COMPUTACIÓN PARALELA APLICADA A LA DETECCIÓN DE ANORMALIDADES EN LA FERMENTACIÓN VÍNICA MEDIANTE MÁQUINAS DE SOPORTE VECTORIAL

Andrés José Ferreira Barragán

Universidad Industrial de Santander
Facultad de Ingenierías Fisico-mecánicas
Escuela de Ingeniería de Sistemas e Informática
Bucaramanga
2016

COMPUTACIÓN PARALELA APLICADA A LA DETECCIÓN DE ANORMALIDADES EN LA FERMENTACIÓN VÍNICA MEDIANTE MÁQUINAS DE SOPORTE VECTORIAL

Andrés José Ferreira Barragán

Trabajo de grado presentado como requisito parcial para optar al título de:
Ingeniero de Sistemas e Informática

Director:
Ph.D. Carlos Jaime Barrios Hernández
Co-director:
Ph.D. Gonzalo Javier Hernández Oliva

Universidad Industrial de Santander
Facultad de Ingenierías Fisico-mecánicas
Escuela de Ingeniería de Sistemas e Informática
Bucaramanga
2016

DEDICATORIA

A Dios, a mis padres y a mi hermana quienes durante todo este tiempo han vivido y disfrutado esta aventura. Sin su apoyo no hubiera podido llevar a cabo esta misión.

Los amo con todo mi corazón.

AGRADECIMIENTOS

Este trabajo no hubiera sido posible sin el interés y animo que me han dado una serie de amigos. En especial al PhD. Carlos Jaime Barrios Hernández y al PhD. Raul Ramos Pollan quienes siempre estuvieron dispuestos a orientarme en este trabajo; y a las personas que hacen parte del grupo de "Supercomputación y Calculo Científico de la Universidad Industrial de Santander", cuyo apoyo nunca falto.

RESUMEN

TÍTULO: COMPUTACIÓN PARALELA APLICADA A LA DETECCIÓN DE ANORMALIDADES EN LA FERMENTACIÓN VÍNICA MEDIANTE MÁQUINAS DE SOPORTE VECTORIAL[1]

AUTOR: Andrés José Ferreira Barragán[2]

PALABRAS CLAVE: APRENDIZAJE SUPERVISADO, CLASIFICACIÓN, MÁQUINAS DE SOPORTE VECTORIAL, COMPUTACIÓN PARALELA, UNIDAD DE PROCESAMIENTO GRÁFICO (GPU).

DESCRIPCIÓN: Las diferentes ciencias, la industria, los investigadores, etc., están produciendo un creciente volumen de datos. La producción y análisis eficiente de los datos son clave para futuros descubrimientos. En diferentes ocasiones estos datos deben ser clasificados, y a partir de esta clasificación, se toman diferentes decisiones dentro de determinada área. Las máquinas de soporte vectorial surgen como modelo de aprendizaje supervisado que hace uso de las diferentes características del problema, para la clasificación de futuros datos de la misma índole.

A pesar de que las arquitecturas actuales en las computadoras son eficientes, el tiempo de cómputo de la implementación de una SVM puede ser alto, según la cantidad de ejemplos de entrenamiento que se tengan para construir el modelo y su cantidad de características del problema. Para soportar este problema, varias estrategias e implementaciones han sido propuestas con cierto impacto en el rendimiento de cómputo. Aun así, escasos trabajos se han realizado en el ambiente de arquitecturas híbridas que soporten aceleradores como GPUs.

En éste trabajo, se propone una solución para reducir el tiempo de computo, mediante la implementación de la LIBSVM (librería que implementa algoritmos para máquinas de soporte vectorial) haciendo uso de estándares de programación paralela como OpenACC, OpenMP, MPI. Dicha solución se enfoca en el uso de recursos computacionales de la máquina y en los procesos concurrentes de la librería. El producto de este trabajo no sólo asegura un procesamiento masivo de datos en un tiempo considerablemente bajo, sino también la fidelidad en los resultados obtenidos por este.

[1] Trabajo de investigación de pregrado

[2] Facultad de Ingenierías Físico-mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Carlos Jaime Barrios Hernández, Doctor en Informática. Codirector: Gonzalo Javier Hernández Oliva, Doctor en Ciencias de la Ingeniería

ABSTRACT

TITLE: PARALLEL COMPUTING APPLIED TO THE DETECTION OF ABNORMALITIES IN WINE FERMENTATION THROUGH SUPPORT VECTOR MACHINES[3]

AUTHOR: Andrés José Ferreira Barragán[4]

KEYWORDS: SUPERVISED LEARNING, CLASSIFICATION, SUPPORT VECTOR MACHINE, PARALLEL COMPUTING, GRAPHICS PROCESSING UNIT (GPU).

DESCRIPTION: Different sciences, industry, researchers, etc., are producing a growing volume of data. Efficient production and analysis of data are key to future discoveries. At different times these data must be classified, and from this classification, different decisions are made within a given area. The vector support machines emerge as a supervised learning model that makes use of the different characteristics of the problem, for the classification of future data of the same nature.

Although the current architectures in the computers are efficient, the computation time of the implementation of an SVM can be high, according to the amount of training examples that are to construct the model and the amount of characteristics of the problem. To support this problem, several strategies and implementations have been proposed with some impact on computational performance. Even so, the scarce work has been done in the environment of architectures that have supported accelerators such as GPUs.

In this work, we propose a solution to reduce computation time, by implementing LIBSVM (library that implements algorithms for support vector machines) making use of parallel programming standards such as OpenACC, OpenMP, MPI. This solution focuses on the use of computational resources of the machine and the concurrent processes of the library. The product of this work not only ensures a massive processing of data in a considerably low time, also the fidelity in the results obtained by this one.

[3]Undergraduate degree research work

[4]Departmen of Physical-Mechanical Engineering. School of Systems Engineering and Computer Science. Advisor: Carlos Jaime Barrios Hernández, Ph.D. in Computer Science. Co-advisor: Gonzalo Javier Hernández Oliva, Ph.D. in Engineering Science

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

INTRODUCCIÓN

En la industria, detectar aquellos vinos que no están fermentándose de la manera correcta es posible hacerlo mediante diferentes métodos, tales como el análisis a través de olores, sabores, liberación de CO_2, nivel de alcohol, el uso de redes neuronales las cuales emplean técnicas de reconocimiento de patrones basadas en inteligencia artificial, además de otras variables químicas [9] [20] [22].

Estos métodos usados en la industria, en su mayoría son realizados por un enólogo, quien es un experto en el área de la elaboración de vinos, no obstante, se pueden presentar fallas que se traducen como una alteración de las propiedades organolépticas deseadas en el vino, por parte de los vinicultores. Estos errores pueden ser clasificados desde el punto de vista químico (fallos químicos o fallos microbianos) y desde el punto de vista de detectabilidad de los sentidos (fallos visibles o fallos detectables por sabor u olor). Es por esto que no resultan 100 % fiables las conclusiones obtenidas. Es importante resaltar que los fallos en el vino se traducen en la mayoría de las ocasiones en pérdidas económicas, es por esta razón por la que la industria investiga y categoriza estos fallos.

Como alternativa al empleo de redes neuronales artificiales, el reconocimiento de cuando una fermentación vínica tiene fallos o no, se puede realizar utilizando otro tipo de técnicas de inteligencia artificial. En concreto, las máquinas de soporte vectorial (también conocidas como SVM), han demostrado ser superiores en muchos campos, especialmente cuando se trata de problemas de clasificación.

En [8] se utiliza una SVM, para predecir en base a tres variables (densidad, alcohol, azúcar), el comportamiento de las fermentaciones. La base de datos utilizada para el análisis del comportamiento de las fermentaciones era de 20.000 datos con 19 variables de estudio, la cual posteriormente creció a más de 40.000 datos con 44 variables de estudio. El aumento de los datos, llevó a los investigadores a preguntarse cómo afectan otras variables la fermentación de los vinos.

Las SVM toman un tiempo considerablemente alto en entregar un modelo, ya sea para un problema de clasificación o regresión, esto es según la cantidad de datos, número de pruebas, procedimientos tales como operaciones matriciales y sobre todo debido a su carácter secuencial.

La primera etapa para la clasificación mediante una SVM, es tener una buena representación de los datos, que servirán como entrenamiento (Input) para el modelo y su respectiva salida (Output). Aquí se debe tener en cuenta la cantidad de características en el problema, ya que

la dimensionalidad del problema es equivalente a este. También es importante estudiar como serian afectados los resultados si los datos de entrenamiento son normalizados ó escalados.

En segunda instancia, para el ajuste del modelo se debe considerar el tipo de función kernel que será usada en la función de hipótesis, que sera ajustada de manera iterativa, teniendo en cuenta los datos de entrenamiento; finalmente el modelo puede ser puesto a prueba con los datos de testeo para verificación de los resultados del modelo y así estimar una probabilidad de veracidad del modelo frente al problema en cuestión.

En el proyecto que nos ocupa, se plantea tratar el problema relacionado a el tiempo en que la SVM genera el modelo, haciendo énfasis en el tiempo de computo del algoritmo y obtención de resultados.

1 OBJETIVOS

1.1. OBJETIVO GENERAL

Implementar un algoritmo que permita acelerar los procesos en el problema de la detección de anormalidades en la fermentación del vino mediante máquinas de soporte vectorial (SVM).

1.2. OBJETIVOS ESPECÍFICOS

1. Analizar el algoritmo y base de datos para identificar propiedades concurrentes a los procesos que se puedan acelerar y establecer las métricas de desempeño del algoritmo.
2. Modificar el algoritmo y datos I/O, de manera que se adapte a las necesidades del problema.
3. Seleccionar un mecanismo de implementación que garantice procesos en paralelo del algoritmo para acelerar la ejecución.
4. Evaluar la precisión de los resultados respecto al problema planteado, para verificar el correcto funcionamiento del algoritmo.

1.3. METODOLOGÍA

El proceso metodológico a seguir es el método científico, a través de este se ve el análisis, diseño, implementación y validación del componente de software para la aceleración de la SVM. Las etapas se organizan como en la figura 1.
El proyecto se realizará en 4 fases que se ejecutarán en constante realimentación.

1.3.1. Fase 1: Documentación y bosquejo del diseño de la solución. Se inicia el proceso de documentación e investigación en el área de fermentación del vino, con el fin de analizar y definir la forma en que se re-implementará el algoritmo para que sea de mayor utilidad en la investigación.
Simultáneamente se realizará un proceso análogo en el área de aceleración, para definir la plataforma y/o librerías que se usarán en el desarrollo.

1.3.2. Fase 2: Reformar el algoritmo. Se redefinirá el componente encargado del manejo de lectura de datos y la forma en que se distribuirán sobre los recursos disponibles, y se desarrollara un modelo para la SVM usando la plataforma y/o librerías definidas anteriormente y definir la mejor forma de subdividir los datos y algoritmo, para la paralelización del cómputo.

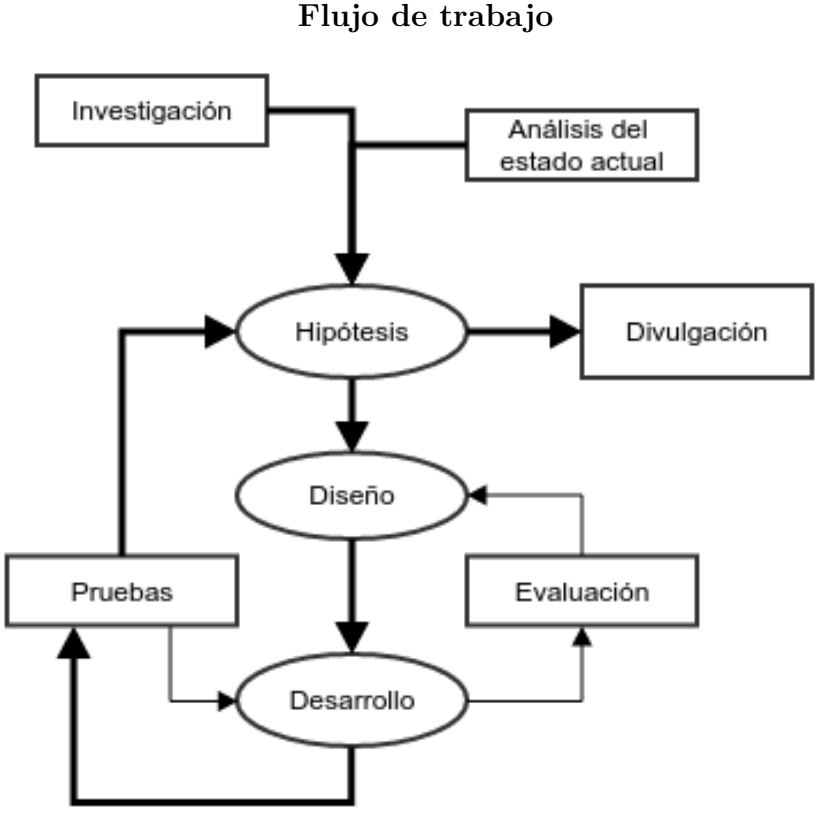

Figura 1: Diagrama de la metodología a usar, corresponde a una adaptación del método científico.

1.3.3. Fase 3: Desarrollo y pruebas del algoritmo HPC. A partir del algoritmo resultante y la subdivisión definidos en la segunda fase, se integrara sobre el(los) lenguajes apropiados para paralelizar, se entrenaran y testearan distintas conjuntos de datos, para realizar pruebas de desempeño y consumo de recursos.

Se realizará un análisis del comportamiento de la aplicación y definirán los lineamientos en los cuales la aplicación se comportará con éxito, la escalabilidad y los recursos de hardware necesarios para un funcionamiento fluido.

1.3.4. Fase 4: Validación y Experimentación. En esta última fase se probarán diferentes entrenamientos y se evaluará su utilidad en la detección de anormalidades en la fermentación del vino, se realimentará y se harán mejoras para lograr resultados más útiles.

A continuación presentaremos una introducción a la teoría de las máquinas de soporte vectorial, para comprender de manera más practica lo que se realiza en este trabajo. Se hablara

de como surge la idea de las SVM, su terminología, junto con la evolución que tuvo este modelo y los problemas que se encontraron en el camino para la construcción de estas.

Nota: Todos los archivos, graficas, imagenes, scripts, papers relacionados con este trabajo pueden ser consultados en `http://forge.sc3.uis.edu.co/redmine/projects/parallel_svm`

2 INTRODUCCIÓN A LAS MÁQUINAS DE SOPORTE VECTORIAL

Las máquinas de soporte vectorial surgieron como un método de clasificación basado en la teoría de minimización del riesgo estructural de Vapnik. En la actualidad, tienen numerosas aplicaciones debido a su versatilidad y a sus prestaciones. Las SVM se han utilizado con éxito en campos como la recuperación de información, la categorización de textos, el reconocimiento de escritura, clasificación de imágenes, entre otros.

Para poder clasificar con las máquinas de soporte vectorial, se comienza realizando una etapa de aprendizaje. Considerando una regresión logística, donde la probabilidad $p(y = 1|x; \theta)$ es modelada por $h_\theta(x) = g(\theta^T x)$. Se tiene entonces una predicción de "1" para una entrada x si y solo si $h_\theta(x) \geq 0{,}5$, o equivalentemente, si y solo si $\theta^T x \geq 0$.

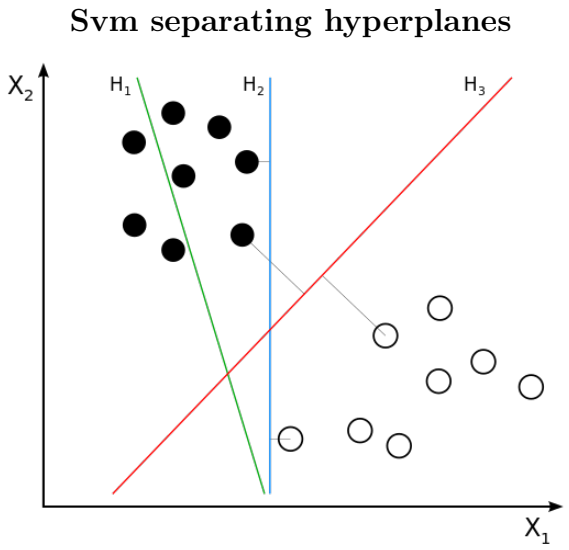

Figura 2: En esta figura se observa un set de datos los cuales son separados por un hiperplano H, en donde H_1 no separa las clases, H_2 las separa, pero solo con un margen pequeño y H_3 las separa con el margen máximo. [24]

Dado un set de entrenamiento, podemos encontrar un buen ajuste para el entrenamiento de los datos (este consiste en encontrar el limite de decisión que mejor separe el set de entrenamiento), si podemos encontrar θ a fin de que $\theta^T x \gg 0$ cuando $y^{(i)} = 1$ y $\theta^T x \ll 0$ cuando $y^{(i)} = 0$, ya que esto refleja un alto grado de confianza y una correcta clasificación para todos los ejemplos de entrenamiento. En la figura 2, esta representando con •'s los ejemplos positivos y denotando con ∘'s los ejemplos negativos, el limite de decisión esta dado por la

ecuación $\theta^T x = 0$ (hiperplano de separación).

2.1. NOTACIÓN

Partiendo de un clasificador lineal para un problema de clasificación binario, notaremos a las etiquetas y y las características como x; usaremos $y \in \{-1, 1\}$ en lugar de $\{0, 1\}$ para denotar las etiquetas de cada clase. También, en lugar de la parametrización de nuestro clasificador lineal con el vector θ, utilizaremos parámetros w, b, y escribir nuestro clasificador como:

$$h_{w,b}(x) = g(w^T x + b)$$

Aquí, $g(z) = 1$ si $z \geq 0$, y $g(z) = -1$ en el caso contrario; con esta notación mantenemos la convención para $x_0 = 1$ y $b = \theta_0$, y por ultimo w toma el rol de $[\theta_1 \cdots \theta_n]$.

2.2. FUNCIÓN DE MARGEN Y MARGEN GEOMÉTRICO

Dado un conjunto de entrenamiento $(x^{(i)}, y^{(i)})$, definimos la función de margen de (w, b) con respecto al set de entrenamiento como:

$$\widehat{\gamma}^{(i)} = y^{(i)}(w^T x + b)$$

Si $y^{(i)} = 1$, entonces para que la función de margen sea grande (en magnitud), es decir, para una predicción confiable, necesitamos que $w^T x + b$ tenga una gran magnitud.

Note que posible reemplazar w con $2w$ y b con $2b$, dado que $g(w^T x + b) = g(2w^T x + 2b)$ no cambia en absoluto el valor de $h_{w,b}(x)$. Es decir, g y por lo tanto $h_{w,b}(x)$, dependen solo del signo y no de la magnitud de $w^T x + b$. Sin embargo, al reemplazar (w, b) por $(2w, 2b)$ la función de margen incrementa en un factor de 2.

Según la teoría de Vapnik, el separador lineal que maximiza el margen (2 veces la distancia al punto más próximo de cada clase) es el que nos da la mayor capacidad de generalización, es decir, la capacidad de distinguir características comunes entre los datos de cada clase, que finalmente permiten clasificar x ejemplo que no pertenece al conjunto de entrenamiento. Este margen nos habla de la confianza que tienen las predicciones de la SVM, en la figura 2 esta representada por la distancia entre el hiperplano y el ejemplo.

El limite de decisión correspondiente a (w, b) es ilustrado a lo largo del vector w, ver figura 3. El vector w es ortogonal al hiperplano de separación. Considere el punto A el cual representa una entrada $x^{(i)}$ del set de entrenamiento con label $y^{(i)} = 1$. La distancia a el limite

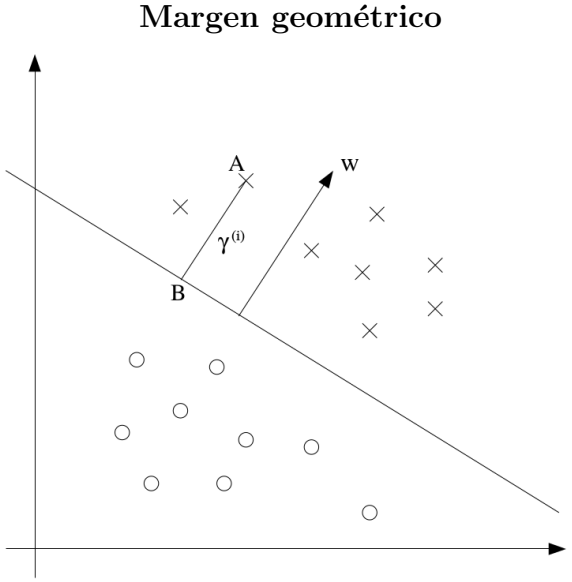

Figura 3: Margen geométrico $\gamma^{(i)}$ de un elemento $x^{(i)}$. [14]

de decisión, $\gamma^{(i)}$, esta dado por el segmento de linea AB.

Partiendo de que $w/\|w\|$ es la unidad de longitud del vector en la misma dirección de w y que A representa $x^{(i)}$, encontrar el punto B esta dado por $x^{(i)} - \gamma^{(i)} \cdot w/\|w\|$. Como el punto B se encuentra en el limite de decisión entonces:

$$w^T\left(x^{(i)} - \gamma^{(i)} \cdot \frac{w}{\|w\|}\right) + b = 0$$
$$\gamma^{(i)} = \frac{w^T x^{(i)} + b}{\|w\|} = \left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}$$

De manera mas general incluyendo los ejemplos tanto positivos como negativos, definimos el margen geométrico de (w, b) con respecto al set de entrenamiento $(x^{(i)}, y^{(i)})$ como:

$$\gamma^{(i)} = y^{(i)}\left(\left(\frac{w}{\|w\|}\right)^T x^{(i)} + \frac{b}{\|w\|}\right)$$

Finalmente dado un set de entrenamiento $S = (x^{(i)}, y^{(i)}); i = 1, \cdots, m$, definimos el margen geométrico de (w, b) con respecto a S, el mas pequeño de los margenes geométricos dentro de los ejemplos de entrenamiento.

$$\gamma = \min_{i=1,\cdots,m} \gamma^{(i)}$$

2.3. CLASIFICADOR DE MARGEN ÓPTIMO

Ya que maximizar el margen geométrico no puede ser resuelto como un problema de optimización puesto que no es convexo. Partimos de que podemos añadir una restricción de

escala arbitraria en w y b sin cambiar nada. Vamos a introducir la restricción de escala que el margen funcional de w, b con respecto al conjunto de entrenamiento debe ser de 1:

$$\hat{\gamma} = 1$$

Al multiplicar w y b por algunos resultados constantes en el margen funcional por la misma constante, esto es de hecho una limitación de escala, puede ser satisfecha por el cambio de escala w, b. Así continuando con el problema anterior, y tomando nota de que la maximización de $\hat{\gamma}/\|w\| = 1/\|w\|$ es lo mismo que minimizar $\|w\|^2$, ahora tenemos el siguiente problema de optimización:

$$\begin{aligned} &\text{mín}_{\gamma,w,b}\ \tfrac{1}{2}\|w\|^2 \\ &\quad \text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1,\ i = 1, \cdots, m \end{aligned}$$

Lo anterior es un problema de optimización con un objetivo cuadrática convexa, con restricciones lineales. Su solución nos da el clasificador margen óptimo.

Podemos escribir la restricción como:

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0$$

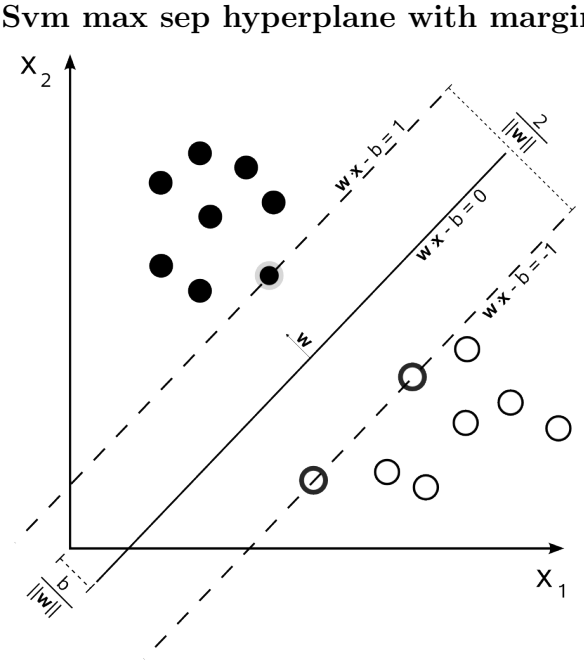

Figura 4: Hiperplano con máximo margen de separación y márgenes para una SVM entrenado con muestras de dos clases. Las muestras en el margen se llaman los vectores de soporte [23].

Tenemos tal restricción para cada ejemplo de entrenamiento. A partir de la condición de doble complementariedad KKT (Karush-Kuhn-Tucker)[1], tendremos $\alpha_i > 0$ sólo para los ejemplos de entrenamiento que tienen margen funcional exactamente igual a uno (es decir, los correspondientes a las restricciones que mantienen con la igualdad, $g_i(w) = 0$).

Considere la figura 4, en la que un hiperplano (margen máximo de separación) se muestra por la línea continua. Los puntos con los márgenes más pequeños son exactamente los más cercanos a la frontera de decisión; aquí, estos son los tres puntos (uno negativo y dos positivos); por lo tanto, sólo tres de los α_i's, a saber, corresponden a estos tres ejemplos de entrenamiento distintos de cero en la solución óptima a nuestro problema de optimización. A estos tres puntos se les llaman **vectores de soporte** en este problema.

Cuando construimos la función de Lagrange para nuestro problema de optimización se tiene:

$$\mathcal{L}(w, b, \alpha) = \tfrac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i \Big[y^{(i)}(w^T x^{(i)} + b) - 1 \Big] \tag{1}$$

Para encontrar el problema dual, es necesario primero minimizar $\mathcal{L}(w, b, \alpha)$ con respecto a w y b, esto es para fijar α; obtener $\theta_{\mathcal{D}}$[2], se resuelve mediante la derivacion de $\mathcal{L}$ con respecto a w y b igual a cero:

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \tag{2}$$

y para la derivada con respecto a b tenemos que:

$$\tfrac{\partial}{\partial b}\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{3}$$

Tomando la definición de w en la ecuación (2) e introduciéndola en la ecuación de Lagrange (1), obtenemos lo siguiente:

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^{m} \alpha_i y^{(i)}$$

Sin embargo teniendo en cuenta la ecuación (3) podemos simplificar el ultimo termino igualándolo a cero.

[1] Lagrange duality
[2] Problema Dual de Lagrange

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}$$

Retomando A la minimización de $\mathcal{L}$ respecto a w y b, junto con la restricción $\alpha \leq 0$ y la ecuación (3), se obtiene el siguiente problema de optimización dual:

$$\begin{aligned} \text{máx}_\alpha \; W(\alpha) &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t. } \alpha &\leq 0, \, i = 1, \cdots, m \\ \sum_{i=1}^{m} \alpha_i y^{(i)} &= 0 \end{aligned}$$

Aquí $\langle x^{(i)}, x^{(j)} \rangle$ es el producto punto entre los vectores en el set de entrenamiento $x^{(i)}$ y $x^{(j)}$.

La ecuación (2), proporciona el valor óptimo de w en términos de α (el valor óptimo). Supongamos que hemos ajustado los parámetros de nuestro modelo a un conjunto de entrenamiento, y ahora queremos hacer una predicción en un nuevo punto x de entrada, entonces podríamos calcular $w^T x + b$, y predecir $y = 1$ si y sólo si esta cantidad es mayor que cero. Pero en la ecuación (2), esta cantidad también puede ser determinada por:

$$w^T x + b = \left(\sum_{i,j=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \tag{4}$$

$$= \sum_{i,j=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \tag{5}$$

Por lo tanto, si hemos encontrado los α_i's, con el fin de hacer una predicción, tenemos que calcular un valor que depende sólo en el producto interno entre x y los puntos en el conjunto de entrenamiento $x^{(i)}$. Por otra parte, hemos visto anteriormente que α_i's todo será cero, excepto para los vectores de soporte (esto es a partir de las KKT). Por lo tanto, muchos de los términos de la suma anterior serán cero, por lo que sólo necesita los productos internos entre x y los vectores de soporte calculado en la ecuación (5) para hacer nuestra predicción.

2.4. KERNEL

Cuando hablamos de kernel, nos referimos a una función la cual nos permitirá mapear los datos de entrada a otro espacio dimensional; un set de entrenamiento x es mapeado a un nuevo set a partir de una operación, estos últimos son pasados al algoritmo de aprendizaje. Denotaremos con ϕ las características de mapeo. En lugar de aplicar las SVM utilizando la entrada original x, es posible aprender usando las características mapeadas de $\phi(x)$. Para ello, simplemente hay que ir sobre nuestro algoritmo anterior, y reemplazar todas las x con

$\phi(x)$.

Dado que el algoritmo puede ser escrito completamente en términos del producto punto entre $\langle x, z\rangle$, esto significa que tendríamos reemplazar todos esos productos punto con $\langle \phi(x), \phi(z)\rangle$. Específicamente, dado un mapeo de características ϕ, se define el kernel correspondiente de la siguiente manera:

$$K(x, z) = \phi(x)^T \phi(z)$$

Entonces, todos los lugares que previamente teníamos $\langle x, z\rangle$ en nuestro algoritmo, podríamos simplemente reemplazarlo con $K(x, z)$, y nuestro algoritmo ahora estaríamos aprendiendo las características de ϕ.

Dado ϕ, es posible calcular fácilmente $K(x, z)$ mediante la búsqueda de $\phi(x)$ y $\phi(z)$ y teniendo su producto punto. Pero lo que es más interesante es que, a menudo, $K(x, z)$ puede ser muy barato para calcular, a pesar de que $\phi(x)$ en sí puede ser muy costoso para calcular (si hablamos de un vector con alta dimensionalidad). Sin embargo podemos obtener una SVM que aprenda el alto espacio dimensional de características dado por ϕ, pero sin tener que buscar de forma explícita o representar los vectores $\phi(x)$.

Supongamos que $x, z \in \mathbb{R}^n$, y consideremos $K(x, z) = (x^T z)^2$; Esto podemos escribirlo como:

$$\begin{aligned} K(x, z) &= \left(\sum_{i=1}^{n} x_i z_i \right) \left(\sum_{j=1}^{n} x_j z_j \right) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i z_i x_j z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

Vemos que $K(x, z) = \phi(x)^T \phi(z)$, donde las características mapeadas por ϕ estan dadas (para el caso de $n = 3$) por:

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

Observemos que mientras el cálculo de alta dimensión $\phi(x)$ tiene un orden de $O(n^2)$, la búsqueda de $K(x, z)$ sólo tiene un orden de $O(n)$ que es un tiempo lineal en la dimensión de los atributos de entrada.

Supongamos que por algún problema de aprendizaje que se está trabajando, se ha llegado a una función $K(x, z)$ tal que arroje una medida razonable de lo similares que son x y z. Por ejemplo:

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

Esta es una medida razonable de la similitud entre x y z, se aproxima a 1 cuando x y z están cerca, y cerca de 0 cuando x y z están muy separados. Este kernel se llama el Kernel Gausseano, y corresponde a una función de mapeo ϕ de dimensión infinita.

Teniendo un conjunto de m puntos $\{x^{(1)}, \cdots, x^{(m)}\}$ y una matriz K de $m \times m$ definida en su posición (i, j) como $K_{i,j} = K(x^{(i)}, x^{(j)})$. Esta matriz es llamada **matriz kernel**. Si K es un kernel valido, entonces $K_{i,j} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T\phi(x^{(j)}) = \phi(x^{(j)})^T\phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{j,i}$ y por lo tanto K es simétrica. Si K es un kernel válido (es decir, si corresponde a alguna característica mapeo ϕ), entonces la matriz kernel correspondiente $K \in \mathbb{R}^{m\times m}$ es simétrica positiva semi-definida.

Teorema (Mercer): Sea $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$. Entonces, para que K sea un kernel válido, es necesario y suficiente, que para cualquier $\{x^{(1)}, \cdots, x^{(m)}\}$, $(m < \infty)$, la matriz del núcleo correspondiente esa simétrica positiva semi-definida, esto es $x^*Kx \geq 0$ para todo $x \in \mathbb{R}^n$ no nulo .

Si se tiene cualquier algoritmo de aprendizaje, el cual puede ser escrito en términos de sólo productos puntos $\langle x, z\rangle$, entre la entrada vectores, entonces reemplazando esto con $K(x, z)$, donde K es un kernel, puede permitir que su algoritmo trabaje de manera eficiente en un espacio de alta dimensionalidad segun corresponda a la función K.

2.5. REGULARIZACIÓN Y CASOS NO SEPARABLES

Si bien los datos de mapeo tienen una alta dimensionalidad en el espacio de características, ϕ generalmente hace aumentar la probabilidad de que los datos sean separables, no podemos garantizar que siempre sea así. Además, en algunos casos, no está claro que la búsqueda de un hiperplano de separación es exactamente lo que nos gustaría hacer, ya que ello podría ser susceptible a los valores atípicos. ver figura 5

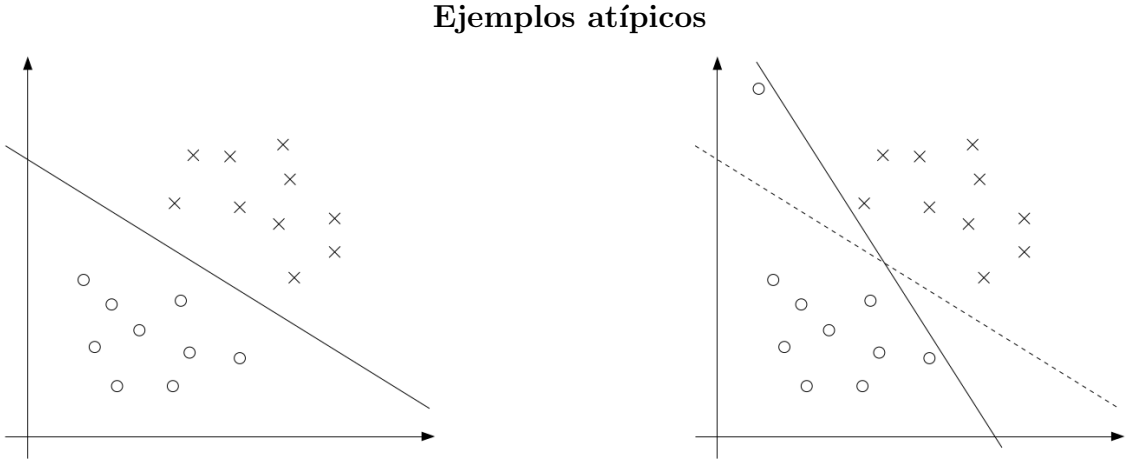

Figura 5: La figura de la izquierda muestra un clasificador margen óptimo, y cuando se añade un solo valor atípico en la región superior izquierda (figura de la derecha), que hace que la frontera de decisión para hacer un giro dramático, y el clasificador resultante tiene un margen mucho más pequeño [14].

Para que el algoritmo funcione para conjuntos de datos no separables linealmente, y sea menos sensible a los valores atípicos, es necesario un parametro de regularización (para el caso de la una SVM se usa la norma l_1):

$$\begin{aligned} &\text{mín}_{\gamma,w,b}\ \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ &\quad \text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i,\ i = 1, \cdots, m \\ &\qquad \xi_i \geq 0,\ i = 1, \cdots, m \end{aligned}$$

Los ejemplos ahora tienen permitido un margen (funcional) inferior a 1, y si existe un ejemplo tiene in margen funcional con $1 - \xi_i$ (con $\xi_i > 0$), se pagaría un costo de la función objetivo con un incrementó dado por $C\xi_i$. El parámetro C controla la ponderación relativa entre el doble objetivo de hacer que el termino $\|w\|^2$ sea pequeño (lo cual maximiza el margen) y asegura que la mayoría de los ejemplos tengan un margen funcional de al menos 1.
En su forma de Lagrange tenemos:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \tfrac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\Big[y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i\Big] - \sum_{i=1}^{m} r_i\xi_i$$

Aquí, la α_i's y r_i's son nuestros multiplicadores de Lagrange (con restricción de ≥ 0). Después de ajustar las derivadas con respecto a w y b a cero como antes, sustituyéndolas de nuevo, y simplificando, obtenemos la siguiente forma dual del problema:

$$\text{máx}_{\alpha}\ W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle$$

$$\text{s.t. } 0 \leq \alpha_i \leq C,\ i = 1, \cdots, m$$
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Las condiciones de doble complementariedad KKT (que en la siguiente sección serán de utilidad para las pruebas para la convergencia del algoritmo SMO) son:

$$\alpha_i = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1 \tag{6}$$
$$\alpha_i = C \Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1 \tag{7}$$
$$0 < \alpha_i < C \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1 \tag{8}$$

2.6. ALGORITMO SMO

El algoritmo SMO (sequential minimal optimization), realizado por John Platt, proporciona una manera eficiente de resolver el doble problema que surge de la derivación de la SVM.

2.6.1. Coordenadas de ascenso. Tratar de resolver el problema sin restricciones, estaría representado de la siguiente manera:

$$\max_{\alpha} W(\alpha_1, \alpha_2, \cdots, \alpha_m)$$

En este caso, pensamos en W como una función con parámetros α_i's, y por ahora ignorar cualquier relación entre este problema y las SVM. El algoritmo que vamos a considerar aquí es llamado coordinación en ascenso:

Loop until converge:{
    For $i, \cdots, m$ {
        $\alpha_i :=$arg $\max_{\hat{\alpha}_i} W(\alpha_1, \cdots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \cdots, \alpha_m)$
    }
}

En el bucle más interior de este algoritmo se llevan a cabo todas las variables, excepto para algunos α_i fijos, y re-optimizarán W sólo respecto a el parámetro α_i. Cuando la función W pasa a ser de una forma tal que la función $argmax$ en el bucle interior se puede realizar de manera eficiente, una coordinación en ascenso puede ser un algoritmo bastante eficiente.

Observe en la figura 6 que en cada paso, coordinar ascenso toma un paso que es paralelo a uno de los ejes, ya que sólo una variable se está optimizando a la vez.

2.6.2. SMO. Cerramos la discusión de las SVM esbozando la derivación del algoritmo SMO. Este es el problema de optimización dual que se quiere resolver:

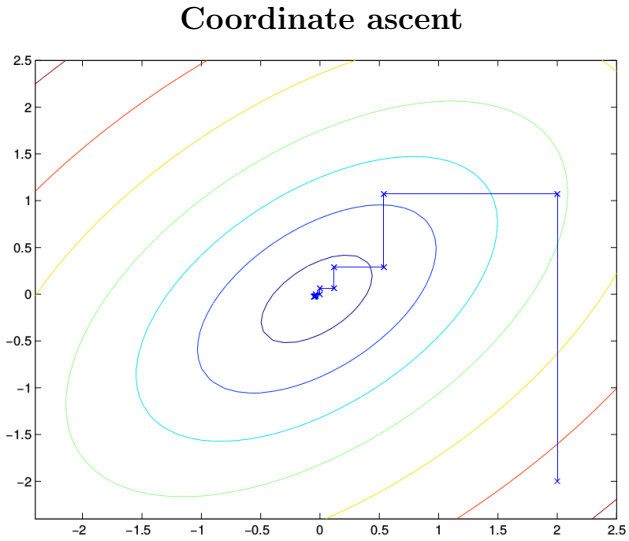

Figura 6: Los puntos suspensivos de la figura son los contornos de una función cuadrática que queremos optimizar. Coordinar ascenso se ha inicializado en (2, -2), y también se representan en la figura es el camino que tomó en su camino hacia el máximo global. [14] .

$$\text{máx}_\alpha \; W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \tag{9}$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \; i = 1, \cdots, m \tag{10}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{11}$$

Tenemos que α_1 se determina exactamente por otra α_i's, pero si mantenemos $\alpha_2, \cdots, \alpha_m$ fijados, entonces no podemos hacer ningún cambio a α_1 sin violar la restricción (11) en el problema de optimización.

Por lo tanto, si queremos actualizar algún α_i, debemos actualizar al menos dos de ellas a la vez con el fin de mantener la satisfacción de las restricciones.

El algoritmo SMO, que simplemente hace lo siguiente:
Repetir hasta la convergencia {

1. Seleccione algún par α_i y α_j para actualizar después (usando una heurística que trata de recoger los dos que nos permitirá conseguir el máximo progreso hacia el máximo global).

2. Re-optimizar $W(\alpha)$ con respecto a α_i y α_j, mientras que se mantienen los demás α_k's $(k \neq i, j)$ fija.

}

Para la prueba de convergencia de este algoritmo, podemos comprobar si las condiciones KKT (ecuaciones 6-8) son satisfechas por un parámetro *tol*, la cual representa la tolerancia de convergencia, y se establece normalmente en torno a 0,01 a 0.001.

Teniendo cierto ajuste de α_i's que satisfacen las restricciones (10-11), supongamos que hemos decidido mantener fijas $\alpha_3, \cdots, \alpha_m$, y se quiere volver a optimizar $W(\alpha_1, \cdots, \alpha_m)$ con respecto a α_1 y α_2 (sujeto a las restricciones). A partir de (11), es necesario que:

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)}$$

Dado que se fija el lado derecho (Como hemos solucionado $\alpha_3, \cdots, \alpha_m$), podemos denotar esto por alguna constante ζ:

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta \tag{12}$$

En la figura 7 se ubican las limitaciones de α_1 y α_2.

Utilizando la ecuación (12), se puede escribir α_1 como una función de α_2:

$$\alpha_1 = \big(\zeta - \alpha_2 y^{(2)}\big) y^{(1)}$$

Como $y^{(1)} \in -1, 1$ entonces $(y^{(1)})^2 = 1$, por lo tanto, el objetivo $W(\alpha)$ se puede escribir:

$$W(\alpha_1, \alpha_2, \cdots, \alpha_m) = W\Big(\big(\zeta - \alpha_2 y^{(2)}\big) y^{(1)}, \alpha_2, \cdots, \alpha_m\Big)$$

El tratamiento de $\alpha_3, \cdots, \alpha_m$ como constantes, la cual es es sólo una función cuadrática en α_2, es decir, este también se puede expresar en la forma $a\alpha_2^2 + b\alpha_2 + c$ para algunos apropiada a, b y c. Si ignoramos de las restricciones de "caja"(10) (o, equivalentemente, a $L \leq \alpha_2 \leq H$), entonces podemos maximizar fácilmente esta función cuadrática mediante el establecimiento de su derivada a cero y resolución.

Vamos a dejar que $\alpha_2^{new,unclipped}$, denote el valor resultante de α_2. Si maximizamos W con respecto a α_2, pero con la limitación de la caja, entonces podemos encontrar el valor óptimo que resulta simplemente tomando α_2 y "recorte"que se encuentran en el intervalo $[L, H]$, para obtener

$$\alpha_2^{new} = \begin{cases} H & \text{si } \alpha_2^{new,unclipped} > H \\ \alpha_2^{new,unclipped} & \text{si } L \leq \alpha_2^{new,unclipped} \leq H \\ L & \text{si } \alpha_2^{new,unclipped} < L \end{cases}$$

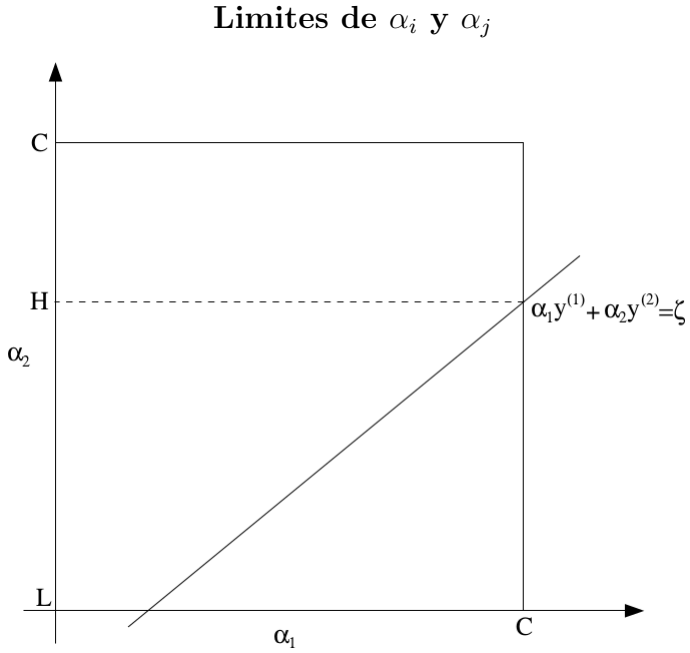

Figura 7: A partir de las restricciones en la ecuación (10), α_1 y α_2 estará situado dentro de la caja $[0, C] \times [0, C]$. También traza es la línea $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$, en la que como sabemos a α_1 y α_2 deben estar. A partir de estas limitaciones, sabemos $L \leq \alpha_2 \leq H$; De lo contrario, (α_1, α_2) no puede satisfacer simultáneamente la restricción de caja y ni la restricción de línea recta. En este ejemplo, $L = 0$, sin embargo, dependiendo de la línea $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$, esto no siempre es así; sino más en general, habrá algún límite inferior L y algún límite superior H de los valores permisibles para α_2 que se asegurará de que α_1 y α_2 se encuentren dentro de la caja $[0, C] \times [0, C]$ [14] .

Finalmente, habiendo encontrado la α_2^{new}, podemos utilizar la ecuación (12) para volver y encontrar el valor óptimo de α_1^{new}.

En este capitulo hemos realizado un recorrido a lo largo del modelo matemático que fundamenta la eficacia de las máquinas de soporte vectorial, pasando desde la entrada que para este modelo, el desarrollo que es efectuado para el aprendizaje y terminando con la predicción de nuevos datos; también se hablo sobre como dentro de las SVM, se construye un modelo para un problema dado.

A continuación serán nombrados algunos de los mecanismos de aceleración teniendo en cuenta diferentes arquitecturas y hardware que se tiene a disposición en la actualidad, los cuales han sido tenidos en consideración para este trabajo.

3 COMPUTACIÓN EN PARALELO

3.1. MPI (Message Passing Interface)

MPI es una librería de especificaciones de interfaz para paso de mensajes. MPI hace referencia principalmente al modelo de programación de paso de mensajes, en el que los datos se mueven dentro de un espacio de direcciones, desde un procesador a cualquier otro a través de operaciones de cooperación en cada proceso. Las extensiones del modelo de paso de mensajes çlásica", se proporciona el las operaciones colectivas, operaciones de acceso de memoria remota, la creación de procesos dinámicos y entradas y salidas en paralelo. MPI no es un lenguaje, y todas las operaciones MPI se expresan como funciones, subrutinas o métodos, de acuerdo con los enlaces de lenguaje adecuadas en las que, para C y Fortran, son parte del estándar MPI [12].

3.2. OpenMP (Open Multi Processing)

OpenMP es una interfaz de programación de aplicaciones (API). OpenMP proporciona un modelo portátil, escalable para los desarrolladores de aplicaciones paralelas de memoria compartida. La API es compatible con C / C ++ y Fortran en una amplia variedad de arquitecturas. Consiste en un conjunto de directivas de compilación, rutinas de biblioteca, y las variables de entorno que influyen en el comportamiento en tiempo de ejecución.

OpenMP es una implementación de múltiples hilos, un método de paralelización por el que un hilo maestro (una serie de instrucciones ejecutadas de forma consecutiva) bifurca en un número especificado de hilos .esclavosz el sistema divide una tarea entre ellos. Los hilos se ejecutan simultáneamente, con el entorno de ejecución de la asignación de hilos a diferentes procesadores.

De forma predeterminada, cada hilo ejecuta la parte de paralelizado de código de forma independiente. Trabajo compartido construcciones se puede utilizar para dividir una tarea entre los hilos de manera que cada hilo ejecuta su parte asignada del código. Se puede lograr paralelismo tanto para tareas como de datos.

El entorno de ejecución asigna hilos a los procesadores dependiendo del uso, carga de la máquina y otros factores. El entorno de ejecución puede asignar el número de hilos en base a las variables de entorno, o el código se puede hacer por medio de funciones. Las funciones de OpenMP se incluyen en un fichero de cabecera marcado omp.h en C / C ++ [2] [17].

3.3. OpenACC (Open Accelerators)

El Application Program Interface OpenACC describe un conjunto de directivas del compilador para especificar los bucles y las regiones de código en C estándar, C++ y Fortran para ser descargado desde una CPU host a un acelerador adjunto. OpenACC está diseñado para la portabilidad a través de los sistemas operativos, procesadores de acogida, y una amplia gama de aceleradores, incluyendo APU, GPU, y coprocesadores de múltiples núcleos.

El modelo de directivas y la programación definida en el documento API OpenACC permite a los programadores para crear programas de acogida junto con aceleradores de alto nivel sin la necesidad de inicializar explícitamente el acelerador, gestionar los datos o transferencias de programas entre el anfitrión y el acelerador, o iniciar el arranque y la parada del acelerador.

Todos estos detalles son implícitas en el modelo de programación y son manejadas por los compiladores y los tiempos de ejecución de la API habilitado OpenACC. El modelo de programación permite al programador para aumentar la información disponible para los compiladores, incluyendo la especificación de datos local para un acelerador, orientación acerca de la asignación de los bucles en un acelerador, y detalles similares de rendimiento relacionados [16].

3.4. CUDA

CUDA es un modelo de plataforma de computación y programación paralela inventado por NVIDIA. Permite a un aumento espectacular en rendimiento informático aprovechando el poder de la unidad de procesamiento gráfico (GPU). Proporcionar un pequeño conjunto de extensiones a los lenguajes de programación estándar, como C, que permiten a una aplicación directa de algoritmos paralelos.

La CPU y la GPU son tratados como dispositivos independientes que tienen sus propios espacios de memoria. Esta configuración permite el cálculo simultáneo de la CPU y la GPU sin contención de recursos de memoria. Las GPU CUDA tienen cientos de núcleos que pueden funcionar colectivamente miles de hilos de computación. Estos núcleos han compartido recursos que incluyen un archivo de registro y una memoria compartida. La memoria compartida en chip permite que las tareas paralelas que se ejecutan en estos núcleos para compartir datos sin enviarlo a través del bus de memoria del sistema [10].

3.5. OpenCL (Open Computing Lenguage)

Open Computing Language (OpenCL) es un marco para escribir programas que se ejecutan a través heterogéneas plataformas que constan de unidades centrales de procesamiento

(CPU), unidades de procesamiento gráfico (GPU), procesadores de señales digitales (DSP), matrices de puertas programables en campo (FPGAs) y otros procesadores o aceleradores de hardware. OpenCL especifica un lenguaje de programación (basado en C99) para la programación de estos dispositivos y las interfaces de programación de aplicaciones (API) para el control de la plataforma y ejecutar programas en los dispositivos de cómputo. OpenCL proporciona una interfaz estándar para la computación paralela usando basado en tareas y el paralelismo basado en los datos [11] [21].

OpenCL define cuatro niveles de jerarquía de memoria para el dispositivo de cálculo:

- Memoria global: compartido por todos los elementos de procesamiento, pero tiene alta latencia de acceso.
- Memoria de sólo lectura: más pequeño, baja latencia, puede escribir en la CPU host, pero no los dispositivos de cómputo.
- Memoria local: compartido por un grupo de elementos de procesamiento.
- Memoria privada por elemento (registros).

4 TRABAJOS RELACIONADOS

En este capitulo mencionaremos algunos de los trabajos realizados por otros investigadores, relacionados con la aceleración de este modelo matemático (las SVM), quienes usando diferentes estrategias y mecanismos, cumplen su objetivo de disminuir el tiempo de procesamiento, manteniendo la fidelidad de los resultados arrogados en su implementación.

En [6] se muestra cómo las SVM pueden ser paralelizados para lograr un rendimiento escalable. En este articulo presentan el software PSVM, el cual distribuye la carga los datos de entrenamiento en máquinas paralelas, reduciendo los requisitos de memoria a través de factorización aproximada de la matriz del kernel. El código realizado esta disponible en `http://code.google.com/p/psvm/`.

En el 2011 los autores en [1] se presenta una versión de la biblioteca LIBSVM asistida por GPU para máquinas de soporte vectorial. Aquí ellos realizan el calculo la matriz kernel en la GPU, disminuyendo el tiempo de procesamiento para la formación de la SVM sin alterar los resultados de la clasificación en comparación con el LIBSVM original. La figura 8 ilustra como interactúa la GPU en la LIBSVM.

En [13] buscan combatir el problema relacionado del tiempo para conjuntos de datos muy grandes. Para mitigar esto, parten del trabajo propuesto en [7] para ejecutar una SVM en cascada. Es un procedimiento sencillo, paso a paso, en el que la SVM se entrenó de manera iterativa en subconjuntos del conjunto de datos, los vectores de soporte resultantes se combinan para crear nuevos conjuntos de entrenamiento. Al ser independientes, este proceso es realizado mediante un simple método de embolsado paralelo. También se introduce un nuevo enfoque por etapas que usa la paralelización de una manera mejor que la cascada SVM y contiene una parada en tiempo de adaptación logrando de esta manera mejorar la precisión.

En el siguiente capitulo presentaremos un trabajo realizado por investigadores en ciudad de Viña del Mar de Chile, quienes impulsados a combatir un problema en su localidad (la detección de anormalidades en las fermentaciones vínicas), hacen uso de una maquina de soporte vectorial; El trabajo presentado en esta tesis parte del trabajo realizado por ellos, contando con su aprobación y colaboración.

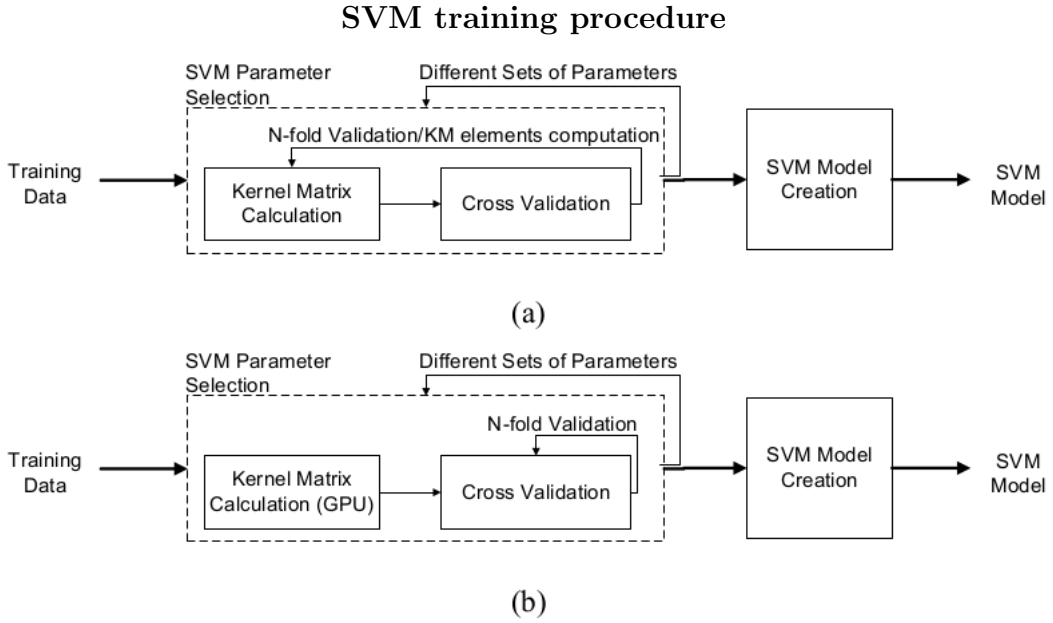

Figura 8: Proceso de entrenamiento dado por (a) LIBSVM original y la (b) GPU-Accelerated LIBSVM [1]

5 PRIMERA ETAPA

En [19] se construyo una base de datos con diferentes variables quimicas, las cuales se encuentran durante la fermentación del vino y adicional a esto, desarrollan un código en lenguaje python, donde usando la librería sklearn [18], replican el trabajo realizado en [9], usando las funciones para implementar una SVM, las cuales están basadas en la LIBSVM [5].

5.1. ANÁLISIS DE LA BASE DE DATOS Y EL ALGORITMO

Partiendo de este trabajo y siguiendo los objetivos planteados para esta tesis, se realiza un análisis del trabajo previo, para identificar mas claramente los procesos realizados en su implementación, se crea diagrama de flujo, encontrando de manera mas sencilla, los posibles procedimientos que pudieran ser mejorados y/o corregidas.

Debido a que la base de datos obtenida finalmente, no contaba con la totalidad de la información correspondiente, realizan una interpolación de los datos, obteniendo una base de datos con mayor cantidad de información; para ello utilizan las librerías scipy (*interpolate*, *barycentric_interpolate*) y numpy (*chebyshev*).

Posteriormente generan un nuevo documento a partir de la base de datos que sera usado por el código que implementara la SVM y a su vez diferentes archivos los cuales son tomados como entrada en la SVM. Ver figura 9. Aquí filtran los datos de entrada obteniendo únicamente las variables de la fermentación que se desean y obtiene el valor respectivo de cada una de las fermentaciones; este script crea un archivo 'example.xls' el cual guarda toda la información obtenida.
Una vez realizado el filtro de los datos a usar, ejecutan el script svm_test.py el cual realiza el muestreo aleatorio según la combinación necesaria y posteriormente entrena la svm y finalmente lee nuevamente los datos para realizar el testing de los datos correspondientes. ver figura 10.

Lectura de datos

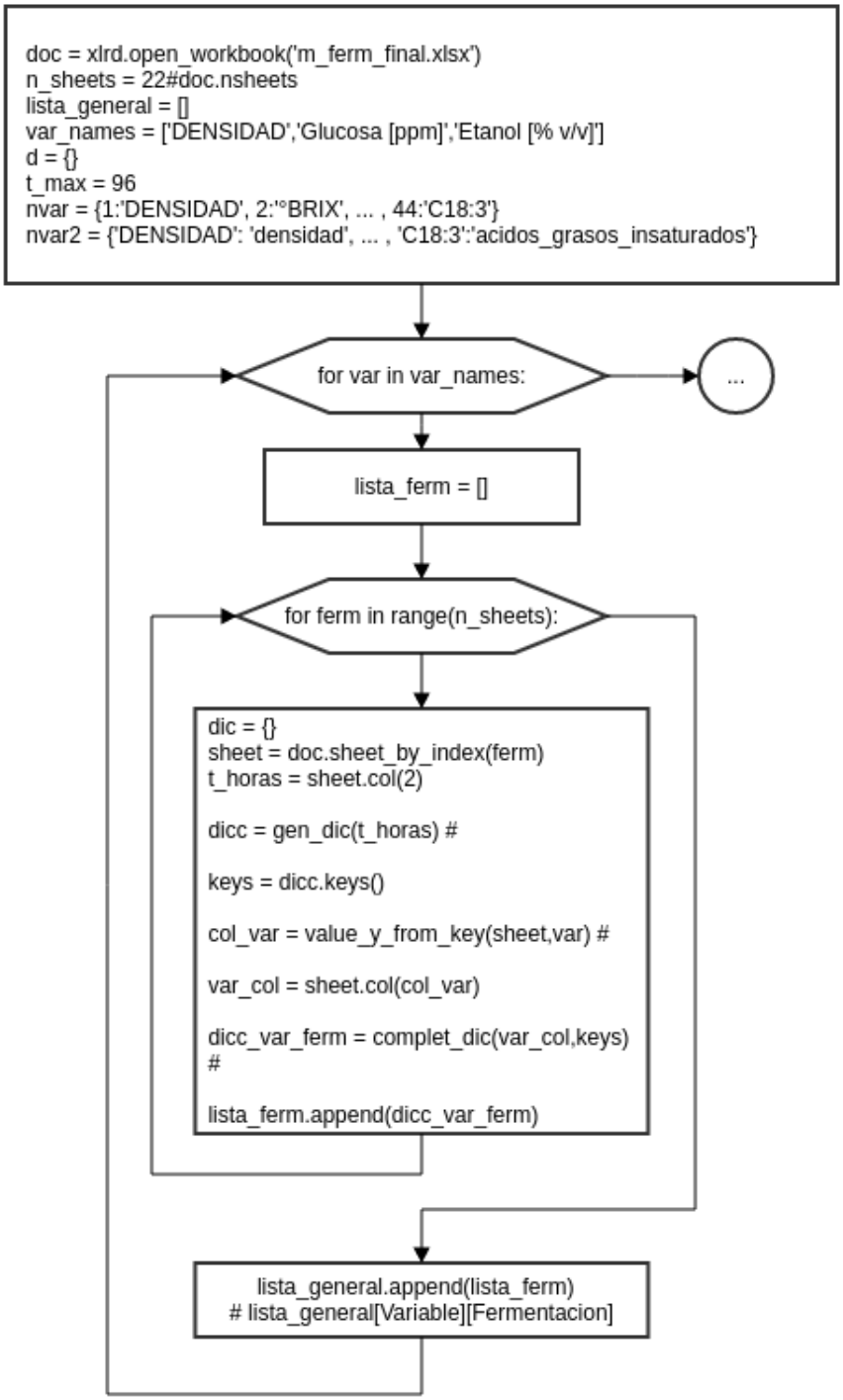

Figura 9: Diagrama de flujo correspondiente al archivo readData.py

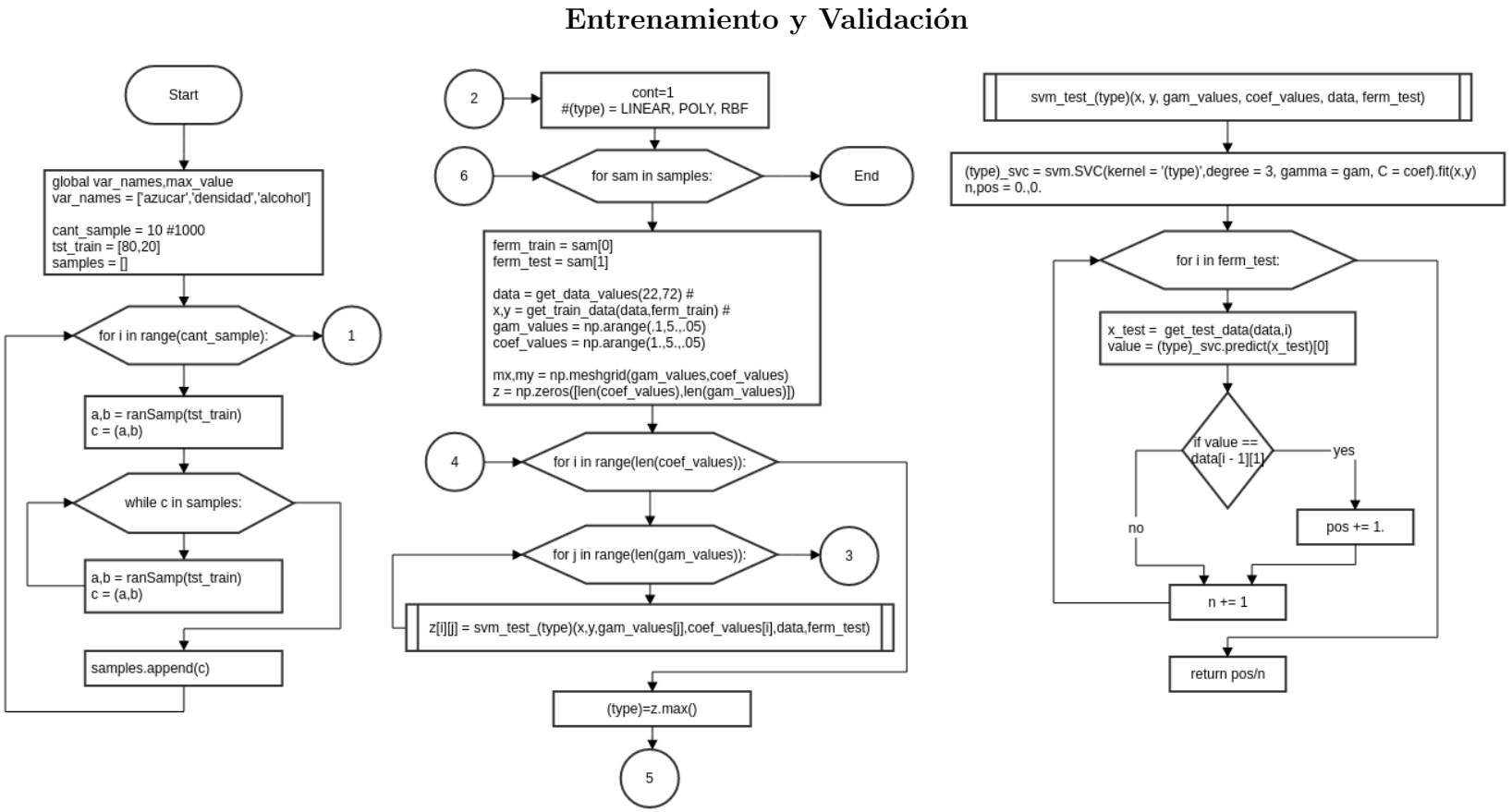

Figura 10: Diagrama de flujo correspondiente al archivo svm_test.py

Archivos .CSV (Comma Separated Values)

(a) input.csv

	A	B	C	D	E
1	0	1113.5571272039	26.6	5.6424423184	338.338
2	24	1105.8451816746	25.3	6.8399390694	312.312
3	48	1061.6113744076	19.4	7.2027399918	0
4	72	1036.1374407583	14.5	10.1076844408	261.6796
5	96	1009.8736176935	11.2	7.5782262612	290.0716
6	120	998.7164296998	9.1	7.4214759061	238.7644
7	0	1113.5571272039	26.6	5.6424423184	338.338
8	24	1103.7717219589	25.3	6.9997744408	287.7056
9	48	1056.7733017378	18.4	7.1621468816	0
10	72	1027.2511848341	13.2	9.7321981714	284.3932
11	96	1002.6658767773	9.8	7.9638608082	308.0532
12	120	990.0276461295	8.1	6.6886081347	285.8128
13	0	1113.5571272039	26.6	5.6424423184	338.338
14	24	1105.6477093207	25	7.2230365469	285.3396

(b) label.csv

	A	B
1	1	
2	1	
3	1	
4	1	
5	1	
6	1	
7	1	
8	1	
9	1	
10	1	
11	1	
12	1	
13	1	
14	1	

Tabla 1: Ejemplo de los archivos .csv

5.2. ADAPTACIÓN DEL TRABAJO

Analizando el trabajo realizado en [19], se encuentra que existe una intensa consulta de los datos, es por ello que se crea un archivo 'input.csv', cuyo formato (comma-separated values), es de mas fácil acceso; este archivo se construye manteniendo un orden respecto a sus columnas, ya que cada columna representa cada una de las variables de la base de datos principal, en la primera columna se mantiene la variable que representa la hora en que fueron tomadas las muestras, y las demás columnas según sea el caso de las variables que se quieran utilizar; es importante mantener la variable hora debido a que existe una relación entre las demás variables y esta.

Revisando en mas detalle la base de datos y el código implementado, se encuentra que en la base de datos están representados los 3 tipos de fermentaciones (stuck, slow y normal), característica que se esta omitiendo en [19], los cuales trabajan toda la base de datos con 2 clasificaciones («slow-stuck»y normal).

También se crea otro archivo 'label.csv' que contiende el "label"de cada una de las filas ("ejemplos") en el archivo 'input.csv' (ver tabla **??**); conservando la coherencia entre a la posición que comparten el ejemplo y su clasificación. Este procedimiento se realizo sin ninguna ejecución de código.

5.3. RECONSTRUCCIÓN DEL ALGORITMO

A partir de esto se desarrolla un script que toma los datos en el input.csv y label.csv y genera dos nuevos archivos, uno para el entrenamiento de la svm y el otro para la verificación del modelo generado.

Algorithm 1 Crear archivos para training y testing

Require: *data*: datos en el archivo input.csv, *data_l*: datos en el archivo label.csv

```
1:  procedure CREATE_FILES(data, data_l)
2:      for line in data do
3:          if random() > 2.0 then
4:              training.write(line)
5:              training_l.write(data_l.readline())
6:          else
7:              testing.write(line)
8:              testing_l.write(data_l.readline())
9:          end if
10:     end for
11:     save training.csv, training_l.csv
12:     save testing.csv, testing_l.csv
13: end procedure
```

Una vez teniendo los archivos de training y testing se ejecuta el un nuevo script que implementa una svm multiclase, usando la librería sklearn para python; este script carga los datos de training y testing en una matriz, consecutivamente se realiza el entrenamiento usando el modelo One-Vs-One (Uno contra uno), seguido de realizar el testing.

Algorithm 2 Implementación de la SVM multiclase

Require: $X\_training, y\_training, X\_testing, y\_testing$: Estas variables hacen referencia a las matrices que representan los archivos de training y testing con su respectivo label

```
    procedure SVM MULTICLASS(X_training, y_training, X_testing, y_testing)
2:      ovo = SVC(decisión_function_shape='ovo', probability=True, kernel = str(args.k),
    gamma = gamma , C = coef)
        ovo.fit(X_training, y_training)
4:      pp = ovo.predict_proba(X_testing)
        create_file(pp, 'predict_proba_OVO.csv')
6:      score = ovo.score(X_testing, y_testing, sample_weight=None)
    end procedure
```

5.4. RESULTADOS DE LA PRIMERA ETAPA

Los experimentos se llevaron a cabo en una maquina equipada con un procesador Intel Core i3-5010U CPU de cuatro núcleos con 4 GB de memoria RAM DDR3, con una velocidad de procesamiento de 2.1 GHz, un sistema operativo Linux-x86_64 Ubuntu 16.04.

El código anterior realizar aproximadamente 10000 dado que el algoritmo implementado modifica para cada set de entrenamiento los valores de γ y el coeficiente regulador C; esto lo hace para los modelos rbf, polinómico y lineal.

Para el set de entrenamiento donde se tiene en cuenta únicamente las variables de densidad, azúcar y alcohol se tiene que el tiempo total en ejecutar estos modelos es de 7105.56543136 segundos esto equivale a que para cada modelo con determinado γ y constante C, se toma un tiempo de 0.7105565431 segundos.

Con el algoritmo realizado en este trabajo como prueba inicial se tiene que en promedio el ajuste del modelo con determinado γ y constante C, se toma un tiempo de 0.2420080278 lo que acelera el procesamiento en un factor de **2.93**

5.5. DISCUSIÓN

A diferencia del algoritmo presentado en [19], en las predicciones se ve reducida la exactitud de clasificación; esto puede deberse a que se están considerando los tres tipos de estados de la fermentación (parada, lenta y normal), y que los vectores de entrenamiento son entregados a la SVM en un formato diferente, razón por la cual el algoritmo presentado aquí también es mas eficaz en términos de velocidad de procesamiento.

6 SEGUNDA ETAPA

En el transcurso del desarrollo del trabajo mencionado en el capitulo anterior, surge la idea de trabajar con el algoritmo implementado en la librería LIBSVM, ya que esta no solo es la que se esta implementado en sklearn de python, sino que es nombrada en diferentes textos, esto impulsa la idea de acelerar esta librería, y de esta manera, hacer un mejor uso de los recursos disponibles de las diferentes arquitecturas frente a la gran cantidad problemas que pueden ser resueltos por la librería.

Para poder cumplir con el objetivo de acelerar este algoritmo, fue imprescindible entender como funciona el modelo matemático que hay detrás de las máquinas de soporte vectorial (ver capitulo 2) . Esto tiene su importancia no solo en nos que permite encontrar que procesos son secuenciales, repetitivos, y sus respectivas dependencias, sino también, en la necesidad verificar la coherencia y correcto funcionamiento del trabajo que se realiza.

"Es claro que para acelerar un algoritmo no es explícitamente necesario entender que esta haciendo este, sino, identificar los procesos repetitivos y dependencias."

6.1. LIBRERÍA LIBSVM

Una vez comprendido de manera general la matemática detrás de las SVM, se procede a estudiar de manera conjunta la estructura del código implementado en la LIBSVM [5].

Este librería esta implementada en diferentes lenguajes, entre ellos C y C++, en el cual se desarrolla esta tesis. La librería consta de 7 archivos, para el caso de C y C++, estos son:

a) README b) Makefile c) svm.h d) svm.cpp

e) svm-train.c f) svm-predict.c g) svm-scale.c

Aquí nos enfocamos principalmente en el entrenamiento del modelo y sus dependencias. Inicialmente se identifican los procesos que pueden ser ejecutados por una unidad de procesamiento gráfico GPU, procesos que deben cumplir con la propiedad de atomicidad, de tal manera que pueda ser asegurada la realización o no de la operación.
Los procesos identificados constan principalmente de ciclos en donde se realizan operaciones matemáticas, como sumatorias, y operaciones vectoriales.

6.2. COMPILADORES PGI

Una vez identificados estos procesos, se realizan los cambios en los respectivos scripts de la librería, pasando por el Makefile, svm.cpp y svm-train.c;

Se opta por usar los compiladores PGI [15], los cuales nos permiten efectuar trabajos tanto en los diferentes núcleos de la CPU por medio de la API OpenMP, como en las unidades de procesamiento gráfico a partir de llamados a la API OpenACC; es de esta manera que se realiza un script híbrido, que nos permite hacer un mejor uso de los recursos.

El archivo Makefile configurado para hacer uso de los compiladores PGI queda de la siguiente manera:

```
PGXX ?= pgc++
PGFLAGS = -acc -mp=allcores -ta=nvidia:cuda7.5,cc35,time -Minfo=accel,mp -Kieee
SHVER = 2
OS = $(shell uname)

all: svm-train svm-predict svm-scale

lib: svm.o
	if [ "$(OS)"="Darwin" ]; then \
		SHARED_LIB_FLAG="-dynamiclib -Wl,-install_name,libsvm.so.$(SHVER)"; \
	else \
		SHARED_LIB_FLAG="-shared -Wl,-soname,libsvm.so.$(SHVER)"; \
	fi; \
	$(CXX) $ $SHARED_LIB_FLAG svm.o -o libsvm.so.$(SHVER)

svm-predict: svm-predict.c svm.o
	$(PGXX) $(PGFLAGS) svm-predict.c svm.o -o svm-predict -lm
svm-train: svm-train.c svm.o
	$(PGXX) $(PGFLAGS) svm-train.c svm.o -o svm-train -lm
svm-scale: svm-scale.c
	$(PGXX) $(PGFLAGS) svm-scale.c -o svm-scale
svm.o: svm.cpp svm.h
	$(PGXX) $(PGFLAGS) -c svm.cpp
clean:
	rm -f * svm.o svm-train svm-predict svm-scale libsvm.so.$(SHVER)
```

Las banderas en la compilación que se usan actualmente de la siguiente manera:

-acc Esta opción habilita las directivas de OpenACC
-mp Con esta opción compilamos programas usando memoria compartida (OpenMP)
-ta Es usada para especificar el tipo de arquitectura GPU que sera usada.
-Minfo Indica al compilador que muestre la información sobre el error estándar y de los procesos en paralelo que serán realizados; también emite las estadísticas de compilación.

-Kieee Esta opción nos permite realizar operaciones de punto flotante en estricta conformidad con la norma IEEE 754 y se utiliza una biblioteca matemática más exacta.

6.3. GPROF

Se usaron diferentes profiler's tales como PGPROF y NVIDIA Visual Profiler, con el animo de encontrar el porcentaje correspondiente al tiempo de ejecución de los diferentes procedimientos y funciones durante toda la ejecución de este; sin embargo estos no nos entregaban una información detallada de los tiempos de ejecución.

Finalmente hicimos uso del profiler que viene junto con los paquetes de C y C++, gprof; este profiler nos retorno una información bastante precisa y detallada respecto a los procesos que se están ejecutando y cuanto tiempo tardan en culminar.

Para poder hacer uso del profiler gprof, fue necesario modificar el archivo Makefile agregando la opción $-pg$, en la linea correspondiente a CFLAGS, el cual hará efecto en cada una de las compilaciones del programa que finalmente serán enlazados; Esto es en el script secuencial tomado de [5].

CFLAGS = -Wall **-pg** -Wconversión -O3 -fPIC

Una vez hecho esto, se ejecuta el programa compilado, el cual genera un archivo llamado gmon.out que sera usado con el siguiente comando, que creará un archivo analysis.txt que contendrá los tiempos de ejecución de cada una de las subrutinas, la cantidad de veces que es ejecutada y su respectivo porcentaje del tiempo total en la ejecución.

gprof svm-train gmon.out > analysis.txt

Es así como pudimos comprobar que funciones requerían de una mayor atención en la ejecución del algoritmo, y como era de esperarse (según el estudio previamente realizado), con un 97 % del tiempo total de ejecución, este proceso se encarga de operar y devolver una sección de la matriz kernel; esta matriz es consultada y reajustada en diferentes ocasiones debido a que es un procedimiento indispensable para resolver el problema de minimización (gradiente descendiente).

Este operación consiste en el producto punto de todos contra todos los ejemplos de entrenamiento (ver ecuación (6-1)), este proceso es realizado por el algoritmo 3

$$X^{(i)} \cdot X^{(j)} = \begin{bmatrix} X_0^{(i)} & X_1^{(i)} & ... & X_m^{(i)} \end{bmatrix} \begin{bmatrix} X_0^{(j)} \\ X_1^{(j)} \\ ... \\ X_m^{(j)} \end{bmatrix} = X_0^{(i)} X_0^{(j)} + X_1^{(i)} X_1^{(j)} + ... + X_m^{(i)} X_m^{(j)} \tag{6-1}$$

6.4. OpenACC

Debido a la no atomicidad del producto punto realizado en este algoritmo, no es posible acelerarlo mediante el uso de una GPU.

Esta atomicidad en el proceso se hace evidente en el *while loop* y en el if.

Algorithm 3 Producto punto entre $X^{(i)}$ y $X^{(j)}$

Require: $*px, *py$: Apuntador al ejemplo de entrenamiento con (index:característica y value:valor)

Ensure: sum: Indica el valor del producto punto entre los ejemplos px y py

```
1:  procedure KERNEL::DOT(*px, *py)
2:      sum = 0
3:      while ((px->index) != -1) && ((py->index) != -1) do
4:          if (px->index) == (py->index) then
5:              sum += (px->value) * (py->value)
6:              ++px
7:              ++py
8:          else
9:              if (px->index) > (py->index) then
10:                 ++py
11:             else
12:                 ++py
13:             end if
14:         end if
15:     end while
16: return sum
17: end procedure
```

Para hacer uso de OpenACC en scripts en el lenguaje lenguaje de C++, es importante tener claro si el proceso a paralelizar se encuentra dentro de una clase o una subrutina, ya que esto puede generar errores en la consulta de los datos por el dispositivo GPU.

Se desarrollan tres nuevas funciones, *reduce_acc_const*, *reduce_acc_vect* y *reduce_gradient_acc* (algoritmos 4, 5 y 6 respectivamente), las cuales por medio de las clausulas *#pragma acc parallel loop*, se comunica con la unidad de procesamiento gráfico en donde se computaran las operaciones inmediatamente siguientes a estas; este proceso es ejecutado en los cores de la GPU.

Al realizar los cambios en los procesos identificados, se prosigue a realizar un test con el dataset protein, para observar la fidelidad y el correcto funcionamiento del algoritmo, esperando conjuntamente una aceleración en su ejecución; esto ultimo no pudo ser evidenciado, debido a que los procesos de comunicación con el dispositivo GPU.

Algorithm 4 Sumatoria de un vector

Require: *int* n, *double* a[ ], *double* y[ ], *double* [ ], *double* summ

1: **procedure** REDUCE_ACC_CONST
2: #pragma acc parallel loop pcopyin($a[0:n], y[0:n], p[0:n], n$) reduction($+ : summ$)
3: **for** *int* $i = 0; i < n; ++i$ **do**
4: summ += a[i] ∗ (y[i]+p[i]);
5: **end for**
6: #pragma acc wait
7: **end procedure**

Algorithm 5 Sumatoria el elemento i-esimo de un vector

Require: *int* n, *double* a, *const Qfloat* *x, *double* y[ ]

1: **procedure** REDUCE_ACC_VECT
2: #pragma acc parallel loop pcopyin($x[0:n], a, n$) pcopy($y[0:n]$)
3: **for** *int* $i = 0; i < n; ++i$ **do**
4: y[i]+ =a∗x[i];
5: **end for**
6: #pragma acc wait
7: **end procedure**

Algorithm 6 Sumatoria el elemento i-esimo del vector gradiente

Require: *int* n, *double* dai, *double* daj, *const Qfloat* *xi, *const Qfloat* *xj, *double* y[ ]

1: **procedure** REDUCE_GRADIENT_ACC
2: #pragma acc parallel loop pcopyin($xi[0:n], xj[0:n], dai, daj, n$) pcopy($y[0:n]$)
3: **for** *int* $i = 0; i < n; ++i$ **do**
4: y[i]+ = (dai∗xi[i]) + (daj∗xj[i]);
5: **end for**
6: #pragma acc wait
7: **end procedure**

6.5. OpenMP

El algoritmo 3 esta construido de esta manera ya que la librería utiliza matrices sparse desde la entrada de los datos, lo que es bastante útil ya que se combate el desbordamiento de memoria, el cual es un problema cuando se tienen problemas con una gran dimensión, tanto de características como de ejemplos.

Este proceso fue posible paralelizarlo usando la API de OpenMP en desde la función *get_Q* (algoritmo 7), que es implementada por las clases SVC_Q, ONE_CLASS_Q y SVR_Q. Esta subrutina del algoritmo realiza un ciclo *for* el cual es independiente en cada iteración, en donde se llama a la subrutina $Kernel :: dot$ (función representada en el algoritmo 3) que pertenece a la clase $Kernel$.

Es aquí donde añadimos la nueva sentencia (previamente al ciclo *for*) esta sentencia usa las clausulas *#pragma omp parallel loop*, dividiendo así el trabajo realizado por un solo core de CPU entre los aquellos que han sido configurados en la compilación. Esto es por medio de las variables de entorno la cual puede ser inicializada de la siguiente manera desde el terminal *export* OMP_NUM_THREADS=NN donde NN corresponde al numero de cores que serán usados.

Algorithm 7 Subrutina get_Q

Require: *int* i, *int* len
Ensure: data : Vector con los valores del producto punto

```
1:  procedure CLASE::GET_Q
2:      Qfloat *data;
3:      int start, j;
4:      if (start=cache− >get_data(i,&data,len) < len then
5:          #pragma omp parallel for
6:          for j = start; j < len; j++ do
7:              data[j] = (Qfloat)(y[i]∗y[j] ∗ (this− >*kernel_function)(i,j));
8:          end for
9:      end if
10:     ⋮
11: end procedure
```

6.6. RESULTADOS DE LA SEGUNDA ETAPA

Los experimentos se llevaron a cabo en una maquina equipada con un procesador Intel Core i7-3630QM CPU de ocho núcleos con 4 GB de memoria RAM DDR3, con una velocidad de procesamiento de 3.4 GHz, un sistema operativo Linux-x86_64 Ubuntu 16.04 y una tarjeta gráfica NVDIA GeForce 920M con 2048 MB de RAM, esta ultima posee un total de 384 cores CUDA.

Los parámetros se utilizan en su configuración predeterminada, estos es que el algoritmo utiliza un kernel Gaussiano y un metodo para multiples clases. Los dataset usados para esta prueba donde se hace uso de la LIBSVM con compiladores PGI, se muestran en la tabla 2.

Datasets

Dataset	**No. de clases**	**No. Trainings**	**No. Testings**	**No. Características**
mnist	10	60000	10000	780
letter	26	15000	5000	16
protein	3	17766	6621	357
rcv1	53	15564	518571	47236
vehicle	3	78823	19705	100

Tabla 2: Datasets usados en la LIBSVM

Aceleración dada por la CPU + GPU NVIDA

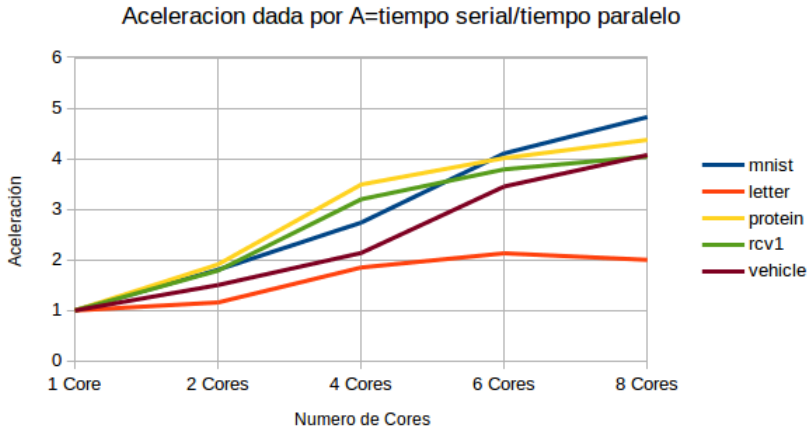

Figura 11: Aceleración obtenida en la LIBSVM usando los compiladores PGI haciendo uso de las API OpenACC y OpenMP

En la tabla 3 se encuentran las aceleraciones obtenidas teniendo en cuenta la cantidad de núcleos usados y el uso de la GPU. Cabe resaltar que los modelos arrojados por la librería son idénticos, por lo que las predicciones también lo serán. La figura 11 ilustra como se comporta la curva de aceleración para con diferentes núcleos CPU.

Aceleración dada por la CPU + GPU NVIDA

Dataset	1 Core CPU		2 Core CPU+GPU		4 Core CPU+GPU		6 Core CPU+GPU		8 Core CPU+GPU	
	tiempo(s)	aceleración	tiempo(s)	aceleración	tiempo(s)	aceleración	tiempo(s)	aceleración	tiempo(s)	aceleración
mnist	26610.874	1	14747.727	1.80	9725.323	2.74	6481.87	4.11	5517.287	4.82
letter	19.744	1	17.049	1.16	10.678	1.85	9.271	2.13	9.874	2.00
protein	351.233	1	184.001	1.91	100.65	3.49	87.514	4.01	80.32	4.37
rcv1	234.779	1	131.327	1.79	73.398	3.20	61.944	3.79	58.063	4.04
vehicle	2709.477	1	1804.864	1.50	1268.62	2.14	785.133	3.45	665.133	4.07

Tabla 3: Aceleracion obtenida usados en la LIBSVM con compiladores PGI

7 CONCLUSIONES

A partir de las secciones **6.4** y **6.5**, donde se presenta una modificación de la biblioteca LIBSVM que se aprovecha de la capacidad de procesamiento de la GPU y la CPU Multicore a través de compiladores PGI, generando una aceleración en el procesamiento, repartiendo el calculo del producto punto entre los diferentes núcleos de la CPU, así mismo en la GPU se realizan algunas operaciones relacionadas a los vectores que son manejados por la biblioteca.

La presentación de los diagramas de flujo permiten encontrar ciclos y subrutinas en el algoritmo, de manera practica y concreta, lo cual fue indispensable para identificar propiedades concurrentes a los procesos, y en este caso en particular fueron primordiales para reconocer que secciones de código podían ser aceleradas.

Encontramos que una buena representación en la entrada de los datos, de la maquina de soporte vectorial (sección **5.2**), tiene un mejor rendimiento en términos de velocidad, debido a que existe una mayor facilidad para extraer la información y/o consultar los datos del problema.

En la sección **6.2** se habla de los compiladores PGI, estos son de gran utilidad, ya que presentan una forma fácil de aprovechar las diferentes arquitecturas y recursos de hardware, facilitando así mismo a los investigadores y desarrolladores, acelerar sus algoritmos de manera rápida y eficiente.

La versión modificada de la LIBSVM produce resultados idénticos con la LIBSVM original, razón por la cual podemos interpretar que tanto el funcionamiento del algoritmo, como la precisión del modelo construido por este, son completamente validos.

Nuestra configuración experimental mostró que la LIBSVM acelerada por una CPU Multicore y la GPU proporciona una mejora de la velocidad de procesamiento, la cual aumenta abruptamente con el tamaño de los datos de entrada (caso serial), lo que permite el manejo eficiente de los grandes problemas. Sin embargo la aceleración proporcionada por la GPU se encuentra acotada por la cantidad de elementos en el set de entrenamiento, por lo que, para problemas pequeños se hace evidente que no se presenta una aceleración significativa durante el entrenamiento del modelo.

Este librería LIBSVM con compiladores PGI, puede ser usada para cualquier tipo de problemas que puedan ser resuelto por una SVM, como lo es el caso de la detección de anormalidades en la fermentación del vino.

Este trabajo pudo contribuir en el trabajo realizado por [3] [4], en donde se tiene en cuenta el análisis de las características de las líneas espectrales moleculares de los cubos de datos astronómicos que dan una idea de la composición de nuestro universo.

El presente trabajo sera presentado en el congreso GTC17 (GPU Technology Conference) con el animo de tener una mayor contribución a la ciencia.

7.1. TRABAJO FUTURO

Así como en [1] cabe la posible de establecer un método que implemente la API de OpenACC con los compiladores PGI de tal manera que la aceleración otorgada por la GPU tenga un mayor efecto en la resolución

del modelo creado por la libsvm.

También puede ponerse en consideración una estructura diferente para almacenar los datos de entrenamiento de tal manera que las operaciones para la obtención de la matriz kernel y el calculo del producto punto entre los vectores de entrada puedan ser efectuados por la GPU; esto implicaría un mayor uso de memoria justificada por la poder de computo de la GPU.

Se expone la idea de que con ayuda de los compiladores PGI, que permiten el uso de las diferentes formas de programación como lo son la memoria distribuida, pueda abordarse problemas con una mayor dimensión en los datos de entrenamiento del algoritmo como lo es en el caso del dataset llamado "splice-site"que puede ser consultado en [5], el cual consta de 2 clases, 50000000 de vectores de entrenamiento, 4627840 ejemplos para la testear el modelo y 11725480 características.

REFERENCIAS

[1] Athanasopoulos, Andreas ; Dimou, Anastasios ; Mezaris, Vasileios ; Kompatsiaris, Ioannis: GPU ACCELERATION FOR SUPPORT VECTOR MACHINES. En: 12th International Workshop on Image Analysis for Multimedia Interactive Services (2011). – Software disponible en `http://mklab.iti.gr/project/GPU-LIBSVM`

[2] Barney, Blaise ; Lawrence Livermore National Laboratory. OpenMP. Disponible en `computing.llnl.gov/tutorials/openMP/`

[3] Barrientos, Alejandro ; Ñanculef, Ricardo ; Solar, Mauricio ; Mardones, Diego ; Gonzalez, Natalia ; Flores, Daniel ; Ferreira, Andres: Machine Learning approaches for classification of astrochemical spectra. En: Astronomy and Computing ASCOM 2016 (En Revisión) (2016)

[4] Barrientos, Alejandro ; Ñanculef, Ricardo ; Solar, Mauricio ; Mardones, Diego ; Gonzalez, Natalia ; Flores, Daniel ; Ferreira, Andres: Machine Learning approaches for classification of astrochemical spectra. En: The National Radio Astronomy Observatory NRAO (Poster) (Abril 2016)

[5] Chang, Chih-Chung ; Lin, Chih-Jen: LIBSVM: A library for support vector machines. En: ACM Transactions on Intelligent Systems and Technology 2 (2011), p. 27:1–27:27. – Software disponible en `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[6] Chang, Edward Y. ; Zhu, Kaihua ; Wang, Hao ; Bai, Hongjie ; Li, Jian ; Qiu, Zhihuan ; Cui, Hang: PSVM: Parallelizing Support Vector Machines on Distributed Computers. (2007). – Software disponible en `https://code.google.com/archive/p/psvm/`

[7] Graf, Hans P. ; Cosatto, Eric ; Bottou, Leon ; Durdanovic, Igor ; Vapnik, Vladimir: Parallel Support Vector Machines: The Cascade SVM. En: Advances in Neural Information Processing Systems 17 (2005), p. 521–528

[8] Hernandez, Gonzalo ; Leon, Roberto ; Urtubia, Alejandra: Detection of abnormal processes of wine fermentation by support vector machine. (2014)

[9] Hernandez, Gonzalo ; Leon, Roberto ; Urtubia, Alejandra: Prediction of abnormal wine fermentations using computational intelligent techniques. En: Journal of Computer Science and Technology 1 (2015)

[10] Kandrot, Edward ; Sanders, Jason: CUDA by Example: An Introduction to General-Purpose GPU Programing. Pearson Education, Inc, 2010

[11] Khronos Group. The open standard for parallel programming of heterogeneous systems. Disponible en `www.khronos.org/opencl/`

[12] Message Passing Interface Forumn. MPI: A Message-Passing Interface Standard. Disponible en `www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`

[13] Meyer, Oliver ; Bischl, Bernd ; Weihs, Claus: Support Vector Machines on Large Data Sets: Simple Parallel Approaches. En: Data Analysis, Machine Learning and Knowledge Discovery (2014), p. 87–95

[14] Ng, Andrew Yan-Tak: CS229: Machine Learning (Course Handouts). En: Universidad de Stanford (2016). – Disponible en `http://cs229.stanford.edu/materials.html`

[15] NVIDIA Corporation ; PGI Compilers and Tools: PGI Compiler User's Guide for Intel 64 and AMD64C PUs. Pearson Education, Inc, 2016

[16] OpenACC Directives for Accelerators. About OpenACC. Disponible en www.openacc.org/About_OpenACC

[17] OpenMP Architecture Review Board. OpenMP Application Program Interface. Disponible en www.openmp.org/mp-documents/spec30.pdf

[18] 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. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python. En: Journal of Machine Learning Research 12 (2011), p. 2825–2830

[19] Pizarro, Omar ; Leon, Roberto: Predicción de fermentaciones anormales de vinos usando Inteligencia Artificial. Universidad Andres Bello, 2015

[20] Roman, César ; Hernández, Gonzalo ; Urtubia, Alejandra: Prediction of problematic wine fermentations using artificial neural networks. En: Bioprocess Biosyst Eng 34 (2011), p. 1057–1065

[21] Stone, John ; Gohara, David ; Shi, Guochun: OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. En: Computing in Science and Engineering 12 (2010), p. 66–73

[22] Urtubia, A. ; Hernandez, G. ; Roger, J.M.: Detection of abnormal fermentation in wine process by multivariate statistics and pattern recognition techniques. En: Journal of Biotechnology 159 (2012), p. 336–341

[23] work, Cyc O. Svm max sep hyperplane with margin. Disponible en https://commons.wikimedia.org/w/index.php?curid=3566688

[24] ZackWeinberg. Svm separating hyperplanes. Disponible en https://commons.wikimedia.org/w/index.php?curid=22877598