Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

"Implementación multicore del algoritmo PSC para la selección de instancias"

T E S I S

Para obtener el título de:
Ingeniero en **C**iencias de la **C**omputación

Presenta
Omar J. Castellanos Santa Cruz

Asesor de Tesis
Dr. Arturo Olvera López

Enero 2014
Puebla, Puebla.

Agradecimientos

A la Benemérita Universidad Autónoma de Puebla, por aceptarme como su aprendiz, a la Facultad de Ciencias de la Computación, mi casa por cinco años, a mis profesores de la facultad, por el conocimiento compartido, al Dr. Manuel Martín y al Dr. Iván Olmos, sinodales de esta tesis, por revisarla y aprobarla, al Dr. Arturo Olvera mi asesor, por sus opiniones, correcciones, comentarios y enfoque a lo largo de este trabajo de tesis, a mi madre, por su apoyo incondicional, a mi hija, por la espera cuando estuve lejos de casa y a Xion, Calipso y Andrómeda, mis computadoras, por las horas de trabajo sin descanso.

Gracias.

Dedicatoria

A mi hijo
Por una nueva sonrisa.

A mi hija
Te debo tanto tiempo...

A mi madre
Todo lo que soy hoy, es por ti, todo te lo de debo a ti, mamá…

A mi abuela
Por cuidarme de niño.

“La especialización es, según nosotros, una forma de propiedad privada del conocimiento. El que algo sabe, lo atesora y complicándolo hasta hacerlo parecer algo extraordinario e imposible, algo a lo que pueden acceder unos pocos, se niega a compartirlo, y su coartada es la especialización. Son como los brujos del conocimiento, como los antiguos sacerdotes que se especializaban en hablar con los dioses”.

De especialistas y especialidades.
Subcomandante Insurgente Marcos.

Índice General

Prólogo 1
1. Marco Teórico 3
1.1 Introducción 3
1.2 Minería de datos y Aprendizaje automático 4
1.3 Reconocimiento de Patrones 6
1.4 Preludio a la clasificación 7
1.5 Clasificación 8
1.6 Clasificadores 10
1.7 Pre-procesamiento de datos 13
1.8 Selección de instancias 16
2. Estado del Arte 18
2.1 Algoritmos de selección de instancias 18
3. Prototype Selection by Clustering (PSC) 23
3.1 Definiciones 23
3.2 PSC 24
3.3 Parallel PSC 28
4. Paralelismo 30
4.1 Computo paralelo 30
4.2 ¿Qué es el Paralelismo? 30
4.3 ¿Qué es Computación Paralela? 30
4.4 Programa Paralelo 32
4.5 Diseño de algoritmos paralelos 33
4.6 Arquitecturas Paralelas 34
4.7 Programación para memoria compartida 38
4.8 Programación con paso de Mensajes 38
4.9 Comunicación 39
5. Implementación 41
5.1 Justificación 41
5.2 Visión general 42
6. Pruebas y Experimentos 47
6.1 Pruebas con datos sintéticos 47

6.2 Pruebas de comparación 49
6.3 Pruebas de precisión 53
6.4 Pruebas de retención 54
6.5 Pruebas de precisión con PSC 60
6.6 Pruebas de variación de parámetros 64
7. Conclusiones 67
8. Referencias 69
Apéndice A 73
Formato TRN 73
Funciones de Distancia 74
Validación Cruzada 77
Tablas de tiempo 77
Apéndice B 80
OpenMP 80
Introducción 80
Origen de OpenMP 80
¿Qué es OpenMP? 81
Programación en OpenMP 83
Directivas 84
Paralelización de bucles 87
Sincronización 89

Prólogo

Ahora que hemos reunido tantos datos, ¿qué hacemos con ellos?

U.M. Fayyad.

En la actualidad, debido a la creciente cantidad de información, los tiempos de ejecución de los sistemas que estudian muchas ramas de la ciencias del mundo en que vivimos se han incrementado, esto ha traído como consecuencia el aumento de forma paralela tanto en la cantidad de información almacenada, como en la necesidad de desarrollar algoritmos que permitan el manejo y clasificación de estos datos en forma automática. En las últimas décadas el cómputo paralelo sólo se podía realizar mediante sistemas especializados creados para investigación, estas supercomputadoras tienen un costo muy alto, por encima de la decena de millones de dólares y son de uso específico.

Una alternativa a estas supercomputadoras, fueron los cluster de ordenadores**,** que suponían un costo más bajo, al interconectar grupos de computadoras a través de una LAN, la gran cantidad de cálculos matemáticos sobre conjuntos de datos con este enfoque paralelo tienen un lapso de tiempo aceptable, pero aún siguen teniendo altos costos de mantenimiento de hardware o software y sólo pueden ser adquiridas por universidades y centros de investigación.

En los últimos años, con el avance en la construcción de microprocesadores de bajo costo utilizados en las computadoras personales, el paralelismo empieza a crecer, las máquinas multi-core nos permiten ejecutar más de un proceso al mismo tiempo, haciendo que los programas puedan realizar sus cálculos en menos tiempo ya que existe más de un procesador ejecutando instrucciones de manera coordinada y simultánea.

Las aplicaciones prácticas del cómputo paralelo se encuentran en muchas disciplinas, tales como la predicción del clima, exploración petrolera, procesamiento de imágenes, fusión nuclear, modelado de escenarios, simulaciones, efectos especiales y áreas de la inteligencia artificial como minería de datos, mundos virtuales y reconocimiento de patrones.

El reconocimiento de patrones, que es el área de interés de esta tesis, se enfoca en la extracción de modelos a partir de un conjunto de información, mediante el cual, se aprenden reglas (se entrena) para asignar a los nuevos datos (información) que ingresan en el conjunto,

una categoría (clase o grupo). A este tipo de métodos o algoritmo que llevan a cabo el proceso de asignación, se denominan clasificadores y el proceso de asignación, recibe el nombre de clasificación.

Regularmente esta clasificación consume mucho tiempo de ejecución, principalmente cuando el conjunto de datos que se usa para la toma de decisiones es muy grande, desgraciadamente la información contenida en estos grandes conjuntos suele ser impura o redundante, por tal, estos conjuntos son sometidos a un pre-procesamiento de datos. El pre-procesamiento engloba todas aquellos métodos de análisis de datos que permiten mejorar la calidad de dicho conjunto, en este ámbito podemos mencionar la reducción de datos. En particular, una de sus técnicas es la selección de instancias, método cuyo fin es descartar los objetos innecesarios con base en un criterio de selección, esta reducción de datos agiliza el proceso de clasificación.

Existen diversos algoritmos con diferentes técnicas para la selección de instancias, en su mayoría son algoritmos secuenciales que no explotan al máximo los recursos de una máquina con tecnología multi-core. En el estado del arte *PSC* resulta ser una de las mejores opciones para resolver el problema de selección de instancias, principalmente por su rapidez en grandes conjuntos de datos, en las que algunos algoritmos de selección podrían ser no aplicables debido a la diferencia en tiempo de ejecución comparado con *PSC*. La disponibilidad multi-core en las máquinas puede ser aprovechada para implementar una versión paralela de este algoritmo, sumando el paralelismo a *PSC* se espera una reducción significativa en los tiempos de ejecución al aplicarlo a grandes bases de datos.

En este trabajo de tesis se implementa el algoritmo *PSC* multi-core tomando como base el algoritmo de *PSC* secuencial, se lleva a cabo un estudio aplicando PSC en su versión secuencial y paralela, a bases de datos; A los conjuntos reducidos obtenidos en estas pruebas se les aplica un clasificador para obtener su precisión y medir los tiempos de ejecución, todos los datos resultantes se analizan y comparan.

CAPÍTULO 1

Marco Teórico

En la actualidad, nada cuesta más que la información

I.A.

1.1 Introducción

Los grandes avances en tecnología de la información han provocado un aumento en el flujo de datos entre dispositivos electrónicos, en la actualidad podemos acceder desde un ordenador comercial o dispositivo móvil a una gran cantidad de información digitalizada y almacenada en una base de datos. La mayor parte de esta información representa situaciones en el mundo real, siendo útil no solo para explicar el pasado, sino para entender el presente y predecir sucesos futuros. Códigos de barra, tarjetas electrónicas, sensores remotos, transacciones bancarias, satélites espaciales o el descifrado del código genético humano son ejemplos de la necesidad de filtrar, analizar e interpretar volúmenes de datos que se miden en terabytes (TB) [Ruiz, 2005].

La necesidad de analizar información en este tipo de tareas requiere distinguir o identificar patrones existentes en el entorno; dicho análisis de información debe transformarse en conocimiento, pero hacerlo de forma manual es un proceso lento, caro y subjetivo. Gracias a la tecnología, hoy, esto puede realizarse mediante un sistema computacional.

Un beneficio de analizar información sobre este contexto, es utilizar su categorización para extraer modelos que nos ayuden a la toma de decisiones posteriores [Jain et al, 2000], pero el gran tamaño de la información hace más tedioso el análisis, aunado a esto, una vez implementada la categorización de la información y la toma de decisiones, cómo hacer que un

sistema computacional mejore con la experiencia y se adapte a los diferentes entornos en los que se desarrolla. Algunas áreas de las ciencias computacionales referentes a este último punto son: Reconocimiento de Patrones (Pattern Recognition), Minería de Datos (Data Mining) y Aprendizaje Automático (Machine Learning); las cuales, entre otras herramientas, nos brindan técnicas y algoritmos para la extracción de modelos en un conjunto de información.

1.2 Minería de datos y Aprendizaje automático

Desde de la creación de la **Mark I** en 1944 por Howard H. Aiken se puso en duda si algún día una computadora llegaría a igualar la inteligencia humana y sería capaz de aprender; A finales de la década de los 80, la creciente necesidad de automatizar los procesos de análisis de datos que debido a su enorme cantidad desbordaban la capacidad humana, abrió el camino para un nuevo enfoque de *análisis de datos,* este enfoque se valía de un conjunto de métodos matemáticos y técnicas de software para *análisis inteligente de datos* utilizados de forma repetida e iterativa, que se definió como ***Minería de Datos*** o ***Data Mining*** (DM). Su tarea fundamental es la de encontrar modelos inteligibles a partir de los datos, dichos modelos deben ser automáticos para un proceso efectivo, generando patrones que ayuden en la toma de decisiones beneficiosas para la organización [Ruiz, 2005].

Para obtener soluciones rentables al aplicar a una base de datos técnicas de *minería de datos*, es necesario complementar este proceso con un análisis posterior de los resultados obtenidos, resultados que pueden variar con el tiempo y ubicación geográfica. Por ejemplo: el comportamiento de un consumidor, no sabemos exactamente que sabor de helado tiende a comprar o el próximo libro que leerá, lo que sí sabemos es que no es completamente al azar. Cuando compra cerveza y helado en verano y vino y ropa abrigadora en invierno hay cierto patrón en los datos [Alpaydın, 2010], si llevamos un registro de estos hábitos por años, o sea, si almacenamos su información, podemos encontrar un patrón más claro de un consumidor, pero, ¿podemos mediante un modelos extraído utilizando técnicas de minería de datos saber qué comprar la próxima vez? , o aún más complejo, ¿podemos hacer que una máquina pueda aprender de estos datos y hacerlo de forma automática para predecir la próxima compra?, ¿Cómo podemos construir sistemas informáticos que automáticamente mejoren con la

experiencia, y cuáles son las leyes fundamentales que gobiernan los procesos de aprendizaje?"[Mitchell, 2006].

Para responder estas preguntas introducimos a nuestro contexto otra área de la Inteligencia Artificial, el ***Aprendizaje Automático*** (Machine Learning). El Aprendizaje Automático (ML) es una disciplina que estudia cómo construir sistemas computacionales que con base en análisis de datos en forma de ejemplos pueden mejorar automáticamente la toma de decisiones mediante la experiencia, una definición formal de aprendizaje es:

Aprendizaje: *un programa de computadora se dice que aprende de una experiencia E con respecto a una clase de tarea T y medida de desempeño D, si su desempeño en las tareas T, medidas con D, mejora con la experiencia E [Mitchell, 97].*

Aunque no se ha logrado que una computadora aprenda tan bien como un humano, muchos de los algoritmos son capaces de comportarse en forma muy efectiva en ciertas tareas.

Los principios seguidos en el aprendizaje automático y en la minería de datos son los mismos: la máquina genera un modelo a partir de ejemplos y los usa para resolver el problema. También se ha definido a la minería de datos como la aplicación de técnicas de aprendizaje automático sobre grandes bases de datos [Mitchell, 2006]. Uno de los modelos de aprendizaje más estudiado es el aprendizaje inductivo[1], que engloba todas aquellas técnicas que aplican deducciones inductivas sobre un conjunto de datos para adquirir conocimiento necesario a partir de ejemplos reales sobre la tarea que se desea modelar.

El resultado de ejecutar el algoritmo de **ML** puede ser expresado como una función que tiene una entrada y que genera una salida. La forma precisa de la función se determina durante la fase de entrenamiento (también conocido como fase de aprendizaje) sobre la base o conjunto de entrenamiento. Una vez que el modelo está entrenado puede entonces determinar la identidad de las nuevas entradas a la función. La capacidad de clasificar correctamente los nuevos ejemplos que difieren de los utilizados para la formación[2] se conoce como ***clasificación,*** la cual es un objetivo esencial en el ***Reconocimiento de Patrones*** [Bishop,2006].

[1] Aprendizaje en el cual las conclusiones no son a base de la transferencia de los conocimientos que un experto humano posee a un sistema informático(aprendizaje deductivo) , estas son creadas automáticamente a partir de muestras de aprendizaje

[2] En las aplicaciones prácticas, los datos de entrenamiento puede comprender sólo una pequeña fracción de todos los datos de entrada posibles.

1.3 Reconocimiento de Patrones

El reconocimiento automático, clasificación y agrupamiento de patrones son actividades importantes en una gran variedad de disciplinas científicas, su carácter interdisciplinario hace que este problema tenga una alta complejidad. Desde el surgimiento de esta área hasta la actualidad, otras ramas relacionadas a la Inteligencia Artificial[3] han participado y evolucionado a la par en el desarrollo exitoso de herramientas con el fin de solucionar problemas prácticos y teóricos dentro del Reconocimiento de Patrones[Vázquez , 2008].

El Reconocimiento de Patrones(RP), a pesar de ser un área del conocimiento relativamente joven, (apenas unos 45 años), al intentar definirla encontraremos diferentes formas dependiendo del área de aplicación, en Biología la encontraremos como taxonomía Numérica o Fenética, Q-análisis en Psicología, Reconocimiento de Patrones en Inteligencia Artificial, agrupamientos, análisis de conglomerados, etc. [Soto, 2005]. El reconocimiento de patrones se enfoca a la extracción de modelos (patrones) a partir de un conjunto de información, comúnmente se le llama conjunto de entrenamiento, los modelos que se extraen de un conjunto de entrenamiento suelen utilizarse para la categorización de nuevos objetos (ver figura 1.1). A éste proceso se le denomina **clasificación**, el cual es uno de los problemas de los que se encarga el RP. Otros problemas que estudia el RP son la selección de atributos y la selección de instancias.

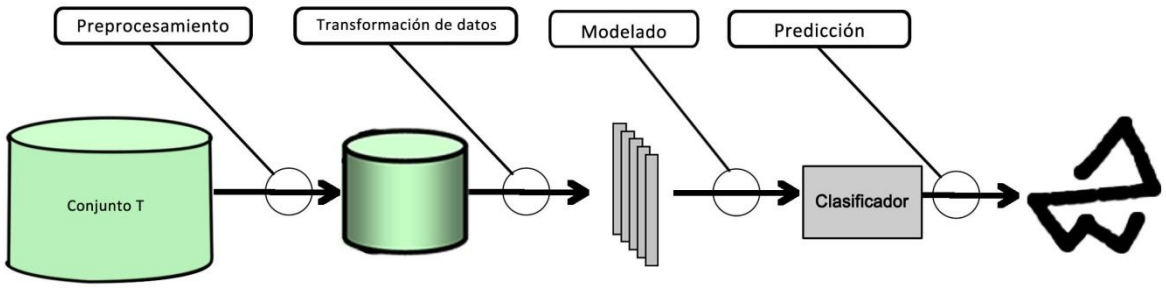


Fig. 1.1. Proceso general del reconocimiento de patrones

[3] Principalmente Minería de Datos (Data Mining), Aprendizaje Automático (Machine Learning).

1.4 Preludio a la clasificación

A continuación se establecen algunas definiciones que describen los conceptos que se manejan a lo largo de esta tesis.

Conjunto de datos: También llamado conjunto de entrenamiento, matriz de aprendizaje o matriz de ejemplos (ver **fig. 1.2**), se puede definir como un conjunto finito y bien definido de ejemplos, cada uno de estos ejemplos recibe el nombre de objeto o instancia, ésta base de datos se caracteriza por el número de ejemplos ***n*** que contiene y por el número ***m*** de atributos de cada ejemplo, generalmente representada como ***T***. ***T*** también se puede representar como una matriz, $\boldsymbol{T_{nxm}}$.

Objeto: Un objeto o instancia $\boldsymbol{O=<a_1,a_2,\ldots,a_m>}$ es un *n-uplo* que representa un elemento físico o abstracto donde cada $\boldsymbol{a_i}$ se denomina atributo descriptivo del objeto, por lo general el atributo $\boldsymbol{a_m}$ representa la etiqueta de clase del objeto, aunque eventualmente no es así.

Atributo: Un atributo $\boldsymbol{a_i}$ también denominado característica, es una descripción de alguna medida que es parte de un objeto, puede ser continuo también llamado numérico o discreto (no numérico). Si es continuo existe un rango $[a,b] \subseteq \Re$ de valores posibles, si es discreto existe un conjunto finito de valores posibles.

Clase: En el marco de aprendizaje supervisado un objeto ***O*** cuenta con un atributo especial, por lo general $\boldsymbol{a_m}$, que indica la pertenencia a un determinado grupo de casos. También es llamado etiqueta o clase. La clase es el atributo sobre el cual se realiza la predicción, por lo que también se le suele llamar atributo de decisión, para diferenciar del resto de atributos.

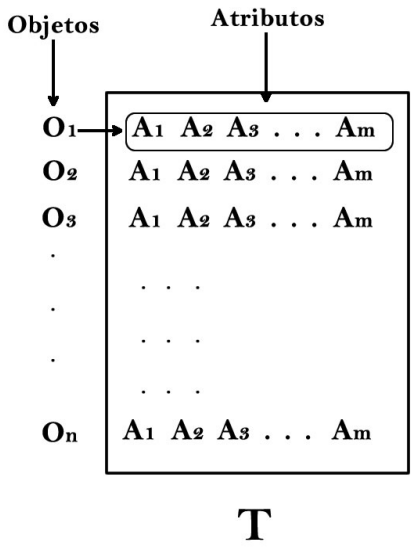


Fig. 1.2. Esquema General de una matriz de aprendizaje

El conjunto de entrenamiento puede contar solo con ejemplos numéricos o categóricos, también cabe la posibilidad de que los objetos contenga atributos de los dos tipos, a este tipo de conjunto se le denomina conjunto de datos mixto, cuando hablamos de un conjunto mixto, ***T*** puede incluso contener ausencia de información.

1.5 Clasificación

Aprender cómo clasificar, es de máximo interés para investigadores, dado que la habilidad de realizar una clasificación y de aprender a clasificar, otorga el poder de tomar decisiones. Se puede definir a la clasificación como:

Definición *Sea* $\mathcal{E}$ *un conjunto de datos, el objetivo de la clasificación es aprender una función* $L: X \rightarrow Y$, *denominada* ***clasificador****, que represente la correspondencia existente en los ejemplos entre los vectores de entrada y el valor de salida correspondiente, es decir, para cada valor de* $\bar{x}$ *tenemos un único valor de* Y [Ruiz, 2005].

Dicho de otra forma, dado un conjunto de entrenamiento ***T***, se construye (entrena) un método para predecir, de acuerdo a lo aprendido con los ejemplos de ***T***, la clase de nuevos objetos. Esto es, reconocer alguna situación cuando ésta se presente y tomar una decisión con la estrategia aprendida. En la **figura 1.3**, se ilustra el proceso de clasificación. En la parte izquierda, ***T*** representa el conjunto de entrenamiento, el cual contiene ***n*** objetos (O_i), cada uno representado por ***m*** atributos (A_i). Dependiendo del tipo de clasificación (supervisada, no supervisada), ***T*** puede o no contener la clase de cada objeto. A partir de la información en ***T***, se obtiene el modelo de clasificación, mediante un entrenamiento, una vez obtenido este modelo ya es posible categorizar nuevos objetos, como se puede notar en la parte inferior, los objetos nuevos a clasificar (On_i) son proporcionados al clasificador, el cual asigna una clase (C_{Oni}) a cada objeto.

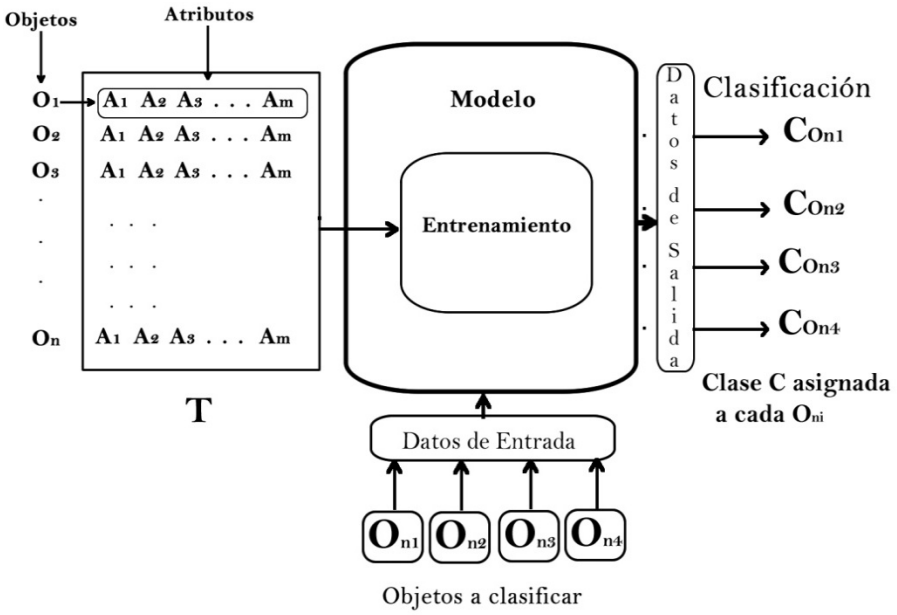


Fig. 1.3. Esquema general del proceso de clasificación

El proceso de clasificación, ampliamente conocido en áreas de la computación, surge debido a la existencia de problemas con la necesidad de respuestas automáticas de acuerdo a la información de casos reales. A estos métodos o algoritmos que llevan a cabo el proceso de clasificación, se les denomina ***clasificadores***. Estas relaciones o reglas obtenidas mediante aprendizaje que pueden asociar la clase a nuevos ejemplos son llamadas también algoritmos de aprendizaje los cuales difieren según el enfoque en el que están basados.

En general, las tareas de un proceso de aprendizaje, al igual que las de ***Data Mining*** y ***Machine Learning***, pueden ser clasificadas en dos categorías: descriptivas y predictivas[4]. Las primeras describen el conjunto de datos de una manera resumida y concisa, presentando propiedades generales e interesantes de los datos. Por otro lado las tareas predictivas construyen uno o varios modelos que realizan inferencia sobre el conjunto de entrenamiento para intentar predecir el comportamiento de nuevos datos. En reconocimiento de patrones nos referiremos a estas dos categorías como clasificación de tipo supervisado y no supervisado. La clasificación supervisada consiste en categorizar nuevos objetos basados en conjuntos de entrenamiento tal que se conoce la clase de cada objeto. Por otra parte, la clasificación no supervisada se basa en conjuntos de entrenamiento en los que se desconoce la clase de los objetos, esta clasificación se lleva a cabo mediante técnicas de agrupamiento (clustering). La manera de clasificar en el enfoque no supervisado es agrupar objetos de acuerdo a la similaridad, con cada grupo más o menos homogéneo y distinto de los demás: El criterio de homogeneidad más simple está basado en distancias, esperando que la distancia entre un mismo grupo sea significativamente menor que la distancia entre objetos de grupos diferentes, el concepto de grupo está relacionado con el de clase, idealmente cada clase debería tener asociado un grupo [Cortijon, 2001].

1.6 Clasificadores

En la literatura existen diversos tipos de clasificadores y se dividen según el tipo de aprendizaje que utilicen para la predicción.

- Aprendizaje basado en similaridad
- Aprendizaje de reglas
- Aprendizaje basado en grafos
- Aprendizaje basado en probabilidad
- Aprendizaje basado en instancias
- Aprendizaje por refuerzo
- Clustering

[4] Se suele utilizar el término aprendizaje supervisado para los métodos predictivos y aprendizaje no supervisado para el resto.

A continuación se describen algunos algoritmos de clasificación:

$\boldsymbol{k-NN}$ (k-Nearest Neighbours) [Cover y Hart , 1967] es un método de clasificación supervisado del tipo basado en instancias, hace la analogía con los individuos de una población donde los miembros de ésta comparten ciertas características con los individuos que los rodean, fundados en esta lógica podemos deducir propiedades de un individuo conociendo a sus vecinos más cercanos. La idea es la siguiente, dado un nuevo objeto $\boldsymbol{O_N}$ a clasificar, ***k-NN*** calcula la distancia entre $\boldsymbol{O_N}$ y todos los elementos del conjunto de entrenamiento. Finalmente, asigna a $\boldsymbol{O_N}$ aquella clase a la que pertenecen la mayoría de los ***k*** objetos más cercanos. La función de distancia es subjetiva, ya que si aplicamos distintas funciones de distancia[5] sobre un mismo conjunto de entrenamiento puede producir resultados diferentes, debido a esto, se debe de seleccionar una función de distancia adecuada de acuerdo al problema.

Otra cuestión importante y determinante es el valor que tomará ***k***, el número de vecinos que se tomará en cuenta es significativo para una buena predicción, con respecto a esto en el estado del arte se han estudiado métodos para una buena estimación de ***k***[6].

En el ejemplo de la **figura 1.4a**, dado un conjunto de entrenamiento con dos clases (cuadrados: clase 1 y triángulos: clase 2) el nuevo objeto, (círculo: sin clase) , si se toma en cuenta la distancia de éste con solo un vecino (k=1, ver **fig. 1.4a**), el vecino más cercano, a este objeto se le asignará la clase 1 (ver **fig. 1.4b**), en cambio si tomamos a tres vecinos (**fig. 1.5a**) para la decisión, el nuevo objeto será de clase 2 (**fig. 1.5b**) .

k-means [Stuart ,1957]***:*** es un método no supervisado de agrupamiento, que se basa en dividir el conjunto de entrenamiento en ***k*** grupos, de tal manera que la similaridad de objetos en el mismo grupo sea alta y la similaridad con objetos de grupos ajenos sea baja. Inicialmente, se eligen aleatoriamente ***k*** objetos llamados centroides, teniendo primero los ***k*** grupos un elemento, el cual, también es la media del grupo (siendo ***k*** el número de grupos que deseamos crear) posteriormente se asigna a cada objeto $\boldsymbol{O_j}$ del conjunto de entrenamiento el centroide más cercano $\boldsymbol{C_j}$; después, se recalculan los ***k*** centroides, siendo el nuevo centroide $\boldsymbol{C_j}$ el objeto promedio en el grupo, algunos objetos pueden cambiar de grupo y como consecuencia los

[5] Para más información de funciones de distancia revisar el apéndice A
[6] Para más información revisar [Dietterich, 95].

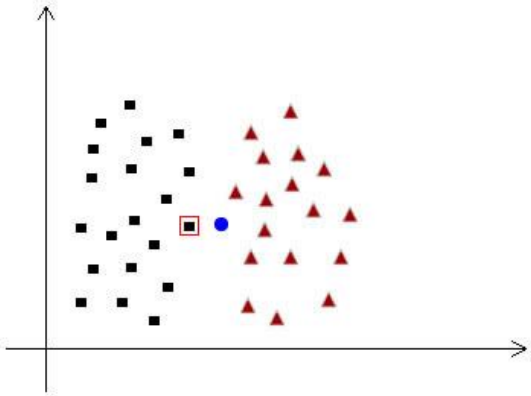


Fig. 1.4a. Ejemplo de k-NN con k=1

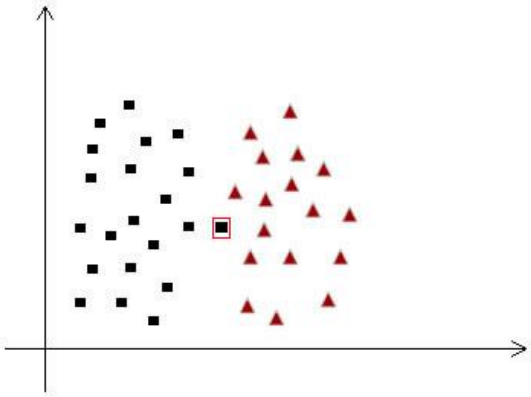


Fig 1.4b. Asignación de clase

centros de estos, si esto ocurre la operación se repite hasta que no hay cambios importantes en los centroides, esto es, que ningún objeto del conjunto de entrenamiento cambie de clúster, cuando esto sucede se considera que se ha encontrado una buena partición del conjunto ***T*** y termina el proceso. En futuras clasificaciones, el nuevo objeto O_N se asigna al grupo cuyo centroide es más cercano.

Los resultados al igual que en ***k-NN*** dependerán del valor del parámetro ***k***, el mayor problema está relacionado con las inicialización de los centroides, si estos pueden seleccionarse de manera que se dispersen uniformemente sobre el espacio de representación, es más fácil que los resultados finales mejoren, además que acelera la convergencia, que es el criterio de paro en este algoritmo.

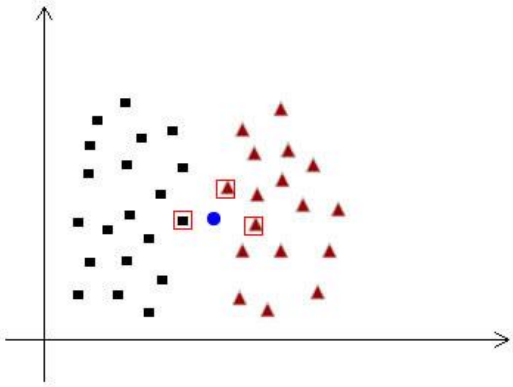


Fig. 1.5a. Ejemplo de k-NN con k=3

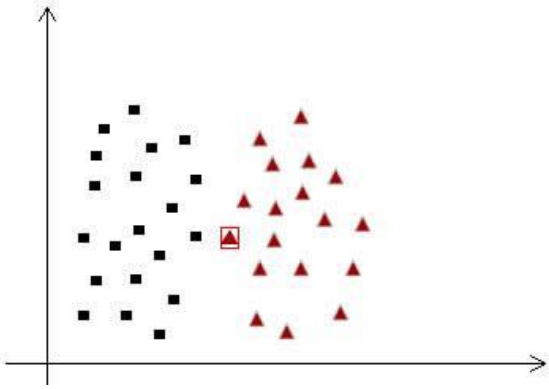


Fig. 1.5b. Asignación de clase

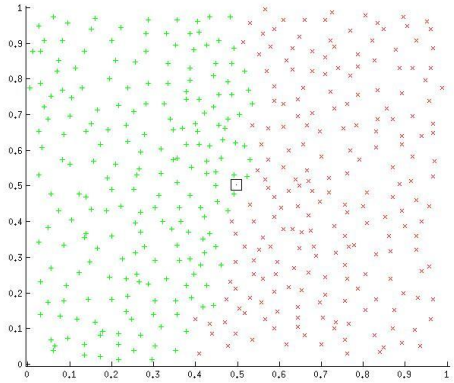


Fig. 1.6a. Cluster formado por k-means con k=1

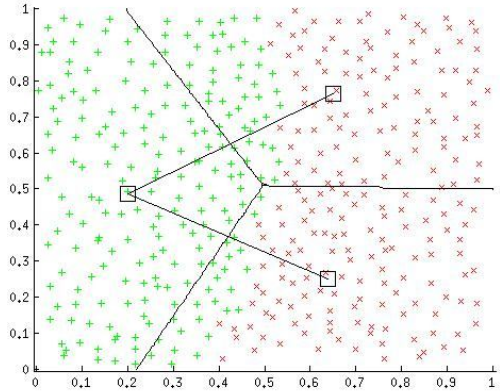


Fig 1.6b. Clusters formados por k-means con k=3

En la **figuras 1.6** se muestra el resultado gráfico de aplicar a un conjunto de entrenamiento con dos clases el algoritmo de ***k-means***, **figura 1.6a** con k=1 y **figura 1.6b** para k=3 los recuadros en el centro de los clusters son los centroides que representan la media del conjunto.

1.7 Pre-procesamiento de datos

La Selección de Atributos y la **Selección de Instancias**, son también problemas en los que se enfoca el ***RP***, orientados a la reducción de datos ambos problemas se sitúan en la fase de ***pre-procesamiento*** que se enfocan en la reducción de dimensionalidad, su principal utilidad es la reducción de tiempo en los procesos de aprendizaje y reducción del el tiempo de respuesta. En la **figura 1.7** podemos observar la arquitectura de un sistema de reconocimiento de patrones, la fase de pre-procesamiento y la de extracción de características en ocasiones suele referirse al término ***preparación de datos*** o ***pre-procesamiento***, esto engloba ambos procesos.

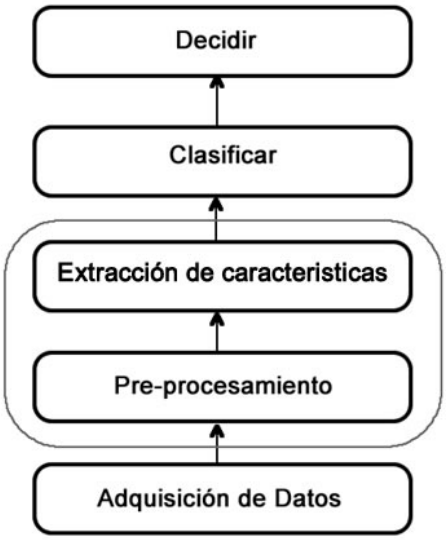


Fig.1.7 Arquitectura de un sistema de R.P.

En la mayoría de las ocasiones los datos provienen de diversas fuentes, y pueden contener valores impuros (incompletos o inconsistentes), rangos de valores muy dispersos u objetos superfluos (ruido, redundancia) que no son útiles para el proceso de clasificación, por ello es necesario que los conjuntos de datos sean sometidos a una fase de pre-procesamiento para preparar los datos para el entrenamiento de algún clasificador. Con estas técnicas se puede reducir el conjunto de datos (selección de características y de instancias), mejorar la eficiencia del proceso de minería de datos posterior. También existe la posibilidad de recuperar la información incompleta, eliminar outliers[7], resolver conflictos, etc. Generando un conjunto de datos de calidad, que conducirá a mejores modelos. La lista de tareas que se incluyen en esta fase, se puede resumir en: recopilación de datos, limpieza, transformación y reducción, no teniéndose que aplicar siempre en un mismo orden (ver ***figura 1.8***), este documento solo detallará una de las técnicas de la fase de reducción que es la selección de instancias por ser el tema de interés de este trabajo.

[7] Es estadística se entiende como observación atípica a aquella que es distinta numéricamente al resto de los datos, se define como: Una observación periférica, o errática, que parece desviarse considerablemente de otros miembros de la muestra en la que pertenece [Grubbs, 1969].

La recopilación se refiere a la extracción de datos de un problema, aunque puede presentar problemas de interpretación, traducción, trasformación, codificación e integración, se resume a la recolección de información a un archivo o base de datos.

La limpieza resuelve el problema con valores ausentes, outliers y ruido, aunque este último proceso también lo abarca la reducción de datos.

En lo que respecta a la transformación de datos, existen técnicas con las que se modifica en iguales proporciones los atributos de todos los objetos en un conjunto de entrenamiento. Ejemplos de este tipo de técnicas son: normalización y discretización. Mediante la normalización se elimina la dispersión excesiva de los datos y con la discretización se delimitan valores por medio de umbrales a ciertos rangos determinados.

Con respecto a la reducción de datos, las técnicas existentes para tal fin son la selección de atributos y selección de instancias, mediante la cual, con base en un criterio de selección, se descartan atributos irrelevantes y objetos innecesarios en un conjunto de entrenamiento respectivamente.

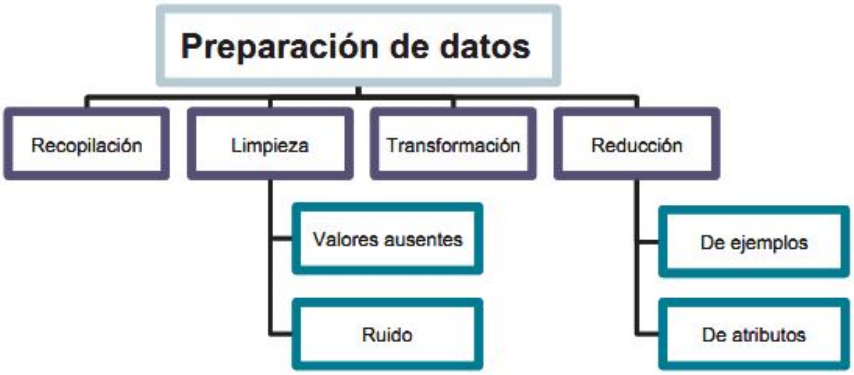


Fig. 1.8. Fase de preparación de datos o pre-procesamiento[Ruiz,2005]

1.8 Selección de instancias

En un problema de clasificación, dada una instancia, el objetivo es determinar la clase a la que pertenece, en general, el inconveniente radica en almacenar todos los ejemplos del conjunto de entrenamiento (generalmente representado por T) para poder predecir la clase de las nuevas instancias, ya que disminuye la rapidez del proceso de clasificación, además, puede afectar también la precisión [García et al,2004], un modo de afrontar este inconveniente es llevar a cabo una reducción del conjunto de entrenamiento mediante selección de instancias.

Mediante la selección de instancias se pretende encontrar un subconjunto, que mantenga o mejore su comportamiento a la hora de clasificar nuevas instancias.

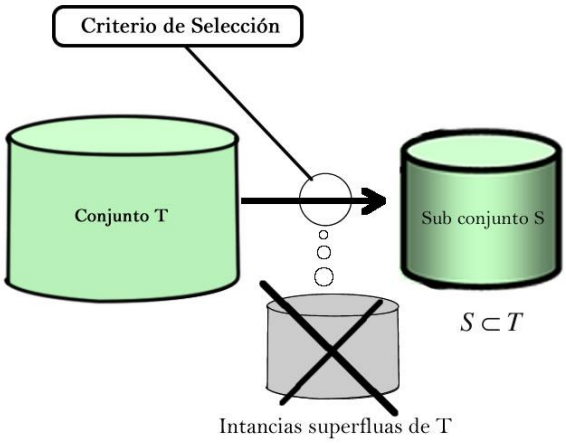


Fig. 1.9. Proceso de selección de instancias

En la **figura 1.9** se muestra el proceso de selección de instancias, donde S denota el conjunto reducido; S puede iniciar como $S = \emptyset$ o con $S = T$, definidos como método incremental y método decremental respectivamente, la diferencia entre estos métodos consiste en que primero agrega instancias de T a S, mientras que el otro enfoque las descarta.

Cuando se procesan conjuntos de datos relativamente extensos es conveniente, aplicar técnicas de reducción a la base de datos, para lograr una reducción del número de ejemplos, con esta reducción se pretende retener en el conjunto de entrenamiento solo cierta cantidad de objetos representativos del conjunto original, aunque también se aplica para eliminar objetos ruidosos, irrelevantes o redundantes.

El objetivo de la selección de instancias es, dado un conjunto de entrenamiento T obtener un subconjunto $S \subset T$ tal que S no contenga instancias superfluas (innecesarias) y que $Acc(S) \cong Acc(T)$, donde $Acc(X)$ es la precisión de clasificación obtenida por el clasificador usando el conjunto X como entrenamiento. Ejemplos de métodos para la selección de instancias son: *DROP*, *GCNN*, *IB*, *SGA* y *PSC*, los cuales se describen en el siguiente capítulo.

CAPÍTULO 2

Estado del Arte

2.1 Algoritmos de selección de instancias

De acuerdo con la estrategia utilizada para la selección de instancias, podemos dividir los algoritmos en dos grupos [Kuncheva & Bezdek, 1998; Bezdek & Kuncheva, 2001]:

- **Reemplazo**: Las instancias que constituyen el subconjunto S seleccionado no son parte del conjunto de entrenamiento, es decir, se crean nuevas instancias que representan el conjunto de entrenamiento.
- **Selección**: El conjunto de instancias seleccionadas S, es un subconjunto de T, las instancias eliminadas son aquellas que no contribuyen significativamente en la precisión del clasificador.

De acuerdo al criterio de selección que utiliza el algoritmo éste puede ser de dos tipos: [Olvera-López et al, 2010b]:

- **Wrapper** (envolvente): El criterio de selección está basado en la precisión dada por un clasificador.
- **Filter** (filtro): El criterio de selección está basado en una función distinta a la precisión de un clasificador.

La gran mayoría de métodos que se han propuesto para resolver el problema de la selección de instancias siguen el enfoque de selección. Algunos de los más relevantes se describen a continuación.

Uno de los primeros métodos de selección de instancias es *CNN* [Hart ,1968], inicia con $\boldsymbol{S} = \emptyset$, su primero paso consiste en incluir una instancia de cada clase a S aleatoriamente. Entonces cada instancia de T es clasificada usando solo las instancias de S, si una instancia es mal clasificada, esta es agregada a S, para asegurar que sean clasificadas correctamente, este

proceso se repite hasta que todas las instancias son correctamente clasificadas. Este método asegura que todas las instancias de T sean clasificadas correctamente, esto es que S sea consistente.

Otra variante de *CNN* es *GCNN* el cual es similar a *CNN* pero este incluye en S instancias de acuerdo a una regla de absorción, la cual es calculada en términos de los vecinos más cercanos y de los enemigos más cercanos (la instancia más cercana con diferente clase) de p (instancia) en S. El proceso de selección termina cuando todos las instancias en T han sido fuerte mente absorbidas, esto es cuando el criterio de absorción satisface el valor de un umbral dado por el usuario.

El método *ENN* [Wilson, 1972] descarta de T aquellas instancias que no pertenecen a la clase de sus *k*-vecinos más cercanos (*k-NN*), este método es usado como filtro de ruido porque elimina instancias superfluas, esto es, instancias con diferente clase en una vecindad. Una variante de este método es *RENN* donde *ENN* es aplicado repetidamente hasta que todos las instancias de S tiene la misma clase que la mayoría de sus vecinos más cercanos.

Otro método basado en *ENN* es Multiedit [Devijver P.A., 1980], este método de selección de instancias, crea m particiones en T de forma aleatoria $P_1 \dots P_m$. Después a cada partición *P* se le aplica *ENN* (usando 1-NN) para encontrar los vecinos de P_i en $P_{(i+1) mod\ m}$, este proceso es repetido hasta que no haya cambios en f iteraciones sucesivas.

DROP1-DROP5 (Decremental Reduction Optimization Procedure) [Wilson y Martinez, 2000] son otros métodos relacionados con *k-NN*, estos métodos son basados en el concepto de *socio,* los socios de una instancia S son todas aquellas instancias tales que p sea uno de sus k vecinos más cercanos. Inicialmente $S = T$ al ser un método decremental, *DROP1* elimina la instancia $p \in T$ si sus socios de $p \in S$ son correctamente clasificados sin p, esto es que la ausencia de esta instancias no afecta el porcentaje de clasificación correcto, *DROP1* descarta instancias ruidosas, ya que los socios de una instancia ruidosa puede ser correctamente clasificados sin dicha instancia, un inconveniente de este método es que cuando el vecino de un objeto ruidoso es descartado primero, dicho objeto no será descartado; Para resolver este problema *DROP2*, similar a *DROP1*, busca los socios de la instancia p en T, esto es, p solo es eliminado si sus asociados en T son correctamente clasificados sin dicha instancia. Por su parte *DROP3* y *DROP4* primero descartan las instancias ruidosas usando un filtro similar a *ENN* y

luego aplican *DROP2*. *DROP5* es basado en *DROP2* pero en este caso inicia descartando los enemigos más cercanos con el fin de suavizar los límites de decisión.

ICF (*Iterative Case Filtering algorithm*) [Brighton and Mellish, 2002] es un método, que al igual que DROP está basado en la idea de *socios*, utilizando dos conceptos más, *Reacheable(p)* y *Coverage(p)*, que son el conjunto de vecinos más cercano(alcance) y socios (cobertura) respectivamente. ICF descarta p si $|Reachables\ (p)| > |Coverage\ (p)|$, dicho de otra forma, se eliminan aquellos objetos cuyo número de vecinos más cercanos es mayor que el de socios, ya que en este caso el alcance es mayor a la cobertura. ICF utiliza *ENN*.

C-Pruner [Ke-Ping et al., 2003] es otro método basado en el concepto de *Reacheable(p)* y *Coverage(p)*, en este caso la regla para obtener los *Reacheable(p) solo considera los* socios de la misma clase, esto con el fin de descartar solo instancias de una clase en particular. Antes de remover algún elemento, este método verifica si alguna instancia es ruidosa, superflua o crítica, entendiendo como instancias crítica a aquel elemento del conjunto que si es eliminado, afectara la clasificación de otras instancias; este método elimina instancias ruidosas y superfluas, pero nunca críticas. Una instancias p es considera como ruidosa si $Coverage\ (p)| < |Reachables\ (p)|$; p es superflua si es clasificada correctamente por $Reachables\ (p)$.

Los algoritmos evolutivos(AE) también han sido usados para la selección de instancias[Kuncheva 1995, 1997; Kuncheva and Bezdek 1998; Bezdek and Kuncheva 2001; Cano et al. 2003], Los *AE* son basados en la evolución natural, la idea principal es la siguiente: dada una población inicial de cromosomas(conjunto de entrenamiento) y una función de aptitud(un clasificador, en el contexto de selección de instancias),la población es evaluada individualmente, seleccionado los mejores cromosomas, (aquellos que maximizan la función de aptitud) para luego combinar y mutar estos cromosomas, este proceso es iterativo y se repite las veces que el usuario especifique, al final el mejor cromosoma de la última generación es seleccionado. Un cromosoma C es representado por un vector binario $C = [1, 0, 0, 1, \dots, 1]$, donde $|C| = |T|$, y representa un subconjunto de$T$, cada C_j representa el *j*-ésimo elemento de T, si C_j=1 esto quiere decir que el ***j***-ésimo elemento de T esta siendo considerado en C, si esto no es así C_j=0. En operación de cruce (combinar) los bits de dos vectores (cromosomas) son intercambiados; La operación de mutación invierte el valor del bit en cromosoma.

Los *AE* combinados con algoritmos genéticos (*AG*) y búsqueda local con un ciclo evolutivo también han sido usados para la selección de instancias [García et al. ,2008]. La búsqueda local se lleva a cabo entre los cromosomas obtenidos durante el proceso evolutivo, esta búsqueda cambia los 1 y 0 en los cromosomas con el fin de mejorar la precisión y reducir el tamaño del conjunto de datos, los mejores cromosomas, (que mejoran la precisión) son seleccionados para los siguientes paso evolutivos (cruce y mutación).El mismo autor [García et al., 2008] propuso un método bioinspirado llamado CSA(Clonal Selection Algorithm) como algoritmo de selección de instancias, este método es un sistema inmune artificial. Cuando un patógeno ataca a un sistema humano, un conjunto de células (antígenos) crean una respuesta de inmunidad para defender el sistema de la invasión, la estrategia de defensa consiste en que las células muten para adaptarse al ataque, con esta mutación solo algunas serán capaces de responder el ataque del patógeno. Con esta misma idea *CSA* selecciona un conjunto de antígenos (instancias) de un cuerpo (conjunto de entrenamiento), *CSA* inicia seleccionado aleatoriamente un antígeno por clase, después un patógeno (conjunto de instancias) ataca el sistema inmune y repetidamente un conjunto de clones es mutado con el fin de selecciona los mejores contra el ataque de los patógenos.

Una solución propuesta para la selección de instancias [Cerverón and Ferri, 2001; Zhang and Sun 2002] es la búsqueda Tabú (BT)[Glover, 1986]. BT es aplicada sobre un conjunto inicial de soluciones $S_i \subset T$, durante el proceso de búsqueda algunas de estas soluciones son marcadas como tabú (soluciones que no deben de cambiar). Las soluciones que no son consideradas como tabú en S_i son evaluadas por un clasificador con el fin de seleccionar la mejor. Para seleccionar las soluciones de S_i se evalúan en los conjuntos vecinos (conjuntos que difieren en un elemento de S_i), esta evaluación se repite según las veces que la tendencia tabú (parámetro) indique, el proceso finaliza cuando S_i es remplazado por un subconjunto que produzca un mejor índice de clasificación.

Los métodos *filter* no utilizan un clasificador para descartar las instancias innecesarias en un conjunto de entrenamiento, esto los hace menos tardados que los *wrapper*, a continuación se explican algunos algoritmos que siguen este enfoque. Para algunos de estos métodos es necesario aclarar dos conceptos, instancia interior e instancia borde. Una instancia $p_j \in T$ es una instancia borde de la clase C_i si $p_j \in C_i$ y $p_j$es el vecino más cercano de otra instancia que

pertenece a una clase $C_k \neq C_i$, si la instancia $p_j \neq C_i$no es una *instancia borde*, entonces p_j es una instancia interior para C_i. Las instancias borde son importantes ya que nos proporcionan información sobre las fronteras de cada clase [Wilson and Martínez, 2000; Brighton and Mellish, 2002].

Algunos autores [Bezdek at al, 2001; Bezdek 2001; Spillmann B., 2006] han propuesto usar clusters para la selección de instancias, la idea consiste en dividir a T en n clusters, así, la selección de instancias se debe centrar en cada uno de estos clusters. *GCM* [Mollineda et al ,2002] (Generalized-Modified Chang Algorithm) es uno de los métodos basado en este concepto, el cual fusiona los clusters vecinos de la misma clase y selecciona los centros del clusters obtenidos en la fusión. *NSB* (Nearest Sub-class Clasisfier) [Venmann at al 2002] es otro algoritmo propuesto, éste permite seleccionar un número diferente de instancias (centros) por clase, utilizando pare ello, el algoritmo de clusters de varianza máxima.

CLU [Lumini A, Nanni L., 2006] es otro método de selección de instancia, el cual selecciona solo los centros de los clusters propuestos. *PSC* (Prototype Selection by Clustering) [Olvera-López et al, 2010a] es otro método basado en clustering, la idea de *PSC* es dividir el conjunto de entrenamiento en clústers o grupos que compartan ciertos criterios de similaridad, una vez dividido los datos en subconjuntos, se determinan si cada uno de estos es homogéneo o no homogéneo (esto es que si todas las instancias del grupo son de la misma clase o no), en los grupos homogéneos se obtiene la media y se descartan todos los objetos del grupo dejando solo uno de ellos, que es el que representa la media de grupo, para los clúster no homogéneos se buscan las instancias borde, el criterio de selección considera la instancia p como una instancia de borde si p es el más cercano a otra instancias en el clúster pero que pertenece a una clase diferente, las instancias interiores (que no están en el borde) puede ser descartadas de T, con el fin de preservar las instancias que están en las fronteras de los diferentes clúster. En las pruebas realizadas por los autores, para comparar los distintos métodos de selección de instancias, los resultados de *PSC* son los más rápidos con precisión aceptable, este método se detallarán en el capítulo de implementación.

CAPÍTULO 3

Prototype Selection by Clustering (PSC)

Yo no quiero lento aprendizaje, necesito rapidez.

Patmo.

En esta sección se describe y analiza el algoritmo PSC para una mejor compresión del criterio de selección que usa al momento de descartar instancias. Este algoritmo es el que se implementa en capítulos posteriores de esta tesis.

3.1 Definiciones

PSC es un algoritmo de selección de instancias que sigue la estrategia de ***selección***[8] para reducir el conjunto de entrenamiento, antes de pasar a la descripción del algoritmo definiremos cuatro conceptos que debemos manejar para su correcta compresión.

$Definición$ 1

$-\ Un\ prototipo\ \ p_j \in T\ \ es\ un\ prototipo\ borde\ para\ la\ clase\ \ C_i\ \ si\ \ p_j \in C_i\ \ y\ \ p_j\ es\ el\ vecino\ mas\ cercano\ de\ un\ prototipo\ de\ clase\ C_k \neq C_i.$

$Definición$ 2

$-\ Un\ pototipo\ \ p_j \in T\ \ es\ un\ prototipo\ interior\ para\ la\ clase\ \ C_i\ \ si\ \ p_j \in C_i\ \ y\ \ p_j\ no\ es\ un\ prototipo\ borde.$

[8] Esta estrategia se definió en el capítulo 2

$Definición$ 3

$-$U$n\ cluster\ homogéneo\ es\ aquel\ que\ solo\ contiene\ prototipos\ de\ la\ misma\ clase$

$Definición$ 4

$-\ \ Un\ cluster\ \boldsymbol{No}\ homogéneo\ es\ aquel\ que\ contiene\ prototipos\ de\ diferentes\ clases.$

En un conjunto de entrenamiento, las instancias borde, son relevantes para encontrar el discriminante de las distintas clases, estas instancias que se encuentran en la frontera de las clases son siempre críticas para el proceso de clasificación, los casos localizados en el interior de estas regiones pueden ser considerados como superfluos ya que su eliminación no conduce a ninguna perdida en la capacidad de encontrar el discriminante de clases [Brighton, Mellish, 2002].

3.2 PSC

Dado un conjunto de datos el problema es cómo seleccionar qué instancias son borde y cuáles son interiores, *PSC*, propone resolver este problema dividiendo el conjunto de datos en regiones con el fin de encontrar las instancias que se encuentran en las fronteras de regiones más pequeñas, para no hacer la búsqueda en todo el conjunto completo, lo cual resulta más costoso. Para dividir el conjunto en regiones, independientemente de la distribución de clases, se utiliza un algoritmo de *clustering*, el cual es ***C-means***, una vez que el conjunto ha sido dividido en ***k*** regiones el siguiente paso es analizar los clusters generados. Para los *clusters no homogéneos* la idea es detectar sus instancias de borde, de las diferentes clases, este tipo de instancias representan el límite colindante donde un grupo de determinada clase termina e inicia otro conjunto de clase diferente. Para las regiones homogéneas ***PSC*** dicta que sólo se debe de conservar una instancia, la cual debe de ser la que mejor represente a la región.

```
1   PSC(matriz T, entero k)
2   S = ∅
3   Clust = k_means(T, k)
4   Para cada cluster A_j de Clust
5           Si A_j es homegeneo entonces
6                   m = media_cluster( A_j )
7                   p_i = vecino_mas_cercano(m)
8                   S = S ∪ {p_i}
9
10          Si no
11                  C_M = clase_mayoritaria(A_j)
12                  Para cada clase C_k de A_j (C_k ≠ C_M)
13                          Para cada prototipo p_j perteneciente a la clase C_k
14                                  p_c = vecino_mas_cercano(p_j) (p_c ∈ C_M)
15                                  S = S ∪ {p_c}
16                                  p_M = vecino_mas_cercano(p_c) (p_M ∈ C_K)
17                                  S = S ∪ {p_M}
18  retorna S
```

Fig. 3.1 Pseudocódigo de PSC

La **figura 3.1** muestra el pseudocódigo del algoritmo *PSC*, *éste* inicia creando ***k*** clusters de ***T***, para cada clusters $\boldsymbol{A_j}$ se analizan las clase a la que pertenece cada instancia de dicho cluster y se decide si $\boldsymbol{A_j}$ es homogéneo o no.

Si $\boldsymbol{A_j}$ es homogéneo quiere decir que todas las instancias del cluster son de la misma clase, al no haber instancias que representen la frontera con algún otra clase todos son instancias interiores, ahora el algoritmo indica que debemos de encontrar la media aritmética ***m*** del cluster, al ser una suma de todos los elementos dividida entre en número total de éstos, la media se puede ver como el centro de gravedad de la distribución de objetos en el cluster(no necesariamente en el centro), este promedio no representa una instancia real en el cluster, por lo que es necesaria la instancia $\boldsymbol{p_i}$ más cercana a la media ***m***. *PSC* solo agrega $\boldsymbol{p_i}$ a ***S***, descartado a las demás instancias y con ello reduciendo al cluster a un elemento representativo del mismo.

Si $\boldsymbol{A_j}$ es no homogéneo entonces en el cluster existen instancias bordes debido a las distintas clases de las instancias contenidos en éste. Para encontrar instancias borde ***PSC*** hace una nueva subdivisión, esta vez el cluster se va a dividir por clases, ahora se formarán subgrupos de acuerdo a su clase de pertenencia, el subgrupo con mayor número de elementos nos indica

la clase mayoritaria C_M en A_j, que es la clase a la que más instancias en A_j pertenecen. Una vez que el subconjunto de clase C_M ha sido encontrado las instancias borde de este subgrupo son las instancias más cercanos dentro del cluster A_j pero pertenecientes a los demás subgrupos de clase C_k ($C_k \neq C_M$). De forma análoga, para obtener las instancias borde de los subgrupos con clase C_k en el cluster A_j, se buscan las instancias más cercanas de éste (*Ck*) a cada instancia borde en C_M.

Para ilustrar de forma gráfica cómo ***PSC*** reduce un conjunto de entrenamiento, en la **figura 3.2a** se muestra un conjunto de datos con dos clases('+','.') y dos atributos, para este ejemplo en particular no dividiremos el conjunto, consideraremos que el conjunto es un cluster, se puede tomar como el resultado de aplicar **k-means** con k=1, como el cluster es no homogéneo el objetivo de ***PSC*** es encontrar las instancias borde, la frontera entre clases se ilustra en la **figura 3.2b** [Olvera et al , 2010ª].

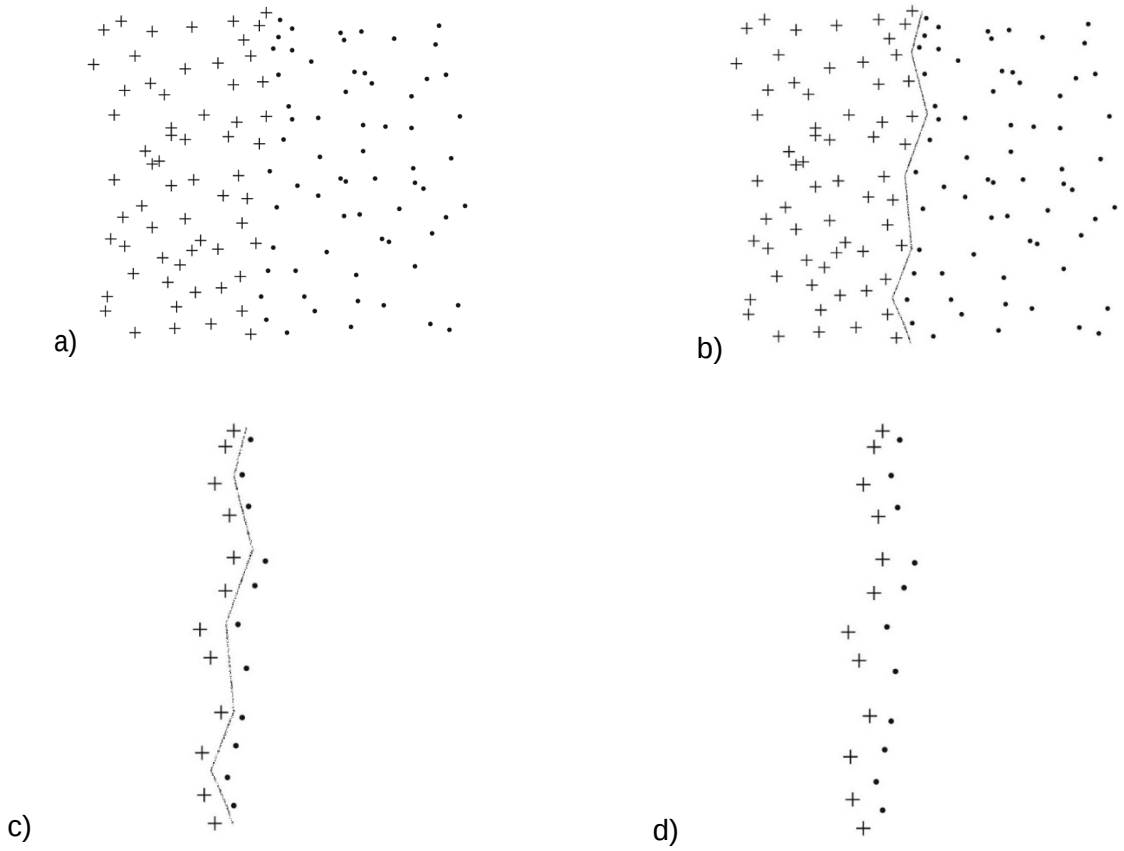


Fig. 3.2 Proceso de reducción de instancias de un conjunto sintético con dos clases mediante el algoritmo **PSC**

Podemos observar que la clase mayoritaria es '+', de esta forma las instancias de clase '.' más cercanas a la clase mayoritaria y las instancias de clase mayoritaria más cercanas a la clase restante son de igual forma las más cercanas a nuestra frontera de clase (ver **fig. 3.2c**), el subconjunto que arrojará *PSC* se puede observar en la **figura 3.2d**.

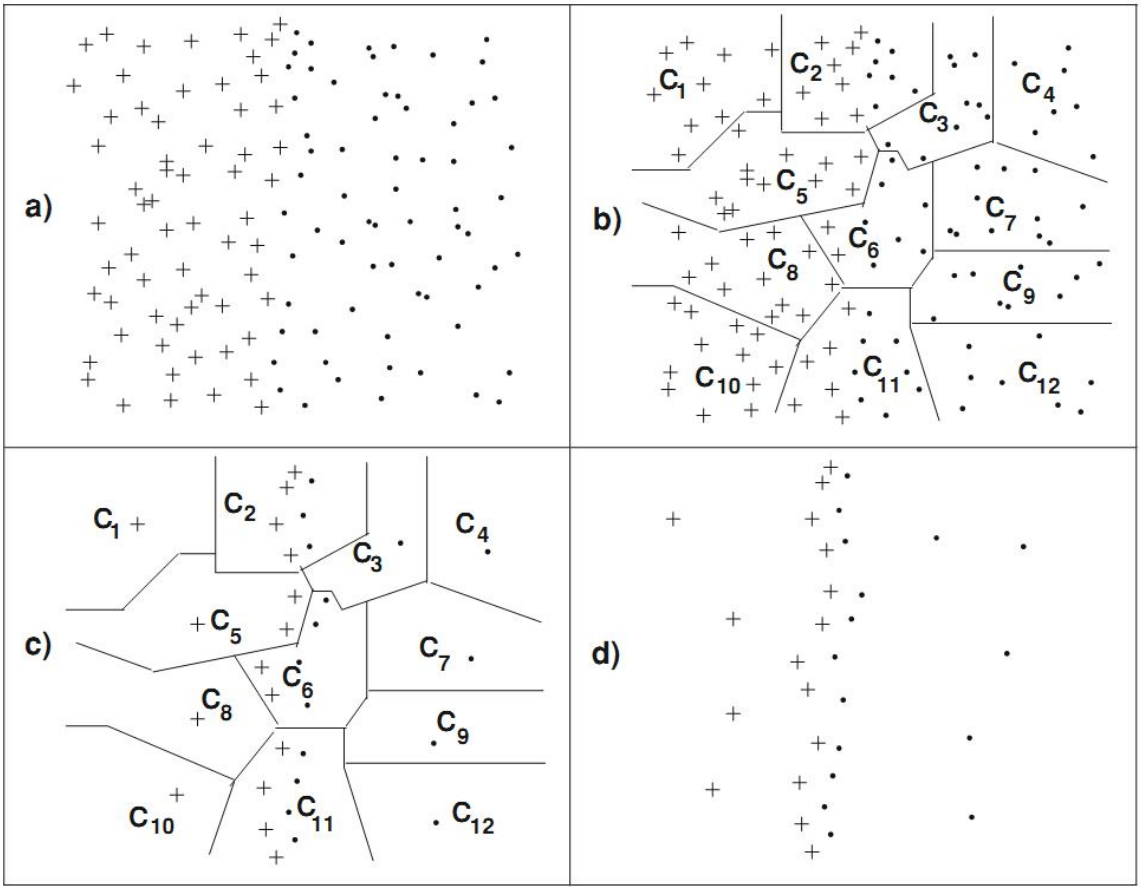


Fig. 3.3 PSC para 12 clusters

De esta misma forma se comporta *PSC* cuando el número clusters aumenta. En la ***figura 3.3*** se ilustra el mismo conjunto sintético (3.3a) al aplicar ***k-means*** con k=12 obtenemos los clusters de la **figura 3.3b**, como se puede notar, en este ejemplo $\mathbf{C_1}$, $\mathbf{C_5}$, $\mathbf{C_8}$, $\mathbf{C_{10}}$ son clusters homogéneos para la clase '**+**' de la misma forma $\mathbf{C_3}$, $\mathbf{C_4}$, $\mathbf{C_7}$, $\mathbf{C_9}$, $\mathbf{C_{12}}$ son homogéneos pero de clase '**.**', los clusters restantes son de tipo no homogéneo. Al aplicar *PSC* al conjunto de clusters la reducción de elementos la muestra la ***figura 3.3c***. Por ser subconjuntos de nuestro ejemplo

anterior, las instancias borde como podemos constatar en la ***figura 3.3d*** [Olvera et al, 2010a] son los mismos, agregando al subconjunto resultante las instancias más cercanas a la media de los clusters homogéneos.

3.3 Parallel PSC

En el algoritmo *PSC* la carga más importante de trabajo se encuentra en la creación de clusters (paso 3 en la **figura 3.1**), y en la detección de bordes para cada clusters (paso 5, ***figura 3.1***). **k-means** gasta la mayor parte de su tiempo de ejecución obteniendo las distancias entre todos los puntos con los centroides existentes y en el cálculo de la recolocación del punto medio, por tanto, al estar basados estos cálculos en iteraciones el paralelismo de datos se puede lograr en esta parte. Existen varias técnicas para paralelizar **k-means**, el enfoque en el que se basa esta implementación de **k-means** paralelo se muestra en el pseudocódigo de la ***figura 3.4***.

$$Parallel\ k-means(PkM)$$

$PkM(matriz\ \boldsymbol{M}, entero\ k)$

1 $Inicializar\ centroides\ globales\ de\ forma\ aleatoria\ \boldsymbol{C} =< \boldsymbol{C_1}, \boldsymbol{C_2}, \ldots, \boldsymbol{C_k} >$

2 $Particionar\ \boldsymbol{M}\ en\ \boldsymbol{P}\ subgrupos, \boldsymbol{P}\ igual\ al\ número\ de\ procesadores\ disponibles, cada\ subgrupo\ de\ igual\ tamaño.$

3 $Para\ cada\ \boldsymbol{P_i}$

$Asignar\ un\ procesador.$

$Enviar\ \boldsymbol{C}\ para\ calcular\ distancias\ y\ asignar\ a\ cada\ elemento\ \boldsymbol{P_i}\ un\ cluster\ mediante\ una\ lista\ \boldsymbol{l}$

4 $Recibir\ lista\ \boldsymbol{l}\ de\ elementos\ pertenecientes\ a\ los\ \boldsymbol{k}\ clusters.$

5 $Recalcular\ los\ nuevos\ centroides\ \boldsymbol{C'}.$

6 $Si\ \ \boldsymbol{diferentes}(\boldsymbol{C}, \boldsymbol{C'}).$

$Entonces\ \boldsymbol{C} = \boldsymbol{C'}.$

$regresar\ al\ paso\ \mathbf{3}.$

7 $Si\ no$

$return\ \ \boldsymbol{C}\ y\ la\ lista\ \boldsymbol{l}\ de\ elementos\ de\ pertenencia\ a\ \ cada\ cluster\ .$

8 Fin

Fig. 3.4. Pseudocódigo PKM (Parallel k-means)

Se puede ver en el algoritmo de la ***figura 3.1***, que el siguiente punto en donde se satura la carga de trabajo se encuentra en el ciclo ***Para*** (línea 4), en esta parte, el programa debe de procesar cada uno de los **clusters** para encontrar sus bordes, en la línea 12 y 13 se encuentran dos niveles iterativos anidados dentro del primer ciclo ***Para***, en los tres casos, al ser ciclos donde se repetirá la ejecución de un grupo de instrucciones , el paralelismo es posible siendo más viable en el ciclo ***Para*** del nivel uno de la anidación, pues, dentro de los dos ciclos restantes en la línea 14 y 16 existen dos zonas críticas , y el acceso a estas debe de ser de forma secuencial, lo que le sumara más tiempo de ejecución al código, al paralelizar este ciclo lo que hacemos es repartir entre los núcleos disponibles el número total de clusters, este particionamiento de trabajo divide los cálculos y al ser cada cluster independiente del resto, el paralelismo es viable. De esta forma el algoritmo de *PSC* paralelo quedaría como muestra la ***figura 3.5***.

```
1   PPSC(matriz T,entero k)
2   S = ∅
3   Clust = Pk_means(T,k)
4   Particionar los clusters en H subgrupos de tamaño igual
5   Para cada  H_i crear un nuevo proceso
6           Para  cada cluster A_j  de H_i
7                   Si  A_j  is homegeneo entonces
8                    m = media_cluster( A_j )
9                    p_i = vecino_mas_cercano(m)
10                   S = S ∪ {p_i}
11
12          Si no
13                C_M = clase_mayoritaria(A_j)
14                Para cada clase C_k de A_j (C_k ≠ C_M)
15                      Para cada prototipo p_j perteneciente a la clase C_k
16                              p_c = vecino_mas_cercano(p_j)  (p_c ∈  C_M)
17                              S = S ∪ {p_c}
18                              p_M = vecino_mas_cercano(p_c)  (p_M ∈  C_K)
19                              S = S ∪ {p_M}
20  retorna   S
```

Fig. 3.5. Seudocódigo Parallel PSC (PPSC)

CAPÍTULO 4

Paralelismo

4.1 Computo paralelo

Recientemente han aparecido en el mercado máquinas con arquitectura multi-core, con 2,4 y hasta ochos procesadores, con precios que están al alcance de mucho usuarios, se podría pensar que al adquirir estos equipos nuestras aplicaciones se ejecutan más rápido, solo que en la mayoría de los casos no es así, ya que dichos software fue diseñado con un lenguaje de programación de naturaleza secuencial, lo cual no permite explotar al máximo los recursos de una computadora con varios núcleos, para explotar al máximo el rendimiento en estos casos, es necesario el rediseño del software para que pueda funcionar en paralelo.

4.2 ¿Qué es el Paralelismo?

Es un mecanismo que involucra más de un proceso a la vez, es decir, la realización de varias tareas o procesos al mismo tiempo. También describe como la creación, destrucción e interacción de procesos que distinguen a los paradigmas de programación [Olmedo, 2004].

4.3 ¿Qué es Computación Paralela?

Por lo general, los programas son diseñados para trabajar en forma secuencial (serial), esto es que se ejecutarán sólo sobre una máquina que posee un procesador ejecutando solo una instrucción al mismo tiempo y una detrás de la otra.

La computación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional. Los recursos computacionales pueden incluir:

- Un único equipo de cómputo con múltiples procesadores
- Un número arbitrario de computadores conectados en red
- Una combinación de los dos recursos anteriores.

El cómputo paralelo es una estrategia para ejecutar tareas complejas, de forma más rápida, descomponiendo una gran tarea en tareas más pequeñas para ser ejecutadas simultáneamente, es decir en paralelo [Guillen ,2004]. La meta es reducir al mínimo el tiempo total de cómputo distribuyendo la carga de trabajo entre los procesadores disponibles, Para fines prácticos es necesario tener en mente que la paralelización es un factor básico para tener un alto desempeño en los equipos de cómputo. Una de las razones principales para utilizar el paralelismo en el diseño de hardware o software es obtener un alto rendimiento o mayor velocidad al ejecutar un programa [Olmedo, 2004].
.

Existen dos razones primarias para usar computación paralela:

- Ahorro en tiempo
- Resolver problemas cuya solución secuencial requiere de un tiempo de ejecución considerable

Otras razones:

- Tomar ventaja de recursos no locales – usar recursos computacionales disponibles en una red.
- Ahorro en costos – el uso de múltiples recursos computacionales más económicos para no rentar tiempo en una supercomputadora.
- Limitación en memoria – una sola computadora tiene recursos de memoria finita.
- Para problemas grandes, usar la memoria de múltiples computadores puede corregir este obstáculo.

Existen diferentes clases de problemas que requieren del procesamiento paralelo, como son:

- Modelaje de clima
- Dinámica de fluidos
- Dinámica de población

- Reacciones químicas y nucleares
- Bioinformática y genoma humano
- Geología y Sísmica
- Simulación de yacimientos Petrolíferos
- Biomedicina
- Manufactura
- Minería de datos
- Servicios de negocios basados en la WEB
- Diagnóstico asistido por computador
- Computación gráfica

4.4 Programa Paralelo

Un programa en paralelo tienen que cumplir ciertos requisitos para un funcionamiento correcto, de lo contrario existen la posibilidad que el funcionamiento de la aplicación no tenga los resultados esperados o sea inviable en comparación de una solución secuencial, también se debe tomar en cuenta con el hardware y software necesario y determinar si vale la pena paralelizar el programa o viceversa.

La programación en paralelo involucra descomponer un algoritmo en partes, distribuir las partes como tareas, las cuales son trabajadas en múltiples procesadores simultáneamente, coordinar el trabajo y la comunicación de estos procesadores. El programador debe conocer el número de procesadores con los que se cuenta, la cantidad de memoria, espacio en discos, niveles de memoria disponible, el sistema operativo que se está manejando, el lenguaje de programación y compilador que se utilizará, las herramientas más comunes en este caso son PVM (Parallel Virtual Machine), MPI, Charm++ (en Sistemas Distribuidos), Power C, Power Fortran u OpenMP (en sistemas de memoria compartida), Cilk o TBB para procesadores Intel.

A continuación se muestran algunas características que debe cumplir un programa paralelo:

Granularidad

Esto es el número de procesadores con los que se cuenta, podemos hablar de dos tipos de granularidad, paralelismo de grano fino que es cuando el código se logra dividir en

una gran cantidad de piezas pequeñas, también se le conoce como paralelismo de datos. Por otra parte tenemos al paralelismo de grano grueso o también llamado paralelismo de tareas, aquí la división de tareas es poca y de trabajo más intenso [García et al ,2004].

Sincronización

Se debe evitar que uno más procesos dependan, accedan o demanden el mismo recurso al mismo tiempo, por ejemplo usar las mismas variables globales, usar el mismo dispositivo de entrada o salida.

Aceleración y eficiencia

La aceleración de un algoritmo paralelo es la razón entre el tiempo que tarda un algoritmo secuencial en ejecutarse usando una computadora con un procesador y el tiempo que tarda el mismo algoritmo en ser ejecutado en la misma computadora usando n procesadores.

Podemos medir la eficiencia de un algoritmo paralelo obteniendo la aceleración del algoritmo ejecutado en n procesadores y dividirlo entre los n procesadores. La ley de Amdahl nos dice que finalmente llegara un límite en que ya no se puede mejorar más estos dos tiempos, o sea, que no se pueda paralelizar más el algoritmo.

4.5 Diseño de algoritmos paralelos

El bueno diseño de un algoritmo paralelo es una tarea importante para la solución de los problemas en los diversos campos de aplicación y no es tarea fácil, el diseño involucra cuatros etapas, estas etapas no necesariamente son secuenciales ni llevan un orden definido. Las etapas para el diseño de un algoritmo paralelo son las siguientes:

1. **Particionamiento**. El cómputo y los datos sobre los cuales se opera se descompone en pequeñas tareas. Usualmente es independiente de la arquitectura o del modelo de programación.

2. **Comunicación**. Las tareas generadas por una partición están propuestas para ejecutarse concurrentemente pero no pueden, en general, ejecutarse independientemente. Los cálculos en la ejecución de una tarea normalmente requerirán de datos asociados con otras tareas. Los datos deben transferirse entre las tareas y así permitir que los cálculos procedan. Este flujo de información se especifica en esta fase.

3. **Aglomeración**. Las tareas y las estructuras de comunicación definidas en las dos primeras etapas del diseño son evaluadas con respecto a los requerimientos de ejecución y costos de implementación. Si es necesario, las tareas son combinadas en tareas más grandes para mejorar la ejecución o para reducir los costos de comunicación y sincronización.

4. **Mapeo**. Cada tarea es asignada a un procesador de tal modo que intente satisfacer las metas de competencia al maximizar la utilización del procesador y minimizar los costos de comunicación [Salcedo, 2007].

4.6 Arquitecturas Paralelas

Para poder resolver el problema de mapeo debemos conocer la arquitectura y organización de las computadoras paralelas según el tipo de hardware,

4.6.1 Multiprocesadores y Multinúcleos

Cuando hablamos de multiprocesadores nos referimos a computadoras con múltiples CPU´s y múltiples zócalos (uno por procesador), que comparten memoria, cada procesador es independiente y ejecuta su propia tarea.

En una arquitectura multinúcleo (o multi-core) cada procesador contiene múltiples procesadores (o núcleos) en un mismo circuito integrado trabajando a una misma frecuencia, donde cada uno de estos cuenta con una unidad de proceso independiente [Luna et al, 2010].

Para ambas arquitecturas, si el acceso a memoria es uniforme (**UMA, uniform memory access**), es decir que todos los procesadores tienen el mismo tiempo para acceder a la memoria y periféricos de *entrada*/*salida*. Este enfoque recibe el nombre de memoria compartida centralizada (**fig. 4.1**, [Bastida Ibáñez, 1995]), esta forma de intercomunicación se implementa a través de un mecanismo de interconexión como puede ser un bus o una red conectada, así, todos los procesadores acceden a este mecanismo para comunicarse con la memoria y los dispositivos *entrada*/*salida*.

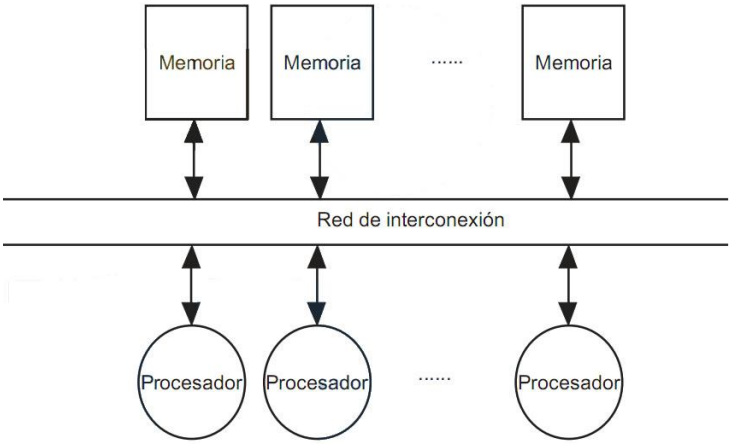


Fig. 4.1. Sistema con acceso a memoria uniforma (UMA)

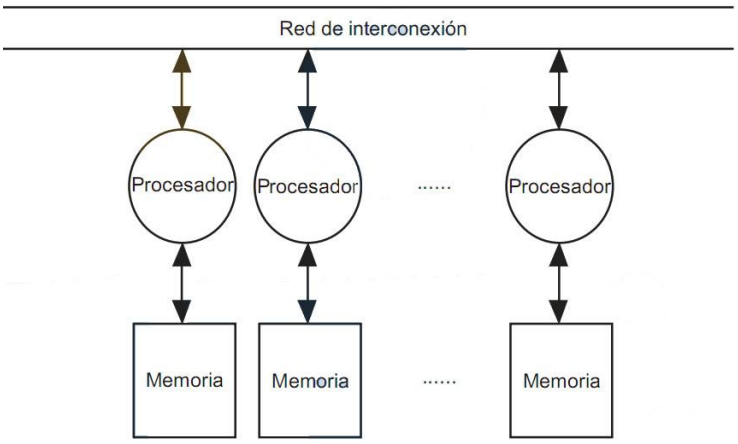


Fig. 4.2. Sistema con acceso a memoria no uniforma (NUMA)

Si en dicha computadora el acceso de la memoria por los diferentes procesadores no es uniforme (**NUMA**, **nonuniform memory access**) se dice que la memoria compartida es distribuida, ya que cada procesador tiene memoria local y el espacio de direcciones compartido está formado por la combinación de estas memorias locales [Salcedo, 2007]. En este esquema

de memoria compartida el tiempo de acceso a una localidad de memoria depende si este fragmento de memoria es local, cada procesador tiene una memoria que puede utilizar directamente (ver **fig. 4.2**, [Bastida Ibáñez, 1995]), pero si se desea hacer uso de otra memoria, dicho procesador tiene que hacer una petición a través de la red para solicitar la conexión y acceso.

COMA (Cache Only Memory Access) Un multiprocesador que solo use cache como memoria es considerado de tipo COMA (**figura 4.3,** [Bastida Ibáñez, 1995]**).** En realidad, el modelo COMA es un caso especial del NUMA donde las memorias distribuidas se convierten en caches, no hay jerarquía de memoria en cada módulo procesador. Todas las caches forman un mismo espacio global de direcciones. El acceso a las caches remotas se realiza a través de los directorios distribuidos de las caches. Dependiendo de la red de interconexión empleada, se pueden utilizar jerarquías en los directorios para ayudar en la localización de copias de bloques de cache. El emplazamiento inicial de datos no es crítico, puesto que el dato acabará estando en el lugar en que se use más [Bastida Ibáñez, 1995], en las máquinas con esta organización la distribución de los datos se realiza dinámica y automáticamente, esto es así porque cada memoria local (llamada memoria de atracción) va cargando copia de los bloques que necesita el resto de las memorias locales (memorias cache).

En los tres tipos de sistemas de memoria compartida cualquier procesador puede acceder a cualquier localidad de memoria.

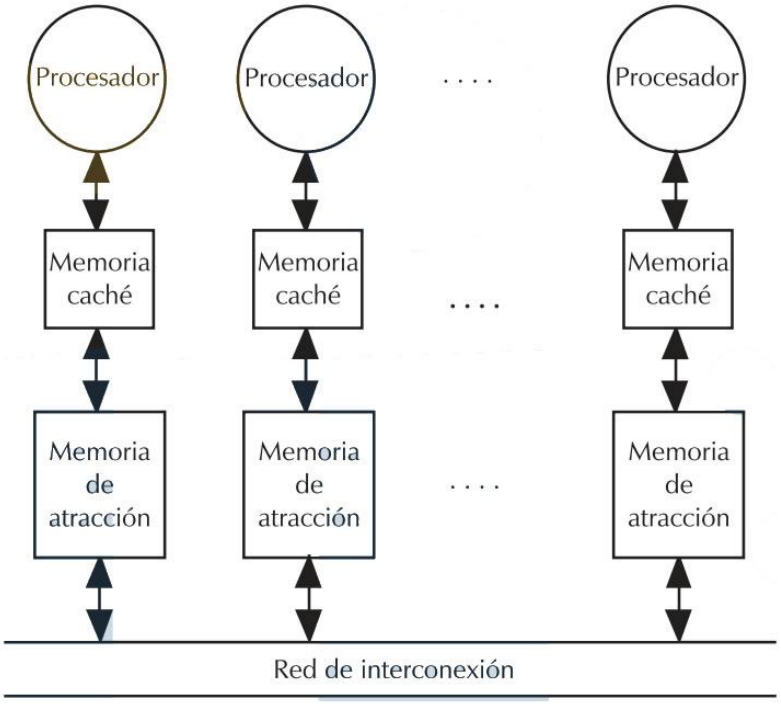


Fig. 4.3. Sistema solo con memoria cache (COMA)

4.6.2 Multicomputadoras

Una multicomputadora es un conjunto de computadoras constituidas por una unidad central de procesamiento que ejecuta programas los cuales acceden a una memoria asociada para lectura y escritura. Cada nodo se considera autónomo en el manejo de su CPU, memoria, discos y periféricos de entrada/salida, cada programa de cada nodo al ejecutarse puede acceder a memoria local y puede enviar y recibir mensajes a través de la red, estos mensajes son usados para la comunicación entre las computadoras interconectadas o para leer y escribir en memoria remota. Al poder acceder cada proceso a su propia memoria la programación de este tipo de sistema de memoria distribuida suele ser más complejo ya que si un proceso necesita la información que está dentro del espacio de memoria de otro proceso, es necesario acceder una petición para la trasferencias de estos datos en forma de mensajes. Si esto no se hace de forma adecuada tomando en cuenta la minimización de comunicación entre procesadores se puede obtener un mal rendimiento de la red.

La ventaja que tienen los sistemas de memoria distribuida es la escalabilidad debido a que se puede anexar un número mayor de procesadores, este enfoque es el más adecuado para sistemas con una gran cantidad de CPU´s. Su principal desventaja es al compartir información ya que ésta debe de ser transferida físicamente a la memoria local de cada nodo en la red antes de la ejecución, por ello este tipo de arquitectura es menos flexible, un ejemplo claro de esto es que los multiprocesadores podrían emular el paso de mensaje al usar localidades compartidas para almacenar mensajes, pero las multicomputadoras son muy ineficientes para emular operaciones de memoria compartida [Salcedo, 2007].

Con base en los párrafos anteriores puede notarse que la programación paralela dependerá del tipo de arquitectura, una arquitectura de multiprocesadores y multi-core es orientada a la programación de memoria compartida, mientras que una aplicación con memoria distribuida cuya arquitectura es de multicomputadoras deberá comunicarse mediante paso de mensajes.

4.7 Programación para memoria compartida

En los sistemas de multiprocesadores y multi-core cada uno de estos es capaz de acceder a toda la memoria libremente, de esta forma, todos los procesadores se encuentran comunicados, pero esta libertad de acceso debe de asegurar que dos o más procesadores no accedan al mismo tiempo al mismo recurso ya que puede provocar un error, por ejemplo, en un índice, si dos procesos acceden a la variable compartida al mismo tiempo y la incrementan, el valor de dicha variable debió ser incrementado dos unidades, pero al haberlo hecho al mismo tiempo solo se incrementó una, por ellos se debe se establecer procedimiento que permitan acceder a estas secciones criticas de manera atómica. Esto es que mediante técnicas de exclusión mutua, como semáforos, candados, monitores y mecanismo de sincronización se evite el acceso a una misma variable de manera simultánea.

4.8 Programación con paso de Mensajes

Cuando la memoria es distribuida como es el caso del esquema paralelo de multicomputadoras, cada procesador tiene su propia memoria, eso hace que la programación sea más compleja, ya que si los datos a utilizar por el procesador local están en la espacio de direcciones de otro, es

necesario solicitarlos mediante una petición y luego transferir dichos datos mediante mensajes. Un sistema con buen rendimiento con este tipo de configuración tiene como prioridad minimizar la comunicación entre procesadores.

Los programas que utilizan paso de mensajes son totalmente estructurados, frecuentemente, todos los nodos ejecutan copias idénticas de un programa, con el mismo código y variables privadas, la ventaja que presenta este tipo de sistemas es la escalabilidad, pudiendo crecer un número mayor que los sistemas de memoria compartida, siendo más adecuado para cómputo paralelo[Salcedo,2007].

4.9 Comunicación

Los algoritmos o modelos de comunicación para la distribución de datos que cumplen con el objetivo de paralelizar la aplicación dependen de igual medida de la aplicación y de acuerdo a la disponibilidad de recursos. En la actualidad, casi todas las aplicaciones están identificadas a un paradigma de programación paralela, como puede ser Divide y Vencerás, Pipeline, Maestro / Esclavo y SPMD (Single Program Multiple Data) a continuación se describen dos de los enfoques de comunicación más comunes.

Maestro esclavo

Este tipo de comunicación consiste en la implementación de dos entidades, una de ellas es el maestro que se encarga de recolectar, dividir y distribuir las tareas de la aplicación, y la otra son los esclavos, quienes reciben información, la procesan y envían los resultados al maestro. Este tipo de comunicación permite el procesamiento en paralelo, pues cada esclavo trabaja independientemente de los demás. También, se le conoce como comunicación global, porque la información se concentra en un procesador (maestro) en un esquema de balanceo dinámico, el cual es útil cuando se tiene una aplicación donde el número de tareas es mayor al número de procesadores que la ejecutan, o si se desconoce el número total de tareas, esto motivado a la habilidad de la aplicación de adaptarse a los cambios de heterogeneidad de los sistemas que la estén ejecutando [Muresano, 2008].

SPMD (Single Program Multiple Data)

En este paradigma se escribe únicamente un programa (código) y todos los procesadores ejecutan dicho programa, los datos son divididos en trozos y cada uno de ellos se asigna a un

procesador, en este enfoque no debe de existir dependencias de datos para que el paralelismo sea eficiente, ya que si existen trozos de código que dependa del resultados de otros datos, se tiene que implementar una rutina de sincronización para que el programa trabaje correctamente.

CAPÍTULO 5

Implementación

En esta sección se aborda el proceso de implementación de PSC paralelo en lenguaje C con rutinas de la API OpemMP, los detalles del funcionamiento de esta API se describen en el apéndice B.

5.1 Justificación

Actualmente existe librerías y software diseñado para las tareas de aprendizaje automático, pre-procesamiento y modelado; La docencia e de investigación son las áreas en las que éstas herramientas son más utilizadas, dichas colecciones de código nos permiten realizar tareas como pre-procesamiento de datos, clustering , clasificación, regresión, visualización, selección y evaluación del desempeño de clasificadores. A continuación se muestran una lista de las principales bibliotecas de códigos para aprendizaje automático, también llamadas toobox o toolkit.

- Statistics toolbox de MATLAB
- Weka(Waikato Environment for Knowledge Analysis)
- MALLET (MAchine Learning for LanguagE Toolkit)
- Milk: Machine Learning Toolkit for Python
- SIGM A:Large Scale Machine Learning Toolkit - Microsoft Research
- PyBrain (Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library)
- Elefant (Efficient Learning, Large-scale Inference, and Optimisation Toolkit)
- Waffles: A Machine Learning Toolkit
- Orange – Data Mining Fruitful & Fun
- SIGMA: Large-Scale and Parallel Machine-Learning Tool Kit .
- NI LabVIEW Machine Learning Toolkit

La mayoría de estos paquetes incluyen los algoritmos más conocidos para clasificación y clustering, algunos paquetes contienen opciones de visualización de gráficas de dispersión, filtros para trasformación de datos y selección de atributos, en las últimas versiones muchos de este software incluyeron versiones en paralelo de los algoritmos más conocidos. Sin embargo *pese a esto,* ningún paquete incluye el algoritmo PSC, lo que nos impide tener un punto de comparación y nos lleva a la tarea de implementar nuestro propio algoritmo.

5.2 Visión general

El diagrama de la ***figura 5.1*** muestra de manera general el flujo de ejecución el programa del archivo principal, los parámetros que recibe son el número de clusters, el umbral de **k-means**(variable de convergencias) y matriz de aprendizaje.

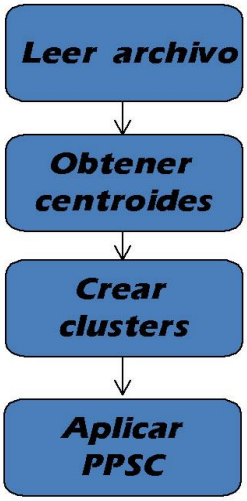


Fig. 5.1. Diagrama de flujo del programa principal

El programa inicia leyendo el archivo que se le pasa como parámetro al ejecutarlo en consola. El prototipo de la función para leer el archivo de entrenamiento es el siguiente:

```
DATA * file_read(char *file_name )
```

La función regresa una estructura tipo **DATA** en la cual se almacena la matriz de aprendizaje, la matriz de cabeceras[9], un vector con la clase correspondiente de cada objeto, y dos variables enteras que contienen el número de objetos y atributos. La estructura tipo DATA se presenta continuación, la cual contiene la lista de variables en el orden antes mencionando.

```
typedef struct {
    double  **  Objetos;
    int     **  cabeceras;
    int     *   clases;
    int         numObjetos;
    int         numCoordenadas;
    }DATA;
```

El prototipo de función para obtener los centroides es el siguiente:

```
double ** k_means_paralelo( double **  objetos,
                            int        numCoordenadas,
                            int        numObjetos,
                            int        numClusters,
                            double     umbral)
```

k_meas_paralelo() devuelve una matriz con las coordenadas de los centroides. El paralelismo dentro de la función se encuentra en los dos ciclos principales **for**, el primer ciclo es el encargado de obtener las distancias de cada objeto con los clusters mediante la función de distancia euclidiana y asignar cada objeto al cluster que le corresponde

```
/* buscar el centroide mas cercano al Objeto i */
#pragma omp parallel for \
                    private(i,j,indice) \
                    schedule(static) \
                    reduction(+:delta)
for (i=0; i<numObjetos; i++)
```

El segundo ciclo es el encargado de obtener la media de cada cluster, si ésta cambia, actualiza los centroides.

9 El formato de archivo .trn se describe en el apéndice A

```
/* Obtener la media de los clusters*/
 #pragma omp parallel for \
                    private(i,j) \
                    schedule(static)
for (i=0; i<numClusters; i++)
```

Para obtener los clúster se utiliza la función siguiente:

```
model * clasificar_set_paralelo( double  ** test_set,
                                  int          numObjetos,
                                  int          numCoordenadas,
                                  int          numClusters,
                                  double  **   centroides);
```

El área paralela de esta función se encuentra en el ciclo que asigna a cada objeto el clusters más cercano de acuerdo a los centroides obtenidos con la función anterior.

```
/*Asignar cluster a cada objeto de T*/
 #pragma omp parallel for \
            private(i) \
            shared(numObjetos,numClusters,numCoordenadas,test_set,centroides) \
            schedule(static)
for(i=0;i<numObjetos;i++)
```

classificar_set_paralelo() retorna una estructura tipo ***model***, la cual encapsula dos tipos de variables, la primera es una arreglo de lista ligada en donde se almacenan el números de objeto de cada clúster y un vector que almacena el número de clúster al que pertenece cada objeto.

```
typedef struct{
   LISTA *L;
   int * V;
}model;
```

El proceso final en el diagrama de flujo la **figura 5.1** es la aplicación de PPSC, la función que realiza este proceso es ***PSC_paralelo()***, el prototipo de esta función es el siguiente:

```
DATA * PSC_paralelo (   model *     S ,
                        float **    objetos,
                        int         numCoordenadas,
                        int *       clases,
                        int         numClusters,
                        float **    centroides,
                        int         numClases);
```

El primer parámetro en su lista de entrada es la estructura **model**, en la implementación del código este parámetro es obtenido de la función ***clasificar_set_paralelo(),*** los siguientes parámetros, a excepción del último, que almacena el número total de clases, representan lo que ya se explicó en las funciones anteriores, y al igual que en estas el paralelismo se encuentra en los ciclos ***for*** principales.

```
#pragma omp parallel for \
private(i,clase,reinsidencias,aux,numObjeto_real,claseMay) \
schedule(static)
for(i=0;i<numClusters;i++)
```

Para este caso el bucle es el encargado de decidir cuáles instancias son las más relevantes de entre el total de clusters, estas son almacenados en una lista, al paralizarse, lo que el compilador hace es dividir el número total de iteraciones (número de clusters) entre el número de núcleos disponibles (***schedule(static)***), con el resultado de esta operación repartirá las ***i*** iteraciones entre los núcleos. Para un caso general supongamos que tenemos **c** clusters y ***k*** procesadores disponibles, lo que el compilador hará es dividir los ***c*** clusters en subconjuntos de $\frac{C}{K}$ elementos y enviará cada subconjunto a un núcleo (ver *figura 5.2*), inmediatamente cada núcleo procesará su arreglo de clústers.

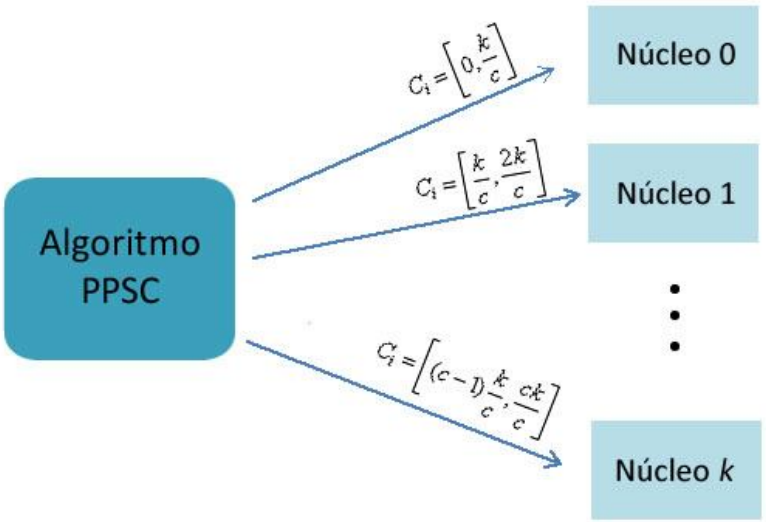


Fig. 5.2. Diagrama que ilustra el reparto de trabajo

La función devuelve una estructura ***Data*** que contiene la matriz reducida y la clase de cada objeto. El programa finaliza escribiendo a un archivo (con el formato .trn) el conjunto reducido.

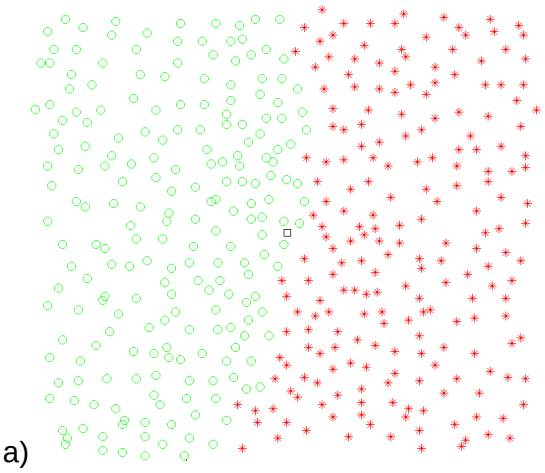
CAPÍTULO 6

Pruebas y Experimentos

En esta sección se realizan las pruebas del algoritmo PSC, se mide el tiempo de ejecución y se comparan los resultados.

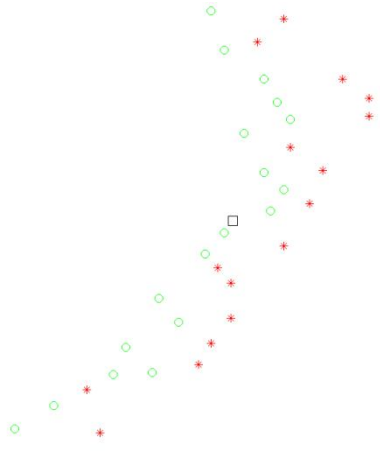
6.1 Pruebas con datos sintéticos

En esta fase, los datos de prueba que recibirá el algoritmo son de un conjunto sintético *T*. Las figuras **6.1a, 6.2a** y **6.3a** muestras la dispersión de instancias de *T*, los centroides y los subgrupos formados al aplicar *k-means* para 1,3 y 9 clusters.



a)

Fig. 6.1a) Conjunto sintético T



b)

Fig. 6.1b) Subconjunto S, resultado de aplicar PSC para 1 cluster.

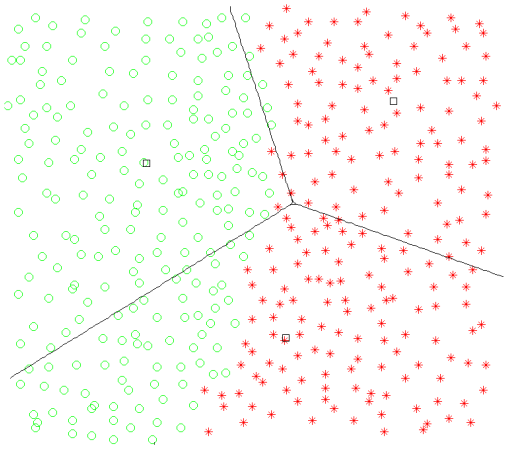


Fig. 6.2 a) Conjunto sintético T dividido en 3 clusters.

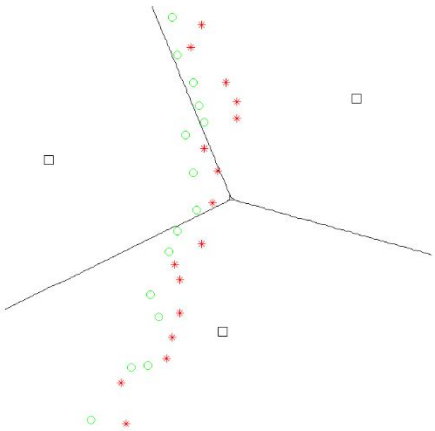


Fig. 6.2 b) Subconjunto S, resultado de aplicar PSC para 3 clusters.

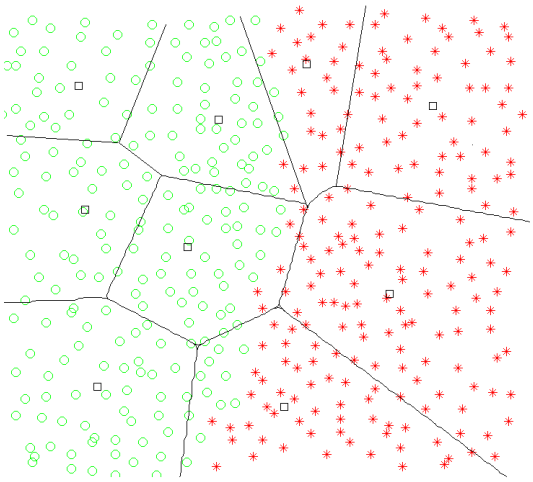


Fig. 6.3a) Conjunto sintético T dividido en 9 clusters

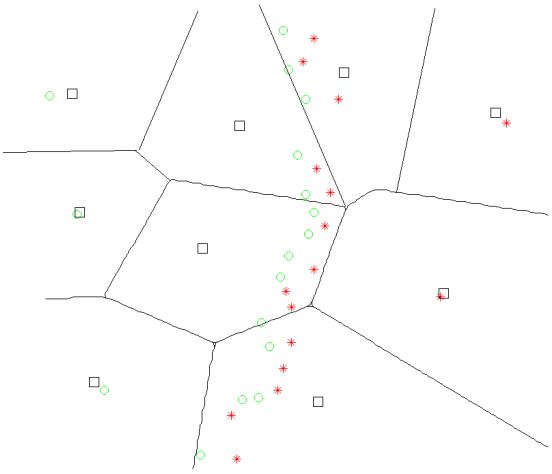


Fig. 6.3b) Subconjunto S, resultado de aplicar PSC para 9 clusters al conjunto T.

Por otra parte las figuras **6.1b, 6.2b** y **6.3b** muestran la retención de *PSC*, el subconjunto S contiene las instancias que son consideradas fronteras de clases en los clusters no homogéneos, para los clusters homogéneos retiene las instancias representantes del cluster, que es el más cercano al centroide. Cuando se ejecuta *PSC* con solo un cluster (**fig. 6.1a**), se retienen las instancias del conjunto que representan las fronteras entre el total de clases,

siento esta la cantidad de retención mínima de instancias, para un mayor número de clusters, la cantidad de instancias que se mantiene en el conjunto debe ser mayor o igual a lo obtenido para un clúster.

La **figura 6.4** contiene las gráficas de frecuencia de clases del conjunto original y el conjunto reducido de la **figura 6.1**, dichas figuras muestran que la retención de objetos por clase tiende a ser proporcional al conjunto original, que contiene el 50% para ambas clases, siendo de 52% para la clase 1(azul) y 48 % (rojo) en el conjunto reducido, esto es de esperarse debido a que el conjunto sintético contiene un distribución homogénea de datos y una distribución uniforme de clases.

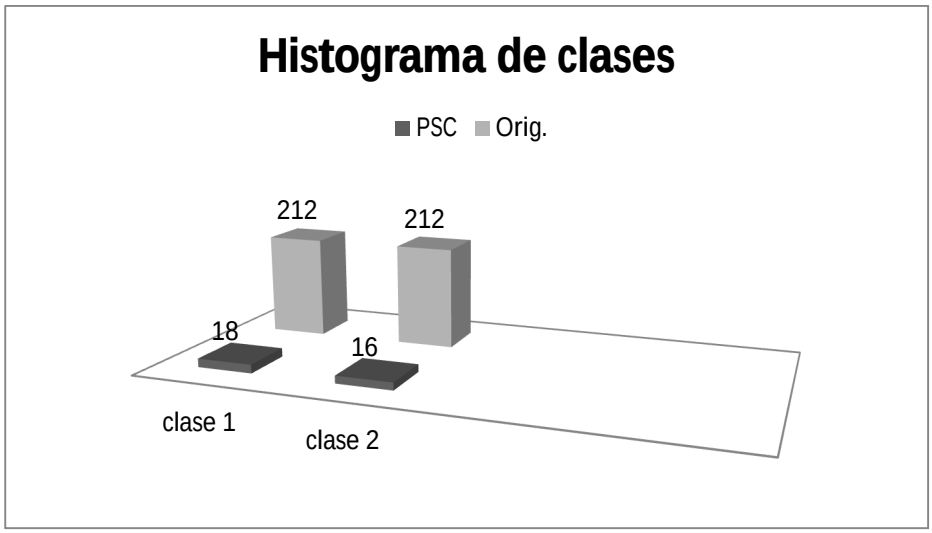


Fig.6.4 Histograma de clase del conjunto T (Orig.) y sub conjunto S (PSC) de T

6.2 Pruebas de comparación

Ahora se aplican las pruebas a los conjuntos que marca la ***tabla 6.1***, la columna marcada con *Articulo* fue extraída de [Olvera et al, 2010a], en este artículo el autor propone el algoritmo PSC, el cual es nuestro punto de comparación, dicha tabla se divide en tres sub columnas, **Acc Org** muestra la precisión obtenida en los conjuntos originales, mientras que **Acc PSC** muestra la predicción obtenida al reducir el conjunto con PSC, la última sub columna llamada **Retención** nos muestra el porcentaje de instancias que no fueron seleccionadas del conjunto original al

aplicar el algoritmo. De forma análoga la columna **Proyecto** con sus tres sub columnas nos muestran los datos que arrojó el algoritmo secuencial que se ha implementado.

Nombre de la B.D.	*Proyecto*			*Artículo*		
	Retención	*Acc. Orig.*	*Acc. PSC*	**Retención**	**Acc. Orig**	*Acc. PSC*
Iris	27(18%)	96.00%	94.70%	38.45%	94.66	94.66
Sonar	80 (38.5%)	83.30%	72.30%	35.41%	82.71	79.76
Glass	95 (44.4%)	69.10%	63.33%	38.45%	71.42	59.09
Thyroid	40 (19.6%)	94.10%	88.60%	13.47%	95.45	90.75
Liver	58(45.2%)	66.10%	54.00%	45.87%	65.22	45.87
Yeast	713(48%)	53.20%	46.40%	43.20%	54.64	43.2
Segmentation	370(16%)	95.40%	88.30%	15.14%	95.1	91.28
Spambase	1,123(24.4%)	81.30%	70.30%	24.74%	80.55	71.95
Pendigits	438(5.8%)	99.40%	94.50%	6.23%	99.43	94.32
Letter	3378(16.9%)	95.00%	81.20%	18.33%	95	79.35

Tabla 6.1 Tabla de precisión y retención de PSC

El porcentaje de retención y precisión obtenido es muy similar en todas las columnas.

Nombre	***Tamaño***			***Tiempo PSC***	
Conjunto de Datos	Instancias	Atributos	Clases	Paralelo	Secuencial
Iris	150	4	3	0.02	0.00
Sonar	208	60	2	0.02	0.01
Glass	214	9	6	0.04	0.01
Thyroid	215	5	3	0.03	0.00
Liver	345	6	2	0.03	0.01
Yeast	1,484	8	10	0.04	0.02
Segmentation	2,100	19	7	0.05	0.04
Spambase	4,601	57	2	0.26	0.50
Pendigits	7,494	16	10	0.20	0.25
Letter	20,000	16	26	0.83	1.45

Tabla 6.2. Tabla comparativa de los tiempos de ejecución de PSC

En la referencia citada anteriormente menciona que las pruebas se realizaron en un CPU Intel Celeron CPU 2.4 GHz, 512 MB RAM, el CPU en las que se ejecutaron las prueba del proyecto es un AMD Turion 64 X2 Dual-Core 1.8 Ghz, 2 GB RAM.

La **tabla 6.2** contiene las rutinas de tiempos de ejecución del algoritmo PSC con ***C =6r***, siendo ***C*** el número de clústers, y ***r*** el número de clase del conjunto[10], los tiempos están expresados en segundos, los datos de la columna *Articulo* son tomados de [Olvera et al, 2010a], mientras que las dos sub columnas restantes *Secuencia* y *Paralelo* representan los tiempos del algoritmo implementados de forma secuencia y de forma paralela respectivamente.

Analizando la tabla podemos encontrar que los tiempos de ejecución no se reducen en la mayoría de los casos, el algoritmo paralelo para los primeros seis conjuntos sobrepasa el tiempo que tarda el algoritmo secuencial.

Las figuras ***6.5*** y **6.6** representan los histogramas de frecuencia de clase de los conjuntos *iris* y *letter*, con dichos histogramas podemos observar la frecuencia de las instancias de acuerdo a su clase antes y después de aplicar *PSC*, la retención de instancias no fue proporcional al porcentaje original como en el conjunto sintético, para la mayoría de bases de datos es más complicado comprender por qué la retención de instancias de ciertas clases es mayor debido a que éstas por lo regular contienen más de dos atributos, lo que no permite representar de forma gráfica la dispersión de objetos en un plano. Se muestran los histogramas de estos dos conjuntos en específico debido a que representan el mejor y el peor resultado en cuanto a precisión al ser sometidos a PSC.

[10] Número de cluster con el que se obtuvo mejores resultados en la precisión de los clasificadores.

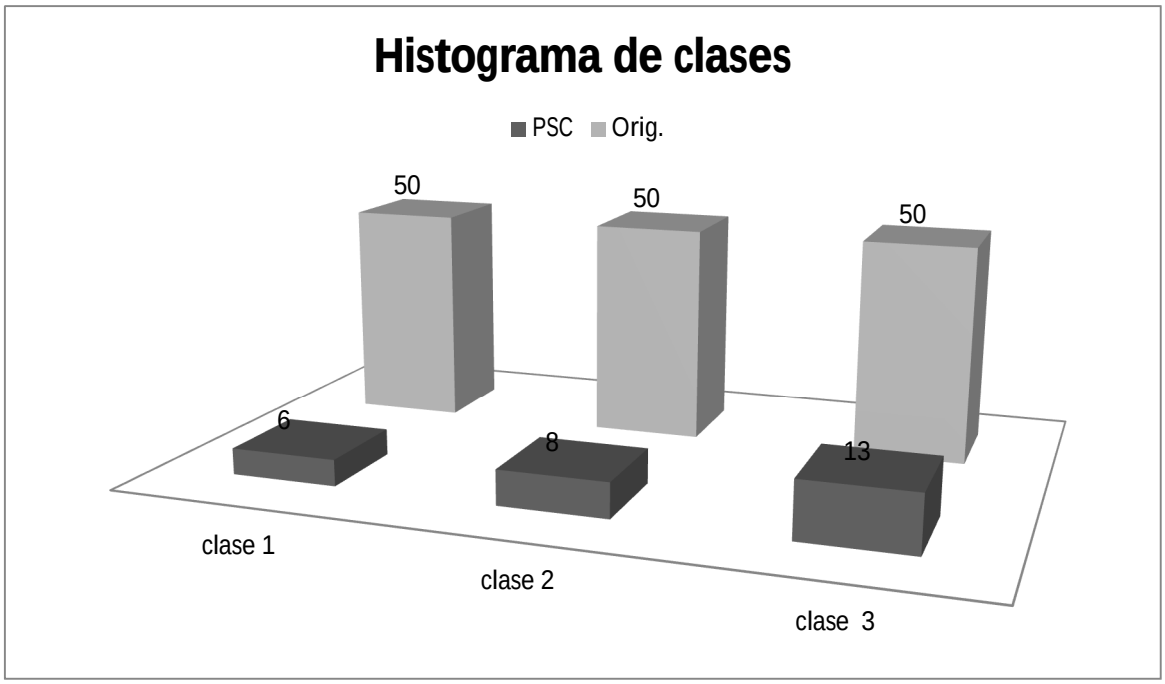


Fig. 6.5. Histograma de la base de datos iris y el sub conjunto S de iris.

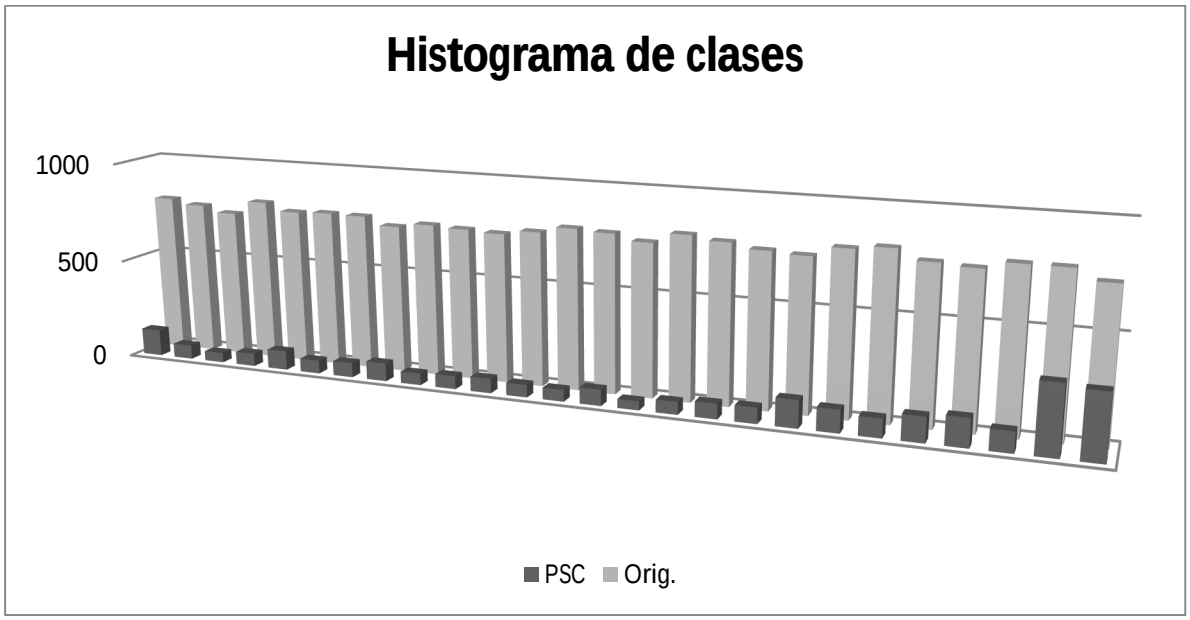


Fig. 6.6 Histograma de la base de datos letter y el sub conjunto S de la B.D. letter.

Por diferentes motivos como son: instancias repetidas, ya que PSC funciona como filtro para el problema de redundancia de datos, instancias demasiado dispersas y mezclado con las demás instancias de las distintas clases, lo cual no permite que exista una frontera bien definida para las distintas clases, la distribución desproporcionada de objetos de acuerdo a su clase, este número de muestras significativamente menor que la de las demás clases puede conservarse después de aplicar *PSC*; Cualquiera de estos casos puede afectar la retención de objetos que dicta PSC y resultar en una baja de precisión a la hora se someter el conjunto a un clasificador.

6.3 Pruebas de precisión

Las últimas pruebas se realizaron con bases de datos mucho más grandes (***tabla 6.3***), debido a que el objetivo principal es explotar al máximo los recursos de una computadora multi-core y analizar la reducción en el tiempo de ejecución, como se observó anteriormente al medir rutinas de tiempo de las bases de datos pequeñas el paralelismo no logra reducir de forma importante estos tiempos, para bases de datos más grandes esperamos que estos resultados cambien. En esta fase se realizan experimentos con bases de datos mucho más grandes, con las cuales se pueda observar una reducción de tiempo de ejecución más notable que en las pruebas anteriores.

	Nombre	**Instancias**	**Desc**	**Atributos**	**Clases**	**3-NN**	**Tiempo**
1	PAMAP2 Physical (pamap4)	703,760	0	52	19	99.90	812.83
2	Record Linkage Comparison(donation)	689,063	2	9	2	96.70	866.50
3	Covertype	581,012	44	10	7	97.04	500.07
4	KDD Cup 1999 (KDDCup)	494,021	7	34	22	99.80	1177.00
5	Skin Segmentation	245,057	0	3	2	98.70	78.23
6	Localization Data for Person Activity	164,860	0	4	11	99.60	36.14
7	Stalog (Shuttle)	58,000	0	9	7	99.80	4.96
8	Adult	48,842	0	6	2	90.20	3.26
9	Bank Marketing	45,211	9	7	2	87.60	2.84
10	p53 Mutants	31,159	0	5,408	2	99.50	117.66

Tabla 6.3.Bases de datos y descripción

La ***tabla 6.3*** muestra diez bases de datos con su descripción, la columna ***Instancias*** contiene el número de instancias de la base de datos, la columna ***Desc.*** muestra los atributos descartados de tipo categórico ya que la función de distancia euclidiana, utilizada en la implementación solo acepta datos de tipo numérico, la siguiente columna representa el número de atributos de cada conjunto, *Clases,3-NN y Tiempo* representan respectivamente el número de clases, la precisión de k-NN paralelo con k=3 y el tiempo de clasificación(minutos) .

En las pruebas de clasificación se utilizó validación cruzada aleatoria estratificada para la obtención de la precisión, con 10 particiones para los conjuntos de datos más pequeños de la lista y 3 particiones para los conjuntos más grandes.[11]

6.4 Pruebas de retención

Como ya se conoce los detalles necesarios de las bases de datos, en las siguientes pruebas se aplica PSC a cada una de los conjuntos de la ***tabla 6.3***, se obtiene su porcentaje de retención y se mide el tiempo de ejecución, primero, se ejecuta el código del programa secuencial, este nos arrojara como salida el número de objetos, el porcentaje de retención, la lista de centroides, el tiempo de ejecución(sin tomar en cuenta la obtención de centroides) y el sub conjunto S, una vez obtenidos los centroides, éstos servirán de parámetro inicial para la ejecución el código paralelo, los parámetros de salida son los mismo que el código anterior; El porcentaje de retención, y el sub conjunto S coinciden ya que es el mismo conjunto, el tiempo que tarda en su ejecución es lo más relevante.

A continuación se muestran, divididos en diez tablas, los resultados de estas prueba, una por cada conjunto de entrenamiento, las columnas en cada tabla muestran el porcentaje de retención, los tiempos de ejecución en segundos y el valor que toma la constante ***c*** para la expresión ***C=cr***, esto quiere decir que en cada fila el número de clusters con el que trabaja el algoritmo es igual al valor de la constante ***c*** multiplicado por el número de clases del conjunto.

Se espera que el tiempo de ejecución aumente en proporción al número de instancias y el número de atributos.

[11] También se utilizó un CPU AMD Phenom II X2, 3.2GHz con 8 Gb. De RAM

cr	Retención	T. secuencial	T. paralelo
2r	0.7%	50.67	28.09
4r	0.7%	47.12	25.23
6r	0.7%	44.57	33.55
8r	0.7%	32.84	14.11
10r	0.8%	35.23	17.61

Tabla 6.4. Resultados conjunto P53 Mutants

cr	Retención	T. secuencial	T. paralelo
2r	16.3%	14.75	12.02
4r	16.3%	8.58	4.87
6r	16.3%	6.48	3.18
8r	16.2%	5.11	3.1
10r	16.3%	4.2	1.86

Tabla 6.5. Resultados conjunto Bank.

cr	Retención	T. secuencial	T. Paralelo
2r	13.5%	13.02	7.69
4r	13.4%	7.93	4.16
6r	13.5%	6.32	2.91
8r	13.4%	4.93	1.96
10r	13.4%	4.61	2.41

Tabla 6.6. Resultados conjunto Adult.

cr	Retención	T. secuencial	T. Paralelo
2r	2.9%)	9.42	4.54
4r	2.1%)	5.22	2.82
6r	2.2%)	2.81	1.19
8r	1.7%)	2.39	1.43
10r	1.4%)	1.75	0.79

Tabla 6.7. Resultados conjunto Shuttle.

cr	Retención	T. secuencial	T. Paralelo
2r	0.5%	65.51	50.71
4r	0.5%	37.44	29.19
6r	0.5%	27.04	18.61
8r	0.6%	20.09	15.82
10r	0.6%	17.07	12.73

Tabla 1.8. Resultados conjunto Localization.

cr	Retención	T. secuencial	T. Paralelo
2r	0.2%	449.4	325
4r	0.2%	139.29	82.38
6r	0.2%	86.08	52.45
8r	0.2%	56.52	29.96
10r	0.2%	50.1	22.39

Tabla 6.9. Resultados conjunto Skin.

cr	Retención	T. secuencial	T. Paralelo
2r	15.6%	2727.31	1572.25
4r	15.6%	2156.03	1226.00
6r	15.6%	1377.72	768.28
8r	15.9%	1157.47	789.72
10r	15.6%	1020.18	669.95

Tabla 6.10. Resultados conjunto Covertype.

cr	Retención	T. secuencial	T. Paralelo
2r	4.7%	6227.15	4087.02
4r	4.7%	2800.58	1763.42
6r	4.7%	1765.63	1182.03
8r	4.7%	1342.64	858.00
10r	4.7%	1126.23	693.96

Tabla 6.11. Resultados conjunto Donation.

cr	Retención	T. secuencial	T. Paralelo
2r	4.7%	3665.46	2292.11
4r	2.3%	1279.9	537.25
6r	2.0%	931.84	521.16
8r	2.1%	646.35	338.29
10r	2.1%	571.7	273.56

Tabla 6.12. Resultados conjunto Pamap.

cr	Retención	T. secuencial	T. Paralelo
2r	0.1%	1197.62	755.1
4r	0.2%	727.19	399.95
6r	0.3%	484.73	252.05
8r	0.3 %	460.44	230.22
10r	0.3%	437.49	214.37

Tabla 6.13. Resultados conjunto KDDCup.

Medir de forma precisa los tiempos no es una tarea fácil, estos resultados pueden variar de una máquina a otra, de un sistema operativo a otro, aun teniendo el mismo hardware, incluso los tiempos varían al ejecutar en la misma máquina el mismo algoritmo, debido a factores como la carga de trabajo del CPU en determinado instante, la cantidad de memoria RAM disponible, la búsqueda de instrucciones, ejecución del control de flujo, operaciones de entrada/salida. Algo que es importante precisar es que la ejecución también puede variar según los datos que se procesan, la aritmética de datos punto flotante realizada por el procesador matemático o coprocesador tarda más que las operaciones aritmética de enteros o que las operaciones lógicas; Debido al uso de particiones en el conjunto de datos en muchos casos algunas de éstas puede quedar con datos más cómodos para su proceso y afectar el tiempo de ejecución.

De acuerdo a los tiempos reportados en [Olvera et al, 2010a], otros métodos de selección relevantes como los DROP, GCNN y POC-NN requieren tiempos prohibitivos para éstas bases, es decir, son inaplicables, ya que los tiempos de ejecución con los conjuntos más grandes de la ***tabla 6.1*** de la sección anterior, oscilaban entre 10,20 y 30 horas, estos tiempos serian muchos más grandes para las bases de datos de la ***tabla 6.3***.

Los tiempos de ejecución se miden con funciones integradas a la API de OpenMP, dichas funciones utilizan el concepto de *CPU time* y *Real time* que son los tiempos que la máquina utiliza exclusivamente para el programa y tiempo real que se ha tardado desde que inició su ejecución hasta que el programa finaliza y proporciona resultados, en la documentación de esta API menciona que ambas mediciones son susceptibles a variaciones imprevisibles.

En las primeras pruebas experimentales, se pudo comprobar que al aplicar PSC sobre la misma base de datos, el tiempo variaba en algunas ocasiones de forma considerable, sin embargo siempre mantienen la proporción con la ejecución de los códigos secuencias, para los cuales el *CPU time* solo difiere del *Real time* por máximo dos segundos cuando el uso de memoria no

está al máximo, lo cual no es problema para estas pruebas. Los tiempos que se muestran en todo el documento son del *Real time*, pues el objetivo principal es observar cómo se reduce el tiempo de ejecución en situaciones del mundo real.

En las tablas de resultados de arriba (***tabla 6.4*** a ***6.13***) podemos observar que los tiempos aumenta según los esperado, y disminuyen al aumentar el número de clusters en el conjunto, el menor tiempo de ejecución se da cuando ***C=10r***, el porcentaje de retención no varía de forma considerable para los distintos valores de ***c***, la mayor variación es de 2.6 % y la menor de 0%, de la misma forma este porcentaje es mínimo para la mayoría de conjuntos.

	i P53	i Bank	i Adult	i Shuttle	i Localization	I Skin	i covertype	i Donation	i PAMAP	i KDDCup
2r	0.55	0.81	0.59	0.48	0.77	0.72	0.58	0.66	0.63	0.63
4r	0.54	0.57	0.52	0.54	0.78	0.59	0.57	0.63	0.42	0.55
6r	0.75	0.49	0.46	0.42	0.69	0.61	0.56	0.67	0.56	0.52
8r	0.43	0.61	0.40	0.60	0.79	0.53	0.68	0.64	0.52	0.50
10	0.50	0.44	0.52	0.45	0.75	0.45	0.66	0.62	0.48	0.49
Media	***0.55***	***0.58***	***0.50***	***0.50***	***0.76***	***0.58***	***0.61***	***0.64***	***0.52***	***0.54***

Tabla 6.14. Índices de proporción de reducción de tiempos

Tabla 6.14. Índices de proporción de reducción de tiempos

Los datos de la ***tabla 6.14*** representan las proporciones resultado de dividir las razones de tiempo, el cociente i resultado de la división se le llamará índice de proporción. Este índice nos indica el porcentaje de tiempo que utiliza el código paralelo con respecto al código secuencial, entre más se acerque a cero ejecuta la tarea más rápido, y por lo tanto tiene mayor desempeño, de lo contrario si su tiempo de ejecución entre ambas versiones es similar, este índice tenderá a uno.

$$i = \frac{tiempo\ paralelo}{tiempo\ secuencial}$$

Debido a que el CPU utilizado es de dos núcleos físicos, el resultado esperado es la reducción de tiempos en 50%, esto es, un índice de proporción de 0.5, para el mejor de los casos, que es un código secuencial totalmente paralelizable, como en la práctica esto no puede suceder ya que la ley de *Amdahl* nos dice que el incremento en la velocidad de un programa de múltiples procesadores está limitada por sus fracciones de código secuencial, el índice de proporción que se espera es ≥ 0.50. El índice más pequeño que se puede encontrar en la tabla es 0.44, esto se debe a la variación de tiempos antes mencionada. Inicialmente cada prueba se repetiría diez veces, para obtener un promedio de los tiempos de ejecución, esta opción fue descartada pues solo para la obtención de precisión de los conjuntos originales se estimaba un tiempo de alrededor de seiscientas horas, tiempo que sobrepasa los límites planteados en la tesis, por tal, se considera que los resultados obtenidos en cada ejecución son los de un caso promedio, una situación típica de ejecución del algoritmo, a pesar de esta consideración se puede esperar que algunos de los tiempos tengan variaciones en cuanto al índice de proporción esperado, ya que al ejecutar en código nos podemos encontrar con el mejor o peor caso para el conjunto de entrenamiento, lo que hace variar los tiempos considerablemente. En la última fila de la tabla se observa la media de las razones de tiempo para cada base de datos, esta media cumple con el índice de proporción que se espera.

B.D.	Máximo	Mínimo	i
P53	50.67	35.23	0.69
Bank	14.75	4.2	0.28
Adult	13.02	4.61	0.35
Shuttle	9.42	1.75	0.18
Localization	65.51	17.07	0.26
Skin	449.40	50.1	0.11
Covertype	2727.31	1020.18	0.37
Donation	6227.15	1126.23	0.18
PAMAP	3665.46	571.7	0.15
KDDCup	1197.62	437.49	0.36

Tabla 6.15. Índices de reducción de algoritmo secuencial.

B.D.	Máximo	Mínimo	i
P53	28.09	17.61	0.62
Bank	12.02	1.86	0.15
Adult	7.69	2.41	0.31
Shuttle	9.42	1.75	0.18
Localization	4.54	0.79	0.17
Skin	50.71	12.73	0.25
Covertype	325	22.39	0.06
Donation	4087.02	693.96	0.16
PAMAP	2292.11	273.56	0.11
KDDCup	755.1	214.37	0.28

Tabla 6.16. Índices de reducción del algoritmo paralelo.

La ***tabla 6.15*** y ***6.16*** muestra las razones entre el tiempo mínimo y el máximo que tarda en ejecutarse el código, los cuales coinciden con ***C=2r*** y ***C=10r*** para ambas versiones. Estos índices de proporción de tiempo muestran una reducción de hasta el 94% como se puede observar en el conjunto *covertype* de la ***tabla 6.16***, de acuerdo con estos resultados podemos esperar que en las pruebas de precisión el tiempo que se reduce debido al paralelismo, se reduzca aún más al aumentar en número de clusters.

$$i = \frac{tiempo\ mínimo}{tiempo\ máximo}$$

6.5 Pruebas de precisión con PSC

Ahora nos resta saber en qué porcentaje es afectada la precisión de clasificación, cuando el conjunto de datos es reducido por este algoritmo de selección de instancias (PSC). El clasificador que utilizaremos es el mismo que se utiliza en las pruebas de precisión con el mismo valor del parámetro ***k***, de igual manera se utiliza la métrica de validación cruzada estratificada con 3 y 10 particiones de acorde al tamaño del conjunto. Como PSC se combina con un clasificador el proceso es el siguiente, una vez creadas los particiones y definido el conjunto de entrenamiento (training) y el conjunto de prueba (test) para cada una de estas, al entrenamiento se le aplica PSC, el sub conjunto *S* que da como resultado es ahora el nuevo conjunto de entrenamiento y es el que utilizara *k-NN* para obtener su precisión. La **figura 6.7** ilustra este proceso.

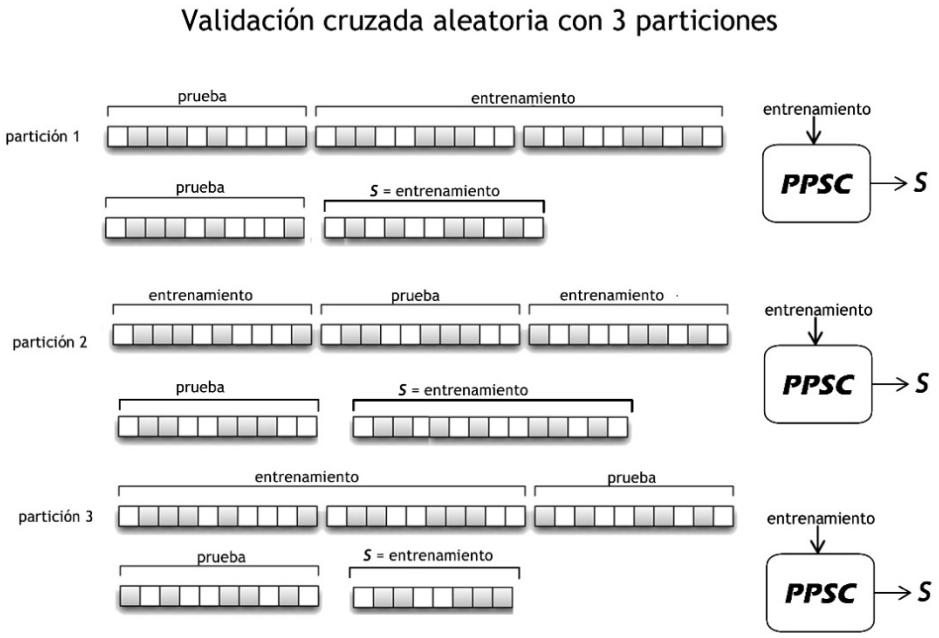


Fig. 6.7. Diagrama de PPSC con validación cruzada

En la siguiente tabla (***6.17***) se exponen los resultados de aplicar *k-NN* con *PSC* a las distintas BD, en la primera fila (Cr=2, 4, 6, 8,10) de la tabla se muestras el número de clusters con el que

PSC trabajó, las siguientes filas muestran la precisión de clasificación que arrojo *k-NN*. Para las BD *P53 mutants*, *PAMAP4*, *Donation* y *KDDCup* las particiones cambian de 10 a 3.

PRECISIÓN DE CLASIFICACIÓN	Cr=2	Cr=4	Cr=6	Cr=8	Cr=10
P53	84.6	86.6	85.6	89.3	92.3
Bank	59.3	59.4	59.5	59.4	59.5
Adult	56.2	56.2	56.2	56.2	56.2
Shuttle	99	97.5	97.7	98.7	98
Localization	29.6	29.8	30.5	31.2	33.1
Skin	87.9	92.8	93	96	96.6
Covertype	82.1	83.67	83.56	83.49	83.59
Donation	51.7	51.7	51.7	51.7	51.7
PAMAP	93.4	94.5	95.8	95.3	93.4
KDDCup	77.88	79.22	85.36	87.96	90.1

Tabla. 6.17. Tabla de precisión de clasificación de las bases de datos

La ***figura 6.8*** es la representación en forma de gráficas de los datos de la ***tabla 6.17***, los puntos en la gráfica correspondientes al porcentaje de clasificación correcta, éstos varían a medida que el ***Cr*** cambia.

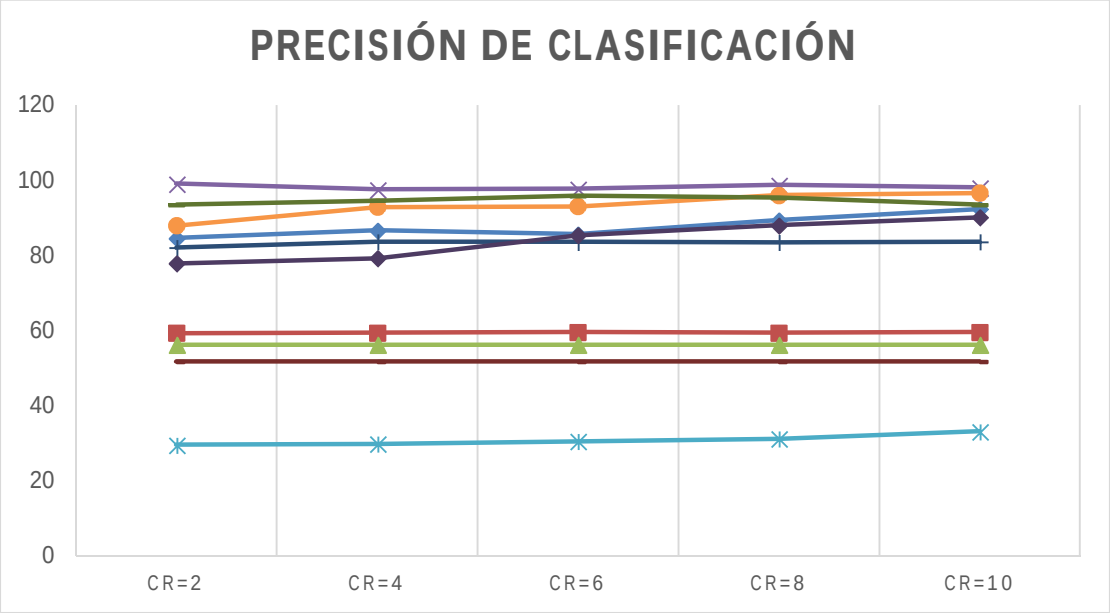


Fig. 6.8. Gráficas de precisión de la tabla 6.17

	i P53	i Bank	i Adult	i Shuttle	i Localization	i Skin	i covertype	i Donation	i PAMAP	i KDDCup
2r	0.78	0.61	0.66	0.57	0.56	0.46	0.56	0.65	0.60	0.60
4r	0.68	0.65	0.62	0.60	0.46	0.41	0.52	0.63	0.60	0.58
6r	0.67	0.67	0.62	0.52	0.52	0.43	0.54	0.61	0.66	0.56
8r	0.66	0.70	0.64	0.54	0.55	0.42	0.52	0.63	0.66	0.53
10r	0.66	0.71	0.66	0.49	0.50	0.42	0.51	0.65	0.68	0.53
Media	0.69	0.67	0.64	0.54	0.52	0.43	0.53	0.63	0.64	0.56

Tabla 6.18. Índices de proporción de reducción de tiempos

Para el caso de precisión de clasificación podemos observar que mejora a medida que el número de clusters con los que trabaja *PSC* se incrementa, su mejor precisión es para *10r*. La media en la ***tabla 6.18*** (ver apéndice A para detalles de tiempos) de reducción de tiempos entre la versión concurrente contra la versión paralela oscila entre 52 y 69 porciento esto quiere decir que a excepción del conjunto Skin (que es del 57 por ciento), hay una reduciendo tiempo de entre 31 y 48 por ciento, acorde a lo planteado originalmente.

B.D.	i secuencial	i paralelo
P53	0.64	0.54
Bank	0.32	0.37
Adult	0.36	0.36
Shuttle	0.19	0.16
Localization	0.22	0.20
Skin	0.19	0.11
Covertype	0.33	0.30
Donation	0.21	0.21
PAMAP	0.27	0.31
KDDCup	0.20	0.17

Tabla 6.19. Índices de reducción secuencia y paralelo.

Las razones entre el tiempo mínimo (Cr=10) y el máximo (Cr=2) que tarda en ejecutarse el código siguen un patrón de reducción similar (**tabla *6.19)***[12], la base de datos P53 siempre mantiene un índice más alto debido a la cantidad de atributos que contiene. En la ***tabla 6.20*** se muestra una tabla comparativa de los tiempos de clasificación con *k-NN*, *k-NN* con *PSC* secuencial y *k-NN PSC* paralelo, las dos últimas columnas muestras los índices de reducción, la diferencia en tiempos es significativa aun con el algoritmo secuencial, cabe señalar que la implementación de *k-NN* es una versión paralela, para la versión secuencial se debe esperar un incremento del doble de tiempo. La columna *i secuencial* mantiene una proporción aproximada del doble del tiempo que la última columna, esta proporción es correcta ya que se espera una reducción del 50 por ciento.

	Tiemplo Clasificación			***índice de reducción***	
Nombre	k-NN	PSC Secuencial	PSC Paralelo	i secuencial	i paralelo
PAMAP	48,769.74	2409.62	1587.11	0.05	0.03
Donation	51,989.73	10352.67	6726.96	0.20	0.13
Covertype	30,004.00	10900	5612	0.36	0.19
kddcup	31475.27	1716.86	909.93	0.02	0.01
Skin	4,693.96	1112.8	470.15	0.24	0.10
Localization	2,168.27	232.22	116.23	0.11	0.05
Shuttle	297.38	118.75	67.3	0.40	0.23
Adult	195.35	59.98	39.51	0.31	0.20
Bank	170.48	61.45	43.6	0.36	0.26
p53 Mutants	5270.87	97.18	64.3	0.02	0.01

Tabla 6.20. Tabla comparativa de tiempos

[12] Los datos de la tabla 6.17 y 6.18 están basados en las medidas de tiempo que se muestran en el apéndice A.

6.6 Pruebas de variación de parámetros

En algunas bases de datos la precisión es muy baja, es necesario saber si la precisión puede aumentar al variar el parámetro *k* de *k-NN*, para estas últimas pruebas tomaremos el *cr* con el que se obtuvo mejor clasificación en las pruebas anteriores, los resultados se muestran en las siguientes tablas

Acc. Org.	Acc. PSC	k
99.5	88	1
99.5	94	5
99.5	95.8	7
99.5	96.5	9
99.5	97.5	11

Tabla 6.21. Precisión de P53 variando el parámetro k

Acc. Org.	Acc. PSC	k
85.2	57.3	1
88.3	61	5
88.3	62.3	7
88.3	62.8	9
88.7	63.7	11

Tabla 6.22. Precisión de Bank variando el parámetro k

Acc. Org	Acc. PSC	k
86.6	55.7	1
91.3	58.4	5
91.7	59.8	7
91.9	61.5	9
91.9	62.5	11

Tabla 6.23. Precisión de Adult variando el parámetro k

Acc. Org	Acc. PSC	k
99.8	99.3	1
99.8	97.2	5
99.8	95.4	7
99.8	94.3	9
99.8	94.1	11

Tabla 6.24. Precisión de Shuttle variando el parámetro k

Acc. Org.	Acc PSC	k
99.6	76.6	1
99.6	42.5	5
99.6	28.7	7
99.6	20.1	9
99.6	21.1	11

Tabla 6.25. Precisión de Localization variando el parámetro k

Acc. Org.	Acc. PSC.	k
99.9	96.7	1
99.9	91.4	5
99.9	88.9	7
99.9	81.7	9
95.4	81.3	11

Tabla 6.26. Precisión de Skin variando el parámetro k

Acc. Org.	Acc. PSC	k
96.3	86.1	1
96.5	79.4	5
96.2	77.2	7
95.8	75.3	9
95.5	73.5	11

Tabla 6.27. Precisión de covertype variando el parámetro k

Acc. Org.	Acc. PSC	k
95.6	51.5	1
97	51.9	5
97.1	52.1	7
97.1	52.9	9
97.2	53.9	11

Tabla 6.28. Precisión de Donation variando el parámetro k

Acc. Org.	Acc. PSC	k
99.9	98.2	1
99.9	95.2	5
99.9	92.5	7
99.9	91.7	9
99.9	91	11

Tabla 6.29. Precisión de PAMAP variando el parámetro k

Acc. Org.	Acc. PSC.	k
99.9	86.7	1
99.9	92.31	5
99.9	93.23	7
99.9	94.17	9
99.9	94.7	11

Tabla 6.30. Precisión de KDCup variando el parámetro k

Se observa de las tablas que las precisión ***Acc Org*** no se altera en gran medida al variar en parámetro *k,* esta precisión es la del conjunto completo, *Acc PSC* por otra parte, varia y se puede observar que se obtiene mejor precisión en la mayoría de los conjuntos. La siguiente tabla muestra una comparación entre la precisión de los conjuntos originales y los sometidos a PSC con su mejor clasificación.

Nombre	Repetidos	***Retención %***	***Acc. Orig.***	***Acc. PSC***
PAMAP	110967	2	99.9	98.2
Donation	0	4.7	96.7	53.9
Covertype	0	13.6	97.04	86.1
kddcup	352542	0.3	99.8	94.7
Skin	193624	0.2	100	96.7
Localization	0	0.6	99.6	76.6
Shuttle	0	1.7	99.8	99.3
Adult	481	13.4	90.2	62.5
Bank	1	16.3	87.6	63.7
p53 Mutants	29094	0.8	99.5	97.5

Tabla 6.31. Comparación de precisión

Basados en la ***tabla 6.31*** podemos observar que al igual que en los conjuntos utilizados en la fase de prueba, la variación de la precisión no sigue una constante o un patrón, dicha variación puede deberse a diversas razones, con los detalles que sabemos de cada base de datos no es posible saber que tanto decaerá al aplicar PSC. La ***figura 6.9*** muestra las gráficas de tiempo de

clasificación, que tarda cada conjunto con los diferentes algoritmos. De acuerdo con la gráfica PSC reduce en gran medida el tiempo de clasificación en los conjuntos de gran tamaño.

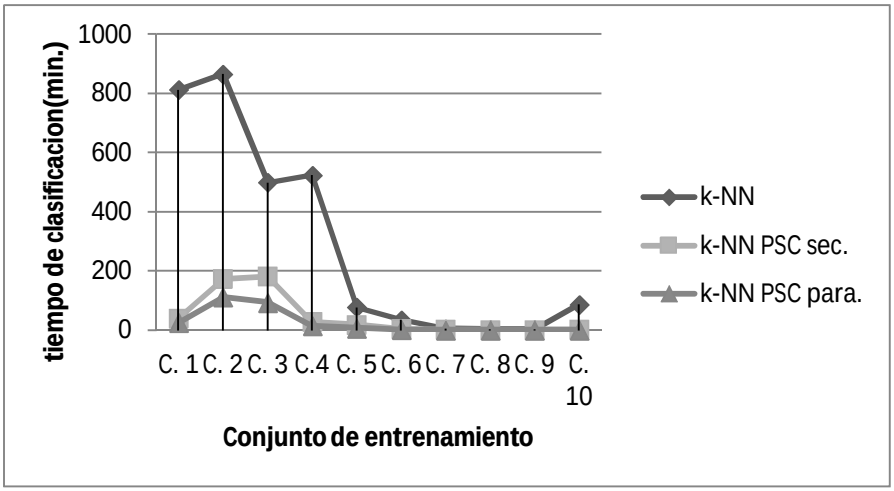


Fig. 6.9. Gráfica de tiempos de clasificación

La ***figura 6.10*** muestra la precisión de los conjuntos originales contra la precisión al usar PSC, esta gráfica esta creada a partir de los datos de la ***tabla 6.31***, es una alternativa para observar cómo varía la precisión para los diferentes conjuntos.

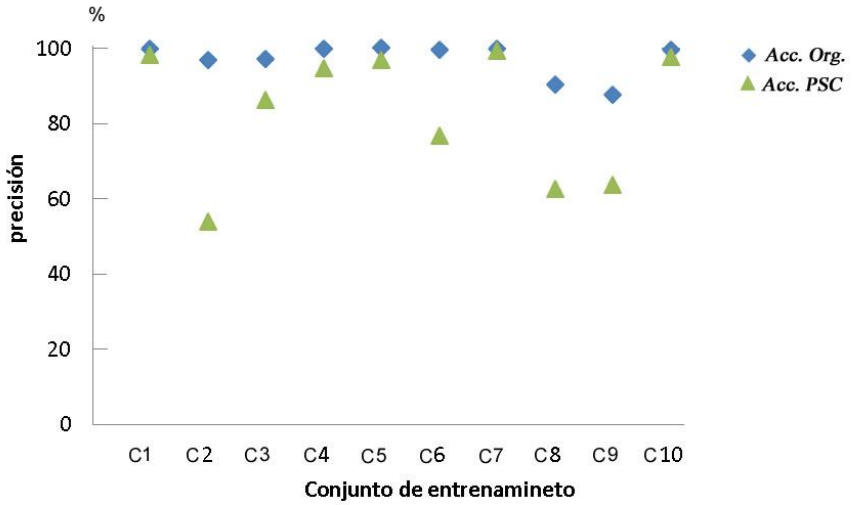


Fig. 6.10. Gráfica de precisión

CAPÍTULO 7

Conclusiones

En el campo del aprendizaje automático y problemas que requieran de clasificación automática de instancias, ya sea de investigación, desarrollo, o de forma didáctica en los cuales el tiempo de ejecución del clasificador al asignar la clase a nuevos instancias es prolongado, PPSC es una buena elección; De igual forma para grandes conjuntos en los que se desea eliminar instancias redundantes, ruidosos o simplemente se desea reducir el conjunto de datos.

De acuerdo con nuestros resultados para conjuntos grandes de datos en los cuales la selección de instancias es de mayor utilidad PSC reducen en gran porcentaje los tiempos de clasificación, usando PSC en forma paralela estos tiempos se pueden reducir aún más, de acuerdo al número de procesadores que se tengan disponibles, y los parámetros de PSC. La precisión de clasificación puede reducirse, pero si experimentamos con los parámetros del clasificador podemos llegar a una precisión bastante aceptable, por lo tanto, se puede concluir que para bases de datos con cientos de miles de objetos, el PPSC es la mejor opción para la selección de instancias y reducción de datos.

Como se demostró en los experimentos, el paralelismo redujo en gran medida los tiempos de ejecución, en los datos de la ***tabla 6.16*** se puede observar que se reduce desde un 60 a hasta un 98 por ciento si comparamos el tiempo que tarda en clasificar k-NN contra el tiempo que tarda el mismo clasificador pero aplicando PSC a los subconjuntos de entrenamiento, este porcentaje varía y no es posible encontrar una proporción, lo que sí es notable es que entre más grande es el conjunto y con mayor número de atributos es mayor el porcentaje de reducción. Al comparar los tiempos de clasificación utilizando k-NN con PSC y PPSC, la reducción se mantiene entre un 41% hasta un 72 %, para el peor caso, al sacar la media de estos datos el resultado concuerda con lo esperado, ya que se mantiene en .55, lo que nos indica que el versión paralela tarda la mitad de tiempo que la secuencial, debido al uso de un segundo procesador que absorbe en promedio la mitad de la carga de trabajo al clasificar.

Para mejorar el algoritmo en un futuro se puede agregar más funciones de distancia para poder clasificar conjuntos mixtos, de la misma forma, podemos experimentar con los parámetros C del algoritmo C-medias para obtener clusters mejor distribuidos con el fin de una mejor retención de instancias fronterizas e instancias interiores que nos llevarán a regiones con discriminación de clase más marcadas y en consecuencia a una mejor precisión de clasificación. También es conveniente aplicar las pruebas en varias máquinas con distinto número de procesadores para obtener una medida de cómo disminuye el tiempo de acuerdo al incremento de procesadores.

Referencias

- Alpaydın Ethem (2010), The MIT Press, segunda edición, Cambridge, Massachusetts, London, England.
- Bastida Ibáñez Javier, Introducción a la Arquitectura de Computadores, Universidad de Valladolid, 1995.
- Bayes T. (1763). An essay towards solving a problem in the doctrine of chances. Philosop-hical Transactions, 53:370-418, 1763.
- Bezdek J.C., Kuncheva LI (2001) Nearest prototype classifier designs: an experimental study. Int J Intell Syst 16(12):1445–1473.
- Bishop Christopher M.(2006), Pattern Recognition and Machine Learning, Springer Science+Business Media, LLC
- Blum AL, Langley P (1997) Selection of relevant features and examples in machine learning. Artif Intell 97:245–271.
- Brighton H, Mellish C (2002) Advances in instance selection for instance-based learning algorithms. Data Min Knowl Discov 6(2):153–172.
- Cerverón V, Ferri FJ (2001) Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbour rule. IEEE Trans Syst Man Cybern B 31(3):408–413.
- Chandra Rohit, Dagum Leonardo, Kohr Dave, Maydan Dror, McDonald Jeff, and Menon Ramesh (2001); “Parallel Programming in OpenMP”, Academic Press, San Diego, California.
- Chapman Barbara, Jost Gabriele, van der Pas Ruud(2008) ,“Using OpenMP: Portable Shared Memory Parallel Programming”,. The MIT Press Cambridge, Massachusetts Institute of Technology, Massachusetts, London, Inglaterra, pág. 8-21
- Contijon Bon J. F.(2001), Técnicas no supervisadas: métodos de agrupamiento, Notas.
- Devijver PA, Kittler J (1980). “On the edited nearest neighbor rule”. In: Proceedings of the fifth international conference on pattern recognition, Los Alamitos, CA, pp 72–80.d
- Dietterich T.(1995), An experimental comparison of nearest neighbor and nearest hype-rrectangle algorithms. Machine Learning, 19(1):5-28.
- Dr. Guillen Pablo(2004). “Introducción a la computación paralela”, trasparencias, Second Workshop: New Frontiers of Bioinformatics in Latin America- Gridding Biology. Centro de Simulación y Modelos, Noviembre 2004.
- García Miguel, Melián Belén, Moreno José A., J Moreno Vega. Marcos, Rivero Raquel(2004), “Búsqueda Dispersa en Selección de Instancias”, Departamento de Estadística, I.O. y Computación, Escuela Técnica Superior de Ingeniería Informática, Universidad de La Laguna. Agosto 2004.pag 1-3.

- García S, Cano JR, Herera F (2008) A memetic algorithm approach. Pattern Recognit 41:2693–2709
- Glover F (1986) The general employee scheduling problem: an integration of management science and artificial intelligence. Comput Oper Res 13(4):563–593.
- Grubbs, F.E., (1969), Procedimientos para la detección de observaciones atípicas en las muestras. Technometrics 11, 1-21.
- Hart P.E. (1968) The condensed nearest neighbor rule. IEEE Trans Inf Theory 14:515–516
- Jain A. K., W. Duin R. P., and Mao J.(2000), Statistical pattern recognition: a review. IEEE transactions on Pattern Analysis and Machine Intelligence, 22(1), pp. 4-37.
- Ke-Ping Z, Shui-Geng Z, Ji-HongG,Ao-YingA(2003)C-Pruner:An improved instance pruning algorithm. In: Proceedings of 2nd IEEE international conference on machine learning and cybernetics, vol 1. pp 94–99.
- Kittisak Kerdprasop and Nittaya Kerdprasop, “A lightweight method to parallel k-means”, international journal of mathematics and computers in simulation, Issue 4, Volume 4, 2010.
- Kuncheva LI (1995) Editing for the k-nearest neighbors rule by a genetic algorithm. Pattern Recognit Lett 16:809–814
- Kuncheva LI (1997) Fitness functions in editing k-NN referent set by genetic algorithms. Pattern Recognit 30:1041–1049
- Kuncheva LI, Bezdek JC (1998) Nearest prototype classification, clustering, genetic algorithms, or random search? IEEE Trans Syst Man Cybern C 28(1):160–164
- Lumini A, Nanni L (2006) A clustering method for automatic biometric template selection. Pattern Recognit 39:495–497
- Luna Rene, Villa Luis, Ramírez Marco (2010). El supercómputo en 2010. Centro de Investigación en Computación del Instituto Politécnico Nacional, Revista digital Universitaria, Volumen 11, Número 3 ,ISSN: 1067-6079.
- Mitchell T.(1997).Machine Learning. McGraw-Hil.
- Mitchell Tom M. (2006), The Discipline of Machine Learning, School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213.
- Mollineda RA, Ferri FJ, Vidal E (2002). An efficient prototype merging strategy for the condensed 1through class-conditional hierarchical clustering. Pattern Recognit 35:2771–2782
- MuseranoRonal (2008),“Aplicaciones Single Program Multiple Data (SPMD) en Ambientes Distribuidos”, Tesis de Grado , Departamento Arquitectura de Computadores y Sistemas Operativos ,Escuela Técnica Superior de Ingeniería (ETSE), Universidad Autónoma de Barcelona,Barcelona , España, pág. 8-20.
- Olmedo Gonzales Enrique(2009), “Programación y monitores de aplicaciones paralelas con Charm ++”.Tesis de Grado, Capitulo3. Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla.

- Olvera-López JA, Carrasco-Ochoa JA, Martínez-Trinidad JF (2005). Sequential search for decremental edition. In: Gallagher M, Hogan J, Maire F (eds) LNCS 3578: IDEAL 2005. Queensland, Australia, pp 280–285.
- Olvera-López JA, Martínez-Trinidad JF, Carrasco-Ochoa JA (2007a). Restricted sequential floating search applied to object selection. In: Perner P (ed) MLDM 2007:LNAI 4571. Leipzig, Germany, pp 694–702.
- Olvera-López JA, Carrasco-Ochoa JA, and Martínez-Trinidad JF (2007b). Object selection based on clustering and border objects. In: Kurzynski M et al (eds) Computer recognition systems 2, ASC 45. Wroclaw, Poland, pp. 27–34.
- Olvera-López J.A., Carrasco-Ochoa J.A., Martínez-Trinidad J.F. (2008). "Prototype selection via prototype relevance". In:Ruiz- Shulcloper J,KropatschWG(eds)CIARP 2008, LNCS 5197.Habana,Cuba, pp. 153–160.
- Olvera-López J. A., Carrasco-Ochoa J. A., and Martínez-Trinidad (2010a). "A new Fast Prototype Selection Method based on Clustering. Pattern Analysis and Applications", Pattern Analysis and Applications, 131-141 13(2), pp. 131-141.
- Olvera-López J. A., Carrasco-Ochoa J. A., Martínez-Trinidad J. F. , Kittler Josef (2010b)." A review of instance selection methods". Artificial Intelligence Review, 13 pp 133-143.
- Paredes R, Vidal E (2000)Weighting prototypes. A new editing approach. In: Proceedings of the international conference on pattern recognition ICPR, vol. 2. pp 25–28
- Pardo Carpio Fernando(2002), Arquitecturas Avanzadas, Universida De Valencia, Ingenieria En Informatica
- Pérez Alcaraz Carlos(2011), "Análisis, desarrollo y optimización de rutinas paralelas de cálculo de funciones de Green", Tesis de grado, Departamento de Informática y Sistemas, Facultad de informática, Universidad de Murcia, España, pág. 11-12.
- Pudil P, Ferri FJ, Novovicová J, Kittler J (1994) Floating search methods for feature selection with nonmonotonic criterion functions. In: Proceedings of the 12th international conference on pattern recognition.IEEE Computer Society Press, pp 279–283.
- Riquelme J.C. ,Aguilar-Ruíz JS,ToroM (2003) Finding representative patternswith ordered projections. Pattern Recognit 36:1009–1018.
- Ruiz Sánchez D. Roberto(2005), "Selección de Atributos mediante proyecciones", Departamento de Lenguajes y Sistemas Informáticos, Tesis de Grado, Universidad de Sevilla, agosto 2005, Sevilla, España.
- Salcedo Haro Miriam(2007), "Diseño, análisis e implementación de un algoritmo paralelo para la resolución de problemas de programación lineal en enteros puros", Tesis de Grado, Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla, Enero 2007. Capítulo 3.
- Sitio WEB de OpenMP [http://openmp.org]

- Srisawat A, Phienthrakul T, Kijsirikul B (2006) SV-kNNC: an algorithm for improving the efficency of k-Nearest neighbr. In: Yang Q, Webb G (eds) PRICAI 2006:LNAI 4099. Guilin, China, pp 975–979.
- Soto E. J. Antonio (2005), Mejorar e las probabilidades de pertenecía en clasificación difusa con aplicaciones al cálculo preciso de orbitas, Tesis de Grado, Departamento de matemáticas, Universidad de Alicante, Alicante, España.
- K. Theodoridos K.,Koutroubas S.(1999),Pattern recongnition, Academics Press
- Vapnik V (1995) The nature of statistical learning theory. Springer, New York.
- Vázquez Mesa Fernando Daniel(2008), Algoritmos de Aprendizaje Continuo Mediante Selección de Prototipos para Clasificadores Basados en Distancias, Tesis de grado, Departament de Llenguatges I Sistemas Informátics , Universitat Jaume I, Castellón de la Plana , España.
- Venmann CJ, ReindersMJT, Backer E (2002) A maximum variance clustering algorithm. IEEE Trans Pattern Anal Mach Intell 24(9):1273–1280.
- Wilson D.L. (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cybern 2:408–421
- Wilson D.R., Martınez T.R. (2000) Reduction techniques for instance-based learning algorithms. Machine Learning, 38:257–286
- Yuangui L, Zhonhui H, Yunze C, Weidong Z (2005) Support vector based prototype selection method for nearest neighbor rules. In: Wang L et al (eds) ICNC 2005, LNCS 3610. Changsha, China, pp 528–535.

Apéndice A

Formato TRN

Un archivo TRN (extensión .trn) es un formato de archivo en texto ASCII similar a los archivos ARFF que utiliza Weka. La primera sección, llamada cabecera, contiene la información que describe a una base de datos o conjuntos de entrenamiento, en ella se almacena el número de objetos, atributos, clases y tipo (numérico, categórico) de cada uno de ellos. La última sección es la que contiene los datos del conjunto de entrenamiento.

```
1 150:4
2 @
3 0,0,0,0,1
4 0,0,0,0,3
5 @
```

Fig. 1 Cabeceras de la base de datos iris

Arriba en la **figura 1** se muestra la cabecera de la BD iris, de arriba hacia debajo, de izquierda a derecha podemos encontrar en número de objetos, el número de atributos, en el siguiente reglón un delimitador obligatorio, en la línea tres se almacenan el tipo de dato de cada atributo, en este caso todos los atributos son de tipo numérico, los cuales se representa con el número cero, para los datos categóricos, se debe de representar con el número uno, por último indicamos que terminó la sección con el carácter @.

```
 5 @
 6 5.1,3.5,1.4,0.2,0
 7 4.9,3.0,1.4,0.2,0
 8 4.7,3.2,1.3,0.2,0
 9 4.6,3.1,1.5,0.2,0
10 5.0,3.6,1.4,0.2,0
11 5.4,3.9,1.7,0.4,0
12 4.6,3.4,1.4,0.3,0
13 5.0,3.4,1.5,0.2,0
14 4.4,2.9,1.4,0.2,0
15 4.9,3.1,1.5,0.1,0
```

Fig. 2 Sección de datos del conjunto iris

El caracter @ también indica el comienzo de la parte de datos en el archivo, cada renglón representa un objeto y cada columna del reglón un atributo del objeto, la última columna siempre representara la clase a la que pertenece el objeto, para el fragmento del conjunto iris que se muestra en la **figura 2**, podemos comprobar lo que nos dice el archivo de cabecera, tres atributos numéricos por objeto y una clase de tipo categórico, el delimitador puede ser cualquier carácter que no sea ambiguo con la base de datos.

Funciones de Distancia

Una vez determinado el tipo de instancias y objetos a clasificar en un problema de *RP*, establecer una medida de proximidad o de distancias adecuada, que cuantifique el grado de similaridad entre cada objeto, en función a lo diferentes que resulten ser cada uno de los atributos, es relevante, puesto que, esto permite clasificar adecuadamente a los objetos.

Distancias

Las medidas **de disimilitud**, **desemejanza** o **distancia** miden la distancia entre dos objetos de forma que, cuanto mayor sea su valor, más diferentes son éstos.

$$\boldsymbol{Definición}: Sea\ U\ \ un\ conjunto\ de\ elementos\ .Una\ función\ d: U\ x\ U \rightarrow \Re$$

se llama una distancia métrica si $\forall x, y \in U$ si tiene :

1. $d(x,y) \geq 0$ 2. $d(x,y) = 0 \leftrightarrow x = y$ 3. $d(x,y) = d(y,x)$

4. $d(x,z) \leq 0 d(x,y) + d(y,z)$, $\forall z \in U$

Disimilaridad

Las **medidas de proximidad, similitud o semejanza** miden el grado de semejanza entre dos objetos de forma que, cuanto mayor es su valor, mayor es el grado de similaridad existente entre ellos. De forma análoga tenemos una definición formal para desimilariad.

Definición:

Sea U *un conjunto finito o infinito de elementos. Una función* $s: U x\ U \rightarrow \Re$ *se llama similaridad si cumple las siguientes propiedades* $\forall x, y \in U$.

1. $s(x, y) \leq s_0$
2. $s(x, y) = s_0$
3. $s(x, y) = s(y, x)$

Donde s_0 es un numero reál finito arbitrario

En la literatura existen multitud de medidas de semejanza y de distancia dependiendo del tipo de variables y datos considerados. A continuación se mencionan algunas de las más utilizadas.[13]

- Distancia de edición o Levenshtein
- Distancia de brecha
- Distancias euclídea
- Distancias de Minkowski
- Distancias de Mahalanobis
- Distancia con N-Gramas
- Distancia de Hamming
- Distancia de Manhattan
- Similitud Smith-Waterman
- Similitud de Jaro
- Medida de Russell y Rao
- Medida de parejas simples

[13] Para otros ejemplos ver Anderberg (1973)

- Medida de Jaccard
- Medida de Dice
- Medida de Rogers-Tanimoto
- Medida de Kulcynski
- Coeficiente de Bray-Curtis

Validación Cruzada

Estimar la exactitud de un clasificador es importante no solo para predecir su futuro comportamiento, sino también para poder escoger un clasificador dentro de un conjunto de posibilidades. Existen diferentes métodos de validación; Matriz de confusión, curvas ROC (Receiver Operation Characteristic) y validación cruzada por mencionar algunos.

La validación cruzada es un método estadístico para evaluar y comprar algoritmos de aprendizaje automático. Este método consiste en dividir en dos segmentos el conjunto de datos, uno se usa para el aprendizaje (traning) y el restante se usa para validar el conjuto (test).

En general se crean ***k*** particiones aleatorias de igual tamaño, ***k-1*** particiones se utilizan para entrenar el modelo y la restante como prueba, el clasificador es entrenado y validado ***k*** veces. Si además en los dos conjuntos, el de entrenamiento y el de prueba, se mantiene la distribución de clases del conjunto original, la validación cruzada se dice que es estratificada. Lo estándar es que esta validación se repita 10 veces y se promedien los resultados. A este diseño experimental se le denomina *stratified k fold cross-validation (validación cruzada estratificada con k particiones)*. En ocasiones, cuando los conjuntos de datos son demasiado grandes dividir dichos conjuntos en 10 subgrupos no es viable, cuando esto sucede se reduce a 2 o 3 particiones, si ocurre lo contrario, es decir, que el conjunto de datos es pequeño, lo más recomendables es que las particiones sean tantas como lo permita el tamaño del conjunto.

Tablas de tiempo

El siguiente conjuntos de tablas muestran a detalle las pruebas realizadas en el capítulo 6, en la sección *Pruebas de precisión de PSC*, la información que muestran las tablas 6.15,6.16 y 6.17 es un resumen de estos datos.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	84.6	150.91	118.01
4	86.6	112.79	76.36
6	85.6	104.8	70.3
8	89.3	91.01	60.18
10	92.3	97.18	64.3

Tabla 1. Resultados de las pruebas de precisión al conjunto P53.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	59.3	191.62	117.24
4	59.4	109.55	71.54
6	59.5	83.44	55.78
8	59.4	69.5	48.84
10	59.5	61.45	43.6

Tabla 2. Resultados de las pruebas de precisión al conjunto Bank.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	56.2	163.17	107.94
4	56.2	100.27	61.93
6	56.2	79.43	49.11
8	56.2	66.46	42.78
10	56.2	59.98	39.51

Tabla 3. Resultados de las pruebas de precisión al conjunto Adult.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	99.0	118.75	67.3
4	97.5	52.53	31.57
6	97.7	34.03	17.56
8	98.7	28.64	15.44
10	98	22.6	11.18

Tabla 4. Resultados de las pruebas de precisión al conjunto Shuttle.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	29.6	1035.91	581.11
4	29.8	497.16	227.57
6	30.5	372.12	195.29
8	31.2	273.21	150.86
10	33.1	232.22	116.23

Tabla 5. Resultados de las pruebas de precisión al conjunto Localization.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	87.9	5633.87	4220.272
4	92.8	2630.45	1220.04
6	93.0	1864.91	758.21
8	96.0	1504.56	653.5
10	96.6	1112.8	470.15

Tabla **6**. Resultados de las pruebas de precisión al conjunto Skin.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	82.1	32940.63	18385.42
4	83.67	20052.9	10512
6	83.56	17189	9305
8	83.49	13800	7210.68
10	83.59	10900	5612

Tabla 7. Resultados de las pruebas de precisión al conjunto Covertype.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	51.7	47308.51	30912.37
4	51.7	36365.23	22932
6	51.7	14797.52	9002.5
8	51.7	11340.25	7092.5
10	51.7	10352.67	6726.96

Tabla 8. Resultados de las pruebas de precisión al conjunto Donation.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	93.4	7306.34	4354.42
4	94.5	2602.18	1560.23
6	95.8	2409.62	1587.11
8	95.3	2204.97	1464.39
10	93.4	1993.23	1354.56

Tabla 9. Resultados de las pruebas de precisión al conjunto PAMAP.

cr	Acc. PSC	T. Secuencial	T. Paralelo
2	77.88	8538.68	5123.2
4	79.22	6489.36	3763.83
6	85.36	4563.16	2555.36
8	87.96	2856.78	1514.09
10	90.10	1716.86	909.93

Tabla 10. Resultados de las pruebas de precisión al conjunto KDDCup.

Apéndice B

OpenMP

Introducción

La implementación del algoritmo de este trabajo de tesis se desarrolla en el lenguaje de programación C con directivas de paralelismo de OpenMP, por ello en esta sección se describe esta API de manera general.

OpenMP es una interfaz de programación de aplicaciones de memoria compartida (API), es compatible con plataformas para programación paralela como C/C++ y Fortran, al igual que con diversas arquitecturas incluyendo UNIX y Windows NT. OpenMP es un modelo portátil y escalable con un interfaz simple y flexible para los desarrolladores de aplicaciones paralelas [openmp, 1997].

Origen de OpenMP

Cuando las empresas tuvieron la suficiente tecnología para construir las SMP (*Symmetric Multi-Processing*) a precios moderados, el siguiente paso fue enfocarse en el aprovechamiento de esta potencia de cálculo en las aplicaciones individuales, el inconveniente principal en esta etapa se reducía a los compiladores, que, aunque siempre han sido los responsables de la adaptación de código para hacer el mejor uso de los recursos de un computadora, por desgracia, aún es muy difícil para ellos hacerlo en una máquina con más de un procesador o núcleo, debido a que no son capaces de saber cuándo un programa debe dividirse y tampoco saber identificar las instrucciones independientes que se pueden ejecutar en paralelo , por tanto estos compiladores ante la ausencia de información a priori necesitan ayuda del usuario.

A final de la década de 1980 los científicos se dieron a la tarea de entender como el programador puede expresar el paralelismo en un código, enfocándose en las regiones en las cuales la información puede ser compartida entre procesos, encontraron que en la mayoría de los programas el código compartido podía guardarse en memoria principal y de ahí otro proceso podía recuperar información según sea necesario , sin embargo el problema era que al

funcionar independientemente uno del otro sus relojes de estos procesadores no estaban sincronizados y podían ejecutar sus porciones de código al mismo tiempo, por ellos los programadores establecieron una notación especial para identificar cuando el programa iba a repartir el trabajo entre los diferentes procesadores así como para establecer el orden en que estos iba a tener acceso a la información compartida. La notación tomó principalmente la forma de instrucciones especiales, o directrices, que podrían añadirse a los programas escritos en lenguajes secuenciales, especialmente Fortran. El compilador utiliza esta información para crear el código real para su ejecución por cada procesador. Aunque esta estrategia funcionó, tenía la deficiencia evidente que un programa escrito para una SMP no necesariamente se ejecutan todas las plataformas y arquitecturas [Chapman et al, 2008].

A finales de 1980 un grupo llamado PCF (Parallel Computing Forum) tomaron un acuerdo para definir un conjunto de directivas para especificar el paralelismo en bucle en los programas de Fortran, esta trabajo fue publicado en 1991; posteriormente X3H5 un subcomité oficial ANSI creo un estándar basado en PCF.

OpenMP fue definida por el OpenMP Architecture Review Board (ARB), un grupo de empresas que se unieron durante la segunda mitad de la década de 1990 para proporcionar un medio común para la programación de una amplia gama de arquitecturas SMP. OpenMP se basa en el trabajo anterior PCF.

La primera versión de OpenMP fue presentada en público en 1997 aunque las directivas y compiladores solo era para Fortran poco tiempo después se ampliaron y se introdujeron para C/C++, hoy en día las principales empresas, varios laboratorios de gobierno y grupos de investigación pertenecen a ARB. Una de las mayores ventajas de OpenMP es que el ARB continúa trabajando para asegurarse de OpenMP evolucione junto con la tecnología informática.

¿Qué es OpenMP?

OpenMP es una API cuyas características se basan en los esfuerzos previos para facilitar la memoria compartida en programación paralela, en lugar de un estándar aprobado oficialmente, se trata entre un acuerdo alcanzado entre los miembros de la ARM, que comparte el mismo

interés por el enfoque portátil, fácil de usar y eficaz para la programación paralela de memoria compartida [Chapman et al 2008].

OpenMP es adecuado para una amplia gama de arquitecturas como son los equipos multinúcleo y multithreading portátiles, de escritorio y de aplicaciones específicas que existen en el mercado, pero también puede ser usado en las computadoras con un solo procesador.

OpenMP no es un nuevo lenguaje de programación, sino un conjunto de directivas que se pueden agregar a un programa en Fortran y C/C++ permitiendo compartir la carga de trabajo entre los diferentes núcleos mediante el acceso a datos compartidos, la adecuada inserción de esta notación de OpenMP en un programa secuencias , permite que las aplicación, en su mayoría, pueda beneficiarse de esta forma paralela de memoria compartida con una mínima modificación en el código, ya que en la práctica muchas aplicaciones tienen un paralelismo considerable que puede ser explotado.

Se puede atribuir el éxito de OpenMP a diversos factores, como son la fuerte énfasis en la programación paralela estructurada , que OpenMP es comparativamente fácil de usar ya que los detalles de las distribución de trabajo corren a cargo del compilador , que se puede ejecutar en muchas plataformas diferentes , pero más que nada OpenMP es oportuna ya que el fuerte crecimiento de SMP (Multi-procesamiento simétrico, también llamada UMA, de *Uniform Memory Access*) pequeños y grandes exige la necesidad de un lenguaje de memoria compartida estándar que sea fácil de aprender. [Chapman et al 2008].

El principal problema de estas arquitecturas se encuentra en su escalabilidad, ya que conforme aumenta el número de procesadores el sistema de memoria se convierte en un cuello de botella si todos tratan de acceder al mismo tiempo.

Fueron varias las empresas que desarrollaron conjuntamente OpenMP y lo propusieron como un estándar para la industria de la computación, entre ellas Intel Corporation, IBM, Silicon Graphics Inc., etc. Además, otras empresas tanto de hardware como de software lo aprobaron y apoyaron, y actualmente puede considerarse el estándar para el desarrollo bajo el modelo de memoria compartida.

El ARB está compuesto por miembros permanentes y auxiliares. Los miembros permanentes son los vendedores que tienen un interés a largo plazo en la creación de productos para OpenMP. Miembros del Cuerpo Auxiliar son organizaciones con interés en la norma, pero que no crean o venden productos de OpenMP.

En particular, los miembros permanentes de ARB son:

- AMD (Dibyendu Das)
- CAPS-Entreprise (Francois Bodin)
- Convey Computer (John Leidel)
- Cray (James Beyer)
- Fujitsu (Matthijs van Waveren)
- HP (Uriel Schafer)
- IBM (Kelvin Li)
- Intel (Jay Hoeflinger)
- Microsoft (Niklas Gustafsson)
- NEC (Kazuhiro Kusano)
- NVIDIA (Yuan Lin)
- Oracle Corporation (Nawal Copty)
- The Portland Group, Inc. (Michael Wolfe)
- Texas Instruments (Andy Fritsch)

Programación en OpenMP

Para los sistemas de **memoria compartida**, de tipo SMP, la herramienta más extendida es **OpenMP**. Para los sistemas de memoria distribuida, el estándar actual de programación, mediante paso de mensajes, es MPI.

La creación de código en OpenMP es muy sencilla, debido a que **OpenMP** no se trata de un lenguaje de programación si no de una API, formada por:

- Directivas para el compilador
- Funciones de biblioteca
- Variables de entorno

OpenMP trabaja con Fortran y C/C++, sin embargo en este capítulo solo se hablara sobre la implementación de código en C.

Directivas

Para controlar las áreas en las cuales el cómputo se hará paralelo, OpenMP disponen de un elemento llamado directiva, el uso de estas **directivas** facilita la portabilidad y la paralización incremental. Dichas directivas siempre inician con $\#\,pragma\ omp$ seguida de la directiva, si se declara más de una directiva estas serán separadas por comas, la **figura 1** muestra la forma general de una directiva, cabe mencionar que el compilador es sensible a mayúsculas y minúsculas.

#pragma omp *directive-name [clause[[,] clause]...] new-line*

Fig. 1. Forma General de una directiva para C/C++[Chapman et al , 2008] .

En OpenMP la directiva más importante es el constructor paralelo (parallel construct) ,este constructor se usa para especificar que el computo se debe ejecutar en paralelo (ver **figura 2**),si no se encuentra esta directiva el código del programa se ejecutara en forma serial, cuando el hilo principal encuentra este constructor, crea un grupo de threads y lo asocia a dicha región, sin embargo, aunque esta directiva asegura que los cálculos se realizará en paralelo , la misma no distribuye el trabajo entre los threads de equipo. En el extremo de esta región que está delimitada por *llaves* ('{' '}') existe una barrera explicita que obliga a todos los threads a esperar a que el trabajo de la región haya sido completado, al final sólo el thread principal seguirá con la ejecución del programa.

#pragma omp parallel *[clause[[,] clause]...]*
structured block

Fig. 2. Directiva **parallel** [Chapman et al 2008].

Cuando el thread principal encuentra al constructor paralelo, por default se crean $k-1$ threads, siendo k el número de procesadores disponibles en la máquina, cada thread creado ejecutará el mismo código dentro de la región paralela, mientras el programador no indique lo contrario.

El número de *threads* que se generan para ejecutar una región paralela se controla mediante funciones, cada thread tiene un identificador, el identificador 0 (cero) es para el thread principal, en la tabla se listan algunas de estas funciones.

Función	Descripción
$\boldsymbol{void\ omp_set_num_threads(int\ num)}$;	Le indica al compilador cuántos procesadores se usarán para ejecutar el código de la región paralela. ***num*** no puede ser mayor a la cantidad de procesadores disponible.
$\boldsymbol{int\ omp_get_thread_num}()$;	Esta función regresa el identificador de cada thread, para el thread principal el identificador será 0.
$\boldsymbol{int\ omp_get_num_threads}()$;	Devuelve en número de thread que está en ejecución.

Tabla 1. Funciones para control de thread en OpenMP, estas funciones se encuentra definidas en el archivo de cabecera <omp.h>

El *thread* máster tiene como contexto el conjunto de variables del programa, y existe a lo largo de toda la ejecución del programa, al crearse nuevos *threads*, cada uno incluye su propio contexto, con su propia pila, utilizada como *stack frame* para las rutinas invocadas por el *thread*. La compartición de variables es el punto clave en un sistema paralelo de memoria compartida, por lo que es necesario controlar correctamente el **ámbito** de cada variable.

Las variables **globales** son compartidas por todos los *threads*. Sin embargo, algunas variables deberán ser propias de cada *thread*, **privadas**. Para poder especificar adecuadamente el ámbito de validez de cada variable, se añaden una serie de **cláusulas** a la directiva **parallel**, en las que se indica el carácter de las variables que se utilizan en dicha región paralela.

private *(list)*	
firstprivate *(list)*	
lastprivate *(list)*	
reduction *(operator:list)*	(C/C++)
reduction *({operator | intrinsic_procedure_name}:list)*	(Fortran)
ordered	
schedule *(kind[,chunk_size)]*	
nowait	

Fig. 3. Clausulas para la directiva **parallel**[Chapman et al 2008]

En la **figura 3** muestra las cláusulas que puede utilizar la directiva **parallel,** la forma de utilizarlas es la siguiente:

- ***shared(X)***

  Se declara la variable ***X*** como compartida por todos los *threads*, sólo existe una copia, y todos los *threads* acceden y modifican dicha copia. Por default todas las variables son de tipo ***shared***, Su valor cambia si fueron manipuladas dentro de la zona paralela.

- ***private(Y)***

  Se declara la variable **Y** como privada en cada *thread*. Se crean ***k*** copias, una por *thread* (sin inicializar). Se destruyen al finalizar la ejecución de los *threads*, al salir de la zona paralela mantienen el mismo valor que tenían antes de entrar en ella, para las variables privada se declaran objetos completos, no se puede declarar un elemento de un array como compartido y el resto como privado. Cada *thread* utiliza su propia pila, por lo que las variables declaradas en la propia región paralela (o en una rutina) son privadas[**Chandra et al 2001**].

- **firstprivate( )**

  Las variables privadas no están inicializadas al comienzo, ni dejan rastro al final, el valor con el que inician en la área paralela está definida por la localidad de memoria que el compilador asigne y al salir de esta área la variable conserva al valor que tenía, antes de

pasar por esta área, para poder pasar un valor a estas variables hay que declararlas **firstprivate**.

- **reduction( )**

  Las operaciones de reducción son típicas en muchas aplicaciones paralelas. Utilizan variables a las que acceden todos los procesos y sobre las que se efectúa alguna operación de “acumulación” en modo atómico (RMW).

  Caso típico: la suma de los elementos de un vector.

  Si se desea, el control de la operación puede dejarse en manos de OpenMP, declarando dichas variables de tipo **reduction**.

- **lastprivate(X)**

  Permite salvar el valor de la variable privada ***X*** correspondiente a la última operación realizada. Se utiliza si deseamos saber el último valor tomado por una variable antes de salir del área paralela (**fig. 4**).

```
#pragma omp parallel for private(i) lastprivate(a)
  for (i=0; i<n; i++)
   {
       a = i+1;
       printf("Thread %d has a value of a = %d for i = %d\n",
              omp_get_thread_num(),a,i);
   } /*-- End of parallel for --*/

  printf("Value of a after parallel for: a = %d\n",a);
```

Fig. 4. Ejemplo de uso de cláusulas **private** y **lastprivate** [Chapman et al 2008].

Paralelización de bucles.

Los bucles son uno de los puntos en los que se puede ejecutar instrucciones paralelas de la forma más simple (también llamado paralelismo de grano fino), se debe tener cuidado, ya que no todos los bucles son paralelizables, la condición indispensable que paralelizar un bucle es

que todas sus iteraciones sean independientes, dentro del bucle no debe de haber dependencia de datos. En la **figura 5** podemos observar un ejemplo de un ciclo *for* paralalelo.

```
#pragma omp parallel shared(n,a,b) private(i)
{
   #pragma omp for
   for (i=0; i<n; i++)
      a[i] = i;

   #pragma omp for
   for (i=0; i<n; i++)
      b[i] = 2 * a[i];
} /*-- End of parallel region --*/
```

Fig. 5. Uso de la directiva **omp for** [Chapman et al 2008]

La directiva **parallel for** implica la generación de ***k*** *threads*, que se repartirán la ejecución del bucle, ***parallel for*** por default implementa una barrera de sincronización implícita al final del bucle, esto hace que el hilo que termine primero debe esperar a que los demás termienen su trabajo, una vez hecho esto el ***thread*** máster retoma la ejecución, cabe aclarar que el índice del bucle es una variable **privada** no es necesario declararla como tal.

Las cláusulas de la directiva **for** son de varios tipos:

- **scope**: indican el ámbito de las variables.
- **schedule**: indican cómo repartir las iteraciones del bucle.
- **collapse**: permite colapsar varios bucles en uno.
- **nowait**: elimina la barrera final de sincronización.
- **ordered**: impone orden en la ejecución de las iteraciones.

Puesto que el ***pragma for*** termina con una barrera que impide el flujo de ejecución del programa hasta que todos los *threads* hayan finalizado su carga de trabajo, si la carga de los *threads* está mal equilibrada tendremos una pérdida notable de eficiencia. La cláusula *schedule* permite definir diferentes estratégias de reparto, tanto estáticas como dinámicas. Dos de los posibles argumentos para **schedule** son:

- ***static,k***

Planificación estática, divide las iteraciones en trozos de tamaño k. Si no se indica k, se reparten trozos de tamaño ***N/k***. siendo **N** el número de iteraciones y **k** en número de threads.

- ***dynamic,k***

  Asignación dinámica de trozos de tamaño k, al igual que en ***static,*** **k** por default es 1.

```
...
#pragma omp parallel num_threads(4), private(n)
      n = 20;
#pragma omp for schedule(static, 5)
      for (i=1; i <= 20; i++)
           a[i] = a[i] + 2*n;
```

Fig. 6. Uso de la cláusula **Schedule** [Chapman et al 2008].

En la **figura 6** se ejemplifica el uso de la estrategia de reparto *static* con k=5.

Sincronización

Cuando en el área paralela o en un bucle paralelizado no es posible eliminar las dependencias de datos entre los *threads*, es necesario sincronizar su ejecución. OpenMP proporciona dos mecanismos para sincronización: exclusión mutua y sincronización por eventos.

La exclusión mutua define un trozo de código que no puede ser ejecutado por más de un *thread* a la vez. OpenMP ofrece varias alternativas para la ejecución en exclusión mutua de secciones críticas. Las dos opciones principales son: **critical** y **atomic**.

- Directiva **atomic**

  Es un caso particular de sección crítica, en el que se efectúa una operación atómica sencilla. Para operaciones complejas o de varios renglones de código es más eficiente definir una sección crítica.

- Directiva **critical**

Define una única sección crítica para todo el programa, esta sección deber ser lo menor posible, para que el paralelismo sea eficiente, la directiva **critical** al contrario de **atomic**, se usa cuando el trozo de código es complejo (ver **figura 7**).

```
int ic, i, n;
ic = 0;
#pragma omp parallel shared(n,ic) private(i)
   for (i=0; i++, i<n)
    {
      #pragma omp atomic
         ic = ic + 1;
    }
printf("counter = %d\n", ic);
```

Fig. 7. Uso de la directiva **atomic** [Chapman et al 2008].