last visited 2015-12-31.

[Joachims, 1999] Joachims, T. (1999). Making large scale svm learning practical. Technical report, Universität Dortmund.

[Juve et al., 2009] Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B. P., and Maechling, P. (2009). Scientific workflow applications on amazon

ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59–66. IEEE.

[Karmarkar, 1984] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM.

[Kwok and Tsang, 2004] Kwok, J. T.-Y. and Tsang, I. W.-H. (2004). The pre-image problem in kernel methods. *Neural Networks, IEEE Transactions on*, 15(6):1517–1525.

[Lawrence et al., 2003] Lawrence, N., Seeger, M., and Herbrich, R. (2003). Fast sparse gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, pages 609–616.

[Ling et al., 2003] Ling, C. X., Huang, J., and Zhang, H. (2003). Auc: a statistically consistent and more discriminating measure than accuracy. In *IJCAI*, volume 3, pages 519–524.

[MacKay, 2003] MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

[Mashey, 1997] Mashey, J. R. (1997). Big data and the next wave of infras-tress. In *Computer Science Division Seminar, University of California, Berkeley*.

[Meyer and Vlachos, 2009] Meyer, M. and Vlachos, P. (2009). Statlib data, software and news from the statistics community, 2009.

[Navia-Vázquez, 2007] Navia-Vázquez, A. (2007). Compact multi-class support vector machine. *Neurocomputing*, 71(1):400–405.

[Navia-Vázquez and Díaz-Morales, 2010] Navia-Vázquez, A. and Díaz-Morales, R. (2010). Fast error estimation for efficient support vector machine growing. *Neurocomputing*, 73(4):1018–1023.

[Navia-Vázquez et al., 2001] Navia-Vázquez, A., Pérez-Cruz, F., Artés-Rodríguez, A., and Figueiras-Vidal, A. R. (2001). Weighted least squares training of support vector classifiers leading to compact and adaptive schemes. *Neural Networks, IEEE Transactions on*, 12(5):1047–1059.

[Neumann et al., 2009] Neumann, M., Kersting, K., Xu, Z., and Schulz, D. (2009). Stacked gaussian process learning. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 387–396. IEEE.

[Osuna and Girosi, 1998] Osuna, E. and Girosi, F. (1998). Reducing the run-time complexity of support vector machines. In *International Conference on Pattern Recognition*.

[Owens et al., 2008] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. (2008). Gpu computing. *Proceedings of the IEEE*, 96(5):879–899.

[Park et al., 2012] Park, C., Huang, J. Z., and Ding, Y. (2012). Gplp: a local and parallel computation toolbox for gaussian process regression. *The Journal of Machine Learning Research*, 13(1):775–779.

[Parrado-Hernández et al., 2003] Parrado-Hernández, E., Mora-Jiménez, I., Arenas-Garcıa, J., Figueiras-Vidal, A. R., and Navia-Vázquez, A. (2003). Growing support vector classifiers with controlled complexity. *Pattern Recognition*, 36(7):1479–1488.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Peng et al., 2013] Peng, J.-X., Ferguson, S., Rafferty, K., and Stewart, V. (2013). A sequential algorithm for sparse support vector classifiers. *Pattern Recognition*, 46(4):1195–1208.

[Pérez-Cruz et al., 2001] Pérez-Cruz, F., Alarcón-Diana, P. L., Navia-Vázquez, A., and Artés-Rodríguez, A. (2001). Fast training of support vector classifiers. In *Advances in Neural Information Processing Systems*, pages 734–740.

[Pérez-Cruz et al., 2005] Pérez-Cruz, F., Bousoño-Calzón, C., and Artés-Rodríguez, A. (2005). Convergence of the IRWLS procedure to the support vector machine solution. *Neural Computation*, 17(1):7–18.

[Platt et al., 1999] Platt, J. et al. (1999). Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods—support vector learning*, 3.

[Poor, 2013] Poor, H. V. (2013). *An introduction to signal detection and estimation*. Springer Science & Business Media.

[Press et al., 1996] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1996). *Numerical recipes in C*, volume 2. Cambridge university press Cambridge.

[Quinonero-Candela and Rasmussen, 2005] Quinonero-Candela, J. and Rasmussen, C. E. (2005). Analysis of some methods for reduced rank gaussian process regression. In *Switching and Learning in Feedback Systems*, pages 98–127. Springer.

[Ranger et al., 2007] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. (2007). Evaluating mapreduce for multi-core and multiprocessor systems. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 13–24. Ieee.

[Rasmussen, 2006] Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. MIT press.

[Samet, 1984] Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260.

[Sanders and Kandrot, 2010] Sanders, J. and Kandrot, E. (2010). *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional.

[Schölkopf et al., 1998] Schölkopf, B., Knirsch, P., Smola, A., and Burges, C. (1998). Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces. In *Mustererkennung 1998*, pages 125–132. Springer.

[Schölkopf et al., 1997] Schölkopf, S. P., Simard, P., Vapnik, V., and Smola, A. (1997). Improving the accuracy and speed of support vector machines. *Advances in neural information processing systems*, 9:375–381.

[Shvachko et al., 2010] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE.

[Smola and Bartlett, 2001] Smola, A. J. and Bartlett, P. (2001). Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems 13*. Citeseer.

[Smola and Schölkopf, 1998] Smola, A. J. and Schölkopf, B. (1998). *Learning with kernels*. Citeseer.

[Smola and Schölkopf, 2000] Smola, A. J. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In *17th International Conference on Machine Learning*.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Vapnik, 1963] Vapnik, V. (1963). Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780.

[Vapnik, 2013] Vapnik, V. (2013). *The nature of statistical learning theory*. Springer Science & Business Media.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408.

[Wang et al., 2014] Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., and Wang, Y. (2014). Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*, pages 167–188. Springer.

[Wiener, 1949] Wiener, N. (1949). *Extrapolation, interpolation, and smoothing of stationary time series*, volume 2. MIT press Cambridge, MA.

[You et al., 2014] You, Z.-H., Yu, J.-Z., Zhu, L., Li, S., and Wen, Z.-K. (2014). A mapreduce based parallel svm for large-scale predicting protein–protein interactions. *Neurocomputing*, 145:37–43.

[Zaharia et al., 2010] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10.

[Zhang, 2012] Zhang, S. (2012). An improved parallel svm algorithm for chinese text classification. In *Proceedings of the 2012 Second International Conference on Electric Technology and Civil Engineering*, pages 270–272. IEEE Computer Society.

[Zhu et al., 2008] Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., Cui, H., and Chang, E. Y. (2008). Parallelizing support vector machines on distributed computers. In *Advances in Neural Information Processing Systems*, pages 257–264.

Glosario

ADN "DeoxyriboNucleic Acid"
(Ácido DesoxirriboNucleico)

AMS "Approximate Median Significance"
(Significancia Mediana Aproximada)

ATLAS "A Toroidal LHC ApparatuS"
(Aparato Toroidal del LHC)

AUC "Area Under the Curve"
(Área Bajo la Curva)

CERN "Conseil Européen pour la Recherche Nucléaire"
(Organización Europea para la Investigación Nuclear)

DBN "Deep Belief Net"
(Red de Evidencia Profunda)

DL "Deep Learning"
(Aprendizaje Profundo)

EM "Expectation Maximization"
(Maximización de la Esperanza)

FPR "False Positive Rate"
(Tasa de Falsos Positivos)

GBT "Gradient Boosted Trees"
(Àrboles Potenciados por Gradiente)

GMM "Gaussian Mixture Model"
(Modelo de Mezcla de Gaussianas)

GP "Gaussian Process"
(Proceso Gaussiano)

GPU "Graphical Processing Unit"
(Unidad de Procesado Gráfico)

HL "Hinge Loss"
(Pérdida de Bisagra)

HPC "High Performance Computing"
(Cómputo de Alto Rendimiento)

ICDM "International Conference on Data Mining"
(Congreso Internacional en Minería de Datos)

IP "Internet Protocol"
(Protocolo de Internet)

IPM "Interior Point Method"
(Método del Punto Interior)

IRWLS "Iterative Re-Weighted Least Squares"
(Mínimos Cuadrados Ponderados Iterativos)

KKT "Karush-Kuhn-Tucker"
(Karush-Kuhn-Tucker)

LAPACK "Linear Algebra PACKage"
(Paquete de Algebra Lineal)

LHC "Large Hadron Collider"
(Gran Colisionador de Hadrones)

LL	"Log Loss"
	(Pérdida Logarítmica)
MAP	"Maximum a Posteriori"
	(Máximo a Posteriori)
MSE	"Mean Squared Error"
	(Error Cuadrático Medio)
NMSE	"Normalized Mean Squared Error"
	(Error Cuadrático Medio Normalizado)
P-GP	"Parallel GP"
	(GP Paralelo)
PICF	"Parallel Incomplete Cholesky Factorization"
	(Factorización Incompleta de Cholesky Paralela)
PIRWLS	"Parallel IRWLS"
	(IRWLS Paralelo)
PS-GP	"Parallel Semiparametric GP"
	(GP Semiparamétrico Paralelo)
PS-SVM	"Parallel Semiparametric SVM"
	(SVM Semiparamétrica Paralela)
PSIRWLS	"Parallel Semiparametric IRWLS"
	(IRWLS Semiparamétrico Paralelo)
PSVM	"Parallel SVM"
	(SVM Paralela)
RAM	"Random Access Memory"
	(Memoria de Acceso Aleatorio)
ROC	"Receiver Operating Characteristic"
	(Característica Operativa del Receptor)
SDA	"Stacked Denoising Autoencoder"
	(Autocodificadores Apilados)

SGEV "Sparse Greedy Evidence"
(Evidencia Codiciosa Dispersa)

SGMA "Sparse Greedy Matrix Approximation"
(Aproximación Matricial Codiciosa Dispersa)

SMO "Sequential Minimal Optimization"
(Optimización Mínima Secuencial)

SVM "Support Vector Machines"
(Máquina de Vectores Soporte)

TPR "True Positive Rate"
(Tasa de Positivos Verdaderos)

UCI "University of California Irvine"
(Universidad de California Irvine)

WAUC "Weighted AUC"
(AUC ponderada)

WFPR "Weighted FPR"
(FPR ponderada)

WTPR "Weighted TPR"
(TPR ponderada)

Índice alfabético